# TABLA DE CONTENIDOS

# ANNEX A. PRELIMINARY STUDY

## A.1. THEORY

### A.1.1. NFC TECHNOLOGY

Near Field Communication (NFC) is a standards-based, short-range high-frequency wireless connectivity technology that enables intuitive, simple and safe two-way interactions among electronic devices. NFC technology allows consumers to perform contactless transactions, access digital content and connect devices with a simple touch.

NFC is primarily aimed at usage in mobile phones, and this is the context where we are going to discuss about, along the current document.

#### CHARACTERISTICS AND TECHNICAL SPECIFICATIONS

The technology is a simple extension of the ISO/IEC 14443 standard (Contactless card, Radio-Frequency identification -RFID). This standard defines a proximity-card used for identification and the transmission protocols for communicating with it. NFC communicates via magnetic field induction, where two loop antennas are located within each other's near field, effectively forming an air-core transformer.

As the standard that it is followed, it operates within the globally available and unlicensed radio frequency ISM band of 13.56 MHz, with a bandwidth of almost 2 MHz. Each country is free to impose certain limitations on the electromagnetic emissions in this RF band. The supported data rates are 106, 212, 424 and 848 Kbps. This communication speed is set at the beginning of the process, but subsequently, the application or the communication environment may require speed adaptation, which can be done during communication.

NFC is both a read and write technology. In each connection there must be one reader device and one writer device so they can establish the appropriate exchange of data. The protocol distinguishes between the initiator and the target of the communication. Any device may be either an initiator or target. The initiator, as follows from the name, is the device that initiates and controls the exchange of data. The target is the device that answers the request from the initiator. Since it is capable of receiving and transmitting data at the same time, it can function as a smart contactless card, passive RFID tag and as a smart medium to exchange data between different devices.

The communication protocol also distinguishes between two models of operation: active mode and passive mode. All devices support both communication modes. The distinction is as follows:

- **Active mode of communication**: Both devices communicate by alternately generating their own RF field. A device deactivates its RF field while it is waiting for data. In this mode, both devices typically need to have a power supply.

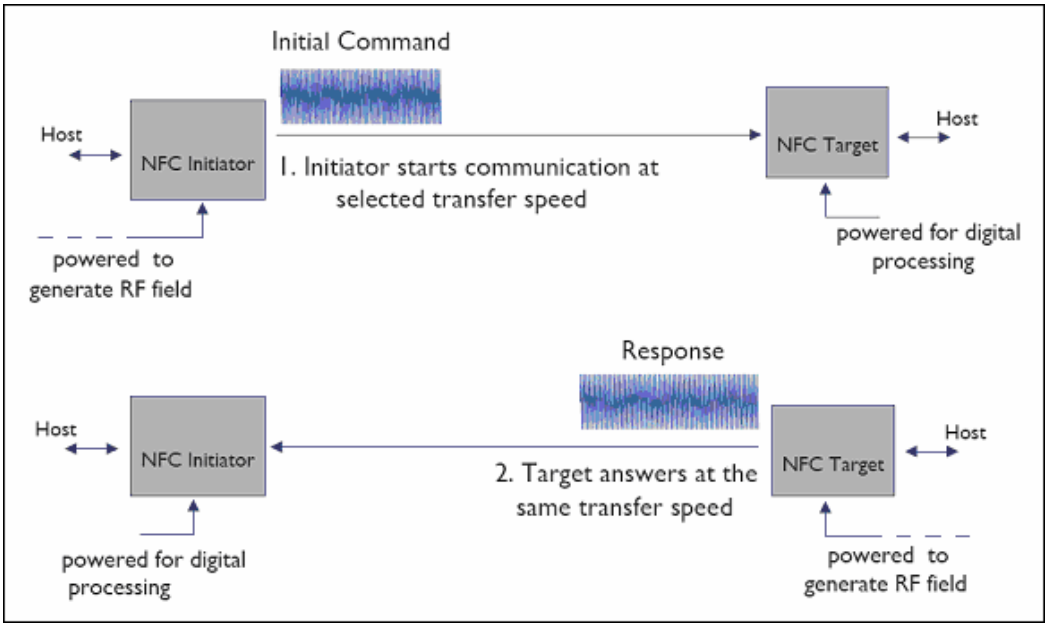



Figure 2 – Active mode of communication

- **Passive mode of communication**: Only one device generates the RF field while the other device uses load modulation to transfer the data. The protocol specifies that the initiator is the device responsible to generate the RF field.

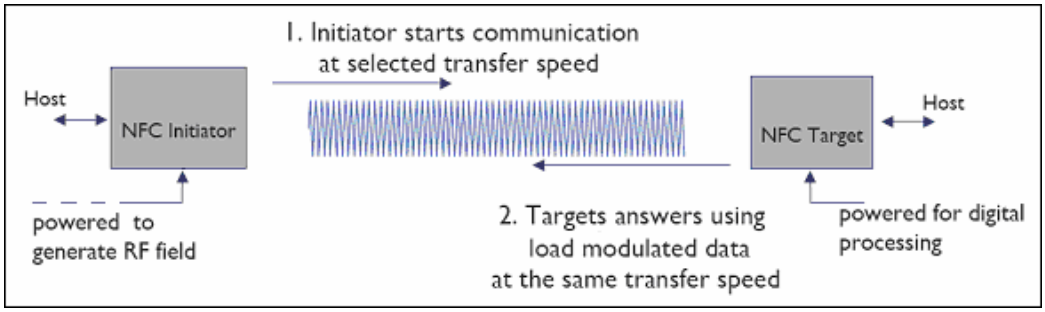



Figure 1 – Passive mode of communication

### SECURITY

A whole view through security issues in NFC transmissions shows that this is a matter which is still being investigated nowadays. Attacks have already been tested and proved [22]; an example of an attack could be the attempt to make a phone crash by changing the content of a tag that it is going to be read by it. If an attacker replaces the content of an NFC tag and adds a new one which has the value 0xFFFFFFFF in the length field of an NDEFRecord, the phone will crash. After four crashes in a row, the phone will switch off. In the other hand, companies (specially the ones that have developed NFC applications to interact with credit cards) are developing new protocols to make the transmissions safer. However, this is still an ongoing problem.

Some NFC devices are provided with a Secure Element, a smart card capable of storing multiple applications in a safe way and carrying out secure transactions.

Besides the technical security aspects and because the transmission range of NFC communications is so short, NFC-enabled transactions are inherently secure. Also, physical proximity of the device to the reader gives users the reassurance of being in control of the process.

### NFC VS. BLUETOOTH

Despite NFC and Bluetooth have some common characteristics, we can say they are complementary technologies. Both technologies are short-range communication, but Bluetooth has a longer range than NFC. In the other hand, it is more complicated to set-up the laborious Bluetooth connection, as the existing technology involves several procedures before transferring data (configuring the Bluetooth functionality as turned on, searching for new devices… ), meanwhile NFC technology makes the process very simple by activating the transfer process by a mere touch of the mobile phones. NFC requires shorter set-up time when compared to the Bluetooth and so connection between two NFC devices will be established within one tenth of a second. It is more or less similar to the wireless technology. The data transfer rate is slower in NFC (848 Kbps) when compared to Bluetooth (2.1 Mbps). NFC is highly suited for crowded areas where connectivity through Bluetooth is much problematic and connections with the partner devices can be established even in switched off conditions (contactless smart cards etc.). Therefore NFC can be used to set-up a Bluetooth connection.

| | NFC | Bluetooth |
|---|---|---|
| NETWORK TYPE | Point-to-point | Point-to-multipoint |
| RANGE | < 0.2 m | 10 m |
| SPEED | 848 kbit/s | 2.1Mbit/s |
| SET-UP TIME | < 0.1 s | 6 s |
| COMPATIBLE WITH RFID | Yes | No |

Figure 3 – NFC technology vs. Bluetooth

## NFC TAGS

An NFC tag is a passive device that stores data that can be read by an NFC-enabled device. The tags can be used within applications such as posters and other areas where small amounts of data can be stored and transferred to active NFC devices. Within the poster, the live area can be used as a touch point for the activation of the NFC device.

The stored data on the NFC tag may contain any form of data, but common applications are for storing URLs from where the NFC device may find further information.

NFC tags are devices with no power of their own. Accordingly when one is used, the user touches an NFC enabled device onto the tag, a small amount of power is taken by the NFC tag from the reader/writer to power the tag electronics and the tag is then enabled to transfer a small amount of information to the reader/writer. Finally the data stored in the tag memory is transferred to the NFC enabled device.

Figure 4 – NFC tags

## NFC FORUM

The Near Field Communication Forum was formed to advance the use of Near Field Communication technology by developing specifications, ensuring interoperability among devices and services, and educating the market about NFC technology. Formed in 2004, the Forum has 150 members now. Manufacturers, applications developers, financial services institutions and more, all work together to promote the use of NFC technology in consumer electronics, mobile devices, and PCs.

Figure 5 – NFC Forum

The goals of the NFC Forum are to:

- Develop standards-based Near Field Communication specifications that define a modular architecture and interoperability parameters for NFC devices and protocols.
- Encourage the development of products using NFC Forum specifications.
- Work to ensure that products claiming NFC capabilities comply with NFC Forum specifications.
- Educate consumers and enterprises globally about NFC.

The NFC Forum provides a highly stable framework for extensive application development, seamless interoperable solutions, and security for NFC-enabled transactions. The NFC Forum has organized the efforts of dozens of member organizations by creating Committees and Working Groups.

### NFC FORUM SPECIFICATIONS

In order that the communication between the active NFC reader/writer and the passive NFC tag was defined, the NFC forum introduced their first standardised technology architecture and standards for NFC compliant devices in June 2006. This included the NFC Data Exchange Format (NDEF), Record Type Definitions (RTD), four NFC Tag types and the Connection Handover.

- **NFC Data Exchange Format (NDEF)**: NDEF is a lightweight binary message format designed to encapsulate one or more application-defined payloads into a single message construct. An NDEF message contains one or more NDEF records. These records can be chained together to support larger payloads. An NDEF record carries three parameters for describing its payload: the payload length, the payload type, and an optional payload identifier. NDEF specifies a common data format for NFC Forum-compliant devices and NFC Forum-compliant tags.


Figure 6 – NDEF Message

  - NDEF Record: A record is the unit for carrying a payload within an NDEF message. Each payload is described by its own set of parameters (length, type, identification).

- **NFC Record Type Definition**: It specifies the format and rules for building standard record types used by NFC application definitions and third parties that are based on the NDEF data format. The RTD specification provides a way to efficiently define record formats for new applications and gives users the opportunity to create their own applications based on NFC Forum specifications.
  - NFC Text RTD: Provides an efficient way to store text strings in multiple languages by using the RTD mechanism and NDEF format.
  - NFC URI RTD: Provides an efficient way to store Uniform Resource Identifiers (URI) by using the RTD mechanism and NDEF format.

  - NFC Smart Poster RTD: Defines an NFC Forum Well Known Type to put URLs, SMSs or phone numbers on an NFC tag, or to transport them between devices. The Smart Poster RTD builds on the RTD mechanism and NDEF format, and uses the URI RTD and Text RTD as building blocks.
  - NFC Generic Control RTD Technical Specification: Provides a simple way to request a specific action (such as starting an application or setting a mode) to an NFC device (destination device) from another NFC device, tag or card (source device) through NFC communication.

- **NFC Forum Tag Type**: The NFC Forum has mandated four tag types to be operable with NFC devices. This is the backbone of interoperability between different NFC tag providers and NFC device manufacturers to ensure a consistent user experience. The operation specifications for the NFC Forum Type 1/2/3/4 Tags provide the technical information needed to implement the reader/writer and associated control functionality of the NFC device to interact with the tags. Type 1/2/3/4 Tags are all based on existing contactless products and are commercially available. The main differences between them are related to available memory, the standard they are based, their capabilities –read, read-only, re-writable– and their communication speed.

- **NFC Forum Connection Handover**: Defines the structure and sequence of interactions that enable two NFC-enabled devices to establish a connection using other wireless communication technologies. Connection Handover combines the simple, one-touch set-up of NFC with high-speed communication technologies, such as WiFi or Bluetooth.

### N-MARK

The N-Mark has been introduced by the NFC Forum. It consists in a stylized 'N' that is designed to enable consumers with NFC phones to quickly and easily spot embedded NFC tags.

The N-Mark is global in scope and is available for use in applications that are compliant with the NFC Forum's NDEF specification and where the tags used meet the NFC Forum Type 1/2/3/4 Tag Operation specifications


Figure 7 – NFC Forum, N-Mark logo

Within a year, the NFC Forum will also be introducing a second logo, the Certification Mark, which will be used to identify devices that meet NFC Forum device specifications and have passed a yet-to-be-determined certification process.

### NFC NOWADAYS

Trials of new technology around the world have successfully illustrated how people carrying mobile phones with built-in NFC technology can make purchases, gain access, get directions, exchange information and buy transportation simply by touching them to NFC-enabled devices embedded in information kiosks, retail registers, gate readers, advertising signs, vending machines and other devices and systems. The uses and applications of NFC are endless and many exciting ideas are in development now.

Nowadays, many cities have already tested pilot applications of NFC technology, i.e.:

- Payment, parking and tourism applications were tested in Caen (France), 2005.
- Electronic ticketing and access to Manchester City Football Club Stadium, in Manchester (United Kingdom), 2006.
- NFC equipped soft drink vending machines in Japan, 2006.
- Access to lecture halls, laboratories and garage, payments in the cafeterias and at vending machines in the University of Applied Sciences campus, in Hagenberg (Austria), 2006.
- Ticketing application for public transport in Berlin, 2007.
- Ticketing and payment applications in London Underground stations and in buses, in London (United Kingdom), 2008.
- Verification of the identity of the security guards in the UK Military Force, nowadays.

### NFC IN THE FUTURE

It is predicted that in a span of 2 to 4 years, around twenty percent of all mobile phones will be NFC-equipped, which implies that several hundred million NFC-enabled phones and devices will be deployed worldwide by 2010, and over 700 million phones will use NFC services such as payments and interaction with the data stored in 'smart objects' by 2013.

