INFORMATION SYSTEMS, WEB AND PERVASIVE COMPUTING SERIES

INTELLECTUAL TECHNOLOGIES SET

Volume 3

The Art and Science of NFC Programming

Anne-Marie Lesas
Serge Miranda

ISTE WILEY

The Art and Science of NFC Programming

Intellectual Technologies Set

coordinated by
Jean-Max Noyer and Maryse Carmes

Volume 3

The Art and Science of NFC Programming

Anne-Marie Lesas
Serge Miranda

ISTE

WILEY

First published 2017 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Control Number: 2016954190

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-78630-057-7

Contents

Foreword vii

Preface xi

Introduction xv

Chapter 1. State-of-the-Art of NFC 1

1.1. Future mobiquitous digital services 2
1.1.1. The era of mobiquity 3
1.1.2. Toward a world of contactless communicating objects 6
1.2. NFC equipment 7
1.2.1. NFC tag 7
1.2.2. NFC smart card. 8
1.2.3. NFC smartphone 13
1.2.4. Reader/encoder: NFC transaction terminals 14
1.2.5. “Smart cities” and sustainable development. 14
1.2.6. Cashless payment with NFC 15
1.3. NFC standards. 16
1.3.1. Analog signal and NFC digital transposition 18
1.3.2. The three standardized modes of NFC 21
1.3.3. NFC forum standards 25
1.3.4. GlobalPlatform (GP) 36
1.3.5. SIMAlliance and open mobile API 42

Chapter 2. Developing NFC Applications with Android 45

2.1. Introduction to Android programming using Eclipse 46
2.1.1. Android in a nutshell 46
2.1.2. Android in Eclipse IDE 49
2.1.3. Intents and Android context 60
2.1.4. The Activity class of Android 61
2.1.5. Android graphical interface: “layout” files 64
2.1.6. Compiling and testing an Android application 67
2.2. Implementing NFC with Android 70
2.2.1. Android manifest declarations 71
2.2.2. Implementing the NFC reader/writer mode 71
2.2.3. Implementing the NFC P2P mode with Android 83
2.2.4. Implementing the NFC card emulation mode with Android 87
2.2.5. Developing NFC services with Android HCE 97

Chapter 3. NFC Use Cases 107

3.1. Usage of the NFC reader/writer mode 107
3.1.1. Use case: management of equipment loans 108
3.2. Usage of the NFC P2P mode 112
3.2.1. Use case: NFC pairing 112
3.3. Usage of NFC card emulation mode 114
3.3.1. Use case: digital wallet in the SE 115
3.4. Usage of the HCE mode 118
3.4.1. Use case: SE in the Cloud with HCE 119

Conclusion 121

Bibliography 125

Index 129

Foreword

"The main rule is to please and touch.

All others are made only to achieve this first one".

MOLIERE

Even if the NFC standard is young (developed in 2004), I have been asked to write this book for several years now, due to the pioneering role played in France (and in Europe) by our Computer Science Master's degree MBDS (www.mbds-fr.org) at the University of Nice – Sophia-Antipolis around the prototyping of innovative services using this standard. MBDS prototyped NFC services in all sectors of economic life: from tourism and culture in Nice, to social payment in India, via campuses in Haiti, museums, airports, hotels, connected houses in Morocco and electric cars in Sophia Antipolis. In 2009, the city of Nice was the first for the deployment of NFC standard in Europe, because of the MBDS innovation research lab.

After half a dozen books published on databases, I was not really eager to write a new book. My first collection of books, following Knuth's book (The Art of Computer Programming), my bedside reading as a fellowship student in California, was entitled "The Art of Databases".

What changed my mind was the enthusiasm of Anne-Marie Lesas, who was working on her PhD on NFC secure services with our industrial partner Gemalto under a CIFRE convention[1] from the ANRT[2] and of an IFCPAR contract[3] (www.cefipra.org) on NFC virtual social currency in India with TATA Consultancy Services (CS) and Bangalore University, as well as a scientific expertise on NFC patent infringement in the United States in 2015. As a brilliant former MBDS student after a professional career, Anne-Marie first showed passion for mobiquitous new technologies (NFC cars, and means of detecting earthquakes by using smartphone sensors). What a delight for a professor, who is nothing but a dream purveyor, to see a student take over.

The title of this book implies the duality of "Art" and "Science", which are the two approaches to perception and understanding of the world; in prefaces to database books, I would write: "the word *art* refers to a way of investigation, recreation and interpretation of the real world in opposition to the *science* which bears an abstract interpretation, based on formal concepts, models and tools". Creativity on NFC applications is unlimited with mobiquitous usages that reinvent the real world by creating new bridges toward the virtual world; these applications are based on strict standardized concepts that we explain, along with their implementation methods. Dealing with this duality is the double purpose of this book.

In this way, this book is the result of a pedagogical encounter between a professor and a researcher in order to allow other IT developers to contribute to changing the world by touching it! I would thus like to thank Anne-Marie for her professional and human skills and, through her, all students who, by their enthusiasm, lead me toward a process of never-ending, spiral innovation. Creativity on content and services is a beautiful spiralist adventure that brings life

1 Industrial contracts for training through research.
2 The French *Association Nationale de la Recherche et de la Technologie*.
3 Indo-French Centre for the Promotion of Advanced Research.

(special thanks to Franketienne for this beautiful concept of *spiralism*, which we shared and discussed in Port au Prince).

I would like to invite readers to dream about their life, to have big dreams while keeping in mind that new technologies must first and foremost serve the good of humankind and the improvement of shared environments, and of the lives of each of us. "*Always put man at the center*" and do not hesitate to be "a *nonconformist*, even an *innovation anarchist*" as shown by Pierre Laffite, the founder of Sophia-Antipolis Science park.

I also have a thought for my friend and colleague, pioneer of all types of databases (and Big Data), Mike Stonebraker, godfather of one of the first MBDS classes and winner of the Turing Award in 2014, who would always stress applied research in information systems, with the obsession of always trying to solve concrete problems and not to only stick to simple theoretical intellectual constructions disconnected from reality.

This book has a double purpose, which corresponds to its two main parts; it aims both toward:

– an exhaustive, theoretical approach of the NFC standard, unavoidable in the future of smartphones, as much in the informational as in the transactional world;

– a pragmatic and systematic approach of the development of NFC applications, based on numerous prototypes of innovative services created within our MBDS Master's program since the birth of the standard in 2004.

This book is for IT engineers (IT generalists as much as students in Bachelor's or Master's programs) who are passionate about new technologies and curious about the use of NFC, particularly in mobile applications; our goal is to explain the technical and functional specificities of the NFC standard through notions essential to the understanding of the ecosystem, its mobile implementation (with Android) and its main applications.

With this book, we hope to give the reader autonomy in order for him to design and develop his own NFC applications. *Innovation is invention meeting usage*: innovate with this invention by imagining new practices.

To conclude this foreword, I will borrow two quotes I often use to end my conferences:

– "Never forget that an ant can carry an elephant", from my friend Sister Flora, who has been managing an orphanage in the South of Haiti, in Ile-à-Vaches for more than 30 years, with admirable love and creativity. Today, someone with a smartphone in his hands has more computer power than the computer used for the Apollo mission, and an access to information greater than President Kennedy ever had. A smartphone is a "world object", as Michel Serres understood this word, and we must not forget that "even the thought of an ant can touch the sky" (Japanese proverb);

– "If you cannot change the world, try to change YOUR world" (Karl Marx's last phrase), which is the basis of a life of *communaction* in future tense and a reality which is not only mobiquitous.

Enjoy this book and enjoy NFC programming, I wish for you to become a *communactor* ant, open to the world to change it.

Professor Serge MIRANDA
2016

Preface

> "The primary form of sense is touch, which belongs to all animals. […]
>
> The sense of touch is necessarily the one whose loss causes the death of living beings".
>
> ARISTOTLE

With the *near field communication* (NFC) standard, an NFC-enabled mobile phone acquires Aristotle's sense of touch.

NFC is a global standard of contactless and very short field (proximity) communication (a few inches) created by Philips, Sony and Nokia in 2004 (three major players and leaders in consumer electronics). The NFC standard is one of the 16 *radiofrequency identification* standards bringing a unique identification to each tagged object known since the 1940s and a wireless reading (through radiofrequency). Today NFC, which is widespread in smart cards (for access, payment and transportation), has been universally chosen by all smartphone manufacturers since 2014, thus allowing new mobile phone uses.

This NFC standard has three operating modes: *reader/writer*, *card emulation* and *peer-to-peer*; with a simple touch on an NFC-enabled

device, a tap (hence the *tap'n play* paradigm) on a tag or on another NFC-enabled device, we can:

– collect information thanks to the NFC reader/writer mode;

– connect to another device and initiate connectivity (e.g. Bluetooth®, Wi-Fi, Li-Fi) thanks to the NFC peer-to-peer mode;

– authenticate, open a door or pay, for example, because of the emulation card mode.

An NFC-enabled mobile phone can thus be seen as a universal connector that increases the phone's sensory capacities. After speaking/listening, reading and viewing (pictures, text messages, e-posts in social networks), thanks to NFC, mobile phones will allow us to *touch* in order to validate an access, get information, exchange content or pay. This interaction mode, which is non-intrusive and intuitive, leads the way to a portfolio of innovative services.

"The NFC smartphone has won the battle of the pocket"! Anything which was in your pockets or your purse will now have a dematerialized version in your mobile: cash, debit and credit cards, loyalty program cards, keys, camera, MP3 player, etc. By the end of 2015, half of the planet (3.5 billion people) owned a smartphone, with a sustained deployment growth rate, all the more with Indian advertisements in the beginning of 2016. Half of these smartphones are NFC enabled. This also means that the 2 billion people with no bank account who own a smartphone will be able to benefit from financial services: new mobile payment actors will arise beyond the banks! *The banker is…in your pocket!*

In biology, life is defined as a pair: information and communication. Thanks to a simple touch, NFC-enabled phones introduce communication toward a remote server carrying the story of this object. Any object with an NFC tag touched by a NFC-enabled mobile phone thus becomes, biologically speaking, a living object. In the future, information systems will have to include this aspect of objects which become living objects.

NFC standard thus allows objects to become living objects, and the places where they are located become *smart places*: check the virtual user guide of a device, automatically setup an environment and personal preferences, launch a scheduled washing program or open the door of your house with your smartphone with a simple tap, among others, using NFC. Through a simple proximity gesture (less than 1 inch), NFC induces the user's desire to interact and create a link between the real world and the virtual world to make *augmented* or *diminished* reality. The use of NFC-enabled mobile phones is one of the supports allowing the accurate location of a user; in this way, we can envision a portfolio of geolocated, personalized and contextualized services.

NFC-enabled mobile phones are carried by a digital revolution that puts people at the center of the interaction between the real world and the virtual world: the owner of an NFC smartphone potentially becomes a *Homo mobiquitus*, *a communactor*, i.e. a data contributor (consumer and producer) to the common space in *bottom-up* mode [MIR 14a, MIR 14b].

We have reached a new era of information systems at the convergence of mobile phones (which have become computers with smartphones) and the ubiquity of the Internet (which has become social with broadband); we sum up, according to Xavier Dallos, with the portmanteau word *mobiquity*. All of the economic sector and the three individual spheres (public, private, professional) will be impacted by mobiquity.

With NFC, the following new multidisciplinary concepts have emerged: *mobiquity*, *Homo mobiquitus*, *communactors*, *one-tap marketing*, *mobiquitous tourism* (from the former "max min" to the future "mini max"), *mobiquitous currency* (cash is no longer the latest payment link), *spiralist innovation*, the "*Assistants Mobiquitaires InformationnelS* (AMIS)" (Mobiquitous Informational Assistants) or the "mobile cyber cafe" in Haïti.

NFC is a breakthrough innovation leading the way to new services and to new architectures of information systems that bring new

business models. *Spiralist innovation* is at the heart of NFC thinking. As creativity on usage is unlimited, it is up to us to innovate and create new applications with this global NFC standard acting as a universal connector between the real world and the virtual world in a non-intrusive way: “the sky is the limit!”

Professor Serge MIRANDA

October 2016

Introduction

Based on radiofrequency identification tested during the Second World War, near-field communication (NFC) standards became global in 2004 after several years of prototyping and development.

Beyond the fact that it is an open standard, NFC has many advantages; it is an economic technology enabling dynamic multimodel interaction which can be summarized by "3S": Security, Speed and Simplicity. Just by enabling your smartphone, you are able to interact with the real world and enrich it with all information from the virtual world.

Objects with an NFC tag become *communicating* and biologically alive: the simple gesture of a tap allows NFC-enabled smartphones to collect and exchange information, to pair two devices allowing them to communicate, to open a door or to pay for something.

This work is divided into three chapters, showing NFC as an essential link in the chain of the future's mobiquitous information systems:

– A theoretical chapter discussing state-of-the-art of NFC. We will show its technical characteristics as well as its operating modes.

– A technical chapter on NFC mobile programming with Android, addressing each aspect of NFC implementation.

– A practical chapter with concrete application scenarios, in which we provide examples of use cases based on MBDS prototyping from 2004, in the three modes of NFC standard.

1

State-of-the-Art of NFC

Since 2014, all smartphone manufacturers have been offering near-field communication (NFC) connectivity; NFC standards use electromagnetic properties of radiofrequency across very short distances of no more than a few inches.

"NFC" refers to several technologies using electromagnetic fields allowing data transfer between two peripheral devices set close to one another. Known since the Second World War, radiofrequency identification (RFID) is a contactless communication system using electromagnetic fields to send messages for identification and automated traceability purposes thanks to tags linked to objects. Tags contain electronically stored data. Some tags are powered through electromagnetic induction from magnetic fields created when brought into proximity with an RFID reader/encoder. Passive tags act as a passive transponder, powered by the electromagnetic radio waves sent by the peripheral device initiating communication (reader).

NFC technology could offer a general purpose *connection* to any other wireless communication system (Bluetooth®, Wi-Fi, GPRS, 4G, Li-Fi, etc.) and allow device pairing with a simple tap ("TAP and PLAY" paradigm). This chapter focuses on the ecosystem in which NFC standards are implemented, its background and its standards.

The Art and Science of NFC Programming, First Edition.
Anne-Marie Lesas and Serge Miranda.

1.1. Future mobiquitous digital services

Mobiquity is not a mere portmanteau word, suggested by Xavier Dalloz in the beginning of the 1990s, with Internet access in mobile phones (during the failure of WAP especially due to a lack of contents and services). Today, it has become a cross-concept between the real and the virtual world, full of new content and services, creating a convergence between MOBility of the cellular phone, which became a computer per se (Smartphone) and the ubiQUITY of the Internet, now "n.0", local (*Local Wide Web)* and marked by its extensive distribution, being in everybody's pockets today. This concept of mobiquity, which matches that of ATAWAD ("Any Time, AnyWhere, Any Device" or "Any Content"), is promising in terms of its innovation and multidisciplinary in research on content, services, architectures and methods.

A new ecosystem developed endogenously: information consumers (target customers) have become information providers – via websites and mobile apps (because of the Internet) for media sharing (pictures, videos on social networks), alternative press (e.g. Agoravox and Alterinfo), free encyclopedia (Wikipedia), open augmented reality (Wikitude), etc. – thus making professionals, who were once suppliers, consumers and information managers; new business models arise, generating a considerable load of raw information involving new real-time predictive processing (*big data*).

We have entered a new era in which usage arises from individual, associative or participative practice (social networks, *communaction*), and no longer from pre-existing models managed by lobbyists. An adaptive "bottom-up" approach tends to take over the traditional "top-down" approach of multimedia technologies market and digital and telecom services in any environment (business, tourism, transportation, healthcare, education, etc.).

NFC is a close contact-free communication mode (through "touch"), which extends connectivity and allows for the emergence of new usages while keeping the principle of prior consent

(recommended by the French Commission on Computer Sciences and Freedoms (*Commission Nationale de l'Informatique et des Libertés*)[1]), since the user is asked to confirm his/her will of interaction (without intrusion) and must make an explicit gesture.

Since the NFC-enabled smartphone has the capacity to read NFC tags but also to act as an NFC tag (or NFC smartcard), information can be distributed across tags, mobile phones or servers driving to very different business models around a strategic question: who will control the information coming from the user's interactions with his/her mobile phone? Mobile network operators (MNOs), Internet providers, banks, cities, service providers, mobile phones manufacturers, tag managers?

1.1.1. *The era of mobiquity*

In 2015, there were as many mobile phone subscriptions as individuals on the planet, with 7.3 billion connected mobile devices, around half of which are smartphones[2] (only half of this population has a bank account). Almost every individual on the planet has (or will have) a communication terminal, with very fast data processing, in his pocket. This means that around 3 billion people have a mobile phone, but no bank account. Half the planet owns a smartphone, among which 64% will be NFC enabled in 2018[3], thus creating new opportunities of Internet services with added value: *a banker*, *a guide*, *a tutor*, *a doctor*, *a counselor in our pocket*. This prospect leads the way to unlimited creativity in terms of content, services and practices in the private, public and professional spheres the mobile phone crosses.

The era of the *mobile Internet* is only starting, the era of the Internet between objects and individuals, *symbiont* (J. de Rosnay),

1 http://www.cnil.fr.
2 Ericsson Mobility, November 2015 (http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf).
3 Forecasts from IHS Technology (https://technology.ihs.com).

suppression of space-time (Michel Serres), *creative destruction* (Schumpeter), and *mobiquity*, which will be very productive in terms of innovations and research in all economic sectors: teaching, tourism, culture, mobile commerce, mobile payment (international fund transfer, virtual money, etc.).

The three key factors to the success of a new technological ecosystem are as follows:

– usages;

– usages;

– usages.

A genuine engineering of mobiquitous services and applications was set up like illustrated by Apple's AppStore concept or Google's Market/Play Store (since 2013, more than 100 billion mobile applications are being downloaded in the world every year, according to Gartner Goup[4]).

Mobiquity is based on several technological concepts as follows:

– Real world tags read by the end user's mobile phone; these tags can be two-dimensional barcodes (such as the data matrix standard and its open source derivative, the QR code), radiofrequency tags (RFID; such as NFC tags), audio tags, even invisible tags with pattern recognition (such as Tokidev's *Snap'n See* or Google's *Goggles*). In 2020, one thousand billion tagged objects will exist, readable by our mobile phones, thus connected and *alive* from a biological understanding since life can be defined by the combination of *information* (here, the object's unique identifier) and *communication* (via the mobile phone that has access to the tagged object's history in a database).

– *Altered reality* (*augmented* or *diminished*) with *open source* platforms such as Wikitude and Layar, allowing to enter database information (for example real estate information or tourist information) on the top of the real world viewed from the mobile

4 Source: Mobile World Congress (http://www.mobileworldcongress.com).

phone, or, on the contrary, to remove/replace real-world elements (e.g. to imagine a new interior design, a new urban perspective, etc.).

– *Transmedia*: A concept created by Martha Fisher in 1991 and followed up in *Convergence Culture*, a book from MIT Professor Henry Jenkins in 2003, with contents adapted to the *five screens in our history* (cinema, TV, computer, mobile phone and tablet); we must note that in this history of communication, the leading companies of contents and services for one screen were never the leaders for the next screen. The next screens will be on the walls (windows, mirrors, clothes, the skin, etc.).

– *Big data*: According to IBM[5], 2.5 trillion[6] data bytes are generated every day, and 90% of the global data was created during the past 2 years. These data come from sensors, messages posted on social networks, pictures and videos published on the Internet, from geolocation (for example, GPS or NFC) from mobile devices (smartphones, tablets, smart watches), traceability logs for online payments (for example from e-payments or m-payments), keywords entered in search engines and, more generally, any digital information that can be used for interpretation (scientific data, fraud detection, medical, personal, behavioral data, etc.). Big data can be defined by "3V": volume ("data tsunami": 20 petabytes[7] of data are processed every day by Google, and this has been the case since 2010, 1.8[8] zettabytes were produced in 2011, meaning a 50% increase per year), variety (*web data* from social networks, *linked data* from semantic web, public data, *open data* and mobiquitous data coming from tagged objects and sensor networks, etc.) and velocity (data flow from sensors and social networks, real-time "on the fly"), to which we add a fourth "V": variability (evolution), and a fifth "V": valorization (improvement and enrichment).

5 IBM is one of the two world leaders (with Oracle) specialized in information systems and database management (www.ibm.com).
6 1 trillion = 1,000 billion, meaning a 10 units number.
7 1 petabyte (PB) = 250 bytes = 1,024 terabytes (TB) = 1,125,899,906,842,624 bytes.
8 1 zettabyte (ZB) = 270 bytes = 1,024 exabytes (EB) = 1,024 PB = 1,180,591,620, 717,411,303,424 bytes.

Figure 1.1. *Interview of Pr. S. Miranda for Docapost (2011)*

1.1.2. *Toward a world of contactless communicating objects*

With NFC standards, any object can become "communicating" (and thus alive). Whether they are on the market, or developing, there is already a wide range of NFC-enabled devices and wearables: smart watches, bracelets, glasses, clothes or directly on our body. Smart devices increasingly become part of our daily life; parking meters, NFC mailboxes, fridges analyzing its users' habits and forwarding the information to a smartphone (power consumption, temperature, door opening, etc.); connected oven able to pair with an NFC-enabled smartphone to suggest cooking recipes to the user and automatically set up a cooking function. NFC and its three operating modes (see section 1.3.2) in tag reader/writer, (smart) card emulation and peer-to-peer (P2P) represent a source of innovation for objects that have become "smart" because, specifically, of the smartphone's connectivity.

NFC-enabled "objects" can be divided into two categories:

–"Active" NFC-enabled objects (with a power supply) usually have an NFC reader: with or without graphic interface, they interact when brought into proximity with another peripheral device or an NFC tag. Most of the time, the "active" peripheral device hosts a "terminal" application, which means that its connectivity allows communication with a remote server paired up with a database in order to take information from the latter. In this way, the end user is able to trigger interactive and contextualized events whether they are

processed locally or remotely for a result in the digital world (e.g. information display or secure payment) and/or in the real world (e.g. door opening).

– "Passive" NFC objects are based on an NFC tag with no power supply; they activate when brought into close proximity with an "active" NFC-enabled object; they can then use the magnetic field power induced by the active device in "reader" mode, which then initiates the communication.

1.2. NFC equipment

A basic NFC device is made up of at least of an antenna coupled to a modulator-demodulator that converts electromagnetic signals into digital data (and vice versa); in an NFC tag, a chip has very low memory capacity (~1 kb), whereas for a more complex peripheral device, an NFC controller enables the NFC chip to interface with the host device's operating system.

1.2.1. *NFC tag*

NFC tags can be of any kind and form: stickers, key rings, bracelets, etc. They offer the same functionalities as a QR code, but they do not require more than a simple "tap" with no user intervention, unlike QR codes that require the user to launch the right application to read it. NFC tags are the link between the real and the virtual world; they can also be used to activate or deactivate functions (alarms, devices, digital processing) and any other event triggered by a simple "tap" using a smartphone.

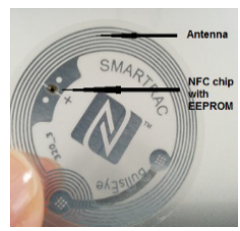

Figure 1.2. *NFC sticker*

Figure 1.2 shows a (passive) NFC tag, here a transparent sticker, which belongs to the NTAG's family produced by NXP.

Identification is a specificity inherent to NFC tags with an unmodifiable universal unique identifier (UID) attributed to them when manufactured. In this way, with no additional data needed, a connected object can simply be uniquely identified because of the UID of the NFC tag it is coupled to, whether the tag is stuck or embedded in this object. A "terminal" application connected to an NFC reader or executed in an NFC-enabled mobile device (in reader mode) can thus read the object's UID depending on the use case of the application:

– display information about the object (e.g. ID/electronic passports, touristic information, user guides);

– execute a task (e.g. make a reservation);

– start an event in the virtual world or even in the real world (door opening, alarm activation/deactivation, etc.);

– save an event by adding available contextual data to it (e.g. the object's UID, time and date stamp, geographical localization, user) for traceability or pointing purposes.

1.2.2. *NFC smart card*

The smart card is a more sophisticated type of tag with a sometimes hybrid communication interface (*dual interface*) providing both a traditional contact interface and an NFC interface enabling communication with services embedded in the smart card.

The contactless NFC smartcard is used as a payment card (in 2014, 600 million NFC payment cards were sold in the world[9]), as a transportation ticket as well as means of identification and authentication of the cardholder, for example for access control. The

9 Source: Mobile Payments Today (http://www.mobilepaymentstoday.com/news/contactless-payments-us-emv-migration-power-sharp-increase-in-smart-card-growth).

SIM card is the most widespread type (similar to a smart card, but smaller): there are as many active SIM cards as there are inhabitants in the world, and this is due to the fact that some users have more than one SIM card.

Embedded in an NFC-enabled mobile (smartphone, tablet), the smart card acts as a *secure element* (SE) in order to safely host services and confidential data. This configuration shows the advantage of extending services based on smart cards to smartphones' advanced technologies (graphical user touch sensitive interface, audio, camera, embedded sensors, GPS, etc.) and, mostly, its web connectivity offers friendly user interfaces and improves services with new functionalities.

1.2.2.1. *Smart cards and security*

Enforced security is a broad field covering many aspects such as protection of privacy, access control, protection against viruses and hacking, integrity and data non-repudiation in the digital world.

Authentication is used to validate user identity. It can be attested through different, possibly combined factors (multi-factor authentication). For example, they can be based on:

– the individual (biometric information);

– a shared secret (password, PIN, encryption key);

– an object (smart card, smartphone, etc.).

Permission/authorization consists of checking that the (authenticated) entity wishing to access a resource is permitted to do so. Permission management is done on an individual level or by a group of entities (permissions or restrictions can be defined according to roles).

Confidentiality is the guarantee that neither authentication data nor shared resources can be intercepted. Data encryption (cryptography) is the most widespread technique to preserve confidentiality.

Data *integrity* consists of making sure that resources do not experience any alteration, during data storage nor during the access to where they must be fully retrieved, with the same precision, but mostly by attesting their authenticity and their validity.

The public key infrastructure (PKI) standard is a method for cryptographic key management based on the principle of private key/public key:

– a private key is generated in order to encrypt (or "encipher") messages. Private keys remain secret and are never distributed;

– one or several public keys are generated and distributed to encrypted message recipients and are used to decrypt the message (and encrypt replies).

In this way, a PKI-based system secures the provenance of the message (only encrypted messages with a private key can be decrypted with the public key) and confidentiality (only the owner of the public key can decrypt the encrypted message with the private key).

The main characteristics of PKI are based on software and hardware security:

– the private key is stored in a secure electronic (SE) chip and never leaves it;

– the public key is exported;

– the encryption mechanism is hard coded in the chip;

– the authentication mechanism is also embedded in the tamper-resistant technology of the smart card's microcontroller.

