Vision towards an Open Electronic Wallet on NFC Smartphones

Glenn Ergeerts*, Dries Schellekens†, Frederik Schrooyen*, Rud Beyers*, Kevin De Kock*, and Thierry Van Herck*
*Department of Applied Engineering
Artesis University College Antwerp
Antwerp, Belgium
Email: glenn.ergeerts@artesis.be
†Department of Electrical Engineering, ESAT/SCD-COSIC and iMinds
KU Leuven
Leuven, Belgium
Email: dries.schellekens@esat.kuleuven.be

***Abstract*—Many recent initiatives indicate an evolution towards an electronic wallet to perform all sorts of electronic transactions, such as for example micro payments, loyalty, and transport ticketing. Furthermore, the smartphone is emerging as an indispensable tool for many, containing more and more personal information. Hence, it seems like the ideal medium for carrying the electronic wallet as well. The fast and intuitive touch-and-go philosophy and the integration in mobile devices, makes Near Field Communication (NFC) the perfect technology for contactless electronic transactions. However, the complex ecosystem is holding back the world-wide integration of this technology in mobile handsets, resulting in a low market penetration of NFC smartphones. In our work, we investigated how an NFC-enabled smartphone can be used as an electronic wallet in an existing solution that originally relied on DESFire tags. We implemented a Java Card applet that is compatible with the DESFire specification and explored how this applet can be deployed on the different possible solutions for the Secure Element (SE), such as an active Bluetooth sticker, a SIM card, and a secure microSD card. Finally, the user-wallet interface was examined carefully, more specifically in terms of accessibility of the wallet content for the different smartphone platforms. Based on the overall experiences and results the final conclusions upon the vision towards an open electronic wallet of the future were drawn.**

***Keywords*-NFC; electronic wallet; secure element; SCWS; Java Card; DESFire; smartphone.**

I. Introduction

Today a lot of multinationals including Google, Apple, and Microsoft are involved in an electronic wallet project one way or another. Because of the difficult ecosystem, everything is kept rather closed, which is not very stimulating for the penetration and use of it. This paper offers a more open solution and gives an overview of what is possible and what is not.

Implementing an electronic wallet on smartphones offers a lot of advantages compared to a classic wallet. For example, it offers a solution for containing all todays electronic payment cards; so in case of a payment transaction a customer only needs to select the appropriate payment method. It saves a lot of time and it can be used to hold much more, for instance loyalty cards, drink/entrance vouchers and coupons. In the case of a festival or an event, visitors are required to obtain drink and food vouchers before they can make an order. This makes them lose precious time by standing in line to exchange their money for vouchers. With the aid of their smartphone, they are capable of carrying out over-the-air transactions and consultations. So essentially, we could throw away our wallet and replace it with our personal smartphone, which we carry already with us all the time now [1].

Mobile and contactless payment systems are finally getting the push in the back they deserve through the upcoming of NFC [2] in handsets, a contactless communication technology designed for electronic payments. Several big companies and research institutes stated that in 2011 up to 50 million NFC handsets were shipped and about 20% of the Points of Sale that were sold had NFC support [3]. These numbers are expected to emerge to 800-1000 million NFC-enabled handsets and up to 80% for Points of Sale with contactless support. Most mobile phones will be NFC-enabled through the use of an integrated NFC controller while others will use a more intermediate solution like a NFC-enabled microSD or active NFC sticker; this paper will discuss all these solutions in more detail. The total market size for mobile payments with NFC is targeted at 680 billion dollar in 2016 [3].

The electronic wallet, that is being developed and tested in the EVENT project funded by the Flemish government, covers a broad range of solutions for the user, from NFC tags to NFC-enabled smartphones. The implemented solution is meant to be open, generic and hybrid. The openness refers to the fact that the wallet will be able to hold several types of payment solutions: existing initiatives such as PingPing (Belgacom) and PayWave (VISA) can be plugged into the wallet, at least from a technical point of view. The design of the wallet will be generic, allowing for usage in a number of different settings. Examples include payments (large as well as small amounts), ticketing, coupons, etc. The hybrid aspect is referring to the fact that value can be stored on the card (offline) as well as on a server (online). Both approaches have their advantages, but this paper will mainly focus on the offline wallet. In the online use case, the wallet is only containing some sort of identifier for authentication and the user interface afterwards, is more or less comparable with the online banking solutions today. Unlike in the offline case, there is no direct link needed between the SE and the OS of the users phone. This will simplify the user interface issue for consulting the wallets content to a plain web interface, though it makes the payment infrastructure completely reliable on a network connection. Although this will be perfectly acceptable in fixed environment, most of the big event organizers, either professional or leisure, will not want to take the risk.

The security issue is also addressed briefly. The wallet has to be deployed on a SE. Since payment is a critical application, a secure solution must be provided and also the consumer needs to be convinced that the system is at least as safe as the traditional payment methods used today, meaning there is no room for tampering with the data that represents money or tickets that were bought. The possible choices of will be discussed more thoroughly later in this paper.

As mentioned earlier the wallet itself can reside both on a passive DESFire tag or an NFC-enabled device and it relies upon terminals equipped with NFC technology to initiate the actual transactions. The latter holds a number of intrinsic advantages over tags such as allowing users to view and interact with the contents of the wallet, whereas tags rely solely on terminals instead, which can be deemed to be a shortcoming. Since current terminals are intended to be compatible with only passive DESFire tags, one of the goals is that the mobile phone counterpart adopts the current protocols used between terminals and tags. A DESFire applet will be implemented and deployed on a Java Card runtime environment to ensure backwards compatibility with the currently used system.

Finally, there is a lot of differentiation between the various handsets currently available on the market. This, in combination with the difficult accessibility of the Secure element, means that the graphical representation of the content of the wallet is quite a challenge. This paper can be divided in two large sections: Section II will highlight the interface between the wallet and the terminal, whereas Section III focusses on the graphical user interface. Finally, in Section IV we will draw some conclusions.

II. Terminal interface

In this section, we will describe the interface between the terminal and the wallet. In Section II-A we will describe how to achieve backwards compatibility with existing tags. Next in Section II-B, we will discuss the development of a DESFire applet. Finally, we will give an overview of the different deployment options in Section II-C.

