INF-3981
Master's Thesis in
Computer Science

Architecture for generic service development on mobile handsets

Peter Dahle
peterda@stud.cs.uit.no

June 15, 2007

Faculty of Science
Department of Computer Science
University of Tromsø

INF-3981

Master's Thesis in

Computer Science

Architecture for generic service development on mobile handsets

Peter Dahle
peterda@stud.cs.uit.no

June 15, 2007

Abstract

Several actors are involved in all layers the handset architecture, from hardware producers to service providers, and the numbers are raising. Lack of collaboration amongst these actors across and within layers has led to a complex development-process of services and applications, which in turn leads to difficult use of such applications and services.

In this thesis we took a closer look at the mobile phone, examined challenges surrounding development and use of services on mobile phones, and found initiatives by actors to handle these challenges. This knowledge was used to design and implement a solution to handle identified challenges.

The solution involves using the UICC as the main application platform and container of state, with the possibility to deploy handset-specific parts of an application on the handset. Standardized tools on the handset give UICC-applications the means to communicate with external processes and users. In addition the network operator is given an important role to administrate and adapt applications on the UICC as services change communication technology and application standard.

The designed architecture facilitates more widespread development and use of services on the mobile handset. The architecture is realizable through current platforms and standards. By implementing a simulation and subset of our design on a handset the design was substantiated.

Acknowledgments

First, I would like to thank my supervisors for guiding me through the process:

- Bjørn Thorstensen, Research Manager, Telenor R&I
- Are Johannessen, Researcher, Telenor R&I
- Weihai Yu, Associate Professor, University of Tromsø

For all the technical support I would like to give thanks to:

- Erlend Pedersen, Researcher, Telenor R&I
- Kjell Myksvoll, Researcher, Telenor R&I

An equal gratitude is given to my family and friends for moral support throughout the process of writing this thesis.

Thank you.

Table of contents

1 Introduction

1.1 Background

In the last three decades, the mobile phone has gone from being a simple voice-service to becoming a multipurpose service platform. In the beginning, the mobile phone was meant to have the same basic functionality as the fixed telephone once invented by Graham Bell. The fixed telephone basically contains a receiver, transmitter, microphone and a loudspeaker. Mobile phones in the start of the mobile phone era contained the same functionality, with the addition of being mobile. This ability made it possible for people to converse with colleagues, friends and family while they were somehow on the move.

As mobile phones became smaller, more powerful and cheaper more and more functionality and technology was added to the phone. In the beginning such phones came out of the car and into briefcases. Displays were added and the phone was looked upon as high tech. But still the phones have come a very long way from the car phone intended for rich businessmen. Having a device that could search all imaginable information around the world, take pictures, play music and show movies, all in a package of the size of a miniature bible, was science fiction when the first mobile phone was presented.

Today the mobile phone is a necessity for people in the industrial countries, and the phone is no longer just a tool for oral conversation. They are advanced devices with varying types of added technologies. Such technologies include cameras, mp3 players, Internet connections, video telephony, office tools and so on. As mobile phones are becoming more powerful (both in terms of electrical power and processing power) and cheaper to produce, existing power/processing-consuming technology and newly invented technology are added to the package. The fast-moving evolution of mobile phone's technology has lead to rapid replacements of phones amongst users.

The mobile phone has several means of communication for applications on several application platforms, in addition to being a phone. As new and emerging technologies such as NFC ([1]) and Zigbee ([2]) are applied to the mobile phone, new actors such as transport- and credit card -actors are starting to provide different services based on these technologies. Such services often take the form of an application deployed on the phone.

Service provision can be defined as an economic activity that does not result in ownership ([3]), or as work done by one person or a group to benefit another ([4]). Services on mobile phones typically include transactions (both economical and other) to perform a task for the user. Such tasks can be to obtain some desired information, pay for products, register for entrances to a controlled area etc., and these transactions take place through communication technologies that the mobile phone can offer to services. We can divide services in two groups:

- Stateless services (e.g. bus-route information through web-browsers)
- Services that require state (i.e. service that need to know something about the user in order to provide the service).

The latter group of services can be further divided in two sub-groups:

- Services that utilize state stored locally (e.g. credit cards)
- Services that utilize stated stored remotely at the service provider (e.g. web-mail).

Services that require state stored remotely at the service provider can be accessed through web-browsers and alike. This has been done for many years, and has caused relatively few problems. New technologies on mobile phone such as NFC and Zigbee facilitate services that require state stored locally on the mobile phone. Examples of such services are ticketing and credit card services on the mobile phone. When state is stored locally on the mobile phone, dedicated applications deployed on the phone are often required to perform the service for the user.

1.2 Problem statement

In the world of mobile telephony there are a wide number of actors, and the number is rising. More people buying and using mobile phones create a larger market for telecom operators and mobile phone producers. New technology creates a larger market for application developers and creates new markets for actors normally not viewed upon as mobile phone actors (such as credit card and transport actors).

With such an amount of actors on the mobile phone marked today, it has so far been difficult to get actors to collaborate. This lack of collaboration has led to slow progress in standardisations on different levels of the architecture of mobile phone's services. Some aspects of the architecture have been standardized, some are undergoing standardization and some are yet to be standardized.

Application development on the mobile phone has several challenges surrounding standardization, both the high number of standards and the lack of standards. Combined with the number service providers and their specifications, development and use of applications intended for services has become a difficult task. Applications often have to be developed to fit a single or small group of mobile phones. If we add this to the fact that users change their mobile relatively often, services on mobile phones are difficult to maintain over time for developers and users. This has spawned several initiatives in order to facilitate more widespread development and use of such services.

1.3 Goal

Our goal in this thesis will be to pick up on these initiatives, and try to *design an architecture that facilitates more widespread development and use of services on mobile phones*. To do this, we have to address these challenges:

1. Lack of interoperability on software platforms
   Handsets have a wide variety of software platforms for development and deployment of applications.

2. Lack of standardization of communication technology for accessing services
   A type of services can be accessible through a number of different communication technologies (E.g. ticketing services can be accessible through NFC, Zigbee, SMS etc. depending on the service provider).

3. Lack of application standards
   Service providers within the same type of services use a wide variety of application standards for their services.

4. Loss of state about services when users change handsets
   Users change handsets regularly, and state about services are often lost in these transitions.

1.4 Outline

In chapter 2 we begin by taking closer look at the mobile phone. The chapter continues with examining how the phone communicates with the network operator, before we look at the different software platforms available on the phone. Then we examine the challenges that surface surrounding services on mobile phones, and we end the chapter by examining existing initiatives and standards to handle these challenges.

Chapter 3 states some goals and requirements we want our own solution to the challenges surrounding services to meet.

Chapter 4 gives a design to satisfy the goals and requirements from chapter 3.

Chapter 5 shows how we implemented the design from chapter 4.

In chapter 6 we test our implementation based on the goals and requirements from chapter 3.

In chapter 7 we analyse and discuss our design, implementation and tests against the goals and requirements from chapter 3.

Chapter 8 concludes on how our thesis met the goal we stated in this chapter, and the requirements in chapter 3.

We end this thesis by coming with suggestions to further work in chapter 9.

2 The mobile handset

In telecommunication terminology the mobile phone is referred to as a mobile terminal or handset. We will start by giving an overview of the mobile handset, including its basic features and additional technologies. Then we take a look at how the handset communicates with the network operator, and take a look at the software platforms that exist on the handset. We continue by examining challenges surrounding services on handsets, before we take a look at initiatives in order to handle some of these challenges.

2.1 Overview of the handset

In order for mobile handset to work as a mobile phone, it needs at least a radio receiver and transmitter, microphone and loudspeaker. Somehow the user needs to type an address for his or her call, and some kind of input device is needed. Common practice is also to add a display so the user can read different kinds of information on the handset. All these components are common on mobile handset today, and are quite similar to a fixed phone. But apart from the fact that the reception and transmission devices are wireless, there is one major difference between fixed and mobile handsets: The Subscriber Identity Module (SIM).

Figure 1: The SIM

Identification of a fixed handset depends on the plug the handset is connected to. Calls are directed to a fixed position in the network, and phones can be replaced on one location without hassle. This type of location-identification is not possible for mobile handsets. Instead the SIM identifies the mobile handset in the network. The SIM is a dedicated application residing on a Smart Card (Figure 1). The physical Smart Card is commonly known as the SIM card since this smart card needs to at least contain the SIM application, and most SIM smart cards currently contain only the SIM application and possibly a SIM toolkit. Plugging the SIM card into a new phone directs calls to the owner of the SIM card, and to the new phone. Hence, the SIM needs to be connected to the mobile handset in order for it to work as a phone.

If a device has a SIM card and radio receiving and transmitting capabilities, it can be considered to be a mobile handset as far as the hardware architecture is concerned. In addition, all mobile handsets sold today include a display and an input device. We can call these features standardized hardware features of a mobile handset.

Figure 2: Nokia 3110 with basic call functionality

In addition to the standard features, mobile handsets can contain a wide spectre of additional devices. These devices can range from additional communication interfaces like Bluetooth, WLAN and NFC to entertainment equipment like cameras and mp3 players. What kind and amount of additional devices added to the handset depend mainly on the price, size and intended use (beyond basic telephony) for the mobile handset. There is no standardization of mobile handsets functionality beyond basic telephony functionality; hence there is no standardization for what extra devices a mobile handset should contain.

Figure 3: Nokia N93 with 3,2 megapixel camera, Mp3 player and WLAN interface.

How the actual hardware is designed is up to the producer of the hardware. But communication devices like the radio receivers and transmitters, Bluetooth and WLAN must follow standardized protocols in order to actually communicate with the outside world. Where this protocol functionality is placed (as fixed circuits, firmware or software) depends on the technology and the producer.

All devices in the mobile handset are connected to a main circuit board (Figure 4). This circuit board has a CPU and memory attached. In order to get the different devices working together and making the phone work, some kind of code needs to run on the CPU. Mobile handsets are connected both physically and logically together with circuit boards and software similar to PC's: An OS on the handset does the logical connection between the different elements. Applications and runtime systems can run on top of this OS. In addition it is possible to place applications on the Smart Card that contain the SIM.

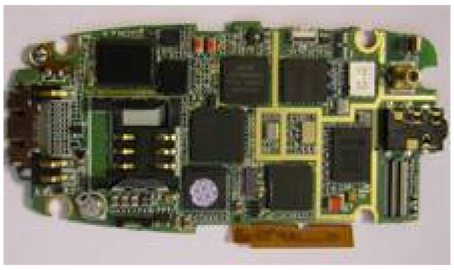
Figure 4: Main circuit board of a mobile handset.

2.2 Network-handset communication

Mobile handsets communicate primarily through radios. Even though you could bring your handset all over the world and probably use it as long as you have coverage, different radio frequency -standards applies to different parts of the world. In Europe and Asia 900 and 1800 MHz is used, and some countries in America use 850 and 1900 MHz. Handset manufacturers often equip their handsets with all frequencies so their handsets can be sold all over the world. Their owners thus can use the handsets as they travel to different frequency regions.

Several base stations build up the network in which the handset is connected to. These base stations are parts of a cellular network, hence the often used name; cellular phone. Every base station is a cell in the network, and handsets reconnect to a new cell every time they leave the coverage of the residing cell. This reconnection is called a handover or roaming. Also, the fact that handsets is within range of one or more base stations as long as it is connected gives the network operator the possibility to offer location-based services ([5, 6]).

There are two ways used to do handover in networks today, the GSM way, and the CDMA and WCDMA way. When a handset in a GSM network detects that it is about to leave a cell, it finds the next cell it should connect to. The handset reserves a new channel on the new cell while it still uses the old channel on the residing cell. When the handset detects that the signal strength from the residing cell goes below a certain threshold, it switches channel to the reserved channel on the new cell. This is called a hard handover. In CDMA (also called IS-95) and WCDMA (also called 3G) both channels can be in use at the same time. Handovers can in these systems also be used for load balancing in overlapping networks.

As long as the handset is connected to a cell, several means of communication is possible on top of the radio connection. The two most commonly used are voice and SMS. Short Message Service (SMS) is standardized in GSM 03.40, and defines the means to send short messages point-to-point. Most commonly, these messages contain plain text sent from one person to another. But it is also possible to send binary data designated for a port on the receiving handset, similar to IP sockets. SMS' are encrypted as they are sent between handsets and base stations. An enchantment to the SMS was the MMS (Multimedia Message Service -3GPP TS 23.140), which made it possible to add pictures, sound and video to messages.

In addition to the basic voice and SMS functionality, other means of communication are offered by the network operator through the base stations. These means of communications are mainly data-traffic services such as GPRS, EDGE, UMTS and HSDPA. GPRS and EDGE are second-generation (2G) data-traffic technologies while UMTS and HSDPA are third-generation (3G) technologies. For these technologies to be usable, both the handset and the base station it is connected to have to have support for the desired technology. Today, most handsets have support for GPRS, many have support for EDGE, all 3G handsets have support for UMTS, while HSDPA is an emerging technology. Base stations in densely populated areas have in most cases support for both 2G technologies and UMTS.