In this way, smart cards are ideally suitable for identification and authentication purposes; among other things, they store an encrypted digital certificate in order to authenticate for a service. The most widely used cryptographic algorithms are triple Data Encryption Standard (3DES) with private and shared keys (in symmetrical reader

mode) and the RSA standard[10] with the distribution of a public key (in an asymmetrical mode). Keys can be loaded or generated on the smart card at the customization stage.

Europary MasterCard Visa (EMV) cards, chip-based debit–credit cards, as well as payment terminals and ATMs allowing the use of bank cards are widely deployed to attest the cardholder's authentication. This is one of the reasons why smart cards are prime targets of security attacks. Smart card attacks go from physical intrusion of electronics (with a ionic probe, a microscope, a chemical attack or a laser, etc.), which lead to the destruction of the smart card, to semi-invasive attacks (with an oscilloscope) and non-invasive attacks (programmatically), which exploit weaknesses in the card's hardware and software. Smart card manufacturers and security standards have developed simultaneously with attacks in order to incorporate countermeasures guaranteeing a maximum security and immunity during the card's lifecycle; this is why it is important to only deploy recent smart card models whose technology incorporates full countermeasures to known attacks.

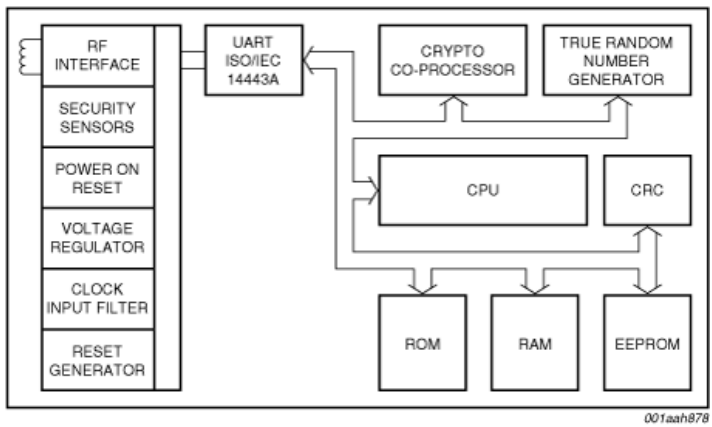

Figure 1.3. *Contactless smart card architecture (source NXP[11])*

10 Algorithm conceived by Rivest, Shamir and Adleman.
11 http://www.nxp.com/scale-image/w-800/documents/blockdiagram/001aah878.gif.

The most sophisticated smart cards are conceived with materials, which especially are dedicated to cryptography and encryption algorithm such as RSA and public key digital signature algorithms based on elliptic curve cryptography, used in asymmetrical operations for key exchanges on a non-secure channel or for asymmetrical encryption. The key pairs are generated inside the smart card in order to remove all risk of revelation of the private key, which is not distributed and thus remains unknown.

1.2.2.2. *Multifactor authentication*

A method of enforcement of digital access control that requires more than one typical authentication factor. They can be login credentials (login, password) and a one-time-use code randomly generated by the system and whose validity is limited in time. The one-time-use code is sent in real-time to the user through another communication channel, for example the sending of an authorization code by SMS is a wide spread example of the two-factor authentication (2FA) used in secure online payment transactions: the user authenticates to the bank interface on the web with his/her login and password and validates the purchase, then the bank system sends a verification code by SMS on the mobile phone of the user and the user must input this received verification code in the Web interface to complete the transaction. The payment is validated only after the check of the 2FA.

Usually, the multi-factors authentication is based on two complementary principles:

– something the user “knows”, for example name and password or PIN code;

– something the user “has” (and the system knows), for example a token, a chip or smart card (containing a key, for example), a mobile phone, a dongle, etc.

1.2.2.3. *Secure communication channels*

Isolated communication channel enables to establish a private network channel between two programs (i.e. VPN/specific point-to-

point socket connection). Setting up a secure communication channel requires prior authentication and identification; the protocol must ensure privacy and integrity (encryption/decryption, state). An isolated communication channel can be secure or not, and a secure communication can be isolated or not.

1.2.3. *NFC smartphone*

NFC smartphones answer five dimensions in a mobiquitous information system (the five "W": *who, where, when, whereabouts and what*): holder's identity (with his habits, his preferences), space and time ("here and now", when he/she taps), goal to reach by tapping (information, transaction?) and result obtained (information, voucher, transaction, appointment, etc.).

The first NFC-enabled mobile phone was launched on the market in 2006 by Nokia: the Nokia 6131 NFC was delivered with an embedded SE (or eSE), for a default use of services (e.g. MasterCard, Visa©, SNCF) available on a service platform.

Today, all major smartphone manufacturers propose NFC-enabled mobile devices: Google first launched the first NFC smartphone in 2010 (Nexus S) with a primary strategy on mobile payment launched in May 2011 in New York. Since then, Samsung, Apple, then LG followed Google, offering their own contactless, mobile m-payment platforms, competing with the universal SIM card as an SE controlled by the MNO.

With or without SE, the NFC smartphone can act as an NFC tag (or a contactless smart card), but it can also act as NFC reader in order to read (or write) the content of tags, or even act as payment terminal, which might as well revolutionize secure transactions; so far, this field was for proprietary systems only, owned and controlled by professionals.

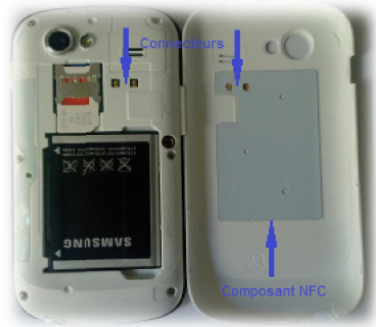

Figure 1.4. *Nexus S NFC antenna incorporated in the casing*

1.2.4. *Reader/encoder: NFC transaction terminals*

The NFC reader can read and write content on tags depending on formats determined by the NFC standard (see section 1.3). With or without a keypad and a screen, NFC readers are accessed by a program that allows reading/writing or communication with another NFC device (for example a tag or a smartphone) depending on the reader's standards compliance.

Figure 1.5. *ASC NFC reader ACR122U*

1.2.5. *"Smart cities" and sustainable development*

Whether it be through infrared, Wi-Fi, Bluetooth® or NFC, we live in the times of "machine-to-machine" and connected objects:

connected infrastructures equipped with sensors (pollution, humidity, light, temperature, motion, traffic jam, etc.), automated triggering (alarms, signals, urban lightening, public garden watering, etc.), electronic autodiagnostic, automatic detection of accidents and troubleshooting, etc.

NFC finds practical applications in eco-citizen life, for example in multimodality, which aims at offering all possible transportation solutions, possible connections from point A to point B: mobile applications allow us to locate, book and pay for ecological means of transportation (*vélibs/blue bikes*, electric vehicles, car share or public transportation), while calculating their availability and based on journey time, schedules, etc.

NFC is a universal connector with which any space can become a smart place: storing real-time medical data in a healthcare platform that would tell us when to go see a doctor, collecting statistics on power consumption directly on our mobile phone, benefiting from directions for use and being able to interact remotely, avoiding waste of time at a cash desk to pay for something.

All these concepts are in fashion, there are existing pilots and their emergence goes hand in hand with a mindset of sustainability. In this context, because of its low power consumption and its secure approach to communication, NFC will play a key role.

1.2.6. *Cashless payment with NFC*

E-commerce was born with the emergence of the Internet and online commerce; this new consumption pattern boomed with the appearance of smartphones and tablets, opening the way to m-commerce (whether it be geolocated or not), and m-payment has become a habit. Today, with technology we can select a product, wherever we are (at home, in the office or outside, alone or with others), whenever, whether it be day or night.

M-payment opened the way to dematerializing traditional means of payment (cash, cheque, bank cards): already globally tested out, the

use of NFC standards seems to be the most appropriate technology to compete with bank cards (as a logical addition to blockchain transactions and cryptocurrencies).

The stakes are high since NFC could disrupt an established ecosystem and initiate a transition between banking monopoly toward new actors (MNOs, Google, Apple, Samsung) and the opening to private individuals with online banking management (money transfer from private bank accounts to third party accounts), then from our mobile phones ("bottom-up" approach in which the final user takes control vs. "top-down" approach where the bank service provider manages and controls everything).

The concept of virtual currency can also apply to unbanked populations: according to a 2012 World Bank study conducted on financial inclusion, 2.5 billion adults on the planet (almost half of the world population) have no bank account (among them, 70% have a mobile phone). This is a huge field for potential dynamization from which mobiquitous projects emerged on crowd funding or microfinance, whose exchange currency is not necessarily financial (for example livestock exchange and seed trade).

1.3. NFC standards

Outside of standards, no salvation for developers!

Standardization meets a common need in a multi-actors ecosystem in order to ease interoperability, durable applications development and promote the deployment of new technologies and their usages.

NFC standards are inherently tied to telecoms and smartphones, on the one hand, due to the fact that its use cases are ideally adapted to mobile use, and, on the other hand, because of the most commonly used smart card, globally deployed in all mobile phones: the SIM card.

In 2002[12], Philips (called NXP in 2006) and Sony invented the NFC technology and standard based upon their know-how of wireless chips (FeliCa for Sony and MIFARE). At that time, Nokia joined Philips and Sony to create the NFC Forum in 2004 (www.nfc forum.org).

NFC is one of the 16 RFID ISO standards, classified according to their frequency range: low frequency (<135 kHz), high frequency (including NFC at 13.56 MHz), ultrahigh frequency (433 MHz) and microwaves (2.45 and 5.8 GHz).

NFC standards can be used within smart cards as well as mobile devices. Encounters of the most popular hardware device (mobile phone) with the most widespread communication platforms (the Internet) and the most recent wireless short-range standard (NFC) gave birth to a new paradigm of services in public transports, mobile payment, marketing, geolocation, contextual information retrieval, social networks, teaching, digital content delivery and sharing, while rethinking physical world interactions. Use cases of contactless applications and services with NFC are unlimited.

In this way, NFC is a half or full-duplex, short-range wireless communication technology (<10 cm) based on RFID and unlicensed 13.56 MHz radiofrequency. NFC standards are recognized by ISO/IEC and ETSI and gathers several norms such as ISO/IEC 18092, ISO/IEC 14443 and ECMA (ECMA-340).

NFC allows two communication modes based on RFID inductive coupling:

– *NFC passive communication mode*: in this mode, only the NFC communication initiator is empowered; the target device of the communication answers load modulation using inductive coupling in the near vicinity of the initiator device such as illustrated in Figure 1.6.

12 Source: TechPats (http://www.techpats.com/evolution-near-field-communication-nfc).

– *NFC active communication mode:* in this communication mode, both devices (initiator and target device) generate their own radiofrequency waves to transmit data.

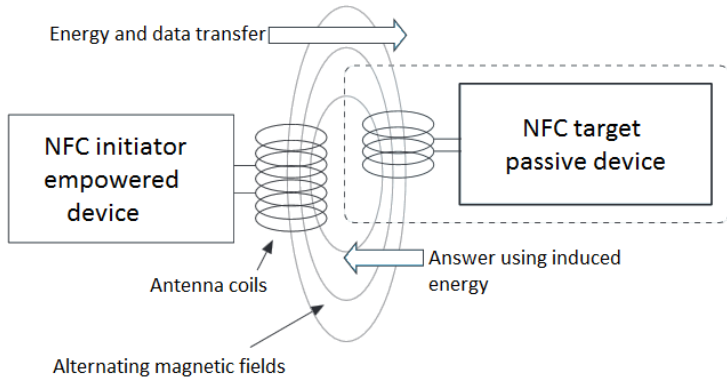

Figure 1.6. *NFC passive communication mode*

1.3.1. *Analog signal and NFC digital transposition*

Contactless communication consists of modulating and demodulating binary data (bits of value 0 or 1) in a signal (analog signal) by sending a wave (electromagnetic) called a "carrier wave": through the creation of a variation in amplitude, phase and frequency, the signal is digitally transposed when received.

In its initial specification, the NFC standard gathers three types of codings:

– NFC-A uses Miller (sender) and Manchester (receptor) codings with an amplitude modulation at 100% amplitude shift keying (ASK) (zero signal during breaks) at 106 kbps[13];

– NFC-B uses non-return-to-zero (NRZ) coding with an amplitude modulation at 10% ASK (still no signal during breaks) when sent and modulation through binary phase shift keying (BPSK) phase shift when received at 106 kbps;

13 Clockrate in kilobytes per second (kbps).

– NFC-F corresponds to FeliCa technology, which is commonly developed in Japan and uses Manchester encoding with an ASK amplitude modulation at 212 or 424 kbps.

NOTE.– The NFC forum is studying a new specification through which the standard (Tag Type 5) would be compliant with the vicinity cards signaling (standard ISO/IEC 15693) in the short range required by NFC.

1.3.1.1. *ASK modulation*

The transition (shift from 1 to 0 bit) occurs because of a shift in the amplitude of the signal: the amplitude is lowered or null.

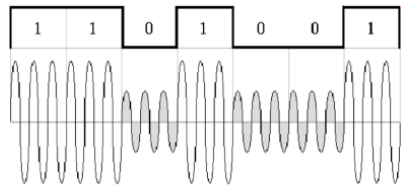

Figure 1.7. *ASK modulation*

1.3.1.2. *BPSK modulation*

With each transition, the signal's amplitude is reversed, thus creating a phase jump.

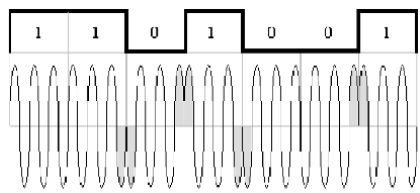

Figure 1.8. *BPSK modulation*

1.3.1.3. *Demodulation*

Demodulation is a data extraction process that consists of converting an analog source into binary encoding.

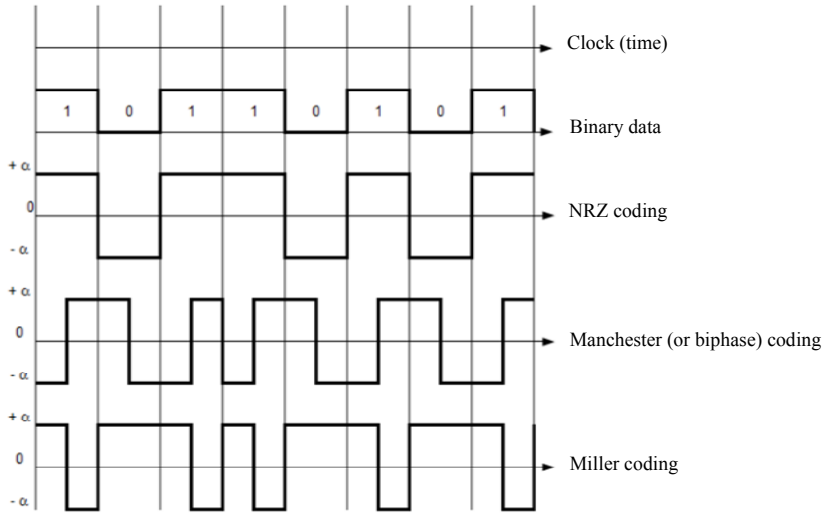

Figure 1.9. *Encoding of digital modulation of analog signals*

1.3.1.4. *NRZ encoding*

NRZ is the simplest coding. A positive tension represents the data bit of value 1, whereas a negative tension represents the bit of value 0.

1.3.1.5. *Manchester coding (or binary phase)*

It is a transition coding (not a level coding): a rising edge transition represents a bit of value 0. A falling edge transition corresponds to a bit of value 1.

1.3.1.6. *Miller coding*

The Miller coding scheme is based on the Manchester coding scheme from which every other transition is discarded: for a bit of value 1, a transition is placed at the midpoint of a bit interval, unless it is followed by a bit of value 0, in which case a transition is placed at the end of the bit interval.

1.3.2. *The three standardized modes of NFC*

NFC-enabled devices can support three operating modes, as illustrated in Figure 1.10:

– reader/writer (i.e. tag reader/writer);

– P2P;

– card emulation (i.e. chip).

Figure 1.10. *Three NFC devices operating modes*[14]

NOTE.– These three complementary operating modes of NFC standard are exclusive to one another.

NFC modes are based upon the ISO/IEC 18092 NFC IP-1, JIS X 6319-4 and ISO/IEC 14443 contactless smart card standards (referred to as NFC-A, NFC-B and NFC-F in NFC forum specifications).

1.3.2.1. *NFC read/write mode*

As illustrated in Figure 1.11, the read/write mode of NFC enables NFC devices to read and write information stored on passive NFC tags consisting of an antenna and an integrated circuitry. Power is supplied by inductive coupling from the NFC device initiator of the communication when it is brought into the proximity of the NFC passive tag.

14 Source: NFC forum (http://nfc-forum.org/what-is-nfc/what-it-does).

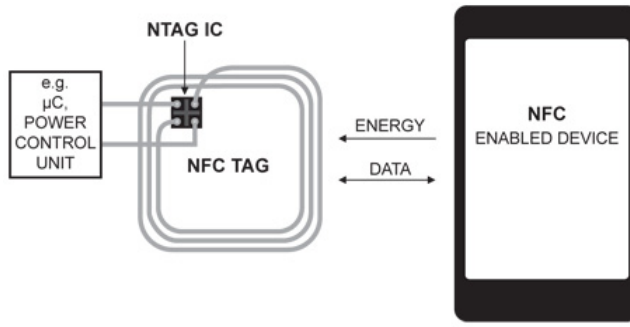

Figure 1.11. *NFC read/write mode*[15]

Information stored on the NFC tag should comply with the record type definition (RTD) standard specified by NFC forum or be a proprietary format (see section 1.3.3.3.2).

Different types of NFC tags are described by the standard (see section 1.3.3.3.1): in the NFC tag, data are stored in NFC data exchange format (NDEF) messages (see section 1.3.3.2) that include type, type format, ID and byte array. For example it is possible to write download links/URLs into the NFC tag, which can store several records. A mere reference can also enable the device to recollect linked data from a remote server, for example using dedicated web service.

The NFC-enabled device can, with prior end user acceptance or automatically, launch the appropriate application to access the linked content (for example text, media picture/audio/video, web page) or trigger an electrical action (for example switch-on the light), a mechanical or olfactory action.

1.3.2.2. *NFC card emulation mode with SE*

NFC card emulation mode enables NFC devices to act like smart cards, allowing users to perform transactions such as mobile payment,

15 Source: NFC World (http://www.nfcworld.com/2013/08/14/325492/nxp-begins-shipping- nfc-tags-that-can-wake-up-a-host-device).

ticketing and transit access control with just a tap. In this mode, the contactless layer and RF fields act as a transport for smart card standard (ISO/IEC 7816), as illustrated in Figure 1.12.

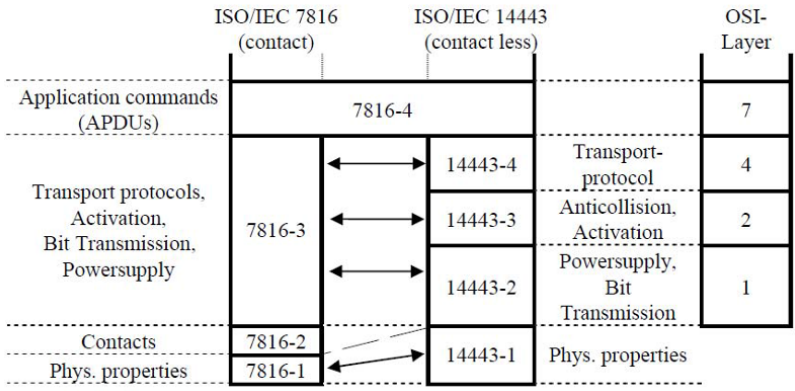

Figure 1.12. *NFC card emulation mode protocols*

In card emulation mode, NFC devices communicate with an external contactless reader: the NFC card emulation mode allows the handling of already existing protocols on top of NFC stacks with no change in infrastructure (see Figure 1.12). For NFC payment applications, this NFC card emulation mode is based upon the EMV standard and PIN card specifications.

NFC card emulation mode involves a tamper-resistant hardware component called “secure element”. The SE is a multiservice chip with its own microprocessor (CPU) and a crypto-processor, its own operating system (JavaCard™) as well as volatile and non-volatile, including Erasable Programmable Read Only Memory (EPROM) with a storage capacity up to 6 Gb, input/output interface to receive messages and send responses and one or several (for example contact/RFID hybrid card) power supply interfaces. It can either be a standalone (contact or contactless) smart card or a component of a

host device (e.g. a smartphone or a tablet) with, in that case, three possible SE configurations, as illustrated in Figure 1.13:

– in the SIM card or the universal integrated circuit card, handled by an MNO;

– embedded in the device (eSE) and handled by the device manufacturer (for example Apple, Samsung, LG);

– external/removable secure memory cards such as the micro Secure Digital (microSD).

The SE could be accessed remotely in the Cloud, and/or emulated by a background service that behaves like a (hardware) SE. This mode, which is not hardware based, is called the *host-based card emulation* (HCE) mode.

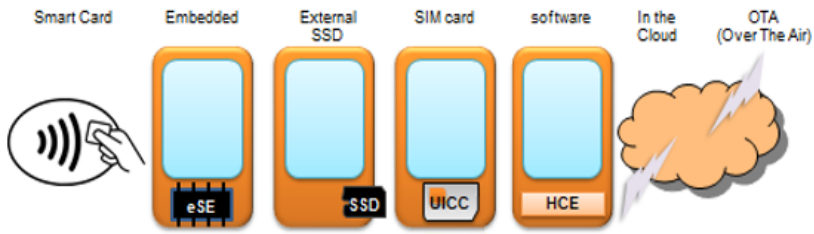

Figure 1.13. *Several configurations of SE*

The SE is mainly intended to securely store confidential data (e.g. account data, identification, user credential, encryption/decryption keys). Processes running into the SE (outside the main OS of the host device) are commonly called *Applets* (inheritance from JavaCard™), *Cardlet* or *Servlet*; the application that allows the user to manage a set of virtual cards and services digitalized into the SE is called a *Wallet*.

SE-related standards are centralized by GlobalPlatform (www.globalplatform.org), the international cross-industry standardization organization adopted as global reference by MNOs

with the GSM Alliance (GSMA) and European Telecommunications Standards Institute, as well as banking institutions (e.g. EMV) and other standardization bodies, including the NFC forum.

1.3.2.3. *NFC P2P mode*

NFC P2P mode enables two NFC-enabled devices to exchange information and share content. NFC P2P mode can be used to pair another NFC device (e.g. computer, speaker, headphones, TV) and boostrap a secondary high-speed connection like Bluetooth® or Wi-Fi (e.g. by exchanging setup parameters). In this case, NFC is used to negotiate a second communication channel and transfer authentication data for the secondary protocol. The file or data (e.g. media file: pictures, videos or audio files) is then sent over the high bandwidth protocol. Two standardized operating modes (either passive or active role) exist (see Table 1.1):

– NFC IP-1 (initiator and target NFC devices);

– NFC *logical link control protocol* (LLCP) (successively active mode/passive mode – passive mode/active mode).

These standards are shown in section 1.3.3.

NFC mode	Active	Passive
Active	P2P	Card emulation
Passive	R/W	–

Table 1.1. *NFC three standardized modes*

1.3.3. *NFC forum standards*

The NFC Forum was founded in 2004 by Philips Semiconductors (then NXP) and Nokia, the inventors of NFC, and then joined by Sony. It is a non-profit industry association whose objective is to

develop and promote NFC technology. NFC forum specifications are based upon the ISO/IEC 14443 standard, which slightly differs from ISO/IEC 18092 standard in terms of RF layers and NFC P2P mode which was addressed by the NFC forum in 2011 (see Figure 1.14).

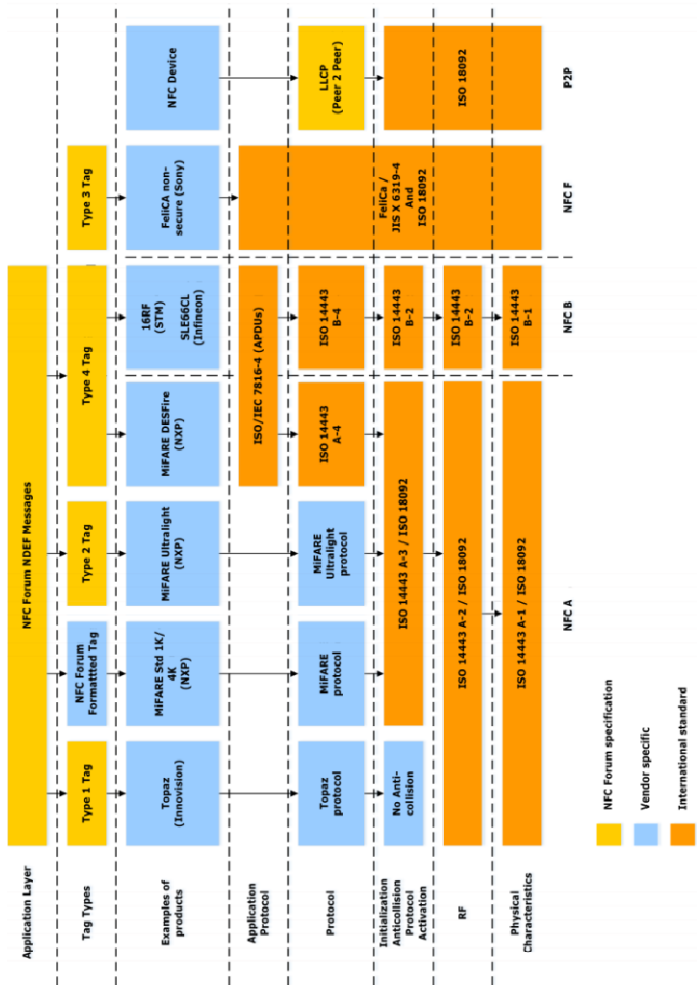

Figure 1.14. *NFC forum standards*[16]

16 Source: Erik Hubers. Licensed under CC BY-SA 4.0.

Figure 1.15. *N-mark of NFC forum*

NOTE.– The use of "N-mark" (see Figure 1.15.) shows compliance with NFC forum specifications. Signing the license agreement for brand use is required in order to acquire N-mark.

NFC forum application documents provide guidance on how to take advantage of contactless NFC technology in specific targeted scenarios. For example, in the application documents (i.e. for NFC) provided by the NFC forum, we can find guidance on how to use NFC for secure Bluetooth® pairing: this application document describes the interactions between Bluetooth® and NFC technologies. It provides examples of both protocol implementation and data transfer for the most appropriate use cases of these two technologies. Developers will find useful guides for their own work. The document was expanded (in June 2014) to the *Bluetooth low energy* technology, a version of Bluetooth technology that offers reduced power consumption (NFC forum, "*Bluetooth Secure Simple Pairing Using NFC Application Document*", 2011).

