WOLF: a Research Platform to Write NFC Secure Applications on Top of Multiple Secure Elements (With an Original SQL-Like Interface)

Anne-Marie Lesas, Serge Miranda, Benjamin Renaut, Amosse Edouard

WOLF: a Research Platform to Write NFC Secure Applications on Top of Multiple Secure Elements (With an Original SQL-Like Interface)

(July 2014, LSIS / MBDS Research Report)

Anne-Marie Lesas, PhD student with Gemalto
and LSIS research lab of Aix-Marseille University,
MBDS innovation lab at the University of Nice – Sophia-Antipolis, France

Pr. Serge Miranda
Director of MBDS Master degree and innovation lab (www.mbds-fr.org)
University of Nice Sophia Antipolis, France

Benjamin Renaut,
F1RST Research Engineer & Project manager
MBDS innovation lab
University of Nice – Sophia-Antipolis,
TOKIDEV, Nice, France

Amosse Edouard, PhD student
I3S Research laboratory
University of Nice – Sophia
Antipolis, France

***Abstract*—This article presents the WOLF (Wallet Open Library Framework) platform which supports an original interface for NFC developers called "SE-QL". SE-QL is a SQL-like interface which eases and optimizes NFC secure application development in making the heterogeneity of the Secure Element (SE) transparent. SE implementation could be "embedded" (eSE) in the mobile device, or inside the SIM Card (UICC), or "on-host" software-based, or in the Cloud (e.g. through HCE); every SE implementation has its own interface(s) making NFC secure-application development extremely cumbersome and complex. Proposed SE-QL solves this problem. This article demonstrates the feasibility and attractiveness of our approach based upon an original high-level API.**

Keywords—Mobiquitous services; Near Field Communication (NFC); Secure Element (SE); Smart card; Structured (English as a) Query Language (SQL); Digital Wallet; TSM / OTA; UICC

I. INTRODUCTION: WOLF AND SE-QL

Based upon both our Near Field Communication (NFC) know-how and our SQL (Structured Query Language for databases) expertise [1], we propose a generic innovative Framework called WOLF (Wallet Open Library Framework) for facilitating Service Providers (SPs) in the process of development and deployment of new NFC secure applications on a wide range of smartphones having different SE implementations.

In this article we give an overview of WOLF platform developed at MBDS innovation laboratory, which allows NFC developers to interact easily with an application based on NFC Card Emulation mode by using SQL-like language, "SE-QL". WOLF framework allows SPs to reduce NFC development costs and the time-to-market, improve and ensure the quality of products applications for new secure NFC services.

WOLF extends "NFC Container" project [2], [3] previously developed at MBDS in 2008-2010 for J2ME cell phones. Its SE-QL interface idea stems from a long-term research background on SQL and database (DB) systems; SE-QL simplifies NFC service development by abstracting the core software complexity associated with the management of multiple SE environments.

WOLF supports the secure ecosystem of F1RST project in India with Tata Consulting Services (TCS), Gemalto and the Indian Institute of Sciences (IISc) of Bangalore under research contract from IFCEPAR | CEFIPRA (www.cefipra.org). F1RST encompasses a portfolio of NFC financial and rural inclusion use cases for unbanked people in India (70% of them owning a cell phone) managed within a F1RST wallet developed by TCS on top of WOLF platform. F1RST primary goals were to demonstrate Financial Inclusion (FI) services for unbanked people based upon:

- Virtual "mobiquitous money" [4],
- The appeal of NFC standard both to strategic use cases (Financial Inclusion, Narega, Mobiquitous NFC Public Distribution System "M-PDS" [5], [6], [7]), and disruptive ones (e-coins, rural animal bank with crowdfunding "BARTER2.0" [8]).

WOLF has been successfully tested in the development of M-PDS use case prototyped at MBDS since 2012 and it is integrated into F1RST generic platform.

We have been working on the Android SE-QL interface with the delivery of WOLF API which is compliant with SIMAlliance Open Mobile API (OMAPI) with Gemalto SE-UICC, as well as Android Host-based Card Emulation (HCE) using WOLF generic applet; this work was presented at the WIMA's research track conference in Monaco (April, 2014) [9], at the Indo-French Conference in New Delhi (October, 2013), and at the Indo-French Center for the Promotion of Advanced Research (IFCPAR | CEFIPRA) meeting in St Malo (May, 2014).

The remaining of this document is organized as follows: in section II, we do a synthetic state of the art around the NFC standard and NFC secure application development; we also discuss the NFC Container project, its context, scopes and benefits for WOLF contribution. In section III, we study the proposed SE-QL interface and the WOLF API based upon related works and existing technologies. WOLF and SE-QL implementation are also described in this section with some uses cases concerning the F1RST project. In conclusion, we summarize the benefits of this research platform and present some promising extensions

II. NFC Ecosystem and Secure NFC Application Development Around the SE

NFC is a very fast establishment and very short-range (for security purposes) contactless communication technology and world standard since 2004. NFC uses the inductive coupling making a source device (acting as initiator) able to provide energy and exchange data with a target passive device (without need battery) by backward induction. NFC will be a standard in next generation smartphones.

Our future will be "mobiquitous" [2], [3], [10] around the convergence of mobility of cell phones becoming computers (smartphones) and ubiquity of Internet (becoming social and broadband)); NFC is an underlying technology and standard supporting mobiquity. NFC connectivity induces five additional dimensions to enrich information services (the five "W"). "Who": the identity of the end-user (with habits, preferences and POIs), "Where & When" space and time ("here and now", when tapping), "Whereabout" the goal, expected result (information? transaction?) and the "What", the final outcome (information, ticket transaction, appointments, service, triggering mechanisms, etc.) [2]. Unlike contactless smart cards services, NFC mobile services benefit from features of the smartphones: network high connectivity, embedded sensors, location-based ecosystem, camera, high-definition and touchscreen user interface bring NFC services at a higher level of expectation and innovation, extending SP information system to the end-user allowing screens, real time interactivity, personalization, traceability and live updates without storage limitation (and Cloud synchronization).

In terms of tracking the smartphone coupled with NFC at least enables to get 3-dimensional transaction identification: space, time and biometrics (as demonstrated in [3]).

NFC standard can be classified into two categories regarding NFC applications: (i) those that do not require security using NFC read / write mode (m-tourism, marketing 2.0… Similar to QR codes) or NFC P2P mode (e.g. initiate Bluetooth® pairing), and (ii) those that need to store confidential data and do transactions in the secure environment provided by the Secure Element (SE) using the NFC card emulation mode (digital identity, ticketing, couponing, electronic money / m-payment, access control, transactions etc.) where the NFC device acts as a smart card.

In our research we focus on the latter case, i.e. on NFC card emulation mode which requires the NFC service and sensitive data to be hosted in the SE.

A. NFC standard

The NFC standard is a very revealing technology of the expected convergence between the worlds of telecommunications, consumer electronics and computing. It is one of the numerous RFID standards (known technology since the 1940s) operating over the unlicensed 13.56 MHz frequency of up to ten centimeters (one to four in practice). It is a wireless technology which could be integrated into mobile phones allowing them to exchange information with other devices (mobile phone, printers, locks or any NFC card readers) and NFC tags (ISO / IEC 14443 - NXP MiFare - Type A or Type B, and Sony-FeliCa).