## A.1.2. JAVA

Java refers to a number of computer software products and specifications from Sun Microsystems that together, provide a system for developing application software and deploying it in a cross-platform environment. Java is used in a wide variety of computing platforms from embedded devices and mobile phones on the low end, to enterprise servers and supercomputers on the high end. Java is nearly ubiquitous in mobile phones, web servers and enterprise applications, and while less common on desktop computers.

An edition of the Java platform is the name for a bundle of related programs, or platforms, from Sun Microsystems, which allow for developing and running programs written in the Java programming language. The platform is not specific to any one processor or operating system, but rather an execution engine (called a virtual machine – Java Virtual Machine-) and a compiler with a set of standard libraries that are implemented for various hardware and operating systems so that Java programs can run identically on all of them. Nowadays, Java Platform is comprised of four editions which consist in some

sets of profiles: Java Card, Java Platform Standard Edition (Java SE), Java Platform Enterprise Edition (Java EE) and Java Platform Micro Edition (Java ME).

In the following paragraphs, some of these Java Platform editions are going to be explained, regarding the requirements of the current project.

### JAVA ME

Java ME, also known as J2ME in its previous versions, is a Java API that provides a robust, flexible environment for applications running on mobile and other embedded devices —mobile phones, personal digital assistants (PDAs), TV set-top boxes, and printers. Java ME includes flexible user interfaces, robust security, built-in network protocols, and support for networked and offline applications that can be downloaded dynamically. Java ME defines a limited version of the JVM, due to the limited size of mobile devices in regards to memory and resource availability.

Java ME can be divided into three elements: a configuration, a profile, and optional packages. A configuration contains the JVM (not the traditional JVM, but the cut-down version) and some class libraries; a profile builds on top of these base class libraries by providing a useful set of APIs; and optional packages, are an optional set of APIs. These packages are actually not packaged by the device manufacturers, and they have to be packaged and distributed with the application developed. The configuration and profile are supplied by the device manufacturers and are embedded in the devices.

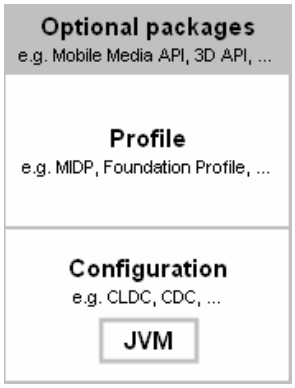



Figure 8 – The Java ME stack

The most popular profile and configuration that Sun provides are the Mobile Information Device Profile (MDIP) and Connected Limited Device Configuration (CLDC), respectively. CLDC is for devices with limited configuration, such as devices that have only 128 to 512 kB of memory available for Java applications. Consequently, the JVM that it provides is very limited and supports only a small number of traditional Java classes. CLDC was thought to be the base for more advanced profiles. In the other side, the Connected Device Configuration (CDC) is for devices with at least 2 MB of memory available and supports a more feature-rich JVM, also known as KVM, but still not a standard JVM. MIDP provides a profile based on CLDC.

The MIDP 2.0 API is comprised of the following packages:

- `javax.microedition.midlet` –life-cycle of the applications
- `javax.microedition.lcdui` –user interface package
- `javax.microedition.lcdui.game` –games package

- `javax.microedition.io` –network support package
- `javax.microedition.pki` –public key package
- `javax.microedition.media` –sound/video/multimedia package, compliant with Mobile Media API
- `javax.microedition.media.control` –specific control types which can be used in the API Media player
- `javax.microedition.rms` –presistence package
- `java.lang` –from Java SE
- `java.util` –from Java SE

### JAVA ME APPLICATIONS: MIDLETS

A MIDlet is a Java application framework for the MIDP that is typically implemented on a Java-enabled mobile phone or other embedded device or emulator.

MIDlets are packaged and distributed as MIDlet suites. A MIDlet suite can contain one or more MIDlets. The MIDlet suite consists of two files, an application descriptor file, with a .jad extension, and an archive file, with a .jar extension. The descriptor lists the archive file name, the names and class names for each MIDlet in the suite, and other information such as location, size, configuration and profile requirements of the JAR file. The archive file contains the MIDlet classes and resource files.

MIDlet applications are subclasses of the `javax.microedition.midlet.MIDlet`, this means a MIDlet always have to extend from `javax.microedition.midlet.MIDlet` class. The methods available in this class allow the application to be created, started, stopped and destroyed. The life-cycle of a MIDlet is comprised of three states: paused, active and destroyed. It just can be in one of them at the same time.

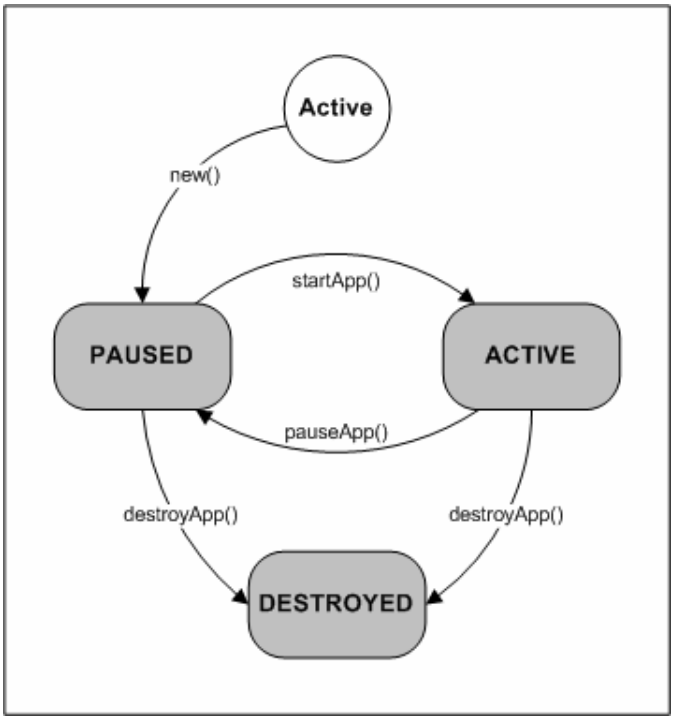



Figure 9 – MIDlet life-cycle

### MIDP 2.0 PUSH REGISTRY

"Push" is a very powerful concept and typically refers to the mechanism or ability to receive and act on information asynchronously, as information becomes available, instead of forcing the application to use synchronous polling techniques that increase resource use or latency.

The push registry enables MIDlets to set themselves up to be launched automatically, without user initiation. The push registry manages network- and timer-initiated MIDlet activation; that is, it enables an inbound network connection or a timer-based alarm to wake a MIDlet up. It is part of the application management system (AMS), the software in the device that is responsible for each application's life-cycle (installation, activation, execution, and removal). The push registry is the component of the AMS that exposes the push API and keeps track of push registrations.

With the PushRegistry API a MIDlet can be registered for push events (inbound network connections and time-based alarms), discover whether the MIDlet was activated by an inbound connection, and retrieve push-specific information for a particular connection.

The push registry is part of the Generic Connection Framework (GCF) and is encapsulated within a single class, `javax.microedition.io.PushRegistry`.

The presence of push registry activation does not change the MIDlet life-cycle, but it does introduce two new ways a MIDlet may be activated: by inbound network connections and by timer-based alarms, as we can see in the next figure.

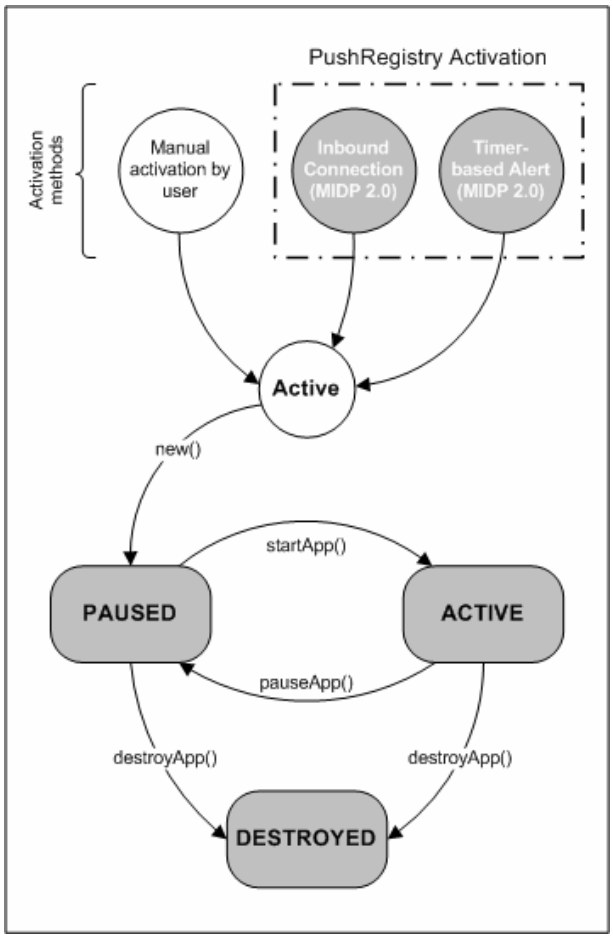



Figure 10 – MIDP 2.0 MIDlet life-cycle and activation

To become push-enabled, MIDlets must register with the push registry, using one of two types of registration:

- **Static Registration**: Registrations of static (well known) connections occur during the installation of the MIDlet suite. They are specified by listing MIDlet-Push attributes in the MIDlet suite's JAD file or JAR manifest. The installation will fail if an address that's already bound is attempted to register. Uninstalling a MIDlet suite automatically unregisters the connection.

- **Dynamic Registration**: Dynamic connections and timer alarms are registered at runtime.

In MIDP 2.0 the responsibility for push is shared between the MIDlet and the AMS. Once a MIDlet has registered itself with the push registry, the responsibility for push processing is split as follows:

- If the MIDlet is not active, the AMS monitors registered push events on behalf of the MIDlet. When a push event occurs, the AMS activates the appropriate MIDlet to handle it. The following figure illustrates the sequence for a network activation.

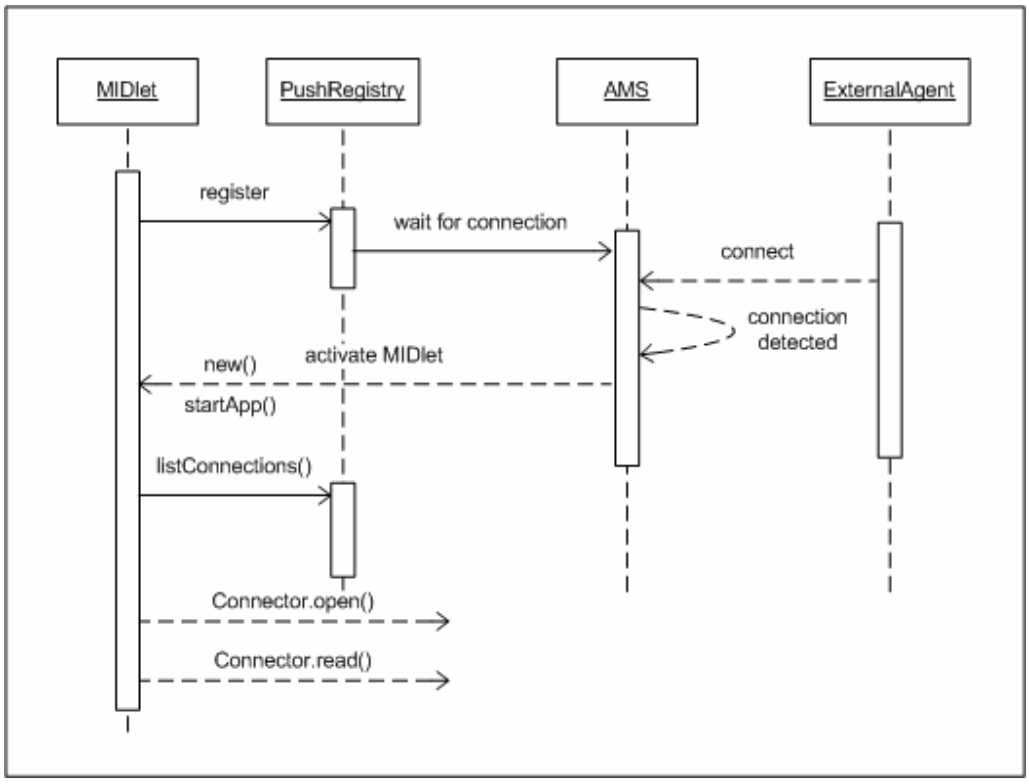



Figure 11 – Network activation sequence diagram

- If the MIDlet is active (running), the MIDlet itself is responsible for all push events. It must set up and monitor inbound connections, and schedule any cyclic tasks it needs - basically standard networking and timer processing.

## JAVA EE

Java EE, also known as J2EE till the current version was released, is a platform which consists of a set of services, APIs and protocols for developing, building and deploying Web-based applications.

Java EE services are performed in the middle tier between the user's machine and the enterprise's databases and legacy information systems. Java EE comprises a specification, reference implementation and set of testing suites. Its core component is Enterprise JavaBeans (EJB), followed by JavaServer pages (JSPs) and Java servlets and a variety of interfaces for linking to the information resources in the enterprise.

### JAVA SERVLETS

Java servlets are Java programming language objects that dynamically process requests and construct responses. The Java Servlet API allows a software developer to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets can maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting.

The servlet API, contained in the Java package `javax.servlet`, defines the expected interactions of a Web container and servlet. A Web container is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for managing the life-cycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

A Servlet is an object that receives a request and generates a response based on that request. The basic servlet package defines Java objects to represent servlet request and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package `javax.servlets.http` defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the Web server and a client. Servlets may be packed in a WAR file as a Web application.

The lifecycle of a servlet begins when it is loaded into Application Server memory and ends when the servlet is terminated or reloaded.