In this way, when used with NFC forum specifications, these tools can help achieve full interoperability across technologies: technical documentation and tools suggested by the NFC forum offer guidance on best practices and NFC implementation solutions (free of charge for NFC forum members, charged for non-members), as listed below:

1) NFCIP-1 (ISO/CEI 18092) communication interface protocol standardized by the ISO/IEC 18092 and ECMA-340 standard (suitable for devices pairing in P2P mode). The protocol stack in NFCIP-1 is placed on top of the NFC-A protocol layer (see section 1.3.1.) defined in ISO/IEC 14443 and type 3 tags of NFC forum (see section 1.3.3.3.1). NFCIP-1 includes both passive and active communication

modes, which allow an NFC device to communicate with other NFC devices in a P2P manner:

– *active* communication mode means both the initiator and the target are using their own self-generated and self-modulated RF field to transmit data;

– *passive* communication mode means that the NFC target device responds to an initiator command in a load modulation scheme (using the energy induced by the RF field generated by the initiator).

NOTE.– NFCIP-2 (ISO/IEC 21484) is the specification for the selection mechanism between different contactless technologies that operates on the same frequency at 13.56 MHz. It supports communication according to ISO/IEC 18092, ISO/IEC 14443. This specification is also compatible with other contactless technologies like Vicinity cards (ISO/IEC 15693) that is also known as NFC-V, currently reviewed by NFC forum (i.e. in order to offer a specification including vicinity compliance with NFC standard).

2) ISO/IEC 14443 standard describes contactless integrated circuits communication interface protocols divided into four parts:

– 14443-1: physical characteristics;

– 14443-2: RF power supply and signal interface;

– 14443-3: initialization and anticollision mechanisms;

– 14443-4: transmission protocol.

3) NDEF logical format of data exchange (see section 1.3.3.2).

1.3.3.1. *NFC forum protocols specifications*

A standard specification document explicitly describes a set of requirements of technical criteria, methods and processes. In systems and software engineering, it is intended to establish a guidance for the development and the implementation of the system (software or hardware) describing for example what must/may/shall or not be done

and how. Each provider can develop their own specification implementation.

NOTE.– The NFC forum proposes a certification program for the providers who want to have their products certified and ensure they are compliant with NFC forum specifications. NFC forum specifications are related to communication protocols and application (messaging) layers of the OSI model including the tag types.

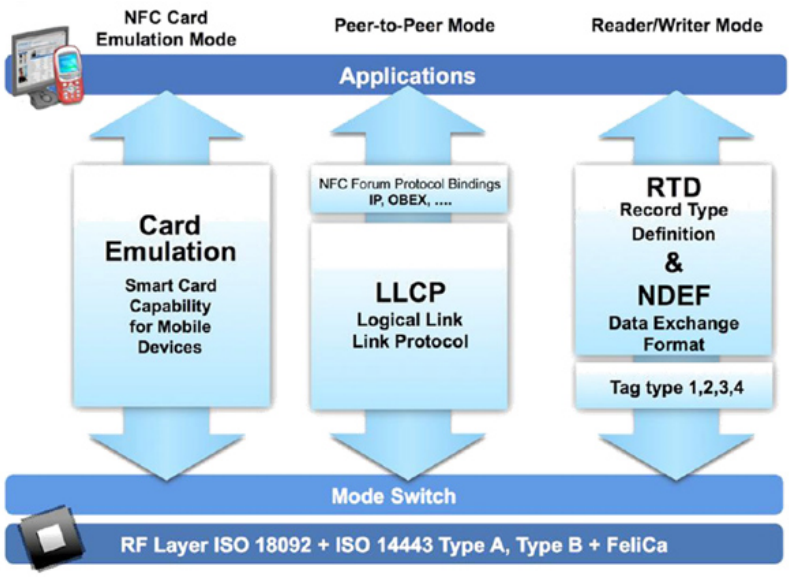

Figure 1.16. *NFC standard stacks (source NFC forum)*

NFC LLCP protocol based on Open Systems Interconnection (OSI) *message authentication code* layer-2 of IEEE 802.2 standard (see Figure 1.17), ISO/IEC 18092 and ISO/IEC 14443 supporting P2P between two NFC-enabled devices for bidirectional communication. It is used on the top of NFCIP-1.

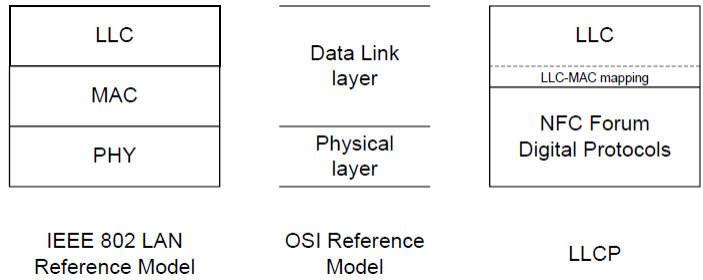

Figure 1.17. *NFC LLCP protocol and OSI model*[17]

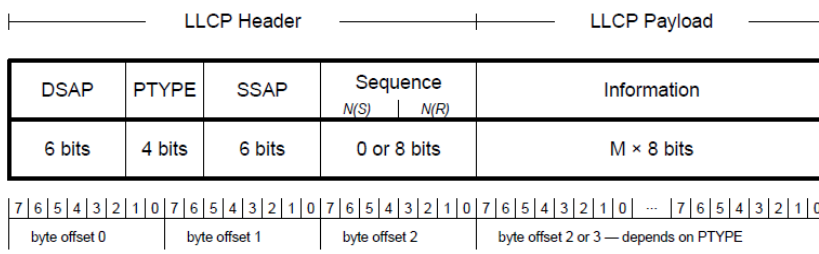

Figure 1.18. *LLCP PDU format*[18]

LLCP protocol specification describes the format of *protocol data unit* (PDU) exchange structures made up of header and payload (see Figure 1.18):

– LLCP PDU format is as follows:

- destination service access point (DSAP);
- payload type (PTYPE);
- source service access point (SSAP);
- PDU sequence;

17 Source: NFC forum, Figure 1, p. 4 in *Logical Link Control Protocol Technical Specification 1.0*, December 2009.
18 Source: NFC forum, Figure 3, p. 14 in *Logical Link Control Protocol Technical Specification 1.0*, December 2009.

- Payload.

– NFC digital protocol on the top of ISO/IEC 18092 and ISO/IEC 14443 and JIS X6319-4 standards gathers common features and digital interface and the half-duplex transmission protocol of the NFC-enabled device in its four roles as initiator, target, reader/writer and card emulator.

– NFC activity provides communication setup between NFC devices: Profiles specific configuration parameters, polling for an NFC tag or NFC device in combination, NDEF data, etc.

– NFC simple NDEF exchange protocol for NFC-enabled devices messaging over LLCP.

– NFC analog: RF interface (signal form, time/frequency/ modulation characteristics) and power requirements of the NFC-enabled device in its four roles (P2P mode initiator, P2P mode target, reader/writer mode and card emulation mode) for all three technologies: NFC-A, NFC-B and NFC-C (depending on signal modulation/demodulation methods) and for all the different bit rates (106, 212 and 424 kbps).

– NFC controller interface between NFC controller and the device's main application processor.

1.3.3.2. *NDEF data exchange format*

NDEF defines the data format structure of an NFC message to be read or encoded in a tag. NDEF messages have a header and one or several NDEF records with a header for each of them.

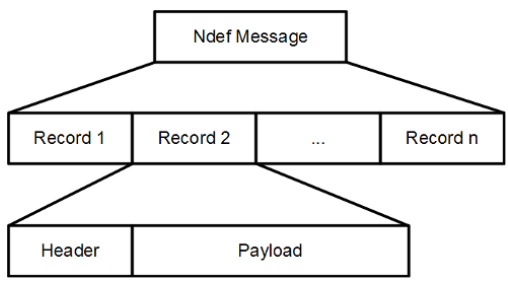

Figure 1.19. *NDEF message structure*

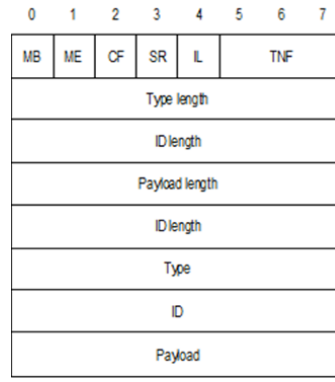

Figure 1.20. *NDEF record structure*

NDEF Record Type	Description	Full URI Reference	Specification Reference
Sp	Smart Poster	urn:nfc:wkt:Sp	NFC Forum Smart Poster RTD
T	Text	urn:nfc:wkt:T	NFC Forum Text RTD
U	URI	urn:nfc:wkt:U	NFC Forum URI RTD
Gc	Generic Control	urn:nfc:wkt:Gc	NFC Forum Generic Control RTD**
Hr	Handover Request	urn:nfc:wkt:Hr	NFC Forum Connection Handover Specification
Hs	Handover Select	urn:nfc:wkt:Hs	NFC Forum Connection Handover Specification
Hc	Handover Carrier	urn:nfc:wkt:Hc	NFC Forum Connection Handover Specification
Sig	Signature	urn:nfc:wkt:Sig	NFC Forum Signature RTD

Table 1.2. *Well-known NDEF Record Types*[19]

19 Source: NFC forum (http://www.nfc-forum.org/specs/nfcforumassignednumbers register).

As illustrated in Figure 1.20, the first bytes are used by the header:

– MB: message begin on 1-bit taking a value of 1 if the message starts, otherwise 0.

– ME: message end on 1-bit taking a value of 1 if the message ends, otherwise 0.

– CF: bit indicates a chunked payload.

– SR: bit indicates a short record on 7 bytes instead of 10.

– IL: bit indicates if ID length bytes and ID must be read.

– TNF: type name format on 3 bits.

– 0x00: empty record.

– 0x01: well-known type defined by NFC forum (see Table 1.2).

– 0x02: MIME type (text, media, picture, etc.).

– 0x03: URI.

– 0x04: external.

– 0x05: unknown type.

– 0x06: unchanged (for chunk records).

– 0x07: reserved for future use.

– Type length: length of the PTYPE field on 8 bytes.

– ID length: size of the payload ID field on 8 bytes.

– Payload length: specifies the length of the payload field whose size is determined by the SR field.

– Type: record type (see Table 1.2) in hexadecimal (e.g. “U” for URI, value 0x55).

– ID: type ID or hexadecimal prefix (e.g., 0x01 is “http://www”).

– Prefix: according to TNF, for example for URI type:

- 0x00: no prefix;

- 0x01: http://www;

- 0x02: https://www.

- 0x03: http://

- 0x04: https://

- 0x05: tel:

- 0x06: mailto:

- 0x1D: file://

- 0x24…0xFF: reserved for future use.

– Payload: content the size of which is determined by payload length field.

1.3.3.3. *NFC forum tag types*

NFC tags' content and capacity differ according to their characteristics, from tags locking and unlocking write capabilities, storing one or several messages linked to Internet content, to more complex tags like smart cards capable of embedding data as well as applications.

1.3.3.3.1. NFC tag types

Four NFC tag types (see Table 1.3) are defined according to compliance with various standards and protocols.

NFC Forum type	Tags type 1	Tags Type 2	Tags type 3	Tags type 4
Memory	96 Bytes . 2KB	96 Bytes . 2KB	Up to 1MBytes (service)	4 . 32KB
Rate	106Kbits/s	106Kbits/s	212 or 424 Kbits/s	106, 212 or 424 Kbits/s
Data access	R/W or Read only	R/W or Read only	R/W or Read only	R/W or Read only
Collision handling	No	Yes	Yes	Yes
Compliance	Broadcom Topaze	NXP Mifare Ultralight, NXP Mifare Ultralight C, NXP NTAG203	Sony FeliCa	NXP DESFire / NXP SmartMX-JCOP
Standard	ISO14443A	ISO14443A	Japan Industrial Standards (JIS) X 6319-4	ISO14443A et ISO14443B
Price	Low cost	Low cost	Expensive	Medium to expensive

Table 1.3. *Four types of NFC tags*

In Table 1.4, a level of protocol in the activation phase of communication and data exchange differentiates the tag types.

<table>
<tr><th colspan="9">Standard</th></tr>
<tr><td rowspan="2">Collisions management</td><td colspan="4">ISO 18092 NFCIP – 1 (ECMA 340)</td><td colspan="2">ISO 21481 NFCIP – 2 (ECMA 352)</td><td colspan="2">ISO 18092 NFCIP – 1 (ECMA 340)</td></tr>
<tr><td colspan="4">NFC-A</td><td colspan="2">NFC-B</td><td colspan="2">NFC-F</td></tr>
<tr><td rowspan="3">Activation</td><td rowspan="3">Protocol NFC-DEP (P2P) based on ISO 18092</td><td colspan="6">Platforme</td><td rowspan="3">Protocol NFC-DEP (P2P) based on ISO 18092</td></tr>
<tr><td colspan="4">ISO 14443A</td><td>ISO 14443B (Proximity) & 15693 (Vicinity)</td><td>JIS X 6319-4</td></tr>
<tr><td>Type 1</td><td>Type 2</td><td>Type 4A</td><td>Type 4B</td><td>Type 3</td></tr>
<tr><td>Data exchange, Deactivation</td><td>Protocol Half-Duplex cmd RW Topaz Innovation (Braodcom)</td><td>Protocol Half-Duplex cmd RW Mifare Ultra Light (NXP)</td><td colspan="3">Protocol ISO-DEP based on ISO 14443 (-4) & EMV_CLESS</td><td>Protocol Half-Duplex cmd RW FeliCa (Sony)</td></tr>
</table>

Table 1.4. *NFC communication standards*

Type 1–4 tags are all based on existing contactless products:

– NFC forum type 1 tag based on the ISO/IEC 14443 standard is rewritten capable over 96 bytes (expandable to 2 kb). The Innovision Topaz Tag (from Broadcom) is the most commonly used Type 1 tag.

– NFC forum type 2 tag based on the ISO/IEC 14443A standard is rewritten capable over 48 bytes (expandable to 2 kb). Mifare Ultralight and NTAG tags (NXP) are the most commonly used type 2 tags.

– NFC forum type 3 tag is preconfigured at manufacturing to be both read and rewritable, or read-only, theoretical memory limit is 1 MB per service. Type 3 tags are based on Japanese Industrial Standard (JIS X 6319-4); Felica tags (Sony) are the most commonly used.

– NFC forum type 4 tag (defined in November 2010) is fully compatible with the ISO/IEC 14443 standard series. Tags are preconfigured at manufacturing to be either read and rewritable, or read-only, memory variable up to 32 kb per service; the communication interface is either Type A or Type B compliant. These tag types can be interfaced on the application level according to ISO/IEC 7816-4 Application Protocol Data Unit (APDU). These type

4 tags are usually used for contactless transaction payments and ticketing, such as Mifare DESfire (NXP).

1.3.3.3.2. Record type definition

Record types used by NFC forum application and third parties are based on the NDEF data format described in section 1.3.3.2. Five specific RTDs are defined (see Table 1.2): Text, URI, Smart Poster, Generic Control and Signature.

1.3.3.3.3. Reference applications

NFC forum connection handover was determined in 2010 to enable a static and dynamic one-touch setup of NFC with high-speed communication technologies, such as Wi-Fi setup or Bluetooth® pairing.

NFC forum also has a standard for communication with peripheral devices dedicated to healthcare technologies (NFC Forum Personal Health Device Communication) and based on ISO/IEC/IEEE 11073-20601 standard (Health informatics).

1.3.4. *GlobalPlatform (GP)*

Based on "Visa© OpenPlatform" standards in the hands of the OpenPlatform consortium, become GlobalPlatform (GP) in 1999, GP was created to meet the interoperability needs emerging from the globalization of payment transactions via smart cards; its main objectives were to promote the security, development platforms and management of the deployment of smart cards and SEs standardization.

GP acts upon interoperability for actors of the NFC ecosystem in card emulation mode, and especially upon SE management. It issues specifications and instructions on the ecosystem, especially security rules, protocols, architectures and card interfaces (card specifications[20]), devices (device specifications[21]) and systems

20 http://www.globalplatform.org/specificationscard.asp.
21 https://www.globalplatform.org/specificationsdevice.asp.

(systems specifications[22]); GP defines use case scenarios, design patterns, frameworks and APIs in order to ease SE services implementation (some resources have limited access: free for members and charged for non-members).

1.3.4.1. *GP cards specifications*

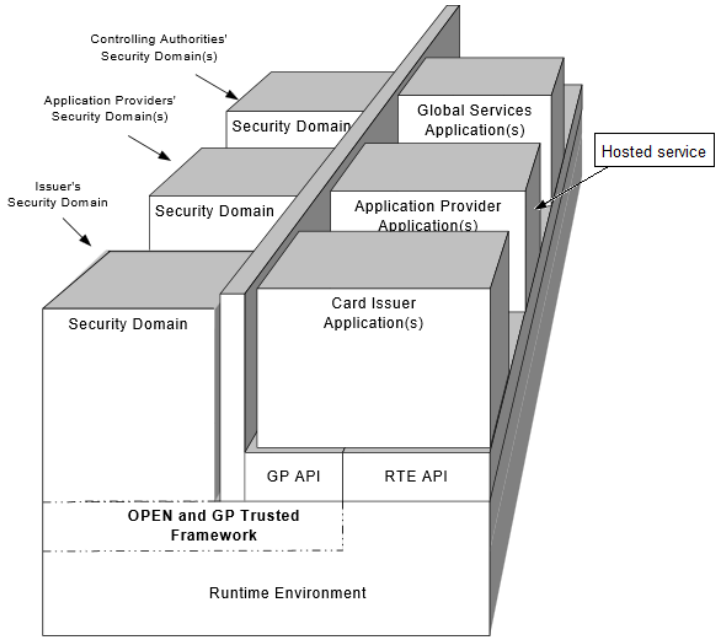

Figure 1.21. *GP architecture of the card*[23]

Specifications of this section address architecture, interactions, interfaces and smart card security, regardless of the underlying technologies (i.e. regardless of the manufacturer or the card OS). Resources in the card section more specifically address smart cards issuers (aiming at hosting third party applications).

22 http://www.globalplatform.org/specificationssystems.asp.
23 Source: Global Platform, Figure 3-1, p. 38 in GP Card Specification V2.2, March 2006.

Figure 1.21 shows hosted services in a dedicated secure execution environment, security domain (SD) including GP standard-compliant API layer for key management, encryption/decryption, signature and access control, support of application portability between GP-compliant cards. We can note the SD intended for the card provider, just like another one dedicated to controlling authority (for keys and certificates management). "OPEN" API provides standardized functions for the interfacing and the lifecycle management of GP-compliant services, including a framework for secure interapplications communication.

OPEN supports the following management functions:

– commands execution;

– application or SD selection;

– logical channel management (i.e. disregarding physical channels);

– command sending;

– card content management;

– content control;

– content setup;

– content removal;

– access control rules for content management;

– security management;

– blocking/unblocking;

– card terminal;

– privileges management;

– traceability and event log.

Specifications thoroughly describe state transitions of the lifecycle, APDU instructions for the management of card content and recommendations for secure communication channel setup (PKI, authentication modes, encryption and decryption, session setup, etc.).

1.3.4.2. *GP device specifications*

The device resources section is dedicated to original equipment manufacturers and more specifically for mobile environments. Specifications are also for developers who interface with peripherals mentioned in this section.

This section addresses two environments:

– SE with its own OS;

– trusted execution environment (TEE) running in the host device memory.

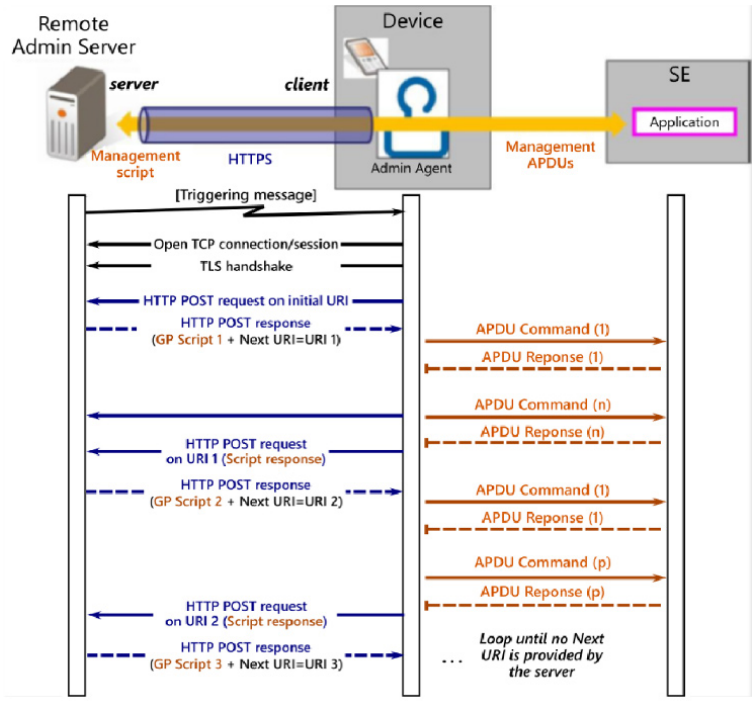

Figure 1.22. *Example of SE remote administration*[24]

24 Source: Global Platform, Figure 6-1, p. 31 in *Secure Element Remote Application Management*, Version 1.0.1, November 2015.

TEE includes secure storage in which functions are executed in an isolated area of the memory space from the OS (separated by firewalls); communication with devices (for example screen, SIM card or SE) uses secure channels, thus securing coding instructions privacy and so-called "trusted" application data.

Applications and tools recommended by GP are related to the architecture, access control and communication interfaces of local (i.e. from the host device) or remote (i.e. from an external server) client applications. For example, Figure 1.22 shows how an administration session request is triggered on the remote admin server demand toward an *Admin Agent* application in the mobile device; then, the mobile sets up a secure HTTPS connection (requiring authentication) with the server to collect APDU instructions to be transmitted to the SE.

1.3.4.3. *GP systems specifications*

This section is devoted for every stakeholder who designs and implements administration systems as well as management systems for smart cards and their content: these specifications define the role and responsibilities of each actors of the ecosystem in a secure infrastructure of multiple application cards' environment and describe message exchange format, protocols and interfaces.

Specifications include a mediator platform called trusted service manager (TSM). The TSM provides a standardized API that allows providers of NFC services hosted in an SE to interface with the management systems of the SE issuers (SEIs) or the MNOs that are handling end-user's subscriptions. GP describes the SOAP secure web services into XML files (WSDL and XSD) intended for the lifecycle management (and audit) of remote applications like (among others):

– service deployment;

– suspension/ lock/unlock;

– update;

– service removal.

Figure 1.23 illustrates the SE content management triggered by an "Actor B" (for example on behalf of the service provider, i.e. TSM) authorized by an "Actor A" controlling the SE (the SEI or the MNO) according to three delegation modes:

– in simple mode, only the SEI is authorized to perform card content management. TSM can verify loading with data authentication pattern (signature);

– in delegated mode, the TSM is authorized to perform card content management by acquiring a prior authorization from the SEI with token management;

– in authorized (dual) mode, the TSM has full access to a dedicated SD by acquiring a prior authorization from the SEI.

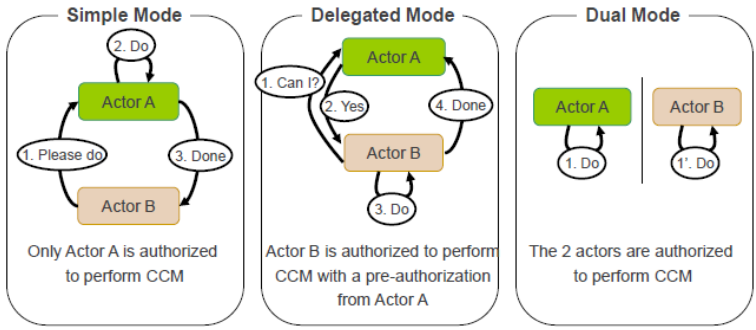

Figure 1.23. *Three modes of content management of SE*[25]

The GP messaging (GPM) system is the standard interface for the remote management of the service lifecycle by the TSM as shown in Figure 1.24. The figure illustrates an example of deployment notification (and function calls) triggered by the TSM to the SE provider as well as to the MNO; then, each can update the service state and proceed appropriate actions. For each actor of this ecosystem, GPM thoroughly describes the end-to-end use cases in the

25 Source: Global Platform, Figure 1-1, p. 26 in *Messaging Specification for Management of Mobile-NFC Services*, Version 1.2, November 2015.

three modes of the SE service management, for each step and events of the lifecycle.

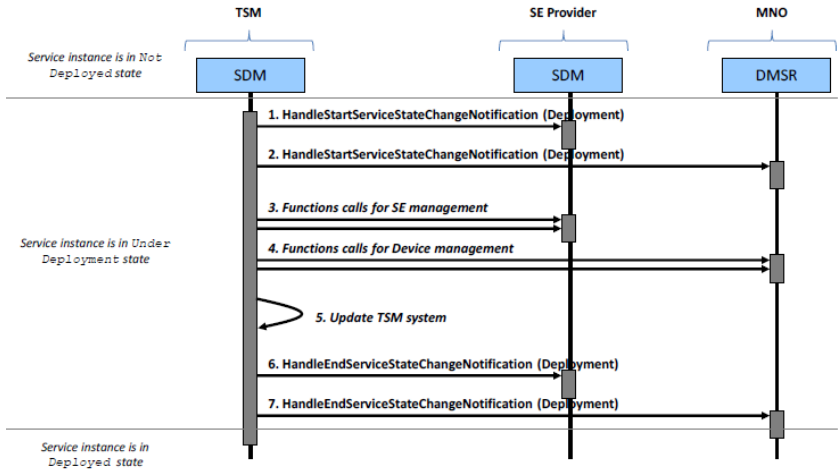

Figure 1.24. *Deployment notification by the TSM*[26]

1.3.5. *SIMAlliance and open mobile API*

SIMAlliance is a non-profit organization involved in mobile device technologies and especially in SIM cards. SIMAlliance aims at easing secure mobile services development and management and simplifying hardware-based device security. Members of the organization are actors of digital security, smart card and mobile services such as Gemalto, Oberthus, Giesecke & Devrient (G&D), Incard and Morpho (Safran).

SIMAlliance gives specifications of its Open Mobile API (OMAPI)[27] standard for mobile applications: OMAPI defines a reader-like interfacing (i.e. smart card reader) with the SE regardless of the configuration (i.e. SIM-SE, eSE or MicroSD); OMAPI was

26 Source: Global Platform, Figure 3-5, p. 131 in *Messaging Specification for Management of Mobile-NFC Services*, Version 1.2, November 2015.
27 See http://simalliance.org/se/se-technical-releases.

adopted as a standard by GP and the API was adapted in order to be compliant with ETSI and GSMA specifications. EMVco also commissioned SIMAlliance to adapt API to contactless transaction payment (NFC Tagify, February 2016[28]).