A. Terminal interface compatibility with passive tags

In earlier work [4], we implemented an electronic ticketing solution based on MIFARE Classic tags. Even though most secure elements of NFC-enabled smartphones emulate a MIFARE tag, we decided not use this technology in the EVENT project, primarily because the security of MIFARE Classic has been seriously compromised [5], [6], [7].

We selected the MIFARE DESFire EV1 product family from NXP Semiconductors instead, because it offers a flexible file system and strong cryptographic mutual authentication. DESFire tags are popular for public transportation, staff or student identification, building access control and canteen payments, and they are available with a non-volatile memory size of 2 to 8 KB. We believe that this technology is well suited for the realization of an open hybrid electronic wallet.

DESFire tags support up to 28 applications and every application can store up to 32 files. Consequently the same physical tag can be used for multiple applications, such as professional or leisure events, public transport, loyalty programs for supermarkets, etc. Files will typically be used to store an identifier (e.g., to identify a customer in a loyalty program or to link to an online banking account), an offline stored counter that represents an amount of vouchers, or a transaction log.

Another attractive feature of the DESFire standard is its fine-grained access control mechanism: up to 14 different keys can be associated with an application and access rights to the files within the application can be enforced based on these keys. Within the EVENT project we implemented drink/food vouchers as DESFire value files and use separate keys to authorize the credit and debit operation on these files. Terminals at the booths where vouchers can be purchased, will authenticate to the tag using the key that authorizes the credit operation, whereas terminals at a bar or a food stand, will use the key that grants the debit right to redeem a voucher. In the project we decided to make the credit key unique for every tag, by using the tags UID (unique identifier) in a key diversification function, but to keep the debit key fixed for the event and the same for all tags. This design choice was made because of a trade-off between transaction speed, scalability and security. On the one hand debit transactions are fast and scalable as terminals will always use the same key. On the other hand credit transactions are a bit slower as terminals must derive the diversified, tag specific key at the start of the transaction, but they offer a high level of security. As for the consult key we decided to leave the security open, which implicates that anybody holding an NFC mobile device can perform a consultation operation. In our opinion this might stimulate the demand for an NFC-enabled smartphone, since the early adopters will demonstrate the possibilities to their peers. If desired, or in case harm is caused, you can easily protect this operation as well of course.

As motivated in the introduction, it is crucial that our smartphone implementation of the electronic wallet provides an interface to the terminals that is fully compatible with the regular tags. The DESFire standard supports the wrapping of native commands in ISO7816 APDUs (Application Protocol Data Units), which makes it possible to implement the DESFire protocol on any contactless smart card. The terminal software that we developed in the EVENT project uses these wrapped ISO7816 APDUs, instead of native DESFire APDUs, to ensure that the communication interface to tags and smartphones is identical.

B. Development of DESFire applet

In 2010, Gemalto announced the world's first implementation of a transport application compliant with the DESFire specification in a SIM (Subscriber Identity Module) or UICC (Universal Integrated Circuit Card) card [8]. This suggests that Gemalto has already developed a Java Card applet to emulate a DESFire tag.

We decided to make our own implementation of the DESFire specification in Java Card, because we wanted to explore different options to visualize the content of the emulated DESFire tag to the user (see Section III). In some approaches for the user interface, we required access to the source code of the DESFire emulator, e.g., when a second applet is used to read the content of the electronic wallet using inter-applet object sharing.

Within the EVENT project Jorge Prado made an initial proof-of-concept implementation of the DESFire specification in Java Card[1] and his results were published in [9]. Afterwards Dries Schellekens independently implemented a DESFire applet, with a clearer structure, a more mature code base and a proper documentation. This second implementation was used in the remainder of the EVENT project and it will be described in this work.

The main design criterium during the design of the applet was compatibility with the DESFire specification. We also tried to write secure and reliable code, that is protected against card tear (by using the transaction functionality of the Java Card runtime enivronment), and we did some basic performance optimizations (e.g., keeping certain session data in RAM memory). Since the DESFire EV1 specification is pretty extensive and rather complex, the decision was made to only implement the subset of the specification that is useful within our project.

We implemented all the card level commands, which are used for basic configuration of the tag and to create or delete applications on the tag, and all the application level commands, which include the listing of all files within a selected application and the creation and deletion of files. At the file level we fully support value files, as these are used to store vouchers in the electronic wallet and partially support standard files,[2] but we do not support record files. As explained earlier, DESFire support up to 28 applications per card and 32 files per application. In theory we could have supported more applications with our emulator, but we respected this limitation.

On Java Card it is recommended [10] to construct all necessary object instances once during applet installation and to avoid constructing objects dynamically at runtime. Since not all Java Card runtime environments support a garbage collector, one can avoid out-of-memory situations by ensuring no additional objects are constructed at runtime. However, we opted not to initialize the maximum of 28 DESFire applications, 32 files per application and 14 keys per application since in practice only a fraction of these will be used. For this reason we instantiate the application, file and key objects dynamically at runtime. If the specific Java Card runtime environment supports it we ask the runtime environment to garbage collect the objects when they should be removed. If garbage collection is not supported we logically delete the object; in this case they remain in memory but they are no longer accessible.

For compatiblity reasons we install the applet using the same Java Card AID as registered by NXP. While selecting an AID on a target before doing a transaction is not strictly necessary on a native DESFire tag, it may be necessary on a secure element where multiple applets are installed or on a secure element where we cannot install an applet as being the default selected applet (e.g., on a UICC the SIM applet is already installed as default selected applet). For this reason our terminals are programmed to always select the DESFire AID before doing a transaction.

DESFire supports two different authentication schemes. The legacy authentication scheme is a challenge-response protocol based on 3DES in a non-standard cipher block chaining (CBC) mode and the standard authentication scheme supports both 3DES and AES in a standard CBC mode. Standard CBC decryption uses the block cipher in decryption mode, but the legacy DESFire CBC decryption uses the block cipher in encryption mode. This signifies that the CBC decryption library from the Java Card runtime environment cannot be used. In our work, we use legacy authentication because this allows for a simpler implementation, but we plan to migrate to standard authentication in the future. Note that a lot of the secure elements that we tested, lack AES support. This is not a problem for our project however, since we are using 3DES, which is supported by most secure elements.