SMS and data-traffic services offered by the network operator offer secure means of communication between handsets and external services. Such secure channels are well suited for traffic between the handset and network operator, and between handset and service providers that require security in their communication, e.g. payment services. SMS has one advantage over the data-traffic services since a handset is always reachable through the users phone-number as long as it is within coverage of a base station. This is not the case with data-traffic services. These services are based on IP addresses, and the IP addresses are assigned dynamically as handsets log on to the network. This means that the processes on the handset need to be connected to a data-traffic service, and an outside process needs to know the IP before it can initiate communication with processes on the handset. Hence, data-traffic services are best suited when the communication is initiated by services on the handset.

2.3 Software platforms

The software running on the mobile handset is similar to the better-known software running on a desktop computer. The software is built in a hierarchical fashion with operating systems, runtime systems and applications (Figure 5). In addition, the SIM card plays a vital role in the software-hierarchy-design of handheld handsets.

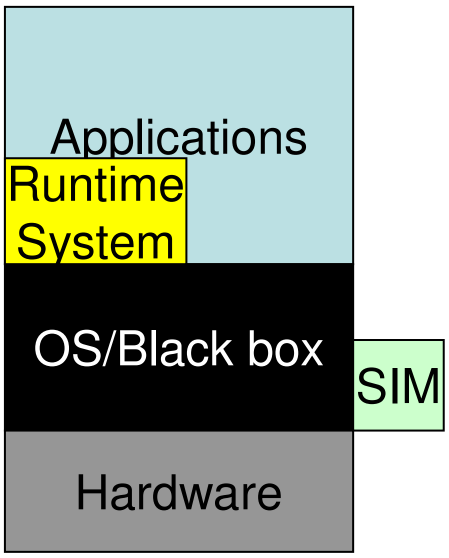

Figure 5: Software architecture

2.3.1 Operating systems

Mobile handsets can have two different types of operating systems (OS): Standardized OS or proprietary OS. The latter is called black boxes on handsets. We hereafter refer to the standardized OS as simply the OS and the proprietary OS as the black box.

Originally, all mobile handset's functionality was defined as a black box. With black box-design the functionality is defined at the time the handset is fabricated, and only the producer fully knows about its internal structure and architecture. Basic telephony services are implemented and tested carefully inside the black box, and extra applications like Mp3 players and alike are considered being extra-offered applications within the black box.

As mobile handsets became more and more sophisticated, the black box opened up and became more similar to operating systems, allowing applications outside the black box to utilize the handsets resources. APIs to the resources on the handset was opened, and applications outside the black box were able to run on the handset. However, what kind of services offered for applications not implemented by the manufacturer is highly dependent on the manufacturer and the handset. How the services are offered (i.e. the API) is up to the manufacturer to decide. Such an API can vary from handset to handset from the same manufacturer, and many black boxes do not offer an API directly to the OS services. Developing applications for black boxes calls for specially designed applications for the different mobile handsets, or applications need to be implemented on top of a runtime system. Runtime systems will be addressed in the next section.

In the latest years, more and more handsets have been produced with OS'. OS' are standardized systems that offer a set of services to external applications outside the OS itself. Examples of OS' for handsets are Microsoft Mobile ([7]), Symbian ([8]), Palm OS ([9]), SavaJe OS ([10]), and various Linux implementations ([11]). OS' provide services like device communication, memory operations, file access, networking and GUI's in a standardized and well-known fashion to developers and their applications. These services are accessible through defined API's and application standards.

Common practice is for OS developers to cooperate with mobile handset developers in order to implement OS' on their handsets. Only the handset manufacturer fully knows the physical and low-level software specifications needed by OS developers in order to develop a functional OS for the different handsets. This close connection between OS developers and handset manufacturers have led to some specialized OS' for a manufacturer's handsets such as Symbian S60 series which is specialised for Nokia User Interface ([12]). These specializations concern the UI-layer in OS' for different manufacturers.

On top of specialized OS, the OS' are divided in versions. Every OS have a wide range of versions, and backward compatibility is not always incorporated. For one version of one type of operating system (e.g. Symbian v8.0) there are some standardized API's for basic mobile handset functionality. Extra API's are also available for handsets with extra devices such as WLAN, cameras and so on. The challenge for developers and potential users of applications is the many different OS' and OS-versions in the market. API's are standardized for one OS and versions, but the wide range of both OS' and OS versions requires re-development of applications for different handsets.

At the moment, Symbian and Windows Mobile (Pocket PC and Smartphone) are the biggest actors in the OS marked for mobile handsets. Symbian has implemented their OS for different

Nokia, Sony Ericson, Foma, Lenovo, Motorola, Samsung, Panasonic and BenQ handsets. Windows Mobile exists on some Cingular, HP, i-mate, HTC, Palm, Alltel, Qwest, Samsung, Sprint, Symbol, T-Mobile, Qtec and Verizon handsets. Less widespread OS such as Palm OS mainly exists on Palm handsets. Typical for all handsets with an OS are that they are high-end handsets, often referred to as smart phones. Applications developed for a certain version of an OS can run on all handsets with the same OS version, except when the handset does not offer a technology needed by the application to work properly.

Common for both the black box and standardized OS are software that manage and control both hardware- and software- components in the mobile handset. The common tasks of both types of operating systems are:

- Process management
- Memory management
- File system
- Networking
- Security
- Graphical user interfaces
- Device drivers

Process management is about giving applications the right to execute as a process. The execution of processes can either be done by letting processes execute from beginning to end without interference, or by letting processes execute concurrent. By concurrent execution we mean that processes can run together at the same time. One processor can only run one process at a time. Hence, applications need to share the processor-time between them to do a virtual concurrent execution. This sharing is controlled and managed by the operating system.

Memory management: In addition to processing time, applications need to store information while they are executing as a process. The OS manages the memory and assigns parts of the memory to the running processes.

File system: Managing the information persistently stored on the handset is the OS's responsibility. It controls the placement of files in the storage device (typically flash memory on mobile handsets), and keeps track of where the files can be found.

Networking is a vital part of the mobile handset. A PC normally have one or two network interfaces, e.g. LAN and WLAN, and mobile handsets must have at least a radio receiver and transmitter. In addition, mobile handsets might have several additional network interfaces such as Bluetooth, WLAN, IR and NFC. Handling and setting up the communication according to different protocols is normally an OS responsibility.

Security concerning authentication, storage of secret key and encryption/decryption (i.e. of voice calls and SMS) is not left to the OS on mobile handsets. This is done by the SIM. We will talk about the SIM card in section 2.3.4.

Graphical user interfaces (GUI) interact with the user of a mobile handset by text, images and widgets (i.e. buttons). The design and complexity of the GUI vary a lot from handset to handset, and is highly dependent on the physical size and specifications (like resolution and so on) of the handset-display.

Device drivers are software that allows interaction with hardware devices. The devices are accessible to applications through defined interfaces constituted by the OS.

2.3.2 Runtime Systems

Services that are not offered by the OS or black box can be offered by a runtime system. Runtime systems can also offer a generic API on different platforms such as different OS's or black boxes. A runtime system is software that runs on top on the OS or black box, and is not considered to be a part of these ([13]).

Just as different operating systems on desktop computers offer different API to applications, so do the operating systems on mobile handsets. Runtime systems try to conquer the challenges with different APIs on black boxes and different OS' by adding an extra layer in the software hierarchy. Runtime systems installed on platforms act as middleware to external applications and should by principle offer the same API to these applications, independent of the underlying platform.

The handset manufacturers often develop runtime systems running on top of their black boxes. Both OS developers and independent runtime system developers can develop Runtime systems offered for OS'. External application developers can create applications on top of these runtime systems and make them run on different handsets, OS' and black boxes. Example of such a runtime system are the Java Virtual Machine developed by Sun for the Java programming language ([14]), or Brew (Binary Runtime Environment for Wireless [15]) developed by Qualcomm for wireless-application-development.

Java
Java Virtual Machine (JVM) offered through Java ME (Micro Edition) by Sun ([16]) is the most widespread runtime system on mobile handsets today. OS and black box developers develop the JVM and Java ME-APIs on handsets themselves. This has led to what the industry has called fragmentation of the Java API ([17]). By fragmentation they mean that the APIs are not concurrent on all implementations of JVM, and in some cases the concurrent APIs on different platforms can produce different results when used. A typical example of fragmentation is how user interface APIs are different from handset to handset. E.g. Motorola only supports one text-size on their handsets and an API call to change text size will not give any results.

Brew
Another runtime system offered for mobile handsets is Brew (Binary Runtime Environment for Wireless [15]). This is a more generic runtime system than Java, but the limitations for BREW is its intended use and limited handset support. BREW is mainly developed for game development, and are mainly deployed on handsets in the US marked.

2.3.3 Applications

Applications on mobile handsets can run in many different manners, inside a black box, outside a black box, on top of an OS or on top of a runtime system.

When manufacturers develop their mobile handset with the black box design, they implement a set of applications inside the box. These applications offer basic functionality such as call-, phonebook- and SMS-functions. Black box-handsets with extra devices such as camera and mp3 players do in addition have applications for taking pictures and playing music inside the box. Such applications are placed on the handset in the production line. Independent developers can implement applications (e.g. games) for these handsets, and they use API's offered by the black box for external applications. Independent applications are installed after manufacturing of the handset. In short, applications developed by the manufacturer are placed inside the box, and applications developed by others are placed outside the box.

In OS' there is a clearer cut in the layering between service provider and applications than in black boxes. While applications in black boxes can be placed both inside and outside the box, all applications in OS' are placed on top of the OS. OS developers often implement a set of applications such as call-, file exploring- and messaging-applications as a part of their OS-package, but these are nevertheless not a part of the core-OS. Such applications can be replaced by other applications developed by someone else than OS-developers without tampering with the OS itself.

Since different OS' and black boxes offer different services and API's, developers can choose to develop applications on top of a runtime system. These applications depend on the runtime system, and the runtime system needs to be pre-installed. By definition, runtime-applications are not dependent on the OS or black box on the handset, and should be possible to deploy on all handsets with the runtime system in which applications depend on.

2.3.4 SIM/UICC

While authentication and security is the concern of the OS on desktop computers, the SIM handles security issues like access-control, network authentication, encryption/decryption and storage of secret keys on mobile handsets. The SIM is an application residing on a Smart Card connected to the handset, and this Smart Card is by definition owned by the network operator. All handsets in GSM and WCDMA networks need to have such a Smart Card with a SIM application in order to function as a phone.

Historically this Smart Card only contained the SIM application, and this has led to name "SIM card". Today three types of Smart Card –architectures containing the SIM application are available. To avoid any confusion about different names of the Smart Card on which the SIM application resides, we hereafter refer to the Smart Card with the SIM application as the UICC (Universal Integrated Circuit Card). We will start by taking a look at the physical characteristics of the UICC, UICC OS', and then we examine different architectures for UICCs.

2.3.4.1 Physical characteristics

Smart cards used for SIM applications have both processing and storage capacity since these cards both encrypt/decrypt data and store different types of information. The physical characteristics, communications interface, security commands and commands for interchange for smart card used for UICC are standardized in ISO 7816 [18]. ISO 7816-4 ("Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange") offers a generic and platform-independent interface between the handset and UICC based on APDU commands (see next section).

In the first specification of the UICC, GSM 11.11 [19], the smart card with the SIM application was viewed upon as "one unit", i.e. the smart card only contained the SIM application. The latest standard for smart cards for SIM applications, ETSI TS 131 101 [20], names this smart card the UICC (Universal Integrated Circuit Card). In reality, the physical characteristics of the smart card has stayed the same with two variations, the ID-1 and Plug-in cards. The difference between these two cards is the size. The ID-1 card is the larger one with the size of a credit card, while the plug-in card is of the type of card that is mostly used in modern handsets today (Figure 1).

Up to recently hardware specifications of UICCs concerned their processing power and storage capacity. The storage capacity of UICC ranges from 16KB to the large memory UICC with up to 1GB of storage capacity.

In November 2006 ETSI voted on a new high-speed interface between the handset and UICC. The vote was in favor of the Universal Serial Bus (USB) 2.0 interface. This interface enables high-speed transfer (12Mb/s proposed) of data between the handset and the UICC. This facilitates the use of the UICC as a mass storage device, secure data streaming through the UICC and other secure, bandwidth-consuming connections (such as IP connections with encryption). Since the vote was recently, there are currently no defined standards, only a suggested standard: ETSI TS 102 YYY V0.8.1.

The physical layout of the USB interface on the UICC leaves one pin free on the UICC. It is foreseen that this pin will be used for contact-less support on the UICC. NFC is the coming star of contact-less technology on the market, and has been used for different payment applications such as ticketing and contact-less credit cards ([21]). Some mobile handsets (Samsung [22] and Nokia [23]) have included NFC capability as one of the handsets technology. These have been tested against different NFC solutions. But current solutions include a separate and independent smart card for the NFC. In the future it is expected that the UICC will take the role as the smart card chip for the NFC through the free pin on USB UICC ([24]). If this becomes a reality, this requires application development on the UICC for using the NFC.