28 See https://nfctagify.com/simalliance-updates-omapi-secure-element-specification.

2

Developing NFC Applications with Android

NFC applications can be developed in the three NFC operating modes (see section 1.3.2):

– in the *reader/writer* mode, the NFC smartphone acts as an active initiator terminal (generating RF fields as power supply) to read and encode a passive target NFC tag;

– the *P2P* mode allows communication between two NFC-enabled devices: they can exchange data by being both initiator and target in turn (in active/active or active/passive mode) thanks to the LLCP layer (see section 1.3.4.1) and its management of collisions;

– in the *card emulation* mode, the passive NFC-enabled target device behaves as a smart card; in this specific mode, the smartphone can host one or several terminal applications executed in the host OS environment, which can act as the user interface and/or as intermediate application (proxy) to interface with a remote system, and one or several services executed in the SEs secure environment (see section 1.3.2.2).

An NFC smartphone or tablet can act as a reader/encoder of NFC tags in the reader/writer mode. In the P2P mode, the smartphone can also exchange data with another NFC device. In the card emulation

The Art and Science of NFC Programming, First Edition.
Anne-Marie Lesas and Serge Miranda.

mode, the smartphone can act as a contactless NFC smart card (with a SE or an HCE service) to be "read" by an NFC reader.

This chapter introduces mobile applications programming on the Android platform in the three NFC modes, including the HCE mode.

NOTE.– In this chapter, we provide a brief introduction to Android application development with Eclipse. This requires basic knowledge in programming, and especially a good understanding of Java language.

2.1. Introduction to Android programming using Eclipse

Several Integrated Development Environments (IDEs) enable Android application development (in native mode). Among the most commonly used free and open source software, the examples are Eclipse™[1] that we will use with the *Android Development Toolkit*[2] (ADT) plugin, IntelliJ IDEA from JetBrain used with the *idea-Android* plugin[3], Netbeans used with the basic *NBAndroid* plugin[4], etc.

However, Google's official and free IDE, which came out in 2013, is *Android Studio*[5]: this IDE is based on the open source version technology of IntelliJ IDEA. A developer specialized in Android apps coding will rather use Android Studio.

2.1.1. *Android in a nutshell*

Android is the open source operating system (based on a Linux kernel) published by Google. The runtime running Android applications is an optimized multi-task virtual machine for mobile

1 Eclipse download: http://www.eclipse.org/downloads.
2 URL of the Android plugin for Eclipse: https://dl-ssl.google.com/android/eclipse.
3 Download Android plugin for IntelliJ IDEA: http://plugins.jetbrains.com/plugin/1792?pr=.
4 Download Android plugin for Netbeans: http://plugins.netbeans.org/plugin/19545.
5 Download Android Studio: https://android.googlesource.com/platform/tools/adt/idea.

devices (using low memory) called *Dalvik*. The programming language for Android applications is based on Java: once the Java classes are compiled in bytecode, they are converted into a "classes.dex" file to be interpreted and processed by Dalvik. The executable file is a compressed archive type of file (jar[6]) of "APK" (Android PacKage) extension.

NOTE.– Any Android application runs in a dedicated memory area and is given a unique user ID (Linux user ID). Moreover, the files of a given Android application can only be seen (by default) by this application, which enhances security.

The *AndroidManifest.xml* declarations file (in XML format) is required to run the Android application: it stores the application package name, its version, the Android version compatibility, permissions to be granted by the user in order to use the features required by the application, declarations and Google API access keys (retrieved from Google developer's console for Maps, Google Play, Street view, Google Cloud Messaging, Prediction, YouTube, etc.), all activities and filters on activities (for example to declare the main activity that will be executed at launch or an activity executed when detecting an NFC-enabled device).

NOTE.– User interfaces and menus are also described into XML files.

2.1.1.1. *Versions of Android*

Android versions are managed according to two levels:

– Android version (OS installed on smartphones);

– software development kit (SDK) number with which the application is compiled.

Major Android versions are identified by a code name, whereas deployed and SDK versions are numbered (see Table 2.1).

6 Java archive.

Year	Version(s)	Name	SDK(s)	Linux Kernel
2007	1.0	Apple Pie	1	2.6
2008	1.1	Banana split	2	
2009	1.5	Cupcake	3	2.6.27
	1.6	Donut	4	2.6.29
2009–2010	2.0, 2.0.1, 2.1	Eclair	5, 6, 7	
2010–2011	2.2, 2.2.1, 2.3	Froyo	8	2.6.32
2010–2011	2.3, 2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7	Gingerbread	9, 10	2.6.35
2011–2012	3.0, 3.1, 3.2, 3.2.1, 3.2.2, 3.2.4, 3.2.6	Honeycomb	11, 12, 13	2.6.36
	4.0.1, 4.0.2, 4.0.3, 4.0.4	Ice Cream Sandwich	14, 15	3.0.1
2012–2013	4.1, 4.1.1, 4.1.2, 4.2, 4.2.1, 4.2.2, 4.3	Jelly Bean	16, 17, 18	3.0.31, 3.0.4
2013–2014	4.4, 4.4.2, 4.4W.2	KitKat	19, 20	3.0.4
2014–2015	5.0, 5.0.1, 5.1.1	Lollipop	21, 22	3.10
2015–2016	6.0, 6.0.1	Marshmallow	23	
2016	Expected 7.0	Nougat	24	4.4

Table 2.1. *Android versions*[7]

2.1.1.2. *Publishing Android apps on Google Play*

The Google Play Developer Console[8] enables Android apps developers/providers to publish theirs apps (file extension “apk” of a

7 Source: Wikipedia, available at http://fr.wikipedia.org/wiki/Historique_des_versions_d’Android# cite_note-gingerbread-dev-blog-35.
8 Console of the Android Google Play developer: see https://play.google.com.

maximum size of 50 MB) on the Google Play Store. For this, you must have a Google account and pay the one time registration fee (USD 25).

On an indicative basis, the allowed price range for an Android application goes from USD 0.99 to USD 200 for a distribution on U.S. market according Google's location pricing table[9]. Transaction fees charged by Google are 30% of the application retail price.

2.1.2. *Android in Eclipse IDE*

Eclipse is a widely deployed multilanguages and multiple OS open source development framework (Windows, Linux and Mac OSX in 32 or 64 bits versions). A large number of plugins can be installed on Eclipse platform; it allows the development not only of many Java-based frameworks (including J2EE, J2ME, etc.), C/C++, PHP, Python, etc., but also of many tools, for example for collaborative work management and synchronization with Git repositories[10] or Apache Subversion[11], or also testing tools and coding performance tools (for example JUnit for Java[12]), or design/modeling tools, and many others. The first step is then to download and install the Eclipse standard version available at the URL link: https://www.eclipse.org/ downloads.

Eclipse is launched by running the "eclipse.exe" file (by double-clicking on the icon or from the start menu) in the "Eclipse" folder; under Windows, a shortcut can be created on the desktop using the shortcut menu (displayed by a right click on the file icon) as shown in Figure 2.1.

9 See https://support.google.com/googleplay/android-developer/table/3541286.
10 See https://fr.wikipedia.org/wiki/Git.
11 See https://fr.wikipedia.org/wiki/Apache_Subversion.
12 See https://wiki.eclipse.org/Eclipse/Testing.

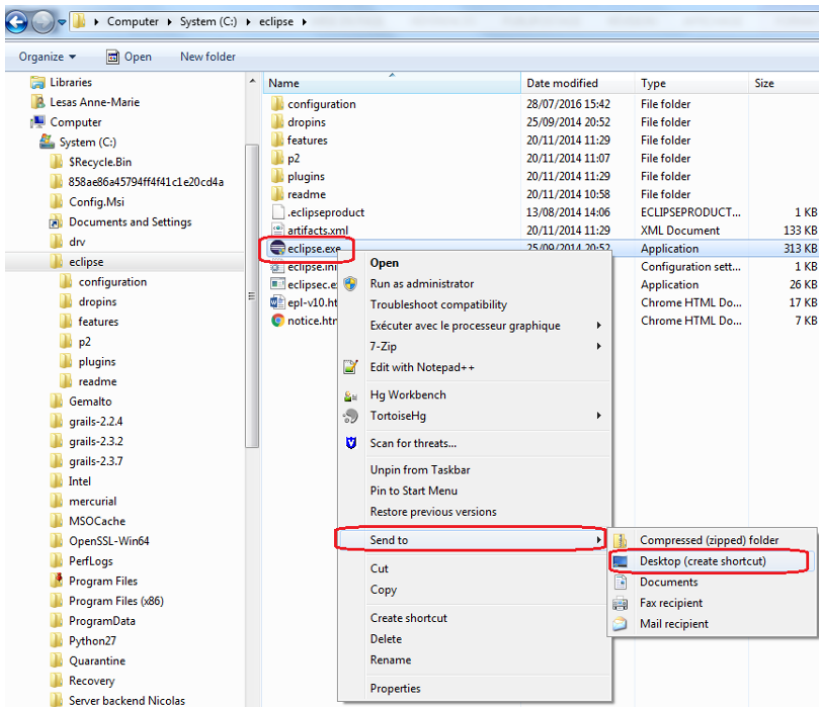

Figure 2.1. *Eclipse: create a shortcut on your desktop*

2.1.2.1. *Android Development Toolkit*

Android Development Toolkit (ADT) SDK can be downloaded from the URL in order to be installed independently: http://developer.android.com/sdk/index.html.

NOTE.– Google provides two SDKs for one version, the open source Android version, fully open, and the version giving access to Google APIs not open and which requires a license agreement (charged or free).

The ADT plugin for Eclipse must be installed via the software installer in the menu *Help/Install new software* (see Figure 2.2). Enter the URL https://dl-ssl.google.com/android/eclipse to install Android plugins (see Figure 2.3) and select the components before clicking the

Next... button on the toolbar. This can take some time, but once you agree on all installations (choose to trust the software in case of system dialogue) and restart Eclipse, new menus and tools are available in Java perspective (see Figure 2.4).

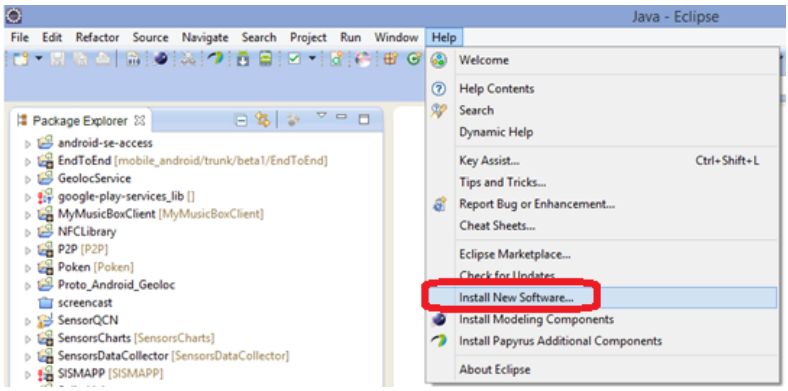

Figure 2.2. *Eclipse: install new software*

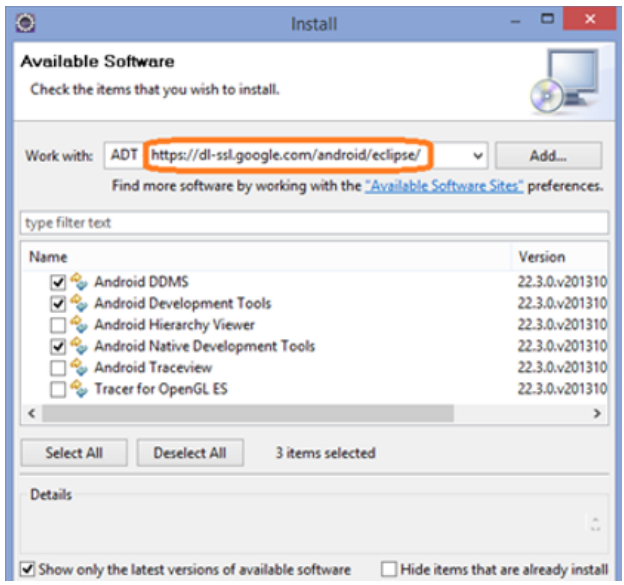

Figure 2.3. *Eclipse: install ADT plugin*

Android SDK manager makes it possible to install (or uninstall) available versions for your Android projects (see Figure 2.5): you may install at least the minimum version with which you want your projects to be compatible.

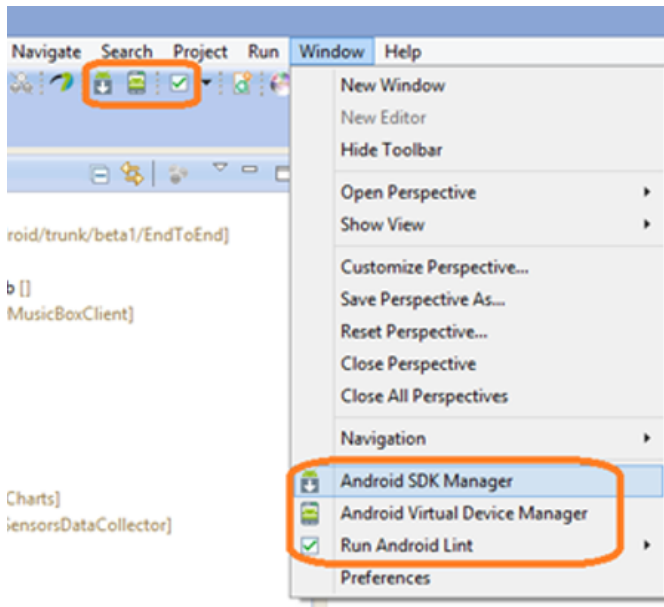

Figure 2.4. *Eclipse: ADT tools and menu*

You are also able to install tools and extras (APIs and Google drivers). Documentation, source code and project examples are located in the installation folder of the Android SDK (e.g. for Windows, usually in the *Program files* folder: C:\Program Files (x86)\Android\android-sdk).

NOTE.– Updates are regularly suggested; in order to only display updates or new versions, you just need to check the *update/new* checkbox, then uncheck *Installed* and *obsolete* (unchecked by default). Installing packages can take a lot of time and bandwidth, it requires opening some connection ports that are not always allowed on some local networks. For this reason, make sure your network configuration is adapted to the installation.

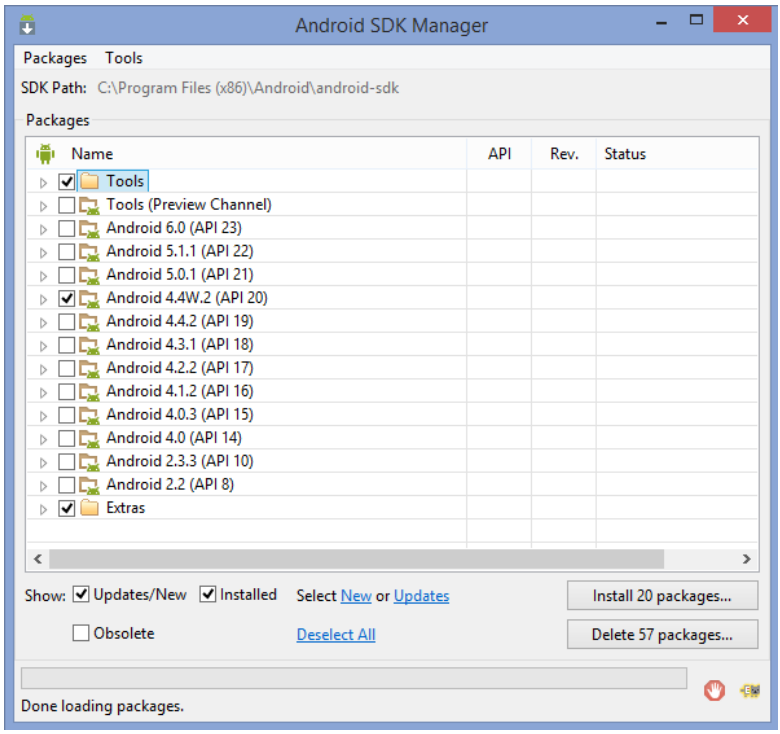

Figure 2.5. *Eclipse: Android SDK manager*

2.1.2.2. *Eclipse working environment window*

Like most IDEs, the main window in Eclipse is a container for internal working windows with a menu bar and one or several toolbars. Internal windows can be displayed, moved, reduced/enlarged or hidden, but we will focus on the default layout and more specifically on the Java perspective; indeed, several perspectives are available, based on installed plugins, for example J2EE, JavaScript, version management and synchronization, and debugging. Figure 2.6 shows the Java perspective (2) with the packages explorer and the projects explorer on the left (1), the source code window in the center (3), displaying or hiding views can be managed from the *Windows* menu (4) and the *LogCat* window (5) (see section 2.1.2.2.1).

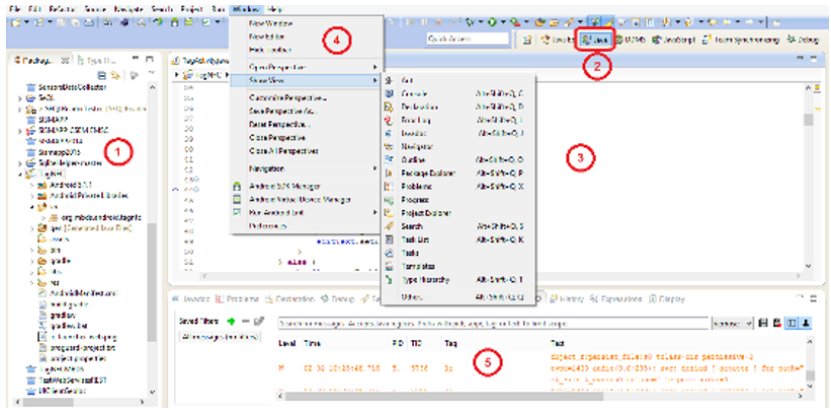

Figure 2.6. *Eclipse: Java perspective*

2.1.2.2.1. LogCat

The Logcat is a console displaying the logs of the mobile connected to the computer's USB port (it requires installing the *Android Composite Android Device Bootloader* (ADB) pilot, provided by Google on Google devices[13]). It is a very useful source of information displaying logs that were voluntarily added in programs (with three possible modes: debugging, info or error). Unhandled exceptions also appear in the Logcat, which indicates their origin in case of processing malfunction.

NOTE.– If the Logcat window is not displayed by default, it can be displayed from the menu "*Windows|show view|other...*" (see Figure 2.6, (4)).

2.1.2.3. *Android project*

This section aims at familiarizing beginners with the Android project based on the example of an existing project.

2.1.2.3.1. Import an existing Android project

From the menu *File|New|Project...|Android Project from existing code*, in the example projects available in the Android SDK, we

13 See http://developer.android.com/sdk/win-usb.html.

can select (for example) the *ApiDemos* project on C:\Program Files (x86)\Android\android-sdk\samples\android-23\legacy\ApiDemos (for Windows environment only) in the import project dialogue window (see Figure 2.7). The option *Copy projects into workspace* makes it possible to copy the project in the user's workspace (the original source project is thus not modified).

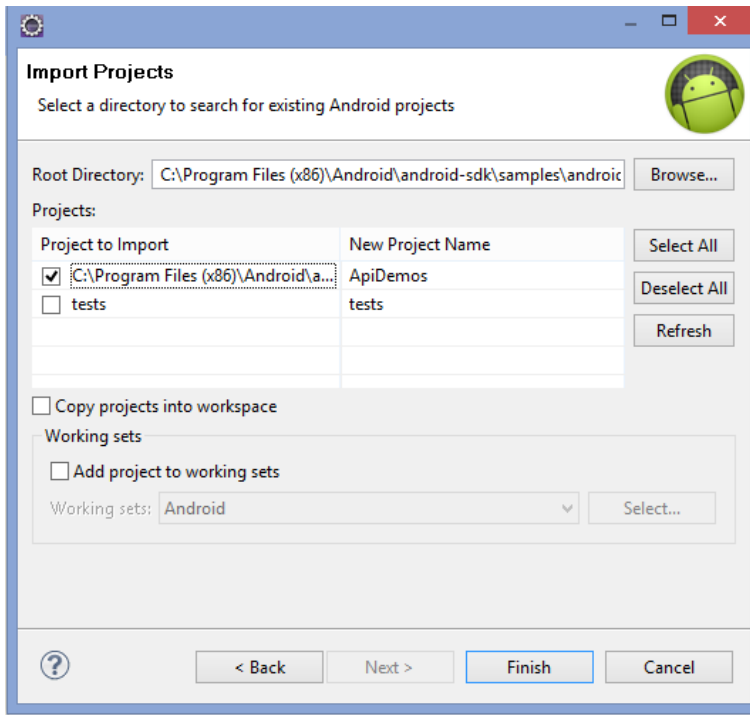

Figure 2.7. *Eclipse: import an existing Android project*

2.1.2.3.2. Android project Properties

Properties can be accessed from the menu *Project|Properties* or from the shortcut menu (right click on the project): the *Android* property section (see Figure 2.8) enables selection of the SDK we wish to compile the application with.

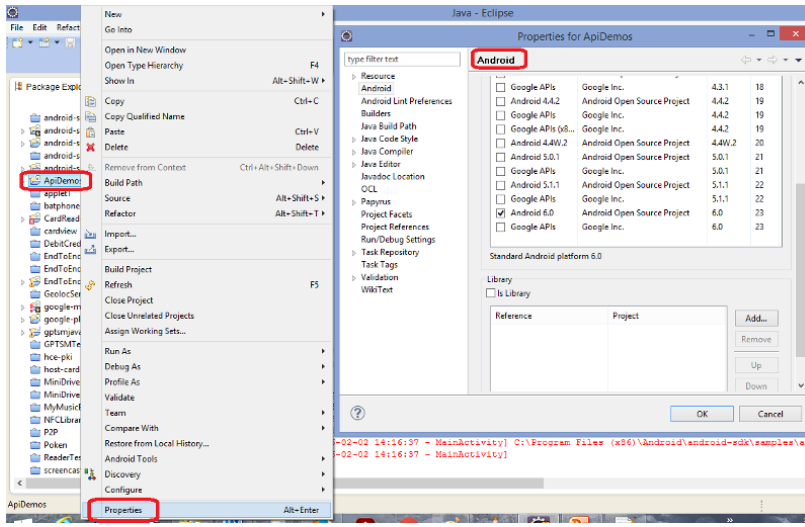

Figure 2.8. *Eclipse: Android project properties*

NOTE.– The Android SDK should match the declarations made into the Manifest file (see AndroidManifest.xml) and we must make sure the coding and the APIs used are compatible, for example to use Google APIs (Maps, Google Cloud Messaging, etc.), we must compile with the proper library version of the Google API.

2.1.2.3.3. Structure of an Android project

The tree view of an Android project (see Figure 2.9) in Eclipse is usually made up of the following folders (under the project root):

– archive files (compatibility libraries and dependencies of Android and external libraries added by the developer);

– the **SRC** folder of the project packages and sources (extension *.java* files);

– the **GEN** folder with java files generated by ADT during the project compilation, for example the *R.java* file ("R" for "Resources") that contains the reference constants of all resources of the project (views, menus, screen and buttons controls, strings, picture files,

themes, etc.) and the *BuildConfig.java* file to manage the log display in debug mode.

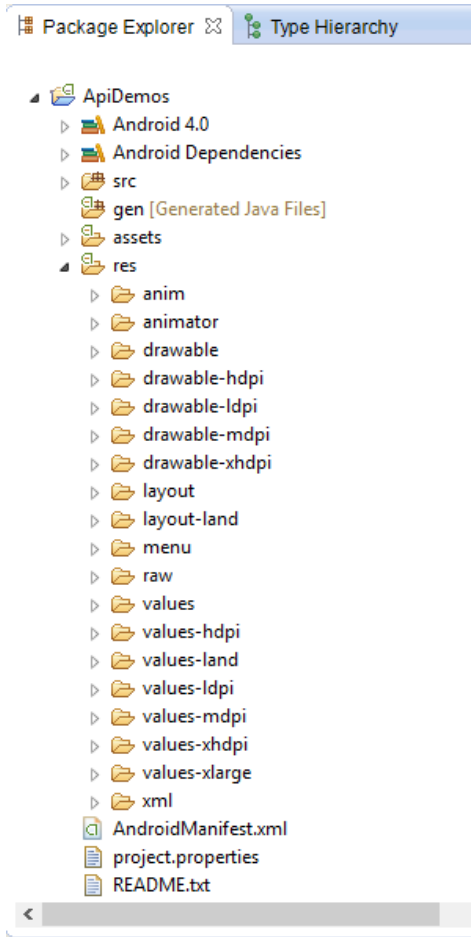

Figure 2.9. *Eclipse: tree view of Android project*

– The **ASSETS** folder: all external resources added by the developer to be used by the application (for example configuration files) are in this folder. Once the project app is deployed on the device, these files are stored in the same unchanged state as during compilation (in read-only) but they can be copied by the application

on the phone in a memory storage area (and then be modified) if needed.

– The **BIN** folder of binary files generated during compilation. That includes the APK archive of the application, which will be deployed on Android devices.

– The **LIBS** folder of external libraries (added and managed by the developer).

– The **RES** folder of resources files: the user interfaces descriptions (in the subfolder **layout**), the **values** subfolder contains styles, strings (including translations), the **menus** can be found in menus subfolder, etc. All these resources are described into *XML* files. Additional descriptions are located in the subfolder **xml** and the pictures files (e.g. icons of the application) are found in the subfolder **drawable-xxx** while media files are in the subfolder **raw**. The subfolders of the Android project can be split according to the layout target resolution and format, or according to the Android version (indicated by an extension in the subfolder name).