The DESFire specification supports three communication settings, namely plain, MACed and fully encrypted. In the MACed setting, a message authentication code (MAC) value

[1]This implementation is available as open source software on http://code.google.com/p/java-card-desfire-emulation

[2]In our implementation the size of the standard file is limited.

is appended to every response from the tag. This MAC value is calculated using the CBC-MAC algorithm in the case of a legacy authentication session and the CMAC algorithm for standard authentication sessions. We made an implementation of the CMAC scheme, as this is not supported natively by Java Card. In the fully encrypted setting, a CRC16 or CRC32 checksum is added to the plaintext message for data integrity, and subsequently the message is encrypted. The Java Card architecture supports both these algorithms natively. However, because of endianess issues the CRC16 checksum from Java Card is not compatible with the version from DESFire. Furthermore, most Java Card runtime environments that we tested, do no support CRC32. Therefore we had to make our own table based software implementation for both these checksum algorithms.

While not fully covering the DESFire specification yet we managed to implement the DESFire applet for all the functionalities we need, in about 3600 lines of Java Card code.

C. Deployment possibilities on SE

According to Mobey Forum,[3] “The SE is a dynamic environment, where applications are downloaded, personalised, managed and removed independent of each other with varying life cycles.” The requirements are portability (if handset is changed, applications need to be available on new phone again), security (certified by payment industry or trusted third party), multi-application (each application provider has access to its own security domain in the SE), and remote management (download of tickets or top-up over-the-air (OTA) should be possible). The SE can both be accessed by the baseband controller (internal) or by the NFC controller (external).

The biggest issue, and main reason for the relative slow breakthrough of NFC, is who is going to control or manage the SE. There are in fact four possibilities, again all with their advantages and disadvantages:

- **Handset manufacturer centric approach:** In this approach the SE is integrated internally in the phone and the SE is managed by the handset manufacturer. In our opinion this is the least feasible option to deploy our applet, since it is seriously lacking the portability requirement.
- **MNO centric approach:** The Mobile Network Operators (MNOs) are already service providers and can therefore easily add a payment service to their list. In this approach, the SE will reside on the UICC (SIM), which fulfills the portability requirement.
- **Service provider centric approach:** External service providers e.g., Visa and Mastercard are very powerful players in the payment industry and always want their brand to be visible. In this approach, the SE is located on an external memory chip such as a microSD card or as an active sticker.
- **Neutral third party:** Because of the complexity of the NFC eco-system, a fourth option imposes itself, where an independent third party manages the SEs.

[3]http://www.mobeyforum.org/

Figure 1 visualizes the three possible locations for a SE in an NFC-enabled smartphone. Additionally there are two more options to add NFC functionality to legacy phones. In the remainder of this section we will discuss these five types of SEs and our experience with deploying the developed DESFire applet on them.

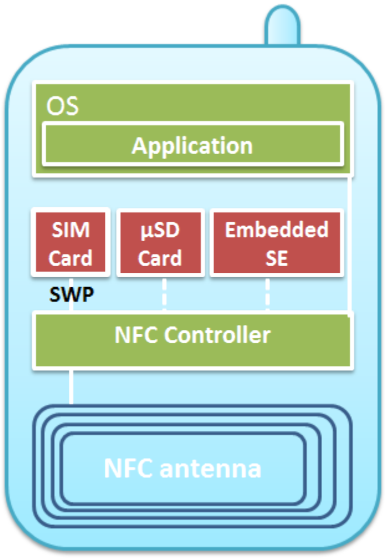


Figure 1. Possible SE locations

1) Embedded Secure Element: Smartphones that have a non-removable, integrated secure element, which is connected to the NFC controller, fall into this category. The NXP PN65 chip, which for example is used in the Google Nexus S, the Samsung Galaxy Nexus and the Samsung Galaxy S III devices, even embeds the NFC controller chip (PN544) together with a SmartMX secure element onto one chip. Other smartphones have a separate NFC controller and secure element.

Most NFC phones with an embedded secure element available today cannot be used to deploy the DESFire applet, because the embedded secure element is locked and can only be controlled by the manufacturer. For example on the Google Nexus S and the Galaxy Nexus the secure element is owned by Google and used for its Google Wallet service. The secure element embedded on the Samsung Galaxy SIII on the other hand is owned by Samsung itself. The relatively old Nokia 6212 feature-phone contains an embedded secure element on which we successfully deployed the DESFire applet onto.

2) Universal Integrated Circuit Card: Most smartphones already include a secure element in the form of a UICC, which is normally issued by an MNO and contains the SIM applet that enables secure authentication on the mobile network. To be able to use this secure element for NFC applications the SWP (Single Wire Protocol) standard was defined. This standard specifies how the communication between the NFC controller and the UICC works. To able to use this setup one has to use a SWP compatible UICC card together with an NFC smartphone, which uses SWP to communicate with the secure element.

Most UICC cards issued today by MNOs are not SWP enabled, probably caused by a lack of applications and the increased price compared to regular UICC cards. From Gemalto we received a UICC card, which supports SWP for research purposes. To test this setup we used the Android-based Nexus S smartphone, which supports SWP. When operating in card emulation mode the NFC controller uses the embedded secure element by default.

It is possible however to patch the libnfc-nxp library, which Android uses to communicate with the NFC controller, in such a way that the NFC controller is instructed to use the SWP link to access the secure element, which then is the UICC. After patching the code we can rebuild the Android OS and flash the device with the new ROM.

Initial testing shows that we can access the DESFire applet on the UICC over the NFC interface. There is still a slight problem with the patch that causes the reader to sometimes detect a Felica target instead of the expected ISO14443A target.

We can conclude that while this is certainly not a feasible method for regular user to apply, this shows that it should be possible for a MNO to decide to distribute SWP enabled UICC cards and a selection of patched smartphones to its subscribers.