Since initial standard specification in 2006 by the NFC Forum (www.nfc-forum.org, funded in 2004 by Nokia, Sony, and Philips semiconductors, now NXP) of NFC standard, many specifications have increased the attractiveness of this world standard especially since it is widely available on Android devices.

NFC Forum specifications apply to the physical and data link layers of Open System Interconnection model (OSI) whereas GlobalPlatform (GP) widely contributes to the standardization of SE security architecture, internal and external mechanisms, etc., and Trusted Environment Execution (TEE). GP is the main reference for SE standardization; GSMA and ETSI also play an important role, as the SIM-centric model is the only one to be standardized end-to-end. Most popular SEs, at the moment, are SIM-based.

The NFC standard (ISO / IEC 14443) has three basic operating modes:

- Read / Write: The NFC cell phone acts as an active reader and can read and / or write data to or from a passive NFC tag.
- NFC Peer-to-Peer (P2P): Allows two NFC devices to be active and exchange information interactively.
- Card Emulation: The phone behaves as well as a passive contactless card (EMV, American Express, access card, Ticketing...) for a NFC reader (POS).

B. NFC card emulation mode with APDU messages

The card emulation mode is the extension of contact-based smart cards to contactless NFC cards; it inherits the smart card programing standards, especially the small data packets called APDUs (Application Protocol Data Unit) used to communicate with the services hosted and running in the smart card / SE (also called "applet" or "cardlet"). This requires a high-profile developer with strong expertise in communication protocols to handle low-level data structures at the byte level and integrate strong environmental constraints related to the execution environment of the SE.

APDU protocol was originally specified in the Java Specification Request (JSR) 177 (Security and Trust Services API for J2ME™) taken up by smart cards standard ISO / IEC 7816-4 (Identification Cards - Integrated Circuit Cards with Contacts: Organization, security and commands for interchange) now extended to contactless smart cards. In

This work within F1RST project is partly supported by IFCEPAR | CEFIPRA under grant No.7115 and Gemalto (supporting MBDS apprentices, and a CIFRE PhD research Scholarship).

addition, there are related standards (GP / OMAPI, EMV, EN 726-3 for prepaid memory cards, etc.).

Remark: APDU protocol does not manage the device connection or the channel opening.

The APDU commands (C-APDU) is the message sent by the client application (initiator) to the service running in the SE. The structure is composed of a mandatory header of 4 fields of one byte: (i) {CLA} (class field) defines the command type (standard, industry, using security or not), (ii) {INS} defines the command instruction, (iii) {P1} is the first parameter (0x00 if not defined), (iv) {P2} is the second parameter (0x00 if not defined), and a conditional body of 3 optional parameters of a variable length: (i) {Lc} is the conditional data length of 1 or 3 (extended APDU*) bytes if not empty,(ii) {Data} is the payload data of {Lc} length if not empty, (iii) {Le} is the expected length of the response data varying from 1 to 3 (extended APDU*) bytes if not empty.

The APDU response (R-APDU) is made of the optional response data (that cannot exceed the length defined in {Le} field provided in the C-APDU), and the 2 bytes response status words {SW1} and {SW2} giving the status of the C-APDU. When the command is successful, the service returns the status words 0x9000.

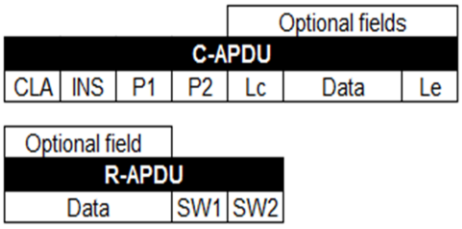

Fig. 1. APDU messages structure

TABLE I. EXAMPLE OF APDU ERROR STATUS WORDS

SW1, SW2	Meaning
0x6A82	File not found
0x6700	Incorrect data length
0x6981	Incorrect file type
0x6982	Security status not satisfied
0x6984	Invalid data
0x6985	Conditions not satisfied
0x6A86	Incorrect P1 and/or P2 parameter(s)
0x6D00	Unsupported command instruction

TABLE II. EXAMPLE OF CLA CODES

CLA byte	Command type
0x00	ISO standard command
0x04	ISO standard command with security
0xB0 to 0xCF	ISO standard INS instruction
0x80	GP standard command
0x84	GP standard command with security
0xFF	Commands for the reader

TABLE III. EXAMPLE OF INS CODES

INS byte	Instruction description
0xA4	ISO / IEC 7816-9 SELECT FILE used to initiate communication with a service identified by its AID (provided in the payload data field)
0x05	OMAPI SELECT SECURE STORAGE ENTRY
0xB0	ISO / IEC 7816-4 READ BINARY
0xD0	ISO / IEC 7816-4 WRITE BINARY
0xD6	ISO / IEC 7816-4 UPDATE BINARY
0xE0	ISO / IEC 7816-4 ERASE BINARY
0x82	ISO / IEC 7816-4 MUTUAL AUTHENTICATION

All standards combined, hundreds INS codes can be found (65536 possible combinations).

Card Query Language (CQL) initially designed by Pierre Paradinas in the 1990s [11], [12], with GEMPLUS (now Gemalto), was the first approach of SQL-like APDU instructions. Smart Card Query Language (SCQL) has been standardized in 1999 by ISO / IEC 7816-7 "Interindustry commands for Structured Card Query Language (SCQL)" [13]. But SCQL is limited to specific smart cards with embedded lite Relational DataBase Management System (RDBMS) whereas SE-QL is not….

TABLE IV. EXAMPLE OF SCQL INS CODES

INS byte	SCQL instructions in the payload
0x10	CREATE, DROP, INSERT, DELETE, DECLARE, FETCH
0x12	CREATE KEY, AUTHENTICATE, CHECK, BEGING TRANSACTION, COMMIT and ROLLBACK
0x14	CREATE USER, CHANGE PASSWORD, UNLOCK, DELETE USER, etc.

C. NFC mobile services basics (card emulation mode)

There are two situations involving the communication with the NFC mobile service: (i) the client is an NFC terminal (e.g. POS), or an NFC handset (acting as a terminal); communication is done via NFC when the handset is approached to the terminal and the terminal has successfully initiated the communication, (ii) the client is a mobile application (typically a user interface); communication is done via a bridge depending on the target SE and platform, through the Radio Interface Layer (RIL), or sometimes through NFC Controller using Single Wire Protocol (SWP)…

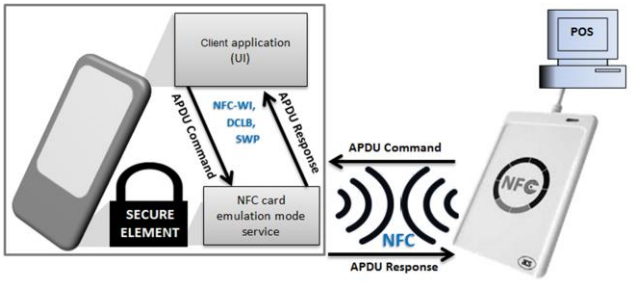

Fig. 2. NFC mobile application architecture (card emulation mode)

In the architecture shown on Fig. 2, the NFC service running inside the SE processes the received C-APDU and

returns R-APDU. However, mobile handset may also act as a reader and be client for an external NFC service: in that case, the mobile application will send C-APDUs to the external NFC device and will receive the R-APDUs.

Remark: A SE can host multiple services and the mobile application (user interface) can be client of a portfolio of NFC services; such an application is called a mobile wallet (m-wallet). A service hosted in the SE is identified by its AID (specified by ISO / IEC 7816-5 "Numbering system and registration procedure for application identifiers"); the AID is used to route the messages.