2.1.2.3.4. The AndroidManifest.xml file

In an Android project, the application's declarative file (manifest) is to be found at the root of the project (see Figure 2.9): it is the *AndroidManifest.xml* file, whose content in XML format is illustrated in Figure 2.10. Manifest information can also be viewed by theme in the Eclipse interface (see Figure 2.11) as well as the tabs *manifest, application, permissions* and *instrumentation*. In the **manifest** section, general attributes will be entered, such as the name of the application package, the version and the name of this version, the compatibility of the Android SDK number and the features the application needs to access (software or hardware), the section with the **permissions** the user must agree to grant the application when being installed; the **instrumentation** section enables linking classes extending the superclass *android.app.Instrumentation* used for example for automated tests, whereas the **application** section contains the attributes of the application and the activities declarations (including filters and metadata, for example to declare the application launcher's main activity or the activities filtered on NFC technology discovery),

as well as the declaration of some APIs with or without activation keys, for example Google APIs (monitored from the console: Maps, Google Play, Street view, Google Cloud Messaging, Prediction, YouTube, etc.).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.mbds.android.tagnfc"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="18" android:targetSdkVersion="18"/>
    <uses-feature android:name="android.hardware.nfc" android:required="true" />
    <uses-permission android:name="android.permission.NFC" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="org.mbds.android.tagnfc.TagActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="org.mbds.android.tagnfc.NFCReader"
            android:launchMode="singleTop"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.nfc.action.NDEF_DISCOVERED" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:host="www.mbds-fr.org" android:scheme="http" />
            </intent-filter>
```

Figure 2.10. *Eclipse: AndroidManifest.xml file*

NOTE.– To go further in the use of the AndroidManifest file, visit the Android developer website at the URL: http://developer.android.com/guide/topics/manifest/manifest-intro.html.

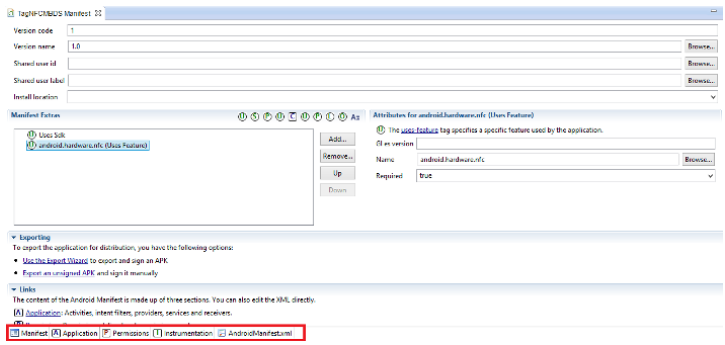

Figure 2.11. *Eclipse: perspectives of the AndroidManifest.xml file*

2.1.2.3.5. Filters intents on activities and services

Filters defined at the *activity* tags section in the Manifest file (sub-category of the tag *application*) are intended to provide the intended use of a given activity.

There are three types of intent filters:

– the **action** filter indicates the type of action executed by the activity, for example for the main activity and the *android.intent.action.MAIN* action, or a filter on NFC technology discovery (see section 2.2.1);

– the **category** filter indicates the category of the activity, for example the application will be launched with the activity declared in the manifest with the filter on *android.intent.category.LAUNCHER*;

– the **data** filter indicates the type of data that the activity expects (i.e. the activity will only be wake-up when this format is detected): a named format, a URL, a domain, etc. For example, the data filter is used to target a specific type of content of NFC tags; a domain, a URL, etc.

2.1.3. *Intents and Android context*

In Android systems, applications run in compartmentalized environments so that they do not impact the behavior of other executions. Intents are part of the exchange mechanism between application resources to inform the system of an intention to access a resource. The *Intent* class describes an action or an event including a context, a component, extras and option flags. The intent also enables sending a *Bundle* (in which parameters or objects are packaged by key–value pairs) between activity instances and their derivatives, for example a background service (i.e. daemon process) or a *Broadcast Receiver* listening to the messages sent through the system and to which it subscribed.

The *Context* object of Android makes it possible to access the application's information and status, but it also enables access to resources available in the application context, as the application is a context in itself, which implements the *Context* interface of Android.

2.1.4. *The Activity class of Android*

Classes inheriting from the *Activity* superclass are Android classes intended for the management of user interactions. An Activity allows the management of the graphical interface (screen): an activity is usually linked to a view (described in XML in *layout* files found in the *res* folder) for the management of user's actions on menu options, buttons, keyboard entries, touch events on the screen or even the voice (*Google Talk*).

NOTE.– An Android application with a user interface (screen) is thus made up of at least one activity; in this case, we can assume that this is the main activity (action filter "MAIN") automatically launched when the user taps on the application icon.

2.1.4.1. *Android activity Lifecycle*

One of the assets of Android development is to release the developer from managing memory resources, especially through the implementation of the *Activity* class.

The lifecycle pattern is presented in Figure 2.12. showing the methods inherited from the superclass that are called at each state of the lifecycle of an activity: these methods can be overridden (*@override* note) to execute the code you implement for this state transition.

NOTE.– When an activity is no longer in the foreground (the user launches another activity), its state (on pause or on stop) can vary according to the flags passed as parameters when the activity was started (see section 2.1.4.2). This has to be taken into account in the way of coding initializations in *callback*[14] methods of the activity's lifecycle so not to launch the same activity several times when one (or several) is already on pause. In the same way, when an activity is started (or resumed), it does not necessarily go through *onCreate* or *onStart* callbacks, but it always goes through the *onResume* method:

14 Callback function when the event occurs.

be careful to properly place initialization and objects release code in the right method.

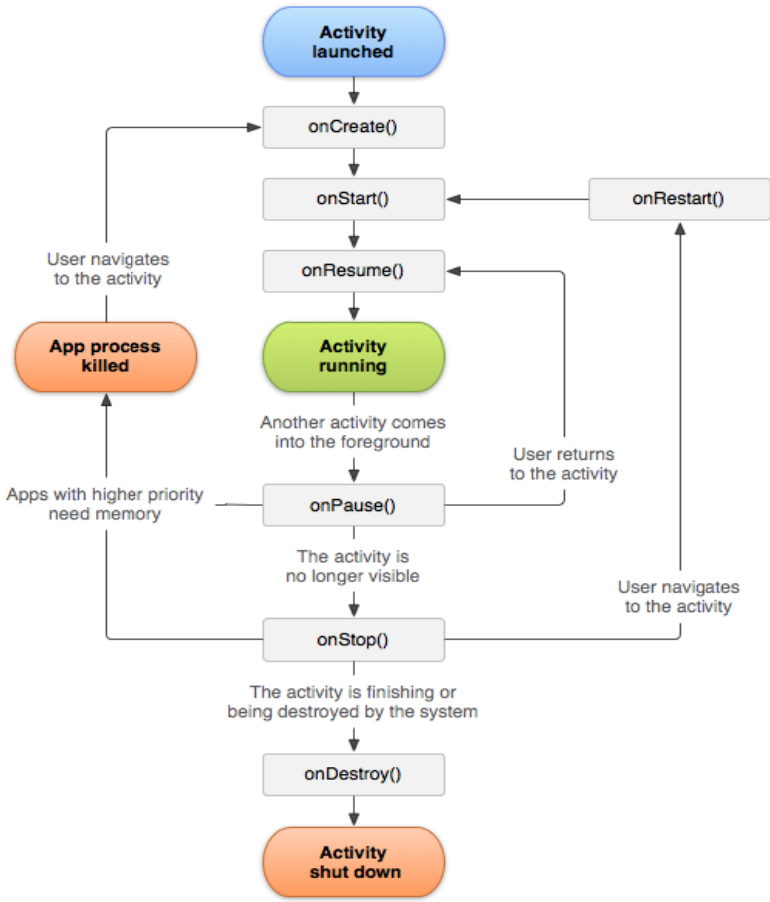

Figure 2.12. *Android: lifecycle of an activity*

2.1.4.2. *Management of an activity*

As illustrated by the code in Figure 2.13, the *startActivity* method launches and puts an activity on the foreground, it takes an intent as parameter (see section 2.1.3) and also an activity class to activate.

```
Intent intent = new Intent(this, TestActivity.class);
startActivity(intent);
```

Figure 2.13. *Android: starting an activity*

NOTE.– Depending on the *intent* and on the parameters passed, the *startActivity* method inherited from the Android context (see section 2.1.3) is also used to launch actions such as e-mail sending, URL display in the browser.

When an activity is called with an expected result in return, using the *startActivityForResult* method will *callback* the *onActivityResult* method of the caller activity. Then, the origin of the request can be identified by a code returned by the activity called joint to a data intent (null value if useless). You just need to override the callback method in order to check the origin of the request and the returned result to carry out the proper processing. For example, the code presented in Figure 2.14 shows how to convert a text into a vocal message and how to start a voice recognition activity with *TextToSpeech*, then collect the text in an array of strings with the *onActivityResult* method.

```
    // Init language (locale)
    if (tts.isLanguageAvailable(locale) == TextToSpeech.LANG_COUNTRY_AVAILABLE) {
    tts.setLanguage(locale);
    }
    // Convert text message into voice
    boolean res = tts.speak(message, TextToSpeech.QUEUE_FLUSH, null)==0;
}
// When click on the speech icone
public void startVoiceRecognition() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    startActivityForResult(intent, REQUEST_CODE);
}
// The result is sent to the activity
@Override
protected void onActivityResult(int requestCode,
    int resultCode, Intent data) {
      if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
          // Get the speech to text result
          ArrayList<String> speech = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
          // Do something with speech
      }
      super.onActivityResult(requestCode, resultCode, data);
}
```

Figure 2.14. *Android: voice recognition (TextToSpeech)*

In order to finish the activity that, for example, was triggered by an action from the user, we just need to call the method *finish()*. In that

case, if another activity is in the top of the stack of calls of the application, it will replace the finished activity on the foreground.

2.1.5. *Android graphical interface: "layout" files*

User interfaces (views) are described in XML files that define controls' positioning (buttons, images, text boxes, etc.) and how they are arranged with one another (below, on the right, on the left, etc.) in a linear container, in absolute position, in a grid, horizontal, vertical, etc.

NOTE.– As illustrated in Figure 2.15, the screen orientation (according to the position detected by inertial sensors: portrait or landscape) can be managed at the activity declaration level in the manifest file.

```
<activity
    android:name=".android.activity.Musics"
    android:label="Musics"
    android:clearTaskOnLaunch="true"
    android:screenOrientation="fullSensor"
    android:theme="@style/Theme.Default" >
</activity>
```

Figure 2.15. *Android: screen orientation of an activity*

2.1.5.1. *Designing a layout*

Two design modes are available from Eclipse: the visual designer ("Graphical layout" tab) that enables to drag and drop components and to graphically place them in the view (see Figure 2.16), and the XML view (see Figure 2.17).

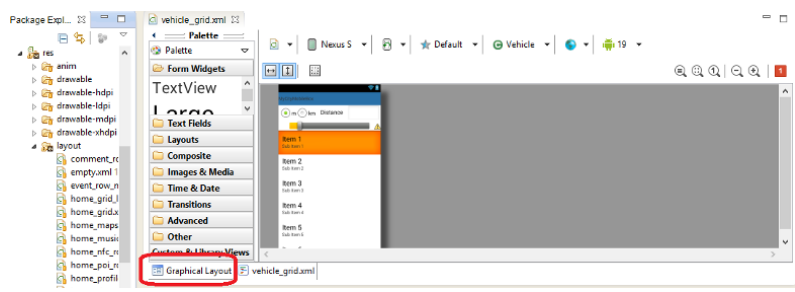

Figure 2.16. *Android: graphical creation of a layout*

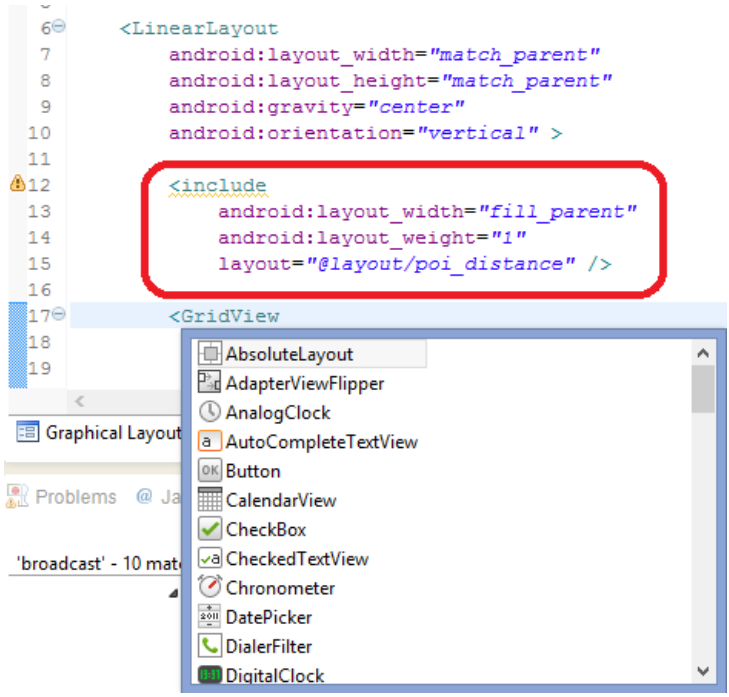

Figure 2.17. *Android: layout inclusion and coding wizard*

NOTE.– A layout can refer to another layout defined in another XML file; this allows reuse of atomic pieces of views. In order to do so, we use the *include* tag referencing the *layout* file that is to be included (see Figure 2.17).

Figure 2.17 also shows the coding wizard menu in Eclipse that is activated by positioning the cursor at the right location and pressing the "Ctrl"+"space" key combination; then, you just need to choose the element you wish to insert from the list of available codes by scrolling (with the arrow keys) and to press "enter" to confirm.

2.1.5.2. *Associating a layout to an activity and handling controls*

A layout can be "set" (method "setContentView") to one or several activities (usually in the "onCreate" method). From this layout,

controls can be differently displayed in order to adapt the targeted devise regardless of the type of display and regardless of the published version of your Android app; for example, it could be installed on a smartphone, on a tablet, on a smart watch, as well as on a smart oven and any other smart object running under OS Android with a display.

NOTE.– *Fragments*[15], introduced since the Honeycomb version 3.0 and Android API 11, ease the modular management of views and the display on screens of different sizes and shapes. In this way, a fragment is a piece of code in charge of controlling a view. It has its own lifecycle, and can thus be dynamically added or removed from the activity based on the events or on the type of display detected during the creation of the activity (*onCreate* method).

2.1.5.3. *Handling the user's actions*

The user's actions are handled at the activity and/or at the layout and/or at the control levels with inherited callback methods overriding or by implementing an interface or a callback object. For example, *onClickListener, onTouchListener, onScrollListener,* etc. The source code provided in Figure 2.18 gives an example on how touch events are triggered and processed after a finger sliding on the screen by managing the *x* and *y* coordinates corresponding to the top left and the bottom right corners of the virtual rectangle shaped by the fingers position compared to the screen landmark at the start of the movement (*ACTION_DOWN)*, during the movement (*ACTION_POINTER_DOWN)*, and when the movement finishes when the user lifts his/her fingers from the screen (*ACTION_MOVE)*.

In the same way, when the user presses "return" and "home" keys of the system, the event is sent to the *onKeyDown* callback that can be overridden in the activity as illustrated in the code in Figure 2.19.

15 See http://www.tutorialspoint.com/android/android_fragments.htm.

```
// Screen touch is handled by
// the onTouch listener
myView.setOnTouchListener(this);
// An activity that implement onTouchListener
// should implement the override onTouchEvent method
@Override
public boolean onTouchEvent(MotionEvent event) {
    // x, y are the current position while moving
    float y = event.getY();
    float x = event.getX();
    switch (event.getAction() & MotionEvent.ACTION_MASK) {
    case MotionEvent.ACTION_DOWN:
    // Do something: Record initial position?
        break;
    case MotionEvent.ACTION_POINTER_DOWN:
    // Do something: Estimate the spacing from initial position?
        break;
    case MotionEvent.ACTION_MOVE:
    // do something: Translate according the final spacing?
        break;
    }
    return super.onTouchEvent(event);
}
```

Figure 2.18. *Android: managing the screen touch*

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
    // Do something
    } else if (keyCode == KeyEvent.KEYCODE_HOME) {
    // Do something else...
    }
    // etc..
    return false;
}
```

Figure 2.19. *Android: screen orientation of an activity*

Actions on menus can also be intercepted (or processed), for example, by overriding the *onMenuItemSelected* method.

2.1.6. *Compiling and testing an Android application*

By default in Eclipse IDE, the Android project compilations are done automatically. This can sometimes be a source of troubles when the coding is in progress (error messages, slowness due to compilation

in the background). You can deactivate this option through the *Project* menu by unchecking the "Build automatically" option of automated compilation.

The "Build project" option is then available in the shortcut menu of the project (right click at the project root in the packages explorer).

Sometimes, a project can contain errors you might not understand: it can be due to binary objects previously generated and which are no longer compatible with the last version of the source code (e.g., removal of layouts). In this case, you should "clean" the generated binary objects from the *Project|Clean* menu (remember to do this every time this kind of problem occurs!).

2.1.6.1. *Launching the application*

Once the application is ready for testing (no compilation errors), it can be launched from the *Run|Run.as|Android Application* menu, which is also available in the project's shortcut menu (see Figure 2.20).

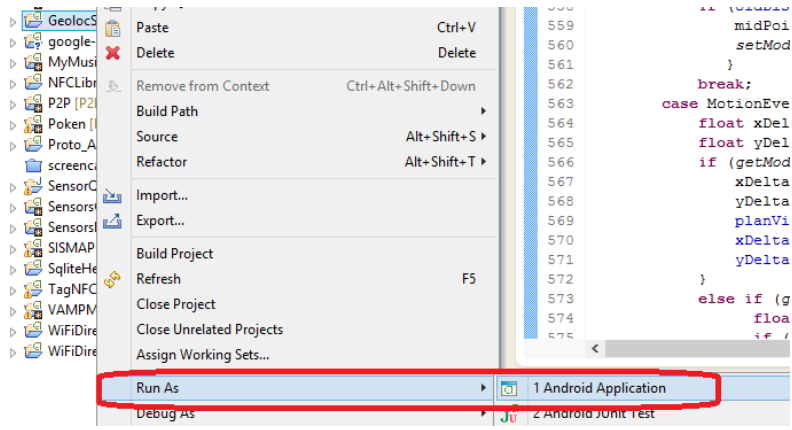

Figure 2.20. *Android: launching the application from the context menu*

Two options for launching Android applications are available from the Eclipse project: ADT provides configurable emulators to make an Android device, but if we have an Android device, it is better to use it

to launch our application project directly on our Android device, because the emulator requires resources allocated to it, and this can drive into our system overhead and result in a very, even extremely slow display.

2.1.6.2. *Using the Android device emulator*

The menu *Window|Android Virtual Device* (AVD) *manager* as well as the associated tool framed in Figure 2.21 enable management of the Android device emulator: the *New* option displays the configuration screen of a new emulator.

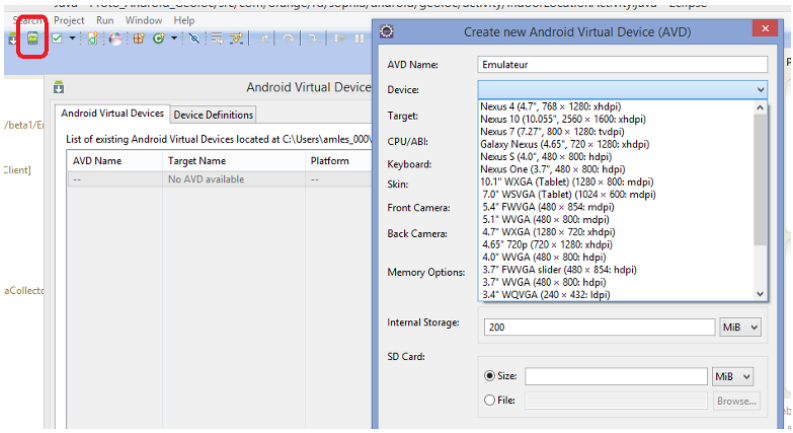

Figure 2.21. *Android: Android Virtual Device (AVD)*

NOTE.– We may note that NFC applications are not easy to test using a virtual device: indeed, you will also need to install an NFC reader emulator as well as an NFC tag or contactless card emulator[16], which is very cumbersome to configure, but this will not ensure the same behaviors as on an NFC-enabled Android device. We thus strongly recommend testing applications with ALL types of devices you wish to deploy your application on.

16 See tools "Android Edition" provided by Open NFC™: http://open-nfc.org/wp/editions/android.

2.1.6.3. *Using an Android device connected to the USB port*

The ADT ADB component detects the Android device connected to the USB port and allows the application launching directly on the plugged device.

NOTE.– We will need to make sure that the installed OS Android version and the features of the device on which we launch the application are compatible with what was declared in the *AndroidManifest* file.

Moreover, for the Android device to be detected by ADB, we need the appropriate ADB driver according the model of the Android device you connect to your system.

NOTE.– Google's drivers comply with a broad range of devices and can be manually installed from the folder *extras* found in the Android SDK directory (for example in usual Windows environments, it can be found at C:\Program Files (x86)\Android\android-sdk\extras\google\usb_driver).

2.2. Implementing NFC with Android

The very first prerequisite to access NFC feature is to enable the NFC option from Android's connectivity parameters.

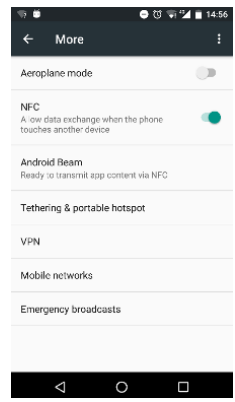

Figure 2.22. *Android: enabling NFC feature in Android system parameters*

2.2.1. *Android manifest declarations*

The use of the NFC hardware feature must be declared. The *required* attribute defines the condition requirements. When set to *true,* makes it possible to only suggest the application to NFC-enabled mobile devices on Google Play (which means that NFC is mandatory to be able to install and use the application).

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

Box 2.1. *Android: declaring NFC feature*

Besides, the NFC feature availability can also be tested in the code.

```
PackageManager manager = this.getPackageManager();
// Check whether NFC is available on device
if (!manager.hasSystemFeature(PackageManager.FEATURE_NFC))
{
    //inform the user that NFC functionality is unavailable
}
```

Box 2.2. *Android: testing NFC availability*

The use of NFC also requires a permission to use declaration that will be notified to the user once he/she installs the application; if the user refuses to grant the NFC permission to use, then the application will not be installed.

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

Box 2.3. *Android: NFC permission declaration*

2.2.2. *Implementing the NFC reader/writer mode*

This section introduces the reader to reading and writing a passive NFC tag.

NOTE.– Mifare technology (from NXP) and contactless smart cards are specific types of tags that are not based on NDEF but on a protocol based on APDUs mentioned in the section on NFC card emulation mode (see section 2.2.4).

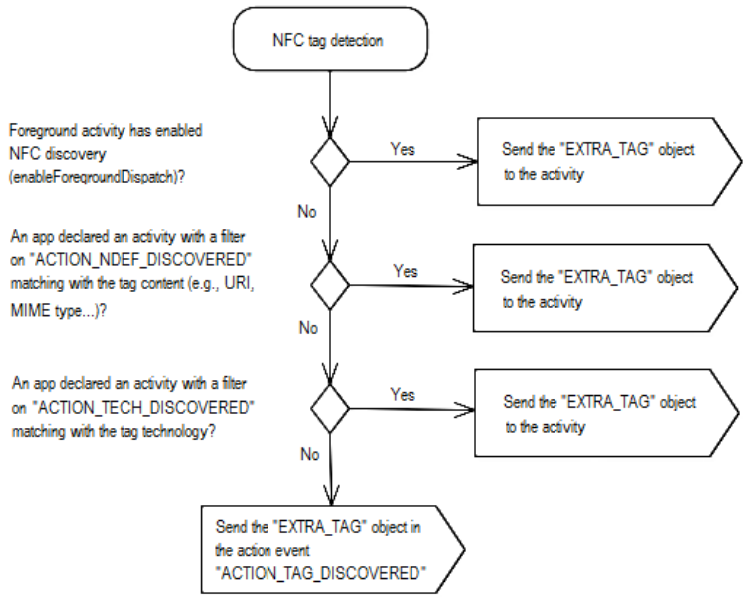

Figure 2.23. *Android: detection of an NFC device management*

In the manifest, the intent[17] filter on NFC events must be declared at the level of the activity (in the application section) implementing the processing of NFC devices detection (for example a tag); Figure 2.23 shows the priority with which the detection of NFC technologies is handled before the event is triggered to the activity.

First, the Android system checks that the current activity (in the foreground) has subscribed to a filter on NFC discovery (dynamically declared into the activity or statically into the manifest) for a matching

17 This word defines the "entity" containing the action to execute, for example the activity.

on the type of content (NDEF), on the technology (according to the list of NFC technologies) or on tag detection; if it is the case, an intent representing the tag (extra tag) is provided to this activity callback method *onNewIntent*.

In the example given in Box 2.4, the *TagActivity* activity declares a filter to detect NFC tags and a filter to detect the NFC technologies that are listed by the developer in a file in XML format located in the *res/xml* folder referred in the "android:resource" attribute of the "meta-data" section in the Android manifest.

NOTE.– It is to be noted that in the example given in Box 2.4, only the first filter is taken into account, the second filter on the NFC technologies discovery is provided here as an example, as well as the "meta-data" tag linked to it.

```
<activity
        android:name=".TagActivity"
        android:launchMode="singleTop"
        …. >
        <intent-filter>
                <action android:name="android.nfc.action.TAG_DISCOVERED"/>
                <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
        <intent-filter>
                <action android:name="android.nfc.action.TECH_DISCOVERED"/>
                <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
        <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/nfc_tech_filter"/>
        ...
</activity>
```

Box 2.4. *Android: filters on NFC discovery declaration example*

NOTE.– The *singleTop* launch mode shows that the activity will only be launched once in the stack of activities calls (for example every time an NFC device is detected, if the activity is already in the stack, it will be placed in the foreground instead of launching twice).

Box 2.5 is an example of a declaration of a list of NFC technologies supported by the application in the XML file (named in the same way it has been declared in the manifest, for example in Box 2.4: “nfc_tech_filter.xml”):

```
<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>
```

Box 2.5. *Android: list of NFC technologies example*

Two simple solutions make it possible to filter the detection of tags on a specific content (i.e. for a specific application):

– referencing a domain (URL format);

– referencing the application.

Filter on a domain:

Box 2.6 shows how to create a filter on a domain. The example placed as comment shows how to use an IP address for potential testing phases before the deployment of a domain name.

NOTE.– This domain might as well be an arbitrary one chosen to identify the tags which will trigger the NFC application: there is no control on the domain validity. Only the tags whose content starts with the name of the entered domain in the filter will activate the NFC activity with no need for action from the user but to put the NFC devices into close proximity.

```
<intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:host="www.mbds-fr.org"
         android:pathPrefix="/application/"
         android:scheme="http"/>
        <!—Another example...                                    -->
        <!--        android:host="134.59.152.102"                -->
        <!--        android:scheme="http"                                    -->
        <!--        android:port="8080"/>                                    -->
</intent-filter>
```

Box 2.6. *Android: NFC discovery filter on NDEF content of NFC tags example*

NFC filter on the application:

To activate the activity, the tag content must be encoded with the same MIME type (or media type[18]) as the one declared in the manifest (see section 2.2.2.2.1), for example by specifying an application package (see Box 2.7). This automatically launches the right application during tag reading, with no intervention from the user.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="org.mbds.android.tagnfc"
 android:versionCode="1"
 android:versionName="1.0" >
....
<activity
 android:name=".TagActivity"
 android:launchMode="singleTop"
 …. >
      <intent-filter>
                <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <data android:mimeType="application/org.mbds.android.tagnfc"/>
      </intent-filter>
</actvity>
...
```

Box 2.7. *Android: example of NFC discovery filter on MIME type NDEF*

18 See Wikipedia: https://en.wikipedia.org/wiki/Media_type.

2.2.2.1. *Reading an NFC tag*

In the activity declared in the manifest for NFC discovery (based on the filter), the instance of the NFC adapter that provides simple interfacing methods with the NFC controller is initialized when the activity is created.

```
NfcAdapter nfcAdapter = NfcAdapter.getDefaultAdapter(this);
if (!nfcAdapter.isEnabled())
{
    //Ask the user to activate the NFC option
}
```

Box 2.8. *Android: initializing the NFC adapter*

NOTE.– It is to be noted that if no NFC controller is installed (in the case of a peripheral device that is not NFC enabled), the NFC adapter instance will be null. You will also need to test that the NFC option was properly activated by the user in the system configuration of the Android device (see Figure 2.22) with the *isEnabled()* method of the adapter, as shown in the source code of Box 2.8.