3) Secure microSD card: Secure elements in the form factor of a microSD card are commercially available. Unlike the NFC UICC solution where there is the standardized SWP communication link between the NFC controller and the UICC, there is no such standard communication link to the SD card. There is an effort in Taiwan where a customized HTC device is used that has an SWP-SD link, however this device is not internationally available. In absence of such a direct communication channel between the NFC controller and the secure element a workaround could be developed in the form of a regular application running on the phones operating system, which can access both the NFC controller and the secure element, effectively acting as a proxy relaying APDU messages between both components. Besides the performance being sub-optimal this setup also requires the relaying application to be running constantly, which can on the other hand be more secure for the end user. Since the application can easily by modified extra care must be taken that the system is not subject to man-in-the-middle attacks by ensuring that all data passing through the application is fully enciphered. A secure system should

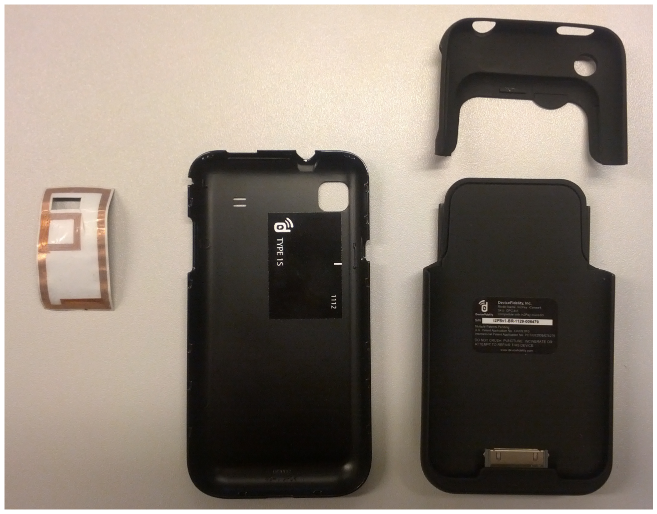


Figure 2. DeviceFidelity In2Pay

always be insensitive against man-in-the-middle attacks to prevent an attack on the radio interface, however such an attack is harder to achieve in practice than executing the attack against the software component.

4) NFC-enabled microSD card: An alternative to this is the use of microSD cards, which have an embedded NFC controller chip that will grant NFC functionality to the host device. The antenna itself can be either external or integrated in the package. Additionally, microSD cards use similar read/write functions as integrated NFC handsets.

Both have their advantages and disadvantages. An NFC microSD card is basically plug and play, thus eliminating difficult setups, but requires the handset to have a microSD slot.

The DeviceFidelity In2Pay product (see Figure 2) provides a microSD based secure element with integrated NFC antenna and external range extender (which is to be glued on the inside of the back cover), which could turn many smartphones with microSD slot into NFC smartphones [11]. The related iCaisse product is a case designed for iPhone. The case contains an NFC antenna, and a slot for the In2Pay microSD card, which together make an iPhone NFC capable.

For Android devices we also have rather bad experiences with the NFC SD card range extender. This should make it possible for an Android smartphone with a microSD slot, to work in the different NFC modes by amplifying the low power RF field generated by the SD card, using two tuned antennas. Even with the supported smartphone types, and matching back covers, the amplifier did not seemed to be reliable enough for our use case.

5) Active NFC bluetooth sticker [12]: Active NFC stickers provide a way to gain NFC functionality for any Bluetooth enabled smartphone. NFC stickers are contactless cards/tags designed to be glued on the back of a mobile phone and are designed to offer a solution to the current complex NFC ecosystem that prevents a world-wide integration of NFC technology in mobile handsets. A ferrite backing layer prevents distortion to occur between the components of the phone and its radio signal. The sticker also has an internal antenna installed for communication purposes. The NFC sticker on the other hand only needs a Bluetooth radio, which is a very common feature in most of the mobile phones currently produced. The drawback is however that these are less user friendly to setup compared to NFC MicroSD cards. The sticker has been designed to be as small as possible. A small low voltage battery is used to power the internal Bluetooth chip. This chip is responsible for setting up a connection with the handset and has the capability of making between 300 and 500 connections before the battery is drained. The battery itself can be wirelessly recharged using a specialized USB-charger.

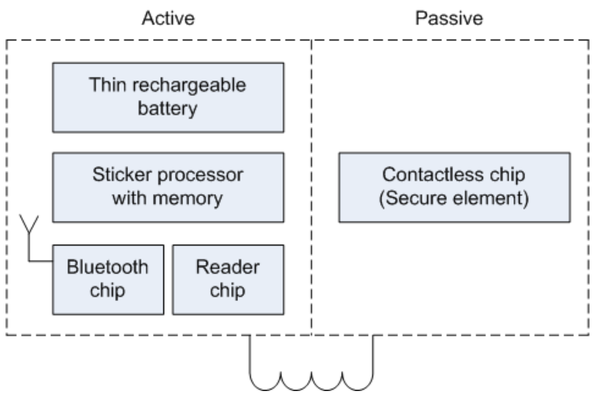

Active
Passive
Thin rechargeable battery
Sticker processor with memory
Bluetooth chip
Reader chip
Contactless chip (Secure element)

Figure 3. MyMax NFC sticker architecture

The MyMax sticker can act either passively or actively (see Figure 3). An active sticker will rely on its own internal power source, while a passive sticker will use an emitted magnetic field from an external reader to draw power from. Furthermore, the sticker can go into 3 different operation modes; the first mode is a passive mode where the sticker will act as a passive NFC tag. An external reader can be used to read and/or alter the contents of the NFC chip/internal SE on the sticker. It is important to note that a sticker operating in this mode will work completely independent from the handset that it is attached to (i.e., the handset is not required to be powered). The second mode requires that both the MyMax sticker and its corresponding mobile phone draw power actively from an internal power source. This mode allows a connection to be established between the sticker and the mobile phone. Consequently, the content of the internal SE/NFC chip can be read or changed by the handset through this link. The third mode takes advantage of the internal reader chip of the sticker, which makes it possible to create a connection to an external tag and allow the mobile phone to read or change the contents of this tag.

III. User interface

Mobile handsets offer a very large advantage in terms of user interactivity when compared to passive tags. The former comes equipped with a functional keyboard and screen and grants the possibility of an interactive interface for the mobile electronic wallet.

A user interface (UI) provides the user a quick overview of the items (both online and offline) available on his electronic wallet. The user should be able to browse through the coupons, loyalty cards etc, and consult the remaining value of the vouchers. Additionaly, it should be possible to buy new items or top up existing vouchers OTA using the 3G connection. Ideally this UI should be available across the different major smartphone platforms.

Section III-A provides an overview of the different possibilities for developing a UI that interacts with the wallet on the SE. In Section III-B and III-C we will describe the implementation of two possibilities.

A. Options for the user interface

We identified the following possible interface types: a native application, a SIM Application Toolkit (SAT) applet and a Smart Card Web Server (SCWS) servlet.