2.3.4.2 Handset-UICC interface –APDU commands

Today all communication goes through the UICC-Handset interface based on APDU commands standardized in ISO 7816. APDU commands are built up by a command-response paradigm. Smart cards (such as the UICC) respond to a command APDU from an external host (in this case the handset). The APDU commands take the form shown in Figure 6.

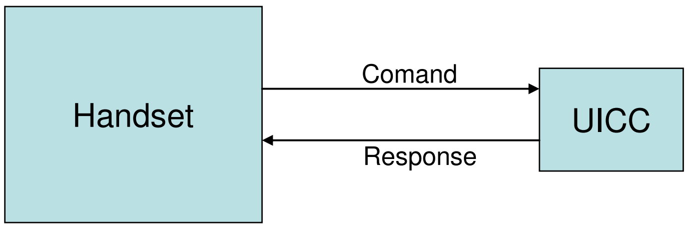

Figure 6 APDU communication flow

Every APDU command (Figure 7) has a mandatory header consisting of a class field (CLA), instruction field (INS) and two argument fields (P1 and P2). The class and instruction fields tells the UICC which action to take similar to function invoking in computer programming, while the argument fields (P1 and P2) are similar to a function arguments in computer programming.

Command
APDU

Mandatory header				Optional body		
CLA	INS	P1	P2	LC	Data Field	Le

Figure 7: APDU command

In addition, the command can contain some data for the smart card. The length of this data field is given in the Lc field. At last, the command APDU can contain the Le field, specifying the size of the expected response from the smart card.

As a response to the command APDU, the smart card must at least respond with the SW1 and SW2 fields (Figure 8). These fields tell the host how the execution of the command went. If the smart card has some data to send to the host, this is added in the beginning of the response APDU.

Response
APDU

Optional body	Mandatory trailer	
Data field	SW1	SW2

Figure 8: APDU response

Since there can be several applications on one smart card, a mechanism for giving the APDU to the correct smart card-application is needed. This is done by the standardized SELECT APDU command (Figure 9). The SELECT command, and other basic commands such as INSTALL and DELETE (install and delete of applications), is standardized in GlobalPlatform Card Specification V2.1.1 [25].

CLA	**INS**	**P1**	**P2**	**LC**	**Data Field**
0x00	0xA4	0x00	0x00	0x05	AID of application

Figure 9: SELECT command

After an application is selected, all commands not concurrent to the GlobalPlatform commands are given to the selected application. It is up to the application to accept or discard the incoming APDU command.

2.3.4.3 UICC OS

Smart cards have the need for operating systems just as computers do. There are two possibilities for OS on UICC: Native ([26]) and Java Card OS ([27]). Native OS for smart cards are similar to Black Box OS on handsets; they are tailored for a specific smart card. Applications developed for a native smart card operating system needs to be tailored depending on the specific smart card.

The Java Card technology from Sun makes it possible to write small Java applications (called cardlets) on smart cards such as UICC. Java Card technology on smart cards is basically a small operating system for smart cards with a Java Virtual Machine incorporated. ETSI have defined a standard for the (U)SIM API for Java Card (ETSI TS 102 241 [28]), and UICC with the Java Card technology is now in use on mobile handsets ([29]).

Java Card Technology has mainly two advantages contra the native operating system. First, applications for the UICC can be implemented in an easy and known high-level language, namely Java. Second, the Java Card platform are an interoperable execution platform, hence applications are implemented for Java Cards can run on all other Java Card smart cards independent of the card manufacturer.

2.3.4.4 UICC standards

The fact that the UICC is a smart card makes it a safe storage medium with authentication, and it is able to encrypt/decrypt it's communication.

There are mainly three software architectures for UICC and the SIM application ([30]). The first UICC is known as the 2G or GSM 11.11 SIM ([19]), the commonly used UICC today is known as the 2G Phase2+ or GSM 11.14 SIM standard ([31]), and the emerging UICC is known as 3G SIM or USIM ([32]).

Independent of the UICC standards, the basic SIM functionality and handset interface is incorporated in all standards. This is the Subscriber Identity Module functionality where the SIM works as the subscriber's entity in the network, and performs identification, authentication and encryption/decryption for the user and handheld handset in the network. The basic functionality is defined in GSM 11.11. The interface between the UICC and handset is based on APDU commands standardized in ISO 7816-4.

2G SIM –GSM 11.11

The first UICC standard was the GSM 11.11 standard, and defined the basic SIM functionality on a smart card: The Subscriber Identity Module.

Every SIM application contains a unique identity number (IMSI –international mobile subscriber identity) used to identify the user within the GSM core network. This identity number tells the network what phone number is mapped to this SIM (and the mobile handset), and what company is the operator for the SIM's user. But this identification number does not support any security by itself. IMSI numbers can be copied and frauds could potentially abuse other users subscriptions. In order for the mobile handset to authenticate itself in the network, a secret key is used. This key is stored on the SIM, and a copy of the key resides in the AuC (Authentication Centre) of the HLR (Home Location Register). By encrypting/decrypting with this key, both the mobile handset (through the SIM) and the network can be certain that they communicate with the intended entity in the network. This authentication is done every time the mobile handset connects to the network (e.g. the handset is turned on).

Since the communication between the connected base station and the handset is wireless, this data can be picked up by anyone who for some reason wants to eavesdrop this data. The solution to prevent abuse of this data, the SIM encrypts the communication between the handset and the base station. A temporary key generated by a random number on the SIM application does this on one side, and the subscriber's home network generates the same key and gives it to the base station on the other side of the link.

In addition to the authentication against the network, the SIM application has a personal identification number (PIN) code for authentication against the user of the handset. This is to prevent abuse of the SIM application from the user side.

Without the SIM a handset is not longer a phone, just another small computer. Most mobile handsets emphasise this by requesting a SIM and refusing to turn itself on when no SIM is present. The mobile handset is a symbiosis of he wireless phone-hardware and the SIM.

In the GSM 11.14 standard the handset had complete control over the UICC, and the UICC only answered the handsets commands. This standard is a part of the second generation GSM or 2G (The first standard, 1G, was the analogue wireless mobile phone e.g. NMT, with no SIM at all). The UICC was the slave of the handset. The architecture of the GSM 11.11 solution is based one single application, and its shown in Figure 10.

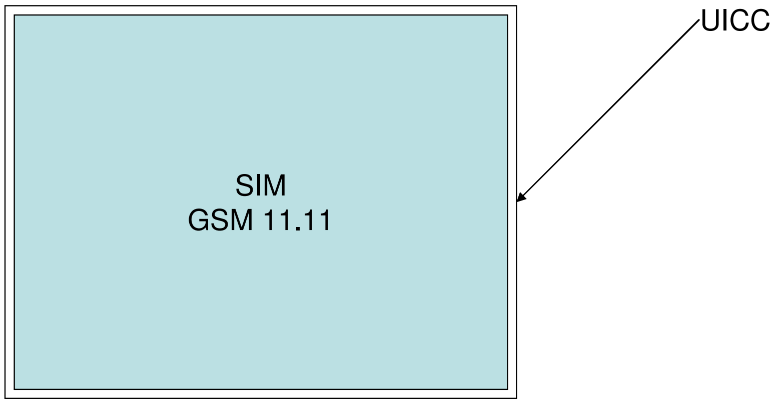

Figure 10: 2G UICC

2G SIM –GSM 11.14

Since the UICC is the property of the network operators and is the only generic part of the mobile phone, the industry found it feasible to give the UICC some power over the handset. The solution was the SIM Application Toolkit (SAT) defined by the GSM 11.14 standard. The UICC was still a smart card and could not initiate the communication, but by adding a flag to the UICC’s responses on handset commands, the UICC could tell the handset that it had commands it would like the handset to perform. The UICC was no longer a slave of the handset, and could now take control over the handset. This standard is referred to as the Phase 2+ SIM (enhancement of 2G).

Smart cards with SAT has basically two applications, the basic SIM application defined by GSM 11.11 and the SIM Application Toolkit (SAT) application defined by GSM 11.14. The architecture is shown in Figure 11. The SAT makes it possible for operators to install simple custom services they want to provide. These services are by definition not independent applications per se, but rather an extension to the SAT application.

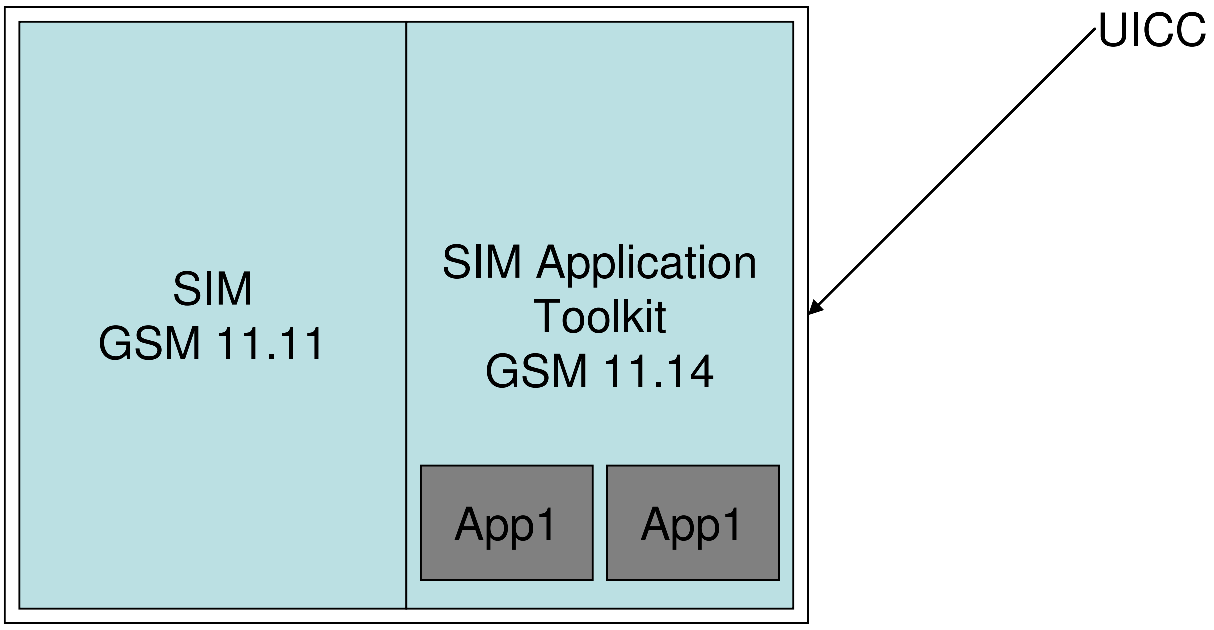

Figure 11: 2G Phase + UICC

Commands that the SAT can give to the handset are typically to set up calls, send SMS, display menu and text etc. The recent and full command-set can be viewed in the GSM 11 14 document ([31]). SAT applications are a set of these commands, and are included as internal applications in the SAT.

The GSM 11.11 and GSM 11.14 application on the smart card is in need of high degree of security. Authentication is a part of the smart card standard, and prevents unwanted access to applications. To further enhance the security, OS and black box developers have made it impossible to reach and use the UICC without proper signature of applications. Access to the UICC is only achieved through SAT APIs and the signature. This signature is only given to handset producers and network operators, and no other developers can get in direct contact with the SIM on the handset.

3G SIM –USIM and USAT

A new UICC standard followed with the arrival of the third generation mobile network (3G [33]).

This new UICC standard is called USIM and is basically an independent SIM (according to the 3GPP 31 111 standard [34]) application on a Universal Integrated Circuit Card (UICC [32]), and the UICC can contain other independent applications. With this arrival the SIM-card (so far commonly thought of as a hardware device) became a multi purpose application platform.

The new USIM standard defined a new SIM application applicable to 3G phones and networks, and at the same time dividing the SIM application from the hardware of the UICC. Recall from SIM's with GSM 11.11 where the smart card only contained one application, namely the SIM application, hence the name SIM-card. With the arrival of GSM 11.14 we had two applications, where one of the applications (GSM 11.14) could be injected with new code to act as an application within the application. The smart card was not reachable for applications not implemented by the OS or handset manufacturers. With the arrival of the USIM standard, the UICC is more similar to a "normal" smart card, and the UICC is reachable from other applications outside the black box or OS (e.g. through J2ME and the JVM).

In addition to the USIM application and independent applications, the UICC contain the USIM Application Toolkit (USAT GSM 31.111). This standard is similar to the SIM Application Toolkit. The architecture of the 3G SIM is shown in Figure 12.

