



Universidad  
Carlos III de Madrid

TELEMATIC ENGINEERING DEPARTMENT

---

---

FINAL YEAR PROJECT

NFC ANDROID APPLICATION  
DEVELOPMENT:  
UNITENFC

---

---

AUTHOR: IZAN DÍEZ SÁNCHEZ

TUTOR: MARIO MUÑOZ ORGANERO

LEGANÉS, OCTOBER 2013



*“The more I see the less I know”*

**Red Hot Chili Peppers**





### **Acknowledgments**

I would like to acknowledge to all the people that took part of my life during the last five years: friends, family, classmates, erasmus people, workmates, football teammates... For good, they all have contributed to make me as I am today.

I am particularly grateful for the company provided by Diego, Miguel and Daniel during all this time in any imaginable situation.

Finally I would like to thank my parents, Manuel and María de los Ángeles, and my sister, Alba, for being always there and giving me the chance to pursuit my dreams.



## Abstract

Virtual world is getting richer every day, everything is connected and people demand smarter devices which allow them to control any situation. New technologies are being developed, trying to integrate as many functions as possible, to deliver innovative solutions. One of this brand new technologies is NFC, which can connect devices instantly without any previous configuration, including passive devices which can store and deliver information of many kinds.

The work presented here reports an Android application project development with the Near Field Communication technology. The application developed is planned to add value to the NFC environment, increasing its discoverability and consumer awareness by building up a cloud of locations in which NFC interaction can be carried out. Besides, a community of users will be created in which they will be able to share their experiences and interact between them. The application will feed from the users, being the ones both producing and consuming information. All the data of the application will be available every-time and every-where with Internet connection, as any other service does nowadays.

Topics discussed in this document include Android programming, NFC, cloud services, software libraries, back-end development, integration with social networks, working methodology and details on the different development stages and tasks. A complete understanding of the system is tried to be transmitted as well.

To conclude, an approach to the market will be carried out, publishing the application on Google Play and analyzing the first impressions.

*Keywords:* NFC, RFID, API, tag, Android, NFC-Forum, NDEF, Reader-Writer, Peer to Peer, Card Emulation, NFC Point, UniteNFC, MVC, REST, cloud, back-end, Android Beam, integration, SDK.



## Resumen

El mundo virtual crece cada día, todo está conectado y los consumidores demandan cada vez dispositivos más inteligentes que puedan controlar cualquier situación. Se están desarrollando nuevas tecnologías, tratando de integrar tantas funciones como sea posible para ofrecer soluciones innovadoras. Una de estas nuevas tecnologías es NFC, que permite conectar dispositivos instantáneamente sin configuración previa. Incluso dispositivos pasivos que, sin alimentación, puedan almacenar y transmitir información.

El trabajo aquí documentado trata del desarrollo de una aplicación Android con la tecnología Near Field Communication. La aplicación desarrollada tratará de añadir valor al entorno de NFC, mejorando su descubribilidad y su reconocimiento, creando una nube de localizaciones en la que se pueda realizar algún tipo de interacción con NFC. Además se creará una comunidad de usuarios en la que se compartirán experiencias y podrán interactuar entre ellos. La aplicación se alimentará de los usuarios, los cuales tanto crearán como consumirán la información. Todos los datos de la aplicación estarán disponibles en cualquier lugar y momento con conexión a Internet, como cualquier otro servicio hoy en día.

Los siguientes temas van a ser analizados: programación en Android, NFC, servicios en la nube, librerías de software, desarrollo de back-end, integración con redes sociales y detalles sobre las diferentes etapas del desarrollo y sus tareas. Se intentará de transmitir una visión completa del sistema al mismo tiempo.

Por último, se realizará una primera aproximación al mercado, publicando la aplicación en Google Play y analizando las primeras impresiones.

*Palabras clave:* NFC, RFID, API, etiqueta, Android, NFC-Forum, NDEF, Lector-Escritor, Peer to Peer, Emulación de tarjeta, Punto NFC, UniteNFC, MVC, REST, nube, back-end, Android Beam, integración, SDK.



# Contents

List of Figures . . . . .	xxi
List of Tables . . . . .	xxii
Notations . . . . .	xxiii
<b>1 INTRODUCTION AND GOALS</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Goals . . . . .	2
1.2.1 To Explore and Understand NFC . . . . .	2
1.2.2 To Dive Into the Android Ecosystem . . . . .	2
1.2.3 To Study the Main Concepts of Cloud and REST API . . . . .	2
1.2.4 To Develop a Long-Term Project . . . . .	3
1.2.5 To Demonstrate and Improve Problem Solving Skills . . . . .	3
1.2.6 To Add Value to the NFC Environment . . . . .	3
1.3 Development Stages . . . . .	3
1.3.1 1st Stage: NFC Technological Study . . . . .	3
1.3.2 2nd Stage: Android Base Application Development . . . . .	4
1.3.3 3rd Stage: Web Service Development . . . . .	4
1.3.4 4th Stage: Final Platform Integration . . . . .	5
1.3.5 5th Stage: Documentation . . . . .	5
1.4 Resources . . . . .	5
1.4.1 Hardware . . . . .	5
1.4.2 Software . . . . .	6
1.4.3 Web Services . . . . .	7
1.4.4 Others . . . . .	7
1.5 Thesis Structure . . . . .	7
1.5.1 Chapter 1: Introduction and Goals . . . . .	7
1.5.2 Chapter 2: State of the Art . . . . .	8
1.5.3 Chapter 3: System Analysis and Design . . . . .	8
1.5.4 Chapter 4: Implementation . . . . .	8

1.5.5	Chapter 5: Testing and Further Development . . . . .	8
1.5.6	Chapter 6: Problems encountered . . . . .	8
1.5.7	Chapter 7: Planning and Budget . . . . .	8
1.5.8	Chapter 8: Conclusions . . . . .	8
1.5.9	Appendix A: Compatible Devices . . . . .	8
1.5.10	Appendix B: User Manual . . . . .	9
1.5.11	Appendix C: Application Code . . . . .	9
<b>2</b>	<b>STATE OF THE ART</b>	<b>10</b>
2.1	NFC Technology . . . . .	10
2.1.1	Description . . . . .	10
2.1.2	Operating Modes . . . . .	12
2.1.2.1	Reader-Writer . . . . .	13
2.1.2.2	Peer to Peer . . . . .	13
2.1.2.3	Card Emulation . . . . .	14
2.1.3	Use Cases . . . . .	14
2.1.4	Commercial Applications . . . . .	15
2.1.5	NFC-Enabled Mobile Phones . . . . .	18
2.1.5.1	Symbian Phones . . . . .	18
2.1.5.2	Android Phones . . . . .	19
2.1.5.3	Blackberry OS Phones . . . . .	19
2.1.5.4	Windows 8 Phones . . . . .	20
2.1.6	Comparison with other Technologies . . . . .	20
2.2	History and Evolution . . . . .	23
2.2.1	Origins . . . . .	23
2.2.2	RFID . . . . .	23
2.2.3	Milestones . . . . .	23
2.3	NFC Forum . . . . .	24
2.3.1	What is it? . . . . .	24
2.3.2	Mission and Goals . . . . .	25
2.3.3	Members . . . . .	26
2.3.4	N-Mark . . . . .	28
2.4	Technical Specifications . . . . .	28
2.4.1	NFC Architecture . . . . .	28
2.4.1.1	NFC Forum Protocols . . . . .	29
2.4.1.1.1	NFC Analog Technical Specification . . . . .	29
2.4.1.1.2	NFC Digital Protocol . . . . .	29
2.4.1.1.3	NFC Activity Technical Specification . . . . .	29
2.4.1.1.4	LLCP . . . . .	29
2.4.1.1.5	SNEP . . . . .	30
2.4.1.1.6	Tag Operation Specification . . . . .	30



2.4.1.1.7	NCI Technical Specification . . . . .	30
2.4.1.2	Reader-Writer Protocol Stack . . . . .	30
2.4.1.3	Peer to Peer Protocol Stack . . . . .	31
2.4.1.4	Card Emulation Protocol Stack . . . . .	31
2.4.1.5	Hardware . . . . .	31
2.4.1.5.1	NFC in Mobile Phones . . . . .	32
2.4.2	NFC Data Exchange Format (NDEF) . . . . .	33
2.4.2.1	RTD (NFC Record Type Definition) . . . . .	35
2.4.3	Tags . . . . .	36
2.4.3.1	NFC Forum Type 1 Tag . . . . .	37
2.4.3.2	NFC Forum Type 2 Tag . . . . .	37
2.4.3.3	NFC Forum Type 3 Tag . . . . .	37
2.4.3.4	NFC Forum Type 4 Tag . . . . .	38
2.4.3.5	Comparison . . . . .	38
2.4.4	Security . . . . .	38
2.4.4.1	Attacks . . . . .	39
2.4.4.1.1	Eavesdropping . . . . .	39
2.4.4.1.2	Data modification . . . . .	40
2.4.4.1.3	Man-in-the-middle . . . . .	40
2.4.4.1.4	Lost property . . . . .	41
2.4.4.1.5	Walk-off . . . . .	41
2.5	NFC Software Development . . . . .	41
2.5.1	APIs . . . . .	41
2.5.1.1	Earliest APIs . . . . .	42
2.5.1.2	Android . . . . .	43
2.5.1.3	Windows Phone 8 . . . . .	44
2.5.1.4	Blackberry OS . . . . .	44
2.5.1.5	Other Open Source APIs . . . . .	45
2.6	Near Future Foresight . . . . .	45
2.6.1	Market Acceptance . . . . .	45
2.6.2	Future Research . . . . .	46
<b>3</b>	<b>SYSTEM ANALYSIS AND DESIGN</b>	<b>47</b>
3.1	Naming and Logo . . . . .	47
3.1.1	Name . . . . .	47
3.1.2	Logo . . . . .	48
3.2	Scenarios . . . . .	49
3.2.1	Fairs and Events . . . . .	49
3.2.2	Tourism . . . . .	49
3.2.3	Augmented Reality Games . . . . .	50
3.2.4	Marketing and Publicity Campaigns . . . . .	50

3.3	Use Cases . . . . .	50
3.3.1	Use Cases Definition . . . . .	50
3.3.1.1	View NFC Points in Map . . . . .	50
3.3.1.2	Filter NFC Points by Type . . . . .	51
3.3.1.3	View a History of Visited NFC Points . . . . .	51
3.3.1.4	View a History of Registered NFC Points . . . . .	51
3.3.1.5	Register a New NFC Point . . . . .	51
3.3.1.6	Scan an NFC Point . . . . .	51
3.3.1.7	Connect with Facebook . . . . .	51
3.3.1.8	Add New Friend . . . . .	51
3.3.1.9	View a List of Friends . . . . .	52
3.3.1.10	View Friend Information . . . . .	52
3.3.1.11	Edit User Data . . . . .	52
3.3.1.12	View Walls . . . . .	52
3.3.1.13	Rate NFC Point . . . . .	52
3.3.1.14	Comment on NFC Point Wall . . . . .	52
3.3.1.15	Administrate Wall . . . . .	52
3.3.1.16	Share Information . . . . .	52
3.3.1.17	Receive Notifications of Nearby NFC Points . . . . .	53
3.3.1.18	Obtain Application Usage Feedback . . . . .	53
3.3.2	Diagram . . . . .	53
3.4	Architecture . . . . .	53
3.4.1	Android . . . . .	53
3.4.1.1	Hardware permissions . . . . .	53
3.4.1.2	Code Structure . . . . .	55
3.4.1.3	Libraries and SDKs . . . . .	56
3.4.1.4	Background Processing . . . . .	56
3.4.1.4.1	Threads . . . . .	57
3.4.1.4.2	Asynctasks . . . . .	57
3.4.1.4.3	Comparison . . . . .	58
3.4.2	Cloud Architecture . . . . .	58
3.4.2.1	Back-end . . . . .	58
3.4.2.1.1	Pattern: Model-View-Controller (MVC) . . . . .	59
3.4.2.1.2	Principle: REST . . . . .	60
3.4.2.1.3	Data Format: JSON . . . . .	60
3.4.2.1.4	Development Framework: Django . . . . .	61
3.4.2.1.5	Hosting: Heroku . . . . .	62
3.4.2.2	Other Services . . . . .	62
3.4.2.2.1	Topoos . . . . .	62
3.4.2.2.2	Google Maps . . . . .	62

	3.4.2.2.3	Facebook	63
	3.4.2.2.4	Google Analytics	63
3.5		Application Flow	63
	3.5.1	Launch Flow	63
	3.5.2	Settings, Report Bug and Sharing Flow	63
	3.5.3	MapFragment Flow	65
	3.5.4	SocialFragment Flow	65
	3.5.5	NFCPointsFragment Flow	66
	3.5.6	ServeActivity flow	67
	3.5.7	WallActivity Flow	67
<b>4</b>		<b>IMPLEMENTATION</b>	<b>69</b>
4.1		User Interface	69
	4.1.1	Look and Feel	69
		4.1.1.1 Tools	69
		4.1.1.2 Styling	70
		4.1.1.3 Interface Navigation	72
		4.1.1.3.1 ActionBar	72
		4.1.1.3.2 Menu	72
		4.1.1.3.3 Tabs	72
	4.1.2	ListView Creation	74
		4.1.2.1 Single Element Layout	74
		4.1.2.2 Single Element Class	74
		4.1.2.3 ListViewAdapter	74
	4.1.3	Map	76
		4.1.3.1 Camera	76
		4.1.3.2 Markers	77
	4.1.4	Dialogs and Toasts	78
4.2		Location	79
	4.2.1	LocationManager and LocationListener Setup	79
4.3		NFC Block	80
	4.3.1	NFC Setup	80
		4.3.1.1 Manifest	80
		4.3.1.2 NFC IntentFilters	81
	4.3.2	Interacting with NFC Points: Tag Reading	83
		4.3.2.1 Catching the Intent	83
		4.3.2.2 Foreground Dispatch	83
		4.3.2.3 Parse Tag Content	84
	4.3.3	Adding Friends: Android Beam	87
4.4		NFC Points	90
	4.4.1	POI topoos	90

4.4.2	Registering New NFC Point . . . . .	90
4.4.3	Checking in NFC Point . . . . .	92
4.5	RESTFUL Web Development . . . . .	93
4.5.1	Implementing MVC with Django . . . . .	93
4.5.1.1	Model . . . . .	94
4.5.1.2	Controller . . . . .	96
4.5.1.3	View . . . . .	97
4.5.2	Django Administration . . . . .	98
4.5.3	Deploying to the Cloud with Heroku . . . . .	98
4.5.4	Communication with Android Application . . . . .	100
4.6	Users Management . . . . .	102
4.6.1	Authentication . . . . .	102
4.6.1.1	Login and Registration . . . . .	102
4.6.1.2	Logout . . . . .	103
4.6.2	User Data . . . . .	103
4.6.2.1	Restoring User Data . . . . .	103
4.6.2.2	Saving User Data . . . . .	103
4.6.2.3	Profile Picture . . . . .	104
4.7	Facebook Integration . . . . .	106
4.7.1	Setting-up Facebook Application . . . . .	106
4.7.1.1	Creating Facebook Application . . . . .	106
4.7.1.2	Importing Facebook SDK . . . . .	107
4.7.1.3	Adding Facebook Lifecycle . . . . .	108
4.7.2	Connecting Account . . . . .	109
4.7.3	Adding Friends . . . . .	110
4.7.4	Publishing on Wall with Hashtags . . . . .	111
4.8	Adding Language Support . . . . .	114
4.8.1	Strings in Android . . . . .	115
4.8.2	Resources Folders . . . . .	115
4.8.3	Summary: How to Add a New Language . . . . .	116
4.9	Notifications . . . . .	116
4.9.1	Launching Service . . . . .	116
4.9.2	Service . . . . .	117
4.9.3	Notification Creation . . . . .	117
4.10	Google Analytics . . . . .	118
4.10.1	Setting-up Account . . . . .	118
4.10.2	Adding Analytics to the Android Life-cycle . . . . .	119
4.10.3	Statistics . . . . .	119
4.11	Publishing on Google Play . . . . .	121
4.11.1	Versioning . . . . .	123

<b>5</b>	<b>TESTING AND FURTHER DEVELOPMENT</b>	<b>125</b>
5.1	Testing . . . . .	125
5.1.1	Debugging the Android Application . . . . .	125
5.1.2	REST API Testing . . . . .	126
5.1.3	System Testing . . . . .	127
5.1.4	Feedback . . . . .	127
5.2	Further Development . . . . .	129
5.2.1	Optimizing for Other Screen Dimensions and Densities . . . . .	129
5.2.2	Adding More Language Support . . . . .	129
5.2.3	Better Instructions in the Application . . . . .	129
5.2.4	Optimizing Image Loading and Cache . . . . .	129
5.2.5	Code Cleaning and Maintenance . . . . .	130
<b>6</b>	<b>PROBLEMS ENCOUNTERED</b>	<b>131</b>
6.1	Testing Hardware Related Functionalities . . . . .	131
6.2	Android Fragmentation . . . . .	131
6.3	NFC Android Beam . . . . .	132
6.4	Topoos Incomplete API . . . . .	133
6.5	Web Development Inexperience . . . . .	133
<b>7</b>	<b>PLANNING AND BUDGET</b>	<b>134</b>
7.1	Gantt Chart . . . . .	134
7.2	Project Budget . . . . .	137
<b>8</b>	<b>CONCLUSIONS</b>	<b>140</b>
8.1	General Conclusions . . . . .	140
8.2	Personal Conclusions . . . . .	141
<b>A</b>	<b>Compatible Devices</b>	<b>142</b>
<b>B</b>	<b>User Manual</b>	<b>146</b>
B.1	Installation and Launch . . . . .	146
B.2	Login . . . . .	147
B.3	Main Screen . . . . .	147
B.4	Map Screen . . . . .	148
B.5	NFC Points Screen . . . . .	149
B.6	Social Screen . . . . .	149
B.7	Wall Screen . . . . .	150
B.8	NFC Points Reader . . . . .	151
B.9	Notifications . . . . .	151
B.10	Settings . . . . .	152

<b>C Application Code</b>	<b>154</b>
C.1 UniteNFC Client-side Android Code . . . . .	154
C.2 UniteNFC Back-End Code . . . . .	154
<b>Bibliography</b>	<b>159</b>

# List of Figures

2.1	NFC supported modes . . . . .	11
2.2	NFC Operating Modes . . . . .	13
2.3	Reader/Writer [10] . . . . .	13
2.4	Peer-to-Peer mode [10] . . . . .	13
2.5	Card Emulation [10] . . . . .	14
2.6	NFC used for multiple purposes [11] . . . . .	15
2.7	NFC application icons [ <a href="https://play.google.com">https://play.google.com</a> ] . . . . .	17
2.8	Symbian NFC mobile phones . . . . .	18
2.9	Android NFC mobile phones . . . . .	19
2.10	Blackberry NFC mobile phones . . . . .	19
2.11	Windos 8 NFC mobile phones . . . . .	20
2.12	Wireless technologies [11] . . . . .	21
2.13	NFC Forum logo [11] . . . . .	25
2.14	NFC Forum environment [11] . . . . .	26
2.15	N-Mark [11] . . . . .	28
2.16	NFC Forum Architecture [11] . . . . .	28
2.17	Reader-Writer protocol stack . . . . .	30
2.18	Peer to Peer protocol stack . . . . .	31
2.19	Card Emulation protocol stack . . . . .	32
2.20	NFC-enabled mobile phone internal scheme [Smart Card Alliance] . . . . .	32
2.21	NDEF Message layout [10] . . . . .	33
2.22	NDEF Record layout [28] . . . . .	35
2.23	NFC Forum Tag Types . . . . .	37
2.24	NFC development. OS logos . . . . .	42
3.1	UniteNFC logo . . . . .	48
3.2	Markers on UniteNFC . . . . .	48
3.3	Use Cases . . . . .	54

3.4	Android application structure . . . . .	55
3.5	Cloud Architecture . . . . .	59
3.6	Typical MVC collaboration . . . . .	59
3.7	Application launch flow . . . . .	64
3.8	Application settings, report bug and sharing flow . . . . .	64
3.9	MapFragment flow . . . . .	65
3.10	SocialFragment flow . . . . .	66
3.11	NFCPointsFragment flow . . . . .	67
3.12	ServeActivity flow . . . . .	68
3.13	WallActivity flow . . . . .	68
4.1	Styled button not pressed and pressed . . . . .	71
4.2	Tag Dispatch System [43] . . . . .	81
4.3	Beam flow . . . . .	89
4.4	Registration Dialog . . . . .	91
4.5	Profile picture selector . . . . .	104
4.6	UniteNFC Facebook page . . . . .	107
4.7	Facebook Dialog . . . . .	110
4.8	Facebook flow (I) . . . . .	111
4.9	Facebook flow (II) . . . . .	112
4.10	Example publication on Facebook . . . . .	114
4.11	Different languages resource folders . . . . .	116
4.12	Notification of nearby NFC Point . . . . .	118
4.13	UniteNFC user sessions statistics (03/10/2013) . . . . .	120
4.14	UniteNFC device statistics (03/10/2013) . . . . .	121
4.15	UniteNFC language statistics (03/10/2013) . . . . .	121
4.16	UniteNFC engagement flow statistics (03/10/2013) . . . . .	122
5.1	Logcat example logs . . . . .	126
5.2	REST API testing . . . . .	128
5.3	UniteNFC in a tablet . . . . .	130
6.1	Android vs. iOS fragmentation [45] . . . . .	132
7.1	Gantt Chart . . . . .	136
B.1	Installation and Launch . . . . .	146
B.2	Login . . . . .	147
B.3	Main screen . . . . .	148
B.4	Map screen . . . . .	149
B.5	NFC Points screen . . . . .	150
B.6	Social screen . . . . .	151



B.7	Wall screen . . . . .	152
B.8	Reader screen and notifications . . . . .	153
B.9	Settings . . . . .	153

# List of Tables

2.1	NFC basic specifications . . . . .	12
2.2	Short-range wireless technologies comparison . . . . .	22
2.3	NFC Forum members . . . . .	27
2.4	NFC Forum tag types . . . . .	39
3.1	Thread vs. AsyncTask . . . . .	58
7.1	Project tasks breakdown . . . . .	135
7.2	Personnel budget . . . . .	137
7.3	Equipment budget . . . . .	137
7.4	Software budget . . . . .	138
7.5	Other services budget . . . . .	138
7.6	Total budget . . . . .	139

# Notations

API: Application Programming Interface	2
IDE: Integrated Development Environment	6
IEC: International Electrotechnical Commission	6
IrDA: Infrared Data Association	21
ISO: International Organization for Standardization	6
LLCP: Logical Link Control Protocol	29
MVC: Model-View-Controller	4
NCI: NFC Controller Interface	30
NDEF: NFC Data Exchange Format	30
REST: Representational State Transfer	4
NFC: Near Field Communication	1
RF: Radio Frequency	2
RFID: Radio Frequency Identification	16
SIM: Subscriber Identification Module	33
SNEP: Simple NDEF Exchange Protocol	30

# 1

## INTRODUCTION AND GOALS

This chapter will introduce the reader to the project described throughout all the document. Background on the NFC technology will be given along with basic details of the development stages and resources used.

### 1.1 Introduction

Not long ago every device was designed to accomplish a single task. Cameras took pictures, TVs showed images, phones made calls, fax sent messages, MP3 players played music, etc. Nowadays devices are taking shapes that were hard to ever imagine. Now all those features can be carried out by a single device that is small enough to carry everywhere, i.e. the smart-phone. Technology is being driven by a rush of innovative solutions that are quickly integrated in devices covering all kind of needs.

One of these technologies is Near Field Communication. NFC is a fresh technology that is gaining ground slowly and which potential is yet to be unveiled. The possibilities are huge. It is really easy to use, involving a physical gesture and getting instant feedback.

Simultaneously Internet services are getting richer, data-centers becoming bigger and connections faster. The Cloud is well-established already and smart devices perform transactions with it regularly.

The application developed arrange all that concepts. First of all it has been developed for mobile devices, Android ones specifically. Moreover, it makes use of an emerging technology, NFC. And last, but not least, it makes use of several cloud services to offer a more complete experience.

UniteNFC will act as a portal to the NFC world, allowing to discover and share NFC Points based on the location. UniteNFC will be the main NFC application on the mobile, managing all the content read through its interface and storing a history. At the same time, it is intended to be as social as possible, allowing interaction with other users by friendship relationships and comments on the NFC Points. In a future where NFC had reached the masses there would have to be a way to unify all the transactions made with NFC and UniteNFC would be such way.

## 1.2 Goals

The main goal of the project is to be able to develop an innovative service for NFC using the platforms and tools available at the moment, applying concepts and skills acquired during the degree.

From that premise more specific underlying goals are going to be described.

### 1.2.1 To Explore and Understand NFC

Explore NFC, understand what makes this technology special, study its state of the art, analyze opportunities and get to know all the technical details. Eventually achieving expertise in some of the fields that NFC involves.

### 1.2.2 To Dive Into the Android Ecosystem

Go into detail on how this operative system for mobile phone works. Learn how Android life-cycle works and what are its main elements. Program according to the best practices stated by the Android community and be able to create a sufficiently complex application.

### 1.2.3 To Study the Main Concepts of Cloud and REST API

Analyze how cloud services are structured nowadays. Understand the concepts of REST API, data models and request flows. With all that in mind, a simple

cloud service for the Android application could be implemented.

#### **1.2.4 To Develop a Long-Term Project**

Carry out a long term project focusing on the final result. An extremely important point is to meet the planned schedule, which can be difficult if the project is not prioritize properly against other life issues.

#### **1.2.5 To Demonstrate and Improve Problem Solving Skills**

Despite of the variety of courses taken in the degree, one common factor bring all together: problem solving. Facing this project individually has to confirm that all these years journey was not fruitless. The tutor will be consulted as little as possible.

#### **1.2.6 To Add Value to the NFC Environment**

The NFC world lacks of a killer application or use case. The developed application should boost it and enhance it as far as possible.

### **1.3 Development Stages**

This section will describe the different stages that the project has gone through. In each stage of the project the tasks accomplished were driven to achieve a common goal, that will be explained in the following subsections.

#### **1.3.1 1st Stage: NFC Technological Study**

Firstly, after the project was chosen and the goals were more or less established, a broad study about the technology selected was carried out. Even-though many of the concepts studied in this stage were not directly applied into the project development, a wide view of the NFC environment and philosophy was attained.

That view was very useful to redefine the initial scope of the project, since the possibilities available for a developer with any hardware but an NFC enabled mobile phone running Android were quite limited due to API constraints [2.5.1] and unavailability of secure element access. Therefore Card Emulation mode [2.1.2] had to be discarded from the beginning and with regard to Peer-to-Peer, developers are tied to Android Beam feature which requires extra boring user interaction.

### 1.3.2 2nd Stage: Android Base Application Development

Once all the constraints of NFC were identified, the Android application development was started. In this stage, the purpose was to built up a simple application where all the (what were named) NFC Points were spotted in a map and a history of visited points could be looked up. Such NFC Points had to be registered by users, which could log in and register in the application.

To cope with these tasks, additional libraries were needed. The representation of the NFC Points was achieved by means of the well known *Google Maps API* [1]. User management and points of interest support was added by *topoos* [2]. This service made really easy to implement the desired operations, although is in beta version. Besides it offers Rest API which can be useful to communicate from a web in the cloud.

The main flow of Android user interface elements (Activity, Fragment, Dialogs, etc.) was also implemented at this moment, acting as the skeleton for the above described functionalities. With it all the NFC actions were programmed too. Using Reader-Writer mode to register NFC Points and adding Peer-to-Peer for future friend addition.

### 1.3.3 3rd Stage: Web Service Development

Services provided by third party can be very powerful and easy to use, but sometimes it may not fit your needs. In this case, *topoos* social features were not mature enough to meet the target needs (by the time this paragraph is written it actually is) and the history returned by the API was not rich enough. Hence, at this point the decision of developing a proprietary cloud web service was made.

The little experience earned by using *topoos* API was helpful to be aware of REST architecture and understand how the client-side application communicates with the server-side application. With help of a web application framework, Django [3], and using the MVC model all the REST concepts were applied and the desired function implemented. To make this service accessible from everywhere, the website was hosted on Heroku [4], a cloud platform.

After the service is running in the cloud, the Android application can call any of the functions implemented with a simple web request. Then, friendship relations were added along with richer information about registered NFC Points storage, gathering comments and ratings.

### 1.3.4 4th Stage: Final Platform Integration

Development stages are coming to an end and still are many interesting features to be added. Furthermore, it is important that the user experience when using the application is smooth, with enough information about what is going on beneath the interface, error handling and, above all, no crashing.

One of the functionalities that was added in this stage is Facebook integration [5]. Almost every application that includes social capabilities is integrated with the biggest social network ever. Doing so, allows you to connect more people, share the content and adds a viral component that can help you application to spread quickly.

Another interesting functionality, that was unplanned at the beginning, is Google Analytics. Really easy to integrate and get statistics about what is going on in you application [6].

Finally, testing was done and necessary changes were made. The application was upload to Google Play and the feedback of the first users was received.

### 1.3.5 5th Stage: Documentation

Before the project can be considered finished, all the knowledge, experience, ideas, questions and conclusions acquired during all these months, need to be recorded so that it depicts, as accurately as possible, the work done.

## 1.4 Resources

All the material used in the realization of the project, divided into hardware and software, and additional services will be explained briefly in this section.

### 1.4.1 Hardware

- Sony Vaio VPCCA1S1E: 4x Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz processor, 4078.5MB RAM memory, 320GB hard disk. Laptop. More than enough if the Android Virtual Device emulator is not used.
- Logitech Wireless Mouse M305: small wireless mouse.
- Sony Xperia S: 1.5 GHz Qualcomm Dual Core processor, 1GB RAM memory, with NFC chip. Android Jelly Bean 4.1. No malfunctioning noticed during the whole project period.



- Samsung Galaxy S2 Plus: 1.2 GHz Broadcom Dual Core processor, 1GB RAM memory, with NFC chip. Android Jelly Bean 4.1. No malfunctioning noticed during the whole project period.
- Mobile NFC Tag Testing Set: set of NFC tags including
  - MIFARE Ultralight: NFC Forum Type 2 tag (ISO/IEC 14443 Type A), 384-bit user memory. [7]
  - MIFARE Ultralight C: NFC Forum Type 2 tag (ISO/IEC 14443 Type A), 1184-bit user memory. [7]
  - Broadcom Topaz: NFC Forum Type 1 tag, 768-bit user memory. [8]

### 1.4.2 Software

- Ubuntu 12.04-13.04 LTS: Linux kernel operating system. Free and open source software.
- Java Platform (JDK) 7u40: Java software development kit. Android is programmed in Java.
- Android SDK 4.0-4.3: set of development tools and libraries, including an emulator.
- Eclipse Helios 3.6.2: multi-language IDE. Needs of ANT plugin to work with the Android SDK.
- Android Studio IDE: exclusive Android IDE. Similar to Eclipse but with much better user interface edition. Use Gradle and Maven to manage the projects.
- GIMP Image Editor 2.8: image editing tool. Same concept as Photoshop but it is free software under the GNU project.
- Sublime Text 2: source code and text editor, cross-platform. Lots of plugins are available making coding fast and efficient.
- Django: Python web framework that follows MVC (model-view-controller) architecture.

### 1.4.3 Web Services

- Google Play Developer Account: Android application distribution system. A fee has to be paid to be regarded as developer and be free to upload as many applications as you want.
- Google Play Services SDK: Which include the Google Maps API necessary to draw the maps and retrieve your location.
- Heroku: Cloud platform as a service. Supports several programming language. Fast setup of a cloud web server. Free support with limited processing power.
- Github Public Repository: Git version control service with free hosting for open source projects.
- Facebook Android SDK: Library to easily integrate with Facebook.
- Topoos Android SDK: Location based and context-aware service provider. Still in beta version.
- Google Analytics Android SDK: application tracking with Google Analytics in Android native application. Easy integration and nice representation of statistics.

### 1.4.4 Others

- ADSL Internet Connection: for daily work.
- 3G Internet connection: essential to make field testings, registry of events while walking on the street.

## 1.5 Thesis Structure

This thesis is divided in eighth chapters with three extra appendices. To facilitate the reading, a brief summary of each chapter is included.

### 1.5.1 Chapter 1: Introduction and Goals

This first chapter that is concluding is meant to give the reader a short introduction to the project along with its motivation. Besides, the development stages and the necessary resources to carry out the project are specified.

### **1.5.2 Chapter 2: State of the Art**

The reader is provided with all the information needed to the understanding of the NFC technology and how it fits into today's digital world.

### **1.5.3 Chapter 3: System Analysis and Design**

Definition of the platform to be developed, architecture, UML diagrams, analysis of technologies and final choices.

### **1.5.4 Chapter 4: Implementation**

How all the described system in the previous section has been built and all the pieces put to work together.

### **1.5.5 Chapter 5: Testing and Further Development**

Methods and tools used to test the correct working of the application and future tasks to be committed if the project was to be continued.

### **1.5.6 Chapter 6: Problems encountered**

The problems and obstacles that have arisen during the nine months the project has last and how have been overcome.

### **1.5.7 Chapter 7: Planning and Budget**

Detailed breakdown of project tasks, timing and costs of implementing the project that will be described.

### **1.5.8 Chapter 8: Conclusions**

A compendium of insights acquired during the development of the project. In addition a personal opinion on what the project has meant to me will be given.

### **1.5.9 Appendix A: Compatible Devices**

A complete list of compatible Android devices (mobile phones and tablets) is given. Basically these devices must fulfill two requirements, to have integrated an NFC chip and to be compatible with Android 4.0. or higher.

### **1.5.10 Appendix B: User Manual**

Instructions on how to, from the user point of view, install, register, play around with NFC tags and squeeze all the functionalities the application provides.

### **1.5.11 Appendix C: Application Code**

Links to the public repositories where the code that has been deployed into the project is stored.

# 2

## STATE OF THE ART

This chapter contains all the relevant information of Near Field Communications. A deep study has been carried out to give a rich outlook of the state of the art of this technology.

### 2.1 NFC Technology

This section will introduce the reader with a background to the technology studied. A basic description is presented, along with a more detailed explanation of its functioning, uses of NFC and in which devices it is found.

#### 2.1.1 Description

Near Field Communication (NFC) is a high frequency, short range, wireless technology used to transmit data between devices through a radio communication. It defines a set of standards specifying data format and protocols to establish such connection. NFC comprises and harmonizes a set of standards for close proximity communication already present in the industry. Devices are connected instantly by touching them together or bringing them into a range of few centimeters. It works in the unlicensed band of 13,56 MHz, so there is no need of license to use it.

NFC always involves an initiator and a target. The former actively generates a field that may power a passive target. This permits NFC targets to take very simple format such as tags, stickers, key holders, or cards that do not require batteries. NFC communication with passive devices is possible thanks to the induction of a magnetic field, in which two antennas are placed inside each other's near field [9].

When both devices are powered, NFC peer-to-peer communication is possible.

According to this NFC supports two modes:

- Active: Each device generates its own RF field. Thus, normally both devices have power supplies. Initiator and target alternate their fields, i.e. a device deactivates its generated field while waiting for data.
- Passive: In this case the initiator is a powered device, whereas the target is a passive transponder that modulates the electromagnetic carrier field provided by the initiator. The required power needed for such modulation is extracted from the initiator RF field.

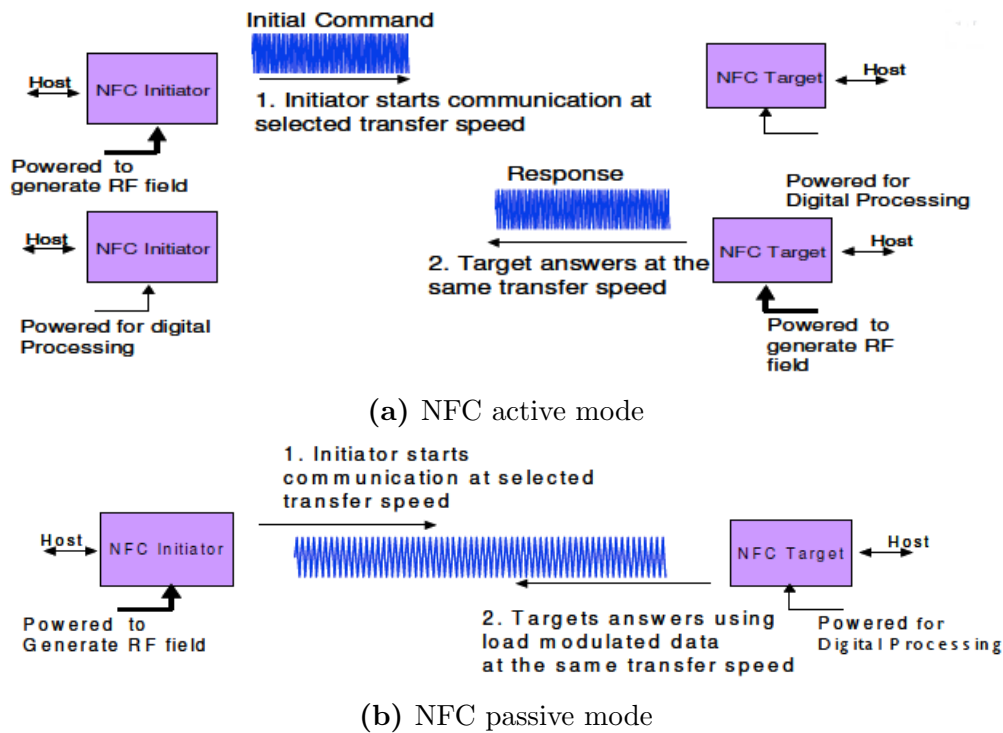


Figure 2.1: NFC supported modes

NFC employs two different codings to transfer data. A modified Miller coding with 100% modulation is used when the active device transfers data at a rate of 106 kbit/s. In all other cases Manchester coding is used with a 10% modulation ratio. As NFC devices are able to receive and transmit data at the same time, they can check for potential collisions if the received signal frequency does not match with the transmitted signal's frequency. Communication finishes either on command or

Basic Specifications		
	Active	Passive
Speed	<b>106 kbit/s</b> , Modified Miller 100% ASK	<b>106 kbit/s</b> , Manchester 10% ASK
	<b>212 kbit/s</b> Manchester 10% ASK	
	<b>424 kbit/s</b> Manchester 10% ASK	
Range	Up to 20cm (typically < 5cm)	
Standards	ISO/IEC 18092, ECMA-340 and ETSI	

**Table 2.1:** NFC basic specifications

when the devices separate. A summary with these basic specifications is presented in table 2.1.