A native application for a specific platform like Android has the disadvantage of not being cross-platform, but is the most flexible in terms of interacting with the hardware like the NFC controller. The cross-platform issue can be partly diminished by making use of framework like PhoneGap[4], which allows the generic UI logic to be written in HTML and JavaScript to allow reuse over different platforms. The platform specific parts like interfacing with the hardware are provided by plugins per platform.

The SAT on the other hand offers more interoperability since everything is stored on a SIM card, which is useable by most handsets and is furthermore deemed to be very secure. While this may sound tempting to use, the downside is that it lacks seriously in terms of visual interface options. Recently more and more smartphones are not including support for SAT interfaces anymore, probably because of the poor usability and the fact that SAT application never were really successfull. For these reasons we did not implement a SAT-based solution.

A rather new and promising option is the use of a SCWS installed on a Java Card. A SCWS is a HTTP 1.1 web server embedded on a Java Card and is available on some secure elements since the Java Card 2.2 version, offering a device independent way of the management of personal user data in a secure fashion. This option still puts the application on the SE for security and interoperability purposes. The difference is however that a relatively good HTML interface can be offered by providing static and dynamic content to the browser of a mobile phone through the http://127.0.0.1:3516/

[4]http://phonegap.com/

address, which is OS independent. If SCWS applications become popular it is to be expected that a closer integration in the smartphone's OS will be provided. For Android a patch [13], which enables the browser to contact the SCWS using the Bearer Independent Protocol instead of the normal TCP/IP stack, is currently required. The application and data can be additionally managed externally through an over-the-air link using web protocols. A secure tunnel will be opened between the SCWS on the Java Card and the OTA platform, which is used for the administration of the SCWS.

The last interface type seems to be the most beneficial in terms of the electronic wallet project, because of its advantages in both maintaining a good visual interface as well as the interoperability, security and user friendliness aspects. However, the disadvantage of the SCWS is that most UICC deployed today do not include support for SCWS. The price of an UICC with support for SCWS is still substantially bigger than a normal UICC. Combined with the fact that there are no compelling SCWS applications yet there is no reason for MNOs to issues the more expensive UICC to its subscriber at the moment. In the next section we will go into more details about a SCWS implementation.

B. User interface based on Smart Card Web Server

The content of the event wallet is displayed in the interface on the mobile phone of the user. This HTML based interface can be divided into a number of interface components that are individually requested from a SCWS residing on the Java Card through various HTTP requests.

The servlet that is responsible for providing the interface with the necessary data is running on the SCWS and will communicate using the Shareable Interface Objects (SIO) mechanism [14], [15] to the DESFire emulator on the Java Card. This emulator is another applet residing on the Java Card and will (after authentication) provide the servlet with the desired data. The servlet will then, based upon the collected data, give an answer to the previously made request by the interface.[5]

Updates to the wallet content on the emulated DESFire card are done mainly through external point of sale terminal, which are available on either the event itself or through an OTA purchase. The design of a mobile interface requires some special attention, more so than its workstation counterpart.

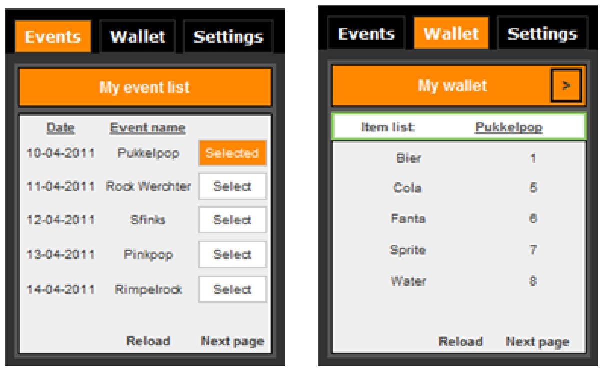
1) Application interface: First, there is a large differentiation between different cell platforms, each whom presents its own interface. Secondly, physical obstacles such as different screen sizes, aspect ratios and physical buttons need to be taken into account as well. Lastly, there is the user aspect, which demands that an interface needs to be intuitive and easy to learn.

[5]For a SAT-based UI, the method to access the DESFire applet is the same, since both are Java Card applets running on the same SE as the DESFire applet itself.

We made the decision to use a SCWS to provide a consistent interface by taking full advantage of the mobile phone browser capabilities, which is tasked to render an HTML based interface for the wallet application. This interface is theoretically universally applicable to any mobile phone that supports Java Card. Recent developments like jQuery Mobile[6] and PhoneGap make it possible to build native looking and responsive HTML and JavaScript-based applications.

The design of the interface of the application itself is based upon known design principles [16], which dictate how information on a page is to be presented to its user. This includes but is not limited to bringing information to the top of the interface by limiting the amount of links a user has to go through thus, minimizing navigation or bringing a collection of relevant information together, based on the desired intent of the user.

2) Interface structure: The wallet interface consists of three main tabs that allow a user to browse through a number of subtabs. The main interface tabs are the ”Events”, ”Wallet” and ”Settings” tab.



Date	Event name	
10-04-2011	Pukkelpop	Selected
11-04-2011	Rock Werchter	Select
12-04-2011	Sfinks	Select
13-04-2011	Pinkpop	Select
14-04-2011	Rimpelrock	Select

Item list	Pukkelpop
Bier	1
Cola	5
Fanta	6
Sprite	7
Water	8

(a) The event tab (b) The wallet tab

Figure 4. The user interface

The ”My event list” subtab (Figure 4(a)) is used to list all the possible events that a user currently has access to. This list is kept up to date through a connection with a backend server. A number of possible items are linked to each specific event and are shown in the ”My wallet” subtab (Figure 4(b)). Items can be bought and/or spend using the local terminals or through OTA functionality.

The settings tab allows for various options, including setting the number of listed items and/or events on the events and wallet tab pages. The interface is resolution independent and can thus be used on a number of different handsets. The layout can additionally be changed altogether depending on the preferred style of the user.

Finally, the wallet tab includes the option of purchasing items OTA. The purchased items will be listed on the ”My

[6]http://jquerymobile.com/

wallet" subtab as "reserved items", meaning that they still need to be synchronized by specialized terminals called "sync points". The main advantage of this is that the wait time for the user at a terminal will be cut down significantly since a user only has to touch the terminal to transfer the previously purchased items over the NFC link.

3) Data flow: The next part of the application consists of two elements, namely the GUI Servlet and the DESFire emulator, which are both tasked with the provision of the actual data to the interface mentioned earlier.