The whole process is as follows:

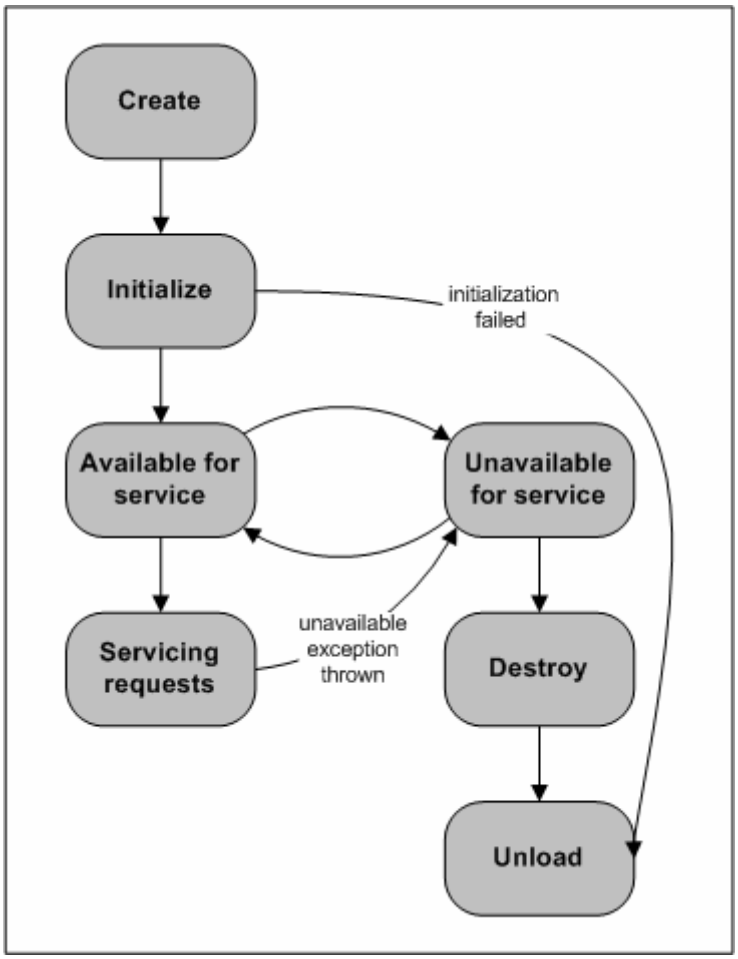



Figure 12 – Servlet life-cycle

### JSR 257 CONTACTLESS COMMUNICATION API

The Contactless Communication API allows Java developers to use RFID technology to read and to write data to various types of tags or to exchange data with other Near Field Communication enabled devices. Contactless Communication API also provides an interface to read data stored on visual tags (bar codes) and to generate visual tag images. This API provides access to the information stored to different contactless targets. It also enables bi-directional peer-to-peer connection between two NFC enabled devices. The API makes it easy and flexible to add new contactless targets to this framework. It also provides information about the contactless targets supported by an implementation.

The new functionalities offered by this API are located in the package `javax.microedition.contacless`.

### KXML 2 API

kXML 2 is a lightweight Java-based XML parser designed to run on limited, embedded systems such as personal mobile devices or MIDP devices. It is based on the common XML pull API. It is a pull parser, which means it reads a little bit of the document at

once. Then the application drives the parser through the document by repeatedly requesting the next piece.

### A.1.3. APACHE TOMCAT

Apache Tomcat is a servlet container developed by the Apache Software Foundation. Tomcat implements the Java servlet and the JavaServer Pages (JSPs) specifications from Sun Microsystems, and provides a “pure Java” HTTP web server environment for Java code to run.

### A.1.4. MYSQL

My SQL is a relational database management system. It runs as a server providing multi-user access to a number of databases. My SQL is used in web applications and acts as the database component.

## A.2. BACKGROUND

The NFC Forum has specified the Generic Control Record Type Definition in order to provide *a simple way virtually to request any specific action to an NFC Forum device (destination device) from another NFC device, tag, or card (source device) through NFC communication*. The purpose of this record type is to avoid unnecessary external record types created by developers.

The structure of the Generic Control Record is as follows: first is a target record identifying the target function or application, second is an action record specifying the desired action for the function identified by the target record and the last record is a data record specifying data to be processed by the function identified by the target record.

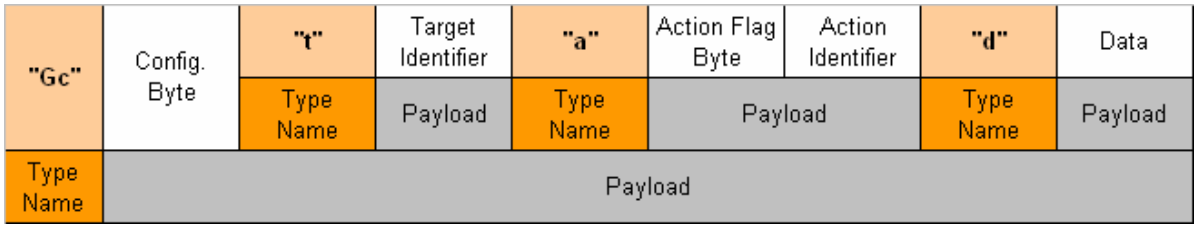



Figure 13 – The structure of a Generic Control Record

The NFC Forum has also defined a “Record Handling Architecture” that should be implemented by the destination device and that is used to launch an appropriate target function.

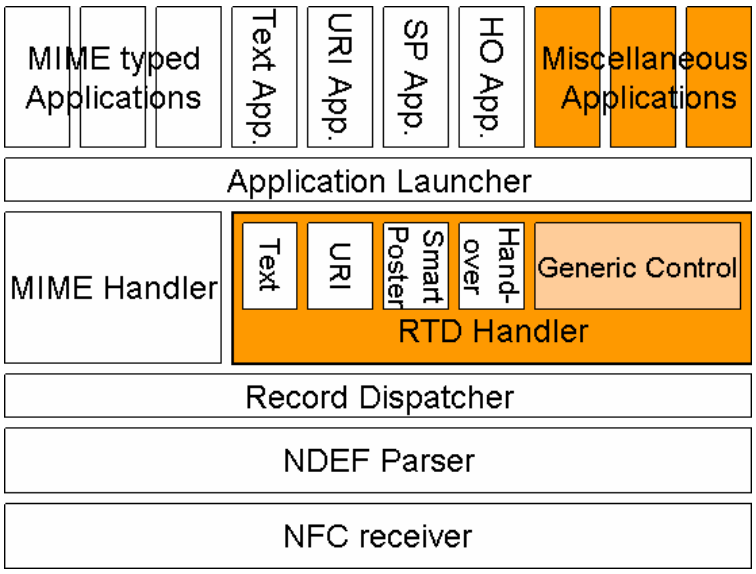



Figure 14 – Record Handling Architecture

# A.3. PROBLEM DESCRIPTION

NFC applications are developed assuming that the reader devices (NFC enabled devices) which will interact with them are able to understand the target function, and so launch the appropriate service in order to make it work as desired. NFC applications are developed more and more each day and users are more and more interested in wireless technology, but new services appear that cannot be read by mobile phones because the devices are not ready yet to process any kind of tag. Nowadays, in order a tag may be read, it is required that the appropriate application or service was already installed in the destination device. If not, the mobile phone will not be able to process the data in the tag.

The NFC Forum Technical Specification says that in case the destination device does not understand the target function, then *the device SHOULD simply ignore the Generic Control record that contains the unresolved Target record*. This way of response may pose significant problems for end users as their phones may not support some tags by default and they may not know how to "make them work", how to download the required services, which applications are needed for interacting with some specific NFC tags or where to look for them.

As we can see, this fact may cause some objections for both end users and tag developers; regarding end users because they will not be able to interact with the tags and get the required information and functionality from them, and regarding the companies because they will miss the opportunity to share their services with new users.

# A.4. SCOPE AND DELIMITATIONS

The objective of this project is to develop a tool that is able to solve the problems found for end-users to read any kind of NFC tags using their mobile phones, and so give them the possibility to interact with any tag, whatever they are and whatever applications that users have installed in their devices. The application that is going to be developed will help the end-users when their devices are not able to process the information read from the tag by finding a solution to get the needed service, in order to process the content on the tag in the desired way.

Once the application is installed, the NFC device will have the ability to know what kind of resources are required and where to find them, so it is then important that the end-users are able to automatically download any applications needed in a user-friendly and transparent way. Once the NFC device has found the location of the needed resource, it will notice the users in some way (event…) to make them know about the availability of the application. The users will be free to decide if they want to follow the process or not. If so, they will be able to interact with the tag, which was the desired scope.

Since the field of mobile platforms has a big amount of possibilities nowadays, it is necessary to find the one that better fixes with the requirements of this project. The main aspects that were borne in mind are the availability of NFC technology and the possibility to test and evaluate the result and previous tests in real mobile devices. As a result, the J2ME platform has been chosen, since it supports NFC development and it can be tested in a device which is already available to be used.

# A.5. METHODOLOGY

Software prototyping has been chosen as framework for developing the activities that comprise this project. This choice makes possible the fact of detecting faults in the requirements of the product.

Several steps will be taken in order to accomplish all the objectives and get the final scope:

- **Requirements identification** –analysis
- **Study of different technologies** –analysis

↓ Incomplete specifications

- **Prototype identification and design** –design
- **First prototype development and implementation** –implementation
- **Prototype testing and evaluation** (clients) –testing and feedback

↓ Complete specifications

- **Product implementation** (gradual expansion from user experience)

Once the distribution of tasks is set, it is required to distribute them along the available time: a period of time of 8 months, but in which 2 of them are not going to be working time.

As a result, the Gantt diagram obtained is as follows:

| ID | Task Name | Duration | Start |
|---|---|---|---|
| 1 | **Pre-study [~4 weeks]** | **20 days** | **Mon 19/01/09** |
| 2 | Near Field Communication technc | 4 days | Mon 19/01/09 |
| 3 | NFC Record Type Definition | 2 days | Fri 23/01/09 |
| 4 | Generic Control RTD | 2 days | Tue 27/01/09 |
| 5 | NFC Data Exchange Format | 2 days | Thu 29/01/09 |
| 6 | Java ME | 3 days | Mon 02/02/09 |
| 7 | Java EE | 3 days | Thu 05/02/09 |
| 8 | Code examples | 4 days | Tue 10/02/09 |
| 9 | **Analysis [~3 weeks]** | **14 days** | **Mon 16/02/09** |
| 10 | Use Case diagrams | 4 days | Mon 16/02/09 |
| 11 | Activity diagrams | 5 days | Fri 20/02/09 |
| 12 | Class diagram | 3 days | Fri 27/02/09 |
| 13 | Test Cases | 2 days | Tue 10/03/09 |
| 14 | **Design [~1 week]** | **18 days** | **Thu 12/03/09** |
| 15 | SW Architecture | 4 days | Thu 12/03/09 |
| 16 | HW Architecture | 4 days | Wed 18/03/09 |
| 17 | GUIs | 10 days | Tue 24/03/09 |
| 18 | **Implementation [~7.5 weeks]** | **51 days** | **Tue 07/04/09** |
| 19 | Prototype implementation | 30 days | Tue 07/04/09 |

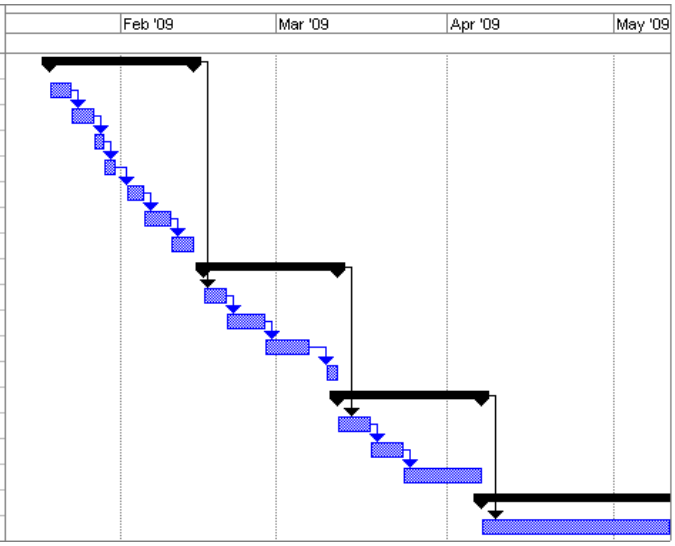



Figure 15 – Gantt diagram, part I

| ID | Task Name | Duration | Start |
|---|---|---|---|
| 20 | Tests and evaluations | 5 days | Thu 28/05/09 |
| 21 | Feedback | 4 days | Thu 04/06/09 |
| 22 | Prodcut implementation | 12 days | Wed 10/06/09 |
| 23 | **Documentation [~4 weeks]** | **26 days** | **Fri 31/07/09** |
| 24 | Documentation | 21 days | Fri 31/07/09 |
| 25 | Presentation | 5 days | Mon 31/08/09 |
| 26 | **Presentation** | **1 day** | **Mon 07/09/09** |

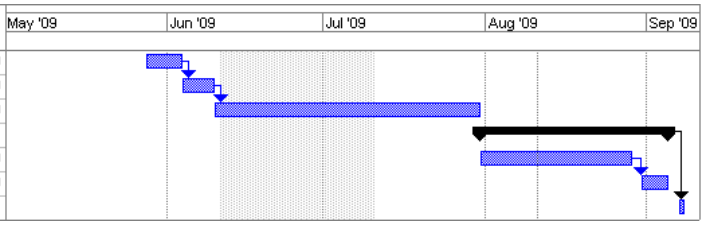



Figure 16 – Gantt diagram, part II

As we can see from the diagram, the project is going to be finished around the first week of September 2009. This will make 6 months of work, in order to get the desired results.