```
PendingIntent pendingIntent =
    PendingIntent.getActivity(this, 0, new Intent(this,getClass())
    addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
```

Box 2.9. *Android: Initializing the PendingIntent to detect an NFC TAG*

The *PendingIntent* instantiation (see Box 2.9) of the NFC discovery activity with delegation of permission (i.e. discovery of the NFC) aiming at the "EXTRA_TAG" object reception is also done upon creating the activity (method *onCreate*).

The activation of the NFC discovery in the foreground will be initialized at the activity wake up (see method *onResume* overriding

example in Box 2.10) with, as parameters, the pending intent for NFC tag reception with the predefined filter options and the list of technologies to be checked (possibility of null values if already declared in the manifest).

```
    @Override
    public void onResume()
    {
        super.onResume();
        // Note: intentFiltersArray and techListsArray can be null objects
        nfcAdapter.enableForegroundDispatch(this, pendingIntent,
intentFiltersArray, techListsArray);
    }
```

Box 2.10. *Android: activating the foreground NFC detection*

In return, the NFC discovery in the foreground should be deactivated (see Box 2.11) when the activity is paused (no longer in the foreground).

```
    @Override
    public void onPause()
    {
        super.onPause();
        nfcAdapter.disableForegroundDispatch(this);
    }
```

Box 2.11. *Android: deactivating NFC foreground detection*

The recovery of "EXTRA_TAG" object is delegated to the NFC activity through the *onNewIntent* callback method called, for example, when the activity is started in SINGLE_TOP mode upon the call of *startActivity*. Then, we just need to test whether the intent

filter matches the discovery of an NFC-enabled device and to process the intent (e.g., in the processNfcIntent method as illustrated in Box 2.12).

```
    @Override
    public void onNewIntent(Intent intent)
    {
     String action = intent.getAction();
     if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action) ||
          NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action) ||
          NfcAdapter.ACTION_TECH_DISCOVERED.equals(action))
     {
          //Method processing the NFC tag
          processNfcIntent(intent) ;
     }
}
```

Box 2.12. *Android: Recovery of the intent of the NFC TAG*

The code provided in Box 2.13 shows how to get the information and the NDEF content of an NFC tag from the NFC activity.

```
    public void processNfcIntent (Intent intent)
    {
          //Info on the tag
          Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
          byte[] id =tag.getId();
          String[] technologies = tag.getTechList();
          int content = tag.describeContents();
          Ndef ndef = Ndef.get(tag);
          boolean isWritable = ndef.isWritable();
          boolean canMakeReadOnly = ndef.canMakeReadOnly();
          //Messages recovery
          Parcelable[] rawMsgs = intent.getParcelableArrayExtra(
          NfcAdapter.EXTRA_NDEF_MESSAGES);
          NdefMessage[] msgs;
          //Loop on records
          if (rawMsgs != null)
          {
               msgs = new NdefMessage[rawMsgs.length];
               for (int i = 0; i < rawMsgs.length; i++)
```

```
            {
                msgs[i] = (NdefMessage) rawMsgs[i];
                NdefRecord record = msgs[i].getRecords()[i];
                byte[] idRec = record.getId();
                short tnf = record.getTnf();
                byte[] type = record.getType();
                String message =record.getPayload().toString();
                //Use?
                //Let Android choose the default application if URI type…
                if (Arrays.equals(type, NdefRecord.RTD_URI))
                {
                    Uri uri = record.toUri();
                    Intent i = new Intent(Intent.ACTION_VIEW);
                    i.setData(uri);
                    startActivity(i);
                }
            }
        }
        else
        {
            //Unknown tag type, test of a recovery of the hexadecimal content?
            byte[] empty = new byte[] {};
            NdefRecord record = new NdefRecord(NdefRecord.TNF_UNKNOWN,
                empty, empty, empty);
            NdefMessage msg = new NdefMessage(new NdefRecord[] {record});
            msgs = new NdefMessage[] {msg};
            …
        }
        //Process information...
    }
```

Box 2.13. *Android: reading of the NFC TAG*

Information such as the UID[19] and the type of technology are then available from the *Tag* object, as well as information on NDEF content (remember to check that the tag is formatted in case of a detection which is not filtered on NDEF content). The example shows

19 Write-protected Universal Unique Identifier of the tag.

how the developer can loop on a list of NDEF messages (in the case of several records) and how he/she can process the content (here, for example, to display a URI), according to the RTD type (see section 1.3.3.3.2).

2.2.2.2. *NFC tags writing*

NOTE.– The initialization of the NFC adapter of the pending intent for NFC discovery, activation and deactivation of NFC discovery in the foreground as well as the intent reception are coded in the same way, whether it be on reader or writer mode (see section 2.2.2.1).

2.2.2.2.1. Building of an NDEF message

The building of an NDEF message is illustrated in Box 2.14 with the example method *createMyNdefMessage* accepting a *String* parameter for the text message and another *String* parameter for the wanted MIME type, and returning a *NdefMessage* object.

```
public NdefMessage createMyNdefMessage(String text, String mimeType)
{
    //Message type MIME
    NdefMessage msg = new NdefMessage(NdefRecord.createMime(
            mimeType, text.getBytes())

            // Another method doing same...
            /**NdefRecord(NdefRecord.TNF_MIME_MEDIA,
                    mimeType.getBytes(), new byte[0],
                    text.getBytes())**/

            //Message type application
            //for example text = "com.mbds.android.tagnfc"

            //enables the application launching when detecting the tag
            /**,NdefRecord.createApplicationRecord(text)**/

            //Message type URI :
            //for example text = "http://www.mbds-fr.org"
            /**,NdefMessage(NdefRecord.createUri(
                    NdefRecord.createUri(Uri.encode(text))**/
    );
    return msg;
}
```

Box 2.14. *Android: Building the NDEF message*

2.2.2.2.2. Writing an NDEF message on the NFC tag

NOTE.– Before writing on a tag, the tag must be read, meaning that it must be recovered in the same way as to read the tag in the *processNfcIntent* method described earlier (see section 2.2.1), after being brought into proximity with the Android device (usually, the NFC reader is on the back of the device).

The code suggested in the Box 2.15 below gives an example of a method accepting an NDEF message as a parameter to write on the tag, as well as the "Tag" object which was detected: here, we assume that the tag is compatible with NDEF (if not, the NDEF object will be null, but we can still try to format it), we then need to connect and test whether it is locked in writing or not, and check that the size of the message to write is not greater than the tag capacity. Then, writing is made possible.

```
public static boolean writeTag(final NdefMessage message, final Tag tag)
{
    try
    {
        int size = message.toByteArray().length;
        Ndef ndef = Ndef.get(tag);
        if (ndef == null)
        {
            //Tags requiring formatting?
            NdefFormatable format = NdefFormatable.get(tag);
            if (format != null)
            {
                try
                {
                    format.connect();
                    //Formatting and message writing:
                    format.format(message);
                    //Example of tag locked in writing:
                    //formatable.formatReadOnly(message);
                    format.close() ;
                    return true;
                } catch (Exception e) {
                    return false;
                }
            } else {
```

```
                    //format == null
                    return false;
                }
            } else {
                //ndef!=null
                ndef.connect();

                if (!ndef.isWritable())
                {
                    return false;
                }

                if (ndef.getMaxSize() < size)
                {
                    return false;
                }
                ndef.writeNdefMessage(message);
                ndef.close();
                return true;
            }
        } catch (Exception e) {
            return false;
        }
    }
```

Box 2.15. *Android: writing an NDEF message on the tag*

NOTE.– For a version prior to Ice Cream Sandwich 4.0 (API 14) that can be tested as shown in the example of Box 2.16, the creation of an NDEF message differs slightly, as illustrated by the code suggested in the example *createMydefRecord* method of Box 2.17. It is to be noted that the same character set must be used for the encoding and the decoding of the NDEF message.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH)
{
        //Process later version...
} else {
        //Process prior version...
}
```

Box 2.16. *Android: testing a version*

```
public NdefRecord createMyNdefRecord(String message)
{
    byte[] langBytes = Locale.ENGLISH
.               getLanguage().getBytes(Charset.forName("US-ASCII"));
    byte[] textBytes = message.getBytes(Charset.forName("UTF-8"));
    char status = (char) (langBytes.length);
    byte[] data = new byte[1 + langBytes.length + textBytes.length];
    data[0] = (byte) status;
    System.arraycopy(langBytes, 0, data, 1, langBytes.length);
    System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.length);
    message=message.trim();
    NdefRecord ndefRec;
    //RTD to choose according to the content type
    ndefRec =new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
                NdefRecord.RTD_TEXT, new byte[0], data);
                //NdefRecord.RTD_URI, new byte[0], data);
                //NdefRecord.RTD_SMART_POSTER, new byte[0], data);
    …
    return ndefRec;
}
```

Box 2.17. *Android: creating an NDEF record*

2.2.3. *Implementing the NFC P2P mode with Android*

It is possible, from the Android Ice Cream Sandwich version 4.0 (API 14), to share e-mail, contacts and favorites with another Android mobile phone, thanks to Android Beam, which is a functionality inherent to Android.

Not only does *Beam* make it possible to emulate the way an NFC tag acts with the mobile and to share NDEF messages (text, URL, SMS, telephone, business card, signature, Wi-Fi parameters, applications, actions, etc.), if another NFC-enabled mobile is brought into proximity, it detects the device as if it was an NFC tag, and it also enables sharing of media files from the mobile's storage memory, from the Android 4.1 Jelly Bean version (API 16) onwards.

We will thus need to declare the permission in order to access the external storage memory in the manifest as shown in Box 2.18.

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

Box 2.18. *Android: permission declaration for access to the storage memory*

Contrary to the reader/writer mode, which enables starting an activity without the need for the user to launch the application according the NFC filter declared in the manifest for NFC discovery, Android *Beam* only works when the activity is in progress (in the foreground): there is no filter on the activity, but the "Beam" option must be activated in the Android mobile's system parameter and Beam availability must be programmatically tested, as the code in box 2.19 shows.

```
If (!nfcAdapter.isNdefPushEnabled())
 {
        //Ask user to activate Beam option
 }
```

Box 2.19. *Android: check the beam option is activated*

2.2.3.1. *Using Android beam to send NDEF messages*

In order to send NDEF messages, you just need to implement the *setNdefPushMessage* of the NFC adapter (see Box 2.20):

```
// NFC adapter recovery
NfcAdapter nfcAdapter = NfcAdapter.getDefaultAdapter(getApplicationContext());
//Use of the previous method (tags writing) to adapt according to the RTD
NdefMessage msg = createMyNdefMessage("Hello World!", "text/plain");
//Send message on the P2P device:
//the user must tap to validate the transfer
nfcAdapter.setNdefPushMessage(msg, this);
```

Box 2.20. *Android: sending an NDEF message in P2P with beam*

2.2.3.2. *Using Android beam with callbacks*

Another more dynamic method consists of using callbacks to trigger the NDEF message sending when the proximity of the NFC device is detected.

```
public class TagActivity extends Activity implements CreateNdefMessageCallback,
 OnNdefPushCompleteCallback
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        NfcAdapter nfcAdapter =
            NfcAdapter.getDefaultAdapter(getApplicationContext());
        //Subscription for the callback of message creation
        nfcAdapter.setNdefPushMessageCallback(this, this);
        //Subscription for the callback for the end of message sending
        nfcAdapter.setOnNdefPushCompleteCallback(this, this);
    }

    @Override
    //This method is used when the NFC P2P device is detected
    public NdefMessage createNdefMessage(NfcEvent event)
    {
        //Use the method created previously:
        NdefMessage msg = createMyNdefMessage("Hello World!", "text/plain");
        return msg;
    }

    @Override
    //This method is used once the message has been received
    public void onNdefPushComplete(NfcEvent event)
    {
        //Notify the user
    }
}
```

Box 2.21. *Android: Android beam implementation with callbacks*

This implementation illustrated in box 2.21 presents several advantages, among them the fact that we know about the beginning and the end of the exchange.

2.2.3.3. *File transfer with Android beam*

The NFC file transfer feature using beam is a push method that takes the URI of the file to transfer as parameter. First, we will need to instantiate a File object with the directory hosting the file, and then another with the File object and the file name (in string of characters format) indicated in the code provided in Box 2.22.

```
//Directory hosting the file to transfer
File myFolder = Environment.getExternalStoragePublicDirectory(
        //Picture folder
        Environment.DIRECTORY_PICTURES);
        //Photo folder
        //Environment.DIRECTORY_DCIM);
        //Video folder
        //Environment.DIRECTORY_MOVIES);
        //Music folder
        //Environment.DIRECTORY_MUSIC);
        //Downloads folder
        //Environment.DIRECTORY_DOWNLOADS);
        //Documents folder
        //Environment.DIRECTORY_DOCUMENTS);
         //Or the directory named...
        //Environment.getExternalStorageDirectory()+"/MyDirectory/";

//Name of the file to transfer with its extension
String myFileName = "monFichier.extension";
//Full File object
File myFile = new File(myFolder, myFileName);
//Make sure the file is accessible by everybody in reader mode
myFile.setReadable(true, false);
//Make the file available by Beam for the NFC transfer.
nfcAdapter.setBeamPushUris(new Uri[]{Uri.fromFile(myFile)}, this);
```

Box 2.22. *Android: file transfer in P2P with beam*

NOTE.– With *Beam*, the user sending the message/file to transfer is invited to "tap" the screen to start the transfer (when source and target devices are brought into proximity for the time of the transfer).

A notification on the transfer progression appears in the notifications area of the target device, and a notification shows the transfer completion: the user receiving it is then asked to "tap" the screen to display the content that was transferred and downloaded in the target device.

2.2.4. *Implementing the NFC card emulation mode with Android*

The specificity of the NFC card emulation mode is the addressing of type 4 tags (see section 1.3.3.3.1) and a communication based on the APDU protocol for smart cards inherited from JavaCard™ environments. Services are hosted in the chip and executed in the smart card or SEs OS. The APDU interface is managed (implemented) in the (external) entry point of JavaCard™ classes inheriting from the Applet superclass in the *process* method. The method receives as a parameter an APDU command (see Figure 2.24) and processes it.

The APDU response is then sent back (in return or on demand with another APDU) to the client application, which establishes a session of communication with the smart card.

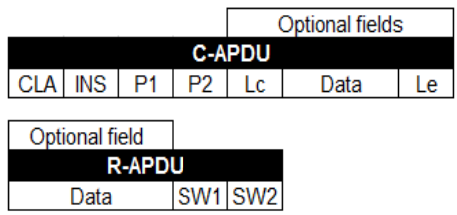

Figure 2.24. *APDU structuring*

The reading of smart cards or the SE communication in the NFC card emulation mode is a master–slave type of communication where the initiator is a client application (master), which is external to the electronic chip, sending APDU commands and receiving APDU responses from the service hosted in the chip (slave).

2.2.4.1. *"SELECT" APDU command: selecting the application*

In order to communicate with the chip's service, you must know its AID identifier[20]: the size of the AID is between 5 and 16 bytes; it is made of the supplier's ID (Registered ID) managed by regulators, and of the proprietary application extension (Proprietary application Identifier eXtension) managed by the service provider.

The setup of a session of communication with a service embedded in the card is initialized with the *SELECT* instruction command of the APDU standard whose payload is filled with the AID of the targeted service. For example, Box 2.23 shows the construction of the *SELECT* APDU instruction to initialize the communication with a service whose AID (fictitious) is "F00000000000000000001": the expected response will only include the status code (success or failure) of the command's execution.

```
byte[] SELECT = {
        //CLA Class: Moneo, Mastercard, Visa© cards…
        (byte) 0x00,
        //INS: Instruction SELECT
        (byte) 0xA4,
        //P1: Parameter 1
        (byte) 0x04,
        //P2: Parameter 2
        (byte) 0x00,
        //Lc command length: 10 bytes
        (byte) 0x0A,
        //Payload: AID
        (byte) 0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01
        //Le Length of the expected response: 0 bytes
        (byte) 0x00,
};
```

Box 2.23. *Setting up the SELECT instruction of the APDU standard*

20 Applet Identifier: In JavaCard™, each applet is identified thanks to its AID (naming convention compliant with ISO 7816 standard).

2.2.4.2. *Communicating with an NFC tag in card emulation mode*

The Android class *IsoDep* enables operations in the NFC card emulation mode by using the protocol specified in the ISO 14443-4 standard (see section 1.3.4).

Box 2.24 shows how, after reading (see section 2.2.2.1) an ISO 14443-4 tag type of NFC smart cards (according to the technology determined in the filter, for example, see section 2.2.2), the Android application becomes a terminal that can send commands and receive APDU responses from the card (or its emulation) thanks to an *IsoDep* object.

```
IsoDep card = IsoDep.get(tag);
card.connect();

//Send an APDU command
byte[] result = card.transceive(SELECT);

//Check response
if (!(result[0] == (byte) 0x90 && result[1] == (byte) 0x00)) {
//Manage error
        ...
}
...
card.close();
```

Box 2.24. *Communicating with an NFC tag in card emulation mode*

2.2.4.3. *Communicating with a SE*

In section 2.2.4.2, we saw how we can implement communication with an NFC smart card tag type when read by the Android application after it has been brought into the proximity of the NFC-enabled Android device. This application can sends APDU commands to an Applet installed in the tag's chip and receive a response back. When the Applet is hosted in the chip of an SE inside a mobile, an application running in the host OS may have to communicate with the SE services. It can be an end-user interface, or an interface for a remote system: for example, in the case of a wallet-type application

that enables the user to check his/her last transaction, or a proxy application for the SE services lifecycle remote management.

NOTE.– As access to the SE is restricted, communication with SE services requires privileges and security levels specific to the implementation of SEI, of the service and of the functions being accessed. Here, we only focus on the communication implementation, but we must stress the obvious importance of security, without which the SE would not be a SE.

Communication with SEs is made through an API (external library) whose availability depends on the Android version, the device model and the SE configuration.

2.2.4.3.1. The "nfc_extras" API of Android

The Android "nfc_extras.jar[21]" library, which requires a declaration on the manifest folder (Box 2.25), contains the singleton *NfcAdapterExtras* (see Box 2.26) and provides static methods to access and exchange APDU messages with Applets of the eSE represented by the *NfcExecutionEnvironment* class.

```
…
<uses-permission android:name="android.permission.NFC"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>
<uses-permission android:name="android.permission.NFC"/>
<uses-permission android:name="com.android.nfc.permission.NFCEE_ADMIN"/>
...
<application…>
        …
        <uses-library android:name=
                "com.google.android.nfc_extras" android:required="true"/>
</application>
```

Box 2.25. *Declarations in the AndroidManifest.xml file*

21 Sources available here: http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.3.7_r1/com/android/nfc_extras/NfcAdapterExtras.java.

NOTE.– The *nfc_extras* API available on the Ice Cream Sandwich (ICS) of Android of a lower level than *IsoDep* also enables the activation and deactivation of the NFC card emulation mode.

```
NfcAdapterExtras adapterExtras =
    NfcAdapterExtras.get(NfcAdapter.getDefaultAdapter(context));
NfcExecutionEnvironment nfceEe =
    adapterExtras.getEmbeddedExecutionEnvironment();
nfcEe.open();
byte[] response = nfcEe.transceive(APDUcommand);
nfcEe.close();
```

Box 2.26. *Example of use of the "NfcAdapterExtras" class*

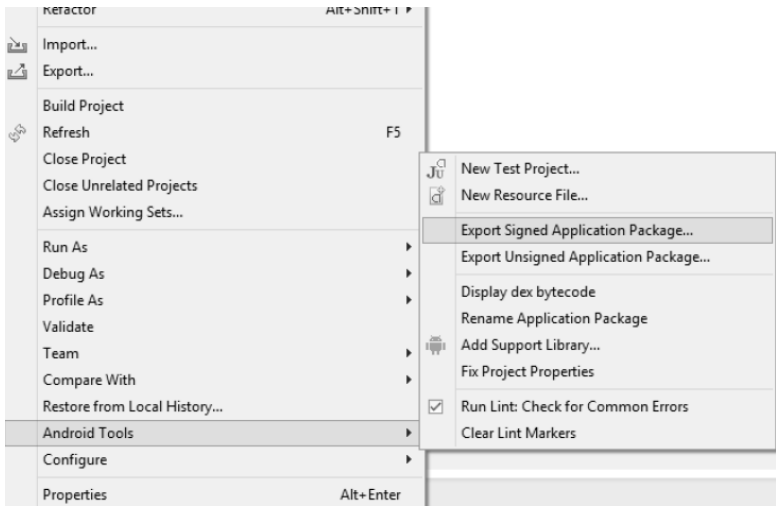

Figure 2.25. *Exporting a signed application*

Since version 4.0.4 of Android ICS (API 15), the application signature is not sufficient: Android uses a certificate generated[22] with a private key in order to (self) sign the applications (see Figure 2.25). An Android application accessing the SE (i.e. Google Wallet) was

22 See signature of applications on the Android website for developers: http://developer.android.com/tools/publishing/app-signing.html.

declared in the "nfcee_access.xml" file located in the "/etc" system directory of the smartphone (see Box 2.27), which requires "superuser[23]" access (special user account with elevated privileges granted on the system).

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
<-- ! Application signature generated with Keytool -->
<signer android:signature="XXXXXXXXXXXXXXXXX">
<package android:name="org.mbds.android.tagnfc">
</package></signer>
 ….
</resources>
```

Box 2.27. *Declarations on the nfcee_access.xml file*

2.2.4.3.2. SmartCard API (OMAPI)

SmartCard API for Android is an implementation of the *Open Mobile API* standard (OMAPI, see section 1.3.6) whose *seek-for-android* project is published on github[24]; the API enables Android mobile applications to communicate with an SE (in card emulation mode). This API can be added to the list of packages in Android SDK manager (see section 2.1.2.1): to do so, we must add the downloading website through the menu *Tools|Add-on Sites|User Defined Sites* in the SDK manager, and add the URL of Box 2.28 where XX must be replaced by the Android SDK number with which the project will be compiled (see Figure 2.26).

```
http://seek-for-android.googlecode.com/svn/trunk/repository/XX/addon.xml
```

Box 2.28. *Downloading URL for SmartCard API*

The OMAPI package then appears in the SDK list (see Figure 2.27).

23 On Android, the superuser is called "Root". A method with comments is suggested on the CNET forum at this URL: http://forums.cnetfrance.fr/topic/193517-comment-rooter-android--methode-universelle-de-root.
24 Seek for Android project: https://github.com/seek-for-android.

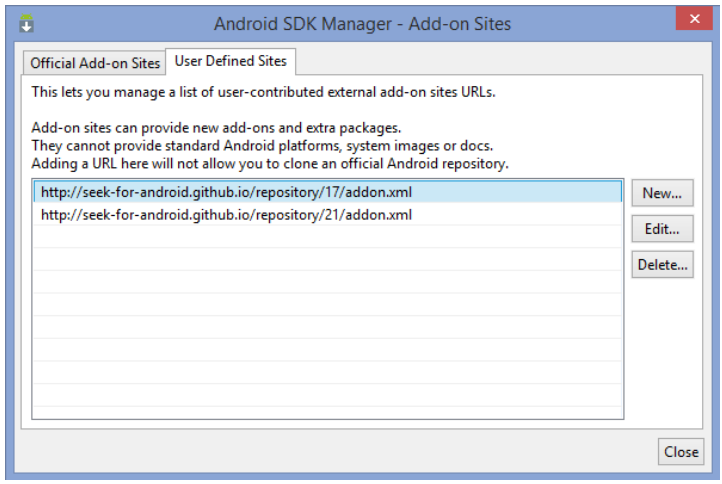

Figure 2.26. *Adding packages in Android SDK manager*

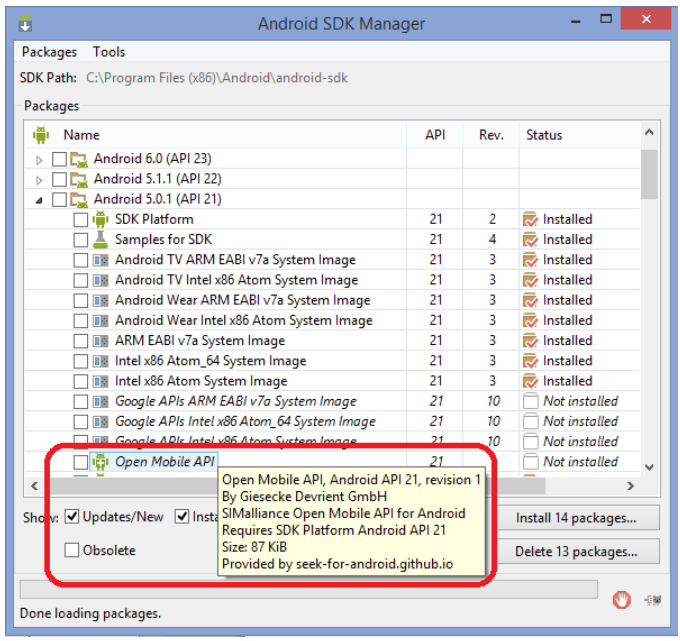

Figure 2.27. *Open Mobile API package*

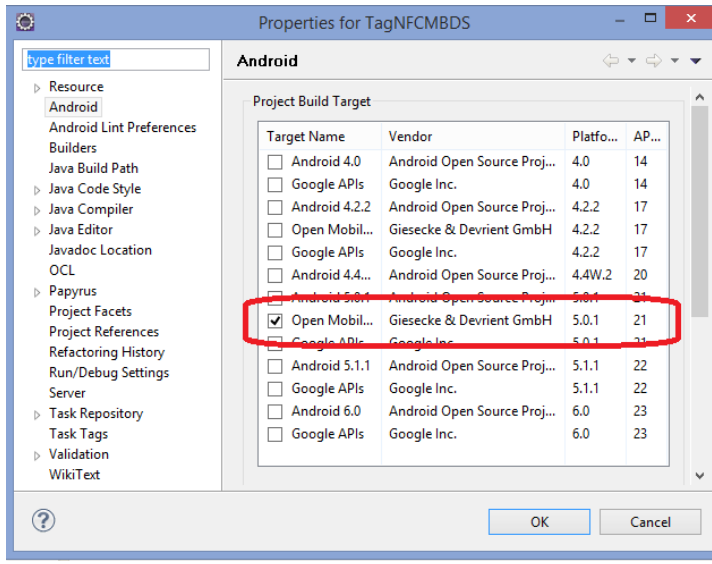

Figure 2.28. *Properties of the Android project: target version*

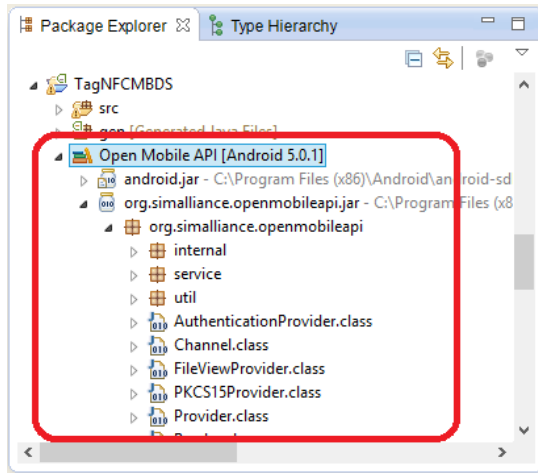

Figure 2.29. *Open Mobile API library*

Once the *Open Mobile API* package is installed, we can select it in the target version of the Android project properties (see Figure 2.28)

to access *SmartCard API* interfaces and compile the project with the requested libraries (see Figure 2.29).

The use of *Open Mobile API* must be declared in the *AndroidManifest* file, as well as the access permission to access the SE (see Box 2.29).