The objectives of this kind of technology is not to be a substitute of the existing ones but to complement them. Allowing devices to interconnect easily, provide ad-hoc communication, transmit data, initialize other type of communication (Bluetooth, WiFi). It seems to be an adequate medium for electronic payments and access control. Besides, it is not necessary that all devices have an own energy source.

Mobile phones are the target to increase the reach and scope of NFC technology, as it is the electronic device everybody has. More than 100 million of commercial handsets are NFC phones, and it is expected to keep growing.

### 2.1.2 Operating Modes

An NFC communication is a point to point connection that involves two devices. Depending on the way the devices interact between them three modes of operation are considered: reader/writer, peer-to-peer, and card emulation. The different operating modes are based on the ISO/IEC 18092 NFC IP-1 and ISO/IEC 14443 contactless smart card standards and are defined by the NFC Forum. In figure 2.2 a simple scheme of the operating modes is presented. Every NFC communication has to be based in one of these three operating modes.

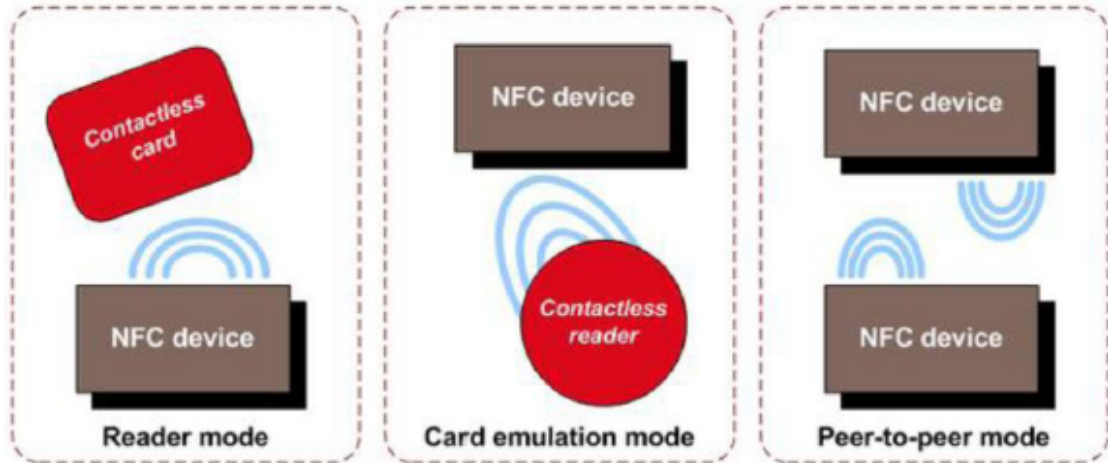


Figure 2.2: NFC Operating Modes

### 2.1.2.1 Reader-Writer

In reader/writer mode, the NFC (active) device is capable of reading (passive) NFC Forum-compliant tag types, such as a tag embedded in an NFC smart poster. The reader/writer mode on the RF interface is compliant with the ISO 14443 and FeliCa schemes [11].



Figure 2.3: Reader/Writer [10]

### 2.1.2.2 Peer to Peer

In Peer-to-Peer mode, two (active) NFC devices can exchange data. For example, you can share Bluetooth or WiFi link set-up parameters or you can exchange data such as virtual business cards or digital photos. Peer-to-Peer mode is standardized on the ISO/IEC 18092 standard [11].



Figure 2.4: Peer-to-Peer mode [10]



### 2.1.2.3 Card Emulation

In Card Emulation mode, the NFC device appears to an external reader much the same as a traditional contactless smart card. This enables contactless payments and ticketing by NFC devices without changing the existing infrastructure [11].

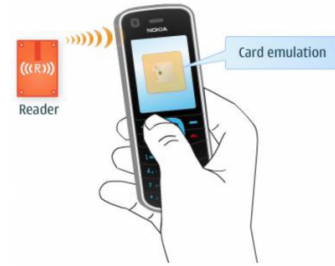


Figure 2.5: Card Emulation [10]

### 2.1.3 Use Cases

What makes NFC unique is that it allows electronic devices to interact with the physical world. In other words, it gives a connection with the digital world, in which the amount of data and transaction is growing every day, with just an intuitive touch. Having this in mind the variety of use cases can be wide and rich.

Generally, the applications follow the tendency of using the NFC operating mode that best fits to its purpose. For example:

- Connect Electronic Devices  $\implies$  Peer-to-Peer.
- Access Digital Content  $\implies$  Reader/Writer.
- Make Contactless Transactions  $\implies$  Card Emulation.

*NFC applications are limited only by the imagination and individual creativity. NFC makes the experience of mobile internet and other mobile services fast and convenient, and opens up a host of new opportunities. Mobiles are bringing the world of NFC to new audiences, including advertisers, marketing agencies and potentially every consumer using a mobile NFC phone. [12]*

Some use cases are presented to make the reader aware of the amount of different applications in which NFC can be deployed[13]:

- |  |   |
|--|---|
| • Pay with NFC phones at contactless POS | • Bluetooth coupling with car phone/radio               |
| • Touch tags to collect shopping list    | • Home entertainment systems                            |
| • Store electronic keys on NFC phones    | • Theater/attraction/event tickets storage on NFC phone |
| • Business card & calendar               | • Download coupons from smart                           |



**Figure 2.6:** NFC used for multiple purposes [11]

- poster to NFC phone
- Gaming & toys
- Read smart posters to NFC phone with information
- Social Media & advertising
- Enhance loyalty programs
- Access rooms and buildings with NFC phone and contactless reader
- Anything inspired by your imagination!

### 2.1.4 Commercial Applications

- a) Foursquare: the popular location-based social network for mobile devices. Users “check in” at venues by selecting from a list of venues the application locates nearby, based in GPS or network location. In 2012 an update was released in which NFC was added to “check in” automatically by tapping on tags available at some venues.
- b) Google Wallet: mobile payment system developed by Google that allows its users to store debit cards, credit cards, loyalty cards, and gift cards among other things, as well as redeeming sales promotions on their mobile phone. Google Wallet uses NFC to “make secure payments fast and convenient by simply tapping the phone on any PayPass-enabled terminal at checkout”[14].
- c) ISIS: The Isis Mobile Wallet stores virtual versions of many things your physical wallet does, such as payment cards. The Wallet allows you to organize payment

cards, offers and loyalty cards on your mobile phone. With NFC capability you are able to communicate with the checkout terminal at any Isis Ready merchant. Isis is a joint venture between AT&T, T-Mobile and Verizon Wireless in the mobile payment space and the main competitor of Google Wallet.

- d) NFC TagInfo by NXP: provided by NXP Semiconductors the TagInfo application allows you to browse the content of any NFC Forum compliant tag and also some other contactless technologies (like RFID) present in public transport and payment cards.
- e) NFC TagWriter by NXP: this application can write contacts, bookmarks, Bluetooth Handover, email addresses, SMS, geo location, plain text and many more to any NFC-enabled tag as well as to items like poster, business cards, watches and many more containing NFC-enabled electronics. The data stored in the tag can be viewed using this app too and launch other applications automatically based on the programmed data.
- f) PayPal: the most widespread global e-commerce business that allows payments and money transfers to be made through the Internet. Online money transfers serve as electronic alternatives to paying with traditional paper methods, such as checks and money orders. NFC support was added in 2011 but it has been recently given up to adopt a different strategy (cloud based).
- g) Madrid Metro|Bus|Cercanias: essential Android application for Madrid citizens and tourists daily life. It contains all the information (maps, schedules, routes calculation) about the three means of public transport present in the city: bus, train and underground. NFC stickers are placed in bus stops to make the app consultation as fast as a tap, avoiding having to search or type in the mobile phone.
- h) PPTLagartoSpock: simple application that emulates the Rock-paper-scissors-lizard-Spock handgame (five-gesture expansion of the classic selection method game rock-paper-scissors). Each player makes the selection in its own mobile phone and then communicate using Android Beam (P2P) to see the winner. It also works storing a player's choice in a tag and reading it to see the result.
- i) BT TagWriter: makes pairing of your Bluetooth speakers and your mobile phone easy and fast. All you need is Android device with NFC support and a writable NFC tag. The handover configuration is written in a NFC tag so that just reading it your Bluetooth speakers get instantly connected.
- j) NFC TagInfo: similar to *NFC TagInfo by NXP* but developed by a company independent person, this app allows you to see all the content stored in NFC

compatible contactless cards. What makes the difference is that, this one, allows you to browse the content sector by sector giving you the information of whether a specific sector is protected, it belongs to the header or whatever.

- k) Think&Go NFC Shopping: NFC-Commerce platform providing all the basic services and ready to be deployed in a retail environment. Tap-to-Basket, personalized product information, couponing, contextual advertising and advanced customer profiling are some of the functions it is ready to provide.
- l) TagCenter NFC Offers: allows you to find and collect exclusive coupons, content and applications from your favourite brands and stores by tapping posters at participating locations. It is still a prototype, but gives an interesting starting point for future apps.



**Figure 2.7:** NFC application icons [<https://play.google.com>]

## 2.1.5 NFC-Enabled Mobile Phones

NFC is reaching more people thanks to its integration within, mainly, mobile phones. Two or three years ago very few people would have known what is about. On the contrary, now NFC is a characteristic that is being included in the best devices and people look for. It will not be too hazard to say almost every mobile phone manufacturer has some NFC-enabled phone among their products.

The main obstacle NFC has encounter to spread even more has been the lack of support from Apple, who decided not to include it in the new iPhone5. This fact may have delayed the awaited final boom of NFC.

Lists with all NFC handsets are available[15] in which you can find every existing device. In the following subsection a classification of NFC mobile phones has been made with some examples. The criteria followed to divide them has been the operating system, which has become the most important reason to buy one mobile phone or a different one.

### 2.1.5.1 Symbian Phones

The first NFC-enabled mobile phones that hit the market were the Nokia running on its own operating system, Symbian. The pioneer was the Nokia 6131 NFC[15]. By the time it was a big revolution and many handsets were expected to follow it. However Symbian was about to lose its popularity and market share in favour of Android and iOS devices, costing Nokia to fight against bankrupt. Nowadays Symbian is almost abandoned. Two of the last released NFC Symbian mobile phones can be viewed in figure 2.8.



**Figure 2.8:** Symbian NFC mobile phones

### 2.1.5.2 Android Phones

Android is a Linux-based operating system, designed for smartphones and tablets. Since it was unveiled Android popularity has grown up to the point of becoming the world's most widely used smartphone platform. In 2010 the Nexus S is released, being the first NFC Android phone. Google has been one the main driving forces to provide NFC to the mass market.



(a) Google Nexus S      (b) Sony Xperia S

**Figure 2.9:** Android NFC mobile phones

### 2.1.5.3 Blackberry OS Phones

Research In Motion, as Nokia, is having some troubles in maintaining good sales numbers. Instead on changing the platform on which they used to work, Blackberry has insisted in its own OS. Lot of research has been made to adapt it to the current consumer needs and to offer the latest technological innovation. RIM has been working with NFC for couple of years already an offers the same capabilities of its main competitors.



(a) BB Bold 9790      (b) BB Bold 9900

**Figure 2.10:** Blackberry NFC mobile phones

#### 2.1.5.4 Windows 8 Phones

The last operating system to be released of the majority ones has been Windows Phone 8. Nokia created an alliance with Microsoft to use Windows Phone 8 as the OS for its devices to try to recover from the failure. The attractiveness of this brand new OS has brought many other manufacturers as HTC and Samsung to launch mobile phones running on Windows Phone 8. Microsoft has provided with a fairly simple API to enhance the experience of users by means of developers work.



(a) HTC 8X

(b) Nokia Lumia 920

**Figure 2.11:** Windos 8 NFC mobile phones

#### 2.1.6 Comparison with other Technologies

Each kind of wireless technology brings us a number of characteristics that make it appropriate for several uses and scenarios. Different configuration and modes of operation with its corresponding characteristics (bit-rate, setting time, range, security, etc), makes every technology suitable for different needs.

A detailed comparison of the main characteristics between some wireless technologies is made in table 2.2. A brief description of each of the technologies appearing in the table is presented to put the reader into context.

- **RFID:** predecessor of NFC. Wireless technology that provides an automatic identification method based in the remote recovery of data stored in RFID tags.
- **Bluetooth:** designed in its origin to replace data cables between devices, such as the RS-232. Bluetooth is a wireless technology standard for exchanging data over short distances from fixed and mobile devices with high levels of security.

- IrDa: low-range wireless technology that transmits and receives data through infrared light. Used in computers and mobile phones.
- ZigBee: wireless technology that allows the control and tracking of industrial applications and domotics. Targeted at applications requiring low data rates, long battery life, and secure networking.

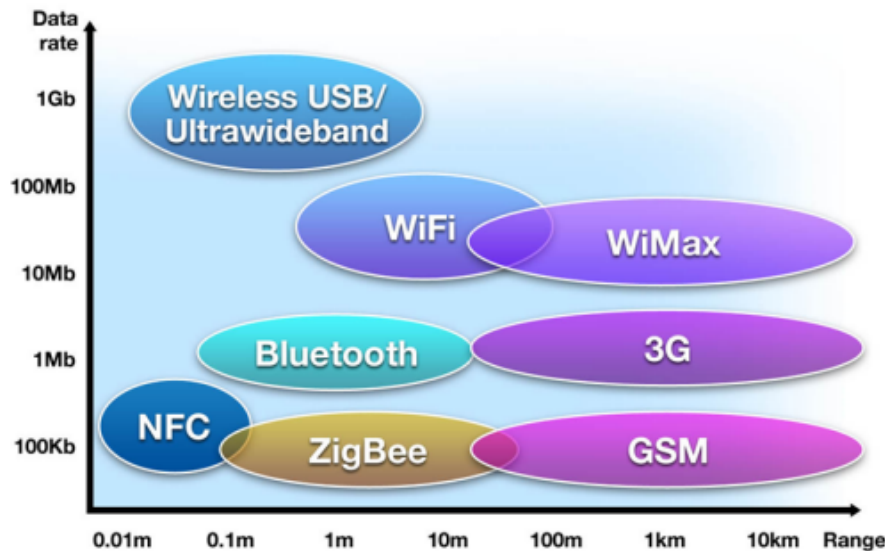


Figure 2.12: Wireless technologies [11]

Looking at the different features of each technology first thing to note is that they can not be compared in absolute terms. One specific technology may have some advantage in one characteristic, but lose it in another. Or, what is more, an attribute that seems to be useless can be very helpful in some special situation.

For instance, the low range that requires NFC can be thought as nonsense for a wireless technology. But as a consequence it provides a intrinsic security against *sniffers* the rest need to implement by other means in higher layers, except for IrDa in which the intrinsic nature of light needs of a line of sight communication being thus difficult to capture (and making the process, at the same time, more tedious). Also the close proximity that devices connected using NFC must be to each other is actually useful in crowded locations, allowing to prevent interferences caused when other devices are present trying to communicate. Bluetooth may have trouble dealing with interference when trying to send signals between two devices, especially when several other devices are in close proximity [16].



	NFC	RFID	Bluetooth	IrDa	ZigBee
Set-up time	<0.1s	<0.1s	~6s	~0.5s	~0.03s
Range	0.01m	3m	30m	1m	70m
Frequency	13.56 MHz	13.56 MHz	2.4-2.5 GHz	>300 Ghz	2.4 Ghz
Bit rate	424 kbit/s	424 kbit/s	3 Mbit/s	2.4 kbit/s- 4 Mbit/s	250 kbit/s
Security	High	Medium	High, PIN based	LOS (Line of sight)	High
Network type	Point-to-point	Point-to-point	WPAN	Point-to-point	WPAN
Ease of use	One touch	One touch	Pairing configuration	No configuration needed	Network configuration
Uses	Building access, payments, bluetooth handover, etc	Identification, product tracking, etc	Data exchange network	Data exchange and remote controls	Domotics, industrial control, etc.

**Table 2.2:** Short-range wireless technologies comparison

The set-up time is another important issue. RFID and NFC are practically instant connection, compared to Bluetooth in which the negotiation delays a bit the initiation. Though in the case a big file needs to be transmitted the higher bit-rate of Bluetooth will give a better general performance. IrDa and ZigBee are able to reach even lower set-up times due to its simplicity, however ZigBee maximum bit-rate is lower and it has been already commented the main problem of IrDa.

So what really makes NFC special is the combination of the lowest operation range and the fast set-up time. This blending results very convenient to complement many of the most popular wireless technologies as the ones listed. Besides, it can be fully compatible with the existing contact-less cards infrastructure and allows the consumer to use only one device across multiple platforms. To sum up, with NFC technology many simple and fast connection can be made, including transactions and data sharing.

## 2.2 History and Evolution

This section will summarize the facts that made the appearance of NFC possible and through its evolution to its actual state.

### 2.2.1 Origins

NFC is a quite new standard based on the already established RFID. NFC keeps most of the characteristics of its forerunner and enhance its possibilities integrating many other contact-less short range technologies. NFC inception takes place during 2002, created mainly by Philips and Sony which were searching for a protocol compatible between existing RFID cards, such as Mifare and Felica. That is why NFC is also an extension of standard proximity cards ISO 14443.

### 2.2.2 RFID

Radio Frequency Identification (RFID) is a system that uses electromagnetic waves to communicate between a device and a tag for the purposes of automatic identification and tracking. The RFID tag may contain a battery or not, in which case it is fed via magnetic induction from the reader device radio waves [17]. Some advantage against other solutions like bar codes or QR codes are that RFID communication does not need to be in line of sight and therefore the token to read can be miniaturized and embedded. A wide range of applications exist for this technology, each one using different frequency bands depending mainly on the communication range.

NFC is a newer standard that take root from RFID, it preserves many of the characteristics and functions but, in principle, is not intended to substitute. NFC enlarge the possibilities that RFID technology can bring. The main advantage is the integration within mobile phones and the increase of security as a consequence of the working range.

### 2.2.3 Milestones

1. 1983: The first patent to be associated with the abbreviation RFID was granted to Charles Walton. [18]
2. 2003: NFC standard issued in 2003. An interface technology for short-range data communication working in the frequency band of 13.56 MHz. [11]
3. 2004: Nokia, Philips and Sony established the Near Field Communication (NFC) Forum. [19]

4. 2006: Initial specifications for NFC Tags and “SmartPoster” records. [11]
5. 2006: The Nokia 6131 NFC was the first commercially available handset with integrated NFC. [15]
6. 2009: NFC Forum Unveiled Target N-Mark. [11]
7. 2009 Jan: NFC Forum released Peer-to-Peer standards to transfer contact, URL, initiate Bluetooth, etc. [11]
8. 2010: Google step in. Samsung Nexus S: First Android NFC phone shown. [20]
9. 2011: Google I/O “How to NFC” demonstrated NFC to initiate a game and to share a contact, URL, app, video, etc. [21]
10. 2011: Research In Motion is the first company for its devices to be certified by MasterCard Worldwide, the functionality of PayPass.[22]
11. 2011: Launching Google Wallet on Sprint and working with Visa, American Express and Discover. [23]
12. 2012: Sony introduces the “Smart Tags”, which use NFC technology to change modes and profiles on a Sony smartphone at close range, included in the package of the Sony Xperia P Smartphone released the same year. [24]
13. 2012: Windows 8 release with NFC API included in the Proximity package. [25]
14. 2013 Jan: NFC Forum Launches Special Interest Groups to Support NFC Market Implementations. [26]

## 2.3 NFC Forum

This section will give an overview of the main promoter of NFC.

### 2.3.1 What is it?

The Near Field Communication Forum is a non-profit association created in 2004 formed by many members in the industry, that promotes the use of NFC.

It collects the set of standards that compose NFC and gives them validity. Any new specification ready to be included in the technology has to be firstly approved by this organization. Through its website, events and campaigns the Forum tries to empower the use of NFC in consumer electronics such as mobile devices and PCs, remarking the impact this technology can (and will) have both in the industry and the final user.



**Figure 2.13:** NFC Forum logo [11]

The NFC Forum was founded in 2004 by Sony Corporation, Nokia Corporation and NXP Semiconductors (formerly Panasonic Semiconductors), and many members have joined since then. Besides the NFC Forum welcomes liaison relationships with relevant external organizations, like the WiFi Alliance and EMVCo.

### 2.3.2 Mission and Goals

The forum states the following as its mission [11]:

*The Near Field Communication Forum was formed to advance the use of Near Field Communication technology by developing specifications, ensuring interoperability among devices and services, and educating the market.*

More precisely it has a set of defined goals to achieve the mission which can be split in the following:

- Develop standards-based Near Field Communication specifications.
- Encourage the development of products using NFC Forum specifications.
- Work to ensure that products claiming NFC capabilities comply with NFC Forum specifications.
- Educate consumers and enterprises globally about NFC

These aspects are essential to ensure interoperability for NFC devices and protocols and a stable and secure framework for application development. The NFC Forum provides a highly stable framework for extensive application development, seamless interoperable solutions, and security for NFC-enabled transactions. It has formally outlined the architecture for NFC technology [11]. The Forum has released 16 specifications to date. The specifications provide a “road map” that enables all interested parties to create powerful new consumer-driven products.



**Figure 2.14:** NFC Forum environment [11]

### 2.3.3 Members

Members in the NFC Forum are companies in different industries that aim to get benefited from NFC technology and the relation with the forum and the other affiliates. As it can be expected many different business are interested in joining the group, from chip and electronic devices manufacturers, to application developers and financial institutions. The figure 2.14, which was extracted from [11] shows the complete business drivers that surrounds the forum. The Forum has now more than 170 members, which are divided depending on the degree of economical involvement in five different groups: Sponsor Members, Principal Members, Associate Members, Non-Profit Members and Implementer Members. At the same time each one has distinct rights inside the committees of the forum. The NFC Forum has coordinated the work of hundreds of member organizations by creating Committees and Working Groups.

- Special Interest Group (SIG) Committee, which has five Working Groups that are open to all members, with voting rights for Sponsor and Principal Members
- Technical Committee, which has three Working Groups that are open to all members except Implementer members, with voting rights for Sponsor and Principal Members

Group	Companies
Sponsor Members	<b>Sony Corporation, Nokia Corporation, NXP Semiconductors</b> , MasterCard Worldwide, NEC Corporation, QUALCOMM Inc., Broadcom Corporation, Intel Corporation
Principal Members	American Express, AT&T, Barclaycard, Cambridge Silicon Radio, CANON INC., <b>Google Inc.</b> , Hewlett Packard, Huawei Technologies Co, Ltd., <b>INSIDE Secure S.A.</b> , Kivio Inc., LG Electronics, NTT DOCOMO, INC., <b>PayPal</b> , Texas Instruments Incorporated
Associate Members	Acer Incorporated, Agilent Technologies, Bell Mobility, China Mobile Communication Corporation, Clear2Pay (Integri), ERICSSON AB, Gemalto NV, Hitachi, Lenovo, MediaTek, Micropross, <b>Microsoft Corporation</b> , Ltd., National Instruments
Implementer Members	3ALogics Inc., Advanced Card Systems Ltd., ALPS Electric Co., Ltd., HTC Corporation, Identive-Group, Inc., <b>Isis</b> , ITN International, Inc., JC Square Inc., Johnson Controls, Inc., Kyocera Corporation
Non-Profit Members	GSM Association, Hungarian Mobile Wallet Association, ITSO, Japan IC Card System Application Council (JIC-SAP), National Payments Corporation of India, Universidad Pontificia de Salamanca- MIMO, WIMA

**Table 2.3:** NFC Forum members

- Compliance Committee, which has three Working Groups that are open to all members except Implementer members, with voting rights for Sponsor and Principal Members

Table 2.3 lists some of the main members of each group. The full board of members is in the NFC Forum website [11]. Some of them have been highlighted for its importance.

### 2.3.4 N-Mark

The N-Mark is the universal symbol for NFC that the NFC Forum has developed. It allows the consumers to easily recognize NFC-enabled and services. Displayed in a screen, packaging or product, it indicates that the product or service has NFC capabilities.



Figure 2.15: N-Mark [11]

## 2.4 Technical Specifications

This section will go deeper in the technical specifications. A description of the architecture and data format will be addressed along with more details of tags. To conclude a security analysis with the main concerns is presented.

### 2.4.1 NFC Architecture

NFC is a converging evolution of already existing standards, aiming to create a contactless technology that allows interoperability. The NFC Forum defines the components that belong to the NFC architecture.

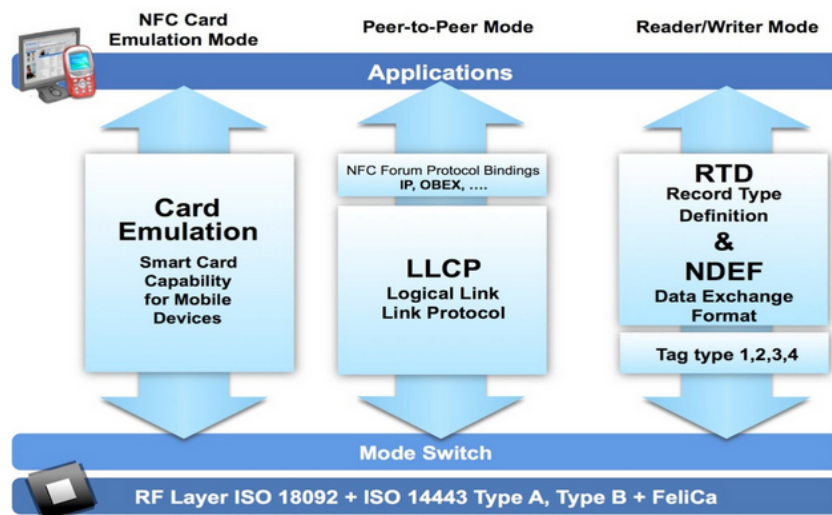


Figure 2.16: NFC Forum Architecture [11]

Figure 3.4 describes the different components of the technical architecture of NFC in more detail, making a distinction between operating modes. Some of the existing recognized standards are ISO/IEC 18092, ISO/IEC 14443-2,3,4, JIS

X6319-4, among others. NFC Forum specifications include parts of these standards. Therefore, compliant devices behave in the most consistent way, and the evolution of existing infrastructure toward full NFC is facilitated.

#### 2.4.1.1 NFC Forum Protocols

In this section a brief description of the protocols and specifications issued by the NFC Forum will be given. For a full and more accurate specification refer to the datasheets [27].

**2.4.1.1.1 NFC Analog Technical Specification** Defines the analog radio frequency characteristics of an NFC Forum device, for example, shape and strength of RF fields. The specification characterizes and specifies the externally observable signals for an NFC-Enabled Device without specifying antenna design, for the different NFC technologies (ISO 14443-3A, ISO 14443-3B, and JIS 6319-4), operating modes and for all the different bit rates (106kbps, 212kbps, and 424kbps) [27].

**2.4.1.1.2 NFC Digital Protocol** Specification for digital aspects for NFC-enabled devices communication, defining the building blocks to be implemented on top of the ISO/IEC 18092 and ISO/IEC 14443 standards. This layer harmonizes contactless technologies defining options and limits of the standards, such as bit level coding, bit rates, frame formats, protocols, and command sets. It can be bound to the LLCP protocol [27].

**2.4.1.1.3 NFC Activity Technical Specification** It is based on the building blocks of the digital protocol, and it specifies the activities, i.e. what to do, to set up an interoperable communication. The specification describes how the digital protocol specification may be used to setting up a communication, being the specific combination of several activities called Profile. Different Profiles are available and can be modified as well [27].

**2.4.1.1.4 LLCP** Logical Link Control Protocol (LLCP). It corresponds to an OSI layer-2 protocol based on the industry standard IEEE 802.2, what makes it able to multiplex between protocols, supporting network protocols such as OBEX and TCP/IP. LLCP is intended to carry out peer-to-peer communication in bidirectional communications. It describes link activation, supervision and deactivation for two service types, connectionless and connection-oriented. LLCP improve the basic functionality offered by ISO/IEC 18092, without any impact in the interoperability [27].



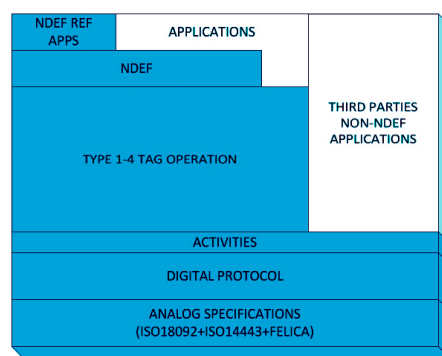
**2.4.1.1.5 SNEP** Simple NDEF Exchange Protocol (SNEP). Protocol that allows a peer-to-peer NFC application to exchange NDEF with another NFC device. It is implemented just on top LLCP in connection-oriented transport mode, providing reliable data transfer [27].

**2.4.1.1.6 Tag Operation Specification** Description of the four tag types that are operable with NFC devices, defining how to read and write from/to a tag. It is considered the backbone of interoperability for NFC, as different tag providers and manufacturers provides the same consistent user experience [27].

**2.4.1.1.7 NCI Technical Specification** NFC Controller Interface (NCI) defines a standard physical interface for NFC controllers to communicate with the NFC device main application processor. NCI provides a logical interface that can be used with different physical transports, such as UART, SPI, and I2C. In this way device manufacturers can integrate more easily different chipsets no matter what the chip manufacturer is or the device wanted to be built [27].

## 2.4.1.2 Reader-Writer Protocol Stack

The main building block for the Reader-Writer operating mode is the *Tag Operation Specification*. Lower level protocols/specifications are shared among the different operating modes, and then are built up from the NFC Digital Protocol by means of the corresponding Activities and Profiles. Tag types will be examined in more detail in section 2.4.3. There are some NFC Forum specific applications, the NDEF Reference Applications, such as smartposters or handover. In addition, third parties applications may be programmed to follow the NFC Forum specifications or not.



**Figure 2.17:** Reader-Writer protocol stack

### 2.4.1.3 Peer to Peer Protocol Stack

Peer to Peer is possible thanks to the LLCP protocol. Furthermore, it gives a flexibility to this operating mode that makes it unique and suitable for almost every application. The role of SNEP has already been described. NPP is an homonym developed by Google before SNEP specification was released. Yet the most common way to go is using the NFC Forum defined protocol for NDEF transmission, there are some other registered protocols, e.g. OBEX, for which the forum provide the *Protocol Bindings*. Finally, as in Reader-Writer mode, other applications can use their own protocol, although being NFC Forum compliant is recommended.

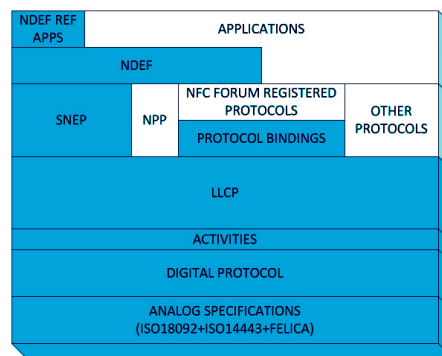


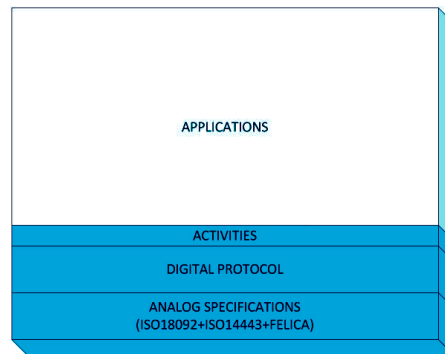
Figure 2.18: Peer to Peer protocol stack

### 2.4.1.4 Card Emulation Protocol Stack

Unlike the two other operating modes, the Card Emulation lacks from standards defined by the NFC Forum at a higher level. So far the few applications made public have used proprietary procedures, making them incompatible. Specification for this mode may be released in the future as the industry evolves to a common solution.

### 2.4.1.5 Hardware

At a hardware level, devices that want to acquire NFC capabilities has to integrate a NFC microcontroller. This microcontroller contain at least one chip called the NFC radio, which has the antenna to generate the RF field and the processor to control and produce the signals that drive the the antenna. Besides, there is need of another chip or mechanism to run applications which deal with sensitive data and a certain degree of security is required. This is called the Secure element. The NFC radio is connected to a host controller, which can be the baseband or

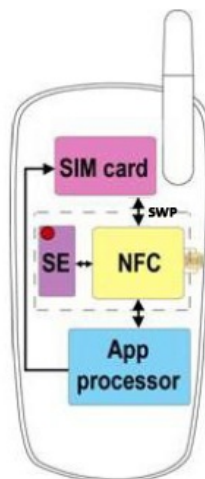


**Figure 2.19:** Card Emulation protocol stack

application processor on a phone.

The secure element (SE) can be implemented in several ways. It must contain a secure processor, a tamperproof storage and execution memory. Note that the secure element's processor has to be different from the host processor to ensure the independence and, thus, a secure path for the data. The SE contains applications which rely on secure keys running inside the secure processor. Applications running on the secure element typically do it on a Javacard OS.

**2.4.1.5.1 NFC in Mobile Phones** The architecture of the NFC chip in mobile phones can be regarded in the following figure. As it can be seen the mobile



**Figure 2.20:** NFC-enabled mobile phone internal scheme [Smart Card Alliance]

phone holds a NFC chip that contains the microcontroller connected to the an-

tenna an communicating with the mobile processor. Also in the same chip a SE is embedded, this element is an smart card and is usually delivered along with the NFC microcontroller in the same package.

There are multiple ways that the SE may be connected to the radio:

- Smart Card: normally integrated with the NFC microcontroller. It belongs to the handset manufacturer.
- SIM (UICC): connected through SWP (single wire protocol), provides the secure environment to install applications to make transactions. It belongs to the network operator.
- SD Card: not showed in figure 2.20. An SD card inserted in the slot could be used to work as the SE. More complicated as different mobile phones handle do not handle the cards in the same way.

### 2.4.2 NFC Data Exchange Format (NDEF)

The NFC Data Exchange Format (NDEF) is one of the initial specification issued by the NFC Forum. NDEF specifies a common data format for NFC Forum-compliant devices and NFC Forum-compliant tags. [28]

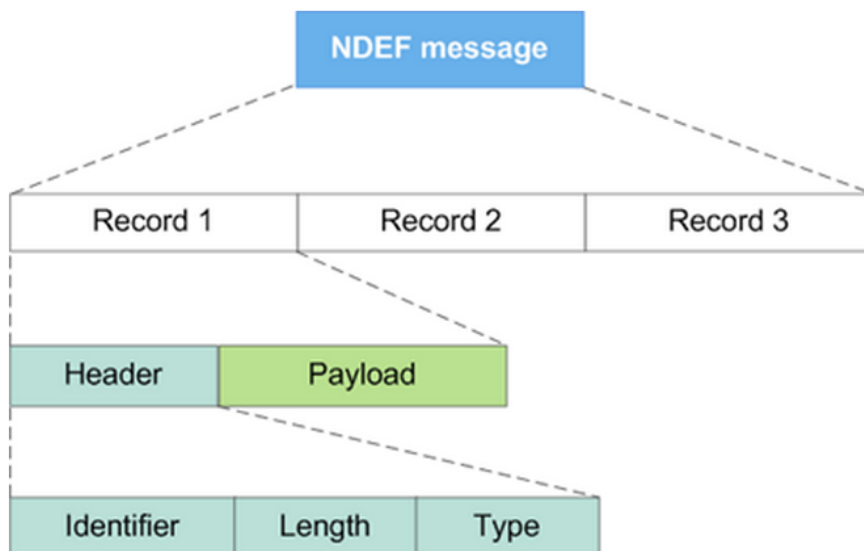


Figure 2.21: NDEF Message layout [10]

NDEF is specified to allow interoperability when transferring data through NFC interface between enabled devices or to store in any storage medium such as a tag. It is a lightweight and compact binary format that can carry URLs, vCards, and NFC-specific data types[10]. Each NDEF message can transport multiple payloads of any type encapsulated in a lower common data format called NDEF Record. In this way NDEF hides all the specific details of the application information and simply relies in the protocol stack to transmit the message. To the receiver, the NDEF message will appear just as a sequence of records with known format and easy to interpret. Figure 2.21 shows what has just been explained.

The NDEF message lacks of header and it contains a minimum of one record and the maximum is unbounded. The different records are cut out looking inside the record's header in which the payload length is specified. An identifier and the payload type is indicated as well. This last one is specially important to be able to parse the payload and later will be explained in more detail. Other important fields are the MB (Message Begin) and the ME (Message End) which are used to identify if the record is the first or the last one in the message respectively. Actually, the header is not as simple and a full description of all the fields is present in figure 2.22. NDEF records do not carry an index number, the order is the one followed when the records are serialized. Moreover, NDEF messages must not be nested using MB and ME. To do so, a whole message has to be encapsulated in another NDEF message (more precisely into a record of it).

The NDEF record information is presented at an octet level. The transmission order is from left to right and from top to bottom in such a way the most significant bit of an octet or octet string is the leftmost bit, being the first one to be transmitted.

The string contained in the TYPE field specifies the name of the record type (record type name). Record type names are used by NDEF applications to identify the semantics and structure of the record content. Each record type definition is identified by its record type name. The record type name format is specified in the TNF (Type Name Format) field of the NDEF record header. The TNF field value indicates the structure that will follow the TYPE field. Record type names can absolute URIs, MIME media types, NFC Forum external type names, or well-known NFC type names.

This combination of TNF and record type name makes the definition of types very flexible, while congruous with existing types specifications, allowing every third party to create its own record type just following the guidelines.