D. The Secure Element (SE)

SE is mostly an electronic chip with its own processor capable of running applications such as JavaCard (applets) and guaranteeing a certain level of security and functionality. Hardware-based SEs have the same characteristics as the smart cards: a minimalist computing environment on a single chip, complete with CPU, ROM, EEPROM, RAM and I/O ports, preprogrammed with a multi-execution environment OS and security domains separated by firewall that guarantees mutual isolation between running applications. Recent smart cards include coprocessors implementing cryptographic algorithms (such as DES, AES and RSA) and conform to TEE specifications.

SEs for mobile phones have all the capabilities of a smart card or even higher; they can theoretically be used for all types of applications using a smart card (prepaid cards, transportation cards, credit / debit cards, health, loyalty, couponing, storage of VPN access parameters, etc.).

1) Hardware-based SE: SIM-SE, eSE or Removable SE

a) The SIM-based SE handled by the Mobile Network Operator (MNO); Gemalto is providing such SE in our F1RST Consortium. The SIM card with its NFC interface is a Universal Integrated Circuit Card (UICC).

b) Embedded SE (eSE) outside the SIM into the terminal and handled by the device retailer (like Google, Nokia, Samsung).

c) Removable SE in an external SD card or a sticker under control of a SP (e.g. banks, retail companies) with or without the NFC chip.

The SE could encompass various applets with their own business models and access keys controlled by their owners. Theoretically the 3 hardware-based types of SE could work together depending on the Host Controller Interface (HCI) routing capabilities

2) Software-based (card emulation) and SE in the Cloud

The software emulation (of a smart card) is an approach to card emulation for NFC phones for services that do not need to be always available (i.e. when the mobile is off). It was introduced to mobile phones by Research In Motion (RIM) on the Blackberry platform. In addition to supporting different types of SE, the Blackberry 7 introduced the card emulation mode of NFC tags with a software application on the mobile phone. Host-based Card Emulation (HCE) is the software card emulation solution available on Android devices since the end of 2013 with Android KitKat. HCE services run in the host processor as well as other services making this solution less secure and most vulnerable to malwares [14].

On the other hand, NFC software-based implementation is much lighter to develop and truly simplifies the deployment; SP can deploy its NFC services itself. Furthermore, even if software card emulation approach cannot be a solution "as is", security could be strengthened by encryption mechanisms; such a "crypted soft-SE" is currently studied by IISc Bangalore in F1RST Consortium.

Another solution is to relocate the SE in the cloud (Cloud-SE) [15]. In this case, NFC services running on the mobile device relay the instructions to the remote Cloud-SE server (e.g. via Web services) and get back the returned responses. This approach reduces the complexity of the chain of deployment of new NFC services. It also has the major advantage to be hardware-independent and offers better safety (neither credentials nor sensible data are stored on the smartphone) and higher (unlimited?) storage capacity than other SEs, but increases the transactions latency. This approach which assumes always-on air reachability was discarded from our F1RST project, but it will also be studied for WOLF API.

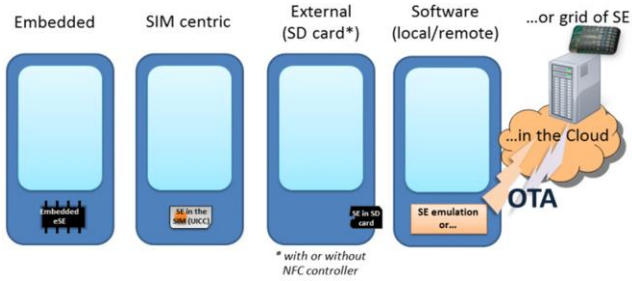

Fig. 3. Several SE architectures

E. Trusted service manager (TSM) and OTA interactions

NFC ecosystem incorporates several actors: computer information systems developers, MNOs, SPs, handsets retailers and chip cards manufacturers, etc. This implies an interoperability of the bodies that govern many heterogeneous domains (telecom, banking, technology, security and cryptography, rights to privacy, government, and other consecutive intermediates). This drives SPs outside their business domain. The TSM standards have emerged as the "split TSM" (SP TSM and MNO TSM, see Fig. 4) proposed by GP in 2011 which is basically provided as Web services.

NFC services hosted in the SE can be initially installed and built-in by MNOs or could dynamically be loaded on demand (by touching an NFC tag or by scanning a 2D tag or through Internet, etc.) onto the cell phone. This is done "Over-The-Air" (OTA), under the responsibility or not of the MNO managing the SE and the applets. Remote dynamic download and provisioning of applications, content, services, tickets, coupons… are then possible in a secure way. Every applet could be managed remotely by the TSM.

When the SE is in the SIM, MNOs have the responsibility of creating security domains on the SIM card for the NFC SPs. TSMs have the responsibility of credentials and encryption keys management of the SP security domain, application

loading into SP security domain, mobile wallet management, and service lifecycle management services (activation, personalization, maintenance, deletion). TSMs implement OTA communication into the NFC-enabled mobile phone and into the SE.

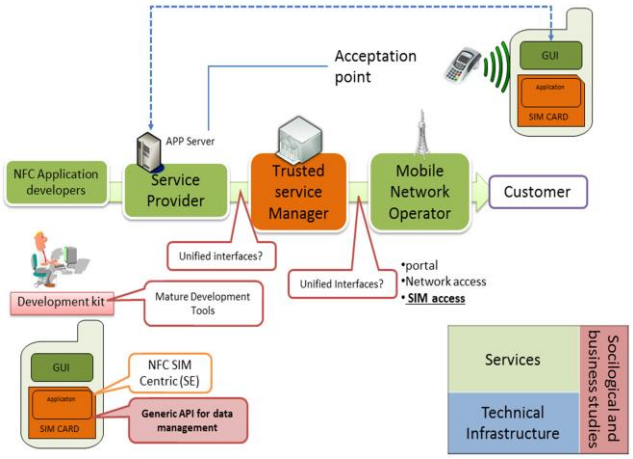

Fig. 4. NFC: a complex ecosystem [1]

F. NFC mobile payment and mobiquitous money

The payment industry seems to be naturally part of the NFC ecosystem by dematerializing payment cards in the SE to allow the phone to be used as a mean of payment.

The payment operators add contactless payment capabilities to their payment card (e.g. American Express ExpressPay, MasterCard PayPass, and Visa PayWave), to create an open loop approach to virtual local emerging money. These services enable end-users to tap and pay their purchase approaching their contactless card or NFC phone to an NFC payment terminal (POS).

In emerging countries, mobiquitous money could replace the cash playing a double attractive role for unbanked people: NO CASH OUT solutions (by accessing a digital wallet) and targeted help / donation through tagged products delivered to the beneficiary and biometric identification like in the prototype of M-PDS use case of F1RST project [5], [6], [7], [8], [9].

G. Related work: "NFC Container" project at MBDS with Gemalto

The project inherits of the know-how of MBDS innovation lab at University of Nice - Sophia Antipolis: in 2011, the university deployed an innovative project based on "NFC container" architecture called "Nice Future Campus" where the student's phone plays the role of a virtual card with a portfolio of services running in the SIM-SE provided by Orange (m-payment for the university cafeteria, access to geotagged campus information, consulting jobs and internships for students, access to the library, etc.).