# ANNEX B. ANALYSIS

## B.1. REQUIREMENTS

It is advisable to study and specify all the constraints that are going to be present in the project. The requirements are divided in four groups, regarding the field they are connected to:

### PRODUCTS

- **Application that writes an NDEF record on an NFC tag**: This is not a requirement, but it must be done in order to test the main application. This application will ask for an empty tag to be touched, and it will write on it some data. Once this process is done and the tag is ready, the phone mobile will be able to read the tag using its NFC reader and so the Application Discoverer will be launched.

- **Application Discoverer**: it is going to be developed in order to solve the main scope of the project. It consists of an application which will be installed in a phone mobile. It will be launched when a specific tag, with specific data written on it, was touched by the device. Once this tag is read, the application will get some data from it and some configuration parameters from the phone, and will download some service in order to make the device available to interact with the tag in the desired way.

The flow diagram situated under these lines shows the path that the whole process must follow. It must be seen also as a requirement since the final product must satisfy its constraints.

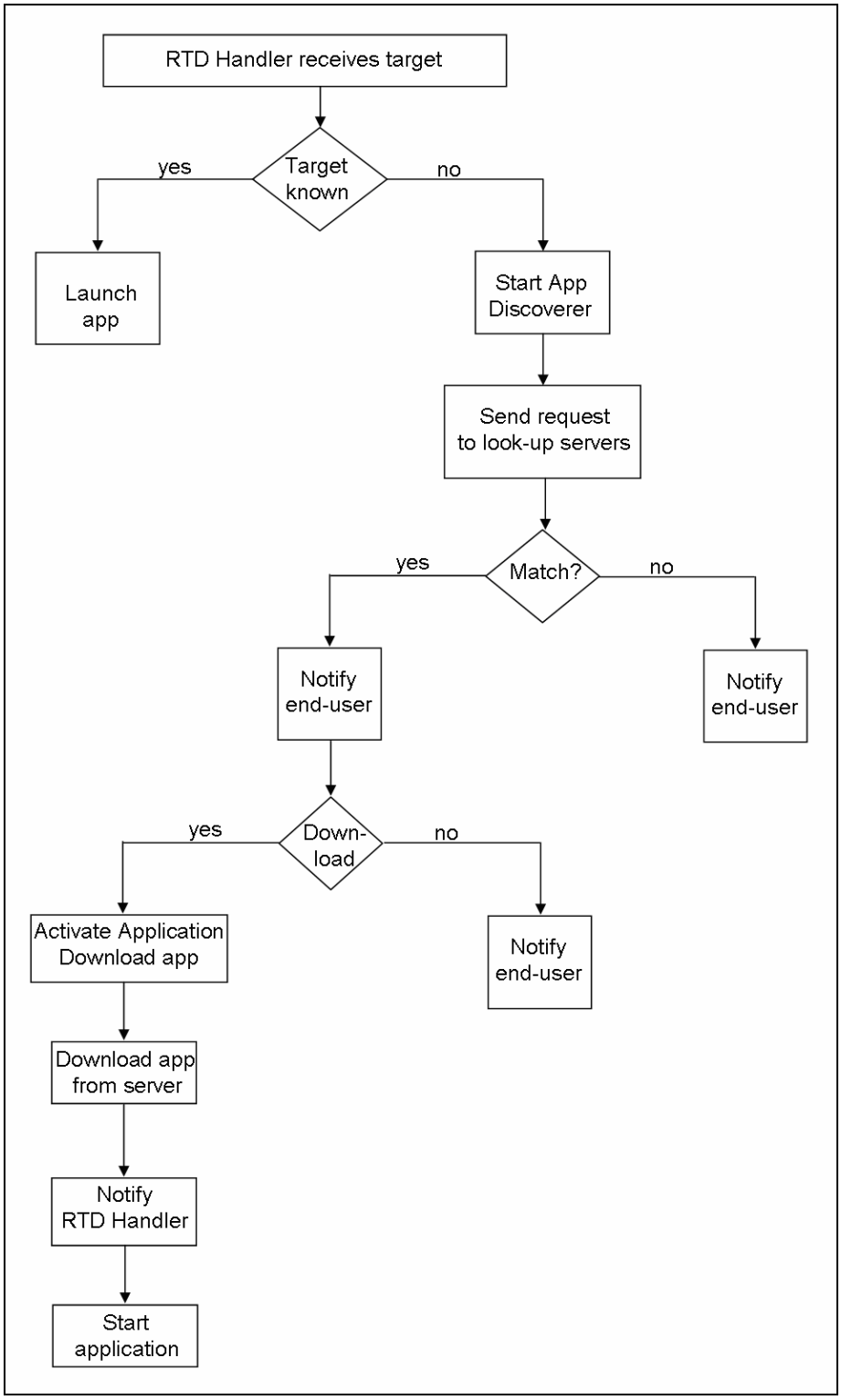



Figure 17 – Flow diagram

- **Web site to upload services**: To make the whole process faster and easier, a web site is required. The companies and organizations that want end users get the benefit of the Application Discoverer with their applications, will be able to use this site to write some information about the service they want to upload and also upload the required files. Once this process is done (that can be done in just few minutes), the new services will be available to be downloaded by end users (also the NFC tags should be written in the appropriate way to make this possible).

#### FUNCTIONALITIES

- Automatic process, started just after touching an NFC tag (Push Registry method)
- Transparent process to end user

#### GUI

- Easy to use and understandable

#### TECHNOLOGIES, METHODS AND SPECIFICATIONS

- NFC: process starter
- NFC Forum specifications –NFC Data Exchange Format, NFC Record Type Definition, NFC Forum Tag Type, NFC Forum Connection Handover
- Java programming language –basic code
- JSR-257 Contactless Communication API
- Java ME: code regarding mobile aspects
- Java servlets: processing requests –server side
- RESTful Web Services in Java –requests and responses to/from the server
- HTML –code regarding the web page
- CSS –design code regarding the web page
- XML –transferring several services from the server to the phone mobile
- PushRegistry –setting MIDlets to be launched automatically

#### TOOLS

- Eclipse SDK, v3.4.2 –development IDE
- Sun Java Wireless Toolkit v2.5.2_01 for CLDC
- Mobile Tools for Java v0.9 –mobile device Java application development support
- Nokia 6131 NFC SDK v1.1 –phone emulator
- Nokia PC Suite v7.1
- Apache Tomcat v6.0 –servlet container
- MySQL v5.1 –database where services and their information are stored

#### DEVICES

- Nokia 6131: testing device
- NFC tags: testing device

# B.2. POSSIBLE SOLUTIONS

Some approaches were thought in order to find the best solution to the problem. The most important are going to be described in the next paragraphs.

**URIs embedded in tags**: The URI where the service or application to be downloaded and installed, in order to make the mobile device interact with the tag, is written on the tag. When the end-user touches the NFC tag it gets the web address where to surf to get the required resource. The end user has to download the service and install it. The mobile phone would be ready to interact with the NFC tag now.

**Target embedded in tags**: The target function to be launched in the mobile device is written in the tag, as well as the data to be read and processed by the target function. If the mobile device does not have the target function installed, another application (ApplicationDiscoverer) is launched, which looks for the required service and downloads and installs it into the device, so the user can now interact with the tag.

## B.2.1. SOLUTION ADOPTED

"URIs embedded in tags" is a good approach but a static solution. If some external modifications occur, the tag would be unusable. These external events could be the change of the location of the service, the change of its name, the server down of the machine where service is located…

"Target embedded in tags" is a more dynamic solution. It finds a customized solution for each end-user since it can deal with some different parameters (language, country…). External changes do not affect the tag since it does not look for a specific service which is located in a specific place. It first checks the available services, and shows the end-user the most suitable ones for the characteristics of his/her mobile phone.

# B.3. SCENARIOS

After some weeks of studying and reading about the technologies that are going to be implemented in the project, some scenarios have been designed in order to give real solutions and uses to the product that it will be developed. These scenarios have been thought, pros and cons have been discussed and some of the most useful ones are going to be explained in the next paragraphs:

- **Smarts posters at the cinema**: Tags located at the main entrance of a cinema or in visible places. By touching the tag, end-users could get the information of the last released movies, movies to be released soon, special kind of movies (action, comedy…), movies schedule… regarding the tag that is touched. The NFC device

would get the information about the group of movies selected and try to show it to the end-user: director, actors, synopsis… and the trailer. The NFC device will be ready to download some Video Player in case that it is not already present in the mobile phone. Also some newer versions, plug-ins, drivers, codecs…

- **News/headlines/weather**: Tags located along the main streets in cities. By touching them, end-users could get the information about the most important news of the day or maybe just the headlines of them, the forecast for the coming days ... Special applications could show the data: PDF readers with links to access to further info…

- **Tourism offices**: Tags located at the main entrance of tourism offices, but in the outside, in order to be available out of the opening hours. By touching them the end-users could get the information about some interesting tourist destinations or sightseeing of the city. Also places around it. It could be several tags with different options. The end-user's NFC device would get, if needed, some application to manage data and to show it in a map, where the most interesting places were marked… An important configuration to pay attention on would be the language of the application.

- **Machines for booking last-minute tickets at train/bus stations**: Tags could be located close to a screen where next buses or trains are displayed. By touching the tag the end-user could book the last-minute ticket. Some application for showing the map of the train/bus in order to choose the seat, and the application to execute the purchase could be needed.

# B.4. USE CASE DIAGRAM

After gathering all the requirements that are needed to implement the application, some use cases are developed in order to get an overall and structured view of the system.

Six actors have been identified in the system:

- End-user: person who will start the process, by touching an NFC tag.
- NFC tag: device that contains the information to be read by the user's device.
- Mobile phone, which is the device that is going to perform the process by starting an application and running it, in order to be able to interact with the tag.
- Lookup Service, which is the machine where the application launched in the mobile phone is going to search for the appropriate data, and so be able to find the required service in the application server.
- Application Registry, where the data regarding available services to be download is stored.
- Application Server, which is the machine where the services are going to be stored. The application launched in the phone, by getting the appropriate URI, is going to download the required service from this server.

Two main use cases have been identified in the process:

- Get data: operations required to get the parameters from the phone and from the NFC tag to create the link that will search the appropriate service to interact with the tag through the mobile device.
- Process service: operations required to search the appropriate service and download it from the application server.

The Use Case diagram that gathers all the previous information is as follows:

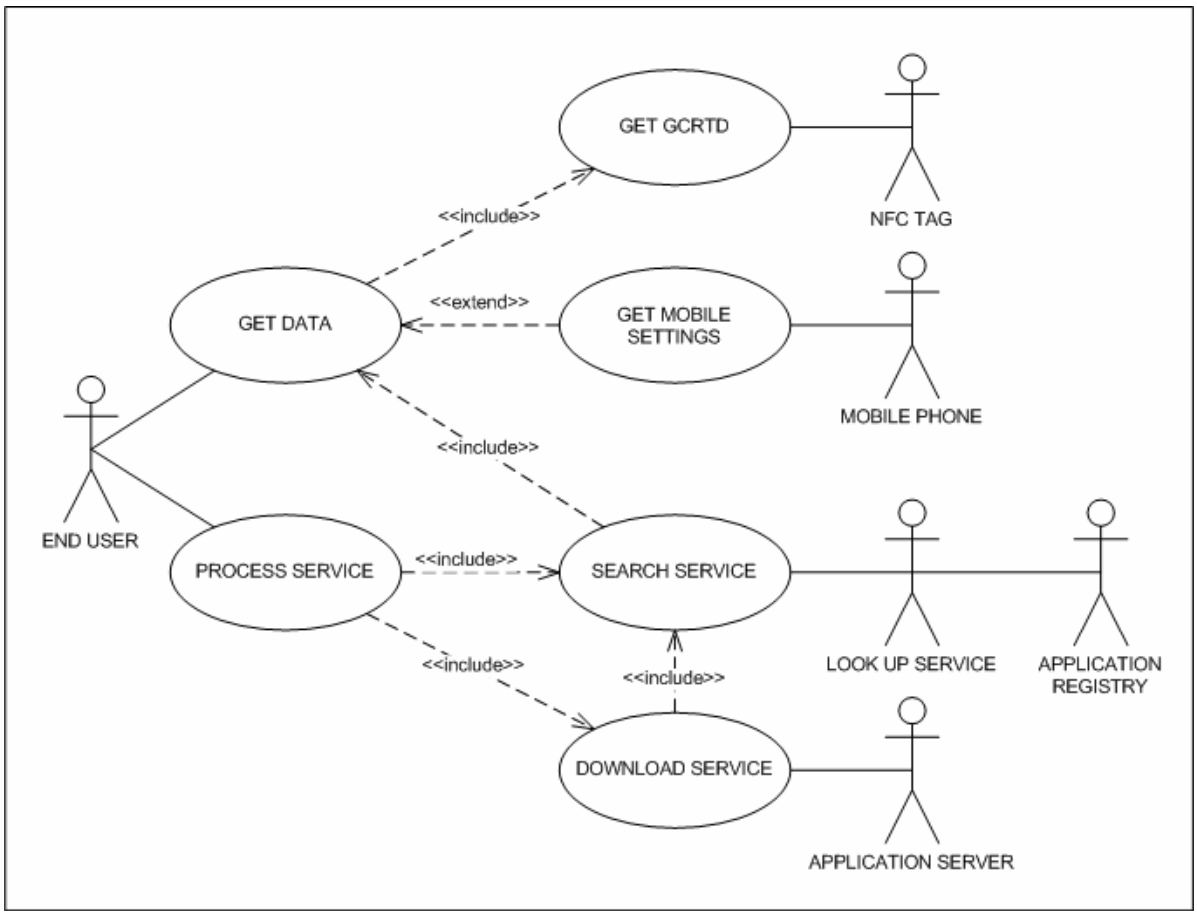



Figure 18 – Use case diagram

# B.5. ACTIVITY DIAGRAMS

Once the use case diagram is designed, the activity diagrams are the next step to do. The most significant use cases are selected, and an activity diagram is designed for each one of them. As a result, three diagrams have been created, regarding the use cases “Get GCRTD”, “Search service” and ”Download service”.