A GUI Servlet is a Java Card application that runs on the SCWS and will act both as a server and a client when data is requested by the interface, since it will serve information to the interface after it has requested the necessary data from the DESFire applet.

4) Retrieval of data: A backend server forms the backbone of the system, because all the event dates, event names and item names are requested from this server. Providing a page in the interface with values requires the servlet to request a list of AIDs and FIDs first from the DESFire emulator.

These IDs represent the various events and items tied to an event respectively and are required to be translated by the back-end server to their actual corresponding names. The item values shown on the wallet page are collected by using an AID to select a specific event on the DESFire emulator and then request the value of a specific item of that event using its FID.

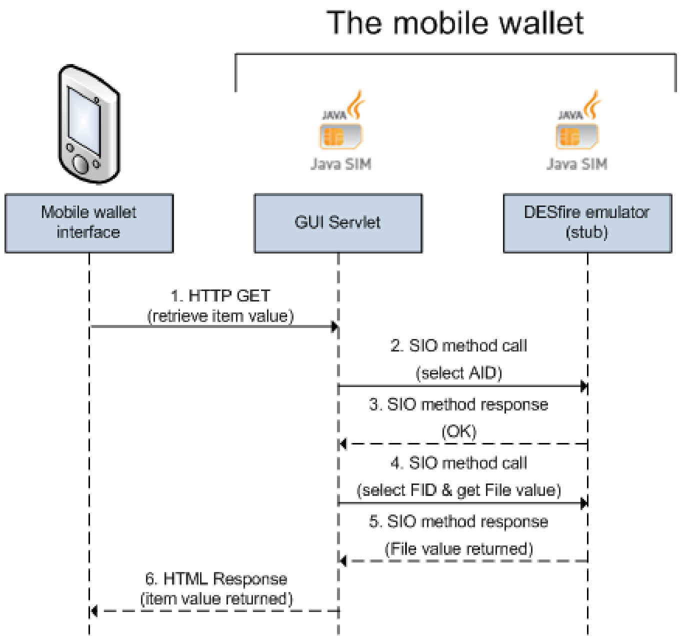

The mobile wallet
Java SIM
Java SIM
Mobile wallet interface
GUI Servlet
DESfire emulator (stub)
1. HTTP GET (retrieve item value)
2. SIO method call (select AID)
3. SIO method response (OK)
4. SIO method call (select FID & get File value)
5. SIO method response (File value returned)
6. HTML Response (item value returned)

Figure 5. Retrieval of an item value

Every request done by the interface will thus trigger a series of steps in the background of the application (Figure 5). The GUI servlet will perform a SIO method call to select the AID that is linked to the desired event. The emulator will respond with a confirmation message and the actual file value will then be requested next by using the FID of a specific item. The File value is passed back to the servlet, which will pass it to the interface for visualization.

The backend server and its data have been temporary replaced in our project by a number of vectors that store the different names and dates mentioned before in a hardcoded manner. The reason for this is identical to the stub in that it is caused by the fact that the EVENT project is still in a relatively early stage. We implemented and tested the SCWS using an emulator, since a SCWS enabled SE was not in our possession at the time of writing.

C. *User interface based on a native application*

The mobile application needs to communicate with the DESFire applet on the secure element using the same APDUs that are used by the terminals. The problem here is that there is not yet a standard API available in the Android SDK to communicate with a secure element from an application. In 2011 Giesecke & Devrient (G&D) proposed patches to the Android Open Source Project that enable a simple interface for a developer list all the supported and available secure elements, and to send APDUs to a secure element [17]. This SmartCard API is an implementation of the SIMalliance Open Mobile API [18] that enables an Android application to communicate with a secure element. This SmartCard API can be extended with plugins (named terminals), which implement support for accessing a specific secure element type. By default terminal types are included for ASSD (Advanced Security SD Card) secure elements, UICC and embedded SmartMX based secure elements. The plugin architecture allows us more secure element types like the MyMax sticker or the DeviceFidelity microSD card to be accessible from applications using the same standard SmartCard API, thus providing an abstraction layer around the specific secure element.

Figure 6 shows the different possibilities of communicating with an SE from an application.

1) G&D Mobile Security Card: This SD card based secure element was used to establish a working link with a secure element through the SmartCard API. Once this link was established communication with the DESFire emulator could be done directly, because it was installed as default selected. This setup was tested with a Samsung Galaxy Y, a smartphone without NFC, to get started on the graphical user interface for the application and secure element communication. The terminal interface could not be tested with this secure element, because of the absence of an NFC controller and antenna.

2) DeviceFidelity In2Pay and iCaisse: Accessing the In2Pay secure element happens with the In2Pay API that is available for all major smartphone operating systems. We