NFC container is a SE content management API proposed by MBDS in 2008 as a result of a research contract with the French DGE ("Direction Générale des Entreprises") of the Ministry of Industry. This project has been driven in order to reduce the development time and the "time-to-market" of new NFC services. It is partly described in [2], and [3].

This project aimed to provide a set of tools in order to:

- Develop secure applications using NFC c.
- Deploy these applications to a large panel of NFC enabled mobile phone customers.
- Interact with those applications within the SE.

NFC Container project led to a high level API for SPs in order to manage their NFC services more easily.

The proposed API was a generic JavaCard application that allows a dynamic content management onto the SE. The implementation was compliant with NFC Forum standard specifications and runnable on every GP compliant SE.

The project was based on a generic applet that can be preloaded into the SE or loaded "on demand" by the end-user (through TSM / OTA).

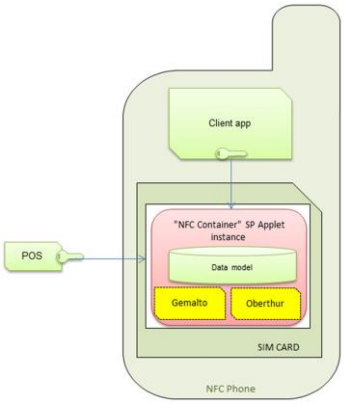

Fig. 5. NFC container instance [2]

The main idea was to access the "SE as a DB" where any privacy data from a client application can be managed. Each user (e.g. SP developer) is identified by a security key on the applet. NFC Container API provides developers some ways to implement additional layers for more complex data management. In that case, the SP develops its own application using the generic API which provides a set of functions (SQL-like) for data management on the card.

We illustrate the added value in terms of code simplification by the following Java coding example retrieved from [2]:

Source code 1. Example of NFC Container code simplification

```
//----------------------------------------//
// Example of Java coding without using    //
// NFC Container                           //
//----------------------------------------//
short index = GetIdxToDO(tag);
if (index == (short) DO_NOT_FOUND) {
  //DO not found-> create a new one//
  short index2free = GetIdxToFreeSpace();
  short freesize= (short) (SIZE MEMORY -
                           index2free);
  //calculate size of free space//
  short DOsize = (short) (lc+LEN_TAG+LEN_LEN);
```

```
  //calculate size of the new DO//
  if (DOsize <= freesize) {
    //it is enough space for a new DO//
    memory[index2free] = (byte) tag;
   //set DO tag//
   memory[(short) (index2free + LEN_TAG)] =
                        (byte) lc;
   //set DO length//
   //copy the DO atomic into the memory//
   Util.arrayCopy(cmd apdu,
     (short)((ISO7816.OFFSET CDATA) & 0x00FF),
       memory, (short) (index2free + LEN_TAG +
                    LEN_LEN), lc);
//----------------------------------------//
// Versus coding with NFC Container       //
//----------------------------------------//
void
insertRecord(byte[]record,
        shortrecordOffset,
        shortrecordLength)
```

NFC Container advantages:

Since access to the hardware-based SE is supervised by the NFC ecosystem owners, the deployment of new NFC services is necessarily related to them. To be able to test a new service to be installed in the SE, the developer has firstly to get the keys of the security domain. He has to deal with (physical) SE providers (MNOs, Gemalto, Oberthur…) and get some SIM cards. This is not simple as those access keys must remain secret in order to keep SE's security level.

Through the NFC Container, the developer has access to the generic pre-installed and fully customizable applet on the SE. Thus, the API allows developing new NFC applications without necessarily needing any high level security access and requires a smaller level of knowledge for NFC developers and services providers, allowing them to focus on their client application not in the applet management.

WOLF API is an extension of NFC Container project to the smartphone ecosystem with the formalization of SE-QL.

III. SE-QL and Wolf API: a Generic SQL-Like Interface to Communicate with the SE

The primary SE function is to store (sensitive and confidential) data: the majority of the (contact or contactless) card applications consist in storing and retrieving data (with or without pre-processing, with or without security mechanisms). The second fact is the building APDUs remains a cumbersome task for the developers: manipulating hexadecimal codes is not simple to human understanding; it is time consuming to be implemented and it is difficult to maintain.

In this section we illustrate the SE-QL foundation based on DataBase Management System (DBMS) concepts. We present our contribution to the project with the WOLF API at the development current stage, and we describe the implementation within use cases prototypes of the F1RST project.

A. Relational databases and SQL background

A database is a set of structured data associated with a schema derived from real world by applying a data model (relational, object). DBMS allow databases management through a standardized interface called SQL [1], [16], [17]. They provide TIPS (Transaction, Integrity, Persistence and Structuration / schema) services for development of information systems development [2].

A DBMS integrates 3 levels:

- Data Description Language (DDL): Defines a language allowing description of objects (tables, domains, databases, views, procedures…).
- Data Manipulation Language (DML): Defines a standard data manipulation language allowing interrogating and updating a DB without specifying any access algorithm (SQL3 being the current standard [1] for relational databases).
- Data Control Language (DCL): Defines some integrity and confidentiality constraints in order to manage user's rights and authorizations on objects.

The DBMS ensures consistency of data in databases and defines some mechanisms such as sharing, security, physical and logical independence of data, access performances in share or exclusive mode.

DBMS can handle several standard request mechanisms allowing manipulating data (read, write, delete, update, sort, etc.). The SQL language, created in 1974 and first normalized in 1986 is used to perform operations on relational databases. It allows developers to interact with a RDBMS without showing physical aspect of data and is compatible with DDL, DML and DCL. SQL's instructions syntax is pretty close to the human language in order to make it easier to learn and to read by a human user.

B. From SQL DBMS to SE's Applet

GP compliant SE is divided in a set of Security Domains (SDs) separated by firewalls allowing several services to be safely executed on the same card.

Each SD is dedicated to a type of business model. On Fig. 6, the first SD is reserved for the card issuer (MNO, manufacturer...); it contains maintenance and additional customer's (SP) services. Others are dedicated to the SPs services (transportation, payment card, loyalty card, coupons, etc.).

A DBMS manages a set of databases contains tables consisted of rows and columns / fields. On the other hand, the SE is composed of SDs hosting running applets managing their own data / fields (see Fig. 7).

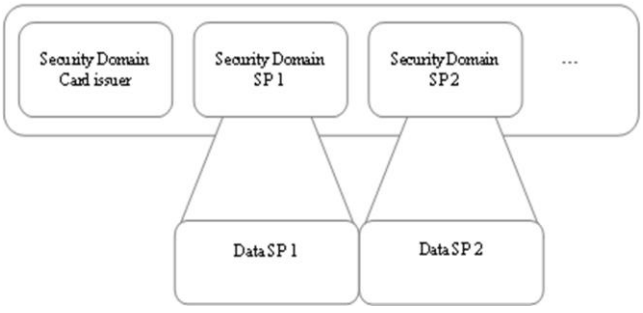

Fig. 6. Security Domains inside the SE

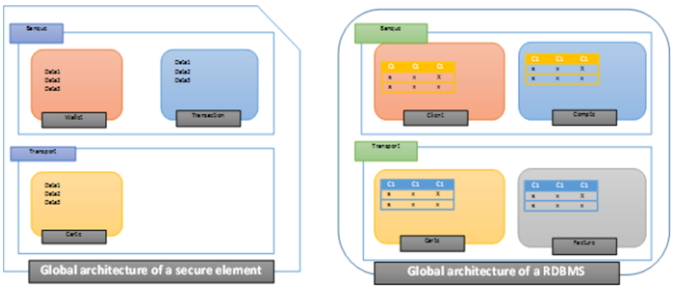

Fig. 7. Similarities between a SE and a DBMS

We can infer the following conclusions:

- The execution environment provided by the SE could be seen as equivalent to a DBMS: the OS manages the SDs and the DBMS manages databases;
- An NFC service hosted in the SE (applet) could be seen as a table of the DB;
- The data fields of an applet could be seen as the columns of the table.