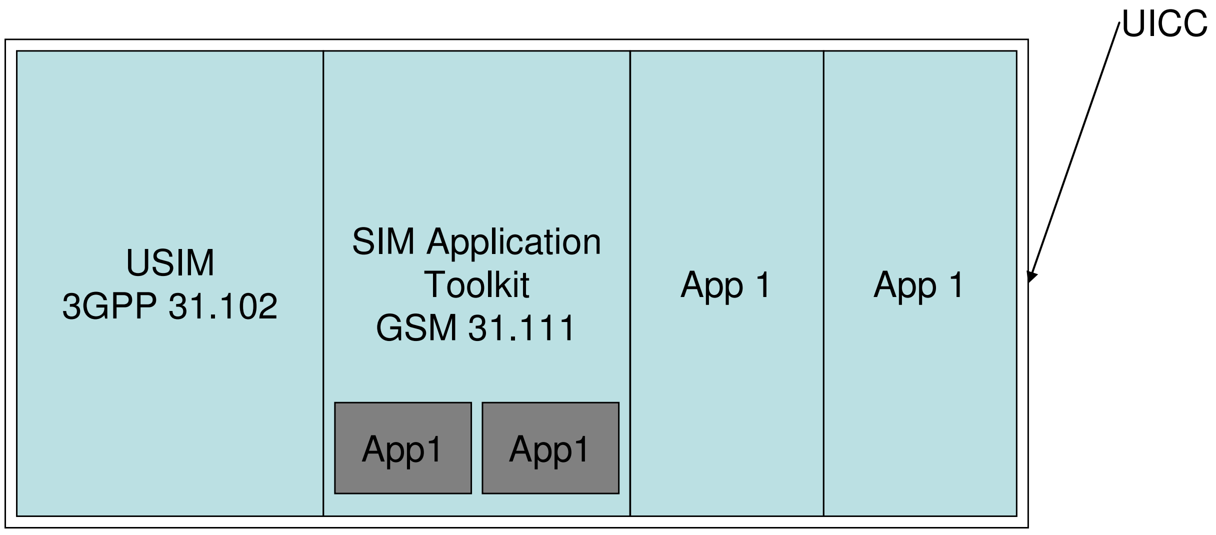

Figure 12: 3G UICC

2.3.5 Summary

The mobile handset at least consists of radio receiver/transmitter, SIM card, display and an input device. Handset manufactures provide the handset with either a proprietary OS commonly called a black box, or with a standardized OS. Applications can be built on top of the black box and OS, or a runtime environment can offer a more generic platform for applications. The SIM application on the UICC identifies and secures the handset both locally and in the network. In addition, the UICC can contain applications within the (U)SIM Application Toolkit, or as independent Smart Card applications on 3G UICCs. (U)SAT applications are only reachable for network operators through (U)SAT, while independent applications can be reachable from other applications outside the OS or black box.

As mobile handsets are becoming more sophisticated, service provisioning through handset applications is becoming more and more popular amongst actors other than the handset manufactures. The diversity in both handset manufactures and their standards, technology on handsets and wide range of application developers creates certain challenges for the users of handsets. These challenges will be addressed in the next section.

2.4 Services on mobile handsets

Oral communication and text messaging has been provided services on handsets since the beginning of the GSM period. But providing new services on handsets are an emerging marked as handsets are becoming more sophisticated and contain more communication technologies suited for transactions. A service is deployed on the handset as an application, and for the application to be able to do a transactions, it needs to communicate through communication technologies on the handset.

There is a wide spectre of communication technologies used by services for transactions on handsets. Most common in today's marked is the SMS technology. Example of services that utilize SMS as their communication technology is MobilHandel [35] and SBB MobileTicket [36].

Technologies like NFC (Near Field Communication [37]) and Zigbee ([38]) on handsets are enabling more services such as Visa and Mastercard on the handset (Visa card on handsets through the EMV standard and NFC [39]) and more sophisticated ticketing systems (e.g. NFC [40] and Zigbee [41]).

Through history, new technology and service enablers have shown that a wide range of application- and technology-standards emerge. This poses many challenges for users of the services, as they have to navigate through a forest of applications fitted for their desired service. In the next section we have exemplified how the wide range of technology and application standards create problems and complicated use of services for users. The following section specifies what part of the handset and application architecture that create these challenges.

2.4.1 Example

> *Note: The example is based on solutions we expect to see in the future, based on the technology we know are currently tested out on proto-type and high-end handsets. Names on network operators and services are fictional, except the EMV standard for NFC. Even so, we know that there is being executed trials on different services similar to the ones we have mentioned in the example (e.g. Tromsbuss etc[21])*

John, our high-end handset user, has just bought the new Nokia 6131 NFC handset. At the same time he purchased some services through his network operator, GBHighEndMobile. These services include:

- London Metro Ticketing System
    - Uses internet access to purchase tickets
- VISA mobile credit/debit card
    - NFC enabled credit/debit card according to the EMV standard
- InFront hotels key-less solution
    - NFC enabled room-key solution
- Heathrow's new check-in system
    - NFC enabled service that does not need paper-tickets or identification papers.

His company has ordered John to go to Tokyo and meet some new business contacts. As he arrives at his local metro station, he logs in to London Metros web pages through his browser with GPRS connection and purchases a single ticket. He receives a SMS as a receipt for ticket control. At Heathrow Airport he sweeps his new Nokia over one of the tickets stands in the departure terminal, confirms his PIN code, and his ticket is brought from its safe storage on the NFC chip and given to the ticketing system. He checks in a suitcase on his handset, and heads for the gate. At the gate he sweeps his handset again over the gate-ticketing stand to confirm that he is entering the plane. His PIN code and encryption key inside the NFC chip confirms that he is actually John.

So far his trip has been without any trouble and confusion, and he has only used his handset as payment, ID and ticket. When he arrives in Tokyo, he realises that he has forgot his valet with the old fashion Visa Credit/Debit card. Luckily for him, Tokyo can offer the same services as London for bus tickets, the NFC EMV standard and wireless check-in at Narita airport.

Before John can get on the bus, he needs to download a new application to his handset from the bus company in Tokyo. To buy tickets, he needs to fill up his bus-ticket-account by doing a payment according to the Japanese mobile payment standard. This requires another application. Since John is used to new technology, he manages to set up and use these applications within half an hour. As he tries to buy a ticket, he discovers that the application for ticketing is based on the Zigbee technology, where the handset connects to the WPAN network ([42]) on the bus and buys a ticket. Luckily, the bus company has implemented another back-up-application mainly aimed at foreigners (since Zigbee handsets are mainly sold in Asia) that uses standard Bluetooth technology. Arriving at his hotel he registers his VISA card through the NFC reader. Instead of a key to the room, the receptionist asks for his mobile number and says that he can use his NFC enabled handset as the key to the door. John is used to this when he stays at InFront 's hotels in the US, and walks to his room. When he tries to open his room-door with the handset, a red blink tells him that he does not have access. Back at the reception desk, the receptionist tells him that he is newly employed, and should have known that the key-less solution is only applicable to JapanTelecom's customers. He receives an apology and an old fashion key instead.

After a dinner at a nice restaurant, John runs back to the hotel in pouring rain. Suddenly he hears a crashing sound, and to his despair he notices that his handset fell to the ground and into a water pit. The handset is destroyed, and he takes out his SIM card and walks into a handset retailer and buys a new handset. This time he buys a Samsung X700 with NFC. Back at the hotel-room he calls GBHighEndMobile and asks them what to do with all his services. Since he has forgotten his wallet, he needs to get the NFC credit card in order. VISA is a big global actor in the marked, and they have implemented their applications for all known handsets with NFC in the marked. The application can then easily be provided for the new Samsung X700 with Over The Air (OTA) provisioning. GBHighEndMobile does not have any known implementations for the other services for the Samsung handset since it is only sold in the Asian region. GBHighEndMobile suggests that John buys a new Nokia 6131 NFC from them when he returns to London.

John pays in cash for the rest of the trip with money he have retrieved from EMV ATMs, and identifies himself with his passport at all steps on the trip back. Back in London he has to buy a new Nokia 6131 NFC. When he has bought the new handset, GBHighEndMobile transfers the applications that he needs to get his services up and running to the handset. Because John had his London Metro Ticket account information on his NFC smart card chip, he needs to block he old NFC chip id, and fill the new NFC chip with ticketing credit. He had also bought some tickets from an airline for a vacation in his free time, but the electronic NFC ticket was lost with the old handset. The airline charges John with a 10$ fee for transferring the ticket to the new NFC chip.

Names on network operators and services are fictional, except the EMV Credit/Debit card solution for NFC. The handset mentioned are manufactured and used for prototyping, but NFC handsets are not far from the mass marked. Still, these solutions are possible to realize with new technology and standards in the marked today, and it is reason to believe similar solutions will be available for users in the near future. The example is given to show some challenges new technology and services give to service providers and users. We will focus on four categories of challenges that we have defined through this example:

- Lack of interoperability amongst handsets
- Different technologies within same type of services
- Different application standards within same type of services
- Loss of state when users change handsets

2.4.2 Lack of interoperability amongst handsets

John from our example experienced challenges with the lack of interoperability when he wanted the services he had on the old handset to be usable on the new handset. Because the Samsung X700 has a different black box than the Nokia 6131 NFC, and GBHighEndMobile had no implementations for the services for the Samsung handset, John had to buy a new Nokia 6131 NFC to use his wanted services.

Different OS' and black boxes offer different APIs for developers to handle. When a developer designs an application he or she needs to consider what handsets the application should run on. As long as the application are to run on a Windows Mobile or Symbian platform of the same version, one implementation of the application is needed. On the other hand, if it should run on different versions of Symbian, or both Windows Mobile, Symbian and a couple of black box design, many individual implementations of the same application is needed. This can be a major headache for developers as knowledge about all APIs are needed, and in some cases different programming languages. C# and the .Net platform is for example the preferred programming platform for Windows Mobile platforms, while on Symbian C++ with a Symbian STK is needed ([43]).

To conquer these problems with different APIs and preferred programming languages, the mobile industry use in great extent the Java platform and API for mobile applications. As mentioned this should by principle solve the incompatibility issues for developers surrounding APIs, but this is far from the real world today. Sun, the owners of the Java technology, have themselves stated that API fragmentation on mobile handsets can result in "400 or more different executables for a given title" ([17]). Thus, Java simplifies the development of applications by offering one programming language and in to some extent a generic API, but is far from as straightforward as it should be by principle.

Various APIs and API fragmentation is clearly a disadvantage for developers since they need to use a lot of time on something that should be straightforward. But this also causes challenges for application vendors since more time and money is needed for development of applications. In addition potential users might find it confusing when they retrieve new applications. They have to ask the question "Do my phone support this application?" and in many cases "Why doesn't it work on my phone?"

These often confusing challenges make it more time consuming and expensive for developers and vendors, and slow down the time to market for applications. After the applications have hit the market, users are often sceptical to new applications not developed by the handset manufacturers, and many users find that their handset does not support a given application.

2.4.3 Different technologies within the same type of services

As John came to Tokyo he discovered that companies offered the same services as he was used to in London. But some of these services were based on different types of technology. While big actors like Europay, Mastercard and VISA can create a global technology standard for their solution (such as their EMV for IC cards and NFC), smaller and independent actors like the London Metro and Tokyo busses will use the technology they find best suited. Their decision can be based on different factors such as price, types of handsets their local user typically have, and knowledge about technologies amongst decision makers.

Since only radio communication and SIM cards are standardized on mobile handsets, these devices have a wide range and varying degree of technologies and abilities that applications can make use of. What kind of technology an application can use depends on the application. E.g. an application that needs a connection to Internet can use GPRS, WLAN or Bluetooth to establish such a connection. Another application for ticketing on busses might make use of Zigbee, Bluetooth, Internet connection and NFC, or even a combination of these (e.g. Internet connection for buying tickets, and NFC for using them). A given handset might only support Bluetooth and GPRS connection, and an application has to be tailored for this handset.

But its not only when the handsets have varying technologies included that the applications need to adapt: The surrounding environment can offer different technologies on different locations. Mobile handsets are by definition mobile, and as they move around, the surroundings change and might offer varying technologies. When John came to Tokyo, the bus tickets were based on a different technology than in London. Hence the technology on the server side was one hinder for John that made his application and service from London useless in Tokyo.

2.4.4 Different application standards within same type of services

The third challenge is about different application standards within the same group of services, and it has similarities with both the first and second challenge. When John tried to unlock his room door at the hotel, the door wouldn't open. He had already bought the service from InFront hotels where he could use NFC on the handset as a key, and the same type of solution was implemented at his hotel in Tokyo. But his hotel in Tokyo did not have any collaboration with InFront on the NFC key project, and the solutions had different application standards.

Application standards are up to the developers in projects. Such standards can be solution specific standards such as communication protocols, algorithms for the application etc. When big actors such as Euoropay, Mastercard and VISA collaborate on a project, they have more or less hegemony within the global payment-marked. As they choose to collaborate, they create a global application standard that will be followed by service providers and users.

Services for less monopolistic markets than the payment marked are more likely to suffer from the lack of global application standards. There are many reasons for this; one being lack of knowledge about other solutions developed or under development by known actors, another being actors that are unaware of each other, thus their solution, and thirdly that actors knowingly choose to create new and different application standards as a differentiator in a competitive marked. Different application standards require a new implementation for every standard, either as an add-on to already implemented application, or as a new and independent application.

2.4.5 Loss of state when users change handsets

According to a survey by Telenor in spring 2006 amongst their customers in the Norwegian marked, the average lifetime of a handset is 13,4 months[1]. Handset manufacturers are obviously pleased with this rapid replacement of handsets. Application developers on the other hand can have a lot of headache around these rapid transitions, the reasons being the lack of interoperability amongst handsets and the varying types of technology on handsets. Tailoring of applications to fit different handsets and their API's and technology is a time-consuming process, and the output is many different versions of one application. Every application version can run on one handset or a group of handsets.