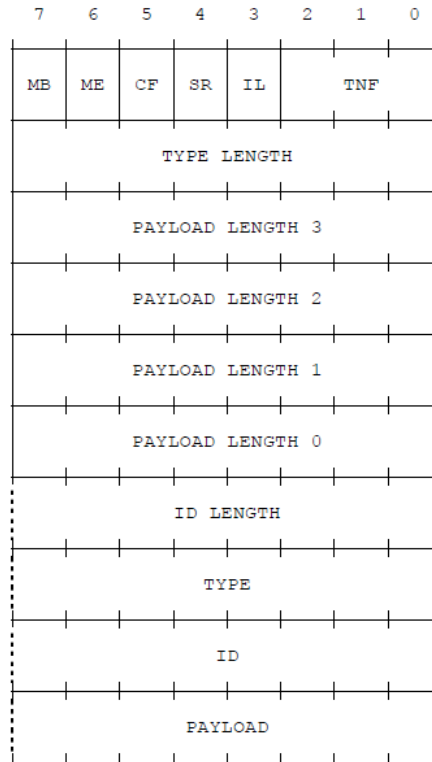


Figure 2.22: NDEF Record layout [28]

### 2.4.2.1 RTD (NFC Record Type Definition)

The RTD specification of the NFC Forum provide the guidelines for the specification of the record type standards that can be included in NDEF messages. As stated before the record type formats can be MIME media types, absolute URIs, NFC Forum external type names and well-known NFC type names. This specification covers the last two of them, as the others are already well-defined in RFC.

The NFC Forum Well-known Type is a dense format designed for tags. It is meant to be used in case there is no equivalent URI or MIME type available, or when message size limitations require a very short name. The Well-known Type can take different forms which will be indicated in the TYPE field, the following are defined [28]:

- Text Record Type: record that contains ASCII characters, i.e. plain text. It can be used to describe any information or, for example, the contents of the tag.
- URI Record Type: NFC RTD describing a record to be used with the NFC

Data Exchange Format (NDEF) to retrieve a URI stored in a NFC-compliant tag or to transport a URI from one NFC device to another.

- Smart Poster Record Type: defines how to put URLs, SMSs, or phone numbers on an NFC Forum Tag or how to transport them between devices.
- Signature Record Type: contains a digital signature related to one or more records within an NDEF message. The signature can be used to verify the integrity and authenticity of the content, i.e., the data records that have been signed.

The External Type Name is meant for organizations or third parties that wish to self-allocate a name space to be used for their own purposes. The pattern to follow is explained in the RTD technical specification document.

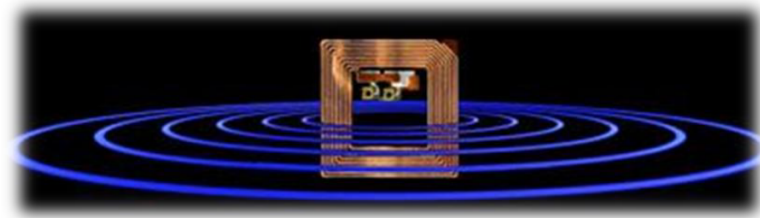
### 2.4.3 Tags

NFC Tags are the passive elements that can be used to store information and communicate with any NFC powered device (in reader/writer operation). Basically the tag is a loop antenna, designed to capture energy and transmit responses, provided with a small chip with EPROM memory, which will keep the received data. Due to the simplicity of its components, the integration is easy and can be done in a flat surface. This kind of tags are easy to fabricate and can be purchased economically in big amounts, and its price is expected to lower as NFC definitely hits the mass market.

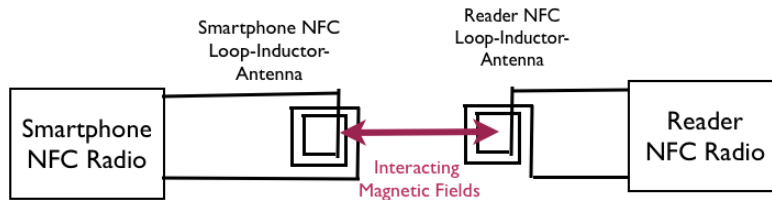
The operation mode is as follows: when the active device (e.g. mobile phone) touches the tag, the small circuit of the tag is fed by the electromagnetic field created by the active player. The energy extracted is enough to activate the circuit works producing a modulated backscatter (signal reflection in the transmitted direction) to communicate with the reader. This fundamental scheme is used both for read and write, the main difference is the response and the storage of the transmitted message. Lecture range is, theoretically, up to 10cm. However test run with commercial mobile phones demonstrate that the typical range is below 3cm.

Tags can be used to store urls, vcards and many other URI types that can be placed, for instance, in so called smart-posters. Also classical applications from RFID can be performed such as transport, identification and access cards with an integrated passive tag. In any case the size of the stored information has to be small. More precise data will be given when analyzing the different types of tag. In the first NFC Forum release in which the a general architecture of the technology was presented, it was defined four types of tag. Designed with numbers from 1 to

4, having each one different formats and capacities, this specification was designed to guarantee the compatibility between technologies, including existing RFID ISO 1443 (Type A, B) ones and Sony Felica ISO 18092.



(a) Electromagnetic field



(b) Induction scheme

**Figure 2.23:** NFC Forum Tag Types

#### 2.4.3.1 NFC Forum Type 1 Tag

Type 1 Tag is based on ISO/IEC 14443A. Tags are read and re-write capable and users can configure the tag to become read-only. Memory size is 96 bytes and expandable to 2 kByte. Transmission speed is 106 kbit/s. Due to its simplicity the price is low [29].

#### 2.4.3.2 NFC Forum Type 2 Tag

Very similar to Type 1. Type 2 Tag is based on ISO/IEC 14443A. Tags are read and re-write capable and users can configure the tag to become read-only as well. Memory availability is 48 bytes and expandable to 2 kByte. Transmission speed is 106 kbit/s. Again the price is low [29].

#### 2.4.3.3 NFC Forum Type 3 Tag

Type 3 Tag is based on the Japanese Industrial Standard (JIS) X 6319-4, also known as FeliCa. Tags are pre-configured at production to be either read and re-writable, or read-only. Memory availability is variable, theoretical memory limit is 1 MByte per service. Transmission speed is 212 kbit/s or 424 kbit/s. This type



of tag gives more flexibility to carry data for more complex application. The price in this case is higher than previous ones [29].

#### 2.4.3.4 NFC Forum Type 4 Tag

Type 4 Tag is fully compatible with the ISO/IEC 14443 standard series (A and B). Tags are pre-configured at manufacture to be either read and re-writable, or read-only. The memory availability is variable, up to 32 KBytes per service. The communication interface is either Type A or Type B compliant with a speed of 424 kbit/s. Variable price depending on the manufacturer [29].

#### 2.4.3.5 Comparison

Regarding the determining aspects when choosing a tag for specific application, *memory* and *price* will be the most critical. As it can be seen in table 2.4 these factors follow an inverse relation. 96 kBytes is enough to store a simple NDEF message with one NDEF record or, perhaps, two; but if the number of records is increased or more sophisticated data is carried in the record will not be possible to be saved.

So, for instance, if the application that is going to be developed requires lots of tags, the information stored in them shall be small enough to fit in Type 1 or 2 tags to keep the project feasible. Unless the resources are not limited, what in general is never true.

Another possible scenario could be that the equipment necessary is already available from a previous RFID application, lets say Felica readers and writers. In that case would be preferable to buy only tags for that technology than to invest in a whole new NFC compliant equipment. Therefore a balance has to be done taking into account the scope of the project and the characteristics of each tag type.

### 2.4.4 Security

Security is one of the main concerns when it comes to deal sensitive data of the users using NFC. By itself, NFC does not provide any type of security service, having to be implemented in higher layers. Nevertheless, due to the characteristics of the radio interface and the short range in which the communication takes place, the security is almost guaranteed for typical threats.

	Type 1	Type 2	Type 3	Type 4
RF Interface	ISO 14443 A-2	ISO 14443 A-2	Felica (ISO 18092)	ISO 14443-2
Initialization	ISO 14443 A-3	ISO 14443 A-3	Felica (ISO 18092)	ISO 14443-3
Speer	106 Kbits/s	106 Kbits/s	212 Kbits/s	106-424 Kbits/s
Protocol	Specific Command Set	Specific Command Set	Felica Protocol	ISO 14443-4, ISO 7816-4 Commands
Memory	Up to 1KB	Up to 2KB	Up to 1MB	Up to 64KB
Cost	Low	Low	Moderate	Moderate
Use Cases	Tags with small memory for single application		Flexible tags with larger memory offering multi-application capabilities	

**Table 2.4:** NFC Forum tag types

An attacker can capture the data transmission via an antenna which may be located at a maximum distance of a few meters. In addition, if the target device to listen is in passive mode, it does not generate its own RF field, being much more difficult to capture the data. Besides, the short range in which the attack has to be performed could make the user to realize it is being attacked. The destruction of the data being transferred is, however, as simple as using an RFID jammer. Although there is no way to prevent the previous attack, it can be detected so the harm is minimum.

#### 2.4.4.1 Attacks

In this section the most alarming security holes are going to be described. These attacks have already been studied and some solutions have been suggested as well[30][31]. In spite of that, none solution has been included in the ISO standard and NFC still does not offer protection against not authorized listenings. So perfect security can only be obtained when dedicated cryptography is used to establish a secure channel between communicating devices.

**2.4.4.1.1 Eavesdropping** Eavesdropping (also known as sniffing) is the most evident attack on wireless communications. An attacker can listen to any data being sent between devices by means of an antenna and analysis equipment. As it have been said the close proximity and low power makes the attack, in this case the

eavesdropping, harder to perform with NFC than any other wireless technology with bigger ranges. How hard it exactly is to eavesdrop a NFC conversation unfortunately depends on too many physical (RF signal emitted, antennas used, receiver used) and environmental (location, noise, position of the attacker) factors. Also, eavesdropping is highly affected by the communication mode. A passive device which does not create its own RF field is much harder to eavesdrop on than an active one. The typical range in which an attacker can eavesdrop is within 10m and 1m for active devices and passive devices.

**2.4.4.1.2 Data modification** Data can be easily destroyed using an RFID jammer causing denial of service and it is almost impossible to prevent. The only thing NFC devices can do is to check the RF field during transmission, detecting collisions and, thence, attacks. However, attackers are usually interested in more useful attacks such as modifying the data being sent out. But modifying the data in such a way that it appears to be valid to users is way more complicated.

An intruder has to deal with the single bits of the RF signal. The feasibility of this attack, is amongst others subject to the strength of the amplitude modulation. If data is transmitted with the modified Miller coding and a modulation of 100%, only certain bits can be modified. This type of modulation makes possible to eliminate a pause of the RF signal, but it is impossible to generate a pause if it has not previously been one. Therefore, only a 1 followed by another 1 could be changed. On the other hand, transmitting Manchester-encoded bits with a modulation ratio of 10% allows a modification attack on all the bits.

To conclude, recalling table 2.1, data modification in a communication is possible for bit rates higher than 106 kbit/s and possible only to some small extend for the 106 kbit/s bit rate.

**2.4.4.1.3 Man-in-the-middle** In MITM attacks, the attacker sits in between two parties that are communicating, presumably, securely with each other. The attacker intercepts messages sent by one device and then sets up a new communication with the second device. Then, he catches the response of the second device and sends his own response to the first device. To ensure both devices do not discover the fraud, the attacker has to jam the signal from the first party while at the same time sends its own data out. The problem here is that NFC devices are able to receive and transmit data at the same time. Thus, collisions can be detected when checking the radio frequency field and received and transmitted signal do not match. Therefore jamming or incoherent signals can be detected. Plus the fact that NFC is a short range communication, MITM type of attacks are practically

impossible to carry out.

However it has been shown that some kind of relay-attacks can be feasible (as NFC include ISO/IEC 14443 protocols) using only two NFC-enabled mobile phones [32].

**2.4.4.1.4 Lost property** Losing the NFC card or the mobile phone will open access to any finder and act as a single-factor authenticating entity. If the mobile phones is protected by a PIN code, this would act as the unique authenticating factor. In any case, this problem is present also in the conventional wallet or cards and should not persuade users to avoid NFC.

**2.4.4.1.5 Walk-off** NFC function normally is shut down when the transaction is completed or after a period of inactivity (time-out closing to protect the open access to NFC secure data). Attacks could happen while the bearer “walks-off” and the countdown to shut down the communication has not finished. Additional features to cover such an attack scenario dynamically shall make use of another wireless authentication key that remains with the bearer during the activity period to validate its identity.

## 2.5 NFC Software Development

This section will review the software development alternatives existing for mobile phones. The main characteristics will be studied and compared between them.

### 2.5.1 APIs

The review on the APIs found will be done highlighting what have been considered the main characteristics that an NFC API can have, that are the following.

1. Programming language: in which it has to be programmed. It will depend on the underlying operating system the API is made for.
2. Supported operating modes: if every mode can be programmed or only some of them.
3. Supported data format: two possible scenarios. The API supports NDEF data (compulsory) or it also support raw data.
4. Access to the secure element: whether is it possible to access the secure element or not. Tightly related to the operating system and manufacturer.

5. Complexity/Ease of use: level of usability and flexibility.
6. Current state: the situation that surrounds the API.
7. Other comments: relevant aspects of a particular API that distinguish it from the others.



**Figure 2.24:** NFC development. OS logos

### 2.5.1.1 Earliest APIs

As it has been mentioned before, Nokia was the first company integrating NFC within its commercial mobile phones. Thus, first APIs that appeared were made for Symbian, Nokia's flagship OS.

JSR-257, released in 2006 and included in the Java ME platform:

1. Java.
2. Every operating mode supported at a low level. In other words, the NFC Forum defined operating modes had to be built by the combination of simple instructions.
3. Every kind of data format allowed. Including a dedicated class for NDEF format.
4. Possibility of unlocking the embedded secure element. Previous request to Nokia.

5. Complex API, specially compared with the more recent ones.
6. Abandoned due to Symbian decay.
7. Complete compatibility with ISO/IEC 14443-4, i.e, some no NFC Forum compliant tag/cards are supported.

QtMobility 1.2, 2010:

1. C++.
2. No card emulation mode. Otherwise, operating modes supported at a low level.
3. Both raw and NDEF format data.
4. Possibility of unlocking the embedded secure element. Previous request to Nokia.
5. Simpler API than JSR-257, similar functionality level than Android one.
6. Complete set of mobility functions, like NFC, hardly used.
7. Only Symbian compatible although is an independent platform.

### 2.5.1.2 Android

Google was one of the first “software companies” on betting for NFC. The Android API was released in 2010 gaining considerable impact after 2011 Google I/O event.

1. Java.
2. Reader-Writer and Peer to Peer fully implemented. No Card Emulation though.
3. Both raw and NDEF format data.
4. No official mean of secure element unlocking.
5. Fairly simple usage. Also flexible as it allows many low instructions to be accessed. Classes for messages, tags and connection management.
6. In development. Upgrades have been released with new Android versions. Card emulation expected in next upgrades.

7. Implements a port of the proprietary library libnfc-NXP which indeed supports card emulation. Also MIFARE classic tag type despite the fact it does not correspond with any NFC Forum specification. WiFi and Bluetooth handover functions already implemented.

### 2.5.1.3 Windows Phone 8

On October 29, 2012, Microsoft released Windows Phone 8, a new generation of the operating system. This system includes NFC capabilities. Previously, a partnership with Nokia had been announced. An alliance that will benefit NFC environment. Within the proximity set of classes of WP8 the ones to deal with NFC can be found.

1. C#.
2. Reader-Writer and Peer to Peer fully implemented. Emulation Card through SWP and SIM card as secure element.
3. Only NDEF format data. Very easy to create NDEF messages and records.
4. Need of a compatible Mobile Network Operator that allows to unlock the secure element for developers.
5. Easiest of the existing APIs. Nice *wrappers*. Some flexibility is lost, low level functions not accessible.
6. Recently released. A higher influence is expected to be achieved as the operating system gets widespread.
7. Allows P2P session encryption with a symmetric key (AES encryption standard).

### 2.5.1.4 Blackberry OS

Blackberry API dates back to 2011 when the seventh version of its operating system was released. Blackberry had been experimenting with NFC for a while already, becoming one of the few manufacturers certified by MasterCard to make mobile payments.

1. Java.
2. Every operating mode supported. Card Emulation supported both with SIM and smart card.
3. Both raw and NDEF format data.

4. Possibility for Blackberry developers to unlock the smart card secure element.
5. More complex API, but less restrictive in return.
6. Despite being one of the last to be released it presents a complete set of functions perfectly integrated with Blackberry devices.
7. Interesting API from the developer point of view. Blackberry OS still creates doubt.

#### 2.5.1.5 Other Open Source APIs

Open NFC [35] is a software stack implementing NFC functionalities on top of an NFC controller chip-set. Sponsored by INSIDE Secure, is an open source NFC software project that wish to become the reference NFC software stack for every platform. Supports all the functions commented for other APIs and is independent of the underlying operating system, previous porting to the corresponding OS. Also different hardware could be adapted by means of a hardware abstraction layer present in the stack.

All these huge advantages have two big inconveniences. The main companies driving the NFC market have opt for other solutions already tested and, besides, is not straight forward to put the stack to work.

## 2.6 Near Future Foresight

This section will bring to a close the state of the art. The market situation will be exposed and future research analyzed.

### 2.6.1 Market Acceptance

The truth is that NFC has not reach the point yet to say is well established and everybody has heard of it. Many companies have decided not to bet for it, the most representative example is Apple. Although there were rumors of iPhone 5 bringing NFC within it, finally it was not included. This means that roughly a third of the mobile phones will not have NFC during years, at least the lifetime of current iPhones. A big slow down to NFC Forum expectation. Moreover, it is still not widespread among all the rest of cell-phone manufacturers.

On the other hand, there are a bunch of big companies pushing ahead. Sony has launched a set of consumer electronics which all have NFC perfectly integrated,



allowing all the devices to connect between them and share content instantly, highlighting the benefits of this “new” technology. It is likely that other companies follow Sony in this initiative.

A long path is yet to be covered. The development of new applications and systems is the way to follow to create a rewarding environment to the user. Prototypes are being tested in different cities all over the world and will hit the market gradually as these test become successful.

Additionally, the mobile payment through NFC is awaited. This is thought to be the *killer app* that will make everybody want to buy a NFC-enabled phone. The problem is that the interest aroused around this functionality has brought many big players to fight fiercely to be the one providing this service. With the added difficulty of the secure element management needed to make this kind of transactions, for which an agreement with a trusted service manager is required.

### 2.6.2 Future Research

Since NFC came to light back in 2003 lot of effort has been deployed on making the technology ready to compete against other solutions. The NFC Forum members have been the main promoters according to the specifications.

Developers embraced NFC eager to build new applications and solutions for all the operating modes. Read-Write and Peer to Peer were dominated, and have become quite simple to work with. However, the more interesting mode, from a developing point of view, is hard to put it to work. Card Emulation should be easily accessible to developers to bring up again their enthusiasm. For that purpose more agreement will be needed between the trusted service managers and network operators, phone manufacturers or other entities controlling the secure element on mobile phones. Besides, additional NFC Forum specifications for this operating mode may help to reach a consensus.

Leaving apart Card Emulation and taking into account the two other modes are easy to implement, research struggle should put into creation of rich systems that really add value to the consumer’s use of NFC. Finally, interoperability, which is fairly good already, must be granted between all the devices using NFC and a definite step should be made.

# 3

## SYSTEM ANALYSIS AND DESIGN

This chapter will present the results of the analysis carried out in the first stage of the development process, when most of the decisions were taken with respect to functionalities to implement, structure of the application and interaction with the user. UML diagrams will be presented when relevant.

### 3.1 Naming and Logo

Every worthwhile application needs a recognizable name and an attractive logo. The main ideas behind the application branding will be described briefly.

#### 3.1.1 Name

Four very simple premises were set as constraints. The name must:

- Contain NFC in the name: NFC is still not well known and needs recognition and promotion. At the same time, as people starts to know NFC more people will discover the application. Thus having the acronym in the name causes a positive feedback.
- Have a meaning related with what the application does: the application can do many tasks like, e.g. reading tags, but its philosophy is to become the neural center of the NFC environment unifying all the content distributed by NFC.

- Sound good: complicated names were avoided. It had to be an easy name to remember.
- Have a domain which is free: a domain in the form *www.yourdomain.com* looks trustful and professional. Sometimes such domains are reserved (either some other activity is being held in the domain or it is just for sale for a big amount of money). Acquiring a domain with the name for future promotion was seen as an important point.

Adding up all together the name **UniteNFC** was created. Being the word *unite* defined by the Oxford English Dictionary [36] as:

*unite: (verb) come or bring together for a common purpose or action*

The main element of the application, NFC tags registered into UniteNFC, will be called from now on *NFC Points*.

### 3.1.2 Logo

The design of the logo followed some of the reasoning exposed in the previous section too. Easy to recognize and contain the N-Mark 2.15. Besides, seemed a good idea that the symbol itself was part of the application interface. As one of the many functionalities is to locate NFC Points in the map, the logo was drawn as a circle with a tip pointing downwards, i.e. resembling a marker.



Figure 3.1: UniteNFC logo

The high resolution (512 x 512 px) logo can be seen in figure 3.1, while the markers with different colors used for the application are shown on figure 3.2.



Figure 3.2: Markers on UniteNFC

## 3.2 Scenarios

UniteNFC was not thought to cover an specific need, but to provide the user with a tool that may be used in different scenarios. These scenarios are related with the possibilities that NFC provides and a new way to interact with the physical world from the digital or vice-versa.

### 3.2.1 Fairs and Events

Imagine a big event like the Campus Party or the NFC World Congress. Normally these events are placed in big constructions where companies share the space and locate their booths where they offer corporative information among others. One original way to distribute such information across the event would be placing an registering an NFC tag in UniteNFC and, if you wish, offer some kind of reward to the ones going to your booth and checking in. Perhaps you are including in your tag a link to all the information the company want to deliver saving this way lots of paper. Moreover, thanks to the social functionalities participants in the event can see, for example, which booths your friends have visited and check out the opinion of all users in the wall of the NFC Point. Why to have a physical map with the location of the booths when you can have it on your phone with additional functionalities.

### 3.2.2 Tourism

UniteNFC allows you to spot the monuments, squares and places worth to visit in a city. In each spot an NFC tag will be available to get historical information about what you are visiting. This information can be an audio record, images, text, etc. In any case the content stored in the tag will be a link to that info since, remember from table 2.4, the small capacity of tags make impossible to store lots of data, so what is normally done is to store a link to the desired information stored somewhere else on the Cloud.

Similar systems already exists using other technologies like Bluetooth. However many advantages are drawn from NFC characteristics along with UniteNFC functionalities. No set up is required and does not require extra hardware, just tap the NFC Point with your mobile and get the information. The exploration is interactive, you see yourself on the map and you can see the NFC Points to which you head. And finally, you obtain extra information from the wall like precise address and other tourists reviews.

### 3.2.3 Augmented Reality Games

Augmented reality is becoming day by day a (forgive the repetition) reality. With it a new way of gaming is appearing as well: augmented reality games. NFC opens a world of opportunities to new ways of gaming across the cities. NFC Points can act as beacons that you need to collect or find the messages inside the tags to solve a mystery or complete a puzzle. This way of gaming falls in between two classical dimensions. Physical and digital, needing an interaction with both to achieve the goals established in the game. This fact should not surprise the reader since NFC acts as an interface between both worlds (needs of a physical proximity to get digital information).

### 3.2.4 Marketing and Publicity Campaigns

NFC tags can be placed in bus stops, posters, magazines... storing a promotion code, a link to an offer or much more. Stores can be located in UniteNFC's map offering some kind of promotion to attract users to the physical store. Advertising in this manner also provides a mechanism to gamificate around a brand, improving user engagement. Furthermore, it gives a channel for users to give reviews on products or services offered in the NFC Points.

## 3.3 Use Cases

This section will deal with the operations that users can do in the application. A use case represents, in a clear and simple way, a coherent functional unit of a system, subsystem or class. Its definition is the main step to determine what the application is capable of.

### 3.3.1 Use Cases Definition

The list of all the functionalities performed by the application are itemized and described individually.

#### 3.3.1.1 View NFC Points in Map

A logged user has to be able to see, located on the map, the NFC Points that surrounds him. They will be identified by the icons in figure 3.2, each color identifying one type of NFC Point.

### **3.3.1.2 Filter NFC Points by Type**

A logged user has to be able to select from a list the type of NFC Point that want to view represented on the map, omitting the ones unselected from the list.

### **3.3.1.3 View a History of Visited NFC Points**

A logged user has to be able to view a list of the last NFC Points that has visited, ordered from most recent to less recent, showing basic information like record date, NFC Point title and type.

### **3.3.1.4 View a History of Registered NFC Points**

A logged user has to be able to view a list of the last NFC Points that has register, ordered from most recent to less recent, showing basic information like record date, NFC Point title and type.

### **3.3.1.5 Register a New NFC Point**

A logged user has to be able to register a new NFC Point, scanning an NFC tag and filling a form with basic information. That NFC Point will be part from that moment of the application, being showed on the map, histories and having a wall.

### **3.3.1.6 Scan an NFC Point**

A logged user has to be able to scan an NFC Point or, in other words, view the content stored in an NFC tag. This action has to be performed from every point of the system when the mobile phone taps the tag. The data stored will be read, parsed and finally shown to the user.

### **3.3.1.7 Connect with Facebook**

A logged user has to be able to connect with Facebook, giving UniteNFC permissions to access his friends list and public on his behalf.

### **3.3.1.8 Add New Friend**

A logged user has to be able to add another user using UniteNFC as a friend. Two alternatives will be possible: through NFC or through Facebook connection.

#### **3.3.1.9 View a List of Friends**

A logged user has to be able to view a list of added friends. In this list, the name of the user and its profile picture will be shown.

#### **3.3.1.10 View Friend Information**

A logged user has to be able to view more detailed information about a friend. For example, a history of NFC Points visited by the friend.

#### **3.3.1.11 Edit User Data**

A logged user has to be able to edit the data that is shown to other users, such as the name and the profile picture. Besides, it has to be possible to change preferences on how some functionalities work.

#### **3.3.1.12 View Walls**

A logged user has to be able to select an NFC Point from a history or the map and view detailed information about it. Including address, last seen date, tag content, mean rating and comments.

#### **3.3.1.13 Rate NFC Point**

A logged user has to be able to rate an NFC Point selecting from a bar a mark from zero to five and submitting the selection. This can be done just once per wall.

#### **3.3.1.14 Comment on NFC Point Wall**

A logged user has to be able to leave a message on an NFC Point wall.

#### **3.3.1.15 Administrate Wall**

A logged user has to be able to administrate walls of those NFC Points registered by himself. The administrator role allows the user to delete comments and make the tag content private or public as desired.

#### **3.3.1.16 Share Information**

A logged user has to be able to share information about the activity taking place in UniteNFC. This information can be shared through the Android sharing system or automatically through Facebook if it is connected.

### 3.3.1.17 Receive Notifications of Nearby NFC Points

A logged user has to be able to receive notifications when is close to an NFC Point. It will be shown in the Android notification bar with a link to the application showing the location of the NFC Point on the map.

### 3.3.1.18 Obtain Application Usage Feedback

The application developer has to be able to obtain feedback from the use that users make of UniteNFC. This can be done directly from user suggestions or from another tool like Google Analytics.

## 3.3.2 Diagram

The diagram with all the use cases defined and its relations is depicted in figure 3.3. The types of relations between use cases are:

- << *Include* >>: when a use case needs of the previous execution of another one.
- << *Extend* >>: when after a use case, another may be executed optionally.

## 3.4 Architecture

Planning the structure of the application constitutes one of the most important steps in the designing process. How the different parts are going to be developed and put together will make the difference between success or failure on meeting the application requirements. The analysis will be divided in two, separating client application (Android) and cloud services, including the back-end that is going to be developed exclusively for UniteNFC.

### 3.4.1 Android

The main aspects of the Android application will be detailed in this section.

#### 3.4.1.1 Hardware permissions

Apart from the essential hardware for the system to run, i.e. processors, RAM and ROM memory, screen, etc; there are other peripherals that an application may make use of. Normally to access extra hardware a permission clause has to be written in the manifest, to let the user decide whether those permissions are



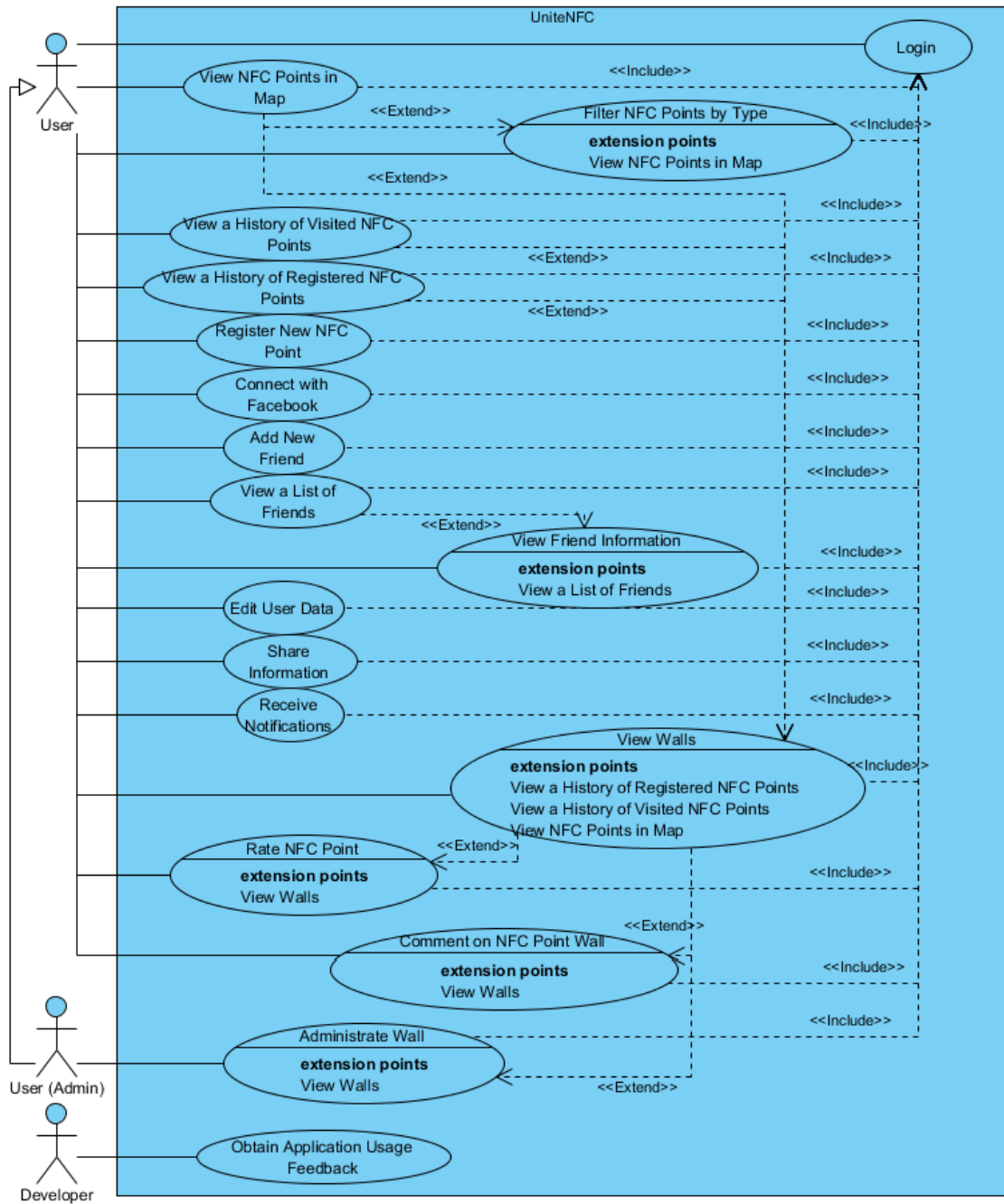


Figure 3.3: Use Cases

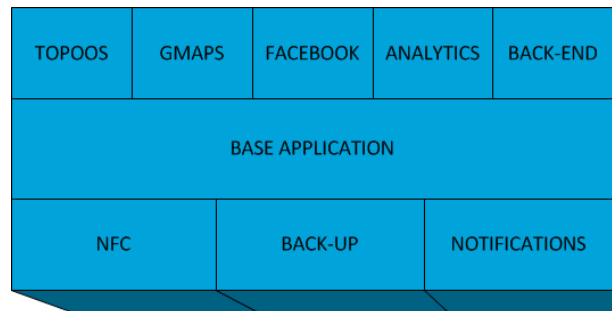
granted or not upon installation. The hardware permissions required for UniteNFC are:

- NFC: access NFC interface to read or write data through it

- INTERNET: permissions to send and receive data through network interfaces
- ACCESS NETWORK STATE: check whether the network interfaces are available and its state
- WRITE EXTERNAL STORAGE: permissions to write to an external memory, e.g. an SD card.
- ACCESS FINE LOCATION: access GPS locations.
- VIBRATE: turn on and off the small motor that makes the device vibrate.

### 3.4.1.2 Code Structure

Figure 3.4 gives an idea of how the Android application will be structured. There will be a main block holding the skeleton of the application, i.e. navigation between activities, fragments and dialogs management, preferences and other structures such as list loaders and model objects. To this base functionalities will be added via new modules.



**Figure 3.4:** Android application structure

Looking at the modules that deal with the Android system (intents, services, broadcast receivers) and lower level operations, three different blocks can be seen. The NFC block will manage all operations regarding NFC, reading and parsing, Android Beam, etc; and its implementation will be explained in detail later. Storing session information and updating it will be carried out by the back-up block which will be synchronized with the back-end. The last module of these three is the notifications, which will be in charge of notifying the user when certain conditions, e.g. being five meters or less near an NFC Point, meet.

The blocks placed on top of the base application are quite similar. These are the ones that communicate with the web services that are being used. The way this

communication is done is through the SDKs that these services provide, except for the back-end communication which has been done through the REST API (HTTP requests) directly, instead of programming another SDK. More detailed information about the web services will be given in 3.4.2 where the cloud architecture is discussed.

### 3.4.1.3 Libraries and SDKs

The Android Software Development Kit includes the basic tools to allow the creation of applications for its operating system. Lots of functions are available to handle all the Android elements and, at the same time, all the Java classical API is compatible.

However, sometimes more sophisticated functions are needed and have been developed by a third party before, being possible to import a library and have them. Besides, normally web services provide with a SDK to ease the access to the service.

The complete list of libraries and SDKs used and needed to build the application is the following:

- Android SDK.
- topoos SDK for Android: facilitate to develop applications on topoos in Android.
- google-play-services: library that includes Google Maps API v2.
- Facebook SDK: functions and objects to integrate easily Facebook into the application.
- gson: library to transform to and from JSON. It maps a JSON string to an specified object or serialize an object to a JSON string.
- mail: necessary to send a mail over an SMTP server.
- Google Analytics Services SDK: contains the latest Google Analytics libraries for mobile devices.

### 3.4.1.4 Background Processing

Android programming language is Java and, thus, concurrency can be handled. The main application thread in Android is in charge of managing user interface, activity life-cycle, callbacks and input events. Intensive tasks can not occur in the

main thread since it will block the other tasks which are essential to the normal flow of the application. Moreover, if common sense is not applied and something heavy is tried to run in the main thread the application will crash.

Internet connections may suffer delays with long waits until all the data is received, specially if using mobile Internet. Tasks such requests to the server have to be pulled out from the main thread using one of the two mechanism available in Android.

**3.4.1.4.1 Threads** The well-known Java Threads, and all the related functions. Part of the Java API. Caution must be taken with this option since synchronization and communication with the main thread are in hands of the programmer.

The minimum amount of code for a Thread to be executed (omitting synchronization and handlers) is:

---

```
new Thread(new Runnable(){
    @Override
    public void run() {
        //your task here
    }
}).start();
```

---

Note that the Runnable does not need to be written every time. If it was going to be used repeatedly, a class inheriting from it would be defined.

**3.4.1.4.2 AsyncTask** The recommended option, provided by the Android SDK. It runs a synchronized thread with callbacks to the main thread during task execution and when the task finishes. AsyncTasks make threading in Android a painless experience for the programmer, who can forget about all the typical problems that come with concurrency.

The minimum amount of code for an AsyncTask to be executed is:

---

```
new AsyncTask<Void, Void, Void>(){
    @Override
    protected void onPreExecute(){
        //executed before task with access to main thread
    }

    @Override
    protected void doInBackground(Void... params) {
```

```

        //your task here
    }

    @Override
    protected void onPostExecute() {
        //executed after task with access to main thread
    }
}.execute();

```

---

Note that all the code of the AsyncTask does not need to be written every time you want to execute it. If it was going to be used repeatedly, a class inheriting from it would be defined.

**3.4.1.4.3 Comparison** Most of the times recommendation was followed and AsyncTasks were used. But still some Threads were used when the task did not compromise any resources and main thread was unnecessary to access after execution. Table 3.1 summarize briefly what has been discussed previously.

	Pros	Cons
Threads	Fast threading	Risky threading
Asynctasks	Safe and rich threading	Slower threading

**Table 3.1:** Thread vs. AsyncTask

## 3.4.2 Cloud Architecture

While most of the user interaction is carried out in the Android application, some services are too heavy, provided by third parties or need to be accessible from everywhere. In those cases, computation is moved to the cloud. UniteNFC, i.e. the client application, then needs to communicate with services resident on the Internet. Figure 3.5 shows a conceptual map of the situation.

In the picture a client application running in an Android mobile phone access to a series of services in the cloud. The services represented are from left to right: Google Maps, Facebook, UniteNFC's back-end, Google analytics, topoos.

### 3.4.2.1 Back-end

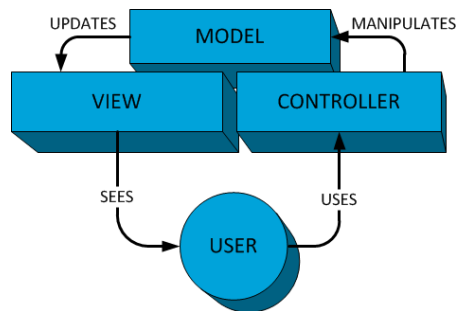
As explained before, a proprietary web service to act as back-end for some functionalities (social features, information about registered NFC Points, comments,



**Figure 3.5:** Cloud Architecture

ratings, etc.) and database storage was developed. Before coding a set of guidelines based on the state of the art on web development were followed.

**3.4.2.1.1 Pattern: Model-View-Controller (MVC)** The goal of this pattern is to improve reusability, decoupling the business logic, i.e. the *Model*, from the presentation, i.e. the *View*. The *Controller* acts as a middle-ware connecting both parts.