```
<uses-library android:name="org.simalliance.openmobileapi" android:required="true"/>
<uses-permission android:name="org.simalliance.openmobileapi.SMARTCARD"/>
```

Box 2.29. *Declarations of the Open Mobile API (OMAPI)*

Implementing the SE connectivity in Android Activity[25]:

Box 2.30 shows how access to the SE is done through a background task service. When the service is available, it notifies the activity by implementing the "serviceConnected" method of the "SEserviceCallBack" interface.

```
import org.simalliance.openmobileapi.*;

public class SeActivity extends Activity implements SEService.CallBack
{
    private SEService seService;
    …
    //Manage connection to access the SE service:
    //instantiate the service with context and callback
    try {
        seService = new SEService(this, this);
    } catch (SecurityException e) {
        // Process the exception
    } catch (Exception e) {
        // Process the exception
        …
    }
    ...
    //Connection callback
    public void serviceConnected(SEService service)
```

25 Inspired from the source to be found at this URL: http://code.google.com/p/seek-for-android/wiki/UsingSmartCardAPI

```
{
    //Communicating with the Secure Element
    try
    {
        Reader[] readers = seService.getReaders();

        //Testing the presence of an NFC reader (i.e. internal)
        if (readers.length < 1)
        {
            return;
        }
        //For example, the 1st reader is used (depends on the targeted SE)
        Session session = readers[0].openSession();
        //Opening the channel with the applet AID
        Channel channel = session.openLogicalChannel(new byte[] {
                        (byte) 0xD2, 0x76, 0x00, 0x01,
                        0x18, 0x00, 0x02, (byte) 0xFF,
                        0x49, 0x50, 0x25, (byte) 0x89,
                        (byte) 0xC0, 0x01, (byte) 0x9B, 0x01
            });

        //Testing communication with Applet
        byte sw1 = (byte) 0x90;
        byte sw2 = 0x10;
        byte[] data = new byte[3] {0x00, 0x00, 0x00};
        byte[] respApdu = channel.transmit(new byte[]{sw1, sw2, data[0],
                                                      data[1], data[2]});

         channel.close();

        //Check returned status code
        If (respApdu[0]==(byte) 0x90 and respApdu[1]== 0x00)
        {
            //Retrieve the « data » part
            data = new byte[respApdu.length - 2];
            System.arraycopy(respApdu, 0, data, 0, respApdu.length - 2);
            //Process the response
            ...
        } else {
            // Manage the error
        }
        ...
    } catch (Exception e) {
```

```
                //Manage the error
                ...
            }
        }
        //Do not forget to deactivate the service
        @Override
        protected void onDestroy()
        {
            if (seService != null && seService.isConnected())
            {
                seService.shutdown();
            }
            super.onDestroy();
        }
}
```

Box 2.30. *Access to the SE with OMAPI*

2.2.5. *Developing NFC services with Android HCE*

Until now, we saw that the development of the card emulation mode with Android consisted of implementing the (Android) client application of a service executed in the OS of an SE hosted in a hardware component of smart card type: the *IsoDep* class makes it possible to implement Android applications acting as an NFC reader able to communicate with the NFC smart cart type tag (including the SE of another device) detected when the other device is brought into proximity, whereas the *OMAPI* API allows communication with an SE hosted in the same device as the client application.

The development of HCE mode is a specificity of the NFC card emulation mode where the service implementing the APDU interface acts as an SE. Yet, contrary to a service executed in the SE, the HCE service is executed in the Android OS as with any other Android background service: from an "external" point of view of an NFC smart card reader, the HCE service is "seen" as an NFC smart card; only the implementation of the client application running in the same Android device differs from the one communicating with a hardware-based SE.

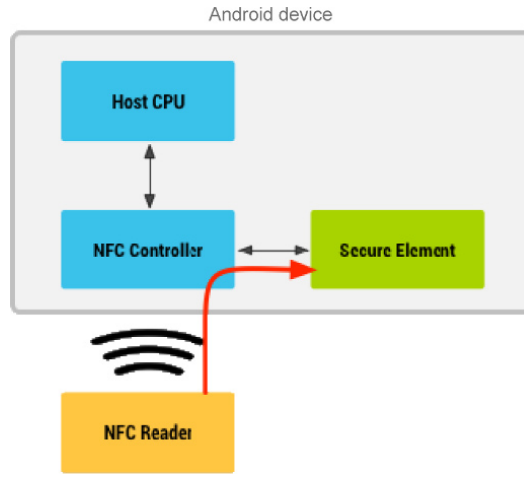

Figure 2.30. *Android: routing of APDU toward the SE*

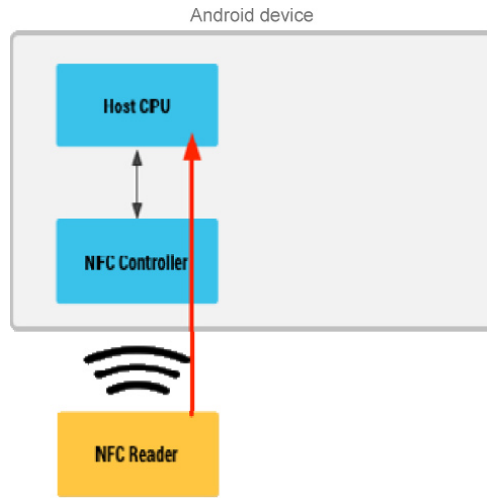

Figure 2.31. *Android: routing of APDU with HCE*

As illustrated in Figure 2.30, when a service is hosted in the SE, the routing of APDU commands does not go through the OS of the

Android device; the communication between the NFC reader and the chip service is thus not "visible" from the Android OS, contrary to the HCE mode: from the Android KitKat version (4.4) with HCE, NFC services in card emulation mode are declared in the Android OS; then, the APDU SELECT instruction indicating the AID of the target service allows Android to determine the routing of APDU commands, as illustrated in Figure 2.31.

NOTE.– The HCE mode has drawbacks and advantages: as HCE is not hardware SE-based and benefits from the access to all the features of the host device, the HCE service has full access to the Android OS connectivity, thus releasing the service providers from the complex and constraining management of the SE service lifecycle induced by the closed environment and the restricted access of the SE, which requires going through one or several third parties (SE issuer and/or TSM), and avoiding the additional costs it triggers. HCE services are accessible through the information system, just like any other mobile application, and can be managed end-to-end by the service provider, without going through a third party. Yet, HCE does not benefit from the security inherent to smart cards technology and the HCE service provider must integrate security in all aspects of his system. Also, contrary to NFC services hosted in the SE, which will keep functioning when the smartphone is turned off or when the battery is discharged, the HCE service is only available when the system is operational (switched).

2.2.5.1. *Declarations for HCE in the AndroidManifest.xml file.*

When an Android project has one (or several) HCE service(s), the NFC permission must of course be declared, but also the use of the HCE feature (see Box 2.31).

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

Box 2.31. *Declaration for HCE in AndroidManifest.xml*

As with any Android activity, Android background services are declared in the *application* section of the manifest (see section 2.1.2.3.4) using the *service* tag. The *android:permission* attribute makes it possible to indicate the *BIND_NFC_SERVICE* permission to establish a connection with the service running in another execution thread, in the background, different from the main thread from an activity. As it is an HCE service, the *HOST_APDU_SERVICE* filter must be added, which will show the system that APDUs must be routed toward this service according to the AID(s) provided in an annex XML file (see example given in Box 2.33) and referenced in the *meta-data* tag "host_apdu_service", as illustrated by the code in Box 2.32.

```
<service
    android:name="my.app.package.MyHCEService"
    android:permission="android.permission.BIND_NFC_SERVICE">
    <intent-filter>
            <action android:name=
            "android.nfc.cardemulation.action.HOST_APDU_SERVICE"/>
            <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
    <meta-data android:name="android.nfc.cardemulation.host_apdu_service"
            android:resource="@xml/my_host_metadata"/>
</service>
```

Box 2.32. *Declaring the HCE service (AndroidManifest.xml)*

In the example provided in Box 2.32, the application has a class, here named "MyHCEService", inherited from the Android *HostApduService* superclass. The service(s) implementing the APDU interface of NFC communication in card emulation mode managed by the HCE service "MyHCEService" are described in an XML file, here named "my_host_metadata", created in the *res/xml/* sub-folder of the application project's tree view (see section 2.1.2.3.3), suggested as an example in Box 2.33: in this example, the text describing services and the group is recovered in the project resources string file (see section 2.1.2.3.3) declaring two services for the routing of APDU destined to AID (here fictitious) "F0000000000000000001" and "F0000000000000000002".

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/my_service_description"
    android:requireDeviceUnlock="false" >
    <aid-group
     android:category="other"
     android:description="@string/my_group_description" >
    <aid-filter android:name="F000000000000000001"/>
    <aid-filter android:name="F000000000000000002"/>
     ...
 </aid-group>
</host-apdu-service>
```

Box 2.33. *Example of Android HCE service description*

NOTE.– Only two categories are managed by Android: the payment services category (CATEGORY_PAYMENT) and other services (CATEGORY_OTHER).

2.2.5.2. *Implementing Android HostApduService superclass*

An Android HCE service extends HostApduService superclass; the *Applet* class of JavaCard™ implementing an HCE service requires overriding the *processCommandApdu* method receiving the APDU command as parameter (see section 2.2.4), as illustrated by the code sample provided in Box 2.34. The HCE service acts like an *Applet*: when received, the APDU command is analyzed and processed by the HCE service, and then a response is returned to the command sender application, including the status code of the execution.

```
import android.nfc.cardemulation.HostApduService;
import android.os.Bundle;

public class MyHCEService extends HostApduService
{

    @Override
    public byte[] processCommandApdu(byte[] commandApdu, Bundle extras)
    {
        //Parse and process commandApdu
        return byteResponse;
    }
```

Box 2.34. *Example of implementation of an HCE Android service*

As mentioned earlier, the advantage of an HCE implementation compared to JavaCard™ is that it benefits from all advanced features of the Android system and especially from the access to the connectivity of the device in which it is executed: this way, APDU commands can be processed locally, but can also be sent to a remote server in order to be processed externally. Moreover, HCE enables keeping the transactions history without suffering from the memory resources limitations of an SE, and additional information can be taken into account when processing commands, such as the location or time stamp of the transaction.

NOTE.– With HCE, the service provider must make sure the service has a sufficient level of security by implementing encryption and authentication mechanisms, for example.

2.2.5.3. *Communicating with an HCE service from the Android OS*

Just like an SE service, the HCE service can be accessed through an external device acting as an NFC smart card reader (see section 2.2.4.2). This is not the case for an application running in the same device hosting the HCE service: indeed, OMAPI enables a generic communication with an SE service (see section 2.2.4.3.2) regardless of the configuration (SIM card, eSE or microSD), but not with an HCE service.

In order to communicate with the HCE service, we must use Android's interprocess communication mechanisms: communication can be made with the help of an *intent* (see section 2.1.3) filtered on one or several predefined actions identified by a string of characters code (see Box 2.35). The *intent* can embed parameters in a *Bundle*, then it is broadcast through the system to be received and processed by a *BroadcastReceiver* attentive to the actions defined in the filter (see Box 2.36).

```
Intent myIntent = new Intent("myfilter.on.myHCEService.action");
myIntent.putExtra(myParamKey, myParamValue);
context.sendBroadcast(myIntent);
```

Box 2.35. *Example of sending a filtered intent on a predetermined action*

NOTE.– Using a *BroadcastReceiver* is not the only solution. For example, we can find other solutions[26] based on Android *Messenger*[27].

Then, we just need to implement a *BroadcastReceiver* dedicated to receiving intents for the HCE service, for example in the service itself (see Box 2.36).

```
public static class MyInnerReceiver extends BroadcastReceiver
{

    @Override
    public void onReceive(Context context, Intent myIntent)
    {
            if (myHceServiceInstance==null)
            {
                    //Process the case in which HCE service is not started?
            }
            //Retrieve the received bundle (matching the action filter)
            if (myIntent != null &&
             myIntent.getAction().equals("myfilter.on.myHCEService.action"))
            {
                    Bundle extras = myIntent.getExtras();
                    if (extras != null)
                    {
                            myStringParam = extras.getString(myParamKey);
                            //Received parameter processing
                    }
             }

    }
};
```

Box 2.36. *Android: implementation of a BroadcastReceiver*

In order for intents to be routed toward the *BroadcastReceiver*, the latter must previously register in the system. Android offers two

26 See implementation example (one among several) can be found at: https://android.googlesource.com/platform/frameworks/base/+/51b6322/core/java/android/nfc/cardemulation/HostApduService.java.

27 http://developer.android.com/reference/android/os/Messenger.html.

solutions to declare a *BroadcastReceiver*: (i) a static declaration in a manifest file (see Box 2.37) or (ii) a dynamic declaration when processed (see Box 2.38).

```
<receiver adroid:name = "com.example.MyHCEService$MyInnerReceiver"
      android:enabled="true"
      android:exported="true">
      <intent-filter>
                  <action android:name = "myfilter.on.myHCEService.action"/>
                  <category android:name = "android.intent.category.DEFAULT"/>
      </intent-filter>
</receiver>
```

Box 2.37. *Static declaration of a BroadcastReceiver*

NOTE.– In the example in Box 2.37, the class "MyInnerReceiver" is a subclass of the "MyHCEService" service: in order to declare a subclass in the Android manifest, the name includes the package's path, separating the master class from the subclass by "$".

```
@Override
public void onCreate()
{
      super.onCreate();
      myReceiver = new MyInnerReceiver();
      IntentFilter myFilter = new IntentFilter("myfilter.on.myHCEService.action");
      registerReceiver(myReceiver, myFilter);
}
@Override
public void onDestroy()
{
      unregisterReceiver(myReceiver);
      super.onDestroy();
}
```

Box 2.38. *Dynamic registration of a BroadcastReceiver*

NOTE.– In the example in Box 2.38, the registration is carried out according to the *onCreate* method and deregistration is carried out

according to the *onDestroy* method in case of a service. In case of an activity, we prefer to use *onResume* and *onPause* methods, more fit to the lifecycle of the Android activity (see section 2.1.4.1).

3

NFC Use Cases

NFC applications are implemented in three different modes:

– NFC applications using the reader/writer mode to read information (e.g. a URL, an activation code);

– NFC P2P mode to exchange information (e.g. computer files exchange) or to pair two NFC devices (e.g. exchange of connection data);

– NFC application transactions based on the card emulation mode of NFC and communication with services offering interfaces based on smart cards standard (e.g. for m-payment transactions, e-ticketing, identification or access control).

In this chapter, we offer tangible examples of the use of the NFC standard illustrating each of the three operating modes of NFC in a classic use case. However, we must keep in mind that NFC can be applied to any sector for a variety of use cases.

3.1. Usage of the NFC reader/writer mode

The NFC standard reader/writer mode is similar to the use case of QR codes, with the advantage that the user does not need to select a program, nor to center a picture in order to access the content, but simply to bring the NFC mobile device (i.e. acting as a reader) into the

The Art and Science of NFC Programming, First Edition.
Anne-Marie Lesas and Serge Miranda.

close proximity of the "target" object to be read (e.g. NFC tag). This thus automatically starts the mobile application linked to the tag content (see section 2.2.2.1). The target object contains a NDEF message; it can be any material object, for example a building, a door, a poster or a book. The content of the NFC tag (e.g. a URL and a code) acts as a link with information in the digital world (e.g. a work of art story in a museum, a user guide, tourist or place information, media video/music, etc.) or as a trigger for an action "run" when "tapping" (e.g. to open a door, turn a device on or track the visit of a user).

The NFC tag is a unique identifier of the "tagged" object (i.e. because of its UID) and the NFC mobile device is an identifier about the end user (i.e. IMEI and login information) and the context (i.e. "here and now": temporality, location, etc.). This property of precise contextualization of the event allows to create sophisticated use cases (e.g. m-payment and access control).

3.1.1. *Use case: management of equipment loans*

We propose a scenario of equipment loan management using the NFC reader/writer mode. The use case involves a user acting as the manager (here called "admin") and a user borrower (here called "user") holding an NFC smartphone, and something to be borrowed; it can be any kind of object (e.g. book, equipment, car share vehicle or hotel room) having an NFC tag linked to it.

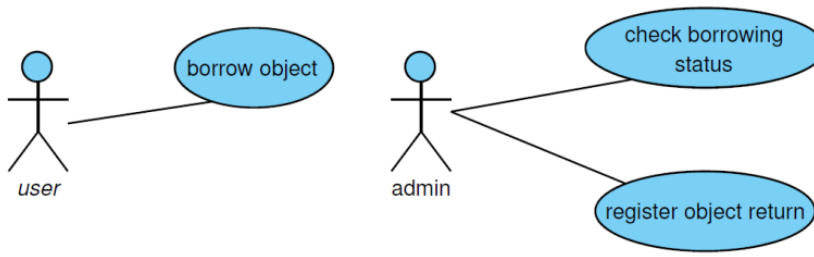

Figure 3.1. *Use case of borrowings management*

The system includes:

– an NFC mobile application for the user intended for the registering of the object borrowing;

– an NFC mobile application for the admin intended for the checking of the borrowing registrations, and for registering the object returns;

– a web server with a paired web services API and a storage unit (database) for information management. We assume that two web user interfaces (UI) will be available: (1) the loans management UI for admin users to control returns and (2) the user's UI to check his borrowing history: As this aspect does not have any impact on the NFC use case, it is not studied here;

– a communicating object having an NFC tag type interface.

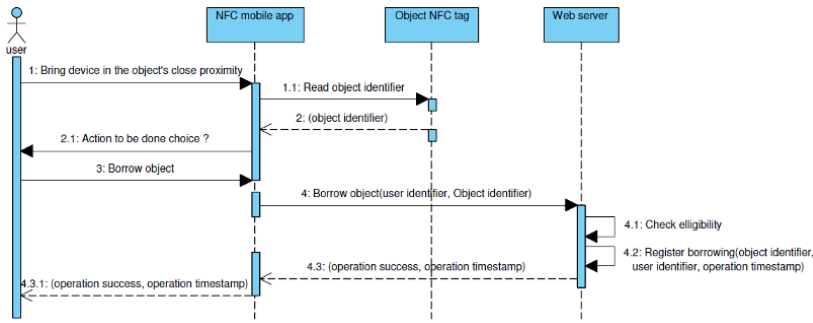

Figure 3.2. *Object borrowing*

Figure 3.2 shows interactions between the system components when the user borrows an object:

(1) the user taps the NFC interface of the object he/she wants to borrow with his/her NFC mobile phone;

(1.1) the application reads the object ID (it can be the UID or a code encoded on the tag earlier);

(2.1) the application asks for the action to perform (indeed, the application can offer several options such as, for example, display information on the object or the history of the user's borrowings);

(3) the user selects the option to borrow the object;

(4) the application connects to the server to register the borrowing;

(4.1) the server checks the eligibility of the request (indeed, the system checks that the user is allowed to borrow the object and that the object has not already been borrowed somewhere else);

(4.2) the borrowing is registered in the database;

(4.3) the server informs the application that the borrowing was properly registered;

(4.3.1) the application informs the user that the object borrowing was successfully registered.

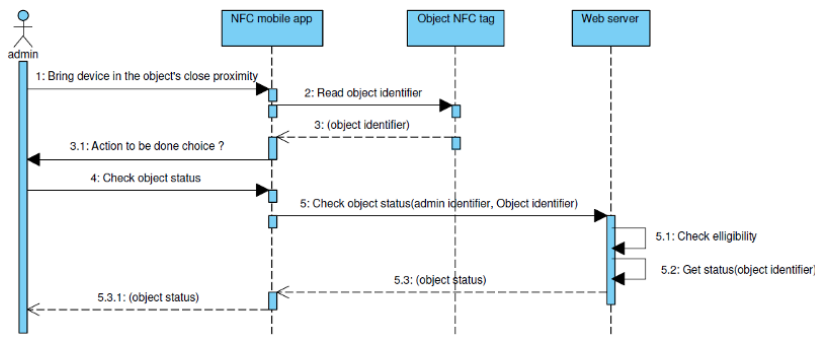

Figure 3.3. *Checking an object status*

An object (borrowed or not) can be checked at any time by a loans admin (for example to check that a user has properly registered the borrowing), as illustrated in Figure 3.3:

(1) the admin taps the NFC interface of the object to control with his/her NFC mobile device;

(2) the application reads the object identifier;

(3.1) the application asks for the action to perform;

(4) the admin selects the check object status option;

(5) the application connects to the server to collect data on the object;

(5.1) the server checks the eligibility of the request;

(5.2) the server collects data on the object status (or its history);

(5.3) the server informs the application that the borrowing was successfully registered;

(5.3.1) the application displays object information.

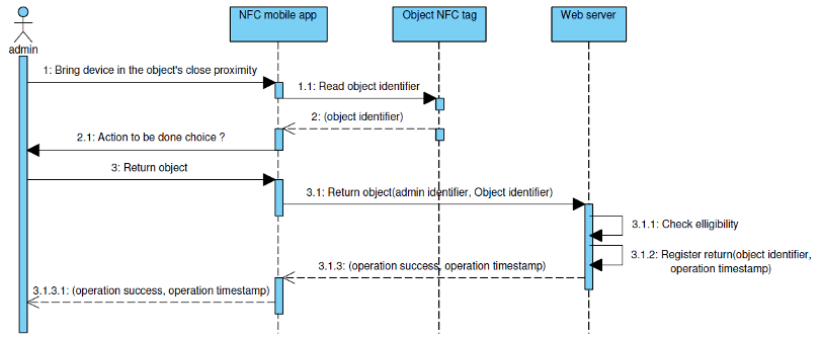

Figure 3.4. *Object return*

Returning a borrowed object is performed by the loans admin once the object has been physically returned by the user (borrower) (see Figure 3.4):

(1) the admin taps the NFC interface of the object to control with his/her NFC mobile;

(1.1) the application reads the identifier of the object;

(2) the application asks for the action to perform;

(3) the manager selects the return option of the object;

(3.1) the application connects to the server to register the object return;

(3.1.1) the server checks the request eligibility;

(3.1.2) the server registers the object return;

(3.1.3) the server informs the application that the return was successfully registered;

(3.1.3.1) the application informs the user that the return was successfully registered.

This example shows how simple it is to perform transactions with NFC technology, even without a costly professional device. The power of such a system is strengthened by incorporating geolocation, provided by the GPS of mobile phones during transaction requests; in this way, inference rules can be added to the eligibility check of transactions and/or additional functions around traceability can be implemented.

3.2. Usage of the NFC P2P mode

The simplest use case of the P2P mode of the NFC standard consists of exchanging data between two NFC peripheral devices. This can simply be media sharing between two users, i.e. where a first user starts the exchange by choosing beforehand the document he/she wishes to share with the target device of the other user. The two users will then bring their two devices into close proximity and the NFC transfer will occur from the initiator device to the target device.

3.2.1. *Use case: NFC pairing*

The use case of NFC P2P mode addressed in this section involves a user having two NFC-enabled devices with another high-speed connectivity (e.g. Bluetooth® and Wi-Fi):

– the “master” device has a user interface (for example a smartphone);

– the “slave” device does not need to have a user interface (for example a speaker, but it could be any object: a coffee machine, a car, a house, etc.).

Here, the “master” device (the smartphone) acts as a “universal remote control” allowing the user to connect and control the “objects”, NFC technology being the “universal connector” to pair devices with “zero manual configuration”.

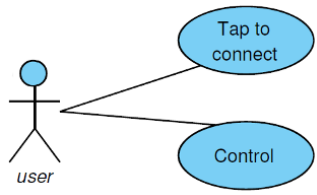

Figure 3.5. *Use case of pairing*

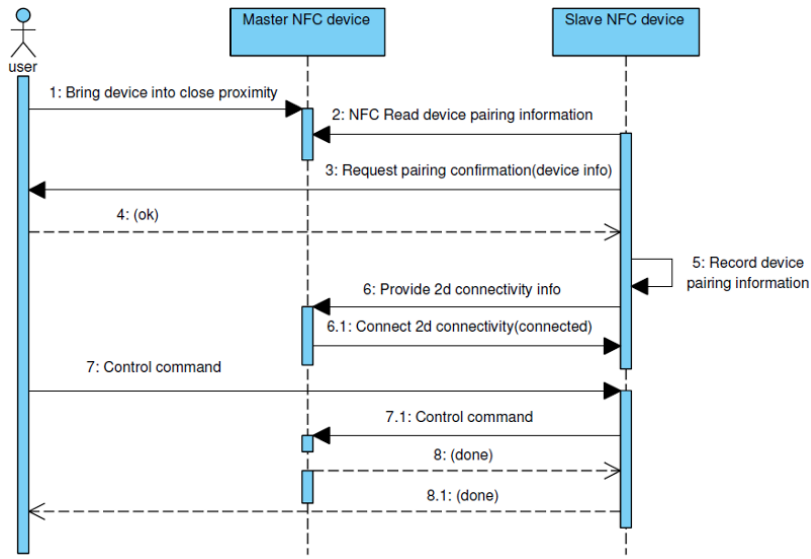

Figure 3.6. *Interactions of pairing scenario*

The interactions of our pairing scenario (see Figure 3.6.) show how a second connectivity can be set up with NFC:

(1) the user brings the slave peripheral device to pair into close proximity with the control device (master);

(2) the master device (e.g. smartphone) reads the pairing information of the device to pair thanks to NFC connectivity;

(3) since this is the first contact, the master device requests a confirmation from the user who wishes to pair with the slave device (after the first pairing, it will connect automatically, with no need for confirmation);

(4) the user confirms the pairing;

(5) the master device registers the parameters of the slave device in order to use it again later (e.g. authentication and control data);

(6) depending on pairing information, the master device sends information to allow the slave device to use the second connectivity (this part is to the developer's judgment with the help of the NFC standard forum mentioned in section 1.3.4 and depends on the API available from the development platform);

(6.1) the slave device connects to the master device through the second connectivity;

(7) the user can then send control commands to the slave device through the user interface of the master device;

(8) the master device sends commands to the slave device according to the pattern provided by the collected data when initializing the pairing (with NFC), through the second connectivity;

(8.1) the slave device notifies the master device that the command was run successfully; the user is notified by the master device.