In addition to giving the application-developers a lot of extra work and creating an opaque versioning system, users need to download and re-install a new version of an application they had on the old handset. When users re-install applications on new handset, user and application state can be lost. This is not an unknown issue for PC users: When we buy a new computer, we normally re-install applications on the new PC. But state can be brought from one computer to the next since it is likely that both the new and the old computer have the same type of storage medium (e.g. CD-ROM, DVD-ROM or USB interface), they both have the same file system (e.g. variations of Windows or Linux file-systems).

On handsets it is not as simple to bring state from one device to another. Handsets do not have any standardized storage medium such as CD-ROM and DVD-ROM, and handsets with black boxes have proprietary file-systems. Handsets can have operating systems with standardized file-systems, but we earlier talked about five different OS' for handsets, hence over twice as many as for PC. Handsets can also have memory cards, but there are many different types of memory cards on different handsets (e.g. SD Mini, RS MMC, Transflash, Memory Stick PRO Duo etc. [44]). The only standardized storage medium is the SIM-card, but it has very limited storage capacity, typically 128KB. Still, the SIM is used to store the contact list and messages, so these can be kept when handsets are changed.

The result of this jungle of black boxes, OS' and storage mediums, is that state is very often lost when people change handsets. John experienced this too when he lost his old Nokia and needed to buy a new Nokia when he came back to London. Since the metro credit and flight tickets were information that needed safe storage, they where stored on the NFC chip inside the old handset. This state was lost together with the old handset. John needed to buy new metro credits, and the airline transferred the flight tickets to the new handset, charging John 10£ for the service. His high-score from Solitaire was lost forever. For John this lost state was both annoying (ref: solitaire high-score) and expensive.

2.4.6 Summary

The mobile handset industry involves a wide range of actors all the way from handset production, software development, service providers and network operators. Lack of collaboration between different actors has led to some challenges that we addressed in this section (2.3). These challenges include:

- Lack of interoperability amongst handsets
- Different technologies within same type of services
- Different application standards within same type of services
- Loss of state when users change handsets

[1] Personal communication with Gunnvald Svendsen, Senior Researcher, Telenor R&I

In the next section we will sketch up different architecture proposals that can facilitate faster development, easier deployment and wider use of services on handsets.

2.5 Existing initiatives and standards

The challenges surrounding services we sketched up in the previous chapter are well known challenges for handset manufacturers, network operators, service providers, and OS and application developers. These challenges are caused by standards, either by the lack of standards (e.g. application standards and technology standards), the wide range of standards (e.g. the large number of OS' and OS-versions and spectre of communication technologies), or the unwillingness to follow standard (e.g. Java ME implementations on different handsets). A lot of effort is put both in standardization and ways to get around challenges surrounding standards. Actors within these areas are associations with members from handset manufacturers, network operators, service providers etc., and platform developers such as Microsoft and Sun. In this section we will take a look at what efforts are made to conquer the challenges we stated in section 2.4 amongst these actors.

2.5.1 Associations

To create a common ground of collaboration, many associations have been founded to create, suggest and front standards for the mobile handset industry. Their motivation is to increase the use of equipment and services within the mobile handset marked, and thereby increasing their members revenue.

The best-known and largest associations in Europe are ETSI (European Telecommunications Standard Institute [45]) and GSMA (GSM Association [46]). ETSI is a non-profit organization whose responsibility is to standardize information and communication technologies within Europe. The organization has 655 members from handset manufacturers, network operators, services providers, researcher bodies and even users. GSMA is a common interest group consisting of 700 members that has as a main goal to: "…ensure mobile phones and wireless services work globally and are easily accessible…". ETSI and GSMA focus primarily on telecommunication technology and standards, e.g. SIM applications and GPRS communication. Other associations such as OMA (Open Mobile Alliance [47]) and OMTP (Open Mobile Terminal Platform [48]) focus on service and handsets interoperability, and they are not standardization institutions: They front requirements their members have surrounding handsets and services through service enablers and technical specifications.

OMA and OMTP have both come up with suggested solutions that are supposed to conquer the challenges regarding lack of interoperability amongst handsets, different technologies within the same type of service and different application standards within the same type of service. ETSI has created a standard that can be used for the same purpose.

OMTP –Technology Agnostic Software Platform

OMTP has come up with a concept-solution they call Technology Agnostic Software Platform (TASP, [49], the architecture is shown in Figure 13).

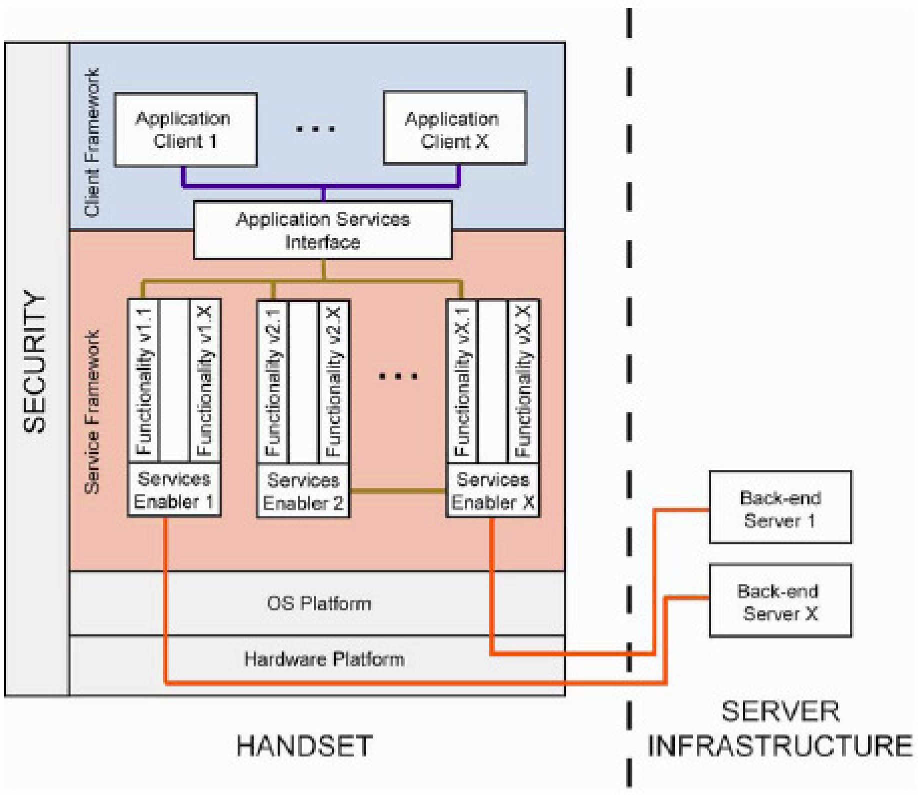

Figure 13 OMTP's Technology Agnostic software (from [49])

Applications run on top of a service framework in OMTP's architecture. This service framework offers access to service enablers the Application Service Interface in a generic manner that is technology agnostic. Service enablers are services such as device management, messaging etc. Applications are focused around user interaction, and leave communication with services to the Service Framework. The service framework is updated through a set of servers, and every service is a service enabler with some functionality. How these service enablers act is a bit vague in [49], and this concept is not a standard, only a suggested framework that the OMTP members want the handset actors to follow.

OMTP's concept-solution, if implemented, will solve the interoperability challenge amongst handset by offering a generic Application Service Interface usable for different technologies and application platforms. It might also solve the challenge with different application standards within the same type of service if the server infrastructure is able to replace service enablers dynamically to adapt to changing application standards. This is however not specified in [49], but should be possible to realize through this solution. The server infrastructure might also adapt to technology change in the same manner. This solution does not solve the problem with loss of state when users change handset.

OMA Service Environment

OMA's concept-solution called OMA Service Environment (OSE, [50]) is very similar to OMTP's solution with an extra layer between applications and the handset platform that offer service enablers (Figure 14).

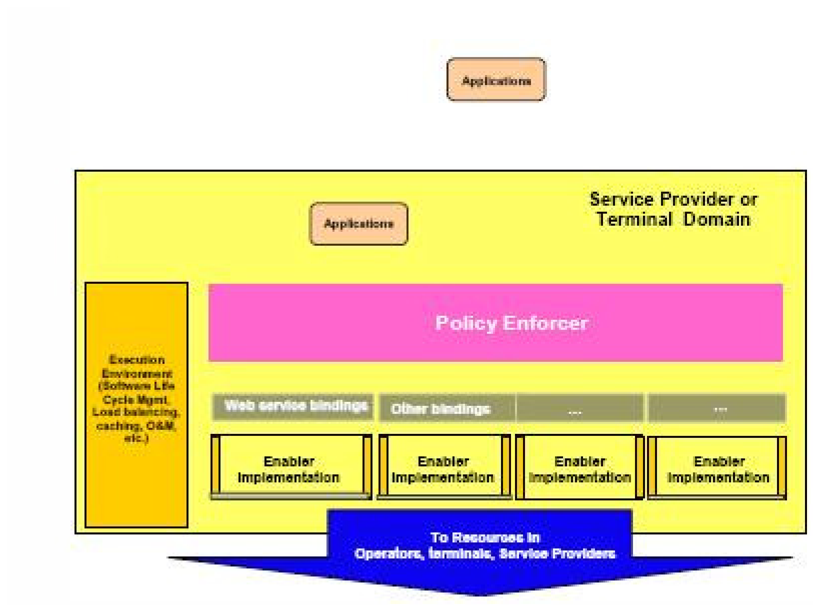
Applications
Service Provider or Terminal Domain
Applications
Execution Environment (Software Life Cycle Mgmt, Load balancing, caching, O&M, etc.)
Policy Enforcer
Web service bindings
Other bindings
...
...
Enabler Implementation
Enabler Implementation
Enabler Implementation
Enabler Implementation
To Resources in Operators, terminals, Service Providers

Figure 14 OMA Service Environment (from [50])

The OMA architecture does not include any external servers that update their enablers, and they are a bit more specific about how this solution should be implemented. They require that applications that run on top of their service environment should be possible to implement in any programming language, and that the interface should be implemented in an interoperable manner. Enabler implementations should provide standardized functions that offer some kind of service to applications, e.g. authentication, messaging and location management. If a service enabler has multiple entities (e.g. multiple servers), each enabler is implemented separately.

As the OMTP solution, this solution does also solve the interoperable challenge. In addition it specifies that a service might have separate entities. It is possible to interpret that these entities should be able to handle both different application and technology standards for a single service, where each entity implements a service with a given technology and application standard. But as for OMTP's solution this does not solve the challenge with loss of state when changing handset.

ETSI –Card Application Toolkit

A standard that surfaced in 3G mobile networks was the Card Application Toolkit (CAT, ETSI TS 102.223 [51]). This toolkit is almost identical to the USAT (section 2.3.4.4), but with the difference that it is intended for the "independent" applications on the UICC. By "independent", we mean applications not implemented under the USAT platform, and third-party developers can implement and access these applications as long as they know the proper signature.

The CAT offers services such as communication interfaces and simple UI on the handset for independent UICC applications. The communication interfaces covered by the CAT standard range from simple SMS' to Bluetooth, and in addition this standard is expandable to cover new network technologies that might emerge. The intention of CAT is to offer all communication technologies present on the handset to the UICC applications.

Even though this is a standardized feature, we have been unable to identify handset manufactures and OS developers that have implemented this standard today. The latest CAT standard was released in February 2007, and it is hard to tell when and if this standard will be implemented on handsets. Sadly, even though a standard has arrived, handset manufactures and OS developers are not forced to follow this standard.

If this standard is implemented on handsets, it can solve the interoperable-challenge since all handsets, regardless of software platform, will follow the CAT standard. And because the applications are placed on a UICC, it will be possible to bring applications, thus state, from handset to handset. But this solution does not implicitly solve the problem with different technology and application standards within a single type of service.

2.5.2 Platform developers

While associations seek to increase the revenue of their many members through standardization and collaboration, platform developers seek to increase their independent revenue by offering a platform that is applicable to a large number of handsets and is easy to use both for developers and users.

Sun –Java ME Open Source
Sun is the best-known provider of a developing platform for mobile handsets through Java ME, and their platform is supposed to be interoperable across OS platforms by providing a runtime system on top of OS'. But as we stated earlier, Java ME has encountered problems with fragmentation in the implementation on different handsets.

To conquer this fragmentation of Java ME, Sun decided to release the source code of Java ME in the end of 2006. One of the reasons Sun did this was to reduce the fragmentation of Java ME on different handsets ([52]). Reducing fragmentation in the Java ME platform will lead to more interoperable applications on handsets.

Microsoft –Connection Monitor Application Block
Microsoft develops their own OS, and does not front interoperability across platforms such as Sun. But they have taken measures to handle the challenges around different communication technology standards within a single type of service.