C. DDL, DML, and DCL applied to the SE-QL

Note: at this stage, the instructions are not yet all been implemented in SE-QL.

1) Data Description Language (DDL)

These instructions are intended for the SE management with special maintenance privileges:

- *CREATE* – to create new services (applets) in the SE
- *ALTER* – to modify the structure of the SE data storage
- *DROP* – delete objects from the SE

2) Data Manipulation Language (DML)

These instructions are intended for the NFC service management with the SP privileges:

- *SELECT* – read data from the applets of the SE
- *INSERT* – write data fields into an applet of the SE
- *UPDATE* – update existing data fields within an applet of the SE
- *DELETE* – deletes the records from an applet of the SE

3) Data Control Language (DCL)

- *GRANT* – provide access privileges
- *REVOKE* – remove access privileges

D. Principle of SE-QL

First objective is to hide this low-level implementation of byte arrays by providing the same generic and developer-friendly interface regardless of the platform (initially developed on Java and Android devices, later on Windows Phone and other platforms like HTML5).

1) SE-QL algorithm

A SQL query is composed of two parts: a mandatory part containing the statement and the object affected by the command, and an optional part using algebraic operators for projection, selection, junction and division.

For a given applet uniquely identified by its alias (matching its AID), SE-QL model makes a correspondence between an SQL-like instruction (CREATE, INSERT, SELECT, UPDATE, DELETE, etc.) and the assigned APDU instruction, whereas the data model gives the records SE-QL alias, their corresponding hexadecimal identification (ID) in the APDU instruction parameter, and the maximum length expected by the applet.

For example, we can illustrate a simplified use case in which the developer would check and update the balance stored by a portfolio applet. This is done in two steps: (i) read "balance" record from "portfolio" applet, (ii) update "balance" record from "portfolio" applet. This will be translated as following in SE-QL language: (1) "SELECT balance FROM portfolio", (2) "UPDATE portfolio SET balance = {value}". To be done, the metadata must provide portfolio applet AID, SELECT and UPDATE translation into APDU protocol, and the "balance" record byte identifier:

TABLE V. EXAMPLE OF SE-QL INSTRUCTIONS

Description	SE-QL meaning/ alias	APDU transcription
Applet / relation {used for the communication initialization}	portfolio	AID: F0000000001
	Class standard & Security compliance	CLA: B0 {ISO / IEC 7816-4}
Instruction	SELECT	INS: B2 {READ RECORD}
	UPDATE	INS: DC {UPDATE RECORD}
Record projection	Balance	P1: 50
	Record value expected length	Le: 0E {14}

TABLE VI. EXAMPLE OF SE-QL TO APDU COMMAND

SE-QL command	APDU command	Return
SELECT *balance* FROM *portfolio*	B0B250000E	Fields values
UPDATE *portfolio* SET *balance* = 2000 {payload data is given according the field length 0x0E defined by the metadata}	B0DC50000E00000000002000	Number of rows updates

In more complex use cases, several APDU commands may be handled within a single SE-QL instruction, for example, a statement regarding multiple fields. The activity diagram shown on Fig. 8 illustrates the processing of several fields algorithm, where APDU command sending corresponds to the greyed task.

These examples illustrate the similarity between an APDU command and a SQL query; the rationale of our approach is to identify mechanisms between these two worlds. Moreover applets are not intended to manage large amounts of data like tables in relational databases; usually there are small pieces of data corresponding to tuples of information in a table in a DBMS. In addition, SE-QL language will be limited by the

already defined standards; SE-QL goal is only to ease NFC secure application development.

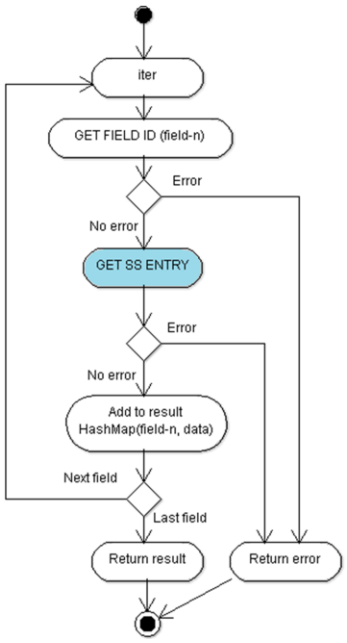

Fig. 8. SE-QL SELECT instruction algorithm

SE-QL is delivered as a Java package integrated in WOLF libraries that manages SE / reader connection and applet communication initialization. This is managed from SE-QL controller that is the entry point for the client applications of the SE. SE-QL controller is in charge to redirect instructions to the appropriate SE-QL interface according targeted platform, implementing the method "execute" that accepts the SE-QL instruction and the command ID. The result for a given command (identified by an ID) is forwarded through an event of the SE-QL callback currently in the form of a HashMap of returned data, concerned field name being the key.

2) Metadata

Metadata is actually provided as a XML (eXtensible Markup Language) file (which could be acquired on the fly during the installation of the application, for example from a web server); it contains one (or more) applet(s) configuration defining two models: the former describes the correspondence APDU / SE-QL commands, while the second describes the pattern of the data managed by the applet. The metadata are used by SE-QL parser on the side of the client application at the APDU commands building, as well as the generic applet for its data storage initialization (could also be provided as a script).

Source code 2. Example of XML metadata file

```
<seqlmetadata>
   <applets>
      <appletModel alias="pds_applet" AID="F0014144500002">
         <instructions>
             <seqlModel>
               <ins>SELECT</ins>
               <cla>80</cla>
               <value>B2</value>
            </seqlModel>
            <seqlModel>
               <ins>INSERT</ins>
               <cla>80</cla>
               <value>D2</value>
            </seqlModel>
            </instructions>
         <tables>
            <tableModel>
                  <name>username</name>
                  <value>30</value>
                  <length>9</length>
               </tableModel>
               <tableModel>
                <name>password</name>
                <value>40</value>
                <length>9</length>
               </tableModel>
               <tableModel>
                <name>key</name>
                <value>31</value>
                <length>255</length>
               </tableModel>
               <tableModel>
                <name>pin</name>
                <value>50</value>
                <length>4</length>
               </tableModel>
            </tables>
      </appletModel>
      <appletModel alias="wolf_hce" AID="F0014144500001">
         <instructions>
                  ...
```

E. WOLF API

As mentioned, WOLF aims to be an ontology-driven Framework based on self-descriptive metadata (being currently XML format). This is our prerequisite to ensure maintainability and portability of the components. The architecture of WOLF is drawn around the central concept of SE-QL being the interface for a simplified and optimized handling of the data that is the same regardless of the mobile and whatever SE type and whatever covered platform. This interface allows developers to define their own configuration in the metadata file (mapping the APDU instructions, description and alias of the data, etc.), making the interface compliant with already existing (contact or contactless) smart card standards or proprietary instructions.