**Figure 3.6:** Typical MVC collaboration

Furthermore, MVC is scaffolded by many web application frameworks, making it very suitable to develop a web service with some guarantees.

**3.4.2.1.2 Principle: REST** Standing for Representational State Transfer is the predominant web API design model. The main principles of REST are:

- Stateless client-server protocol: all the information required to process one request is self-contained, so there is no need to monitor communications.
- Requests defined by the HTTP protocol: GET, POST, etc. instructions.
- Universal syntax: resources identified by the called URI.
- Hypermedia as the engine of application state: i.e. state derived from hypertext received from the server, typically HTML, XML or JSON.

These constraints will be respected in order to develop what is known as a RESTful web service.

**3.4.2.1.3 Data Format: JSON** Acronym for JavaScript Object Notation, it is a lightweight human readable format for data exchange. JSON is a language to represent objects, simple data structures and arrays. Due to its simplicity its becoming widely used on the Internet, in many cases substituting XML. An example of the JSON representation of an object modeling the solar system would be:

---

```
{
  "galaxyName": "Milky Way",
  "star": {
    "name": "Sun",
    "ageInBillions": 4.6,
    "isActive": true
  },
  "planets": [
    {
      "name": "Mercury",
      "satelites": null
    },
    {
      "name": "Earth",
      "satellites": [
        {
          "name": "Moon",
          "distanceInKm": 384400
        }
      ]
    }
  ]
}
```

```
    ]
}
```

---

In this example the JSON syntax and data types can be appreciated:

- String: surrounded by double quotation marks (“String”).
- Boolean: *true* or *false*.
- empty or null: keyword *null*.
- Number: double precision.
- Array: surrounded by square brackets ([Array]).
- Object: collection of key:value pairs surrounded by brackets ({Object})

Besides, different items are separated by comma and all the keys are treated as Strings and hence surrounded by double quotation marks.

**3.4.2.1.4 Development Framework: Django** A web application framework is a software tool designed to facilitate and speed up web development. It provides libraries that abstract developers from details such as access to databases, managing user session and follow MVC pattern.

Since there was not previous experience in web development any framework could have been chosen. The main reasons for choosing Django among others were:

- Python: the programming language should be easy to learn. In that sense Python’s syntax seeks legibility and transparency. Besides, it is a multi-paradigm language that uses dynamic type and automatic memory management.
- Documentation and community: support should be as big as possible. It is extremely important to have manuals and examples well explained. Django documentation [3] is carefully written. Moreover, a simple search in Google will show lots of recent entries about the framework indicating its actual magnitude.

Some interesting features of Django are the automatic admin interface that it generates useful to monitor your database, the URL design allowing you to easily stick to REST and the object-relational mapper (ORM) which gives you access to databases through a model entirely defined in Python.



**3.4.2.1.5 Hosting: Heroku** The developed web service need to be hosted somewhere on the Internet to be accessible from any point by the client application. Heroku is a platform in which you can deploy your web application to be hosted on the cloud. It offers a software layer that makes very simple to manage the infrastructure of your application and scale it if necessary. A free developer account is available with limited virtual processing power (1 CPU and 512MB RAM [4]). If more resources were needed, extra power could be hired as your application grows thanks to its distributed architecture.

Heroku offers a lot of plug-ins too, some of them for free, such as PostgreSQL support. Precisely this one will be used, since PostgreSQL is one of the databases supported by Django.

### 3.4.2.2 Other Services

Many of the SDKs used in the Android application and listed in 3.4.1.3 provides the functions to communicate with a cloud service. In this section those services, also shown in figure 3.4, are going to be described.

Access to those services, in general, is granted through token-based authentication [34]:

*The general concept behind a token-based authentication system is simple. Allow users to enter their username and password in order to obtain a token which allows them to fetch a specific resource without using their username and password. Once their token has been obtained, the user can offer the token which offers access to a specific resource for a time period to the remote site.*

This principle is implemented by the OAuth protocol which is an open protocol to secure API access and it defines the flows to obtain the required token.

**3.4.2.2.1 Topoos** Cloud service that provides you with a back-end to construct context-aware applications. Functionalities used from it are location tracking, points of interest registration and localization, user authentication and image hosting.

**3.4.2.2.2 Google Maps** Google Maps gives you the tools to add maps to your application, handling for you how the map is represented, user interaction with it and access to Google Maps servers to get information like routes.

**3.4.2.2.3 Facebook** The biggest social network allows you to profit from it adding Facebook functionalities to your application making it more engaging. UniteNFC can use Facebook relationship to connect new users with friends from Facebook and to publish using hash-tags to share user experience with all their contacts.

**3.4.2.2.4 Google Analytics** Google Analytics makes the task of gathering statistics about usage as simple as adding a line of code in your application. From that premise you can get charts and plots showing you information about users, devices in which the application is installed, locations of the sessions, blocks and exceptions, language configurations used and much more.

## 3.5 Application Flow

Another important decision that is convenient to make at the design stage is how the user interacts with the application, what screens are shown and how the navigation through activities, fragments and dialogs is implemented. To give an understanding of the application flow, sequences diagram, i.e. objects interactions in chronological order, will be shown.

### 3.5.1 Launch Flow

When a user clicks the application icon an *SplashActivity*<sup>1</sup> begins, showing the application logo for a few milliseconds. Then, if the user is already logged-in, it will go directly to the *MainActivity*. Otherwise, a *LoginActivity* will be shown where the user can login or navigate to a *RegistrationActivity*. Once the user is logged-in correctly, as it was said, it goes to the *MainActivity*, in which, if it has not already done before, it will pop out a *FacebookDialog* asking to connect with Facebook.

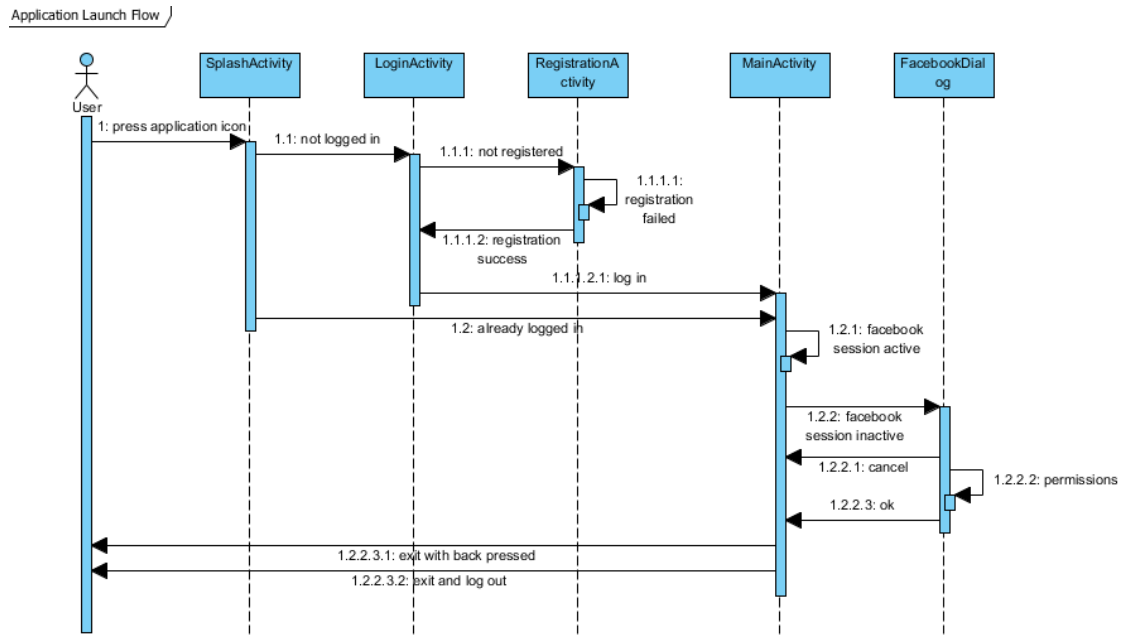
### 3.5.2 Settings, Report Bug and Sharing Flow

While the user is logged-in and in the *MainActivity*, three actions can be performed. First, he can give some feedback by means of the *ReportBugDialog*. Second, he can share what there is on the screen by means of the *ShareDialog*. And third, he can open the *Settings* activity and change some preference thanks to the *CustomSettingDialogs*.

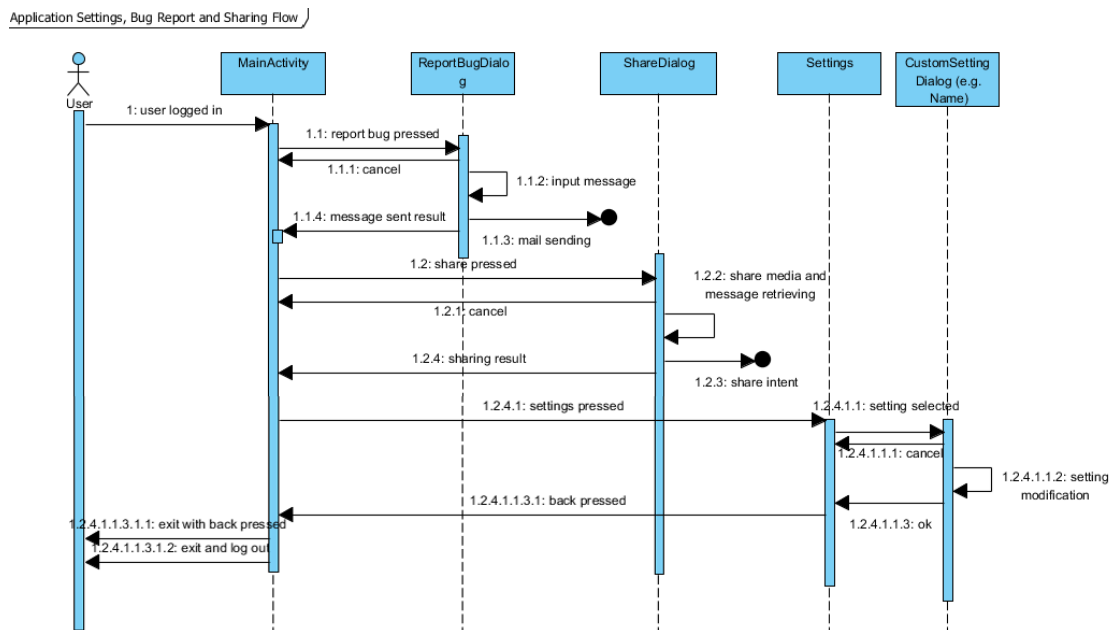
---

<sup>1</sup>Names of activities, fragments and dialogs may not be exactly equal in the application code.

## CHAPTER 3. SYSTEM ANALYSIS AND DESIGN



**Figure 3.7:** Application launch flow



**Figure 3.8:** Application settings, report bug and sharing flow

### 3.5.3 MapFragment Flow

The *MainActivity* acts as a container for the three main screens (fragments). The navigation will be implemented with tabs, changing the fragment loaded onto the activity depending on which is selected. One of these is the *MapFragment* which will deal with all the functions related to the map. The type of NFC Points can be filtered in the *VisiblePOIDialog*. Besides, if the NFC Points markers are clicked a *MarkerInfoWindow* is shown inside the map with a name and a description. Finally, if the marker window is pressed the *WallActivity* will be loaded showing information about the corresponding NFC Point.

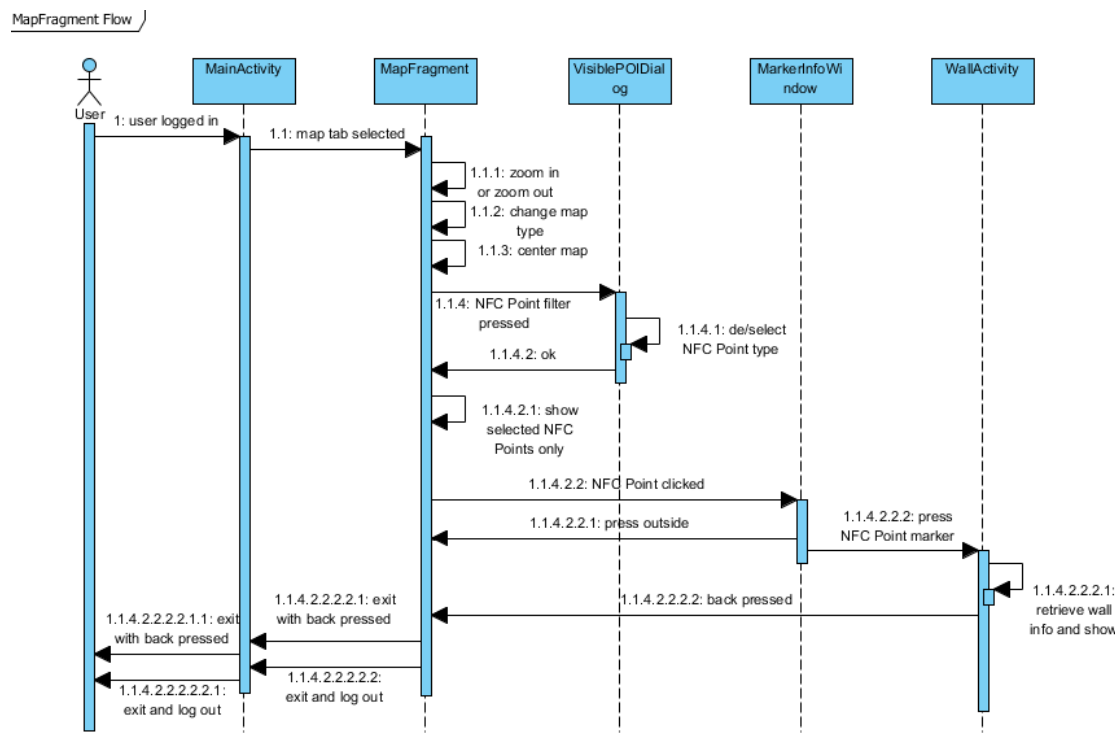


Figure 3.9: MapFragment flow

### 3.5.4 SocialFragment Flow

The *SocialFragment* is also loaded onto the *MainActivity*. In this case it will be in charge of the social functionalities. A selectable list of friends will be displayed in the *SocialFragment*. The *UserCardActivity* will have two duties, showing any friend's information when they are clicked and being the interface for adding friends with Android Beam. If one wish to add a friend but NFC is not active, an intent will be sent to launch network settings and activate it. Last thing to mention is

that, from a friend information in the *UserCardActivity* the *WallActivity* can be accessed for the last NFC Points visited by that friend.

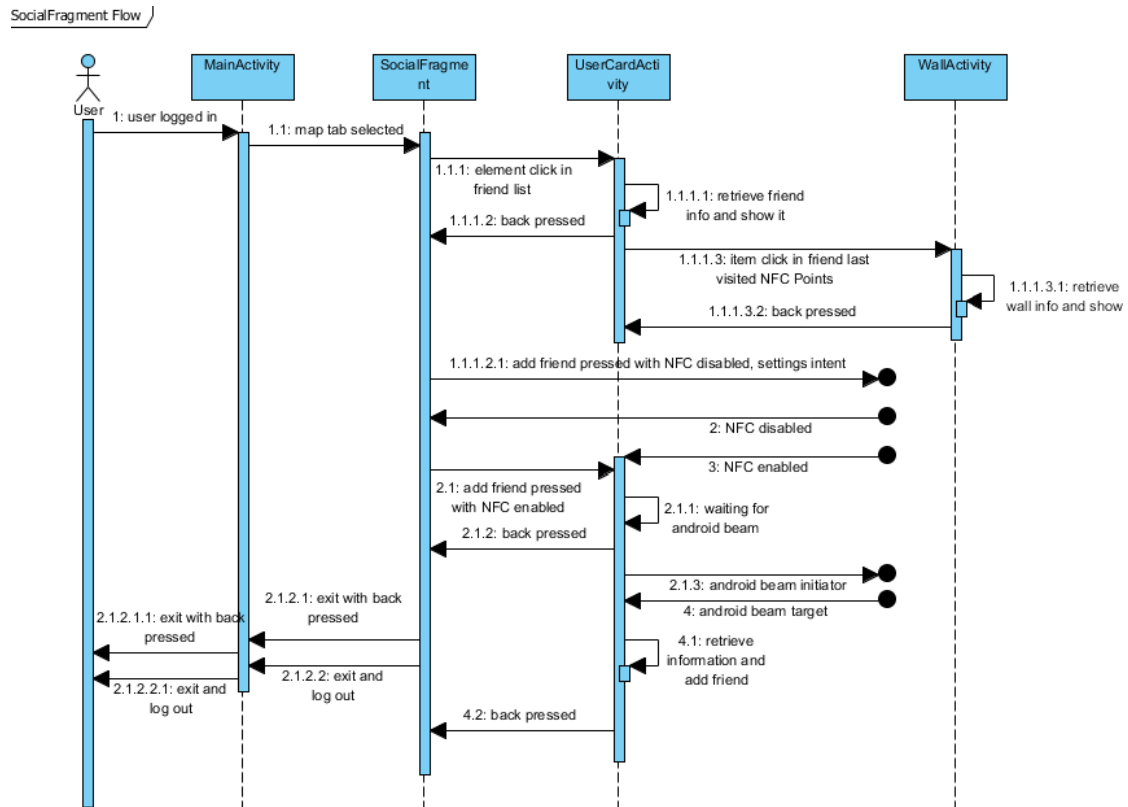


Figure 3.10: SocialFragment flow

### 3.5.5 NFCPointsFragment Flow

The third fragment loaded onto the *MainActivity* is the *NFCPointsFragment*. This one will hold the history of visited and registered NFC Points. The lists will be clickable constituting another access point to the *WallActivity*. New NFC Points will be registered from this fragment too. As in the *SocialFragment* if NFC is not active the network settings will appear. If not, a loading wheel, i.e. *NFCPendingIntentLoadingDialog* will appear waiting for the user to scan a NFC tag. If the detected tag has not been registered before, the *RegisterNFCPointDialog* will appear to fill in some information.

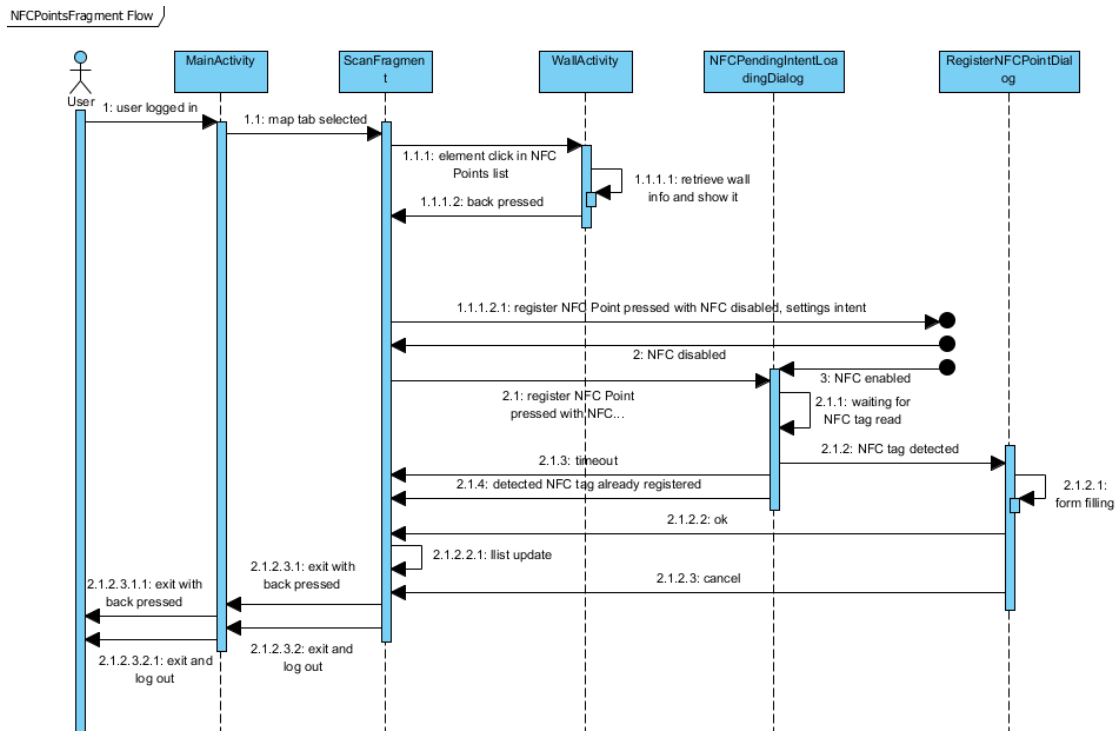


Figure 3.11: NFCPointsFragment flow

### 3.5.6 ServeActivity flow

UniteNFC will parse the data of NFC tags scanned with the mobile phone and then show the content in the *ServeActivity*. No matter what the state of the application is, it will be launched when a tag is scanned. This activity will manage the functions of check-in if the NFC Point is registered but with no interaction of the user. Depending on the content it may have some associated behavior, e.g. a telephone number will yield in a call to that number, so the corresponding intent will be sent. To conclude, the *WallActivity* will be accessible from here too if the NFC Point is registered.

### 3.5.7 WallActivity Flow

The last flow to describe is the one of the *WallActivity*. User interaction in the previous flows may end-up in this activity. From the *WallActivity* comments will be added, through the *AddCommentDialog*, and administration tasks will be selected on the *ContextMenu*.

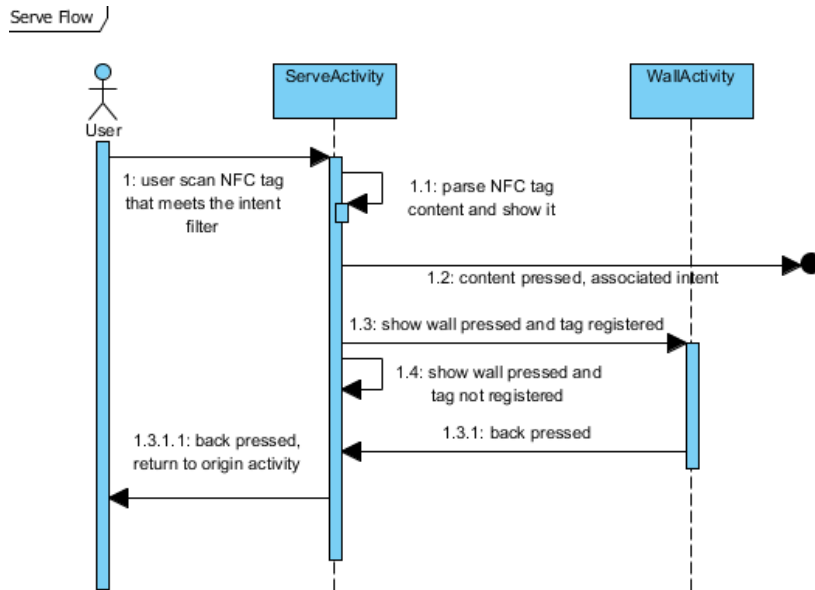


Figure 3.12: ServeActivity flow

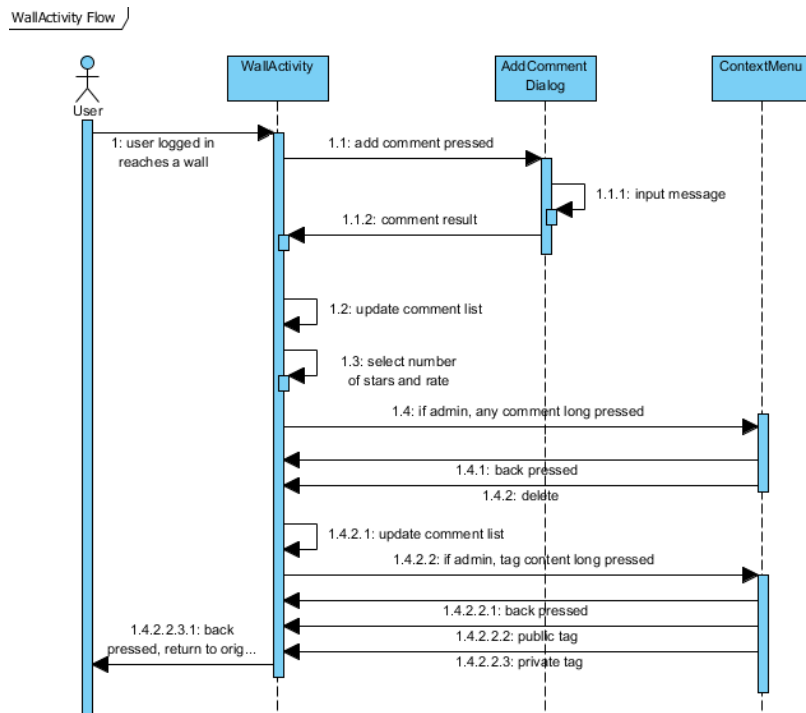


Figure 3.13: WallActivity flow

# 4

## IMPLEMENTATION

This chapter will cover the solutions developed to build-up the designed application. While some subjects will be intentionally skipped, like general Android programming, interesting pices of code or methods will be described.

### 4.1 User Interface

The user interface comprises all the visual elements the application uses to communicate with the user, either text messages, menus, buttons and other widgets or icons.

#### 4.1.1 Look and Feel

Android uses XML files to define the layout of the activities and fragments in which then are inflated (loaded). Editing XML files directly can be tedious unless you are very used to all the widgets and fields and how they affect the layout.

##### 4.1.1.1 Tools

Fortunately, the Android community works hard to deliver powerful solutions to make life easier to developers. The following tools have been used to create and edit the user interface of UniteNFC:

- Android Studio: the main reason to change from Eclipse to Android Studio IDE when it was released was the clean and effective graphical editor included in the later. Quite complex layouts were created just dragging widgets from



the palette and nesting them properly. Besides, it also provides an XML editor in which you can see changes in a display on the fly.

- Android Asset Studio [39]:
  - Launcher icons: adapts your icon to different resolutions of launcher icons.
  - Action bar and tab icons: same as before. Providing it an icon (must be with transparent background) it gives you the set of assets adapted for each resolution.
  - Simple nine-patch generator: imagine you want to place a background image in a layout. You only have this image at a single resolution. If you just use this one for every screen density the image will be shown deformed specially in the borders (the image is enlarged to fit the screen). To avoid this effect, the nine-patch generator generates a set of images that keeps the proportions between screen densities.
  - Action bar style generator: as its name says, it generates all the assets like background, tabs colors, etc; to style your action bar according to the colors you use.
  - Android holo color generator: generates colors so that all the widgets are coherent with your application style.

UniteNFC follows a Holo Light with Dark ActionBar style with color `#0254c2` for the ActionBar and buttons, and `#99cc00` for secondary details, selections and the splash screen background.

#### 4.1.1.2 Styling

When an style is specified in the manifest all the widgets in the application will be styled by default in that way. However, there are no restrictions to create custom styles for the widgets you want and apply them. For the purpose an XML file with a special syntax is added to the *res/drawable* folder and then called by the attribute *android:background* within a widget. Sadly, no graphical editor exists yet for this task.

In UniteNFC a custom style was created for the buttons to make them look better and keep the style used for the ActionBar. The following code defines the *custom\_button* style:

---

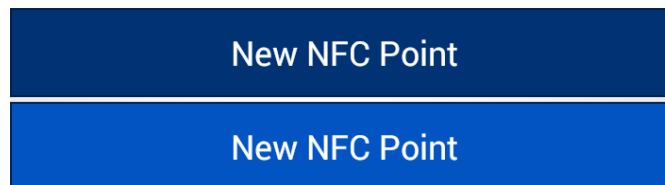
```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true" >
```

```

<shape>
  <solid
    android:color="#0254c2" />
  <stroke
    android:width="1dp"
    android:color="#01214c" />
  <padding
    android:left="10dp"
    android:top="10dp"
    android:right="10dp"
    android:bottom="10dp" />
</shape>
</item>
<item>
  <shape>
    <solid
      android:color="#003274" />
    <stroke
      android:width="1dp"
      android:color="#01214c" />
    <padding
      android:left="10dp"
      android:top="10dp"
      android:right="10dp"
      android:bottom="10dp" />
  </shape>
</item>
</selector>

```

Looking at the code you can see that two identical items are defined, with the only difference of the *android:state\_pressed* attribute which indicates that the first item style will be applied when the widget is pressed. Borders and padding are defined resulting in the button shown in figure 4.1.



**Figure 4.1:** Styled button not pressed and pressed

### 4.1.1.3 Interface Navigation

In this section navigation and action items used in UniteNFC user interface are going to be described as well as how to implement them.

**4.1.1.3.1 ActionBar** The ActionBar is the upper most bar of the application which can be hidden or not, holding buttons for specific actions and tabs. In this case an ActionBar for tab navigation was chosen. The following code shows how to set it.

---

```
ActionBar actionBar = getActionBar();
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

---

Buttons appearing in the ActionBar are just menu items that are marked to be placed there if there is enough room as it is going to be explained in the following section.

**4.1.1.3.2 Menu** Applications normally have menus with many options to choose. The menu in UniteNFC will give access to less used actions or screens. Android provides an XML syntax to define menus in a simple way. The different items appearing on the menu are specified by an ID, a title and, optionally, by the attribute *android:showAsAction* it can be placed in the ActionBar. A piece of code of UniteNFC's menu make evident what have been explained:

---

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/menu_share"
          android:actionProviderClass="android.widget.ShareActionProvider"
          android:title="@string/action_share"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/settings"
          android:title="@string/action_settings"
          android:showAsAction="never"/>
</menu>
```

---

Menus XML files are placed under *res/menu* and then inflated in the Activity during the *onCreateOptionsMenu(Menu menu)* method. The listener to menu items is then implemented in *onOptionsItemSelected(MenuItem item)* which passes you the item that has been selected by the user.

**4.1.1.3.3 Tabs** To navigate between fragments, tabs will be added to the ActionBar. For each tab, a tab listener associated to a fragment will be created. This listener will perform the transaction of removing one fragment and loading

the following. The next snippet shows how tab navigation is loaded during *onCreate(Bundle savedInstanceState)*.

---

```

TabListener maptablistener = new CustomTabListener(new
    CustomMapFragment());
TabListener scantablistener = new CustomTabListener(new ScanFragment());
TabListener socialtablistener = new CustomTabListener(new
    SocialFragment());
Tab map_tab =
    actionBar.newTab().setText(getString(R.string.map_tab)).setTabListener(maptablistener);
Tab scan_tab =
    actionBar.newTab().setText(getString(R.string.nfc_points_tab)).setTabListener(scantablistener);
Tab social_tab =
    actionBar.newTab().setText(getString(R.string.social_tab)).setTabListener(socialtablistener);
actionBar.addTab(map_tab);
actionBar.addTab(scan_tab);
actionBar.addTab(social_tab);

```

---

As was said, one tab listener manages one fragment and is associated to the click event of one tab. The tab listener is implemented in the following way:

---

```

public class CustomTabListener implements TabListener{
    private Fragment fragment;
    private boolean isActive;

    public CustomTabListener(Fragment fragment){
        this.fragment = fragment;
    }

    @Override
    public void onTabReselected(Tab tab, FragmentTransaction ft) {
    }

    @Override
    public void onTabSelected(Tab tab, FragmentTransaction ft) {
        isActive = true;
        ft.add(R.id.fragment_holder,fragment, null);
    }

    @Override
    public void onTabUnselected(Tab tab, FragmentTransaction ft) {
        isActive = false;
        ft.remove(fragment);
    }
}

```

---

```

    public boolean isActive() {
        return isActive;
    }
}

```

---

When the tab is selected the overridden method *onTabSelected(Tab tab, FragmentTransaction ft)* is called adding the fragment to the *fragment\_holder* (which is a *LinearLayout* of the Activity layout) and when another tab is selected the *onTabUnselected(Tab tab, FragmentTransaction ft)* method removes the fragment.

## 4.1.2 ListViews Creation

ListViews are holders for a group of scrollable items. Such holders are loaded by an Adapter which needs to know from the single item layout and content. Note that the creation of Adapters is recommended by the Android community and is considered a good design pattern [41].

### 4.1.2.1 Single Element Layout

The layout is defined as any other. For example, an *ImageView* with a profile picture next to a *TextView* with the name of a user. This means that every item in the *ListView* will be the same.

### 4.1.2.2 Single Element Class

Then a Java class has to be defined to model the object with the content that will fill up the layout. Following the example, the class will have two attributes: an *String* to write on the *TextView* and a *BitMap* to load into the *ImageView*. Setters and getters need to be implemented too. Additionally, the abstract class *Comparable* may be interesting to implement ordering on your lists, for example, alphabetically.

### 4.1.2.3 ListViewAdapter

Finally the Adapter maps the content of an object to the layout and inflates it into the *ListView*:

---

```

public class CustomListViewAdapter extends ArrayAdapter<RowItem> {

    private Context context;

    public CustomListViewAdapter(Context context, int resourceId,

```

```

        List<RowItem> items) {
    super(context, resourceId, items);
    this.context = context;
}

/*private view holder class*/
private class ViewHolder {
    ImageView imageView;
    TextView txtTitle;
}

public View getView(int position, View convertView, ViewGroup
parent) {
    ViewHolder holder;
    RowItem rowItem = getItem(position);
    LayoutInflater mInflater = (LayoutInflater) context
        .getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
    if (convertView == null) {
        convertView = mInflater.inflate(R.layout.list, null);
        holder = new ViewHolder();
        holder.txtTitle = (TextView)
            convertView.findViewById(R.id.text1);
        holder.imageView = (ImageView)
            convertView.findViewById(R.id.icon);
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }
    holder.txtTitle.setText(rowItem.getTitle());
    holder.imageView.setImageBitmap(rowItem.getImageId());
    return convertView;
}
}

```

---

It extends `ArrayAdapter` of type your class to adapt (in this case *RowItem*). To conclude the array of items is passed to the adapter with its layout and the `ListView` adapter is set:

---

```

CustomListViewAdapter adapter1 = new
    CustomListViewAdapter(getApplicationContext(),R.layout.rowitem, r1);
list.setAdapter(adapter1);

```

---

Being *r1* a list of *RowItem* and *list* a `ListView` object.

### 4.1.3 Map

One of the use cases was to view the NFC Points on a map with different colors for different types. The map is going to be held in a Fragment which will be loaded pressing the first tab of the ActionBar in the main Activity. This is possible thanks to the MapFragment of the Google Maps Android API v2 which includes typical functionalities like map exploration by gestures. To add custom functionalities to the ones provided, the MapFragment was extended and new functions to control markers, map type, camera and click events on markers were implemented.

---

```
public class CustomMapFragment extends MapFragment implements
    OnInfoWindowClickListener{
    public CustomMapFragment() {
        super();           //when super() is called the map is initialized
    }

    @Override
    public void onInfoWindowClick(Marker marker) {
        //
    }
}
```

---

The *CustomMapFragment* created is kept as an attribute in the main Activity class being able to alter what the map shows calling methods defined in it.

The map will not be shown unless proper configuration is carried out. You need to register your application in the Google API Console, add Google Maps Android API v2 service, generate an API key from the package name of the Android application and the SHA1 hash of the certificate used to sign it. At last, you need to add the resulting API key to the manifest.

#### 4.1.3.1 Camera

The view of the map can be changed with a CameraPosition object:

---

```
CameraPosition camPos = new CameraPosition.Builder()
    .target(new LatLng(40.317,-3.782)) //geolocation
    .zoom(19)
    .bearing(45)
    .tilt(70)
    .build();
CameraUpdate camUpd = CameraUpdateFactory.newCameraPosition(camPos);
this.getMap().animateCamera(camUpd);
```

```
}

```

---

Besides, the transition can be animated, as the code shows, to the specified point on the map with the desired zoom, camera bearing and tilt.

#### 4.1.3.2 Markers

Markers are going to show the position of NFC Points. For that task the following method is defined:

---

```
public void POIMarkers() throws NullPointerException{
    if(poi_list !=null){
        getMap().clear();
        Marker mrk;
        for(POI poi:poi_list){
            int poiType = poi.getCategories().get(0).getId();
            LatLng POIloc = new LatLng(poi.getLatitude(),
                poi.getLongitude());
            BitmapDescriptor icon;
            String title = poi.getName().substring(16);
            String description = poi.getDescription();
            boolean visibility;
            switch(poiType){
                case POICategories.TURISM: icon= nfc_orange;
                    visibility = poi_vis[1];
                    break;
                case POICategories.LEISURE: icon=
                    nfc_violet;
                    visibility = poi_vis[2];
                    break;
                case POICategories.EVENT: icon= nfc_green;
                    visibility = poi_vis[3];
                    break;
                default: icon= nfc_blue;
                    visibility = poi_vis[0];
                    break;
            }
            mrk = this.getMap().addMarker(
                new MarkerOptions().icon(icon).position(POIloc)
                    .title(title).snippet(description));
            mrk.setVisible(visibility);
            mrklist.add(mrk);
        }
    }
}
```



```

        }
    }
}

```

---

Every-time the NFC Points list is updated this method is executed. The NFC Points are in the *poi\_list* variable as POI objects which is a class provided by the topooos API and will be explained later in section 4.4.1. If the list is not null (meaning that the update is correct) all the markers are cleared out and then the list is iterated getting title, description and type. The title is gotten cropping out the first 16 characters of the name field of the POI because those are where the NFC tag ID is stored (more on this in section 4.4.2). Depending on which NFC Point is, a different icon is loaded and will it be visible whether the user has selected it or not from a menu to filter them.

#### 4.1.4 Dialogs and Toasts

The last elements to comment on regarding user interface are Dialogs and Toasts. Both are pop-ups used to give information to the user but, while Dialogs allow more complex layouts and user interaction, Toasts only show a message during a small period of time. Therefore, Toasts where used to give feedback to the user about errors or other information messages.

Toasts can be created and shown just writing one line of code in the following way:

---

```

Toast.makeText(getApplicationContext(), "Hello World!",
    Toast.LENGTH_SHORT).show();

```

---

Care must be taken when calling a function to get the context, since is not the same to call `getApplicationContext()` from an Activity than from a Dialog in which will return null, for example.

Dialogs can hold buttons and a layout with widgets and associated listeners. All this is configured during the *onCreateDialog(Bundle savedInstanceState)* method of a class extending `DialogFragment`. In the following example a layout called *comment* is loaded along with a confirmation button an a negation one.