### ACTIVITY DIAGRAM FOR THE USE CASE “GET CGRTD”

To get the GCRTD is required that the tag was touched by the mobile phone so the application is launched. This way, the connection to RTD handler is established, and the data in the tag is transferred to the mobile device. The phone should extract then the required information (target, action, data) from the record.

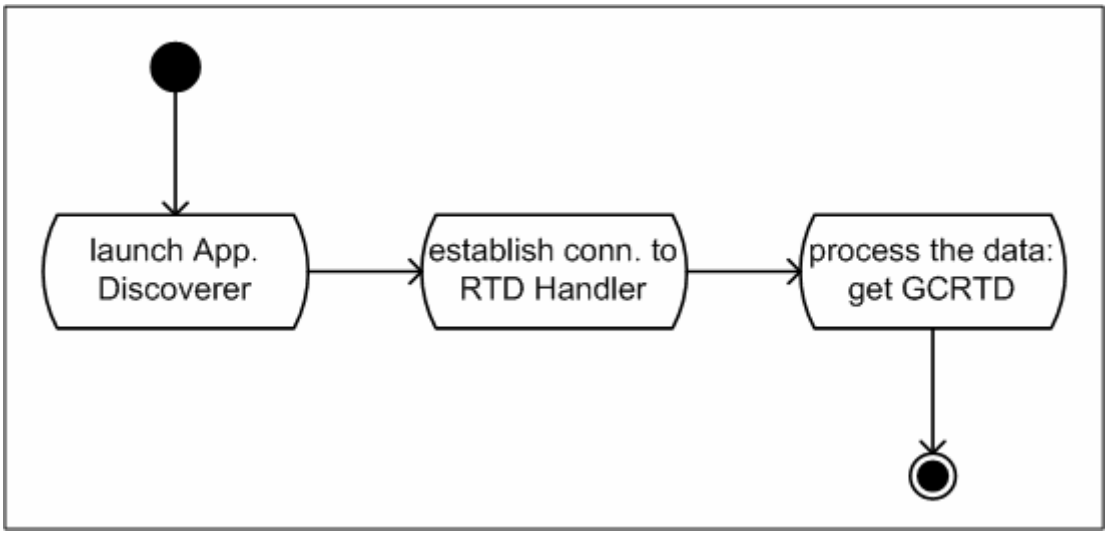



Figure 19 – Activity diagram: Get GCRTD

### ACTIVITY DIAGRAM FOR THE USE CASE "SEARCH SERVICE"

The operation that regards to search the service starts when the data (GCRTD and phone settings) is obtained. With this information, a link is created and sent to the Lookup Service, where the appropriate SQL requests are executed. Whether there are some matches (URIs) or not, the response is sent back to the mobile phone and the result is shown to the end user.

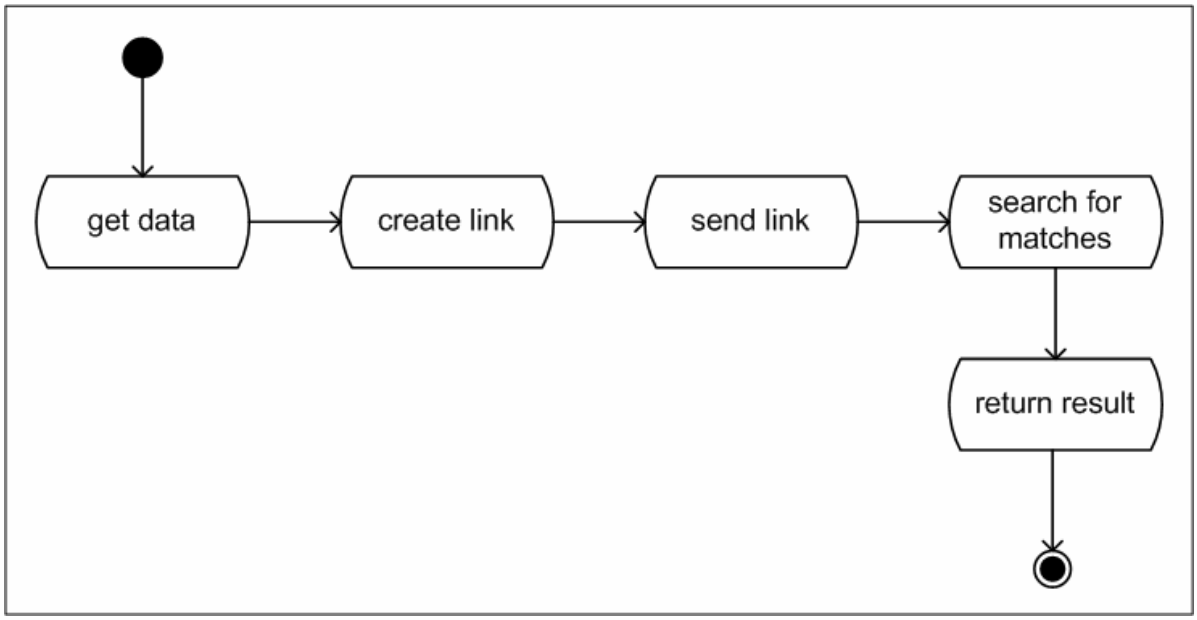



Figure 20 – Activity diagram: Search service

### ACTIVITY DIAGRAM FOR THE USE CASE "DOWNLOAD SERVICE"

In order to get the service, it is required an URI where to download it. Once the desired URI is chosen, a connection to the server is established and so the service can be downloaded.

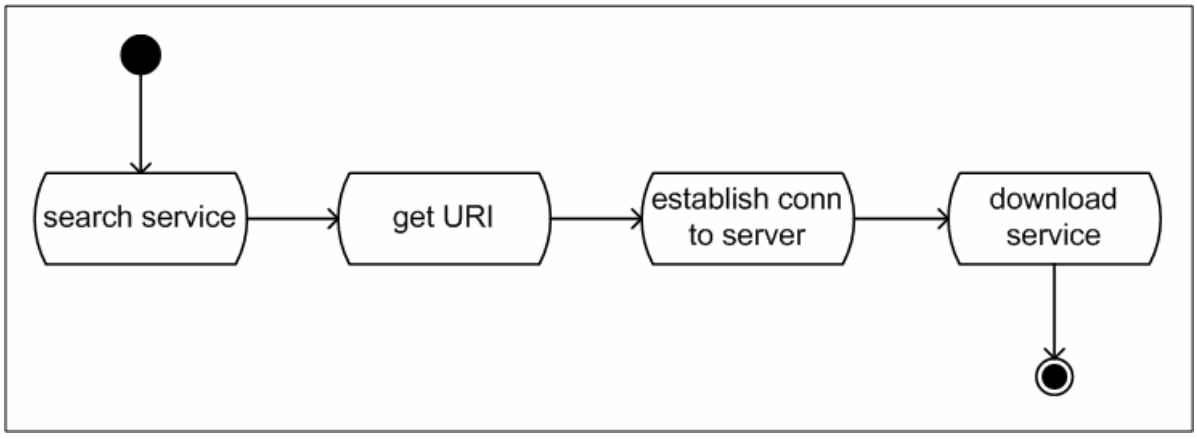



Figure 21 – Activity diagram: Download service

# B.6. SEQUENCE DIAGRAM

Once the activity diagrams are designed, a sequence diagram is required in order to show how processes operate with one another, in what order and when objects are created and destroyed.

The diagram situated below these lines shows the different objects that comprise the system, when they are created and when destroyed. The GUI is the object that initiates the process, and it is guided by the user's events.

Looking through this diagram we can get an idea about how the system works and how are the interactions between the different objects which comprise it.

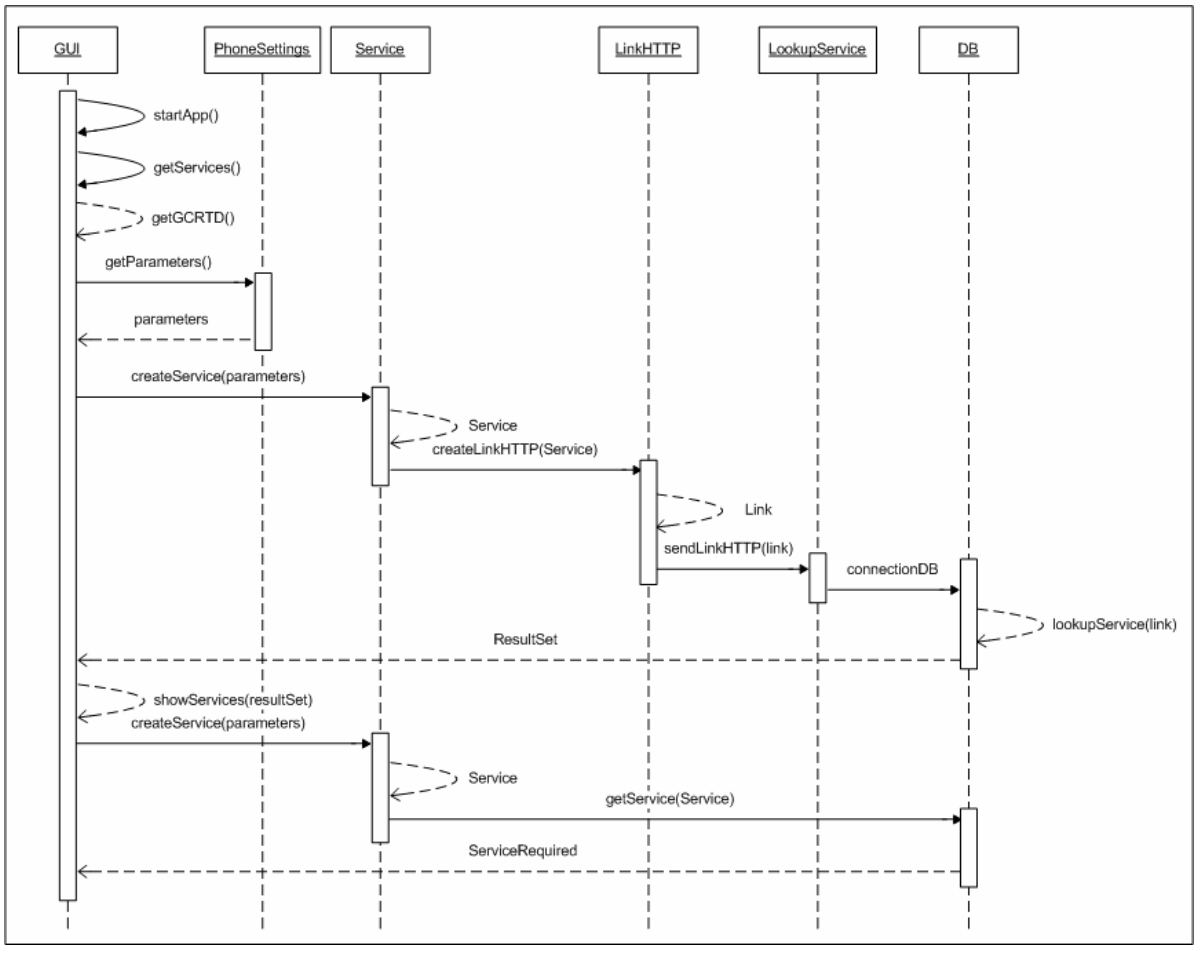



Figure 22 – Sequence diagram

# B.7. CLASS DIAGRAM

The class diagram is a static structure diagram that describes the system by showing its classes, the relationships between them and, in this case, the operations of each one. The class diagram for the current project is as follows:

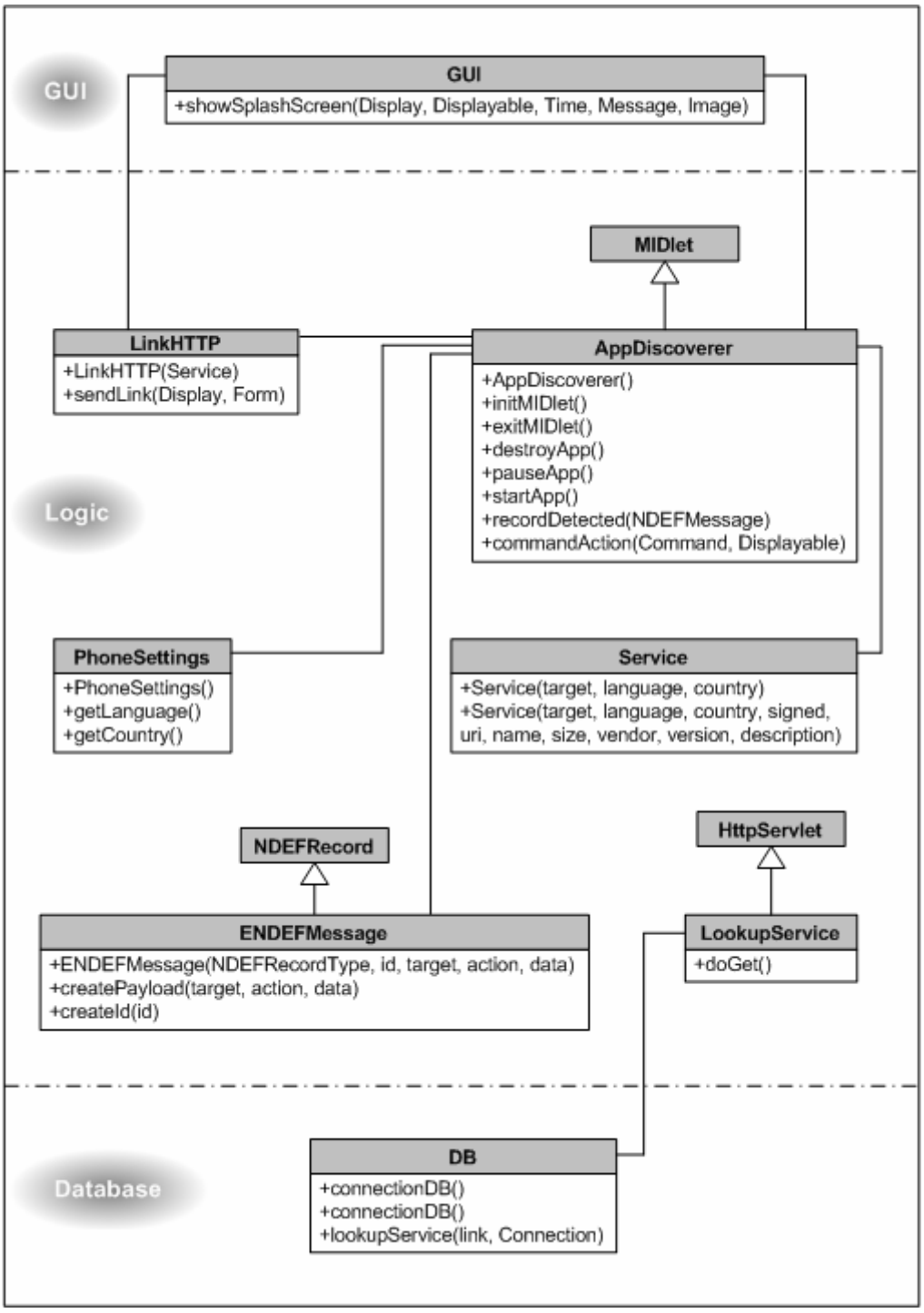



Figure 23 – Class diagram

# B.8. EVALUATION TESTS

Before the implementation stage and after the design stage, some evaluation tests will be done to end users. They will be asked to test a prototype on a real device.