WOLF API encompasses a generic applet and a generic wallet each based on its own metadata that can be easily personalized for a rapid implementation (see Fig. 9).

1) WOLF current progress

Current API provides:

- SE-QL package which is common for all Java-based platforms contains the SE-QL generic interface, APDUs handling classes, SE-QL callback interface, SE-QL parser, and metadata objects.
- Android packages contains: (i) OMAPI SE-QL controller to access the SE with OMAPI API on Android SDK17 platform, (ii) HCE controller for Android SDK19 platform SE-QL, and (iii) the generic WOLF HCE service (that emulates the SE) able to communicate with (iv) WOLF generic applets. Android package will also contain the off-host APDU service** (to communicate with the SIM-SE), the generic Wallet UI, and Android reader for external devices (work in progress).

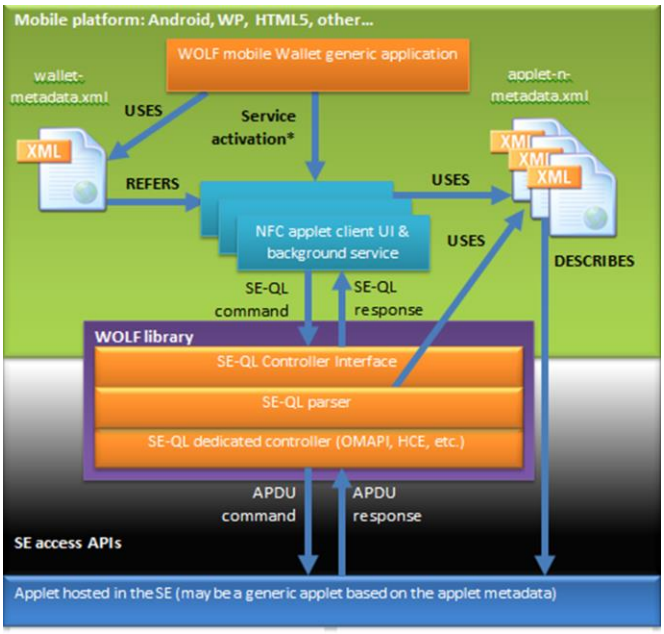

Fig. 9. WOLF architecture overview

- Smart card reader package provides the SE-QL controller for Java™ Smart Card I/O API compliant with PC / SC (NFC) readers.
- (4) HTTP package contains the HTTP requests helper for the sending of GET, POST, PUT, and DELETE requests (e.g. to access RESTful Web services), and the SOAP Web services helper.
- Cryptography helper (encode / decode, generate hash key) is provided in the tools package.
- TSM package contains tools dedicated to TSM (work in progress).

WOLF for Android and SE-QL have been tested for three use cases so far : (i) SIM-based NFC service on Android device compiled with SDK17 (Android Jelly Bean, build 4.2.2 or earlier on nonstandard build of CyanogenMod) using OMAPI, (ii) Android HCE generic service compiled with SDK19 (Android KitKat, build 4.4.2), and (iii) a Java application to test the communication between a USB NFC reader plugged to the PC (using standard PC / SC driver, see www.pcscworkgroup.com/) and an NFC device.

The beta version of WOLF plugin for Android was implemented last year for CyanogenMod with seek-for-Android [47] implementation of OMAPI (need the device bootloader to be unlocked and CyanogenMod build to be installed in replacement of standard OS on the phone). Afterward, OMAPI was formally integrated as an external API for Android standard build 4.2.2. But since Android KitKat, OMAPI is no more supported by the official Android build; it has been replaced by Android HCE, the software based card emulation mode. ***On Android KitKat, SIM-SE access must be routed by Android off-host APDU service…*

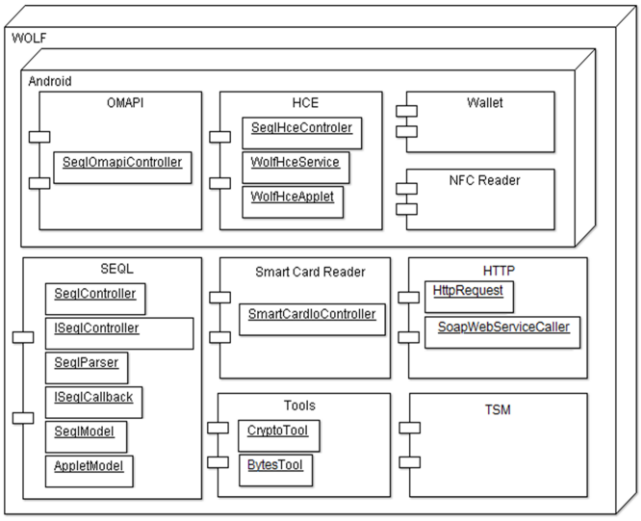

Fig. 10. Major WOLF current components

2) WOLF experimentation within F1RST project

The F1RST wallet for Financial Inclusion (FI) uses SE-QL commands and WOLF platform in the end-to-end supply chain NFC services for the secure traceability of aids delivery. WOLF was successfully tested in the development of two use cases prototyped at MBDS in 2012-2014 within F1RST project; M-PDS (Mobiquitous NFC Public Distribution System) [8] and BARTER 2.0 (Bank of Animal in Rural TERritories 2.0 / Social Network of Donators) [8]: WOLF API and SE-QL are the kernel of mobile NFC services modules within use cases prototypes of F1RST wallet as shown in Fig. 11.

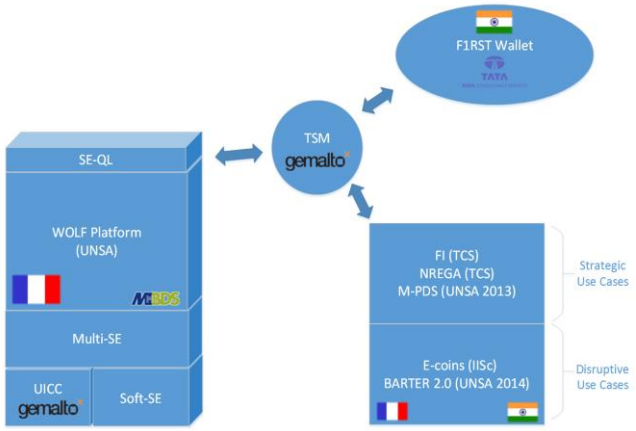

Fig. 11. Architecture of F1RST ecosystem [8]

3) WOLF / SE-QL implementation

On Android platform, WOLF library has to be referenced in the project properties. Once it is done, SE-QL controller class can be instantiated once with appropriate parameters (targeted platform, application context if Android, metadata file). Then, you can get the controller and use SE-QL to communicate with the SE, and receive events with the returned responses from the applet as shown in source code 3:

Source code 3. Android implementation of WOLF: SE-QL controller instantiation

```
1.      import org.mbds.wolf.seql.ISeqlCallBack;
2.      import org.mbds.wolf.seql.SeqlController;
3.      import org.mbds.wolf.seql.exceptions.ApduError;
4.
5.      public class MyApplication extends Application
6.                          implements ISeqlCallBack {
7.            SeqlController ctrl = null;
8.            //........
9.            protected void init() {
10.           boolean ok = false;
11.           try {
12.              SeqlController.OS os =
13.               SeqlController.OS.ANDROID_HCE;
14.              if (android.os.Build.VERSION.SDK_INT<=19)
15.                  os = SeqlController.OS.ANDROID_OMAPI;
16.              ctrl = new SeqlController(
17.                getApplicationContext(), os,
18.                 File metadata, this);
19.              ok = true;
20.           } catch (ClassNotFoundException e) {
21.              e.printStackTrace();
22.           } catch (NoSuchMethodException e) {
23.              e.printStackTrace();
24.           } catch (InstantiationException e) {
25.              e.printStackTrace();
26.           } catch (IllegalAccessException e) {
27.              e.printStackTrace();
28.           } catch (InvocationTargetException e) {
29.              e.printStackTrace();
30.           }
31.           if (!ok) {
32.              Toast.makeText(this, "SE-QL controller
33.                could not be instantiated,
34;               application will finish!",
35.                Toast.LENGTH_LONG).show();
36.              quitApp();
37.           }
38.        }
39.     }
```

Source code 4. Android implementation of WOLF: executing SE-QL instruction

```
1.  public class MyActivity extends Activity
2.                          implements ISeqlCallBack {
3.      private SeqlController ctrl;
4.      //…
5.      @Override
6.      protected void onCreate(Bundle
7.                              savedInstanceState) {
8.              super.onCreate(savedInstanceState);
9.              MyApplication act =
10.              (MyApplication)getApplication();
11.             ctrl = act.getController();
12.     }
13.     @Override
14.     protected void onResume() {
15.             super.onResume();
16.             ctrl.setCallback(this);
17.             if(!ctrl.isServiceConnected()) {
18.                 ctrl.initService();
19.             }
20.     }
21.     private boolean executeSeqlCommand(String
22.                         statement, int commandId) {
23.             return          ctrl.execute(statement,
    commandId);
24.     }
25.     @Override
26.     public void onPINRequired() {
27.             startActivityForResult(new
    Intent(this,
28.                         PinEntryView.class),
29.                     Constant.PIN_REQUEST);
30.     }
31.     @Override
32.     public void onResponse(Map<String, Object>
33.                         results, int commandId) {
34.             //Process results
35.     }
36.     //...
37. }
```

Fig. 12 gives an overview of components interaction showing how the client module (service UI) requires the name and the password of the handset end-user, stored by the applet using SE-QL; then, WOLF plugin initiates the connection and the applet requests the user PIN entry. When user PIN entry has been successfully transmitted, the previous SE-QL instruction can be processed. This sequence derived from F1RST use cases prototyped at MBDS was a first proof-of-concept; the client module has been tested with the applet embedded in the SIM and the generic HCE applet of WOLF without requiring any code changes except for the manifest (and the Android host-apdu-service XML metadata), since HCE services must be declared in the client application.

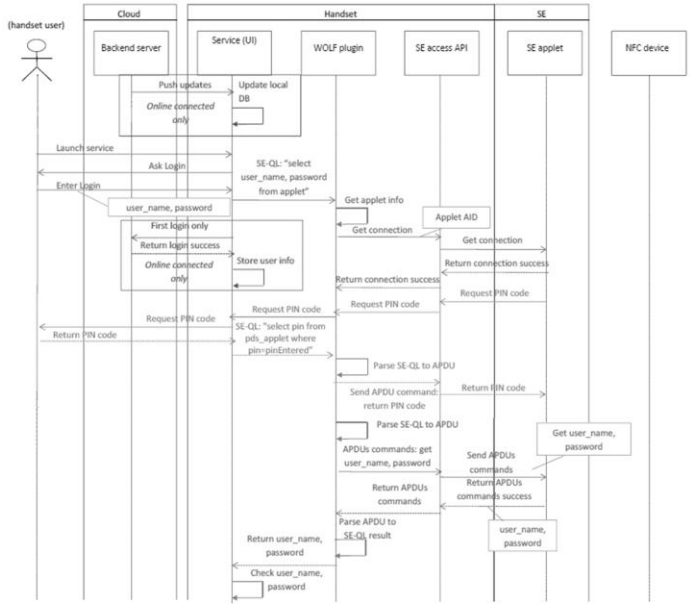

Fig. 12. Sequence diagram example using WOLF API

The WOLF SE-QL reader Java tester is intended to test the communication with the applet hosted into the SE of the mobile phone: the tester frame is shown in Fig. 13: user grabs the SE-QL instruction in the input text area and clicks on the "Execute" button. Then, he is asked to place the device on the reader and the SE-QL instruction is parsed into APDU(s) command(s); the result is shown in the log trace view.

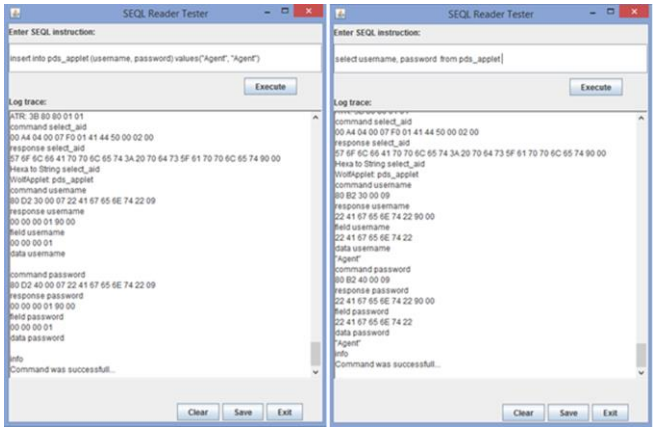

Fig. 13. SEQL tester for PC / SC Reader

IV. Conclusion and Future Works

In this report, we demonstrated SE-QL high added value making easy and fast the development of secure NFC mobile services using card emulation mode. The great advantage of SE-QL is that it is compatible with most APDU standards and can be used for already existing applets. SE-QL is a very good solution for applications that do not need complex transactions.

Withal, it does not already fully meet ACID (Atomicity, Consistency, Isolation, and Durability) properties that guarantee the transactions are processed reliably; in the next release, we are focusing on the transactions integrity by implementing the mechanisms for the handling of a set of transactions that need to be completely executed in order to maintain system consistency. This will be done using the "BEGIN transaction", "COMMIT", and "ROLLBACK" instructions (taking care of the limitation of RAM will require a temporary serialization before complete execution) and the management of a time-out. But, as we talk about contactless communication, we may face untimely interruptions transactions when the devices are removed prematurely.

Several researches have been done on database implementation in smart cards (that can apply to the SE) [11], [12], [13], [18]; this highlights the need for embedded DB system. Nowadays, lite databases have been adopted for low capacity systems like SQLite (www.sqlite.org) which is multiplatform compliant and it is provided as part of Android OS. This is why we will also explore other areas of investigation around embedded DB, and also other object-oriented structures in the payload such as RESTful (Representational Sate Transfert) services, and the ability to provide XML, JSON (JavaScript Object Notation), etc.

Another current important stuff is to manage security according standards specification. This will be done combining IISc Bangalore research on the crypted SE to ours, e.g. using cryptographic APIs (e.g. Cipher) and studying existing protocols, or by proposing new ones and Cloud-based solutions (studied at LSIS lab of Aix-Marseille University [11]).

Acknowledgment

Special Thanks to Gemalto (supporting MBDS apprentices and a CIFRE research Scholarship) and IFCEPAR | CEFIPRA organization supporting F1RST (Financial Inclusion based upon Rural mobiquitous Services Technological platform) project (2012-2015) and more personally to Mr. Ilan Mahalal, Program Manager at Gemalto, Mr. Debi Pati, CEO lead at TATA CS, and Nicolas Pastorelly, Olfa Arfani, Mohamed Sidime, Guillaume Larroque, Pierrick Morizot from University of Nice Sophia Antipolis and MBDS for their technical contributions to NFC Container, WOLF platform and F1RST project.