---

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    // Use the Builder class for convenient dialog construction
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    LayoutInflater inflater = getActivity().getLayoutInflater();

```

```

builder.setView(inflater.inflate(R.layout.comment, null));
builder
    .setTitle(getString(R.string.report_title))
    .setPositiveButton(getString(R.string.report_send),
        positiveListener)
    .setNegativeButton(getString(R.string.new_nfc_no),
        negativeListener);
return builder.create();
}

```

---

## 4.2 Location

UniteNFC will show on the map the user position constantly. For that to be possible the user location has to be known. Android SDK provides the tools to implement it in a very simple way.

### 4.2.1 LocationManager and LocationListener Setup

In section 3.4.1.1 was explained that extra hardware need to be specified in the manifest. To access location capabilities, permissions need to be requested on the manifest.

```

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>

```

---

*ACCESS\_FINE\_LOCATION* includes both possible location providers, i.e. GPS and network localization. The alternative would be to use *ACCESS\_COARSE\_LOCATION* instead that only access the network localization information. However, as the name suggest, the alternative results in worse location estimates.

The main Activity will hold a MapFragment and will be in charge of updating its information when changes occur in the user position. LocationManager and LocationListener are the Android objects that manage the reception of new location as requested. The following code shows how these elements are created and configured when the Activity is created:

```

mCustomLocationListener = new RegisterPosition();
mLocationManager = (LocationManager)
    this.getSystemService(Context.LOCATION_SERVICE);
mLocationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
    1000, 0, mCustomLocationListener);

```

```
mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    1000, 0, mCustomLocationListener);
```

---

A `LocationManager` is created and then it asks for location updates from both providers passing the listener that will handle them. The integers passed to the method are time in milliseconds and distance in meters between updates respectively. Hence, both providers will be receiving at most one update per second no matter what the distance difference is. The class *RegisterPosition* is actually implementing the `LocationListener`. Its code is not going to be shown, since it is basically the same as in Android Developers documentation [42]. To summarize, when a new location is obtained the map is updated with that location.

## 4.3 NFC Block

This section contains all the aspects of how NFC works in Android and how it has been implemented into the application. Therefore, it is going to be explained thoroughly trying to be as clear as possible.

### 4.3.1 NFC Setup

As long as the device has an NFC chip integrated in it, it can be accessed in software with the Android SDK. However, some software configuration needs to be carried out to let Android know that is going to use NFC functionalities and when to pass the Intents to `UniteNFC`.

#### 4.3.1.1 Manifest

The first step to integrate NFC into your application is to ask for permissions on the manifest.

---

```
<uses-permission android:name="android.permission.NFC"/>
<uses-feature android:name="android.hardware.nfc"
    android:required="true" />
```

---

The *uses-feature* clause is optional, but recommended. It guarantees that only devices with NFC capabilities can find the application in Google Play, avoiding in this way unpleasant user experience for those trying to use the application without knowing the requirements.

### 4.3.1.2 NFC IntentFilters

The Android Tag-Dispatch System is continuously looking for NFC events when the device screen is on, identifying them and building the Intents with the event information (tag content, format, etc). The Intent created is served to the Activity that filters for it. Android IntentFilter System is designed so that the Intent is sent to the most suitable Activity without asking the user what to do with it.

There are three types of Intents defined for NFC events:

- *ACTION\_NDEF\_DISCOVERED*: Intent created when the payload has NDEF format. Remind that this is the format specified by the NFC Forum 2.4.2.
- *ACTION\_TECH\_DISCOVERED*: Intent created when not NDEF payload is found but the NFC technology is known.
- *ACTION\_TAG\_DISCOVERED*: Intent created when neither the payload nor the technology are identified.

These Intents are ordered from highest to lowest priority, i.e. Android will try to create the first type and if not possible, continue with the next option. This is illustrated in figure 4.2 which has been extracted from Android Developers.

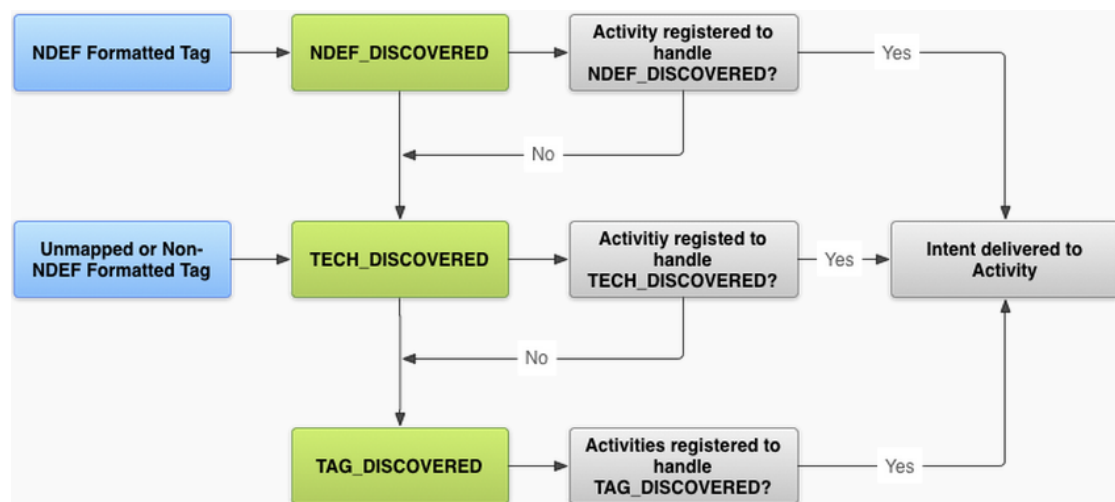


Figure 4.2: Tag Dispatch System [43]

So that the Intent is delivered to the correct Activity, it needs to have defined an IntentFilter in the manifest. The following filters have been defined for UniteNFC:

```

<intent-filter>
  <action android:name="android.nfc.action.NDEF_DISCOVERED"/>

```

```

    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
    <data android:mimeType="text/x-vCard"/>
</intent-filter>
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="sms"/>
    <data android:scheme="tel"/>
    <data android:scheme="geo"/>
    <data android:scheme="mailto"/>
    <data android:scheme="wifi"/>
    <data android:scheme=""/>
    <data android:scheme="http"/>
    <data android:scheme="https"/>
    <data android:scheme="urn:nfc:ext:android.com:pkg"/>
    <data android:scheme="file"/>
</intent-filter>
<intent-filter>
    <action android:name="android.nfc.action.TECH_DISCOVERED"/>
</intent-filter>
<meta-data android:name="android.nfc.action.TECH_DISCOVERED"
    android:resource="@xml/nfxc_tech_filter" />

```

Note the rich variety of IntentFilters that can be defined. As UniteNFC would like to handle all the NFC tags read by the device to record the visits to NFC Points, most cases have been tried to be covered. The *NDEF\_DISCOVERED* IntentFilter is defined in two parts to indicate that data with the MIME type *text/plain* or *text/x-vCard* and data scheme as specified has to be served to this Activity. Intuition may suggest to put the *android:mimeType* and *android:scheme* inside the same IntentFilter, but in that case any of them would work. For the *TECH\_DISCOVERED* IntentFilter a list of technologies has to be specified in an XML file stored in the *res/xml* folder. Whenever one technology present in the file is found (without NDEF format) the filter will match the Intent and the Activity will receive it. No filter is defined for the third kind of Intent since there is no way to know the data format and show something meaningful to the user.

The list of technologies filtered for UniteNFC is:

```

<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.IsoDep</tech>
        <tech>android.nfc.tech.NfcA</tech>
    
```

```

    <tech>android.nfc.tech.NfcB</tech>
    <tech>android.nfc.tech.NfcF</tech>
    <tech>android.nfc.tech.NfcV</tech>
    <tech>android.nfc.tech.Ndef</tech>
    <tech>android.nfc.tech.NdefFormatable</tech>
    <tech>android.nfc.tech.MifareClassic</tech>
    <tech>android.nfc.tech.MifareUltralight</tech>
</tech-list>
</resources>

```

---

Basically, this covers all NFC Forum compliant tags, i.e. type 1, 2, 3 and 4; plus the MifareClassic tag which is widely used in many RFID systems.

### 4.3.2 Interacting with NFC Points: Tag Reading

Reader-Writer NFC mode is going to be the basis for the application, since it will be used to interact with NFC Points. More precisely, only reading will be implemented working on the assumption that tags have already been written. There already exist many writer tools and there is no added value in having one integrated in UniteNFC. In this section NFC tag reading is going to be explained. In other words, how to get information from the NFC Points.

#### 4.3.2.1 Catching the Intent

When an Intent matches any of the filters described, UniteNFC Activity will receive it during its *onCreate* method calling *getIntent*. If two Activities have defined the same filter Android will ask the user to choose between those. The Intent will hold all the information read through the NFC radio interface.

#### 4.3.2.2 Foreground Dispatch

Another way to read a tag is using a PendingIntent. This type of Intent works, somehow, the other way around. Instead of being an action what creates the Intent, an Intent is created and the action is awaited to occur to fill up the Intent. The wait is performed in the foreground and all the IntentFilters are disabled. Therefore, is a useful option when the set-up IntentFilters need to be overridden for a short period of time.

The NFC adapter is gotten, a PendingIntent is created for the current Activity with the associated filter and the foreground dispatch is enabled passing both the PendingIntent and the IntentFilter.

---

```

NfcAdapter mAdapter =
    NfcAdapter.getDefaultAdapter(this.getApplicationContext());
PendingIntent pendingIntent =
    PendingIntent.getActivity(this.getActivity(), 0, new
        Intent(this.getActivity(),
            getActivity().getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP),
        0);
IntentFilter [] intentfilter = new IntentFilter[] {new
    IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED)};
mAdapter.enableForegroundDispatch(this.getActivity(), pendingIntent,
    intentfilter, null);

```

---

A comment should be made on the `IntentFilter`, as the ones defined in the manifest are not available while the foreground dispatch is enabled, the lowest priority filter has been set to comprise all the possible options. The foreground Activity will receive the Intent on the `onNewIntent` callback method.

This type of Intent will be useful to register new NFC Points, the user will press a button indicating he wants to do it, the button's listener will launch the `PendingIntent` and wait for 5 seconds to receive the Intent of the NFC Point the user wants to register. The wait limit is set to not block the application flow and should be enough to put the device over the tag. How the NFC Point is registered will be explained in more detail in section in section 4.4.2.

### 4.3.2.3 Parse Tag Content

Once the Intent has been received the tag data has to be parsed. The most important part in this process is to identify data format. If the Intent is of the kind `NDEF_DISCOVERED`, it implies the data format will be NDEF for which Android provides automated method to be parsed. In any other case, the data received will be treated as RAW data, i.e. without any format.

---

```

Intent intent = getIntent();
String action = intent.getAction();
if (NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)
    || NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
    Parcelable[] rawMsgs =
        intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
    NdefMessage[] msgs;
    if (rawMsgs != null) { // NDEF
        msgs = new NdefMessage[rawMsgs.length];
        for (int i = 0; i < rawMsgs.length; i++) {

```

```

        msgs[i] = (NdefMessage) rawMsgs[i];
    }
} else { // Unknown tag type
    byte[] empty = new byte[0];
    byte[] id = intent.getByteArrayExtra(NfcAdapter.EXTRA_ID);
    Parcelable tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
    byte[] payload = dumpTagData(tag).getBytes();
    NdefRecord record = new NdefRecord(NdefRecord.TNF_UNKNOWN,
        empty, id, payload);
    NdefMessage msg = new NdefMessage(new NdefRecord[] { record });
    msgs = new NdefMessage[] { msg };
}
}
}

```

---

The Activity may be created by more of one Intent type, in that case is advisable to check the action received. In the code above it checks whether the Intent has been created by an NFC event. In that case, the message has to be parsed. The method *getParcelableArrayExtra(NfcAdapter.EXTRA\_NDEF\_MESSAGES)* will extract the NDEF messages if any. When that is the case an array of NDEF messages is created just casting the obtained ones. Otherwise, all the bytes will be extracted without any distinction and an NDEF message will be created with that raw bytes as the payload of a single record. Now, both types can be treated equally.

Next step is to determine the record type that the messages hold. Four types of records are expected:

- URI record: record with TNF equals TNF\_WELL\_KNOWN and record type RTD\_URI or TNF equal TNF\_ABSOLUTE\_URI.
- Text record: record with TNF equals TNF\_WELL\_KNOWN and record type RTD\_TEXT.
- Smart Poster record: record with TNF equals TNF\_WELL\_KNOWN and record type RTD\_SMART\_POSTER.
- Unknown/Raw record: any not belonging to the previous groups.

Once the record type is known, the process to parse each of them is slightly different.

For text records nothing else needs to be done, the payload will contain a string with the message.



URI records are more complicated to parse, since there are many possibilities. If the TNF is `TNF_ABSOLUTE_URI`, all the payload will be a URI that can be parsed with the Android `Uri` class. In the other case, the URI type has to be identified checking the first byte of the payload before parsing, again, with the `Uri` class.

---

```
public static UriRecord parse(NdefRecord record) {
    short tnf = record.getTnf();
    if (tnf == NdefRecord.TNF_WELL_KNOWN) {
        return parseWellKnown(record);
    } else if (tnf == NdefRecord.TNF_ABSOLUTE_URI) {
        return parseAbsolute(record);
    }
    throw new IllegalArgumentException("Unknown TNF " + tnf);
}

private static UriRecord parseAbsolute(NdefRecord record) {
    byte[] payload = record.getPayload();
    Uri uri = Uri.parse(new String(payload, Charset.forName("UTF-8")));
    return new UriRecord(uri);
}

private static UriRecord parseWellKnown(NdefRecord record) {
    Preconditions.checkArgument(Arrays.equals(record.getType(),
        NdefRecord.RTD_URI));
    prefix = URI_PREFIX_MAP.get(payload[0]);
    byte[] fullUri =
        Bytes.concat(prefix.getBytes(Charset.forName("UTF-8")),
            Arrays.copyOfRange(payload, 1, payload.length));
    Uri uri = Uri.parse(new String(fullUri, Charset.forName("UTF-8")));
    return new UriRecord(uri);
}
```

---

To fully understand the example it is necessary to know that *UriRecord* is an object representing a parsed URI record that is what is being built. Note that when parsing in the `TNF_WELL_KNOWN` case, the URI type is obtained from the `URI_PREFIX_MAP` which is a dictionary with the bytes being the keys and the URI types being the values.

---

```
private static final BiMap<Byte, String> URI_PREFIX_MAP =
    ImmutableBiMap.<Byte, String>builder()
        .put((byte) 0x00, "")
        .put((byte) 0x01, "http://www.")
        .put((byte) 0x02, "https://www.")
        .put((byte) 0x03, "http://")
```

```
.put((byte) 0x04, "https://")
//etc
```

---

So when the first byte is *0x01* the parser knows the URI is a URL starting with *http://www..*

There is no much to explain left of how to parse a Smart Poster, as it is just the addition of a text record and an URI record. After the records are identified parsing can be done as explained for each type.

The last type of record is not parsed indeed. What is done is to extract the bytes of the payload as a string and hopes the user can understand it.

Finally the content is shown in the screen, inside a `ListView`, along with the date. If the message has an action associated, e.g. send an email, the content itself is a link which launches an `Intent` to call the `Activity` that performs that action.

### 4.3.3 Adding Friends: Android Beam

The second functionality for which NFC is used is to quickly and seamlessly add new friends. For the purpose NFC Peer-to-Peer mode will be used. Android P2P capabilities are restricted to Android Beam, that is a software implementing the SNEP protocol with an extra software layer. Even-though this system can ease a lot P2P programming, it also limits many aspects that makes P2P an interesting mode. Such limitations include:

- Unidirectional communication: whilst P2P is defined as bidirectional.
- Application level: being impossible to define your own protocol.
- Requires user interaction: making it slower.

Android Beam works like follows, from a user point of view. When NFC and Android Beam are activated application content can be transferred between devices. Two devices are put back to back so that both NFC antennas face each other (the antenna is placed in different positions for different devices, but it tends to be in the center of the back part), and the Android Beam screen will appear. The application screen is shrunk and floating with a message that says *Touch to transfer*. After the user touches the screen in any point it starts the communication as the initiator and, while holding the devices still together, an NDEF message is sent from initiator to target. The message contains the foreground `Activity` and, if such `Activity` has implemented the `onNdefPushMessageCallback` it will add

content to the message. For example, Youtube transfers not only the link that is being watched but the state of the reproduction like volume and the reproduction time. The target receives an Intent that sends to the same application and in the case that application does not exist in the target device it looks for it in Google Play and shows it to the user.

Now, some insights about how to implement it in your application, more than just transferring screens, will be given. One mechanism has already been commented, the *onNdefPushMessageCallback*. This callback allows you to generate a custom NDEF message to send over. You can generate your own MIME type to uniquely identify you application and put whatever you want inside the payload field of the NDEF message.

---

```
public final String MIME_TYPE = "application/es.quantum.unite NFC";

@Override
public NdefMessage createNdefMessage(NfcEvent arg0) {
    return createMessage();
}
private NdefMessage createMessage(){
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
    byte[] payload = (prefs.getString("session",
        "")+";"+prefs.getString("username",
        "")+";"+prefs.getString("imageuri", "dummy_4")).getBytes();
    byte[] mimeBytes = MIME_TYPE.getBytes(Charset.forName("US-ASCII"));
    NdefRecord cardRecord = new NdefRecord(NdefRecord.TNF_MIME_MEDIA,
        mimeBytes, new byte[0], payload);
    return new NdefMessage(new NdefRecord[] {cardRecord});
}

```

---

This example is how a user information message is created to send it with Android Beam in UniteNFC. The MIME type is specified to be on the NDEF record header and the payload just contains the necessary information to identify a user. The second callback that the initiator can manage is the *onNdefPushComplete*. It is used to define what is done after the transaction has finished.

---

```
@Override
public void onNdefPushComplete(NfcEvent event) {
    //method body
}

```

---

The code inside the method has been omitted since it does not provide any relevant information on Android Beam.

Looking at the target side, everything works as if an NFC tag is read. So the first task to do is define the appropriate IntentFilter.

---

```
<intent-filter>
  <action android:name="android.nfc.action.NDEF_DISCOVERED" />
  <data android:mimeType="application/es.quantum.unitenfc" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

---

Note that the MIME type is defined to match the one the initiator has created. This filter guarantees that no other Activity reacts to the message sent by your application. Once the IntentFilter has been defined, remember that the Activity received the Intent on the callback *onNewIntent* where it can handle it and decide what to do.

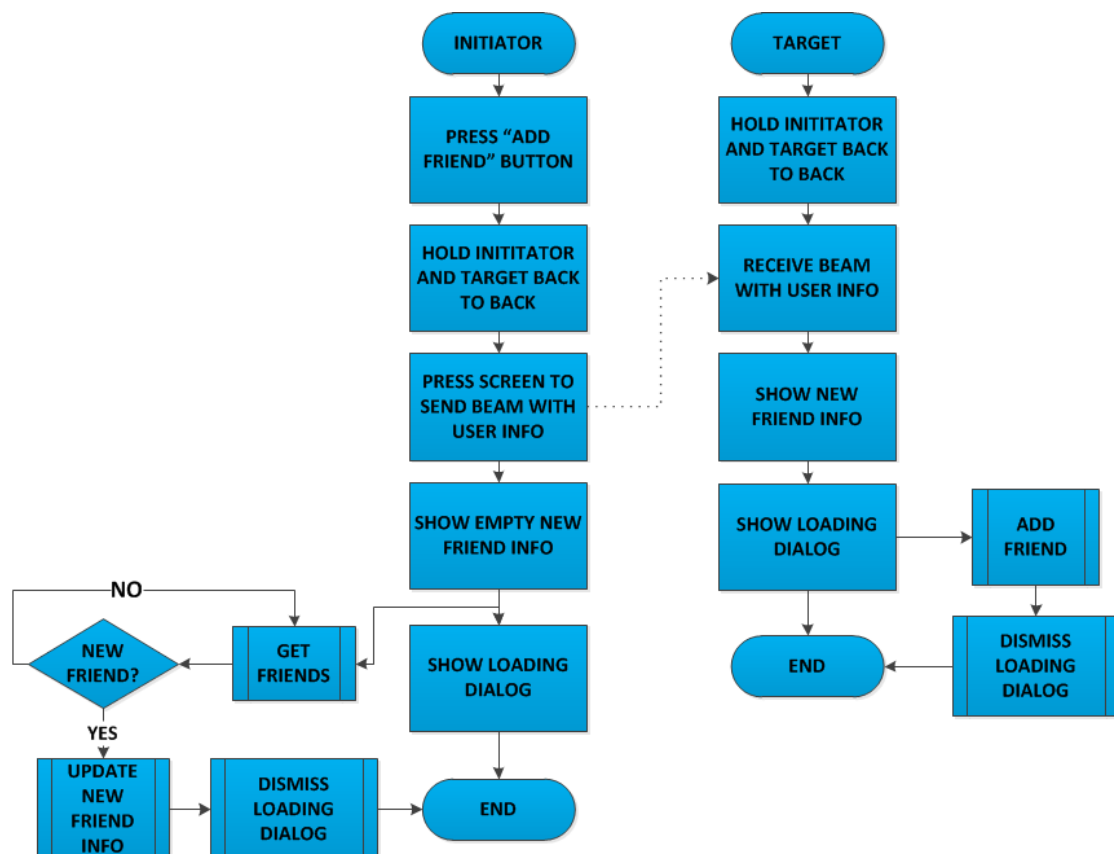


Figure 4.3: Beam flow

As was said at the beginning of the section, Android Beam has been used as the communication channel to add new friends. Due to the restrictions already commented, the initiator sends its user information and the target will carry out the communication with the back-end to create the relationship. The initial idea was to show instantly the data of the new friend on the screen, right after the Android Beam transmission takes part. There is no problem for the target since it receives the information directly through NFC. However, the initiator needs to poll the server until the target has finished his request. To make the interactions taking part clearer the flow diagram of figure 4.3 has been created.

## 4.4 NFC Points

NFC Points are the key element around which UniteNFC is created. An NFC Point is a point of interest in which an NFC operation can be performed. More precisely, it is a location in which an NFC tag is available to be read. UniteNFC needs to perform operations with them to offer a solution to the analyzed scenarios in section 3.2. Such operation comprises NFC Point registration, NFC Point check-in and NFC Point positioning.

### 4.4.1 POI topoos

Most operations to be implemented are related to the management of points of interest (POI), for which topoos is a powerful tool. With topoos you can register a POI with a series of attributes such as name, description, POI type and location; and then get all the POIs filtering by distance to represent them in a map, for example. So, to sum up, an NFC tag will be read and its information plus its location will be stored in the topoos server as a POI (that in UniteNFC is called NFC Point).

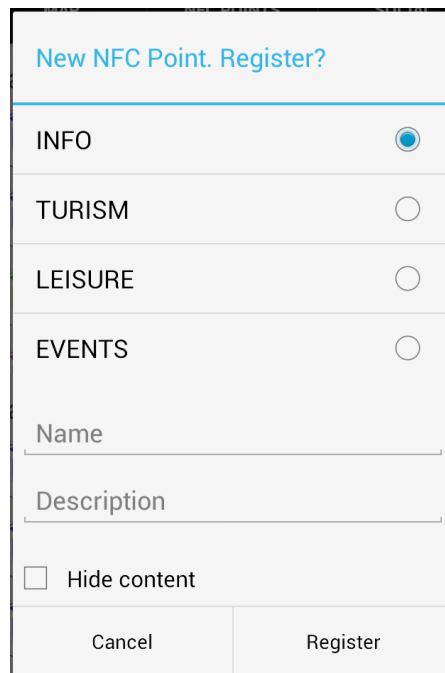
### 4.4.2 Registering New NFC Point

The process of registering a new NFC Point will be as follows: an NFC tag will be scanned by the user, if the tag is not registered yet a dialog will be prompted asking the user to complete the information about the NFC Point and finally the NFC Point is registered both in topoos as a POI and in UniteNFC's back-end.

Explained in more detail step by step, first of all the user has to communicate the application that he wants to register an NFC Point. This will be done using a button in one of the screens. Then a PendingIntent will be launched waiting for

the user to scan the tag, exactly as was explained in section 4.3.2.2. As soon as the Intent is received the tag ID is extracted to check whether it has been registered before or not. An NFC Point can be registered only once and to check upon that condition the unique tag ID is used. All the registered tags' ID are stored along with the name in topoos making it possible.

When it has been checked out that the NFC Point is not registered yet a dialog will ask the user to input some information as in figure 4.4. There are 4 types of NFC Points, that can be mapped with each of the scenarios previously proposed but are not tied to them. Name and description does not need further explanation. The check-box gives the user the option of hiding or showing the tag content in the NFC Point wall. Note that the content will always be available physically into the tag, but may not be of interest to post it on-line.



The image shows a mobile application dialog box titled "New NFC Point. Register?". It contains a list of four categories: INFO, TURISM, LEISURE, and EVENTS. The "INFO" category is selected, indicated by a blue radio button. Below the list are two text input fields labeled "Name" and "Description". At the bottom left, there is a checkbox labeled "Hide content" which is currently unchecked. At the bottom right, there are two buttons: "Cancel" and "Register".

**Figure 4.4:** Registration Dialog

At last, after the user presses *Register* in the dialog, the NFC Point is stored in the cloud. Since topoos does not provide all the functionalities that were required to implement a wall system with user interaction, the NFC Point is registered both on topoos and on UniteNFC's back-end. The following code is the corresponding to register a POI in topoos:

---

```
Integer[] categories = new Integer[2];
```

```

categories[0] = poiType;
categories[1] = POICategories.NFC;
try {
POI newPoi = topoos.POI.Operations.Add(ctx, tagid+name,
    location.getLatitude(), location.getLongitude(), categories,
    (double)0, (double)0, (double)0, description, null, null, null,
    null, null, null, null);
} catch (TopoosException e) {
    e.printStackTrace();
}

```

---

This function must be carried out in a different thread, as they imply network connections. Observe how simple topoos SDK makes it. The variables *poiType*, *name* and *description* are directly obtained from the dialog, while *location* is the current user position and *tagid* is obtained when the tag is scanned. As there is no other field to store the *tagid* and it is of fixed length (16 bytes) it is concatenated with the name string. The application knows this convention and tag id and name will be split when necessary.

Registration of the NFC Point in the UniteNFC's back-end will be explained later in section 4.5.4 in which rules on how to communicate with the REST server will be addressed.

### 4.4.3 Checking in NFC Point

When a user visits an NFC Point and reads a tag, UniteNFC will record that event. Of course, the NFC Point must be registered before. Otherwise, the check-in will not be performed when the tag content is served. This allows the user to store a history and compare with other friend history, discovering perhaps an interesting NFC Point that was not aware of. So once the NFC Point content has been properly parsed and shown, the check-in operations take part. Again as for registration, the check-in will be carried out with topoos and with the own developed server. Principally, one big limitation of topoos API is that only stores the last check-in, so that limitation has been overcome with a tailor-made function in the back-end.

---

```

User me = topoos.Users.Operations.Get(getApplicationContext(), "me");
Position current_pos =
    topoos.Positions.Operations.GetLastUser(getApplicationContext(),
    me.getId());
Location location = new
    topoos.Objects.Location(current_pos.getLatitude(), current_pos.getLongitude());

```

```

Integer[] categories = new Integer[1];
categories[0] = POICategories.NFC;
List<POI> poi_list = topoos.POI.Operations.GetNear(ctx,
    location.getLatitude(), location.getLongitude(), 10, categories);
for(POI poi:poi_list){
    if(poi.getName().substring(0, 16).compareTo(tagid)==0){
        topoos.Checkin.Operations.Add(ctx, poi.getId(), new
            Date());
    }
}

```

---

This snippet looks for NFC Points around the user with the function provided in `topoos.POI.Operations.GetNear` and from the obtained NFC Point list, iterates it searching for a match comparing tags ID. If the scanned NFC Point tag ID coincides with one of the list, it means that the NFC Point is registered in the system and, then, a check-in is done.

## 4.5 RESTFUL Web Development

UniteNFC needs of many functions that are not provided by the software libraries that are being used, as was explained in section 3.4.2. To carry out the task of developing a REST compliant web server, the Django Framework is going to be used.

### 4.5.1 Implementing MVC with Django

Django framework structure follows the MVC pattern itself. It isolates each part of the pattern (models, controllers, views), even in a different file, to ease the programmer with independent developments. Django uses a nomenclature that can be confusing. While models go into the `models.py` file, controllers go into the `views.py` file. Views are directly served from controllers but stored into the `templates` folder in which, for example, HTML files that are the actual views will be stored.

Django fosters code reusing, separating functionalities in small application that should be portable between projects. In the case of UniteNFC, a single application has been created. This application called *objects* will manage all the readings from and writing to the database, in which all the application information will be stored. This operations will be accessed through HTTP requests, more precisely GET requests for reading and POST for writing. Furthermore, each resource must have an unique URL to accomplish REST. The responses to the requests commented



will have JSON format containing a representation of the requested resource.

It is not the aim of this section to serve as tutorial of how to program on Django, all the documentation necessary is available in the Django website [3]. What is going to be explained includes general concepts and specific implementation of the back-end developed for UniteNFC.

#### 4.5.1.1 Model

In the model, the items that need to be stored in the database are defined as simple classes, there is no need to make any operation directly on the database, such as creating tables or making queries, thanks to the object-relational mapping abstraction layer. ORM as it names suggests, translates manipulation of objects into transactions in a relational database. Django offers a quite simple syntax to make any kind of query, with the possibility of filtering for multiple criteria simultaneously. Moreover, this practice protect against SQL injections automatically, which is one of the most common attacks.

Django works with most SQL flavors like MySQL, SQLite and PostgreSQL. The latest has been used for the reasons addressed in the analysis chapter. To ensure that the database is properly synchronized with Heroku some configurations needs to be written in the *settings.py* that is automatically generated when the Django project is started.

The objects defined in the model are the following:

- **UserInfo:** it contains all the information related to a user, i.e. name, profile picture URI, ID and a link to a list of friends.
- **ContactRelationship:** this object represents a friendship relationship, it links one user with another.
- **NFCPoint:** represents an NFC point, either registered or visited. Its attributes are name, ID, date, address, a boolean to know if it is a registration record or a check-in record and a link to the user that has perform the operation.
- **Wall:** an screen in which all the information of an NFC Point will be shown, containing comments and user ratings. The attributes for this objects are ID, NFC Point type, title, description, last seen date, NFC Point (tag) content and if the content is private.

- Entry: a comment entry on the wall. It contains a message, an author name and picture and a link to the wall it belongs.
- Rating: a mark that a user gives to a wall or NFC Point. An integer number from 0 to 5, a link to the rated wall and a link to the user that rates form the model.
- Fblink: maps a user ID with its Facebook ID, therefore it just contains two char fields one with each ID.

A Python class is similar to the ones in other object oriented programming languages, a series of attributes can be defined (which can be initialized in the constructor) and then methods performed. So that the class is included into the database through the ORM, it needs to inherits from the Model class of Django. In that case all the attributes will be stored into the database. Objects relationships are created with two fields: ForeignKey for one to N and ManyToManyField for N to N relations. In the second case it is advisable to create an intermediate object manually, although Django will create it for you if necessary.

The following example corresponds to the *UserInfo* object which has the fields described recently. The link to a list of friends is implemented by the *friends* attribute, which is a ManyToManyField that makes the relation through the *ContactRelationship* object. Having a look at the code it can be noticed that the *simmetrical* option is disabled, however symmetry is manually created so that if user “A” is friend of “B”, “B” is also friend of “A”.

---

```
class UserInfo(models.Model):
    def __unicode__(self):
        return self.user_name
    user_id = models.CharField(max_length=50, unique=True)
    user_name = models.CharField(max_length=50)
    user_pic_uri = models.CharField(max_length=50, default =
        "dummy_4")
    friends = models.ManyToManyField('self', symmetrical = False,
        blank = True, null = True, through='ContactRelationship',
        related_name='friends+')
    def getJsonInfo(self):
        result =
            simplejson.dumps(UserInfo.objects.all().filter(user_id=self.user_id)
                .values('user_name', 'user_pic_uri')[0][:-1]+'',
                "registered": ['
```

```

registered =
    self.nfcpoint_set.filter(registered=True).order_by('when')
    .reverse().values('name', 'posId', 'date', 'wall')
for reg in registered:
    result = result+simplejson.dumps(reg)+" , "
result= result.replace('user_pic_uri', 'pic_uri')
if len(registered) == 0:
    result = result+'], "visited":['
else:
    result = result[: -2]+'], "visited":['
#method cropped, keep building the JSON
return result

```

---

The method *getJsonInfo* returns a JSON representation of this specific object and others associated to it. It has been included to show the simplicity of database queries. *UserInfo.objects.all()* gets all the elements on the database of the *UserInfo* class. Then a filter to that list can be added with *.filter(user\_id=self.user\_id)*. What it is actually happening in the second case is that a SELECT query with the condition specified in the filter is being performed, as simple as that.

#### 4.5.1.2 Controller

Controllers wire user requests with models and views by means of URLs. Django has a file for it, *urls.py*. In that file a URL with a specific syntax is associated with a method of the controller. All the access points to UniteNFC's back-end are the following:

---

```

urlpatterns = patterns(
    url(r'^objects/users/name/(?P<user_req>+)/$',
        views.nameupdate_view, name='nameupdate_view'),
    url(r'^objects/users/picuri/(?P<user_req>+)/$',
        views.picuriupdate_view, name='picuriupdate_view'),
    url(r'^objects/users/fblink/$', views.getfb_view,
        name='getfb_view'),
    url(r'^objects/users/friend/$', views.addfriend_view,
        name='addfriend_view'),
    url(r'^objects/users/new/$', views.reguser_view,
        name='reguser_view'),
    url(r'^objects/users/(?P<user_req>+)/$', views.restore_view,
        name='restore_view'),
    url(r'^objects/nfcp/(?P<user_req>+)/(?P<isReg>+)/$',

```

```

    views.regpoint_view, name='regpoint_view'),
url(r'^objects/wall/rate/$', views.ratewall_view,
    name='ratewall_view'),
url(r'^objects/wall/updateprivacy/$',
    views.wallupdateprivacy_view,
    name='wallupdateprivacy_view'),
url(r'^objects/wall/(?P<wall_req>.+)/(?P<user_req>.+)/$',
    views.getwall_view, name='getwall_view'),
url(r'^objects/entry/delete/$', views.deleteentry_view,
    name='deleteentry_view'),
url(r'^objects/entry/(?P<wall_req>.+)/$', views.addentry_view,
    name='addentry_view'),
url(r'^admin/', include(admin.site.urls)),
)

```

To read the URL some guidelines may be necessary. The `r'^` characters denote the host part or the URL, and the regular expression `(?P<user_req>.+)` means that a string of characters is expected and will be assigned to the variable `user_req`.

Each resource is associated with a single URL, being this the only requirement to know what to answer. As you can see, every URL has associated a method in the same line, which are the ones written in the `views.py` file.

```

def restore_view(request, user_req):
    user_obj = get_object_or_404(UserInfo, user_id=user_req)
    return HttpResponse(user_obj.getJsonInfo() ,
        mimetype='application/json')

```

This method corresponds to a GET request, i.e. reading an object, the parameter passed to the URL goes as a variable inputed to the function, with it the requested object is extracted from the database and returns the view.

A write operation will be carried out through a POST request and the object to be written will go in JSON format on the request body.

#### 4.5.1.3 View

The view would be the one the `HttpResponse` method returns in the code above. Following the example, if a GET request is made to `http://unitenfc.herokuapp.com/objects/users/c1984937-5592-4868-82d0-cdde05195166/`, being `c1984937-5592-4868-82d0-cdde05195166` an actual user ID, the controller `restore_view` will

be called and will access the model generating the following view in JSON format:

---

```
{
  "pic_uri": "quantum_2", "user_name": "Test Account 2",
  "registered": [], "visited": [], "friends": [
    {
      "friend_pic_uri": "izan10saz_9",
      "friend_name": "Izan D\u0000edez",
      "friend_id": "1e9f756a-accd-4171-a5d6-c9a3afcdd7bc"
    },
    {
      "friend_pic_uri": "unitenfc_5",
      "friend_name": "Test Account 1",
      "friend_id": "6f1ffd8f-9bf8-406b-a2f5-79d39deb991b"
    }
  ]
}
```

---

## 4.5.2 Django Administration

Django provides a very interesting feature: to build an administration website for your resources. The website created will allow you to see the objects stored in your database, create new ones, edit the existing ones and delete them. Having this tool created, it is going to be of great help for testing of both the server and the complete system.

Once defined the model, just a few lines of code are needed to indicate which objects are going to be shown in the administration site. The *admin.py* file is in charge of holding the code to add more objects to the administration template. The administration created contains all the links, buttons, spinners and so on to interact with the administrator, to such point that even someone without a technical background could perform maintenance operations on the database. Access credentials will be created upon database configuration.

## 4.5.3 Deploying to the Cloud with Heroku

One of the main objectives of Heroku is to make deployment as fast as possible and with the smallest effort. Just installing the Heroku *django-toolbelt*, a command-line interface to the Heroku Web API, you can create a new application and push your project repository with Git to the Heroku server. Heroku will automatically recognize the programming language, install the required software and wire all the dependencies for you. When that process finishes you are already able to access your website from the host URL associated to your Heroku application.

Supposing you are working with VCS and a git repository, the essential steps are the following:

- Install Heroku Toolbelt:

---

```
$ pip install django-toolbelt
```

```
Installing collected packages: Django, psycopg2, gunicorn,
  dj-database-url, dj-static, static
...
Successfully installed Django psycopg2 gunicorn
  dj-database-url dj-static static
Cleaning up...
```

---

- Add a file called *Procfile* with the following content to allow Heroku identify the application and the number of processes:

```
web: gunicorn hellodjango.wsgi
```

---

- Freeze your requirement to let Heroku know the software it needs to install:

```
$ pip freeze > requirements.txt
```

---

- Create the Heroku application:

```
$ heroku create fasttest
Creating fasttest... done, stack is cedar
http://fasttest.herokuapp.com/ | git@heroku.com:fasttest.git
```

---

- Push your committed repository to Heroku:

```
$ git push heroku master
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 4.01 KiB, done.
Total 11 (delta 0), reused 0 (delta 0)
-----> Python app detected
-----> No runtime.txt provided; assuming python-2.7.4.
-----> Preparing Python runtime (python-2.7.4)
-----> Installing Distribute (0.6.36)
-----> Installing Pip (1.3.1)
-----> Installing dependencies using Pip (1.3.1)
      Downloading/unpacking Django==1.5 (from -r
        requirements.txt (line 1))
...