Microsoft released in 2006 something they called Mobile Client Software Factory (MCSF [53]). This is a development package for the Microsoft Windows Mobile Platform 5.0 that amongst other things contains the Connection Monitor Application Block (CMAB). This CMAB is a connection tool that seamlessly keeps a connection to Internet-services for an application. Handsets have several ways of connecting to Internet, e.g. GPRS, EDGE, WLAN, and CMAB tries to keep a connection to one of the possible Internet available interfaces according to some priority (priorities through pricing) list. By offering this tool through their platform, they reduce the developing cost for Internet connections, and help the user to stay connected to an Internet service as long as some means of connection is available.

2.5.3 Discussion of existing initiatives and standards

We have presented five related works that can solve the challenges we stated in chapter 2.4. OMTP's and OMA's concept solutions solve the interoperability problem, and their system can also be used to solve the challenge with different technology and application standard within a single service. CAT is a standard from ETSI that offer a service such as communication interfaces and UI for applications on an UICC. Applications on the UICC solve the interoperability challenge since all handsets must support a generic interface to

UICC applications. In addition, these applications can be brought from handset to handset, solving the challenge surrounding preserving of state.

Sun and Microsoft have presented two solutions, respectively the Java ME Open source and CMAB. Sun and Microsoft have presented these solutions to enhance their own position in the mobile marked, and does not front collaboration. Still, Java ME Open Source could increase the interoperability within the Java ME platform that is available on most handsets today. Microsoft has acknowledged that the Internet service can be obtained through different communication technologies, and by offering the CMAB they try to seamlessly offer the Internet service independent of which communication technology that is suitable any given time.

	OMTP's TASP	OMA's OSE	ETSI CAT	Java ME Open Source	Microsoft CMAB
Interoperability	**I**	**I**	**I**	**P**	**N**
Technology	**O**	**O**	**N**	**N**	**P**
Application standard	**O**	**O**	**N**	**N**	**N**
Preserving State	**N**	**N**	**I**	**N**	**N**

I = Challenge which the suggestion/standard/tool is designed to solve
O = Opportunity for solving problem through the suggestion/standard/tool
P = Partially solving a problem through suggestion/standard/tool
N = Challenge that is not affected by the suggestion/standard/tool

Table 1:Comparison of standards and initiatives

In Table 1 we have listed the five related works and put them up against the four challenges we have identified. We have divided the affect the related works have on the challenges in four groups: I, O, P and N. OMTP's and OMA's solution can solve three out of four challenges, CAT solves two challenges while Java ME and CMAB partially solve two challenges. We say that they partially solve the challenges because these solutions are platform-dependent.

If we examine Table 1 we see that the five solutions combined solve all four challenges regarding services on handsets. But of the five solutions we only know of one that is implemented on some handsets: The Microsoft CMAB solution. This is in addition only offered on handsets with the Microsoft Mobile platform. Java ME Open Source is an ongoing project that with time might lead to more interoperability within the Java ME platform. TASP and OSE are only concept solutions with no current implementation or standard. CAT on the other hand is a standard, but the authors are unaware of any handset with this solution implemented.

2.6 Summary

In this chapter we started by taking a look at the basics of the mobile handset, both its basic features and additional, complementary technologies. We further examined the application platforms available on handsets, and revealed some challenges developers and users of applications experience when it comes to services on handsets. At the end of this chapter we took a look at some initiatives from other actors in order to handle some of these challenges. In the next chapter we will state a goal, some secondary objectives related to this goal, and some requirements to reach the goal, in order to design and implement an architecture to handle the challenges we have identified surrounding services on mobile handsets.

3 Requirements

3.1 Goal

The problem statement of this thesis stated that it is important to understand how we can build applications and clients in a more platform agnostic approach, and how we could naturally divide platform, protocol and service specific functionality on the handset. We exemplified some of the challenges these issues raise in chapter 2. Our goal is:

> *To design an architecture that facilitates more widespread development and use of services on mobile phones.*

To specify, we divide this goal in four secondary objectives.

3.1.1 Platform interoperability

Find an architecture that facilitates cross-platform development and use of services (interoperable services), thereby making it possible for applications to run on different platforms without re-development of services.

3.1.2 Communication technology standard

Since different communication technologies are applicable to the same service we will try to find an architecture where services can dynamically adapt to varying external communication technology (i.e. technology change through changing surroundings of the handset) and internal communication technology (i.e. technology changing through handset change).

3.1.3 Application standard

We will find an architecture that by some mechanism is able to adapt to varying application standards within the same type of service (i.e. different ticketing services).

3.1.4 State

We will find and architecture that maintain state about users and services as users change their handsets, enabling users to keep their services even if their change their handset.

3.2 Requirements

In order to reach our goal and secondary objectives, we will state some requirements to the outputted architecture in this thesis. The requirements reflect the secondary objectives from the last section.

3.2.1 Platform interoperability

In order for the architecture to be able to facilitate cross-platform development and use of services, applications need to be executable across handsets. Hence:

> **R1.1:** The whole or parts of a service should be placed on an interoperable platform connected to the handset.
>
> **R1.2:** The interoperable platform should be transparent to the user.
>
> **R1.3:** The interoperable platform should support at least 5 handsets with different OS'

3.2.2 Communication technology standard

The applications residing on the interoperable platform need to access the communication technologies available on the handset, hence

> **R2.1:** All communication technologies on the handset should be accessible to applications on the interoperable platform through a generic interface.

Since different external communication technologies might offer the same operation for a service, we need a mechanism that changes the technology in use as the surroundings change. We get the requirement:

> ***R2.2:*** *If any known pair of internal and external communication technology can offer an obtained service, this should be made available transparent to the user.*

3.2.3 Application standard

For the architecture to be able to adapt to application standards, application standards have to be replaceable and dynamically updated on the interoperable platform. We require that:

> ***R3.1:*** *Application standards implemented in the applications on the interoperable platform should be transparently replaced on demand from a remote service.*

3.2.4 State

Users change their handsets often, and state surrounding services need to be preserved in these transactions. Hence:

> ***R4.1:***
> *State about applications and the user should be possible to bring from handset to handset without overhead administration from the user-side.*

4 Design

In this chapter we will show a design that meets the requirements from chapter 3. In chapter 2 we found some challenges surrounding services on mobile handsets. These challenges are known for actors in the industry, and in section 2.5 we saw some work that has been done to handle these challenges. We will design a system that meets the requirements from chapter 3, and in this design we will use elements from section 2.5.

4.1 Overview of the design

The requirements from chapter 3 were divided in four parts, namely platform interoperability, communication technology standard, application standard and state. We will give an overview of the design, and show how the main elements of our design can fulfil these requirements.

We have chosen to put applications (applications in this sense is implementations of services) on a 3G UICC. This is similar to the ETSI's CAT solution where applications are placed on the UICC (see section 2.5.1), and where CAT offers all communication interfaces present on the handset and simple user interfaces to these applications. Placing the applications on the UICC requires that we design a service similar to CAT. We call this service "UICC service", and base the design of this service on the CAT standard (CAT –ETSI TS 102 223). The design will not be equal to CAT, but follow the same basic principles.

Placing the applications on the UICC solves the issues surrounding interoperability. We achieve this since all handsets have a UICC. 3G UICC equipped with the Java Card OS is an interoperable platform, independent of handset. The interface between handset and UICC is generic by the fact that all handsets support the standardized ISO 7816 interface. With the help from the UICC service or CAT, these applications will have access to all communication technologies present on a given handset.

In addition, we also solve the challenge surrounding loss of state when users change handset. We achieve this because users can bring the UICC from handset to handset when they change handsets. State is stored in the UICC; hence we preserve state in the transition between handsets.

Since there is generally less memory available on UICCs compared to handsets and the UI we can offer through the UICC service is limited, we have also chosen make it possible to place applications with these properties partially on the handset.

There are now two challenges left to handle: Communication technology standards and application standards for services. To handle these challenges, we include an external service. We have chosen to let network operators offer this external service since operators are able to determine handset's positions through location-based services. If the network operators in addition have information about locations ability to offer services through communication technology and application standards, the network operator can tell the applications which communication technology and application standards they should use at any given location. This should solve the challenges surrounding communication technology and application standards.

This solution is similar to OMTP's solution (see section 2.5.1) where external servers update what OMTP calls "service enablers" in their framework. The difference between our and OMTP's solutions is that our solution uses knowledge about handsets locations in order to update service enablers. When and how OMTP pushes their enablers to the handsets, or what these enablers include, is not specified in their specification. Hence, we did not base our solution on OMTP's solution even though there are similarities.

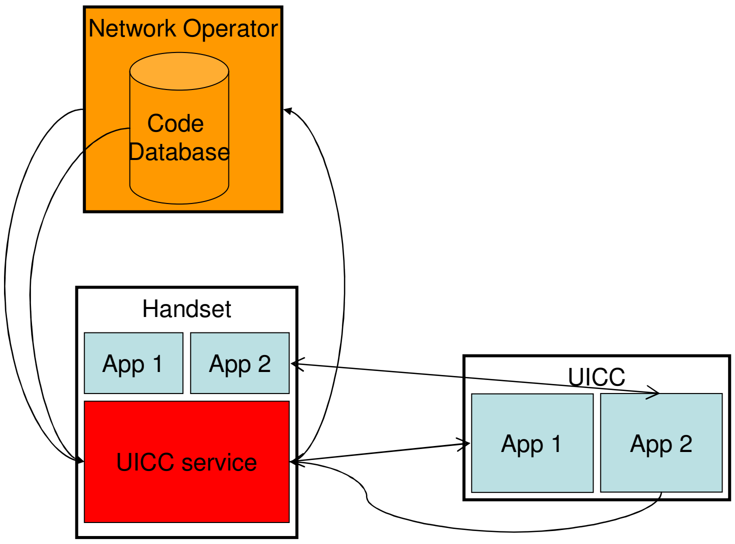

Figure 15: Overview of the design

Figure 13 shows the main elements in our design. Applications are placed mainly on the UICC, and can in addition have a part placed on the handset. A UICC service offers communication interfaces and simple UI to the UICC applications. In addition we have a network operator that notifies the UICC applications about communication technology and application standard-change. The network operator communicates with the UICC applications through the UICC service. We will now show more thoroughly how the architecture is built up before we show how elements in the design communicate through protocols.

4.2 *Architecture design*

Our architecture consists of three physical elements: handset, UICC and network operator (Figure 15). We can further divide these elements in two parts: UICC and handset are the local elements and the network operator is the remote element. We can say that the network operator acts as the server for the clients: UICC and handset. First we will design the architecture of the local elements, and then show how the local elements are put together with the remote network operator's server.

4.2.1 UICC and Handset architecture

The architecture of the UICC and handset we have three main parts (Figure 16): Applications on the UICC, applications on the handset and the UICC service on the handset. The UICC service is based on the ETSI TS 102.223 standard.

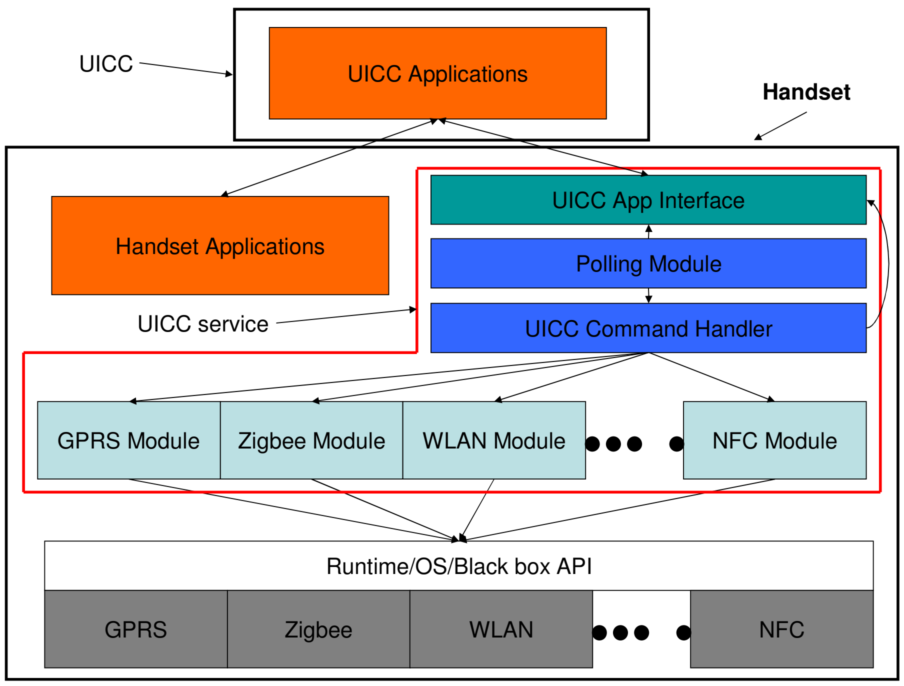

Figure 16 Handset architecture

Applications

Services, implemented as applications, are divided in at most two parts on the UICC and handset. This division is done because parts of an application might have to be tailored for specific handsets, typically UI specific code. Also, if applications are large in size, memory-consuming parts can be placed on the handset since it normally has larger storage capacity than the UICC. These handset specific parts are placed on the handset, and state and control parts of the application are placed on the UICC.