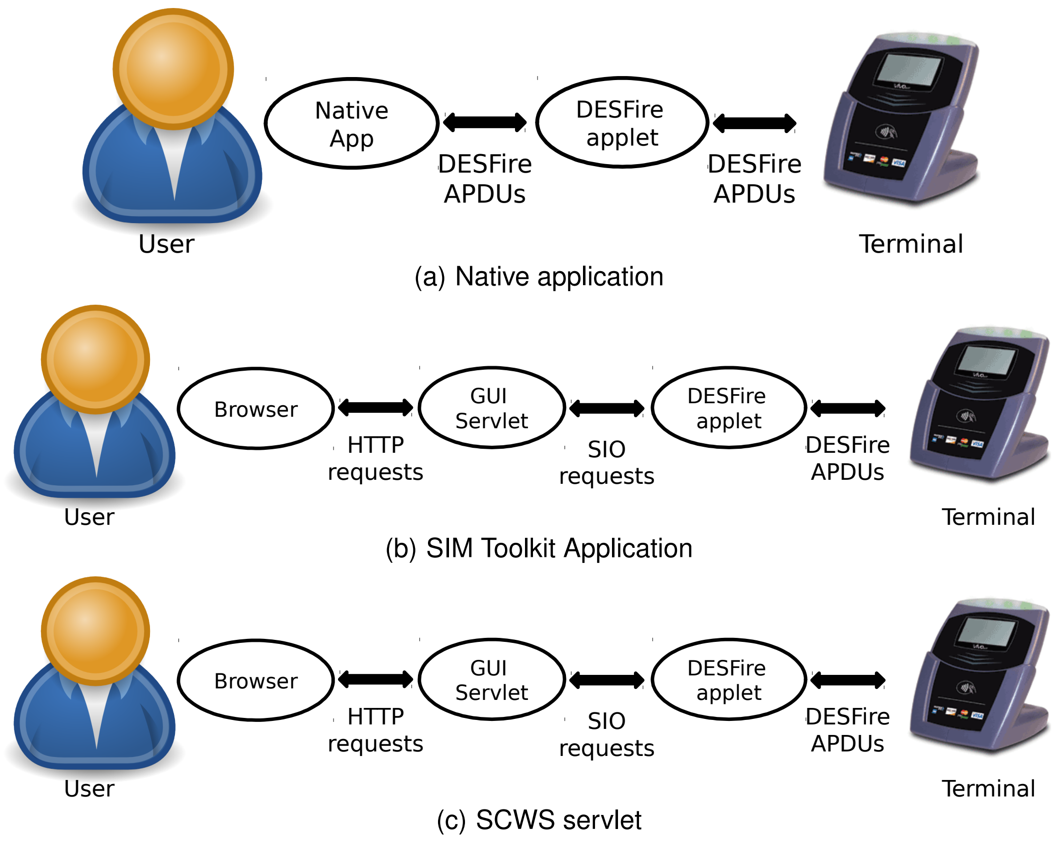


Figure 6. Interface types

tested this on the Android OS, and on iOS (making use of the iCaisse). Communication with the DESFire applet through this API gave us timeout errors on some APDU commands, while accessing the applet over the NFC interface worked as expected.

Looking to resolve this issue we established that communication time with the DESFire emulator on the Device-Fidelity card took longer than when deployed on MyMax NFC sticker, using Jload. The command to retrieve a list of all applications that the DESFire emulator contains, took about 312 ms while deployed on the sticker it took only 140 ms. Similar differences were found for other commands. Remarkable was that a select command on an application on the DESFire emulator took 203 ms and gave no time out error, whereas the command to get all value files from a application, when only one was available, took only 16 ms more. When looking closer we established that the command time to get the file value, was only 94 ms while it took 125 ms to select the corresponding application.

3) Active NFC sticker: The Twinlinx MyMax sticker comes with an Android API, which provides applications any easy interface towards the sticker. The API allows the application developer to set up the Bluetooth connection, to connect to the internal SE and transmit APDUs. There are some usability aspects that cause this solution to be more complex and cumbersome for a user than an integrated solution however. Before the first usage one has to set up the Bluetooth pairing manually. Before executing a transaction the user has to remember to turn the sticker on. The sticker will shutdown automatically after a short period, to conserve battery. We perceived accessing the sticker from the application as rather slow and not user friendly. The next version of the sticker and API will have a feature that allows the phone to vibrate using a certain pattern, which will be detected by the sticker so it can turn itself on. We have not yet received this new version, but this could enhance usability.

4) UICC: To build a UI to interact with the wallet stored in the DESFire applet on our UICC, we need to be able to communicate with the UICC using APDUs. Currently there are several obstacles preventing us to do this on Android. The Android operating system runs on the application processor, where the UICC is usually connected to the modem or baseband processor. The OS can communicate with the UICC by using a small set of AT commands that are supported by the baseband processor. The 3GPP 27.007 specification [19] defines the following additional AT commands that enable generic SIM access using APDUs:

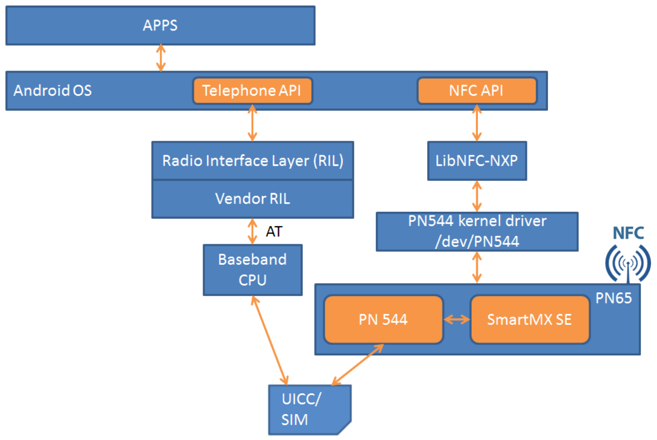


Figure 7. Android UICC access overview

- AT+CSIM: Generic SIM Access
- AT+CCHO: Open Logical Channel
- AT+CCHC: Close Logical Channel
- AT+CGLA: Generic UICC Logical Channel Access

We do not know of an Android device that has a baseband processor supporting the mentioned AT commands at the moment. These changes also need to be made in the Android Radio Interface layer (RIL), this is an abstraction layer between the Android telephony manager services and the baseband modem. RIL exists out of two parts (see Figure 7): the RIL daemon, which is a wrapper to have a standardized interface to the second part, and a vendor specific library. At the moment, only a subset of the SIM ToolKit (STK) [10] is available and transparent APDU communication with the UICC is not allowed. The AT Commands that are needed [20] are specified by the RIL extension specification from G&D [21].

The vendor specific RIL is a closed sourced component that cannot be patched to allow generic APDU access. However, the SmartCard API project contains a patch for the Android emulator that allows applications running in the emulator to access a UICC that is attached over a Personal Computer/Smart Card (PC/SC) interface to the host computer. The SmartCard API also proposes a patch to Androids Telephony API, extending this API with methods for SIM access. Since accessing the UICC using the official RIL way seems not feasible today without extensive manufacturer support we wanted to try to communicate with the UICC via the NFC controller. This would be theoretically possible, by adapting the libnfc-nxp driver, if the NFC controller chip (e.g., PN65) would allow this. We did not succeed in getting access to the PN65 datasheet from NXP however.

IV. Conclusions

In this paper, we demonstrated that backwards compatibility with a tag-based solution can be accomplished on NFC-enabled smartphones. We have developed a JavaCard applet that implements the DESFire specification, and we have tested the applet on many different physical SEs. For our use case, which starts from an existing payment infrastructure, no changes were needed to the terminal software, proving the backwards compatibility. Consequently, if NFC technology becomes widely available in the nearby future, the payment infrastructure that is already in place, can be reused. Although no extensive studies were performed, the smartphone solution gives satisfactory results regarding transaction performance.