```

```

    Successfully installed Django psycpg2 gunicorn
    dj-database-url dj-static static
    Cleaning up...
-----> Collecting static files
    0 static files copied.

-----> Discovering process types
    Procfile declares types -> web

-----> Compiled slug size is 29.5MB
-----> Launching... done, v6
    http://fasttest.herokuapp.com deployed to Heroku

```

---

Of course some configuration steps have been omitted. Nevertheless, it should not take more than half an hour to set up everything for the first time and subsequent deployment just need of a single git push.

#### 4.5.4 Communication with Android Application

So now the server is up, the Android application has to find a way to communicate with it. As the server is REST, the client just needs to know the requests that it accepts. Those requests must be carried out in a background thread, e.g. in an `AsyncTask`.

Assuming the HTTP request was successful the server will respond with a JSON object. The objects returned by the back-end are modeled in Android as Java classes. Then with the help of the *gson* library, parsing a JSON into a Java object requires no more than a couple of lines of code and the JSON fields can be accessed through the object attributes. After this step all the data is cached with `SharedPreferences` to avoid making connections repeatedly, which will slow the application.

Although there are different URLs for different operations, the request process itself is always the same with the subtle difference of whether is GET or POST request and the received object. An example of such process would be the following:

---

```

String filename = usr.getId();
HttpClient httpclient = new DefaultHttpClient();
HttpResponse response = null;
try {

```

```

        response = httpClient.execute(new
            HttpGet(http://unitenfc.herokuapp.com/objects/users/+filename));
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
StatusLine statusLine = response.getStatusLine();
if(statusLine.getStatusCode() == HttpStatus.SC_OK){
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    try {
        response.getEntity().writeTo(out);
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    String responseString = out.toString();
    Gson gson = new Gson();
    UserInfo session = gson.fromJson(responseString,
        UserInfo.class);
} else{
    try {
        response.getEntity().getContent().close();
    } catch (IllegalStateException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    try {
        throw new IOException(statusLine.getReasonPhrase());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

---

Note that once the response String is obtained, just passing the class model to the *Gson* object is enough to parse it. Finally, further handling of bad responses and Exceptions can be implemented inside the *catch* clauses.



## 4.6 Users Management

Users using UniteNFC should be able to identify themselves, have personal records and history and interact with other people. Besides, will be interesting to offer the possibility that more than one user can use the application in the same device. The implementation of these requirements are going to be divided in user authentication with session management and user data administration.

### 4.6.1 Authentication

The topoos API provides user management, helping you to avoid implementing access forms and security flows. topoos user authentication is based on OAuth 2.0, hence, security is guaranteed. When the user is properly identified in the server a token is provided to access the services that are allowed, like getting NFC Points or uploading a picture to the topoos server.

#### 4.6.1.1 Login and Registration

Login and registration are implemented in topoos by a single Activity, *LoginActivity*. Actually, this Activity consist of a single WebView component which connects with topoos service as if a browser was it. The website is adapted for mobile phones keeping it simple trying to resemble the best to a native design. To link logged and registered users with your topoos application, the developer needs to add the *CLIENT\_ID*, i.e. one of the keys obtained when the application was registered, to the intent launching the Activity.

When the Activity starts a login form is shown. If the user is not registered yet, there is a link to a registration form. Just after registration, topoos platform will ask the user to give permissions to UniteNFC. After that it will be redirected to the login form. One interesting thing of the *LoginActivity* is that after a successful login it stores the received token into the application Context, being this the object you need to pass to all the functions of topoos Android SDK to prove your identity.

This piece of code shows how to start the mentioned *LoginActivity*. You may notice that the chosen method to launch the Activity is *startActivityForResult*. Doing so, when the Activity finishes the launching Activity receives a callback, managing in this way the result of the launched Activity.

---

```
Intent intent = new Intent((Context) this, LoginActivity.class);
    //being "this" the launching Activity
intent.putExtra(LoginActivity.CLIENT_ID, CLIENT_ID);
```

```
this.startActivityForResult(intent, 1);
```

---

#### 4.6.1.2 Logout

User session will be maintained as long as the token does not expire or the user wants to logout. First condition can be checked with topoos SDK. To check the second one, a variable called *saveuser* will be saved on the SharedPreferences (framework to store and retrieve session persistent key-value pairs of primitive data types) that will be set to true every time a user logs-in and set to false when the logged user decides to logout selecting the available option from the menu.

### 4.6.2 User Data

Once user session is properly managed, the mechanism to restore data from previous sessions and storing the changes should be provided. User data will be stored in two different ways: in the back-end's database which is available always, in the SharedPreferences of the application which is available as long as the user session is active. In other words, the user data will be stored in the cloud and cached when the user is using the application.

User data to be stored are user ID and name, profile picture URI, a list of friends and a list of NFC Points either visited or registered.

#### 4.6.2.1 Restoring User Data

When a user logs-in the application will send a request to the back-end to recover all the data belonging to this user, specifying it in the request by its ID. Section 4.5 explains in detail how the request has to be. While the application is waiting for all the data to be downloaded a loading dialog will be shown. As soon as the response is received the dialog is dismissed, the response (recall it is received in JSON format) is parsed and the data stored into SharedPreferences for quick access.

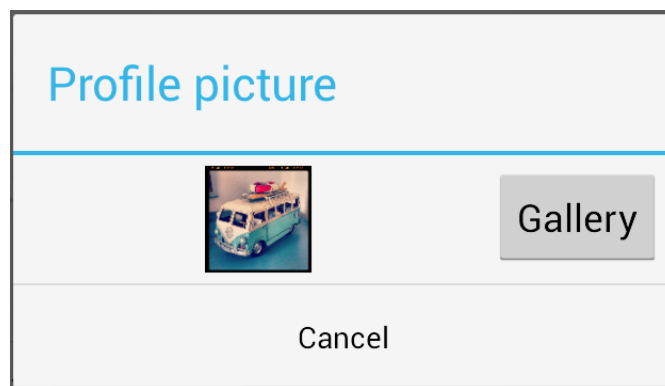
#### 4.6.2.2 Saving User Data

Every-time new information is generated or existing one is modified by the application, a back-up is sent to the cloud. This information can be of many kinds, e.g. a new NFC Point is registered, the user name is changed, a new user is created, a friend is added, etc. The changes are stored locally in the SharedPreferences (cached) and sent to the server in a POST request to the appropriate URL.

### 4.6.2.3 Profile Picture

Users have the choice of selecting an image to identify them. By default, an image with the topoos mascot will be the profile picture for new users. This image is stored within the application resources at *res/drawable*.

If a user wish to change the profile picture it can be done in the preferences menu. So far, only pictures from the gallery can be selected. After pressing the *Gallery* button, the Android gallery will be launch allowing the user to select one picture from it. That picture is uploaded to the topoos image storage service and set as profile picture.



**Figure 4.5:** Profile picture selector

Before the image is uploaded to the server, some processing is carried out to lower the weight and resolution. The following snippet check if any of the two sides of the image is larger than 98 pixels. In that case, it takes the smallest side and compress it to 98 pixels, compressing the larger size by the same ratio.

---

```

BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
if(route != null) BitmapFactory.decodeFile(route,options);
else BitmapFactory.decodeResource(ctx.getResources(),
    R.drawable.dummy, options);
int height = options.outHeight;
int width = options.outWidth;
int inSampleSize = 1;
if (height > 98 || width > 98) {
    int heightRatio = Math.round((float) height / (float) 98);
    int widthRatio = Math.round((float) width / (float) 98);
    inSampleSize = heightRatio < widthRatio ? heightRatio :

```

```

        widthRatio;
    }
    options.inSampleSize = inSampleSize;
    options.inJustDecodeBounds = false;
    Bitmap bmp = BitmapFactory.decodeFile(route,options);
    ByteArrayOutputStream stream = new ByteArrayOutputStream();
    bmp.compress(Bitmap.CompressFormat.PNG, 100, stream);
    byte[] byteArray = stream.toByteArray();
    topoos.Objects.Image i =
        topoos.Images.Operations.ImageUploadPosition(ctx, byteArray,
            name, 0);

```

---

In addition to set the selected picture (or downloaded picture in case the image is retrieved from the server when the session starts) more processing is done. Profile pictures in UniteNFC have square shape and therefore if the obtained image does not follow this convention it has to be cropped.

---

```

Bitmap bmp = TopoosInterface.LoadImageFromWebOperations(i);
int width = bmp.getWidth();
int height = bmp.getHeight();
Bitmap croppedBmp;
if(width == height) {
    croppedBmp = bmp;
}
else if(width > height) {
    croppedBmp = Bitmap.createBitmap(bmp,(width-height)/2, 0,
        height, height );
}
else {
    croppedBmp = Bitmap.createBitmap(bmp,0, (height-width)/2,
        width, width );
}
String path = Environment.getExternalStorageDirectory().toString();
File dir = new File(path,"/unitenfc");
dir.mkdir();
File file = new File(dir,"profile.png");
FileOutputStream out;
try {
    out = new FileOutputStream(file);
    Bitmap.createScaledBitmap(croppedBmp,98,98,false)
        .compress(Bitmap.CompressFormat.PNG, 100, out);
}

```

```

} catch (FileNotFoundException e) {
    e.printStackTrace();
}
catch (NullPointerException e) {
    e.printStackTrace();
}

```

---

The code shown above crops the image exactly in the center, cutting out the same number of pixels on both extremes of the larger side.

## 4.7 Facebook Integration

This section will cover the topic of adding sociability with Facebook to the application. Furthermore, it can serve as an important diffusion and promotion channel the day of tomorrow.

### 4.7.1 Setting-up Facebook Application

How to configure your application to use Facebook services is going to be explained in two parts. Firstly the application need to be registered in the system. Secondly, the SDK has to be added to the Android project and some lines of code written.

#### 4.7.1.1 Creating Facebook Application

If you want to use Facebook information resources and APIs you need to register an application within Facebook with a developer account. A native Android application is selected and the package name of the application and the main Activity class need to be indicated. Then, a key hash has to be provided, to allow Facebook identify your application securely. To obtain the key hash, as it is explained in the Facebook Getting Started tutorial [5], the following command has to be executed in the terminal.

---

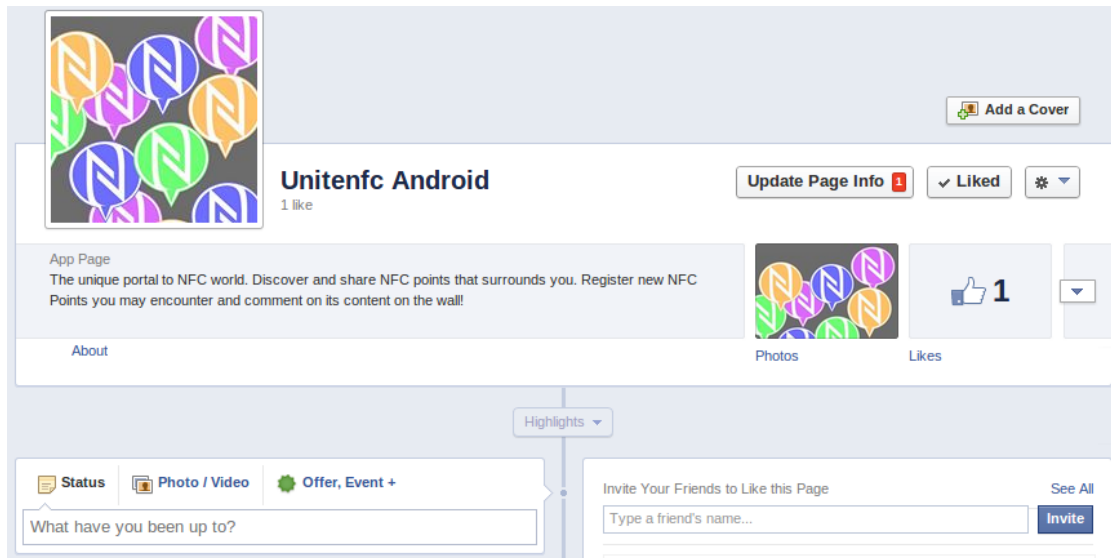
```

keytool -exportcert -alias <RELEASE_KEY_ALIAS> -keystore
    <RELEASE_KEY_PATH> | openssl sha1 -binary | openssl base64

```

---

Next step is to complete the application details for which it is compulsory to create a Facebook Page. The details of creating the page are going to be skipped, but filling up a form is all you need to do. The result can be seen in figure 4.6.



**Figure 4.6:** UniteNFC Facebook page

Permissions required for the application have to be explicitly specified. There are dozens of different types of permission, e.g. access user email, user interests, user books, friends likes, friends locations, photo upload, create event and almost every functionality available on Facebook. The permissions necessary for UniteNFC are, access to user friends and publish action. These two permissions are included by default, so there is no need to add extra permissions. The token received will be able to handle them, asking for user confirmation when a specific resource is going to be access for the first time. For example, the first time UniteNFC is going to publish on the user behalf, a prompt will be shown to ask the user for consent.

By far, Facebook set-up has been the most tedious one. Still has not finished, further configuration is needed in the Android Application to make it work.

#### 4.7.1.2 Importing Facebook SDK

Facebook SDK is somehow different than what has been found for others SDKs. Instead of being distributed as a compiled JAR file as most libraries are, the developer gets the project folder which includes its own manifest and can confuse the compiler. The steps to follow, in Android Studio, are the following: copy the *facebook* library folder to the *libs* folder. Go to *File/Project Structure...* and mark the *facebook* module as a library. Add the dependency of you application module to the *facebook* module. And, to conclude, make sure that every library is only

added once. A common error is to add the *android-support-v4* library twice, one when the *facebook* module is compiled and another one when the application module is compiled, falling into a compiler error.

As for most SDKs, the API key has to be specified to identify the application requests. In this case, a *meta-data* tag is included in the manifest indicating it.

---

```
<meta-data android:name="com.facebook.sdk.ApplicationId"
           android:value="@string/fb_app_id"/>
```

---

#### 4.7.1.3 Adding Facebook Lifecycle

Facebook SDK includes a class called *UiLifecycleHelper* that manages Facebook session status. It knows every-time the state, helping to restore previous sessions, and reacts to events by means of callbacks. According to Facebook documentation, it needs to be integrated in the Activity life-cycle for which you want Facebook functionalities. As it can be seen in the code implementing this, the helper object needs to call its own *onCreate*, *onPause*, *onResume*, *onPause*, *onSaveInstanceState* and *onDestroy*.

---

```
private UiLifecycleHelper uiHelper;
private Session.StatusCallback callback = new
    Session.StatusCallback() {
    @Override
    public void call(Session session, SessionState state, Exception
        exception) {
        onSessionStateChange(session, state, exception);
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    uiHelper = new UiLifecycleHelper(this, callback);
    uiHelper.onCreate(savedInstanceState);
}

@Override
public void onResume() {
    super.onResume();
    uiHelper.onResume();
}
```

```

@Override
public void onPause() {
    super.onPause();
    uiHelper.onPause();
}
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    uiHelper.onSaveInstanceState(outState);
}
protected void onDestroy() {
    super.onDestroy();
    uiHelper.onDestroy();
}

```

---

Note that the callback object class is also provided in the SDK and that the method called in it is not shown but will handle what actions are taken when the session status changes, e.g. when a user connects with an existing session when the Facebook button is pressed.

## 4.7.2 Connecting Account

After a long set-up, fortunately, many automated functions and widgets are ready to use. The user will be asked to connect with Facebook right after the login. Facebook SDK provides a *LoginButton* element that handles the connection, making automatically all the necessary requests. That button is going to be embedded into the dialog that is going to be shown to be user on session start.

---

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <com.facebook.widget.LoginButton
        android:id="@+id/authButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>
    <CheckBox
        android:layout_width="wrap_content"

```



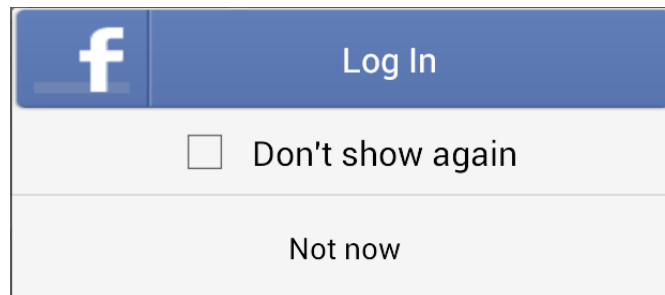
```

        android:layout_height="wrap_content"
        android:text="@string/fb_reminder"
        android:id="@+id/checkbox"
        android:layout_below="@+id/authButton"
        android:checked="false"
        android:layout_centerHorizontal="true"/>
</RelativeLayout>

```

---

This XML code will be inflated into the dialog shown to the user, being the result the shown in figure 4.7. The user can choose to login with Facebook, cancel or cancel and not show again the dialog. Besides, this dialog can be accessed from the preferences, allowing to login or logout at any time.



**Figure 4.7:** Facebook Dialog

The complete flow of Facebook login and logout is depicted in figures 4.8 and 4.9, one for each entry point that has been defined.

### 4.7.3 Adding Friends

How friends are added is almost completely explained in the two flow diagrams. Furthermore, although the user interaction may be different, how friends are searched and found is done in the same way for both cases.

When a user is linked what is being done is to create a one to one relation in UniteNFC's back-end with the topoos ID (i.e. what is treated as UniteNFC ID) as a value and the Facebook ID as a key. Then, a query is carried out to Facebook to obtain all the Facebook user friends and another one to UniteNFC's server to get all the mentioned one to one relations in the shape of key-value pairs. Every user that has previously connected with Facebook will be among them, allowing to ask for a given key (all the Facebook friends IDs) and, if not null, obtain the

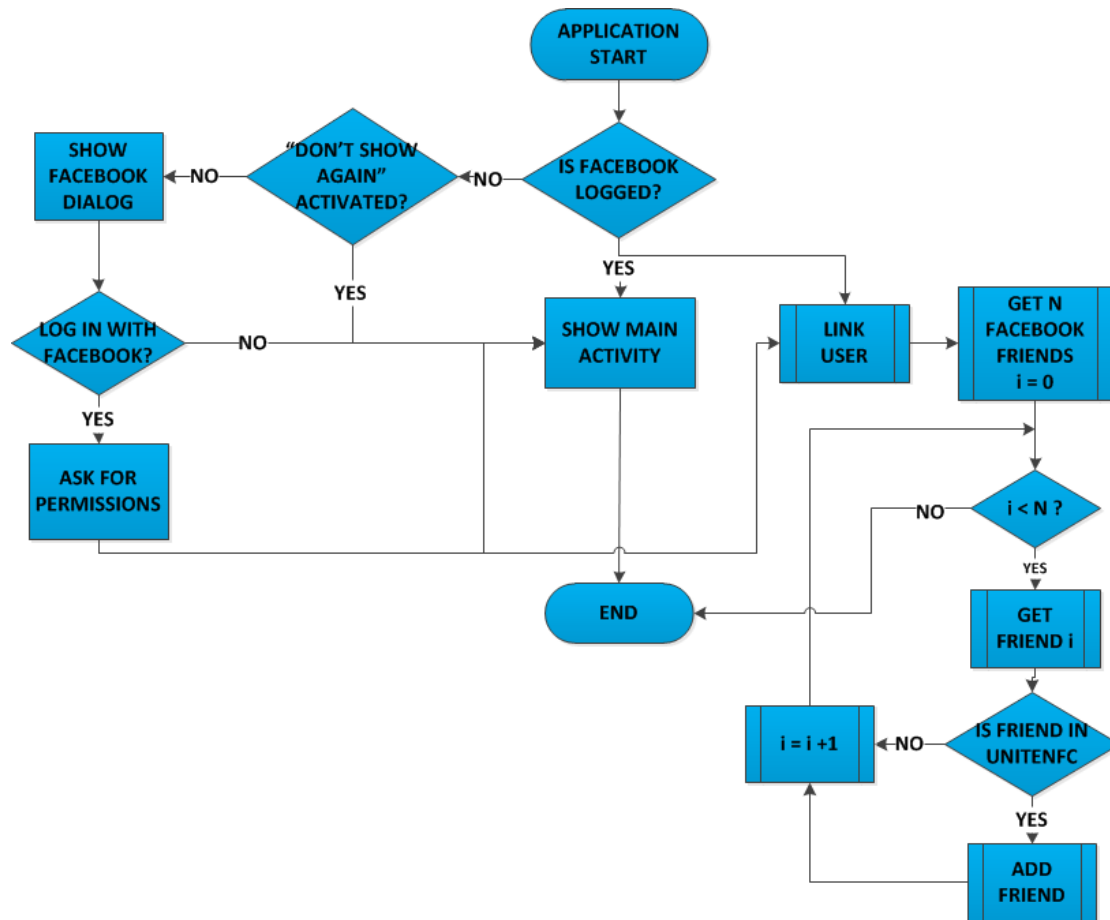


Figure 4.8: Facebook flow (I)

linked user in UniteNFC. Finally, when a friend is found it is added in UniteNFC and backed-up to the server.

#### 4.7.4 Publishing on Wall with Hashtags

Last functionality to explain is how to publish in user’s wall automatically. This feature will allow the user to share his activity in the application, when the user registers a new NFC Point or visits an existing one, in Facebook. Besides, thanks to the hashtag marks that recently Facebook has activated an interesting opportunity of classifying messages and user interaction is opened. The following method contains all the instructions needed to publish a message.

Permissions are requested at the beginning if they have not been provided previously and then, a Bundle is created in which all the data to be posted is

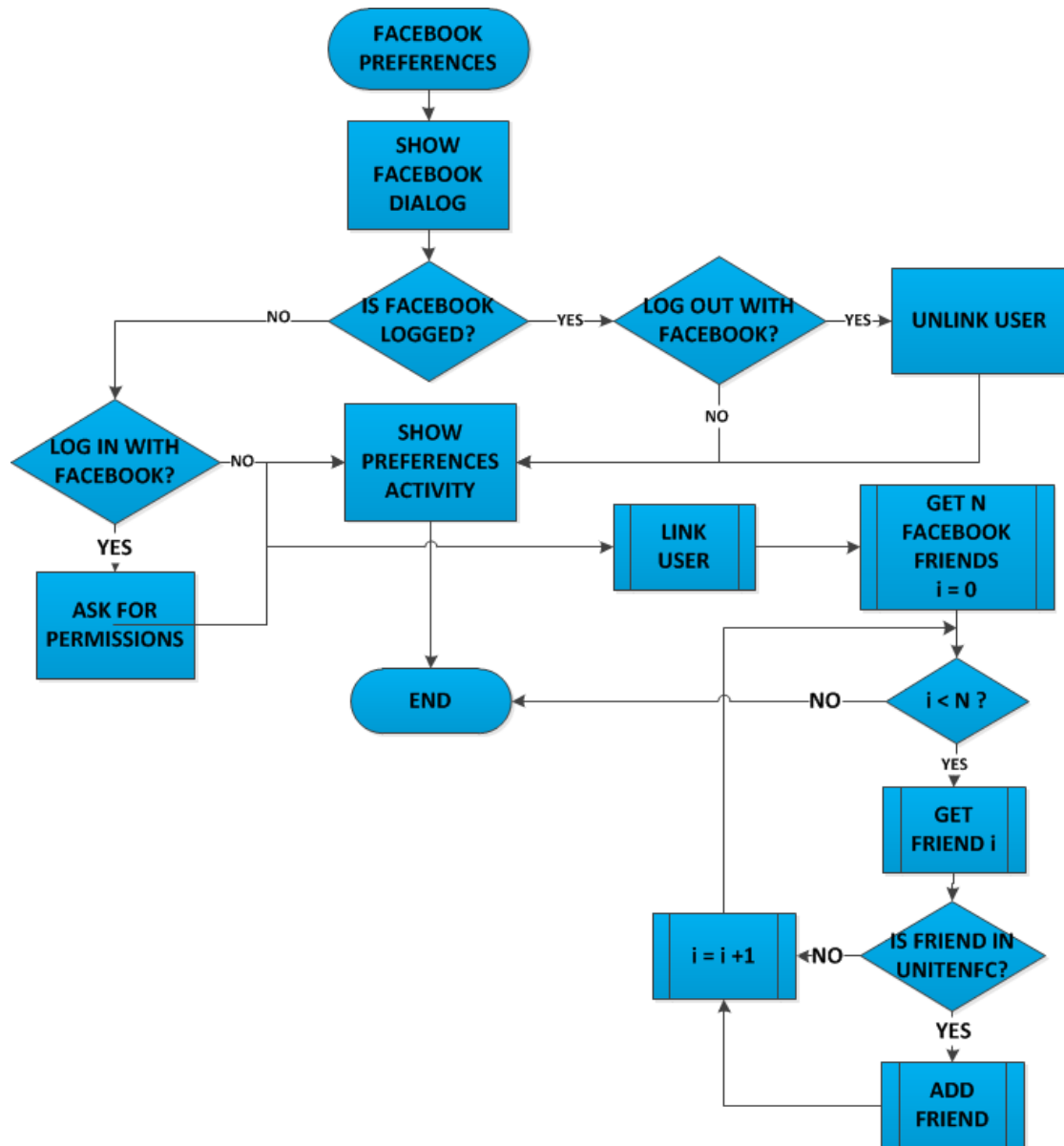


Figure 4.9: Facebook flow (II)

appended. The actual message that will be published is the *message* String inputted to the method.

---

```

public static void publishStory(Activity act, String message) {
    Session session = Session.getActiveSession();
    List<String> permissions = session.getPermissions();
    if (!isSubsetOf(PERMISSIONS, permissions)) {

```

```

        pendingPublishReauthorization = true;
        Session.NewPermissionsRequest newPermissionsRequest =
            new Session.NewPermissionsRequest(act, PERMISSIONS);
        session.requestNewPublishPermissions(newPermissionsRequest);
        return;
    }
    Bundle postParams = new Bundle();
    postParams.putString("message", message);
    postParams.putString("name", "UniteNFC for Android");
    postParams.putString("caption", "The unique portal to NFC
        world.");
    postParams.putString("description", "Discover and share NFC
        points that surrounds you. Register new NFC Points you
        may encounter and comment on its content on the wall!");
    postParams.putString("link",
        "https://developers.facebook.com/android");
    postParams.putString("picture",
        "http://s23.postimg.org/bq7oqpfzr/nfc_blue.png");
    Request.Callback callback= new Request.Callback() {
        public void onCompleted(Response response) {
            JSONObject graphResponse = response
                .getGraphObject()
                .getInnerJSONObject();
            String postId = null;
            try {
                postId = graphResponse.getString("id");
            } catch (JSONException e) {
                Log.i("TAG", "JSON error "+ e.getMessage());
            }
        }
    };
    Request request =new Request(session, "me/feed",
        postParams, HttpMethod.POST, callback);
    RequestAsyncTask task = new RequestAsyncTask(request);
    task.execute();
}
}

```

---

The structure of the message will always be the same: the name of NFC Point, the action performed (visited or registered), the type of the NFC Point and a series of hashtags including the physical tag ID, the *NFC* word, and the application

name. Note that if someone wants to see all the messages posted on Facebook from UniteNFC it could be done just clicking the hashtag. Moreover, imagine how easy is to track all the records for a particular NFC Point. Possibilities if users tend to share their activity in Facebook are huge. An example of a message posted when an NFC Point is visited has been captured in figure 4.10.

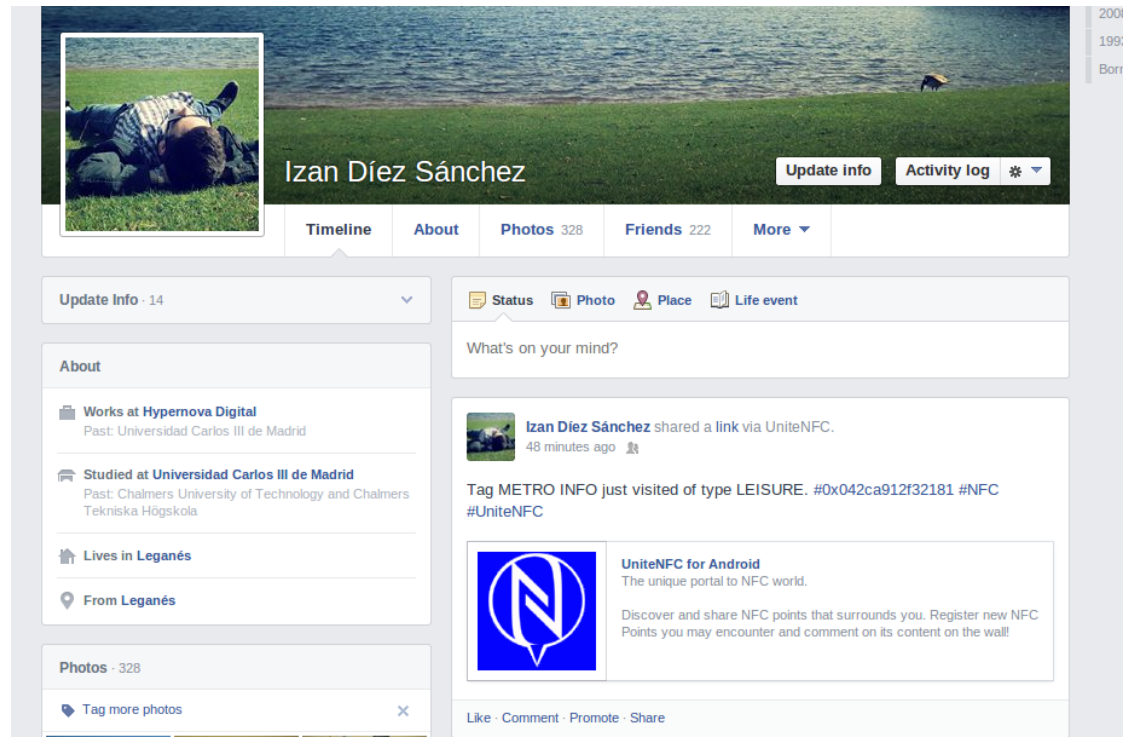


Figure 4.10: Example publication on Facebook

## 4.8 Adding Language Support

Users expect to use the application in its local language and not being able can establish a big entry barrier. Supporting several languages is a key aspect to attract users all over the world. Android provides a very simple mechanism for language support that is going to be described briefly.

Currently UniteNFC supports two languages, English and Spanish.

### 4.8.1 Strings in Android

Strings in Android can be handled in two ways, either you declare the String directly in the application code or you get the String from a resource file where all the Strings are declared. One of advantages of having a resource file with the Strings is reutilization, you write you String once and use it in several places of your code, for example an error message that is shown in different activities. Secondly, you can have different String resource file for different languages without changing a letter on you code, since strings are identified by an id (i.e. *name* field of the *string* item in the XML file) and Android knows which resource file has to look at. The following piece of code shows how to declare the String in the resource file.

---

```
<resources>
  <string name="action_share">Share</string>
</resources>
```

---

Get it in XML for the layouts:

---

```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/action_share" />
```

---

Get it in Java:

---

```
String string = getString(R.string.action_share);
```

---

The same concept applies for String arrays. They need a resource file for each language you want to support.

### 4.8.2 Resources Folders

Under the *res* folder all the resources including XML and images are placed. Strings and arrays, among others, are inside the *res/values* folders. It has been said that Android knows which resource folder corresponds to each language, and it does just looking at the folder name.

Knowing the system language, Android looks for the language code after the folder name. In figure 4.11 *values-es* stands for Spanish language. If no folder matches the sought code, it falls back to the default language residing in the *values* folder, which in UniteNFC is English.

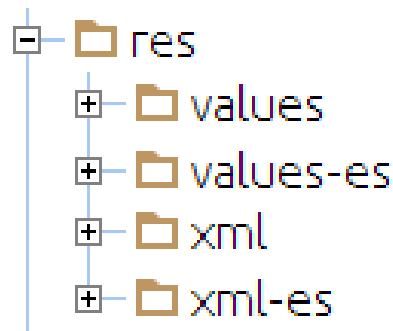


Figure 4.11: Different languages resource folders

### 4.8.3 Summary: How to Add a New Language

To summarize, if support for a new language, e.g. German, was going to be added the following steps would need to be followed:

- Create a new *values* folder with the language code at the end, e.g. *values-de*.
- Add all the Strings from other language resource and, keeping the id, substitute all the values for the corresponding translated String, e.g.

---

```
<string name="action_share">Teilen</string>
```

---

## 4.9 Notifications

An important feature of UniteNFC is the possibility of receiving notifications on proximity of NFC Points. Notifications improves discoverability of NFC Points that the user would not be aware of unless a direct interaction with the application map was taking part. The concept is simple, an Android Service will run periodically polling for NFC Points on the surroundings of the user location. If any NFC Point is found, and has not been notified recently (in case the user is not moving), a notification is sent to the notifications bar. How to program these notifications will be explained in more detail.

### 4.9.1 Launching Service

A Service is an Android element corresponding to a task of an application that does not need user interaction nor graphical interface or a task to supply functionality for others applications. UniteNFC needs a Service to be run periodically and to do so, Android provides the AlarmManager which allows you to set periodical alarms.

---

```

Intent intent = new Intent(this, ProximityNotifier.class);
PendingIntent pintent = PendingIntent.getService(this, 0, intent,
    0);
AlarmManager alarm =
    (AlarmManager) getSystemService(Context.ALARM_SERVICE);
int PERIOD = 30000; //milliseconds
alarm.setRepeating(AlarmManager.ELAPSED_REALTIME,
    SystemClock.elapsedRealtime() + PERIOD, PERIOD, pintent);

```

---

In the snippet above an alarm is created to be repeated every *PERIOD*, i.e. 30 seconds. When the alarm fires, the `PendingIntent` waiting for it sends the explicit Intent that launches the Service which will determine whether an NFC Point is near the user.

### 4.9.2 Service

The `IntentService` will handle the sent Intent in the `onHandleIntent(Intent intent)` in which all the logical work will be performed. The user location will be updated and then a function will retrieve all the NFC Points in terms of a center location and a radius. This function belongs to the *topoos* API which makes the operation really simple. From the returned NFC Points the closest one is chosen an broadcast to the Android system in an Intent. In the code, an Intent is filled up with the NFC Position (latitude and longitude) and type.

---

```

Intent localIntent = new
    Intent().setAction(Constants.BROADCAST_ACTION)
        .putExtra(Constants.EXTENDED_DATA_STATUS, poi.getCategories().getId())
        .putExtra("lat", poi.getLatitude())
        .putExtra("lon", poi.getLongitude());
sendBroadcast(localIntent);

```

---

### 4.9.3 Notification Creation

The last step consists of creating the `BroadcastReceiver` that will get the message with the NFC Point position and type and create the notification. The Notification is constructed with a `Builder` and several items can be added.

---

```

PendingIntent contentIntent = PendingIntent.getActivity(context, 0,
    new Intent(context,
        MainActivity.class).setAction(Constants.NOTIFY)

```

---



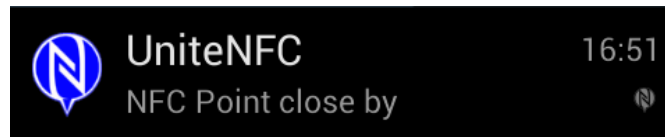
```

        .putExtra("lat",latitude).putExtra("lon",longitude), 0);
NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(context)
    .setSmallIcon(R.drawable.scan_tab)
    .setLargeIcon(BitmapFactory.decodeResource(context.getResources(),res))
    .setContentTitle(context.getString(R.string.app_name))
    .setContentText(context.getString(R.string.notification_text));
mBuilder.setContentIntent(contentIntent);
mBuilder.setDefaults(Notification.DEFAULT_SOUND);
mBuilder.setAutoCancel(true);
mBuilder.setVibrate(new long[]{100, 100, 100, 400});
mBuilder.setLights(Color.CYAN, 300, 300);
NotificationManager mNotificationManager = (NotificationManager)
    context.getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(1, mBuilder.build());

```

---

In this case, a large and an small icons are added, a title and a description and a PendingIntent that will fire when the users selects the notification. That Intent will have the NFC Point position information to locate it at the map automatically for the user. Besides, the notification sound, vibration pattern and LED blinking color and pattern are set as well.



**Figure 4.12:** Notification of nearby NFC Point

## 4.10 Google Analytics

In this section how to use Google Analytics and what you can get from it is going to be described.

### 4.10.1 Setting-up Account

Any Google user can access this service and start tracking websites or mobile applications such as UniteNFC. The first step is registering the application filling in a simple form in which you specify the application name, the industry sector, time zone and a few more options. Given that you obtain a track ID, which will

be the application identifier when it communicates with the service. That is all concerning account set-up. Lots of options are available but default configuration is enough for beginners and still you get a lot of information.

### 4.10.2 Adding Analytics to the Android Life-cycle

Once you have the track ID it is possible to integrate Analytics services into the Android Application. Google Analytics SDK library needs to be added to the *libs* folder. Now, the following XML file has to be added to the *res/values* folder with name *analytics.xml* to indicate the track ID and other characteristics.

---

```
<resources>
  <string name="ga_trackingId">UA-XXXXXXXX-X</string>
  <bool name="ga_autoActivityTracking">true</bool>
  <bool name="ga_reportUncaughtExceptions">true</bool>
</resources>
```

---

The string parameter is the already mentioned track ID which is indispensable for Analytics to work. The second and third line configure automatic tracking and exceptions reports respectively, and are set to *true*. At last, just a few lines of code are left to be written to have the application monitored by Analytics services.

---

```
@Override
public void onStart() {
    super.onStart();
    EasyTracker.getInstance(this).activityStart(this);
}
@Override
public void onStop(){
    super.onStop();
    EasyTracker.getInstance(this).activityStop(this);
}
```

---

In this code it can be seen that just one line of code is necessary to start *EasyTracker* and another one to stop it, including them in the normal Android life-cycle for every Activity that is going to be tracked. *EasyTracker* encapsulation will do all the work of communicating relevant events to the server.