The prototype will be comprised of graphic user interface screens, with no functionality, and the users will be able to test the application as if it was the real one. Just after these tests, they will be asked to fill a short form, in order to get some feedback to make changes, if needed, in the application flow.

The template of questions that were asked is as follows:

**QUESTIONS TO BE ANSWERED BY END USER, AFTER TESTING THE GUI OF THE APPLICATION DICOVERER (2009-mm-dd)**

- Do you think that the messages (error, warning and informative messages) displayed by the application are helpful enough?
  ☐ Yes ☐ Not enough ☐ No
- Was it easy and understandable to use and surf through the application?
  ☐ Yes ☐ Not enough ☐ No
- Do you think it is a useful application?
  ☐ Yes ☐ Not enough ☐ No
- Would you use it in the future, if it was allocated in your phone mobile?
  ☐ Yes ☐ Probably ☐ I do not think so ☐ No
- Would you allow the connection to the Internet in order to get the services needed?
  ☐ Yes ☐ Probably ☐ I do not think so ☐ No
- In your opinion, is something missing that you would like to have seen?

  _____

  _____

  _____

- Mobile phone with which the application was tested: _____
- Your phone mobile: _____

Figure 24 – Evaluation test

# ANNEX C. SYSTEM DESIGN

## C.1. HW ARCHITECTURE

The architecture of the system consists of six components:

- **NFC tag**: RFID tag which contains some information to be displayed in an end user's mobile phone.
- **NFC enabled mobile device**: Device that starts the process by touching an NFC tag. Once the tag is recognized and the data (record) is read from the tag, the phone checks if the appropriate service is installed, so it can process the data. If not, the Application Discoverer is launched: a link containing some parameters from the record and the mobile device is then sent to a lookup service. The device must be provided with Internet connection (WiFi) and be NFC enabled.
- **Lookup Service**: Web server where the links sent from the mobile devices are processed. It stores the Java servlets that are executed when an HTTP connection is detected. In order to process these petitions, the server may make a connection to a database.
- **Application Registry**: Database where the information about available services is stored.
- **Application Server**: Server where available services are stored. Once mobile devices have the appropriate URI to get the resource, they make a connection to this server, so the service is downloaded.
- **Personal Computer**: Computer used to upload new applications and their info to the Application Server and Application Registry respectively. It connects to the Lookup Service, and this server stores information and files in the appropriate places.

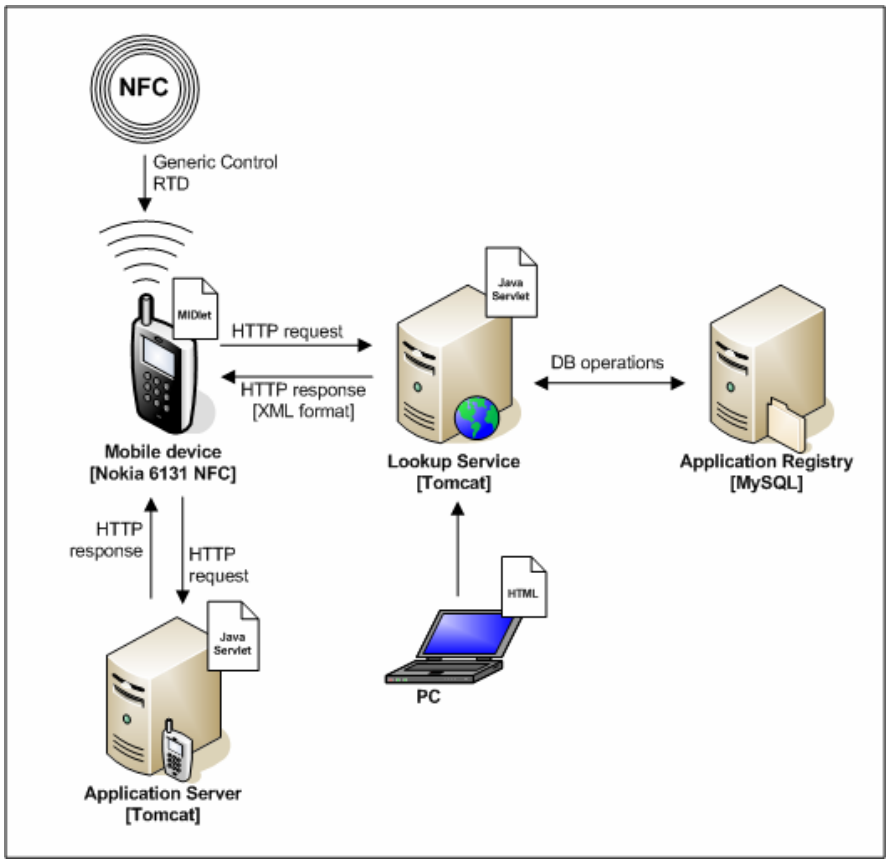



Figure 25 – System architecture

In the previous picture, some aspects regarding SW architecture have been added. References to them are going to be done in the following section (see 3.2 C.2SW Architecture).

# C.2. SW ARCHITECTURE

The whole application has been developed as a three-tier structure; this means the presentation tier, the logic tier and the data tier have been developed and can be maintained as independent modules, which provides scalability and performance. In the following paragraphs, each one of the three parts in which this project can be structured, are going to be described, regarding their software architecture.

- **Application Discoverer**: This is the application that will be launched in a mobile phone when this device is not able to interact with an NFC tag, using the services that it has already installed. The three tiers are implemented as follows:

- Data tier: The data regarding the services available to be downloaded are stored in a MySQL database server. It is spread in two different locations. The data regarding the information about services is stored in the Application Registry, and the services are stored in the Application Server. Both databases can be stored in the same server or be located in two different machines.
- Logic tier: It is deployed between the Apache Tomcat server and the mobile device. Java servlets and classes regarding data access –which are called by servlets- lie in the server, and classes regarding other functionalities such as data structures, lie in the device side. Logic (servlets and data access operations) and data tiers are physically located in the same machine, though they can be separated without causing any problem. There is only one required modification that should be done: the references from the servlets to data, such as URIs or IPs address, should be replaced with the appropriate ones.
- Presentation tier: It is located in the mobile device and it comprises some graphic classes and MIDlet extended class. It contains the code regarding visual and user interaction aspects. It also contains operations regarding the detection and interaction with NFC tags.

- **WriteNDEFMessage**: This is a help tool that allows writing in an NFC tag. It has "static" parameters: it always write the same message on the tag, though it can be changed just modifying some strings in the Java class. It comprises two tiers:

  - Logic tier: It comprises some classes regarding data structures, required for handling NFC records. This code is written in Java language.
  - Presentation tier: It contains some graphic design and operations to detect and empty tag and write on it. It is written using MIDlets technology.

- **Web page**: It is the means through which external people are able to upload services to the database, so these services would be available to be downloaded using the Application Discoverer. The web page is a three-tier application:

  - Data tier: It shares the stored data with the Application Discoverer; both applications use the same resources. The web page allows users to upload the data to the database, and the Application Discoverer downloads this data to the end users phones. As a result, de data tier is composed of the Application Registry and the Application Server.
  - Logic tier: It consists of some classes regarding data structures and a servlet located in the Apache Tomcat server, which carries out the required operations to store the information of the service in the appropriate database, and the service in the appropriate location.
  - Presentation tier: HTML code and CSS sheets have been used to implement this tier.

#### RECORD HANDLING ARCHITECTURE

There is also an interesting architecture that must be taken in account: the Record Handling Architecture. This architecture must be implemented by the mobile device that starts the whole process, and it looks as we have already seen in Figure 14, in the first chapter of the current document.

Once a record type is identified by the NDEF parser, the record data is forwarded to the record dispatcher and the dispatcher determines which handler will take care of the record. After the association between the data and a corresponding application (or function) is resolved by a handler (MIME handler for MIME type records and RTD handler for NFC Forum Global Type records), the application launcher launches the associated application, if necessary, and delivers the data to the application.

Nowadays, to get this functionality, it is required to add the Application Discoverer to the architecture by using Push Registry methods, so it will be ready to be launched when the mobile device touches the appropriate NFC tag.

The result regarding the Record Handling Architecture, after doing this operation, is as follows:

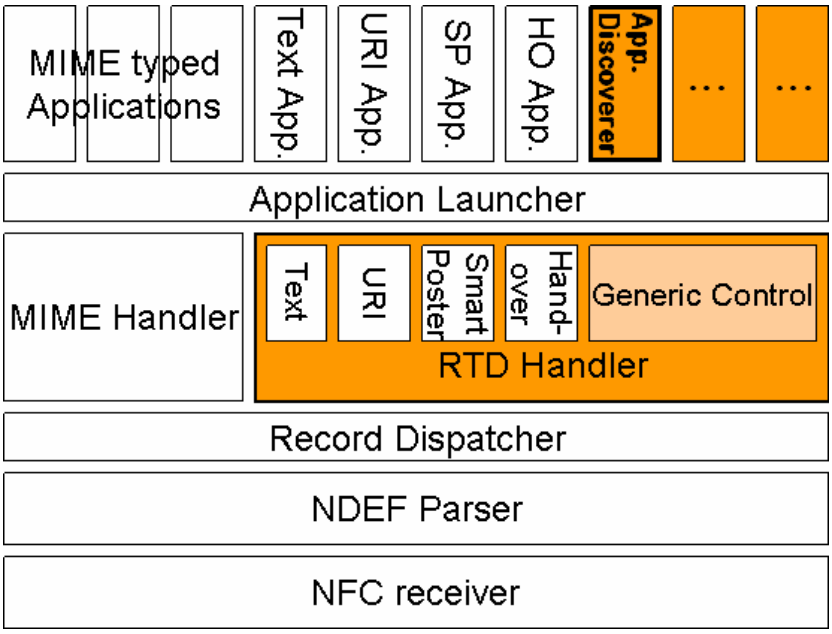



Figure 26 – Extended Record Handling Architecture

The ideal s solution is to implement the ApplicationDiscoverer as a native application on the platform. This process should be done by the device manufacturer. Thereby, the application would be more integrated into the Record Handling Architecture.

# C.3. DATABASE DESIGN

The database model is made up of one table which contains the appropriate fields where to store the data regarding services. The table is called "services" and its Primary Key if formed by four fields: name + version + language + country. All the fields are going to be detailed in the following paragraphs:

- **name, VARCHAR(20)**
  Names of the services (files). An example can be the following one:
  *service_name.jar*

- **version, VARCHAR(20)**
  It specifies the version of the application. Example:
  *2.4_13*

- **target, VARCHAR(20)**
  It represents the target of the services; this is the main parameter that the lookup service takes in account when searching for the appropriate service. It looks as follows:
  *audio*

- **size, INT(10)**
  The size of the service (files) is specified in this field; the unit of measurement must be kilobytes. Example:
  *120 –regarding 120 KB*

- **uri, VARCHAR(255)**
  It represents the path where the service is stored; the links stored in this field looks like this:
  *http://xxx.xxx.xxx.xxx/looupservice/services/service_name.jar*
  - where:
    *xxx.xxx.xxx.xxx* is the IP address of the Application Server
    */lookupservice/upload/* is the path where the service is located
    *service_name.jar* is the name of the file

- **language, VARCHAR(2)**
  It indicates the language in which the application or service is written. The data is stored in the same way as mobile phones store this kind of data; this is, as it is specified in the ISO 639-1. Example:
  *sv –regarding Swedish*

- **country, VARCHAR(2)**
  It stores the country where the service has meant to be used. As language field, the data is stored as mobile devices do. In this case, the specification is the ISO 3166-1. Example:
  *SE* –regarding Sweden

- **vendor, VARCHAR(20)**
  The field stores the vendor who has developed the service. Example:
  *Ericsson AB*

- **signed, TINYINT(1)**
  Whether it is a trusted (!0) or untrusted (0) service, it is specified in this field. In case this field is empty, it automatically be assigned the "untrusted" value. It looks as follows:
  *1* –regarding a trusted service

- **description, LONGTEXT**
  A short description of the service goals is stored in this field. It must give a global idea of the use of the service. It could look as follows:
  *Music player in HiFi quality*

The database model diagram is as follows:

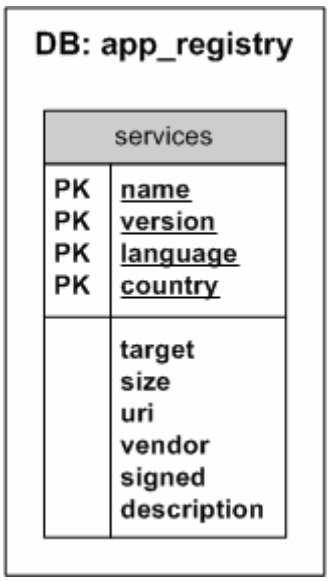



Figure 27 – Database model diagram

Since there is just one table in the database and no relations, it makes no sense to create an entity-relationship model.