References

[1] S. Miranda, "Relational Objects Databases (Bases de données objets relationnelles (SQL3 et ODMG))," Dunod, 2004.

[2] S. Miranda et al., "Mobiquitous Information Systems (Systèmes d'Information mobiquitaires)," in Ingénierie des systèmes d'information, RTSI Série ISI, Vol. 16 no 4, Hermes Lavoisier, France, 2011.

[3] S. Miranda, N. Pastorelly, V. Ishkina, D. Torre, V. Chaix, "Lessons inferred from NFC mobiquitous innovative information service protyping at UNS", in [1], 2011, pp. 15-47.

[4] M. Della Peruta, A. Atour, "Business models for mobiquitous (social) money: Application to M-PDS program in India" research report, to be published in International Journal of Complementary Currency Systems (IJCCS), 2014.

[5] D. Pati, S. Miranda, confidential document, "FIRST Project: Collaboration Agreement," Feb, 2012.

[6] D. Pati, "Architecture of F1RST system," IFCEPAR | CEFIPRA Report, Nov., 2013

[7] O. Arfani, M. Sidime "M-PDS (mobiquitous Public Distribution system) USE CASE for F1RST project," M.S. thesis, MBDS CS department, University of Nice – Sophia-Antipolis, France, Oct., 2013.

[8] G. Larroque, P. Morizot, B. Renaut, S. Miranda: "Proposal of a disruptive Use Case for F1RST : Mobiquitous Bank of Animals – BARTER 2.0 project", draft Nov. 2013.

[9] A.-M. Lesas, "F1RST Research project Mobiquitous NFC Financial services for unbanked people", presentation at WIMA Conf. in Monaco, NFC Research Track, April 22, 2014.

[10] C. Papetti, K. Sok, S. Miranda, "Mobiquitous NFC Tourism (Une plateforme de gestion de Tags pour le tourisme mobiquitaire du Futur)," Monde du Tourisme, to be published, 2014.

[11] P. Paradinas, J.-J. Vandewalle, "A personal and portable database server: the CQL card," 1994, available: http://cedric.cnam.fr/~paradinas/presentation/CQL.pdf

[12] P. Paradinas, J.-J. Vandewalle, "How to integrate Smart Cards in Standard Software without writing specific code?," 1994, available: http://cedric.cnam.fr/~paradinas/presentation/CTST.pdf

[13] 3GPP TSG-T3, "Phone book management with ISO 7816 part 7 (SCQL)," Document T3-99167, Source: Gemplus, Miami, June, 14-16th, 1999.

[14] M. Roland, "Software Card Emulation in NFC-enabled Mobile Phones: Great Advantage or Security Nightmare?," NFC Research Lab Hagenberg, Univ. of Applied Sciences, Austria, IWSSI / SPMU, June, 2012, available: http://www.medien.ifi.lmu.de/iwssi2012/papers/iwssi-spmu2012-roland.pdf

[15] L. Pesonen, "TSM Point of View and Issues Faced," EC/ETSI Workshop on Collaborative Ecosystem for M-Payment, Sophia Antipolis, France, Jully, 1, 2014, available: http://docbox.etsi.org/Workshop/2014/201407_MPAYMENTWORKSHOP/S02_ECOSYSTEM_and_ISSUES/S02_Pesonen_GD.pdf

[16] C.-J. Date, "Introduction to data base systems," Addison-Wesley Educational Publishers Inc., U.S, 1975.

[17] E.-F. Codd, "A Relational Model of Data for Large Shared Data Banks," IBM Researrch Report, San Jose, Aug. 19, 1968.

[18] N. Anciaux, L. Bouganim, P. Pucheral, "Embbeded RDBMS within a smart card feedback (SGBD embarqué dans une puce : retour d'expérience)," Techniques et Sciences Informatiques, vol. 27, no 1-2, Sept., 2008, pp. 141-180.

[19] S. Miranda, A.-M. Lesas, " VAMP project: Mobiquitous NFC car (Architecture logicielle du projet Vamp. Plateforme mobiquitaire embarquée à bord de véhicules mobiles)," Revue du Génie Logiciel, No103, Dec., 2012, pp. 38- 48.

[20] Li, Y., Boucelma, O., Provenance Monitoring in the Cloud, IEEE 6th International Conf. on Cloud Computing, June 27-July 2, 2013, Santa Clara, USA.

[21] V.Coskun, K. Ok, B. Ozdenizci, "NFC application development for Androïd," Wrox Ed, John Wiley&Sons, UK, 2013.

[22] E. Coleen Coolifge, P.Hourani, "Securing cloud and mobility," CRC Press, Auebach Publication, USA, 2013.

[23] L. Francis, G.-P. Hancke, K.-E. Mayes, K. Markantonakis, "Practical Relay Attack on Contactless Transactions by Using NFC Mobile Phones," Cryptology ePrint Archive, Report 2011/618, 2011, available: http://eprint.iacr.org/2011/618.

[24] V. Alimi, "An Ontology-based Framework to Model a GlobalPlatform Secure Element," presentation at WIMA Conf. in Monaco, NFC Research Track, April 11, 2012.

[25] M. Roland, "Secure Element APIs and Practical Attacks on Secure Element-enabled Mobile Devices," presentation at WIMA Conf. in Monaco, NFC Research Track, April 11, 2012.

[26] L. Francis, G.-P. Hancke, K.-E. Mayes, K. Markantonakis, "Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones," RFID Security and Privacy Issues, LNCS vol. 6370/2010, Heidelberg, 2010, pp. 35-49.

[27] G.-P. Hancke, K.-E. Mayes, K. Markantonakis, "Confidence in smart token proximity: Relay attacks revisited," Computers & Security, Elsevier Ltd., Springer Berlin, 2009, pp. 615-627.

[28] H. Ailisto, T. Matinmikko, J. Häikiö, A. Ylisaukko-oja, E. Strömmer, M. Hillukkala, A. Wallin, E. Siira, A. Pöyry, V. Törmänen, T. Huomo, T. Tuikka, S. Leskinen, J. Salonen, "Physical browsing with NFC technology," VTT Research Notes 2400, 2007.

[29] T. Tuikka, M. Isomursu, "Touch the Future with a Smart Touch," VTT Research Notes 2492, 2009.

[30] V. Coskun, K. Ok, B. Ozdenizci, "Professional NFC Application Development for Android™," Wrox, John Wiley & Sons, Ltd., 2013.

[31] D. Schall, "Service-Oriented Crowdsourcing: Architecture, Protocols and Algorithms," SpringerBriefs in Computer Science, 2012.

[32] K. Finkenzeller, "RFID handbook," third edition, John Wiley & Sons, Ltd., 2010.

[33] T. Igoe, D. Coleman, B. Jepson, "Beginning NFC," O'Reilly Media, Jan., 2014.

[34] Urien, P., Piramuthu, S., "Towards a secure Cloud of Secure Elements concepts and experiments with NFC mobiles," IEEE International Conf., CTS, May 20-24, 2013, San Diego, USA.

[35] P. Pourghomi, G. Ghinea, "Managing NFC payment applications through cloud computing," IEEE, ICITST, Dec. 10-12, 2012, London, UK.