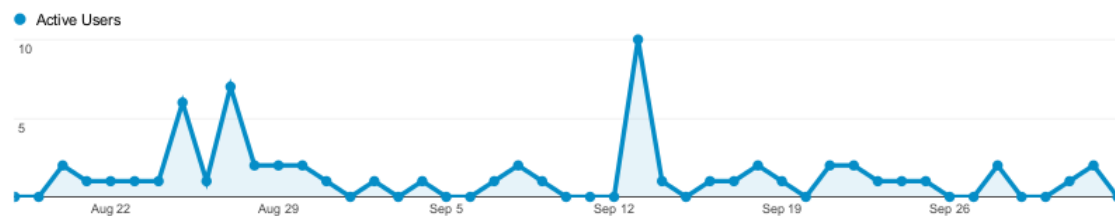
### 4.10.3 Statistics

It has been explained how easy is to put Google Analytics to work. Yet no result has been shown. Statistics gathered are diverse and include, among others:

- New users
- Active users
- Location
- Language
- Operating system
- Service provider
- Device
- Screen resolution
- Number of screens per session
- Sessions duration
- Screen views
- Crashes and exceptions
- Engagement flow

There is no doubt about the usefulness of this information. Google Analytics reports can determine which languages are chosen to be supported in the first place or to what screen dimensions is interesting to adapt the application to please the consumer, help to know how user recruitment is evolving or which Activities (screens) are preferred or, on the contrary, causes users to stop using the application.

To round off, data are shown in a very attractive way. Figure 4.13 shows a chart of the number of active users per day on UniteNFC. Note the peaks correspond to the testing spikes for two different versions of the applications.



**Figure 4.13:** UniteNFC user sessions statistics (03/10/2013)

On figure 4.14 statistics about devices on which UniteNFC has run, with its operating system and network provider are depicted. These numbers are biased by the main testing device which clearly was an Xperia running an Android 4.1.2 and with network provider Telefonica.

The languages of devices using UniteNFC can be seen in figure 4.15. Again note that most of the sessions correspond to tests which were carried out in a device with Spanish language configured.



Figure 4.14: UniteNFC device statistics (03/10/2013)

Language	Sessions	% Sessions
1. es-es	448	87.33%
2. en-us	60	11.70%
3. ar-eg	3	0.58%
4. ca-es	1	0.19%
5. pl-pl	1	0.19%

Figure 4.15: UniteNFC language statistics (03/10/2013)

Last example, figure 4.16, shows screen flow represented as a net of connected Activities and gives a very intuitive view of user costumes when using the application in such a way you could think of manipulating them to serve your purposes.

## 4.11 Publishing on Google Play

Even though when the application development started there was no plan to publish UniteNFC in the market, once it was finished there was no reason to stop doing it and check whether some people downloads it or not.

## CHAPTER 4. IMPLEMENTATION



**Figure 4.16:** UniteNFC engagement flow statistics (03/10/2013)

Accessing Google Play is easy and uploading your first application very quick with very few restrictions. The only thing you need to do is to register as a developer and pay the 25\$ lifelong fee. As soon as you do it you can start publishing. The information required to let you publish the application is the following:

- Default language: should be the language by default in the application, since people expect the application info in the same language as the Google Play entry. For UniteNFC, default language is English.
- Title: the application name.
- Description: description up to 4000 characters.
- Screen-shots: at least two screen-shots of the application.
- High resolution icon: 32 bits PNG file with resolution 512x512 to be shown in the application page. UniteNFC's was already shown in figure 3.1.
- Application type: game or application.
- Category: application purpose or field. Among the available options UniteNFC best fits to the communication category.
- Content classification: maturity level. Since it is a first release and its publication information is not as complete as it should, low maturity level was chosen for UniteNFC.
- Website: application website. At this moment the website is [www.unitenfc.com](http://www.unitenfc.com) and redirects to the Facebook application page.

- Email: developer contact email.
- Privacy policy: URL with the privacy policy for the application. It can be left blank, as for UniteNFC.
- Distribution price: price to charge for its download. To try to attract the most users as possible it will be free for now.

That is all the information you need to provide to publish the application. You can complete it with more screen-shots for different resolutions, adding an advertising text or indicating your last changes. Then Google Play allows you to upload your application in three different schemes: production, beta Testing and alpha Testing. In alpha and beta testing you can choose who has access by specifying a Google Group or Google+ community. UniteNFC was launch directly in production as no group of testers were planned to analyze the application and anyhow this could be done in production as well.

The application is uploaded in APK (Application PacKage File) format, i.e. the file format used in Android to distribute applications. It must be digitally signed with a certificate whose private key is held by the application's developer [44]. Generating the signed APK with Android Studio IDE is straight forward. Going to *Build/Generate Signed APK...* a wizard is shown in which you can select a private key under your control or create a new one and finally create the APK of your application to upload it. Caution must be taken to store the key since all the versions of the application have to be signed with the same key.

### 4.11.1 Versioning

Different versions of the application can be uploaded as it is improved. Users who have installed the application will receive the notification of a new version or it will be updated automatically depending on the configuration. To define a new version two fields need to be changed in the manifest.