# C.4. USER INTERFACES

When designing an application, the full context of use must be taken into account. One of the most important aspects in this context regards users, their tasks and goals and the environment in which they use the application. Knowing the users that interact with the application is the key when including usability thinking in application development processes.

## C.4.1. USABILITY

Def: *Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. This means that usability is not a property of the product itself - it is a property of the entire system, including the product, the user, the user's goals, and the context of use*.

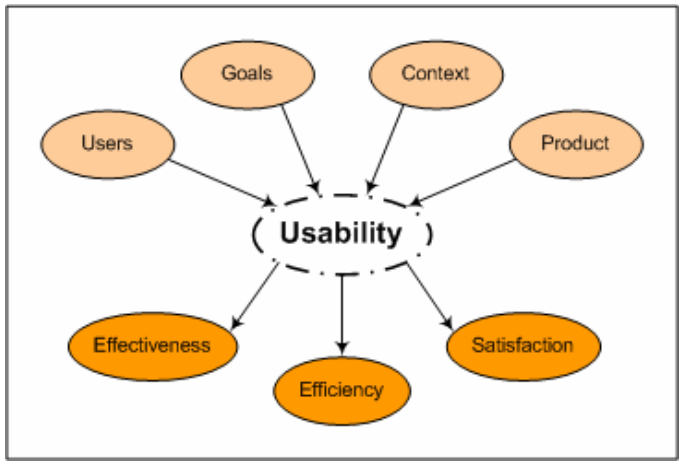



Figure 28 – Definition of usability by ISO 9241-11 (1998)

Taking into account the context of use and user goals, usability work attempts to examine and improve levels of effectiveness, efficiency, and user satisfaction, as we can see in the previous picture.

## C.4.2. VISUAL DESIGN

'Visual design' is usually tended to mean only the way something looks. In fact, visual design can play a much larger role in the success of any project. Although the visual style, personality, colour, fonts and imagery are all part of visual design, another critical aspect is the information design, which supports an application's functionality, contributes to its usefulness, and impacts its chances of success.

## C.4.3. INTERACTION DESIGN

Def: *Interaction design is the act of defining the behaviour of a product or a service. Interaction design is essentially communication: gathering product requirements, analysing user needs and turning them into design that allows users to effectively perform their tasks on a target platform or device.*

Interaction design is an essential part of the development of any product or service with a user interface (UI), whether it is a Web page, a desktop application, mobile application, or an embedded UI. Industrial design (designing the physical hardware or device) and visual design are closely connected to interaction design – interaction design must acknowledge and support the hardware characteristics, and all visual elements should have a meaningful purpose as defined in the interaction design.

## C.4.4. MOBILE VS. DESKTOP APPLICATIONS

Mobile devices and applications are used in a significantly different context than desktop applications. A mobile user may be practically anywhere, under any conditions when using the application, whereas the mobile user may be…

- … using the device while moving
- … performing another task at the same time
- … with or without a network signal, or with a slow data transfer connection
- … in an extremely dark or bright environment
- … running out of battery
- … in an extremely noisy or distracting environment

Since the design cannot be prepared for every possible situation, the key to mobile design is to keep it simple. Simplicity is achieved by focusing on the most important features and keeping the design clear and accessible. Key tasks should be able to perform with low effort and important information should be easily available.

## C.4.5. USER INTERFACE DESIGN. USABILITY TESTS

The process of designing the user interfaces was done in several steps. In the initial phases several prototypes were designed. First was a basic sketch, were global ideas were captured. This was an attempt to understand better the application, its functionalities, and its navigation. This prototype was discussed and studied in order to get some feedback to create a more developed second one, in which the usability issues were solved.

After these early designs, we got some help and comments from Didier Chincholle, an Interaction Design Senior Specialist, from the Usability & Interaction Lab Multimedia Technologies of Ericsson Research in Kista, Stockholm. He put forward his opinion about the prototype and we tried to find some solutions to the problems we found.

Aspects such as saving the application for future uses or deleting it just after the interaction with the NFC tag, structure of the messages displayed, simple and automatic process versus guided and detailed navigation… where hardly discussed and solved. It

was not easy to find proper solutions in some aspects since there are many different users with significant differences in their phone navigation knowledge level.

A third prototype was then design. This one was implemented in order that it could be displayed and tested in a real device. At this point, a usability test was done: some randomly chosen end-users were asked for several questions regarding the application. They first had to test the prototype, by navigating through its screens. No functionality was implemented, but a real scenario was explained to evaluators, so they could understand the goal of the application.

The usability test gave some issues that could not be seen in previous phases, and so all these aspects were discussed, once again, with the User Interaction specialist, Didier Chincholle. The result of this last meeting worked out the final prototype. Modifications were carried out and the final product, regarding screens design, was produced.

# C.4.6. USER INTERFACE DESIGN: CONCLUSIONS

Regarding usability: The goal of the main application of this project is to provide the end user with a specific service that allows him/her to interact with an NFC tag. This fact implies that what is important for the user is to get the service in a fast way, with few interactions or events. As a result, just some important decisions are going to be required from the end user, such as allowing Internet connection or choosing the service it works for him/her. Other secondary aspects are going to be omitted.

Regarding visual and interaction design: High level API has been chosen, Vs. Low level API. High level design APIs allow developers to create applications which main characteristics are the information displayed, compatibility in all devices, high usability level (commands, colours, … are going to be displayed in the same way as other applications do), … In the other hand, it is not a requirement for the application to show a special look and feel, given that the only goal for the user is to get a service that is able to interact with the tag. Summing up, keeping application's appearance with global device appearance is the best choice since the user already knows this interface, so this makes the interaction faster.

# C.5. UI: PROTOTYPES AND FINAL DESIGN

## C.5.1. PROTOTYPES

The following images correspond with the first three prototypes designed. Some of the screens are shown, and the errors that were detected in them are commented:


Figure 29 – Prototype screen

The initial idea was to display in green the services that could be the best options for the end-user, and in red colour the rest ones.

The final product will just show the services that are good options for the end-user device, so it makes no sense colouring the result services shown.


Figure 30 – Prototype screen

We thought it was a good idea to give the possibility to the end-user of saving or discarding the application, once this had interacted with the NFC tag.

The final product does not implement this functionality since it is not a common feature in mobile applications. End-users know how to uninstall applications, so it makes no sense to ask every time for it.

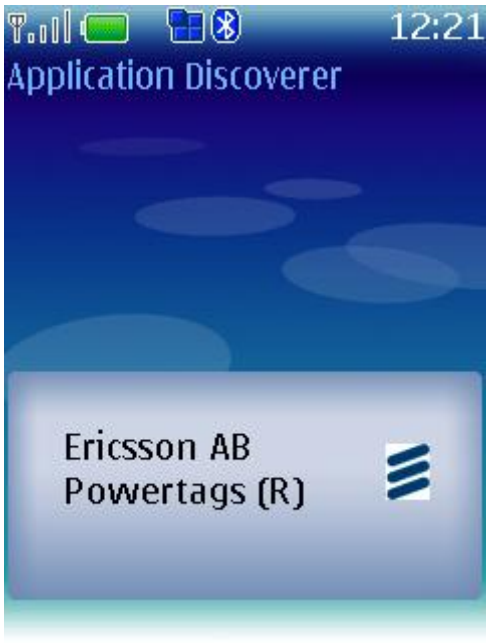

Figure 31 – Prototype screen

An application that accesses to the Internet must always ask for permission to the user. This screen was implemented also in the final product.

Just one modification was done: we realized that the estimated time was something that depends on the connection and it can not be estimated in advance.

## C.5.2. FINAL DESIGNS

The screens implemented for the final product differ from the prototypes in their appearance. They are shown as they are displayed in the real mobile device used for testing the system, Nokia 6131.




Welcome screen

Figure 32 – Final screen


Searching services screen

Figure 33 – Final screen

The final version of the web page looks as follows:

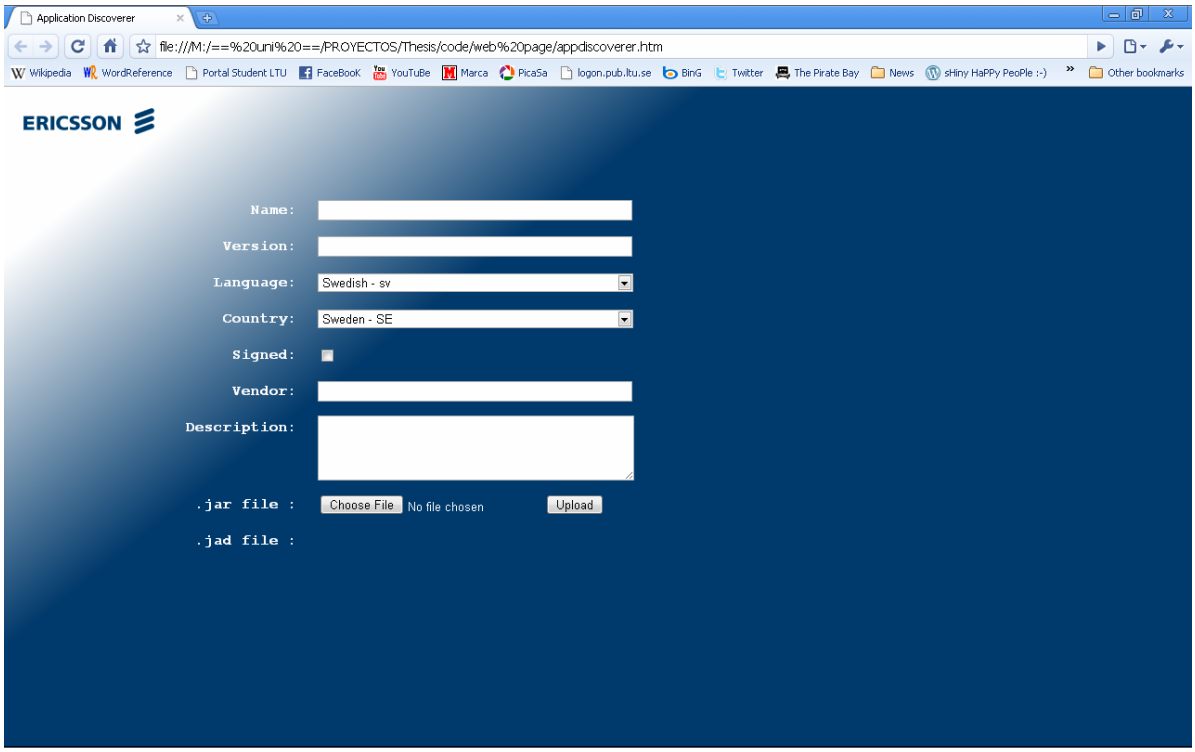



Figure 34 – Final design of the web page

# ANNEX D. IMPLEMENTATION

Along the whole development of the current project, several technologies have been used. Next some of the main aspects and important implemented functionalities are going to be detailed.

## D.1. WRITENDEFMESSAGE APPLICATION

The WriteNDEFMessage application gives the possibility to write the required NDEFMessage on an empty NFC tag. An NDEFMessage can contain one or more records (NDEF records) which are located in the payload field of the message. In turn, an NDEF record comprises three fields: record type, identification and payload. In order to follow the specifications of the NFC Forum, the NDEF messages that are written using this application have the structure of a Generic Control Record into its payload field.

As a result, the NDEF messages sent and received from/to the system contain an extended version of an NDEF record, which has been called “Extended NDEF Record”. As NDEF records do, this extended version includes a record type, an identification and a payload field. This last component, the payload field, is made up of three fields: target, action and data, which follows the rules of the Generic Control Record specification.

Since the payload component is a byte data type, the structure and its integration have been defined and designed as follows:

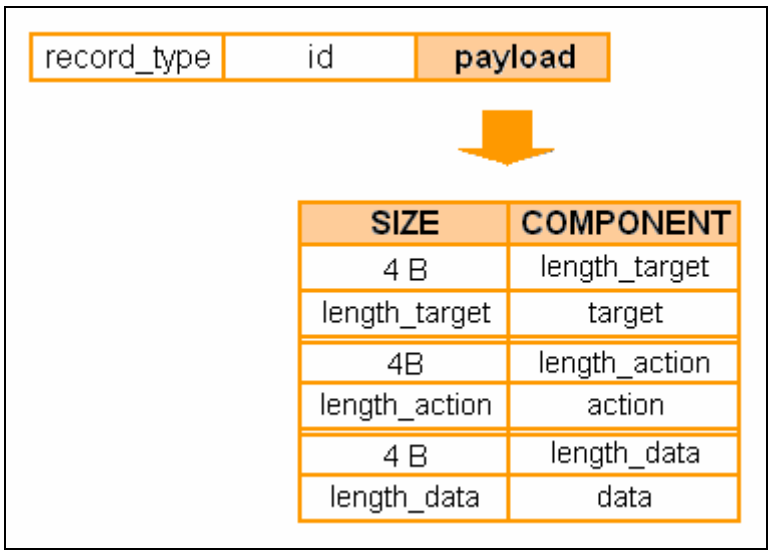



Figure 35 – NDEF Message + ENDEFRecord byte structure

The byte structure includes 3 groups of 2 fields each one. The first field of each group is a 4 byte field, and it stores a “int” data type. The reason why this fields have a static size is due to the data is sent (from the mobile phone to the NFC tag, and vice versa) as a byte stream, and using this structure, it is available to locate any of the fields in the record. If the size were variable, we could not know which bytes correspond to each field.

The byte structure is defined in the ENDEFRecord class, which extends from the NDEFRecord class. Its more relevant method is called createPayload(), which gets a String target, a String action and a String data parameters as input, and creates the payload field, as a byte component.