Since some applications might be simple in terms of UI, it is possible to put the whole application on the UICC and let the UICC service on the handset do UI for the application. This is typically simple UI such as setting up menus and displaying text. These simple UI operations were first supported through GSM 11.14 SIM Application Toolkit, and on handsets with 3G UICC support, this is supported both through USIM Application Toolkit (USAT – section 2.3.4.4) and in the near future Card Application Toolkit (CAT –section 2.5.1). Since our architecture is based on the CAT ETSI TS 102.223 standard, such simple UI operations are also supported by the UICC service.

A service will always have an application residing on the UICC. We can call this application the main part of an application. If the application needs more UI support than the UICC service can offer through the CAT standard, another part of the application can be placed on the handset. Applications are given an ID, and applications on the UICC and handset are related through this ID. This ID maps handset applications to the UICC application. The correlation is shown in Figure 17:

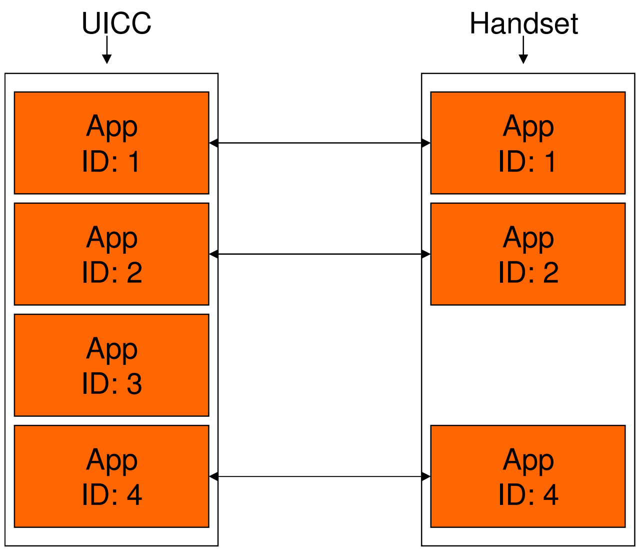

Figure 17: UICC and handset applications ID mapping

Communication between the applications is addressed in the next section.

UICC service

The UICC service is a handset specific module that offers communication and simple UI services to the applications similar to the ETSI 102 223 standard. We don't know the architectural design of CAT, and our architectural design of the UICC service is based on our knowledge about the CAT-functionality.

Since the UICC is a smart card, it is designed to only respond to commands. This design is called reactive design. In GSM 11.14 the UICC card became proactive, but it was still a smart card. This was achieved by polling the UICC every 1 ms, hence giving the UICC applications the possibility to respond with a command to the handset, and they became pro-active. Our UICC service is based on the same principle. The architecture is shown in Figure 18.

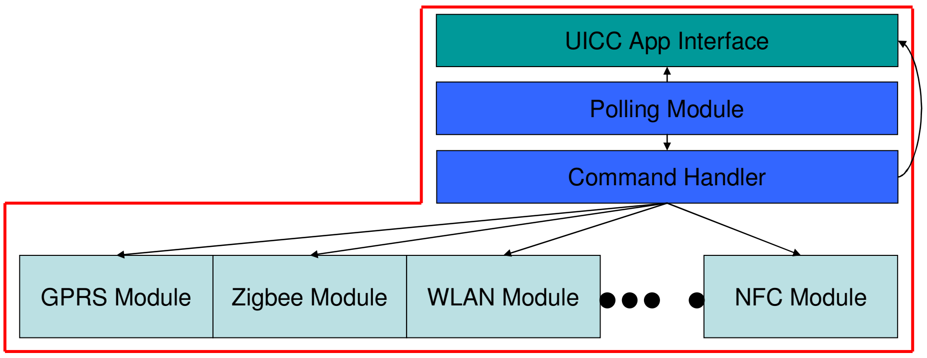

Figure 18 UICC service

We have a module that polls the applications every ms, we call it the Polling Module. If applications have commands for the UICC Service, this is passed to the Command Handler. The Command Handler interprets the commands from the UICC applications and does the desired action. Every communication technology has a module in the UICC service for establishing contact and sending and receiving data through the communication interface. In addition, the UICC service has a module for simple UI needed by applications that only reside on the UICC. The Command Handler loads and uses the correct module depending on the desired action from the UICC application.

The Command Module can contact the UICC applications directly if any external event has occurred. Such events can be received data from one of the communication interfaces intended for a UICC application (e.g. data received on a socket), or it can be user input from the UI module (e.g. activating a menu item).

4.2.2 Network Operator and handset/UICC

The network operator's role in the architecture is to be the maintainer of the system. Maintenance of the system involves adapting to changes in service communication technology and application standard.

We have earlier talked about both external and internal technology –change for use of services. Internal technology refers to the available communication technology on a handset, and external technology is the available communication technology for services in the handset's current environment. For the network operator to enable adaptation to available external technology and application standards, it needs to acquire knowledge about the service-user's location, and the location's service availability through certain communication technology and application standards. This knowledge is obtainable for the network operator since it knows which base station the handset is connected to, and the location of this base station.

When the network operator has obtained knowledge about the user's handset, location and the location's ability to provide services, the network operator can adapt the applications on the user's handset. There are three scenarios for this adaptation: Handset change, environment's communication technology change and environment's application standard change.

Handset change

As users change handsets, two challenges surfaces in our design. First, the UICC service (or parts of it) is missing on the new handset as long as the handset don't have the whole 102.223 standard implemented. The UICC service needs to be transferred to the new handset as the user inserts his/hers UICC in the new handset. The UICC service is implemented according to the handsets communication technologies, UI and best-suited developing language (depending on the handsets platform).

Second, parts of applications that resided on the old handset are missing on the new handset. These parts need to be installed on the new handset. The handset-specific parts of an application are tailored for the current handset. The network operator has a database containing a range of implementations of the UICC service and applications that are tailored for different supported handsets. These implementations are transferred to the handset through either Internet (GPRS, WLAN etc.) or as binary SMS (Figure 19).

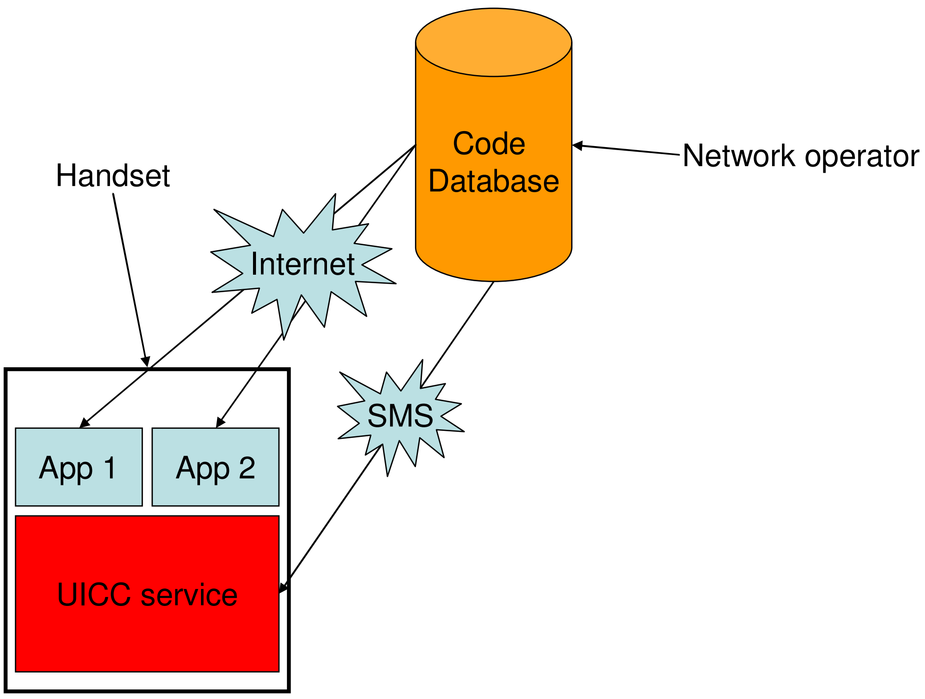

Figure 19: Network operator application/UICC service transfer

Environment's communication technology change
When the network operator has determined what communication technology an application can use to offer a service for the user, the network operator needs to tell this to the application. Since the part of the application on the UICC is the part that chooses what communication interface at the UICC service to use, the UICC application receives a message from the network operator every time the environment technology change (Figure 20). Since the UICC service offers an interface for every available communication technology on the handset, no more than this message is needed by the application (no additional implementation is needed). The received message about communication interface is stored in the UICC application as persistent state until the next message is received from the network operator.

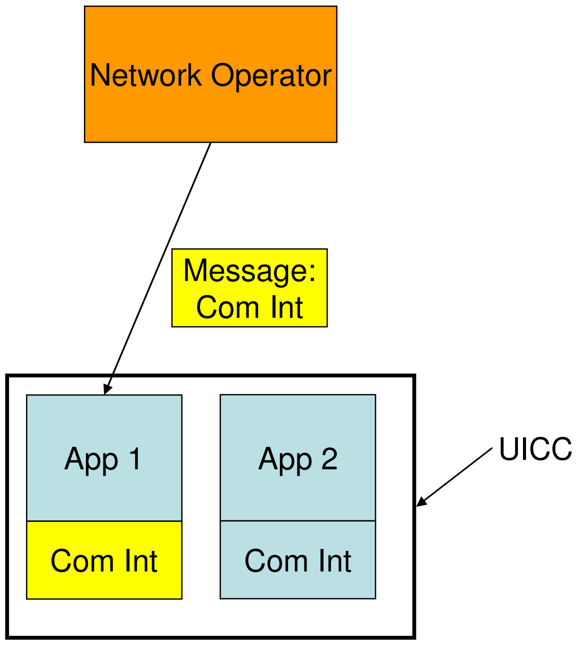

Figure 20: Network operator's message about change of communication interface

Application standard change
In addition to communication technology change, the network operator can determine that the application standard has changed based on the handsets location. The network operator adapts services to application standard by transferring new applications to the UICC that follow the new application standard (Figure 21).

When an application on the UICC is replaced to fit a new application standard, the state of the UICC application needs to be preserved. Letting the UICC application know that it is being replaced enables it to send its state to the network operator. The network operator stores this state, sends a new application to the UICC that replaces the old one, and finally sends the state back to the new UICC application for storage on the UICC.

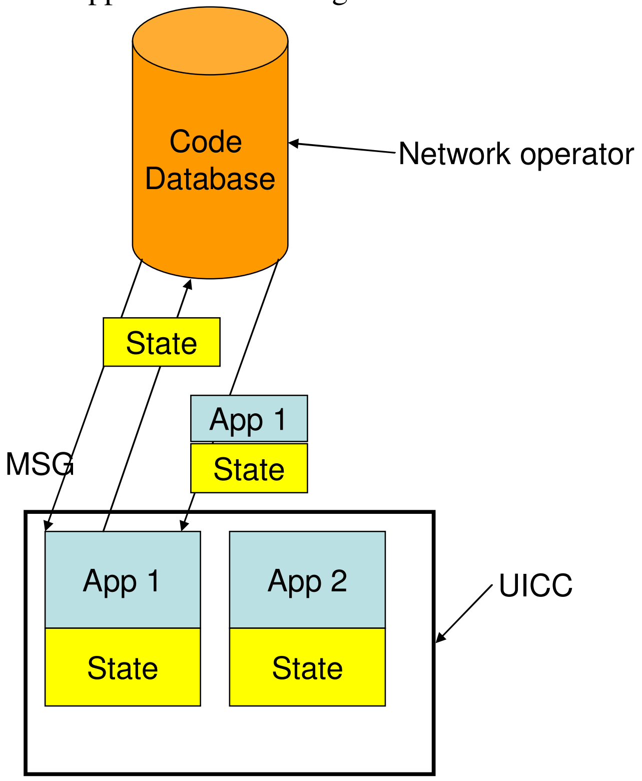

Figure 21: Replacement of UICC application

4.3 Communication design

We have three elements (Handset, UICC and network operator) in our architecture that need to communicate with each other. The communication flows like this:

- Handset applications to UICC applications
- UICC service to UICC applications
- Network operator to UICC applications

All communication ends up with the UICC applications. UICC applications communicate through APDU commands; hence we will start by taking a look at the APDU commands design. Then we design a conceptual communication paradigm between handset applications and UICC applications, UICC service and UICC applications, and network operator an UICC applications.

4.3.1 UICC applications and handset applications

In order for the architecture with applications both on the UICC and handset to work as intended, where applications on the UICC can be replaced without altering the appearance of handset applications, we need to keep the communication between the UICC and handset applications generic. The use of the APDU commands does help us in some extend by offering a generic interface and protocol-structure through the ISO 7816 standard and APDU commands. But this is not sufficient. The APDU commands sent between the handset and UICC applications do also need to be generic because the handset and UICC applications need to agree on actions to take based on the CLA and INS fields.