---

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.quantum.unitenfc"
  android:versionCode="2"
  android:versionName="1.1" >
```

---

The *versionCode* field is a positive integer indicating version order with respect to others. A newer version must always have a greater value than previous versions, allowing the system to check for upgrades or downgrades. On the other hand

## CHAPTER 4. IMPLEMENTATION

*versionName* is only the version code shown to users. To conclude, note that the *package* field must remain the same for all the versions.

# 5

## TESTING AND FURTHER DEVELOPMENT

This chapter will address the methodology followed to test the application in the different development stages. In addition, a set of possible future tasks will be listed.

### 5.1 Testing

One fundamental step in the software development flow is testing. When talking about software testing several approaches are available: depending on how frequently you test, whether you test functions in a unitary way, by blocks or the complete system; whether test are carried out in an automated way or manually, etc. [37]

Due to the mixed characteristics of the project and the need, at some points, of testing directly the hardware, different methods were used for each phase of the project, not following any particular convention. The next sections will describe the four main processes repeated for testing the developed application.

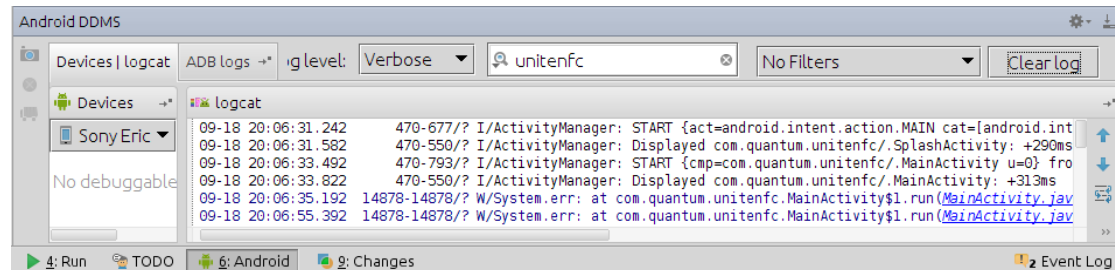
#### 5.1.1 Debugging the Android Application

Android AVD provides a way to test you application without owning a physical device. It comes with the SDK and you can define a wide range of virtual devices. Some are predefined, but you can define your own specifying its characteristics: CPU architecture, RAM memory, hardware available on the emulated device and screen density. So far, a very flexible and useful tool, but it is not that simple.



When it comes to performance there is still a lot of work to be done by Google, the emulator is just too slow. Apparently they have not found the way to convert between architectures efficiently. The requirements are huge and having a powerful equipment does not guarantee emulation smoothness either. Furthermore, not every hardware can be emulated. For example there is no way to test NFC into the emulator. For these reasons all the tests were made directly into the mobile phone.

The fact of needing a physical device slows testing but, apart from that, all the tools can be used normally. Logcat provides an interface to Android system messages, allowing you to filter by type of message, process, intent, activity and date. Logcat was the main tool used to monitor the application flow during testing. Figure 5.1 shows a sample screen-shot where Logcat verbosity can be appreciated.



**Figure 5.1:** Logcat example logs

When an error was detected in Logcat but the cause was hard to figure out, a more precise analysis of the code had to be performed. The Java code could be debugged as in the virtual machine when running on the mobile phone, so some breakpoints were enough to make the application stop when desired and inspect if the variable values were as expected.

In summary every time a new functionality was added to the application the mentioned steps were followed, iterating until no errors arose. Direct interaction with the mobile plus Logcat examination. If better understanding was needed the debugger made the rest.

### 5.1.2 REST API Testing

Testing on the server was completely isolated from the client application thanks to its RESTful behavior. As explained in 3.4.2.1.2 the cloud service is stateless in such a way that the response only depends on the request. Therefore, testing every kind of request and assuring the response is the correct one is more than

enough. There is no need to care on how client and server are connected or how the data will be handled.

For this task the REST console [38] was the selected tool, which is just an HTTP request visualizer and constructor tool. Basically the request was constructed with the form *host+intruction* if it was a read request, i.e. GET, or *host+intruction* plus a JSON body if it was a write instruction, i.e. POST. Then, the response was viewed and checked, first, that the JSON format was correct and, second, that the received item was the expected according to the data present in the database. The first verification was made in <http://jsonviewer.stack.hu/>. The second verification could be easily made thanks to the administration site provided by Django, where all the objects in the server can be seen. Figure 5.2 shows these steps for a better understanding of the reader.

Worths to mention other services being REST, e.g. topoos, could be tested in a similar way with the REST console.

### 5.1.3 System Testing

Once the server and all the services are integrated, further testing has to be carried out. As a result of not having followed an strict testing methodology, when all the working parts were assembled some of them broke. Hence the full application was tested (like an alpha testing) and fixes were made until everything worked. No specific method was followed and, depending on the bug, a mixture of the already described procedures were followed.

### 5.1.4 Feedback

Getting feedback is an interesting tool to have somewhat a continuous like testing. Feedback allows you to know what is happening on you application, what works and what does not, and many more things. Two feedback sources were added to the application, thus being able to detect bugs from the user experience:

- User bug report: an option was added in the menu to allow users report unpleasant experience like crashes. The first users reported some of them, spotting some sloppy code that was overlooked.
- Google Analytics: not only gives you information about the users, it gives activity recurrence information and exceptions thrown. Very useful to monitor the application health.

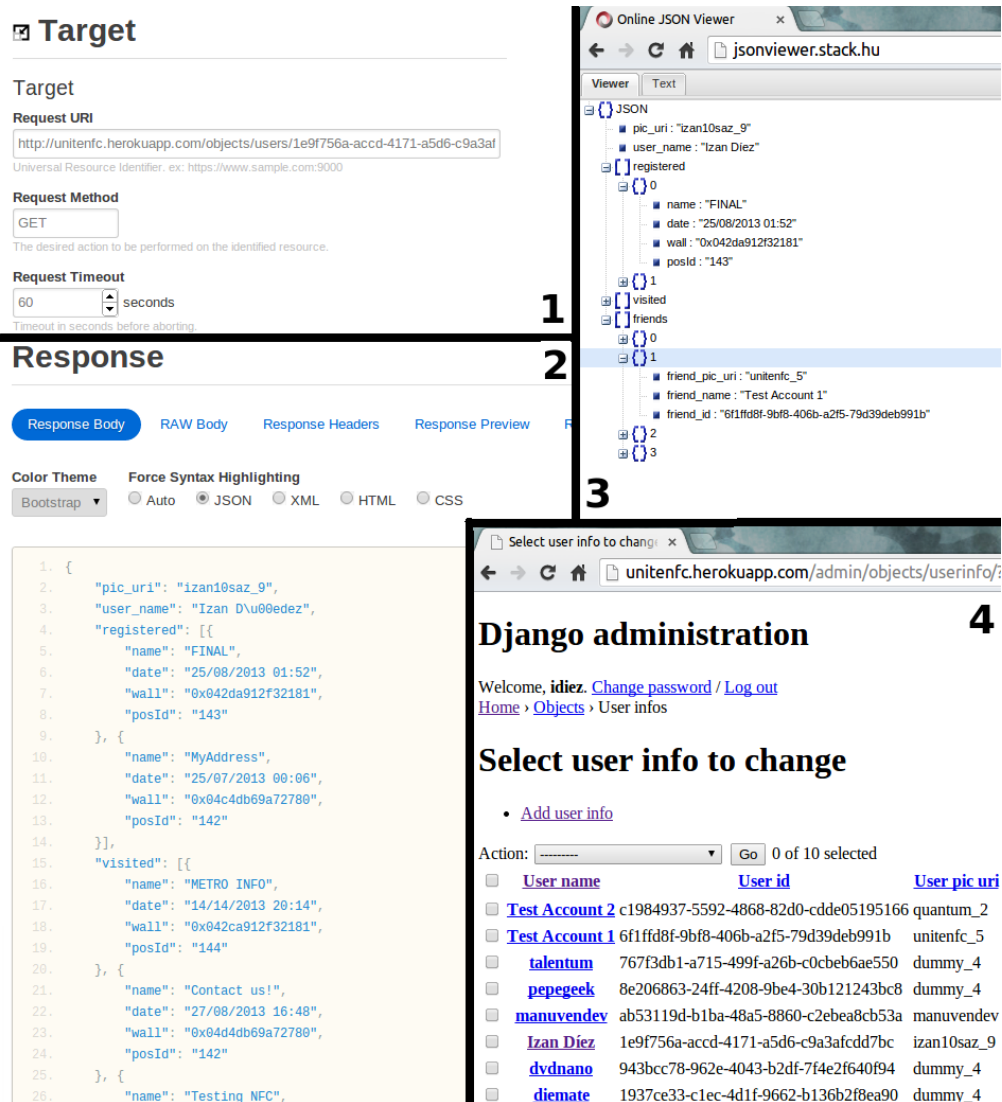


Figure 5.2: REST API testing

Finally, although no formal testing team was put to play around with the application, some acquaintances agreed to give a try to the first version and very valuable feedback was obtained. Many bugs were identified and solved in later versions.

## 5.2 Further Development

This section will describe some tasks that would need to be performed if the project is continued in the future. Most of them do not cover any of the technical issues discussed on this thesis so, due to the lack of time, have been considered to fall apart from the scope of the project.

### 5.2.1 Optimizing for Other Screen Dimensions and Densities

Android fragmentation is a nightmare for developers as will be discussed later in section 6.2, one of the reasons is the diversity of screen dimensions and densities. UniteNFC has been developed only for the testing device, Sony Xperia S, which holds a 342 dpi screen falling into XHDPI classification and in portrait mode. The application interface may be oversized for smaller densities and not well distributed over a tablet as can be observed in figure 5.3.

### 5.2.2 Adding More Language Support

UniteNFC offers multi-language support from its inception, in Spanish and English. To increase the scope and reach many more countries additional languages should be added. Simply including one strings resource file for each language is enough.

### 5.2.3 Better Instructions in the Application

Many people may not know what is NFC, or find the application complex or nonsense. Clearer messages on what it is and how to use it has to be added both in the application and in the Google Play entry. This step is crucial to attract new users. Video promotions and tutorial may be of great help.

### 5.2.4 Optimizing Image Loading and Cache

Interfaces containing images or data loaded from the web show a loading progress bar and then inflate at once all the graphics. Caching better that data, some it is actually done, and allowing to asynchronously inflate the interface by dynamic graphics inflation could improve a lot user experience.

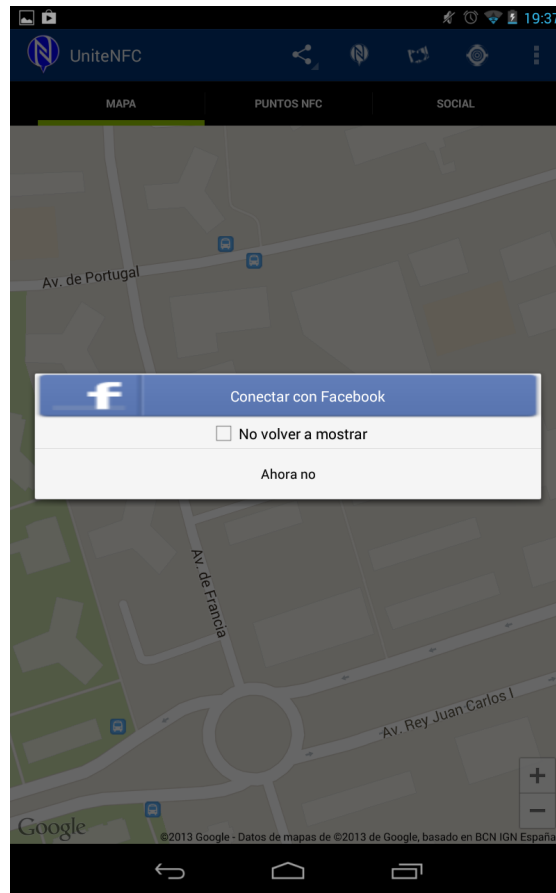


Figure 5.3: UniteNFC in a tablet

### 5.2.5 Code Cleaning and Maintenance

Code's value decreases as its entropy increases. Code tidiness has been tried to keep during the programming stages. However, when the code got bigger and the deadlines became closer was almost impossible to avoid messing up the code. If a new developer is to come and looks at the code, should be able to understand it. For that to be possible some refactoring needs to be done ensuring posterior maintenance.

# 6

## PROBLEMS ENCOUNTERED

This chapter will comment on the difficulties and problems that arose during the project development.

### 6.1 Testing Hardware Related Functionalities

Needing to test NFC slowed down the programming a lot. The main problem is that you cannot test a function without building the whole system into the mobile phone. As was commented in the previous section this limited the testing to the system testing, also known as black-box testing. To deal with this problem lot of patience was invested, but not smart solution was found. The greatest delay was caused by the testing of Android Beam, which needs of two devices. One was borrowed but was not available always.

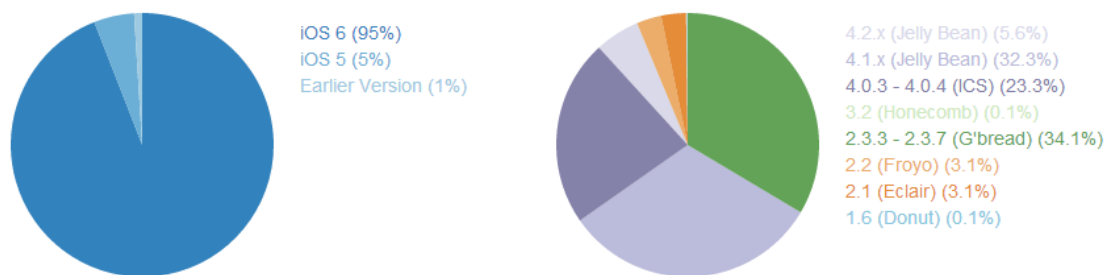
One possible solution to think of for next projects is to use the Open NFC Simulator [35]. Running on Windows, it emulates two NFC controllers and a variety of virtual cards and can be connected with an Android Virtual Device. In principle, the application could be tested without the need of any hardware. However, the lack of examples on the website can be taken as a reference of the state of the project.

### 6.2 Android Fragmentation

Fragmentation in Android ecosystem is a controversial topic and there are opinions of all kinds. On one hand you have the freedom of an open system, everyone can tweak the operative system and make it work on their device making it very

easy to spread. On the other hand you have device and version fragmentation, having different hardware specification for the former and software features for the later, losing control and optimization.

As a developer you can find fragmentation a bit of a mess. Extra work has to be done to fit properly an application to all the screen densities and sizes, you cannot use the same hardware in all the devices or some functions you use from the SDK may not be supported in previous versions. These issues hardly appear in the main competitor, i.e. iOS, which higher control helps to avoid them.



**Figure 6.1:** Android vs. iOS fragmentation [45]

For the sake of the project the scope of Android versions was reduced. A critical point was the introduction of the new NFC functions in API 14 that allowed to develop NFC applications easily and compatible with 4.0 or higher versions. Therefore, that API was set as minimum SDK version leaving out, approximately, half of the total Android devices. Nevertheless, note that very few of those devices have NFC support and would not be compatible devices in any case. A complete list of compatible devices can be found in appendix A.

### 6.3 NFC Android Beam

Peer-to-Peer protocol stack allows to built many protocols on top of LLCP as was seen in 2.18 and not restricted exclusively to SNEP or NPP. The first idea was to built up an own protocol to transfer friend data and perhaps some other data like tag content. However, this is not possible with the current Android SDK, in which the low level functionalities are inaccessible to the developer.

P2P specification says it is bidirectional, but Android Beam is not. That issue, does not allow to perform a two way operation in a single NFC communication. To simulate this behavior, the response had to be sent over the Internet and processed

on the server making the communication in one single mobile.

Finally, Android Beam requires user interaction (touching the screen) and loses much of the NFC charm. If two people are holding their unlocked mobiles back to each other they certainly know they want to share something, why asking again?

## 6.4 Topoos Incomplete API

Topoos API offer is attractive. Point of interest support, geolocation and social functions, etc. But it is still young. Before being chosen as one of the main building blocks of the application, more research should have been done to know exactly what worked and what did not. The BETA word was trying to warn but was overlooked. Due to this bad election in the beginning some restructuring was carried out to implement for example, friendship. Nevertheless, it has to be said that what was working by then was performing rather well and by know its promises are closer to reality.

## 6.5 Web Development Inexperience

When the project was first depicted, one of the main challenges was to learn some good practices on web development and applying them to develop a web server to store and deliver data to the client applications.

Learning a new framework is not easy and documentation is not always good enough. That was one of the reasons of choosing Django and Python. But still some headaches were produced by the complicated documentation.

Although the project was slowed down a bit because of this, the planned services were implemented correctly. But security was too big to handle. An oauth2 flow should be implemented. Support of a more experienced web developer would be needed.



# 7

## PLANNING AND BUDGET

This chapter will present a Gantt Chart with all the tasks involved in the project development and, in a later section, a complete budget will be calculated in order to estimate the cost of replicating the project.

### 7.1 Gantt Chart

To be able to calculate the budget of the presented project, a breakdown of the main tasks carried out is shown. Note that these tasks are part of the stages described in section 1.3. An estimation of the hours spent in each task is also presented to give an outlook of the weight of each task in the project.

Another issue to have into consideration when regarding the tables and figures that will follow is that: during the first five months, only four hours a day (five days a week), i.e. part time, were dedicated to the project development; whereas the last four months, eight hours a day, i.e. full time, were invested in it.

In table 7.1 all the tasks are listed in chronological order. Overlapping tasks share the period of time fairly, in other words, time is split equally in both tasks as mostly often was.

Finally a more visual schedule is presented by means of the Gantt chart in figure 7.1.

CHAPTER 7. PLANNING AND BUDGET

Task	Starting Date	Ending Date	Number of hours
NFC Research	07-01-2013	25-01-2013	50
NFC Documentation	21-01-2013	01-02-2013	30
Workbench Setup	04-02-2013	08-02-2013	10
System Definition	04-02-2013	15-02-2013	30
GUI Implementation	18-02-2013	08-03-2013	30
NFC Block Implementation	25-02-2013	10-05-2013	210
Topoos Block Implementation	13-05-2013	21-06-2013	130
RESTFUL Web Development	24-06-2013	26-07-2013	200
Friendship Implementation	29-07-2013	02-08-2013	40
Walls Block Implementation	05-08-2013	16-08-2013	80
Facebook Integration	19-08-2013	23-08-2013	40
Complete System Integration	26-08-2013	30-08-2013	20
Google Analytics Integration	02-09-2013	06-09-2013	20
Testing and Bug Fixing	26-08-2013	20-09-2013	95
Publication on Google Play	16-09-2013	20-09-2013	5
Thesis Writing	09-09-2013	04-10-2013	100
Thesis Revision	07-10-2013	11-10-2013	40

**Table 7.1:** Project tasks breakdown

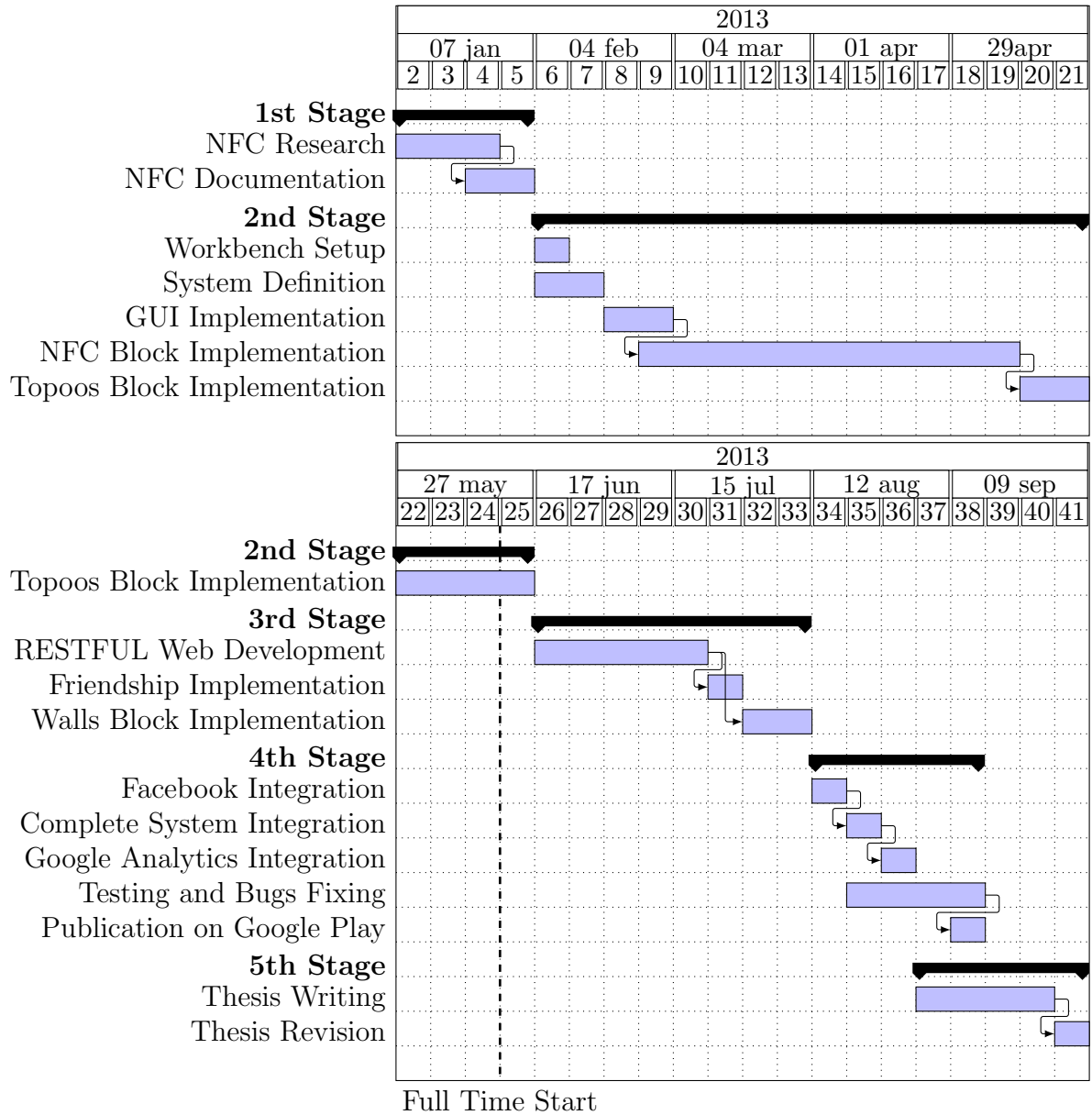


Figure 7.1: Gantt Chart

Surname and Name	Professional Category	Devoted Time Worker x Month	Worker Cost by Month (Euros)	Total Cost (Euros)
Díez Sánchez, Izan	Engineer	6.5 <sup>1</sup>	2,694.39 €	17,513.54 €

**Table 7.2:** Personnel budget

Description	Cost (Euros)	Project Devoted Time (%)	Devoted Time (Months)	Depreciation Cost	Attributable Cost
Sony Vaio VPCCA1S1E	696.18 €	100	9	60	104.43 €
Logitech Wireless Mouse M305	24.56 €	100	9	60	3.68 €
Sony Xperia S	329.75 €	100	8	60	43.97 €
Samsung Galaxy S2 Plus	230.58 €	100	2	60	7.69 €
Mobile NFC Tag Testing Set	49.59 €	100	8	60	6.61 €
Total					166.38 €

**Table 7.3:** Equipment budget

## 7.2 Project Budget

Once all the tasks have been defined and the project duration is well known, all the costs for the project can be calculated. To do so, the template eased by the Universidad Carlos III de Madrid has been followed.

The attributable cost for perishable goods has been calculated with the following formula:

$$amortization = \frac{number\ of\ months}{depreciation\ period} \cdot equipment\ cost \cdot use\ \% \quad (7.1)$$

Appart from the personal budget, all the elements listed in the rest of budgets (equipment, software, services) correspond to the resources earlier mentioned in 1.4.

---

<sup>1</sup>January-May Part time. June-September Full time.

CHAPTER 7. PLANNING AND BUDGET

Description	Cost (Euros)	Project Devoted Time (%)	Devoted Time (Months)	Depreciation Cost	Attributable Cost
Ubuntu 12.04-13.04 LTS	0 €	100	9	60	0 €
Java Platform (JDK) 7u40	0 €	100	8	60	0 €
Android SDK 4.0-4.3	0 €	100	8	60	0 €
Eclipse Helios 3.6.2	0 €	100	3	60	0 €
Android Studio IDE	0 €	100	5	60	0 €
GIMP Image Editor 2.8	0 €	100	2	60	0 €
Sublime Text 2	0 €	100	9	60	0 €
				Total	0 €

**Table 7.4:** Software budget

Description	Company	Cost (Euros)
Google Play Developer Account	Google Inc.	16.05 €
Cloud Platform as a Service	Heroku	0 €
Git Version Control Public Repository	GitHub Inc.	0 €
Total		16.05 €

**Table 7.5:** Other services budget

At last, a 20% of indirect costs accounting for other costs not taken into account in previous tables and hardly measurable (such as derivated costs of using the premises where the project has been developed, Internet connection, possible trips, etc.) is added.

## CHAPTER 7. PLANNING AND BUDGET

Description	Cost (Euros)
Personnel Costs	17,513.54 €
Equipment Costs	166.38 €
Software Costs	0 €
Other Costs	16.05 €
Indirect Costs	3,539.19 €
VAT Free Costs	21,235.16 €
<b>TOTAL</b>	<b>25,694.54 €</b>

**Table 7.6:** Total budget

Adding the value added tax, that in Spain is 21%, the total project budget amounts to *twenty five thousand six hundred ninety four euros with fifty four cents*.

# 8

## CONCLUSIONS

This chapter will bring to a close the thesis. Conclusions on the technologies used and some personal opinion about the project result will be summarized.

### 8.1 General Conclusions

Technology needs time to settle down. Standardization, development, market penetration and people awareness need to be achieved. NFC is a valuable technology, useful to the user, respectful with existing standards and can work together with already well-known technologies such as Wifi and Bluetooth. It is just a matter of time for NFC to become widespread and the reference for all short range contact-less communications. The expectation for coming years is that NFC will keep growing interest at the actual pace. Then, by the time it reaches all kind of mobile phones and interesting applications are ready to use it will finally become popular.

Regarding web services, the cloud is growing at an unimaginable speed, thanks to a solid and wide accepted architecture like REST and the proliferation of data-centers and host providers. This infrastructure allows developers to find solution for almost any issue, being always connected, with the collaboration of many actors that enhance this environment and profit from it offering their services.

With respect to the back-end development carried out, it has been quite impressive the huge shortcut that web application development frameworks provide thanks to the MVC abstraction, while making it very easy to be REST compliant.

Finally, the Android ecosystem is still at its peak with millions of downloads in Google Play every day. Android sustainability strongly depends on developers, so they usually tend to make things easier for them. A very mature platform has been found with lots of tools and documentation, both official and from third parties.

## 8.2 Personal Conclusions

Personally, I have to admit that I am very satisfied with this period that is about to end. Not only for the result, but for the experience I have acquired during these months. As they say, the journey is the destination.

I have learned to program in Android at a good level, built up a back-end by myself, integrate as many libraries and services as I have proposed to and, in general, demonstrate to myself how capable I am of overcoming problems that can arise during a project development being able to find a compromise. In that sense I consider I have achieved most of the goals one day I established.

However there is always a margin for improvement. For example, I could have been more organized. But what I am, somehow, disappointed of is to have developed an application which purpose is a bit complicated to understand and to use if you do not have any previous knowledge of what NFC is. A bittersweet taste comes to my mouth when I think of the slow evolution NFC has had since I began studying it. Moreover, I realized, from the insignificant amount of downloads I have gotten so far, how important is to have an strategy defined when an application is published.

In any case the overall feeling about my final year project, as I said at the beginning, is very positive.



# A

## Compatible Devices

- Acer
  - CloudMobile S500-a9
  - E330-C7
  - Anydata
  - Philips W336-Crane
- Asus
  - PadFone Infinity-ASUS-A80
  - PadFone Infinity-A80
  - PadFone 2-A68
- Coolpad
  - CP9970-9970
  - Vodafone Smart 4G-cp8860u
  - STARADDICT III-cp8861u
- Foxconn International Holdings Limited
  - IN810-VKY
- Fujitsu
  - ARROWS A SoftBank 201F-SBM201F
  - ARROWS NX F-06E-F06E
  - ARROWS V F-04E-F04E
  - ARROWS X F-02E-F02E
  - ARROWS Kiss F-03E-F03E
- STYLISTIC M702-M702
- ARROWS S EM01F-EM01F
- ARROWS A SoftBank 202F-SBM202F
- ARROWS Tab Wi-Fi FAR70B-FAR70B
- Disney Mobile on docomo F-07E-F07E
- ARROWS Tab F-05E-F05E
- Fujitsu Toshiba Mobile Communications Limited
  - ARROWS ef FJL21-FJL21
- Google
  - Nexus S-crespo4g
  - Nexus S-crespo
  - Nexus 7-deb
  - Nexus 7-grouper
  - Nexus 4-mako
  - Galaxy Nexus-toro
  - Nexus 7-tilapia
  - Nexus 10-manta
  - Nexus 7-flo
- HTC
  - HTC Desire 600c dual sim-cp3dcg
  - HTC One-m7cdwg
  - HTC One SV-k2plccl
  - HTC Desire 500-z4u
- HTC EVA UTL-evitautl
- HTC One X-endeavoru
- KDDI Infobar A02-imnj
- HTC One-m7cdtu
- Butterfly s-dlxpul
- HTC Amaze 4G-ruby
- HTC 608t-cp3dtg
- HTC One SV-k2u
- HTC J Butterfly-dlxj
- HTC One-m7wls
- HTC first-mystul
- HTC Desire 500 dual sim-z4dug
- HTC PO091-csndug
- Desire 601-zara
- HTC One SV-k2ul
- HTC EVO 4G LTE-jewel
- HTC One XL-evita
- Droid DNA-dlx
- AT&T HTC One X+-evitareul
- HTC J One-m7wlj
- HTC Butterfly-dlxu
- HTC One-m7wlv
- HTC One-m7
- HTC Desire 600-cp3dug
- HTC One SV-k2cl
- HTC One X+-enrc2b
- ADR6410LRA-fireball
- HTC One-m7cdug
- HTC One VX-totemc2

## APPENDIX A. COMPATIBLE DEVICES

- Hisense
    - M470BSE-m470bse
    - M470BSA-m470
    - M470BSS-m470bss
    - M470BSD-m470bsd
  - Huawei
    - HUAWEI U8666N-hwu8666n
    - HUAWEI U8950N-51-hwu8950N-51
    - HUAWEI U8950N-1-hwu8950N-1
    - TURKCELL MaxiPRO5-hwu8860
    - P2-hwp2-6070
    - HUAWEI U8815N-hwu8815n
    - G526-hwG526-L11
    - HUAWEI P2-6011-hwp2-6011
    - HUAWEI T8950N-hwt8950N
  - Intel
    - Xolo X900-blackbay
    - AZ210A-noonhill
    - Orange San Diego-AZ210A
  - KT Tech
    - KM-S220-s220
    - KM-S300-s300
  - Kyocera Corporation
    - Torque-E6710
    - URBANO L01-KYY21
    - KYL21-KYL21
    - Hydro Elite-C6750
  - LGE
    - Optimus 4X HD-x3
    - Optimus LTE-11a
    - Optimus Vu2-vu2sk
    - Optimus L7-u0
    - LG Optimus L5 II-vee5nfc
    - Optimus Vu-batman lgu
    - LG Optimus L9II-l9ii
    - LG MachÃ©Ã©Ã©Ã©-l2s
    - LG Optimus G-geehrc4g
    - Optimus G Pro-geefhd
    - LG Optimus F7-fx1
  - LG Optimus LTE Tag-cayman
  - Optimus G Pro-geefhd4g
  - LG G2-g2
  - Optimus Vu2-vu2kt
  - LG optimus LTE2-d1lkt
  - LG optimus it-L05E
  - Optimus Vu-batman
  - LG optimus LTE2-d1lsk
  - Optimus 3D Cube-cx2
  - Optimus LTE-i skt
  - LG Optimus LTE3-fx1sk
  - Optimus Vu2-vu2u
  - Intuition-batman vzw
  - LG-P875h-11e
  - Optimus LTE-i u
  - Optimus Vu-vu10
  - Optimus Vu-batman skt
  - LG optimus LTE2-d1lu
  - LG Optimus G-geeb
  - Spectrum 2-d1lv
  - Optimus G Pro-geevl04e
  - LG Optimus G-geehrc
  - Optimus L9-u2
  - Optimus GK-gvfhd
  - Optimus G-geehdc
- Lenovo Mobile
  - Lenovo K800-K800
- Motorola
  - DROID RAZR HD-vanquish
  - RAZR D3-hawk40 umts
  - XT905-scorpion mini u
  - Droid Ultra-obake
  - XT897-asanti c
  - DROID RAZR i-smi
  - Droid MAXX-obake-maxx
  - DROID RAZR HD-vanquish u
  - XT901-solstice
  - DROID RAZR M-scorpion mini
  - Droid Mini-obakem
  - Moto X-ghost
- NEC
  - G'zOne CA-201L-CA201L
  - NE-201-NE-201
- CASIO G'zOne Com-mando 4G LTE-C811
- MEDIAS X N-06E-N-06E
- Oppo
    - X909-FIND5
  - Panasonic Corporation
    - JT-B1-B1
  - Panasonic Mobile Communica-tions
    - EB-4063-X-EB-4063-X
    - ELUGA P-P-03E
    - ELUGA X-P-02E
    - ELUGA-pana2 4o
  - PantechÃġ
    - IM-A870K-ef52k
    - IM-A830KE-ef45kv
    - IM-A770K-ef34k
    - PTL21-maruko
    - IM-840SP-IM-A840SP
    - IM-A830L-ef46l
    - IM-A850K-ef49k
    - IM-A860L-ef51l
    - IM-A860K-ef51k
    - IM-A810K-ef40k
    - Vega LTE M-ef65l
    - P9090-magnus
    - IM-A830S-ef47s
    - IM-A850S-ef48s
    - IM-A860S-ef51s
    - ADR930L-ADR930L
    - IM-A880S-EF56S
    - IM-A800S-ef39s
    - IM-A840S-IM-A840S
    - AT1-at1
    - IM-A810S-ef40s
    - IM-A760S-ef33s
    - IM-A850L-ef50l
    - IM-A870S-ef52s
    - IM-A830K-ef45k
    - IM-A870L-ef52l
    - IM-A775C-ef34c
  - Philips Electronics
    - PI3900-T7p Duo 93
  - SHARP
    - AQUOS PAD SHT21-SHT21
    - SH-06E-SH-06E
    - AQUOS PHONE SERIE SHL22-SHL22

## APPENDIX A. COMPATIBLE DEVICES

- AQUOS PAD SH-08E-SH-08E
  - PANTONE 6 SoftBank 200SH-SBM200SH
  - SoftBank AQUOS PHONE Xx 203SH-SBM203SH
  - Disney Mobile DM014SH-DM014SH
  - AQUOS PHONE EX SH-04E-SH04E
  - AQUOS PHONE si SH-07E-SH-07E
  - AQUOS PHONE Xx 206SH-SBM206SH
  - AQUOS PHONE ZETA SH-02E-SH02E
  - AQUOS PHONE SERIE SHL21-SHL21
  - AQUOS PHONE SERIE ISW16SH-SHI16
- Samsung
    - Galaxy Mega-melius3g
    - Galaxy S Advance-GT-I9070P
    - Galaxy Note II-t03gctc
    - Galaxy S4-jfteMetroPCS
    - Galaxy Fame-nevisp
    - Galaxy S III-d2can
    - Galaxy Note-SGH-I717M
    - Galaxy Note-SGH-I717R
    - Galaxy S III-d2vmu
    - Galaxy S III Mini-golden
    - Galaxy Pop-superiorlteskt
    - Galaxy S III-d2att
    - Galaxy Note II -t0ltevzw
    - Galaxy Mega-meliuslteatt
    - Galaxy Grand-baffinltekt
    - Galaxy Mega-meliusltecan
    - GT-I9260-superiorchn
    - Galaxy S III-d2vzw
    - Galaxy S4 mini-serranolte
    - Galaxy Express -expressatt
    - Galaxy S III-d2cri
    - Galaxy S 4-jftetmo
    - Galaxy S4 Zoom-mprojectqlte
    - Galaxy S3-d2xar
    - Galaxy S III-clatt
    - Galaxy Note II-t0ltecan
    - Galaxy S II-SGH-I757M
    - Galaxy S 4-jalteskt
    - Galaxy S II-SHW-M250K
    - Galaxy Grand-baffinvekt
    - Galaxy Note-SC-05D
    - Galaxy S 4-SC-04E
    - Galaxy S III-d2spi
    - Galaxy S II-SHV-E120S
    - Galaxy Note-SHV-E160L
    - Galaxy S 4-ja3g
    - Galaxy Note II-t0ltegt
    - Galaxy Exhilarate-SGH-I577
    - Galaxy S4 Zoom-mproject3g
    - Baffin-baffinltegt
    - Galaxy MEGA-meliuslte
    - Galaxy Note II-t03g
    - Galaxy S4 LTE A-ks01ltegt
    - Galaxy Note II-t0lte
    - Galaxy-aruba3gcmcc
    - Galaxy S4 Active-jactivelte
    - Galaxy S 4-jftevzw
    - Galaxy S III-d2tmo
    - Galaxy S4 Active-jactivelteatt
    - Galaxy S II-SHV-E120L
    - Galaxy Golden-ks02lteskt
    - Galaxy R-Style-jaguark
    - Galaxy S3-d2tfmvzw
    - Galaxy Note II-t03gcmcc
    - Galaxy S III-d2ltetmo
    - Galaxy S4 LTE A-ks01lteskt
    - Galaxy Victory-goghvmu
    - Galaxy Rugby-comanchecan
    - Galaxy S III -m0ctc
    - Galaxy S 4-jftecri
    - Galaxy S III-cllgt
    - Galaxy Note II -t0ltespr
    - GT-I9210T-GT-I9210T
    - Galaxy R-Style-jaguars
    - Galaxy S III-m0skt
    - Galaxy S II-SGH-T989
    - Galaxy Express-expresslte
    - Galaxy S 4 Google Play edition-jgedlte
    - Galaxy Tab 3 7.0 Wifi-t02wifilgt
    - Galaxy S II-ISW11SC
    - Galaxy S4-jfteaio
    - Galaxy Note 3-ha3g
    - Galaxy S 4-jfteusc
    - Galaxy S III-d2usc
    - Galaxy Rugby Pro-comancheatt
    - Galaxy S III-d2tfnspr
    - Galaxy Note II-t0ltekt
    - GT-I9100P-GT-I9100P
    - Galaxy S III-m0ctcduos
    - Galaxy Express-expresszigtteatt
    - Galaxy S II Plus -s2vep
    - Galaxy S II-GT-I9210
    - Galaxy S4 mini-serranoltekt
    - Galaxy S 4-jfteatt
    - Galaxy Ace II-GT-I8160P
    - Galaxy Golden-ks02ltekt
    - Galaxy Note-SGH-I717
    - Galaxy Grand-baffinltekt
    - Galaxy S III-m0
    - Samsung Stratosphere II-aegis2vzw
    - Galaxy S4 TD-LTE-jftdd
    - Galaxy Mega 6.3-meliuslteusc
    - Galaxy S III-m0cmcc
    - Galaxy S4 Zoom-mprojectlteatt
    - SHV-E110S-SHV-E110S
    - Galaxy S4 LTE A-ks01ltekt
    - Galaxy Premier-superior
    - Galaxy S III-d2mtr
    - Galaxy S Blaze-SGH-T769
    - Galaxy Victory-goghspr
    - Galaxy S II-SGH-I727R
    - Galaxy S 4-jaltektt
    - Galaxy S II-SHW-M250S
    - Galaxy Note-SHV-E160K
    - Galaxy Note II -t0ltetmo
    - Galaxy Nexus-toroplus
    - Galaxy S III-d2spr

## APPENDIX A. COMPATIBLE DEVICES

- SPH-L500-stunnerltespr
- Galaxy Note II-t0lteatt
- Galaxy Note II-t03gcduos
- Galaxy S4-jfftelra
- Galaxy S III-m0apt
- Galaxy S BlazeQ-apexqtmo
- Galaxy S 4 Duos-ja3gchnduos
- Galaxy Note II-t03gchn
- Galaxy Mega 6.3-meliusltselgt
- Galaxy S4-jfftecspr
- Galaxy Note II-t0lteskt
- Galaxy S 4-jffte
- Galaxy SII Skyrocket-SGH-I727
- Galaxy ACE 3-loganrelte
- Galaxy S III-clktt
- Galaxy Axiom -infiniteusc
- Galaxy Mega 6.3-meliusltektt
- Galaxy S II-SHV-E120K
- SCH-I425-godivaltevw
- Galaxy Nexus-maguro
- Galaxy Premier -superiorcmcc
- Galaxy S 4-jfftespr
- Galaxy S 4-jaltelgt
- Galaxy Note-SGH-T879
- Galaxy R-Style-jaguarl
- Galaxy S II-SC-03D
- Galaxy S III-clskt
- Galaxy Note-SHV-E160S
- Galaxy S II-SGH-T989D
- Galaxy Note II -t0lteusc
- Galaxy Mega 6.3-melius3gduosctc
- Galaxy Young-royssnfc
- Galaxy Note II-t03gchnduos
- Galaxy Mega-meliuslteskt
- Galaxy S4 mini-serranoltektt
- Galaxy S III LTE-m3
- Galaxy Note-SGH-I717D
- Galaxy S 4-jfftecan
- Galaxy S 4-ja3gduosctc
- Galaxy Grand-baffinveskt
- Galaxy S III-m0chn
- Galaxy S3 Mini-goldenteatt
- Galaxy S4 LTE-A-ks01lte
- Galaxy Note3-hltespr
- Galaxy Note 3-hlte
- Galaxy S III-d2lteMetroPCS
- Sony Ericsson
  - LT28h-LT28h
  - LT22i-LT22i
  - Xperia UL-SOL22
  - LT26i-LT26i
  - Xperia TX-LT29i
  - Xperia T-LT30p
  - Xperia Z-C6616
  - Xperia AX-SO-01E
  - LT28i-LT28i
  - Xperia V-LT25i
  - Xperia Z1-C6903
  - Xperia Z-C6606
  - Xperia A-SO-04E
  - Xperia M-C1905
  - Xperia Tablet Z-SGP311
  - Xperia sola-MT27i
  - Xperia acro S-LT26w
  - Xperia T-LT30a
  - Xperia SP-C5303
  - Xperia T-LT30at
  - Xperia Z -SO-02E
  - Xperia ion-LT28at
  - Xperia VL-SOL21
  - Xperia SL-LT26ii
  - Xperia Tablet Z-SO-03E
  - Xperia ZL-C6506
  - Xperia ZL-C6502
  - Xperia Z-C6602
  - Xperia Z-C6603
  - Xperia SP-M35c
- TCT Mobile Limited (Alcatel)
  - Vodafone 975N-SmartIII4
  - Vodafone Smart III (with NFC)-Vodafone 975N
  - Orange infinity 996-one touch 996 gsm
- Vertu
  - VERTU Ti-hermione
- Xiaomi
  - MI 2A-taurus
- ZTE
  - N9510-warplte
  - Turkcell Maxi Plus 5-nice
  - N9810-quantum
  - N9100-hayes
  - N9500-gordon
  - NX501-NX501
  - STARXTREM-prindle
- iRIVER
  - W1011A-w1011a
- Desconocido
- Buddy Connect
- crane-a721
- nuclear-Polaroid705
- PMP5580C
- crane-a106
- e100
- crane-a702-Imobile
- jeldd
- smq u
- GT-I9300
- generic
- SGP312
- GT-P3100
- endeavordd
- crane-a760
- nuclear-f727
- Nexus 7
- F-ONE
- HTC One X
- C2105
- C5502
- C5302
- A12
- GT-N7100
- crane-a702
- rk2906
- X909
- DROID RAZR
- crane-m799
- NX40X
- kai
- hwG510-0100
- nuclear-a7028
- TOUCHPAD 9
- TouchPad 7
- nuclear-a702UNIDEN
- crane-hn
- C2104
- nuclear-kf012
- crane-a1001Polaroid
- stvmx
- C5503
- WEXLER
- C5306
- s330
- C1904
- SGP321
- FunTab Pro
- wing-s738
- f6
- C6503
- ADM8000KP A
- C6802
- goghcri
- nuclear-f900
- crane-a702jh-Starmobil

# B

## User Manual

### B.1 Installation and Launch

UniteNFC can be downloaded from <https://play.google.com/store/apps/details?id=com.quantum.unitenfc> or searching for *unitenfc* in Google Play search bar.



Figure B.1: Installation and Launch

Once installed, every-time the application is launched an splash screen showing the application name and logo will appear for a few milliseconds.

## B.2 Login

When the application starts a login screen will appear. If it is the first time using UniteNFC and you do not have an account on topooos platform a registration form is available from the login screen. After successful login UniteNFC will ask you to connect with Facebook. Pressing on Facebook button will synchronize your Facebook data with UniteNFC. Eventually, Facebook will ask you to grant permissions to UniteNFC when necessary. You can dismiss this dialog and choose to not show again, otherwise it will remind you to connect with Facebook every-time you launch UniteNFC.



Figure B.2: Login

## B.3 Main Screen

UniteNFC's main screen is divided at the same time in three different screens one for each key functionality of the application that can be accessed from the

tabs on the top. From left to right: map screen, NFC Points history screen and social screen. In the map the NFC Points can be viewed and explored. The NFC Points tab will show a list of registered NFC Points and another one with visited. Besides, in the bottom there is a button to register new NFC Points. The last one, the social tab consists of a list of friends showing their name and picture. Detailed information can be obtained clicking in them. Also, in the bottom the button to add a new friend is available.

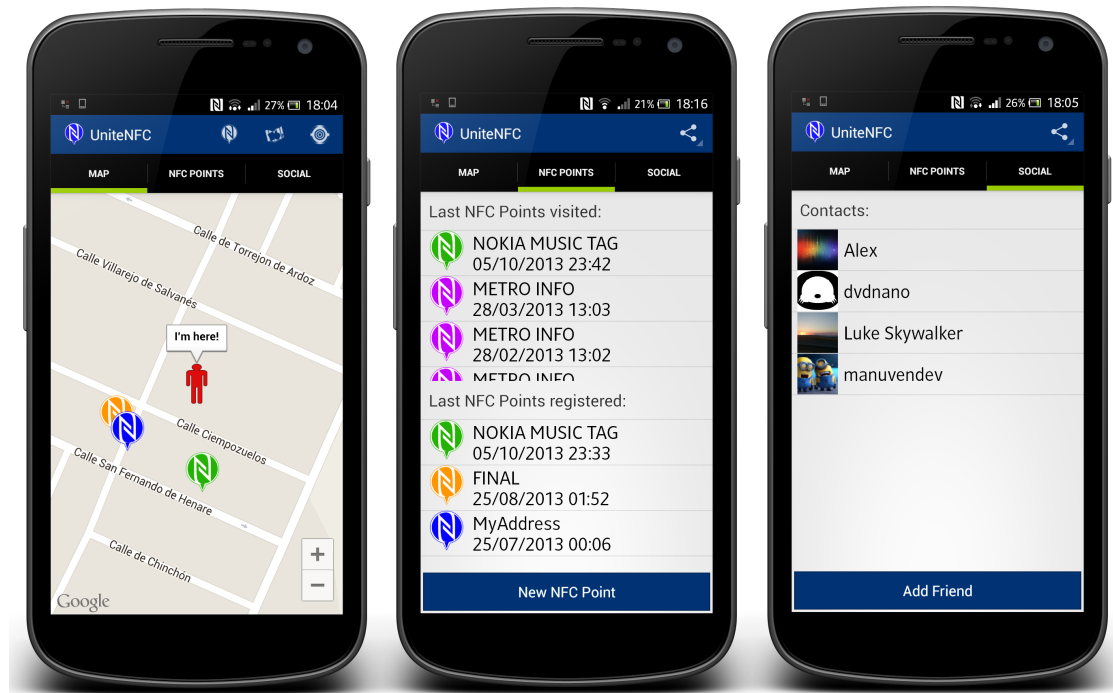


Figure B.3: Main screen

## B.4 Map Screen

NFC Points can be discovered on the map, where the ones around you will be represented and can be selected to obtain information about them. Three actions are available on the right part of the top bar. Map centering, map type switching between standard and satellite view and NFC Points filtering by type to show only relevant items to you.

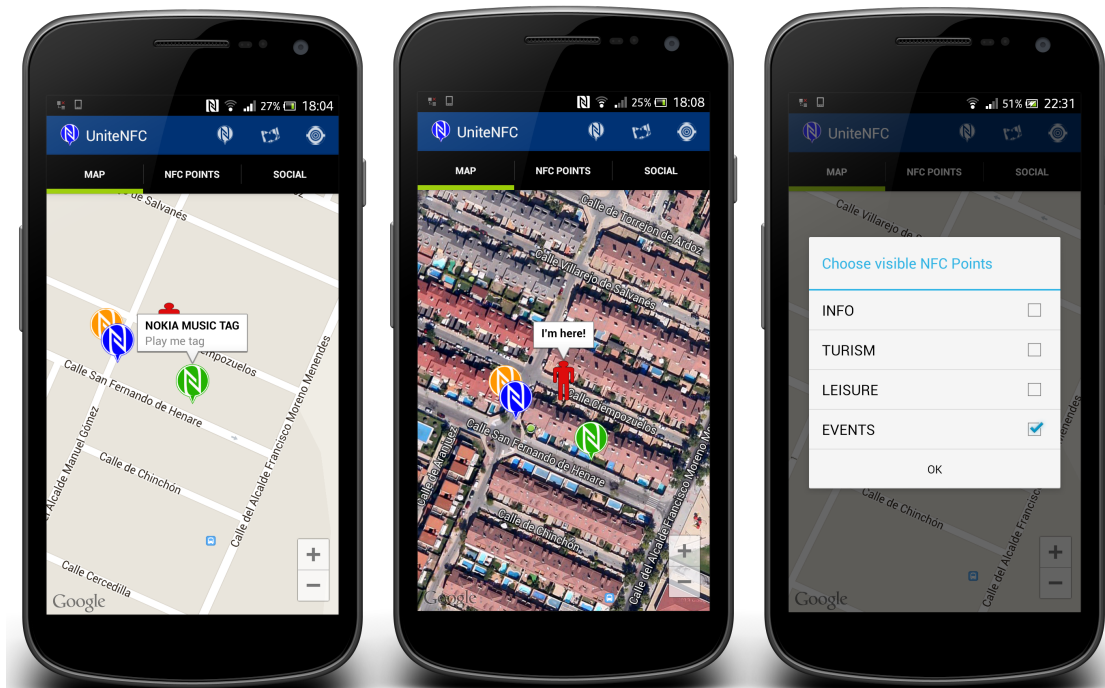


Figure B.4: Map screen

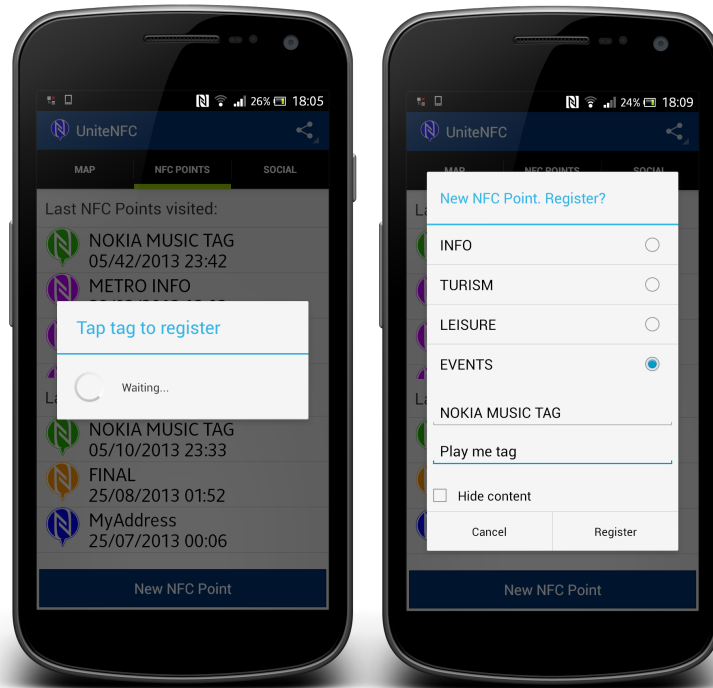
## B.5 NFC Points Screen

Apart from showing visited and registered NFC Point records, new NFC Points can be registered from this view. For that NFC must be enabled. Pressing the button a dialog will appear asking you to scan a tag to register. If the NFC Point is not in the system yet a form will then appear asking you to fill up a name and description, set the NFC Point type and choose whether you like to show the tag content or make it private.

## B.6 Social Screen

From the friend list in the social tab, individualized information can be accessed. This view is called User Card, in which you can find the NFC Points visited by your friends. If a new friend is going to be added, your own User Card will be shown and then it is time to use Android Beam to interchange your user data.





**Figure B.5:** NFC Points screen

## B.7 Wall Screen

Every NFC Point will have an associated wall. Selecting the NFC Point from the map or from any history list the wall will be shown. It contains some relevant data like name, description, content address and last seen date. Furthermore, there is an space to rate the NFC Point and see the mean rating and another one to leave a comment on the wall and see other users' comments. If the user is viewing the wall happens to be the one that registered that NFC Point and (*Admin*) clause will appear, meaning that you can delete any comment and hide or show the NFC Point content whenever you want.

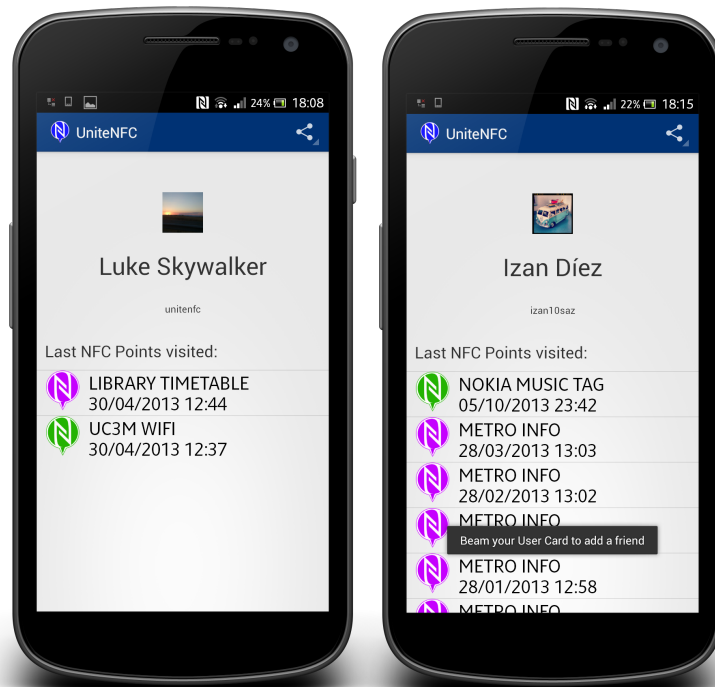


Figure B.6: Social screen

## B.8 NFC Points Reader

Unite NFC will read the NFC Points you may find and record them as a visit if they have been registered previously. The content will be shown and will be click-able if any action is associated to them. A button in the bottom will show the associated wall if the NFC Point is registered.

## B.9 Notifications

UniteNFC will alert you of NFC Points near you in the Android notifications bar. Clicking the notification you will be directed to the point of the map where the NFC Point is in UniteNFC.

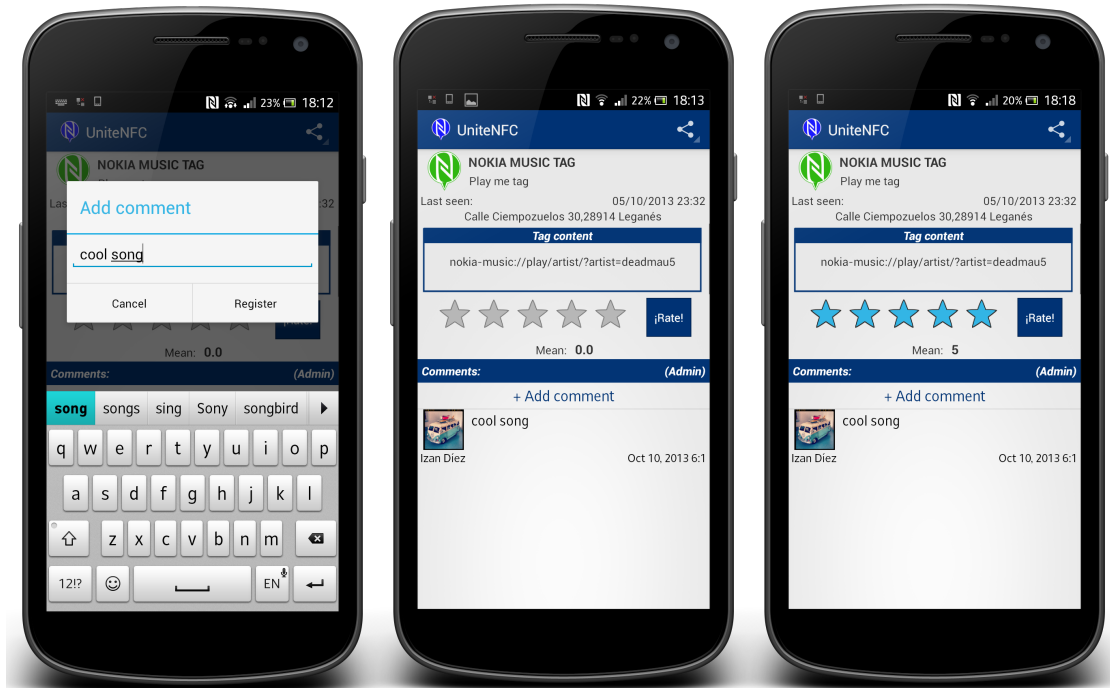


Figure B.7: Wall screen

## B.10 Settings

Some characteristics in UniteNFC can be modified. Pushing menu button in your device and *Settings* an screen showing all the editable fields will appear. They are divided in two: user account and map settings. In the former user name, profile picture and connection with Facebook can be modified. Regarding the map settings the radius of NFC Points discover-ability can be set from 10 meter (for high density areas) to 100 kilometers (for low density areas). You can also choose to track your position so that the map is moving with you, much like a GPS. Last option allows you to enable or disable the notifications.

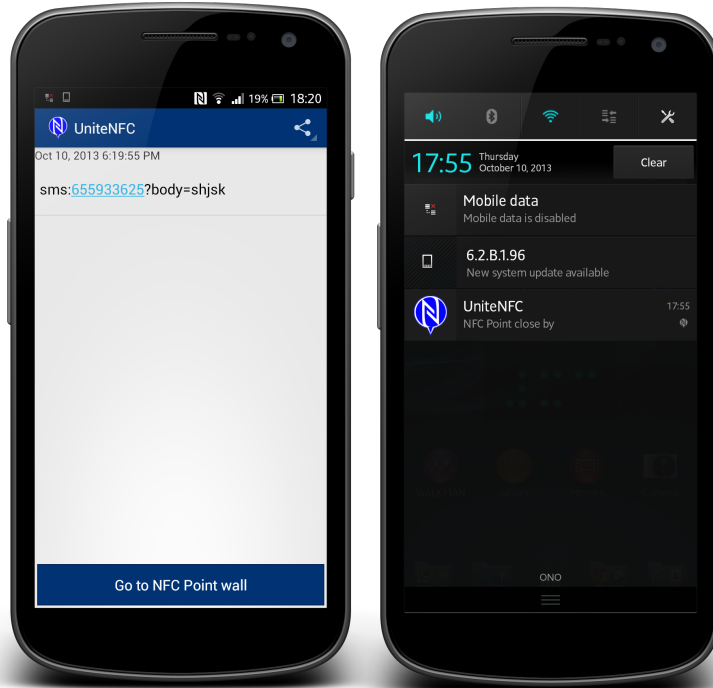


Figure B.8: Reader screen and notifications

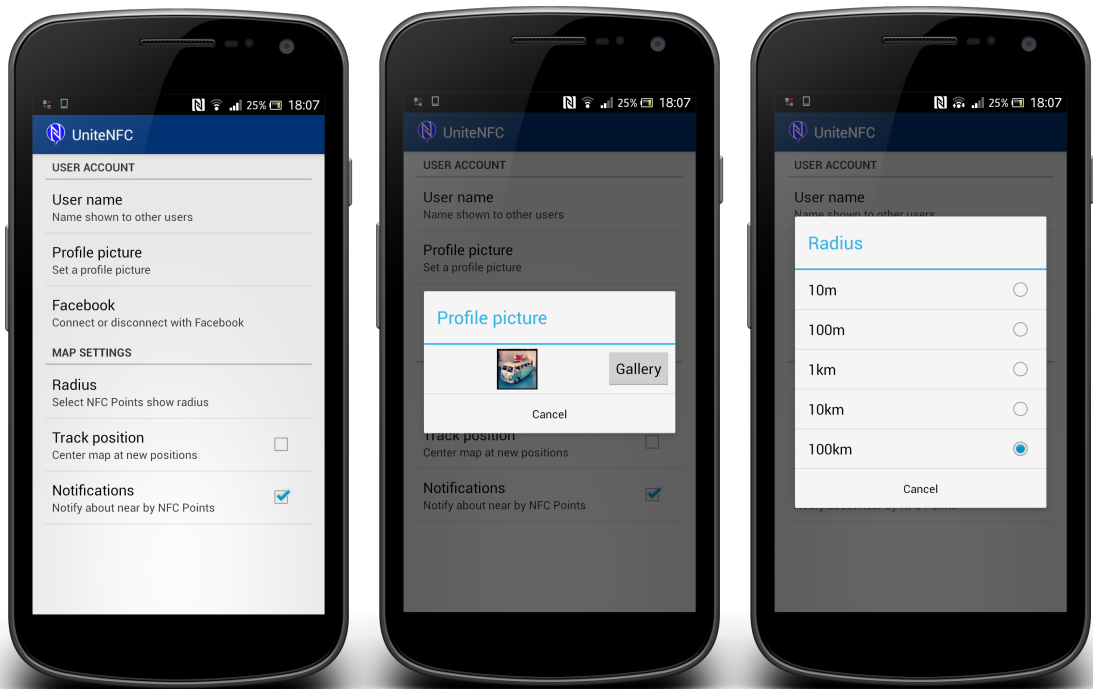


Figure B.9: Settings

# C

## Application Code

### C.1 UniteNFC Client-side Android Code

<https://github.com/idiez/UniteNFC>

### C.2 UniteNFC Back-End Code

<https://github.com/idiez/unfcbe>

# Bibliography

- [1] Google Maps Android API v2 [Last access: apr. 2013]:  
[https://developers.google.com/maps/documentation/android/start#installing\\_the\\_google\\_maps\\_android\\_v2\\_api](https://developers.google.com/maps/documentation/android/start#installing_the_google_maps_android_v2_api)
- [2] Creating context-aware applications has never been easier [Last access: aug. 2013]:  
<http://www.topoos.com/home.aspx>
- [3] The Web framework for perfectionists with deadlines | Django [Last access: sep. 2013]:  
<https://www.djangoproject.com/>
- [4] Heroku | Cloud Application Platform [Last access: sep. 2013]:  
<https://www.heroku.com/>
- [5] Facebook SDK for Android [Last access: sep. 2013]:  
<https://developers.facebook.com/docs/android/>
- [6] Android Native Application Tracking Overview [Last access: aug. 2013]:  
<https://developers.google.com/analytics/devguides/collection/android/>
- [7] Mifare Smart Cards ICs [Last access: mar. 2013]:  
[http://www.nxp.com/products/identification\\_and\\_security/smart\\_card\\_ics/mifare\\_smart\\_card\\_ics/](http://www.nxp.com/products/identification_and_security/smart_card_ics/mifare_smart_card_ics/)
- [8] Which NFC Tag is Right for my Project [Last access: mar. 2013]:  
<http://www.nfctags.com/nfc-applications-which-tag>

## BIBLIOGRAPHY

- [9] Everything You Need to Know About Near Field Communication [Last access: mar. 2013]:  
<http://www.popsoci.com/gadgets/article/2011-02/near-field-communication-helping-your-smartphone-replace-your-wallet-2010/>
- [10] Introduction to NFC, Nokia Developers, 8 July 2011. Version 1.1. [Last access: apr. 2013]:  
[http://www.developer.nokia.com/dp?uri=http%3A%2F%2Fsw.nokia.com%2Fid%2Fbdaa4a0f-fcf3-4a4b-b800-c664387d6894%2FIntroduction\\_to\\_NFC](http://www.developer.nokia.com/dp?uri=http%3A%2F%2Fsw.nokia.com%2Fid%2Fbdaa4a0f-fcf3-4a4b-b800-c664387d6894%2FIntroduction_to_NFC)
- [11] NFC Forum Web-Site [Last access: jun. 2013] :  
<http://www.nfc-forum.org>
- [12] NFC Use-cases [Last access: jun. 2013]:  
<http://www.nfctags.com/nfc-usecases>
- [13] Peter Preuss, NFC Forum Marketing Committee Chairman, “NFC Forum and NFC Use Cases” [Last access: mar. 2013]:  
[http://www.nfc-forum.org/events/oulu\\_spotlight/Forum\\_and\\_Use\\_Cases.pdf](http://www.nfc-forum.org/events/oulu_spotlight/Forum_and_Use_Cases.pdf)
- [14] MasterCard Press Releases. Google, Citi, MasterCard, First Data and Sprint Team Up to Make Your Phone Your Wallet, May 26, 2011 [Last access: mar. 2013]:  
<http://newsroom.mastercard.com/press-releases/google-citi-mastercard-first-data-and-sprint-team-up-to-make-your-phone-your-wallet/>
- [15] NFC phones The definitive list [Last access: sep. 2013]:  
<http://www.nfcworld.com/nfc-phones-list/>
- [16] Near Field Communication versus Bluetooth [Last access: mar. 2013] :  
<http://www.nearfieldcommunication.org/bluetooth.html>
- [17] Patauner, C, “EuraSIP”: High Speed RFID/NFC at the Frequency of 13.56 MHz.
- [18] Charles A. Walton “Portable radio frequency emitting identifier” U.S. Patent 4,384,288 issue date May 17, 1983.

## BIBLIOGRAPHY

- [19] Nokia, Philips And Sony Establish The Near Field Communication (NFC) Forum [Last access: mar. 2013]:  
[http://www.nfc-forum.org/news/pr/view?item\\_key=d8968a33b4812e2509e5b74247d1366dc8ef91d8](http://www.nfc-forum.org/news/pr/view?item_key=d8968a33b4812e2509e5b74247d1366dc8ef91d8)
- [20] Google unveils first Android NFC phone [Last access: mar. 2013]:  
<http://www.nfcworld.com/2010/12/07/35385/google-unveils-first-android-nfc-phone-but-nexus-s-is-limited-to-tag-reading-only-for-now/>
- [21] Nick Pelly and Jeff Hamilton. How to NFC [Last access: jun. 2013]:  
<http://www.google.com/events/io/2011/sessions/how-to-nfc.html>
- [22] MasterCard Certifies Two BlackBerrys to Run PayPass on SIMs [Last access: mar. 2013]:  
<http://nfctimes.com/news/mastercard-certifies-two-blackberrys-run-paypass-payment-sims>
- [23] Launching Google Wallet on Sprint and working with Visa, American Express and Discover [Last access: mar. 2013]:  
<http://googleblog.blogspot.com.es/2011/09/launching-google-wallet-on-sprint-and.html>
- [24] Sony's SmartTags could change phone habits [Last access: mar. 2013]:  
[http://news.cnet.com/8301-17938\\_105-57359901-1/sonys-smarttags-could-change-phone-habits/](http://news.cnet.com/8301-17938_105-57359901-1/sonys-smarttags-could-change-phone-habits/)
- [25] Proximity for Windows Phone 8 [Last access: mar. 2013]:  
[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207060\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207060(v=vs.105).aspx)
- [26] NFC Forum Launches Special Interest Groups to Support NFC Market Implementations [Last access: mar. 2013]:  
[http://www.nfc-forum.org/news/pr/view?item\\_key=31fbda01baded31092b2835e952ccc6b6ee9c47d](http://www.nfc-forum.org/news/pr/view?item_key=31fbda01baded31092b2835e952ccc6b6ee9c47d)
- [27] NFC Specification Download [Last access: mar. 2013]:  
[http://www.nfc-forum.org/specs/spec\\_license](http://www.nfc-forum.org/specs/spec_license)
- [28] NFC Data Exchange Format (NDEF) Technical Specification NFC Forum<sup>TM</sup> NDEF 1.0.



- [29] NFC Specification List [Last access: jun. 2013]:  
[http://www.nfc-forum.org/specs/spec\\_list/](http://www.nfc-forum.org/specs/spec_list/)
- [30] Ernst Haselsteiner and Klemens Breitfuss. Security in near Field communication (NFC), Philips Semiconductors. In Workshop on RFID Security RFIDSec 06, jul 2006.
- [31] Gauthier Van Damme and Karel Wouters. Practical Experiences with NFC Security on mobile Phones, Katholieke Universiteit Leuven Dept. Electrical Engineering, jun 2009.
- [32] Gerhard Hancke. A Practical Relay Attack on ISO 14443 Proximity Cards, University of Cambridge, Computer Laboratory.
- [33] Koichi Tagawa, NFC Forum Chairman (SONY). NFC: The Evolution Continues [Last access: mar. 2013]:  
[http://www.nfc-forum.org/resources/presentations/NFC\\_The\\_Evolution\\_Continues\\_WIMA\\_2011.pdf](http://www.nfc-forum.org/resources/presentations/NFC_The_Evolution_Continues_WIMA_2011.pdf)
- [34] Token Based Authentication - Implementation Demonstration [Last access: sep. 2013]:  
[http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token\\_based\\_authentication/](http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/)
- [35] Inside Secure, Open NFC [Last access: sep. 2013]:  
<http://open-nfc.org/wp/>
- [36] The Oxford English Dictionary (OED), Second Edition.
- [37] Software Testing, Wikipedia [Last access: sep. 2013]:  
[http://en.wikipedia.org/wiki/Software\\_testing#Testing\\_levels](http://en.wikipedia.org/wiki/Software_testing#Testing_levels)
- [38] REST Console [Last access: sep. 2013]:  
<http://restconsole.com>
- [39] Android Asset Studio [Last access: oct. 2013]:  
<http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html>
- [40] Gamma, Erich and Helm, Richard and Johnson, Ralph and Vlissides, John. Design patterns: elements of reusable object-oriented software, 1995 Addison-Wesley Longman Publishing Co., Inc.

## BIBLIOGRAPHY

- [41] Lauren Darcey, Shane conder. Programación Android 4, 2012 Ediciones Anaya Multimedia S.A.
- [42] Android Developers, Location Strategies [Last access: apr. 2013]:  
<http://developer.android.com/guide/topics/location/strategies.html>
- [43] Android Developers, NFC Basis [Last access: aug. 2013]:  
<http://developer.android.com/guide/topics/connectivity/nfc/nfc.html>
- [44] Android Developers, Signing Your Applications [Last access: sep. 2013]:  
<http://developer.android.com/tools/publishing/app-signing.html>
- [45] Android Fragmentation 2013, Open Signal, jul. 2013 [Last access: oct. 2013]:  
<http://opensignal.com/reports/fragmentation-2013/fragmentation-2013.pdf>