The way of how this functionality has been implemented can be seen in the next code lines:

```
public static byte[] createPayload(String target, String action, String data) {
      int targetLength, actionLength, dataLength;
      byte[] targetBits = new byte[target.getBytes().length];
      byte[] actionBits = new byte[action.getBytes().length];
      byte[] dataBits = new byte[data.getBytes().length];

      try {
            targetBits = target.getBytes("utf-8");
            actionBits = action.getBytes("utf-8");
            dataBits = data.getBytes("utf-8");
      } catch (UnsupportedEncodingException e) { e.printStackTrace(); }

      targetLength = targetBits.length;
      actionLength = actionBits.length;
      dataLength = dataBits.length;

      // 12 last bytes are created for the target/action/data
      length fields
      byte[] payload = new byte[targetBits.length + actionBits.length +
      dataBits.length + 12];
            // target length
      for (int i=0; i<4; i++) {
            payload[i] = (byte)(targetLength >> (8*i));
      }
      // target
      for (int i=0; i<targetBits.length; i++) {
            payload[i+4] = targetBits[i];
      }

      // action length
      for (int i=0; i<4; i++) {
            payload[i+4+targetBits.length] =
            (byte)(actionLength >> (8*i));
      }
      // action
      for (int i=0; i<actionBits.length; i++) {
            payload[i+8 + targetBits.length] = actionBits[i];
      }

      // data length
      for (int i=0; i<4; i++) {
            payload[i+8 + targetBits.length + actionBits.length] =
            (byte)(dataLength >> (8*i));
      }
      // data
      for (int i=0; i<dataBits.length; i++) {
      payload[i+12 + targetBits.length + actionBits.length] =
            dataBits[i];
      }
      return payload;
}
```

Once the identification and the payload fields are created, we just have to create the record type field. This is an important field since it is the one that is going to be used to start the communication between the tag and another NFC enabled device. It must be created based on the string that is going to be used in the Push Registry method (see next section), so both devices will be able to interact by matching both strings.

To detect an empty tag and get the connection, it is compulsory that the WriteNDEFMessage class implements the TargetListener interface, and that a TargetListener was added. Also some properties must be configured.

The code that finally creates the communication between the tag and the phone mobile, creates the Extended NDEFMessage and writes it on the NFC tag looks as follows:

```
public class writeNDEFMessage implements TargetListener {

     private DiscoveryManager dm;
     private NDEFRecordType recordType =
            new NDEFRecordType(NDEFRecordType.EXTERNAL_RTD,
            "urn:nfc:ext:ericsson.com:fare");

     public writeNDEFMessage() {
            //Add targets that are listened by the application
            dm = DiscoveryManager.getInstance();
            dm.addTargetListener(this, TargetType.NDEF_TAG);
     }

     public void targetDetected(TargetProperties[] properties) {
            if (properties.length == 0) {
                  return;
            }

            // Create a new NDEF Connection to write an NDEF EXT to the tag
            NDEFTagConnection conn = null;
            try {
                  // creates an NDEF tag connection
                  String url = properties[0].getUrl(Class.forName
                        ("javax.microedition.contactless.
                        ndef.NDEFTagConnection"));

                  // open the NDEF Connection
                  conn = (NDEFTagConnection)Connector.open(url);

                  // create the NDEFRecord and Data-Container for Tag
                  NDEFRecord record = new NDEFRecord(recordType,
                        EGCRTD.createId("ericssonId"),
                        EGCRTD.createPayload("target", "action", "data"));

                  NDEFRecord[] recordArray = new NDEFRecord[]{record};
                  NDEFMessage myMessage = new NDEFMessage(recordArray);

                  // write Information to Tag
                  conn.writeNDEF(myMessage);

                  // Writing succeeded!
            } catch (Exception e) {
                  //Error Writing to Tag!
            }
            try {
                  conn.close();
            } catch (Exception e) { e.printStackTrace(); }
     }
}
```

# D.2. APPLICATION DISCOVERER AND LOOKUP SERVICE

The developments of the Application Discoverer plus the Lookup Service provide the system the abilities to download some specific services that are required to make a mobile phone interact with an NFC tag.

### APPLICATION DISCOVERER

- pushRegistry

Since the main Java class of the Application Discoverer is a MIDlet, it has a Java Application Descriptor file (JAD file). In this file several parameters can be defined, such as the name of the MIDlets that we want to launch, the permissions we want to give to the execution, the size of the file, vendor, version… and also the MIDlet-Push-n entry (see MIDP 2.0 Push Registry in Chapter "0. Preliminary Study", for detailed information about Push Registry). Static registration method has been used in this case. The string that must be added looks as follows:

```
MIDlet-push-1: ndef:external_rtd?name=urn:nfc:ext:ericsson.com:demo,
AppDiscoverer, *
```

Where `ndef:external_rtd?name=urn:nfc:ext:ericsson.com:demo` is the ConnectionURL, `AppDiscoverer` is the MIDletClassName and `*` represents the AllowedSender (all). The generic entry is:

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
```

After establishing the entry, the permission to use Push Registry must be added into the JAD file too:

```
MIDlet-Permissions: javax.microedition.io.PushRegistry
```

The MIDlet class needs some modifications as well: an event listener must be included. To get it, an instance of the object `DiscoveryManager` is required. The code looks as follows:

```
public class ApplicationDiscoverer extends MIDlet implements NDEFRecordListener
{
     public ApplicationDiscoverer() {
            NDEFRecordType recordType =
                  new NDEFRecordType(NDEFRecordType.EXTERNAL_RTD,
                  “urn:nfc:ext:ericsson.com:demo”);
            DiscoveryManager dm = DiscoveryManager.getInstance();
            dm.addNDEFRecordListener((NDEFRecordListener)this, recordType);
     }
}
```

The application is now ready to be launched. This will happen when the end user touches with his/her mobile phone an NFC tag which was written with the same `ConnectionURL` as the string written in the Push Registry entry.

- recordDetected

Next step in the process is to get the information written on the NFC tag. In order to do it, the method `recordDetected(NDEFMessage msg)` must be implemented. In fact, the implementation of this method is compulsory, since the class implements the `NDEFRecordListener` interface. This method provides the NDEFMessage that has been read from the tag, and once we have the message, we can get the records that are included in it.

The code regarding this functionality looks as follows:

```
public void recordDetected(NDEFMessage msg) {
     NDEFRecord[] records = msg.getRecords();
     for (int r=0; r<records.length(); r++) {
            /* process the records */
      }
}
```

- getTarget, getAction, getData

In order to get the information regarding the target, the action and the data of the NDEFRecord read from the tag, we are going to use the byte structure that we have already seen in the previous section of the current document. Due to the static size of some of its fields, we can easily get each of the components.

The way that has been implemented to process the records is as follows:

```
try {
     byte[] payload = record.getPayload();
     int targetLength = 0;
     for (int i = 0; i < 4; i++) {
            int n = (payload[i] < 0 ? (int) payload[i] + 256
                   : (int) payload[i]) << (8 * i);
            targetLength += n;
     }
     // target
     byte[] btarget = new byte[targetLength];
     for (int i = 0; i < targetLength; i++) {
            btarget[i] = payload[i + 4];
     }
     target = new String(btarget, "utf-8");

     // action length
     int actionLength = 0;
     for (int i = 0; i < 4; i++) {
            int n = (payload[i + 4 + targetLength] < 0 ? (int) payload[i
                   + 4 + targetLength] + 256
                   : (int) payload[i + 4 + targetLength]) << (8 * i);
            actionLength += n;
     }
     // action
     byte[] baction = new byte[actionLength];
     for (int i = 0; i < actionLength; i++) {
            baction[i] = payload[i + 8 + targetLength];
     }
     action = new String(baction, "utf-8");

     // data length
     int dataLength = 0;
     for (int i = 0; i < 4; i++) {
            int n = (payload[i + 8 + targetLength + actionLength] < 0 ?
                   (int) payload[i + 8 + targetLength + actionLength] + 256
                   : (int) payload[i + 8 + targetLength + actionLength]) <<
                   (8 * i);
            dataLength += n;
     }
     // data
     byte[] bdata = new byte[dataLength];
     for (int i = 0; i < dataLength; i++) {
            bdata[i] = payload[i + 12 + targetLength + actionLength];
     }
     data = new String(bdata, "utf-8");
} catch (Exception ex) {
     // Exception
}
```

- getPhoneSettings

Some of the phone mobile settings are also good parameters to take in account in order to search the right service. These attributes are called *defined J2ME system property names*. They provide two services: to indicate the availability of an optional package and to

provide platform-dependent configuration data. To query system properties, `java.lang.System.getProperty()` should be used, as in:

```
System.getProperty("microedition.configuration");  // CLDC-1.0
System.getProperty("microedition.profiles");// MIDP-1.0
System.getProperty("microedition.platform");       // j2me
System.getProperty("microedition.encoding");// ISO-8859-1
System.getProperty("microedition.local");          // se_SV
```

- sendLink

The next step to be done is to create and send a link from the mobile phone to the Lookup Server. The link includes all the parameters that have been detected in the previous phases and is sent to the server. RESTful Web Services methods are used to perform the communication. The main code lines that implement this functionality are the following ones:

```
HttpConnection http_conn = null;
InputStream in = null;
int resp_code, chr;
StringBuffer sb = null;

try {
     http_conn = (HttpConnection)Connector.open(url);
     http_conn.setRequestMethod(HttpConnection.GET);

     resp_code = http_conn.getResponseCode();
     if (resp_code == HttpConnection.HTTP_OK) {
            sb = new StringBuffer();
            in = http_conn.openDataInputStream();
            while ((chr = in.read()) != -1) {
                  sb.append((char)chr);
            }
     }
} catch (Exception ex) {
     if (in != null) { in.close(); }
     if (http_conn != null) { http_conn.close(); }
     return null;
}
```

When the link connection is sent to the server using the HTTP protocol, the servlets located in the server get the request and create a response. The way this process is done is described in the next section (server side implementation), but there is a characteristic that must be mentioned now: the server sends the response (appropriate services found) using XML format. The data of the XML string is extracted by using the methods of the kXML package. It provides the needed functionality to get the required information in the required format.

In the following example, we can see how the data is parsed from the XML string:

```
Vector services = new Vector();
Service service;
String sUri = null;
boolean sSigned = false;
if (sb != null) {
     try {
            byte[] xmlByteArray = sb.toString().getBytes();
            ByteArrayInputStream xmlStream =
                  new ByteArrayInputStream(xmlByteArray);
            InputStreamReader xmlReader = new InputStreamReader(xmlStream);
            XmlParser parser = new XmlParser(xmlReader);
            ParseEvent pe = null;
            parser.read(Xml.START_TAG, null, "services");
            boolean trucking = true;
            while (trucking) {
                  pe = parser.read();
                  if (pe.getType() == Xml.START_TAG) {
                        String str = pe.getName();
                        if (str.equals("service")) {
                              while ((pe.getType() != Xml.END_TAG) ||
                              (pe.getName().equals(str) == false)) {
                                    pe = parser.read();
                                    if (pe.getType() == Xml.START_TAG &&
                                    pe.getName().equals("uri")) {
                                          pe = parser.read();
                                          sUri = pe.getText();
                                    }

                                    // ...

                                    service = new Service(sUri...);
                                    services.addElement(service);
                              } else {
                                    while ((pe.getType() != Xml.END_TAG)
                              || (pe.getName().equals(str) == false)) {
                                    pe = parser.read();
                              }
                        }
                  }
                  if (pe.getType() == Xml.END_TAG &&
                        pe.getName().equals("services")) {
                        trucking = false;
                  }
            }
     } catch (Exception ex) {
}
```

The services are shown to the end user and he/she has the opportunity to decide which one is the appropriate for his/her needs. To make this decision easier, some information is also displayed, regarding each service.

When the user chooses one service and selects the “download option”, a new connection is established between the mobile phone and the Application Server. Since what we want to get this time is a resource that is going to be installed in the device, the method platformRequest(url) is the appropriate command to perform the task. The service will be downloaded and the user will be asked for installing the service in the mobile device. If the operation is allowed, the user will have the opportunity to launch automatically the new service just installed in his/her phone.

### LOOKUP SERVICE

In the Lookup Server the servlets and the classes that implement the methods for accessing to the database are located.

- DB access

Regarding the database access two methods have been implemented: connectionDB() and disconnectionDB(). As the names suggest, the first one creates a connection to the Application Registry and the second one closes the connection. In order to make this functionality work, the appropriate DB driver must be included in the Lookup Server files. In this case, the standardized database driver for Java platform that access to a MySQL DB can be found in http://dev.mysql.com/downloads/connector/j/5.1.html.

- Java servlets

As we have already seen, the Lookup Server also includes the Java servlets. Both methods, doGet() and doPost() have been implemented. doGet() performs the requests regarding the searches of services. The doPost() method executes the requests the web page does.

doGet() also implements the functionality for creating the XML structure, based on the services found after doing the appropriate query to the database. The XML format is returned as a String. The code lines look as follows:

```
public StringBuffer lookupService(String link, Connection conn) {
     Statement stmt = null;
     ResultSet rs = null;
     StringBuffer services = new StringBuffer("");
     String query = "SELECT * FROM services WHERE ...";
            try {
                  stmt = conn.createStatement();
                  rs = stmt.executeQuery(query);
                  services.append("<services>");
                  while (rs.next()) {
                        services.append("<service>");

                              services.append("<uri>");
                              services.append(rs.getString("uri"));
                              services.append("</uri>");

                              // ...

                              services.append("</service>");
                  }
                  services.append("</services>");

                  rs.close();
                  stmt.close();
            } catch (SQLException e) { e.printStackTrace(); }

            return services;
     }
```

# D.3. WEB PAGE

- Client side

The client side of the web page is implemented in HTML. One aspect to keep in mind is that the enctype field of the form that contains the data to be uploaded, must be defined as follows:

```
<FORM ENCTYPE='multipart/form-data' method='POST' ... >
```

- Server side

The server side is located in the Lookup Service. It is comprised of a servlet file. The servlet gets the data and the files that the user has uploaded, and it stores each data in the appropriate location (Application Registry and Application Server).