3.3. Usage of NFC card emulation mode

NFC card emulation mode is characterized by the APDU layer (smart cards protocol and JavaCard™ platforms). NFC technology is thus a contactless transport layer for services embedded in a chip called "secure element": the SE (see section 1.3.2.2). When the SE is integrated into the mobile device (i.e. in a smartphone or a tablet), an API accessible from the OS of the mobile phone is necessary to communicate with the SE. This API is not always available for developers. On the Android platform (see section 2.2.4), OMAPI

(see section 1.3.6) enables communication with the SE in its three hardware form factos (SIM, eSE and microSD), regardless of the configuration, in the way of an internal reader (see section 2.2.4.3.2). Theoretically, all existing services based on a smart card (e.g. credit cards, transportation cards, identification cards, subscription cards and health insurance cards) can be hosted in an SE interfaced internally with an application benefiting from the advanced features of the mobile device.

An application in NFC card emulation mode is made of an on-card service (with restricted access) running in the SE, and of an application run in the host mobile device OS.

For the use case given as an example in section 3.3.1, we address a digital wallet scenario.

3.3.1. *Use case: digital wallet in the SE*

We present the original use case[1] of a digital wallet as an example for the NFC card emulation mode. The digital wallet can act as alternative virtual currency for community trades; it can thus be a barter currency between two individuals within the community, or a dedicated currency virtualizing service vouchers, or loyalty points (e.g. meal vouchers, holiday vouchers, gift vouchers), which are only valuable in a specific context.

The system involves two users equipped with NFC devices acting, respectively, as initiator and target for a debit/credit transaction (see Figure 3.7), where the target user authorizes the transaction by bringing into proximity and "tapping" the initiator mobile. The digital wallet of the payer user will then be debited and the wallet of the debit user will receive the same amount. We assume an opening balance of zero and the possibility of negative balance (i.e. in a balanced economic system, scales must balance with transactions). The system includes an application acting as a user interface to manage

1 All intellectual property rights reserved to the author.

debit/credit transactions interfacing with a digital wallet service hosted in the SE.

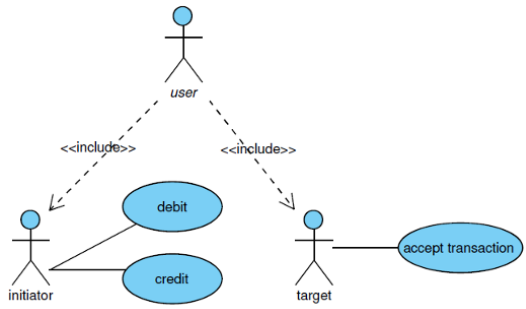

Figure 3.7. *Use case of a digital wallet*

NOTE.– In this example, we forget the more complex aspects of the lifecycle management of the on-card SE service, which is standardized and well documented by GlobalPlatform (see section 1.3.5).

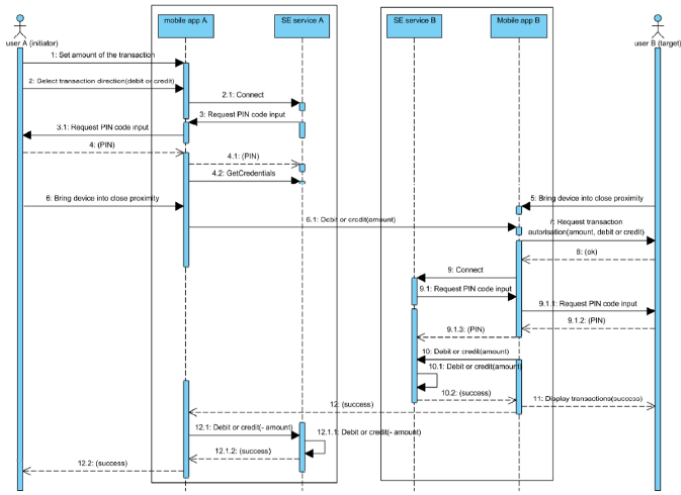

Figure 3.8. *Scenario of the digital wallet interactions with an SE*

Figure 3.8 shows the interactions between two NFC mobile devices during a debit/credit transaction:

(1) user A (initiator) enters the transaction amount;

(2) user A enters the direction of the transaction (debit or credit);

(2.1) the mobile app of user A connects in NFC card emulation mode with the digital wallet service hosted in the SE of user A's mobile phone through an internal reader (i.e. through the SE communication API);

(3) SE A sends an authentication request via PIN code;

(3.1) mobile app A asks user A to enter his/her PIN code;

(4), (4.1) user A enters his/her PIN code, which is transmitted to the SE service A;

(5), (6) users A and B bring their mobile devices into close proximity;

(6.1) mobile app A sends the debit/credit request to mobile app B via the NFC P2P mode;

(7) mobile app B asks user B for credit/debit transaction authorization;

(8) user B gives authorization for the transaction;

(9) mobile app B connects in NFC card emulation mode to the digital wallet service hosted in SE B;

(9.1) SE B sends back an identification request via PIN code;

(9.1.1) mobile app B asks user B to enter his PIN code;

(9.1.2), (9.1.3) user B enters his/her PIN code, which is sent to the service of SE B;

(10) mobile app B sends back a debit/credit command to the service of SE B;

(10.1) the digital wallet service of SE B registers the debit/credit transaction;

(10.2), (11) the SE service B notifies that the debit/credit transaction was processed successfully, which is notified by the mobile app to user B;

(12) mobile app B notifies mobile app A that the transaction succeeded by using NFC P2P mode (at this point, the two devices must touch);

(12.1), (12.1.1) mobile app A sends a payment reversal request (debit/credit) of the same amount (i.e. of reverse sign) to the digital wallet service of SE A that registers the transaction;

(12.1.2), (12.2) the service of SE A notifies mobile app A the transaction succeeded, which is notified to user A.

NOTE.– In our original example of NFC card emulation mode usage, the SE service is not directly accessed externally. However, the configuration we propose is fully adapted to transactions processed with a terminal connected to a contactless NFC reader.

3.4. Usage of the HCE mode

The Host-based card emulation (HCE) mode is a special case of NFC card emulation mode: the on-card SE service is replaced by a background service running in the OS of the mobile device that has an interface able to manage APDUs (see section 2.2.5). The HCE mode does not require a special communication API to interface the SE (as there is no SE involved) and benefits from the access to advanced features of the mobile device (contrary to SE services running in the restricted environment of the smart card's microchip). Moreover, the management of the lifecycle of an HCE service is the same as the management of an ordinary mobile application, which makes its implementation much simpler and does not require interfacing with third parties (i.e. TSM/SEI).

The use case of the NFC card emulation mode implementing an SE suggested in the previous section (see section 3.3.1) could also be implemented in the same way with an HCE service instead of the on-card SE service. In the use case example of HCE proposed in section 3.4.1, the HCE service acts as a gateway for an SE accessed remotely in the Cloud.

3.4.1. *Use case: SE in the Cloud with HCE*

Our scenario of the usage of HCE mode describes the interactions of a use case where the mobile HCE service receives APDU instructions from a terminal NFC reader and sends them to a remote SE service hosted in the Cloud:

(1) the user brings his/her NFC mobile device into close proximity with the NFC reader;

(2) the terminal application connected to the NFC reader sends APDU instructions to the mobile app (received by the HCE service);

(2.1), (3) the mobile app connects to a remote web server;

(3.1) the mobile app forwards the APDU instruction (or the sequence of instructions) to the web server;

(4) the web server sends the instruction (C-APDU) to the hosted SE service;

(4.1), (4.2) the SE service processes the instruction(s) and returns the response (R-APDU);

(4.2.1) the web server returns the APDU response to the mobile app;

(5) the mobile app returns the APDU response to the terminal application.

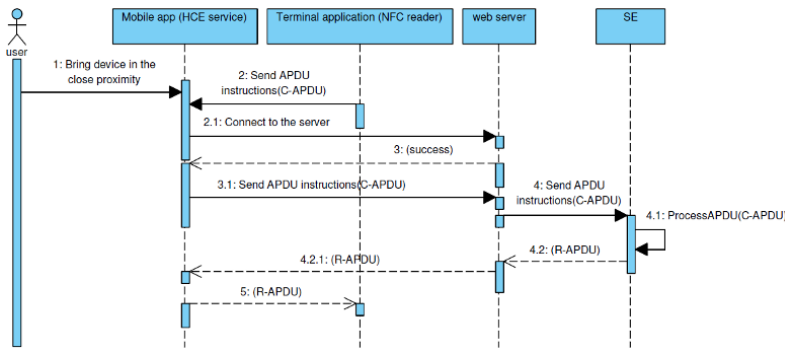

Figure 3.9. *Interactions scenario SE in the Cloud with HCE*

NOTE.– This use case solution of SE in the Cloud using HCE requires Internet connectivity.

Conclusion

When equipped with the NFC standard resulting in a combination of RFID and smart card technologies with a close proximity requirement, smartphones are powerful interaction initiators between material objects, becoming communication and digital services. The three NFC modes cover a great variety of use cases with unlimited applications and unlimited imagination.

With a quick and simple targeted proximity gesture (the 3 "S" and "Tap'n Play", see Introduction), the user can access geolocated, customized, contextualized and multimedia information finding applications in tourism, leisure, culture, technology, education, commerce, etc. These examples are not limited, because the NFC event is characterized by a spatial temporality ("here and now") and identified components (like a tag identified by its UID or its content, or the mobile device's holder identified by its IMEI) with the NFC reader/writer mode.

In P2P mode, NFC reaches a new dimension and allows to pair two devices that will then be able to exchange data (e.g. Wi-Fi, Bluetooth® and Li-Fi) when the initiator decides to do so (his/her will is induced in the fact that he/she brings the two devices into close proximity, with or without set-up during the first contact).

In card emulation mode, NFC not only allows to secure transactions with the implementation of protocols and cryptographic

The Art and Science of NFC Programming, First Edition.
Anne-Marie Lesas and Serge Miranda.

techniques inherited from smart cards technologies, but it also brings a major added value to static services hosted in smart cards that are not very ergonomic and ephemeral (e.g. credit/debit cards, loyalty cards, health insurance cards, transportation cards, access cards, ID cards and other digital documents); all smart card services can be dematerialized in the SE and benefit from the user-friendliness, mobiquity and intelligence of smartphones.

In Chapter 1 on the state-of-the-art of NFC, we addressed the NFC standard normalized by the NFC forum since 2004 that can be applied to many sectors. Yet, in its most noticeable use, the implementation of secure NFC services (in card emulation mode) requires the converging interoperability of industry leaders in heterogeneous sectors (service providers, computer science, manufacturers, telecoms, banks and finances, governments…); the market is growing toward a standardization of secure management interfaces centralized by GlobalPlatform organization around a complex ecosystem.

In Chapter 2, we detailed an introduction to NFC mobile programming with Android in the three NFC modes. The coding examples will help the developer in implementing his own NFC applications with an Android device equipped with NFC technology.

In Chapter 3, we discussed the examples of concrete use cases in the three NFC modes:

– we illustrated the reader/writer mode of NFC with a scenario of NFC applied to object borrowing;

– for P2P mode, we proposed a pairing scenario between two NFC devices;

– the card emulation mode of the NFC standard was illustrated by the example of a digital wallet with two use cases:

- using an SE (i.e. with a service running in the SE),

- using HCE mode (i.e. with a service running in the mobile device’s system and a wallet managed in the Cloud).

To conclude, this book provides all of the basics for a good understanding of the development of NFC applications (with Android) and gives its reader full autonomy.

Now is your turn to offer innovative services, thanks to the NFC standard.

Bibliography

[AHS 12] AHSON S.A., ILYAS M., *Near Field Communication Handbook*, CRC Press, Boca Raton, 2012.

[AIL 07] AILISTO H., MATINMIKKO T., HÄIKIÖ J. *et al.*, *Physical Browsing with NFC Technology*, VTT Tiedotteita, Finland, 2007.

[ATT 13] ATTOUR A., DELLA PERUTA M., Le rôle des connaissances architecturales dans l'élaboration de la plateforme technologique d'un écosystème en émergence : le cas des plateformes NFC, GREDEG Working Papers Series, France, 2013.

[CHA 10] CHAIX L., TORRE D., "Different models for mobile payment", European Research Group, Monnaie Banque Finance, Bordeaux, France, 2010.

[COS 12] COSKUM V., OK K., OZDENIZCI B., *Near Field Communication, From Theory to Practice*, Wiley, 2012.

[COS 13] COSKUM V., OK K., OZDENIZCI B., *Professional NFC Development for Android,* Wiley, 2013.

[FIN 10] FINKENZELLER K., *RFID Handbook*: *Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication,* 3rd ed., Wiley, 2010.

[HAJ 09] HAJJI F., ZIAD A., DE LAZZARI T. *et al.*, "A multi-touch NFC virtual poster platform for m-learning: Al Andalus project", *M-learning IADIS International Conference*, Barcelona, Spain, February 2009.

The Art and Science of NFC Programming, First Edition.
Anne-Marie Lesas and Serge Miranda.

[IGO 14] IGOE T., COLEMAN D., JEPSON B., *Beginning NFC, Near Field Communication with Arduino, Android, and PhoneGap*, O'Reilly, 2014.

[JIT 13] JITHESH S., ANOOP N., NAVIN N. *et al.*, *A Comprehensive Guide to Enterprise Mobility*, CRC Press, Boca Raton, 2013.

[KNU 68] KNUTH D.E., *The Art of Computer Programming*, *Fundamental Algorithms*, vol. 1, Addison-Wesley, 1968.

[LES 14a] LESAS A.M., MIRANDA S., RENAUT B. *et al.*, "WOLF: a research Platform to write NFC secure applications on top of multiple Secure Elements", *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 8, pp. 20–31, 2014.

[LES 14b] LESAS A.M., "A fast implementation of TSM web services within WOLF API for F1RST android mobile-NFC services", University of Nice Sophia Antipolis, France, 2014.

[LES 14c] LESAS A.M., "F1RST research project mobiquitous NFC financial services for unbanked people", *WIMA Conference*, Monaco, April 2014.

[LES 16] LESAS A.M., "Détecter et monitorer les séismes grâce aux capteurs embarqués dans les smartphones", *INFormatique des ORganisation et Systèmes d'Information et de Décision (INFORSID)*, Grenoble, France, May 2016.

[MIR 09] MIRANDA S., "Robotics platform integrating wireless communicating objects for assisted living (RoboDOMO project)", *Keynote WASM Conference*, Oulu, Finland, November 2009.

[MIR 11a] MIRANDA S., LESAS A.M., "Architecture logicielle du projet VAMP: Plate-forme mobiquitaire embarquée à bord de véhicules automobiles", *Génie Logiciel,* vol. 103, pp. 38–48, 2011.

[MIR 11b] MIRANDA S., PASTORELLY N., ISHKINA E. *et al.*, "Lessons inferred from NFC mobiquitous innovative information service prototyping at UNS", in Ingénierie des systèmes d'information, vol. 16, no. 4, pp. 49–62, 2011

[MIR 14a] MIRANDA S., PAPETTI C., "Les nouveaux paradigmes du tourisme mobiquitaire", in CHRISTOFLE S. (ed.), *Tourisme, destinations et entreprises. Leadership dans le luxe, l'"événementiel" et les TIC*, Mondes du tourisme, France, 2014.

[MIR 14b] MIRANDA S., PAPETTI C., SOK K.H. *et al*., "Une application de gestion de tags pour le tourisme mobiquitaire: application au projet Thom au Cambodge", in SPINDLER J., CLERGEAU C. (eds), *l'Immatériel Touristique*, L'Harmattan, France, 2014.

[MIR 14c] MIRANDA S., "Homo mobiquitus, communacteur sur les territoires", in CARMES M., NOYER J.M. (eds), *Devenirs Urbains,* Presse des Mines, France, 2014.

[MIR 16] MIRANDA S., "BIG DATA et Innovation Spiraliste (Big Data and Connected Objects II)", *7th International Research Meeting in Business and Management (IRMBAM-2016)*, Nice, France, 2016.

[NAR 11] NARNI-MANCINELLI G., BENOUALI H., LEITZELMAN M. *et al*., "MBDS2.0, plateforme générique de gestion de tags NFC et 2D pour des espaces culturels intelligents et communautaires 2.0", *Ingénierie des systèmes d'information*, vol. 16, no. 4, pp. 49–62, 2011.

[PAR 12] PARET D., BOUTONNIER X., HOUITI Y., *NFC Near Field Communication, Principes et applications de la communication en champ propre*, Dunod, Paris, 2012.

[PAS 11] PASTORELLY N., BENOUALI H., LEBLANC C. *et al*., "Nice Future Campus, un bouquet de services NFC dans une carte virtuelle étudiant", *Ingénierie des systèmes d'information*, vol. 16, no. 4, pp. 64–86, 2011.

[TUI 09] TUIKKA T., ISOMURSU M., *Touch the Future with a Smart Touch*, VTT Tiedotteita, Finland, 2009.

Index

A, B, C

access control, 8, 9, 12, 23, 38, 40, 107, 108
active, 6, 7, 9, 18, 25, 28, 45
analog, 18–20, 31
Android, 45
  Development Toolkit (ADT), 46, 50–53, 56, 68, 70
Applet, 24, 87, 89, 90, 101
  Identifier (AID), 88, 99, 100
Application Protocol Data Unit (APDU), 35, 38, 40, 87, 88, 97, 98, 114, 118, 119
asymmetrical, 11, 12
ATAWAD, 2
authorization, 9, 12, 41, 117
authorized (dual) mode, 41
big data, 2, 5
BroadcastReceiver, 102–104
card
  emulation, 22–25, 31, 36, 45, 89, 91, 92, 97, 99, 100, 107, 114–118
  specifications, 23, 36
Cloud, 24, 118–120
connector, 15, 113

D, E, G

delegated mode, 41
dematerialized, 122
device specifications, 36
ECMA-340, 17
ecosystem, 1, 2, 4, 16, 36, 40, 41
Europary MasterCard Visa (EMV), 11, 23, 25
geolocation, 5, 17, 112
GlobalPlatform (GP), 24, 36–42, 116
Google Play, 47, 59

H, I, J

host-based card emulation (HCE), 24, 97–103, 118, 119, 120
identification, 1, 8, 10, 13, 24, 107, 115, 117
implementation, 28, 29, 37, 61, 85, 90, 92, 97, 101–103, 118
information system, 13, 99
initiator, 17, 18, 21, 25, 28, 31, 45, 112, 115, 117

The Art and Science of NFC Programming, First Edition.
Anne-Marie Lesas and Serge Miranda.

integrity, 9, 10, 13
interaction, 3, 17, 37, 61, 109, 113, 116, 119
interoperability, 16, 36
ISO/IEC
  14443, 17, 26, 28, 29, 31, 35
  15693, 19, 28
  18092, 17, 26, 28, 29, 31
  7816, 23, 35
IsoDep, 89, 91, 97
JavaCard™, 23, 24, 87, 101, 102, 114
JIS X 6319–4, 35

L, M, N

lifecycle, 11, 38, 40–42, 61, 66, 90, 99, 105, 116, 118
location, 49, 65, 102, 108
logical link control protocol (LLCP), 25, 29, 30, 45
manifest, 56, 58–60, 64, 71, 73–77, 84, 90, 100, 104
mobiquitous, 2, 3, 5, 13, 16
mobiquity, 2–4, 122
modulation, 17–20, 28, 31
m-payment, 5, 13, 15, 107, 108
NFC
  data exchange format (NDEF), 22, 31
  Forum, 17, 19, 22, 25, 26–30, 33–36
NFC-A, 18, 21, 31
NFC-B, 18, 31
NFC-C, 31
NFC-F, 19
NFCIP-1, 27, 29
NFCIP-2, 28
NFC-V, 28
N-Mark, 27

O, P, R

Open Mobile API (OMAPI), 42, 92–95
pairing, 1, 27, 36, 112–114, 122
passive, 1, 7, 8, 17, 18, 21, 25, 27, 28, 45, 71
peer-to-peer (P2P), 6,
private key, 10, 12, 91
programming, 46
public
  key, 10–12
  key infrastructure, 10
radio frequency identification (RFID), 1
reader/writer, 31, 45, 71–75, 84, 107–112
record type definition (RTD), 22, 36
routing, 98–100

S, T

secure element, 9, 23, 114
security, 9–11, 36–38, 42, 47, 90, 99, 102
server, 3, 6, 22, 40, 102, 109–112, 119
signal, 7, 15, 18–20, 28, 31
SIMAlliance, 42–43
simple
  mode, 41
  NDEF exchange protocol (SNEP), 31
smart card, 8–12, 37, 87, 107
standardization, 16, 24, 25, 36
symmetrical, 10–12
trusted execution environment (TEE), 39

traceability, 1, 5, 8, 38, 112
transmedia, 5
trusted service manager
(TSM), 39–42
type
1 tag, 35
2 tags, 35
3 tags, 27, 35
4 tags, 35
name format (TNF), 33

U, V

unique identifier (UID), 8, 79, 109
use case, 107–112, 115, 119
virtual money, 4

Other titles from

in

Information Systems, Web and Pervasive Computing

2016

BEN CHOUIKHA Mona
Organizational Design for Knowledge Management

BERTOLO David
Interactions on Digital Tablets in the Context of 3D Geometry Learning (Human-Machine Interaction Set – Volume 2)

BOUVARD Patricia, SUZANNE Hervé
Collective Intelligence Development in Business

EL FALLAH SEGHROUCHNI Amal, ISHIKAWA Fuyuki, HÉRAULT Laurent, TOKUDA Hideyuki
Enablers for Smart Cities

FABRE Renaud, in collaboration with MESSERSCHMIDT-MARIET Quentin, HOLVOET Margot
New Challenges for Knowledge

GAUDIELLO Ilaria, ZIBETTI Elisabetta
Learning Robotics, with Robotics, by Robotics (Human-Machine Interaction Set – Volume 3)

HENROTIN Joseph
The Art of War in the Network Age (Intellectual Technologies Set – Volume 1)

KITAJIMA Munéo
Memory and Action Selection in Human–Machine Interaction (Human–Machine Interaction Set – Volume 1)

LAGRAÑA Fernando
E-mail and Behavioral Changes: Uses and Misuses of Electronic Communications

LEIGNEL Jean-Louis, UNGARO Thierry, STAAR Adrien
Digital Transformation

MONINO Jean-Louis, SEDKAOUI Soraya
Big Data, Open Data and Data Development (Smart Innovation Set – Volume 3)

NOYER Jean-Max
Transformation of Collective Intelligences (Intellectual Technologies Set – Volume 2)

VENTRE Daniel
Information Warfare – 2nd edition

VITALIS André
The Uncertain Digital Revolution

2015

ARDUIN Pierre-Emmanuel, GRUNDSTEIN Michel, ROSENTHAL-SABROUX Camille
Information and Knowledge System (Advances in Information Systems Set – Volume 2)

BÉRANGER Jérôme
Medical Information Systems Ethics

BRONNER Gérald
Belief and Misbelief Asymmetry on the Internet

IAFRATE Fernando
From Big Data to Smart Data (Advances in Information Systems Set – Volume 1)

KRICHEN Saoussen, BEN JOUIDA Sihem
Supply Chain Management and its Applications in Computer Science

NEGRE Elsa
Information and Recommender Systems
(Advances in Information Systems Set – Volume 4)

POMEROL Jean-Charles, EPELBOIN Yves, THOURY Claire
MOOCs

SALLES Maryse
Decision-Making and the Information System (Advances in Information Systems Set – Volume 3)

SAMARA Tarek
ERP and Information Systems: Integration or Disintegration
(Advances in Information Systems Set – Volume 5)

2014

DINET Jérôme
Information Retrieval in Digital Environments

HÉNO Raphaële, CHANDELIER Laure
3D Modeling of Buildings: Outstanding Sites

KEMBELLEC Gérald, CHARTRON Ghislaine, SALEH Imad
Recommender Systems

MATHIAN Hélène, SANDERS Lena
Spatio-temporal Approaches: Geographic Objects and Change Process

PLANTIN Jean-Christophe
Participatory Mapping

VENTRE Daniel
Chinese Cybersecurity and Defense

2013

BERNIK Igor
Cybercrime and Cyberwarfare

CAPET Philippe, DELAVALLADE Thomas
Information Evaluation

LEBRATY Jean-Fabrice, LOBRE-LEBRATY Katia
Crowdsourcing: One Step Beyond

SALLABERRY Christian
Geographical Information Retrieval in Textual Corpora

2012

BUCHER Bénédicte, LE BER Florence
Innovative Software Development in GIS

GAUSSIER Eric, YVON François
Textual Information Access

STOCKINGER Peter
Audiovisual Archives: Digital Text and Discourse Analysis

VENTRE Daniel
Cyber Conflict

2011

BANOS Arnaud, THÉVENIN Thomas
Geographical Information and Urban Transport Systems

DAUPHINÉ André
Fractal Geography

LEMBERGER Pirmin, MOREL Mederic
Managing Complexity of Information Systems

STOCKINGER Peter
Introduction to Audiovisual Archives

STOCKINGER Peter
Digital Audiovisual Archives

VENTRE Daniel
Cyberwar and Information Warfare

2010

BONNET Pierre
Enterprise Data Governance

BRUNET Roger
Sustainable Geography

CARREGA Pierre
Geographical Information and Climatology

CAUVIN Colette, ESCOBAR Francisco, SERRADJ Aziz
Thematic Cartography – 3-volume series
Thematic Cartography and Transformations – volume 1
Cartography and the Impact of the Quantitative Revolution – volume 2
New Approaches in Thematic Cartography – volume 3

LANGLOIS Patrice
Simulation of Complex Systems in GIS

MATHIS Philippe
Graphs and Networks – 2nd edition

THERIAULT Marius, DES ROSIERS François
Modeling Urban Dynamics

2009

BONNET Pierre, DETAVERNIER Jean-Michel, VAUQUIER Dominique
Sustainable IT Architecture: the Progressive Way of Overhauling Information Systems with SOA

PAPY Fabrice
Information Science

RIVARD François, ABOU HARB Georges, MERET Philippe
The Transverse Information System

ROCHE Stéphane, CARON Claude
Organizational Facets of GIS

2008

BRUGNOT Gérard
Spatial Management of Risks

FINKE Gerd
Operations Research and Networks

GUERMOND Yves
Modeling Process in Geography

KANEVSKI Michael
Advanced Mapping of Environmental Data

MANOUVRIER Bernard, LAURENT Ménard
Application Integration: EAI, B2B, BPM and SOA

PAPY Fabrice
Digital Libraries

2007

DOBESCH Hartwig, DUMOLARD Pierre, DYRAS Izabela
Spatial Interpolation for Climate Data

SANDERS Lena
Models in Spatial Analysis

2006

CLIQUET Gérard
Geomarketing

CORNIOU Jean-Pierre
Looking Back and Going Forward in IT

DEVILLERS Rodolphe, JEANSOULIN Robert
Fundamentals of Spatial Data Quality