In contrast, accessing the wallet content with a UI remains a challenge. We have identified three approaches to interact with the DESFire applet. We had limited success with developing a native Android application to access the DESFire applet on an external SE (such as a microSD card or the bluetooth sticker). However, we were unable to test a solution using an embedded SE, because the DESFire applet cannot be deployed on the SE without support of the handset manufacturer. Furthermore, in the case of a UICC-based SE the vendor specific radio interface layer prevents access to a custom applet on the UICC. Again for this solution to work support from a handset manufacturer is required.

The SAT UI approach appears technically feasible on selected devices. However, this text-based solution does not provide a modern UI experience and therefore we did not investigate this option in our work.

A SCWS-based UI seems a promising approach and we showed the feasibilty in emulation. However, the commonly available SEs do not yet support SCWS. Additionally, support for SCWS is lacking in today's smartphone OSes.

From a users perspective, the NFC UICC appears the most suitable solution because it allows to transfer the electronic wallet from one phone to another. The same applies to a microSD based solution, but not all devices are equipped with a microSD slot. Embedded SEs lack this transferribility feature, which is beneficial as in our use case vouchers are stored offline on the SE; for an online scheme this is not a strict requirement.

Finally, we would like to point out that, while the development of an contactless electronic wallet in a smartphone is technically feasible, the main hurdle preventing market deployment of such a system is the difficult NFC ecosystem. There are too many SE options and it is unclear which stakeholder is in control of the SE.

Acknowledgements

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the Flemish iMinds projects, and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT-II. In addition, this work was supported by the Flemish Government, IWT SBO MobCom and IWT Tetra EVENT.

The Technology Transfer (Tetra) program of the IWT, the government agency for Innovation by Science and Technology[7], is meant to help Flemish companies and research centres in realizing their research and development projects. Together with the support and feedback from Antenor, Buzy.be, easyFairs, Events Catering Bevers, Event Drive, First of Kind Solutions, Mobile For, and TwinLinx both research groups managed to finish the EVENT project successfully by giving the partners a well-founded vision on the electronic wallet of the future through live test events in the field.

References

[1] K. De Kock, T. Van Herck, G. Ergeerts, R. Beyers, F. Schrooyen, M. Ceulemans, and L. Wante, "Building the bridge towards an open electronic wallet on NFC Smartphones," in *MOBILITY 2011 - Barcelona, Spain*, 2011.

[2] N. Forum, "Whitepaper: Essentials for successful NFC Mobile Ecosystems," p. 24, 2008, [accessed 10-July-2011]. [Online]. Available: http://www.nfc-forum.org/resources/white_papers/NFC_Forum_Mobile_NFC_Ecosystem_White_Paper.pdf

[3] NFCInsight, "NFC Payments Fact Pack," in *2nd Annual NFC Payments Europe 2012 Conference & Exhibition*, 2012, pp. 2–15.

[4] J. Neefs, F. Schrooyen, J. Doggen, and K. Renckens, "Paper Ticketing vs. Electronic Ticketing Based on Off-Line System 'Tapango'," in *Near Field Communication (NFC), 2010 Second International Workshop on*, april 2010, pp. 3–8.

[5] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia, "A Practical Attack on the MIFARE Classic," in *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, ser. Lecture Notes in Computer Science, G. Grimaud and F.-X. Standaert, Eds., vol. 5189. Springer, 2008, pp. 267–282.

[6] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs, "Dismantling MIFARE Classic," in *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, ser. Lecture Notes in Computer Science, S. Jajodia and J. López, Eds., vol. 5283. Springer, 2008, pp. 97–114.

[7] F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur, "Wirelessly Pickpocketing a Mifare Classic Card," in *30th IEEE Symposium on Security and Privacy (S&P 2009), 7-20 May 2009, Oakland, California, USA*. IEEE Computer Society, 2009, pp. 3–15.

[8] Gemalto, "Worlds First: Gemalto Integrates DESFire Transport Card into NFC Mobile Phone," Feb. 2010, [accessed 12-December-2012]. [Online]. Available: http://www.gemalto.com/php/pr_view.php?id=704

[9] J. Prado Casanovas and G. Van Damme, "DESfire Emulation Using Java Card," in *Trustworthy Embedded Devices*. Leuven, Belgium: Conference Publishing Services IEEE, 2011, p. 5.

[10] Gemalto, "Java Card & STK Applet Development Guidelines," 2009, [accessed 10-July-2011]. [Online]. Available: http://developer.gemalto.com/fileadmin/contrib/downloads/pdf/Java_Card_STK_Applet_Development_Guidelines.pdf

[11] DeviceFidelity, "Mobile Contactless Technology Backgrounder," Jun. 2011, [accessed 12-December-2012]. [Online]. Available: http://www.devifi.com/assets/whitepaper.pdf

[12] TwinLinx, "MyMax2 NFC sticker presentation – Convenience in your hands," Nov. 2011, [accessed 12-December-2012]. [Online]. Available: http://www.twinlinx.com/upload/file/mymax2presentation.pdf

[13] G&D SmartCard API, "Bearer Independent Protocol," 2011, [accessed 10-July-2012]. [Online]. Available: http://code.google.com/p/seek-for-android/wiki/BIP_Extensions

[14] M. Montgomery and K. Krishna, "Secure Object Sharing in Java Card," pp. 14–14, 1999.

[15] D. Perovich, L. Rodriguez, and M. Varela, "A Simple Methodology for Secure Object Sharing," p. 7, 2000.

[16] K. Holtzblatt, "Customer-centered design for mobile applications," in *Personal and Ubiquitous Computing archive,Volume 9 Issue 4, July 2005*, 2005, pp. 227–237.

[17] G&D, "Secure Element Evaluation Kit for the Android platform," 2012, [accessed 10-July-2012]. [Online]. Available: http://code.google.com/p/seek-for-android/

[18] SIMAlliance, "Open Mobile API Specification," 2011, [accessed 10-July-2011]. [Online]. Available: http://www.simalliance.org/en/resources/specifications/

[19] 3GPP, "3GPP Specification Detail," 2012, [accessed 10-July-2012]. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/27007.htm

[20] Telit Wireless Solutions, "AT Commands Reference Guide," 2012, [accessed 10-July-2012]. [Online]. Available: http://www.telit.com/module/infopool/download.php?id=542

[21] G&D, "RIL extension specification," 2012, [accessed 10-July-2012]. [Online]. Available: http://code.google.com/p/seek-for-android/wiki/UICCSupport

[7] http://www.iwt.be/english/welcome