This calls for a standardization of commands within a group of applications, e.g. ticketing applications. By standardization we mean that e.g. ticketing applications need to use the same APDU commands (identical CLA, INS etc.) and data field within APDU commands. For example a “buy-ticket” operation could take the form of an APDU with a specific CLA and INS field, and a 4-byte data field with a desired credit field. With a number of standardized commands such as buy, use and available credit, corresponding applications on the UICC and handset can be replaced and updated independently of each other. This standardization is needed if we want several actors to develop UICC-applications within a type of service, and where service providers’ implementations can be replaced as users move within range of service providers’ domains.

As an example we have designed a ticketing application residing on both the UICC and handset. This application will have simple functions such as “Buy credit”, “Check credit” and “Use ticket”. For every command there will be a corresponding APDU command. When the user initiates any of these functions through UI events on the handset application, the corresponding APDU is sent from the handset application to the UICC application. The UICC application responds with an OK or Not OK response (Figure 22).

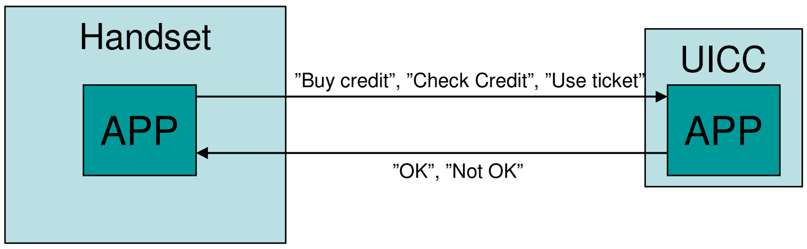

Figure 22: Handset and UICC application command flow

The APDU commands from the handset application and responses from UICC applications is shown in Figure 23:

APDU commands between handset and UICC-appcliations

Action:	CLA	INS	P1	P2	Lc	Data
Buy credit	0xA0	0x01	0x00	0x00	N	Credit to buy
Check credit	0xA0	0x02	0x00	0x00	0	N/A
Use ticket	0xA0	0x04	0x00	0x00	0	N/A

Expected response

Response to:	*Action:*	**Data**	**SW1**	**SW2**
Buy credit	**OK**	N/A	0x90	0x00
Check credit	**OK**	Available Credit	0x90	0x00
Use ticket	**OK**	N/A	0x90	0x00

Optional Response

Action:	**Data**	**SW1**	**SW2**
Not OK	N/A	0x6D	0x00
Not OK	N/A	0x6D	0x00
Not OK	N/A	0x6D	0x00

Figure 23: APDU commands and responses between handset and UICC-applications

4.3.2 UICC applications and UICC service

In ETSI 102 223 the Card Application Toolkit (CAT) is specified. This toolkit offer the services we want for our UICC applications. In ETSI 102 223 a set of operations and APDU commands are specified, and the design of these are available in this document. We are not in possession of a handset where the CAT is implemented; hence we need to make our own implement of CAT. CAT and the ETSI 102 223 document is an extensive specification and it is outside the scope of this thesis to implement CAT in its completeness. Thus we design our own set of commands based on the CAT principles.

Since we want the UICC applications to be pro-active, we need to poll them with a poll-command. If the UICC applications have commands (called pro-active commands) for the UICC service, these commands are fetched by a fetch-command. The pro-active commands obtained by the fetch-command are requests for use of communication interfaces or UI on the handset. These pro-active commands are also covered by the ETSI 102 223 document. We have designed a subset of these pro-active commands. Pro-active commands might produce return-data for the UICC, and this is returned to the UICC application with a Command response. The flow of the polling, fetching, pro-active commands and command response are shown in Figure 24.

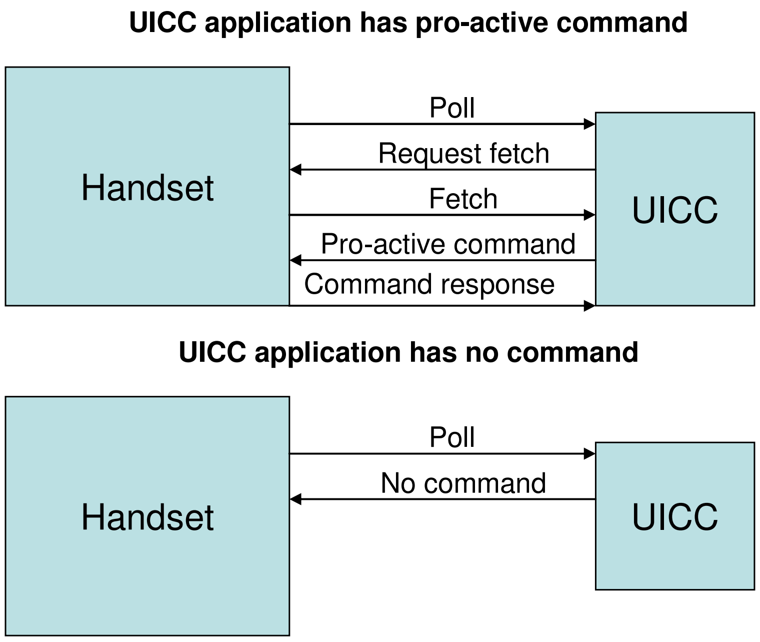

Figure 24 Pro-active commands

In addition to polling and fetching of pro-active commands, data can sent to the UICC applications from an external source. This data can be UI operations (if the UICC service UI is used), incoming data on a network interface etc. We have divided this in two commands: Incoming data (UI and network interfaces) and Incoming data from network operator. This division is done because we want to distinguish incoming data from external processes and the network operator. The operator has the privileges to alter the execution of a UICC application by telling it to change network interface in use etc.

UICC service/UICC

Action:	CLA	INS	P1	P2	Lc	Data
Poll	0xB0	0x10	0x00	0x00	0x00	N/A
Fetch	0xB1	0x12	0x00	0x00	0x00	N/A
Command response	0xB2	0x02	0x00	0x00	N	Response after UICC pro-active command (opt.)
Incomming data	0xB3	0x03	0x00	0x00	N	Data from other external process
Incomming data (OP)	0xB4	0x04	0x00	0x00	N	APDU from OP

Figure 25: Pro-active commands from UICC service

Expected response					**Optional Response**			
Response to:	***Action:***	**Data**	**SW1**	**SW2**	***Action:***	**Data**	**SW1**	**SW2**
Poll	**No command**	N/A	0x90	0x00	**Have command for UICC service**	N/A	0x91	Size command
Fetch	**The command**	Command to UICC service	0x90	0x00	**N/A**			
Command response	**OK**	N/A	0x90	0x00	**OK, additional command**	N/A	0x91	Size command
Incoming data	**OK**	N/A	0x90	0x00	**OK, additional command**	N/A	0x91	Size command
Incoming data (OP)	**OK**	N/A	0x90	0x00	**OK, additional command**	N/A	0x91	Size command

Figure 26: Response from UICC applications to pro-active commands

Figure 24 shows the designed pro-active commands from UICC service to UICC applications. Figure 25 shows the designed responses from UICC applications.

4.3.3 UICC and network operator

The network operator has an important role in our design, since it controls the behaviour of the UICC applications. Communication between the network operator and UICC applications should for this reason be secured. Smart cards offer secure communication through encryption of APDU commands. The network operator needs to communicate through the UICC service for delivering APDU commands to the UICC applications, and the UICC service can be viewed upon as an unsafe link in the communication. This is because the UICC service can be replaced by a false UICC service. For this reason, we have chosen to create APDU commands on the network operator-side that can be encrypted with a key known only by the UICC applications and the network operator. The network sends the application ID and the APDU to the UICC service. The UICC service creates a new APDU command with the received network operator APDU as the data field; hence the network operator command becomes nested in the UICC service command. When the UICC application receives the command, it extracts the network operator APDU, executes the command, and creates a response for the network operator. This response is encrypted and put in the data-field of the response-APDU. The flow of this operation is shown in Figure 27

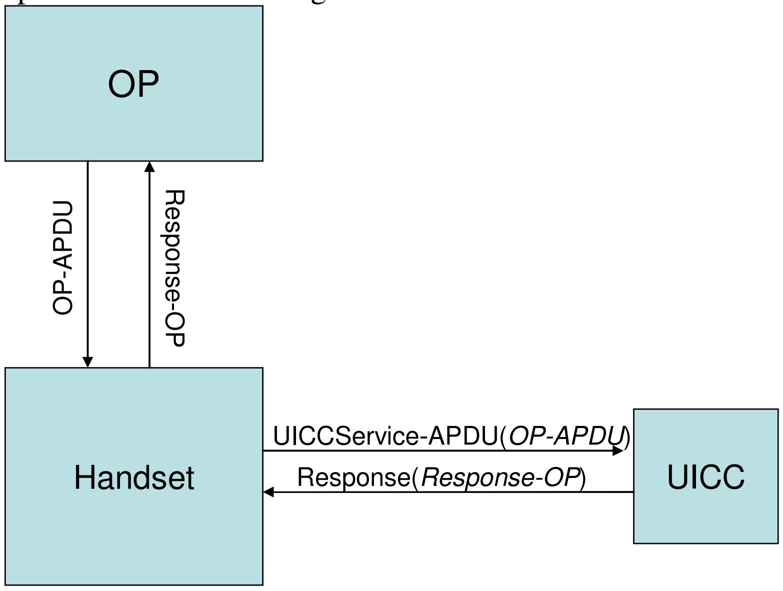

Figure 27: Network operator and UICC applications' command flow

This nesting does not only ensure secure communication between the OP and UICC application, it also increases the number of available APDU commands between the OP and UICC application. APDU command has 2^16 available commands through the CLA and INS fields (2^8 CLAss and 2^8 INStructions within the CLAss). But since many CLA fields have been reserved for different standardized commands (such as Global Platform and CAT commands), the number of commands is much smaller. By using one INS field within the

UICC service commands for nesting OP-commands, we have freed 2^16 different commands for the OP.

APDU commands between operator and UICC applications

OP/UICC						
Action:	CLA	INS	P1	P2	Lc	Data
Switch buy com.	0x01	0x01	0x00	0x00	N	Com. Interface, address, port etc.
Switch use com.	0x01	0x02	0x00	0x00	N	Com. Interface, address, port etc.
Request state	0x02	0x00	0x00	0x00	0	N/A
Receive state	0x03	0x00	0x00	0x00	N	Chunk of State data

Figure 28: APDU commands from network operator to UICC applications

Figure 27 shows the commands that are sent from the network operator to UICC applications. These commands are nested in the data field of the "Incoming data (OP)"-command from Figure 24.

4.4 Summary

In this chapter we designed an architecture in order to meet the goal, secondary objectives and requirements from chapter 3. In the architecture we wanted to use CAT as an element, but because of no known implementation of this tool we chose to make our own design of a UICC service, though based on the same principles as CAT. The design was meant to handle four challenges we identified in chapter 2 surrounding services on mobile handsets. We used the UICC as an interoperable platform and gave access to handsets' communication technologies and simple UI through the UICC service in order to meet the first and fourth secondary objectives. The network operator was given an administrative role of telling the UICC applications to change communication technology and replace UICC applications to meet the second and third secondary objectives. In the next section we will implement this design on a handset.

5 Implementation

The implementation is based on the design from chapter 4. We have chosen to implement the UICC service and one test application. This application is a simple ticketing application placed both on the handset and the UICC. The UICC service will offer a SMS, NFC and GPRS interface to the UICC application. In addition, we will simulate messages from a network operator for change of network interface on the UICC application, and replacement of the UICC application.

We used a Samsung SGH-X700N ([54]) as the handset for our implementation for reasons explained in the next section. The development-tools we used in this implementation were Eclipse ([55]) for developing Java ME midlets, and Eclipse with a JCOP plug-in ([56]) for developing cardlets for smart cards. To transfer and communicate with cardlets on the smart card we used a SDI 010 card reader ([57]).

The implementation is a "proof of concept"-implementation, and the goal of the implementation is to show that our design is feasible in a real system.

5.1 Limitations

Some challenges surfaced when we implemented our design. First of all we do not have network operator privileges. Without these privileges we are unable to access a UICC on the handsets we are in possession of. Unfortunately we do not have a handset with a UICC with a Java Card implementation that does not require network operator privileges for access. Therefore we have chosen to use a Samsung SGH-X700N with a Java Card and NFC radio as the handset platform for our implementation. With this handset we can use the Java Card to simulate a UICC with the Java Card OS.

Samsung SGH-X700N has a black-box OS, and application development is only possible on the Java ME platform (Java ME specification for Samsung SGH-X700N is available in Appendix A). This poses a second challenge: Only one Java application can run at a time. In other words, we can run either an application or an UICC service (since both needs to be implemented in Java). Hence, the ticketing application and the UICC service must be implemented in the same application.

A third challenge also appeared related to SMS communication because of the specific handset. The SMS implementation in Java ME on the Samsung SGH-X700N is buggy when receiving SMSs. This created a problem when we tried to receive and send SMS successively, and when receiving SMS and sending to the Smart Card successively. After extensive testing and conferring with other developers we found that the receive routine in the Java implementation on this specific handset generates several interrupts when receiving a single SMS. If we try to do an IO operation after receiving a SMS (such as writing to the Smart Card or send a SMS), these interrupts will force the application to halt. For this reason we chose to not receive SMS in our implementation.

5.2 Overview of the implementation

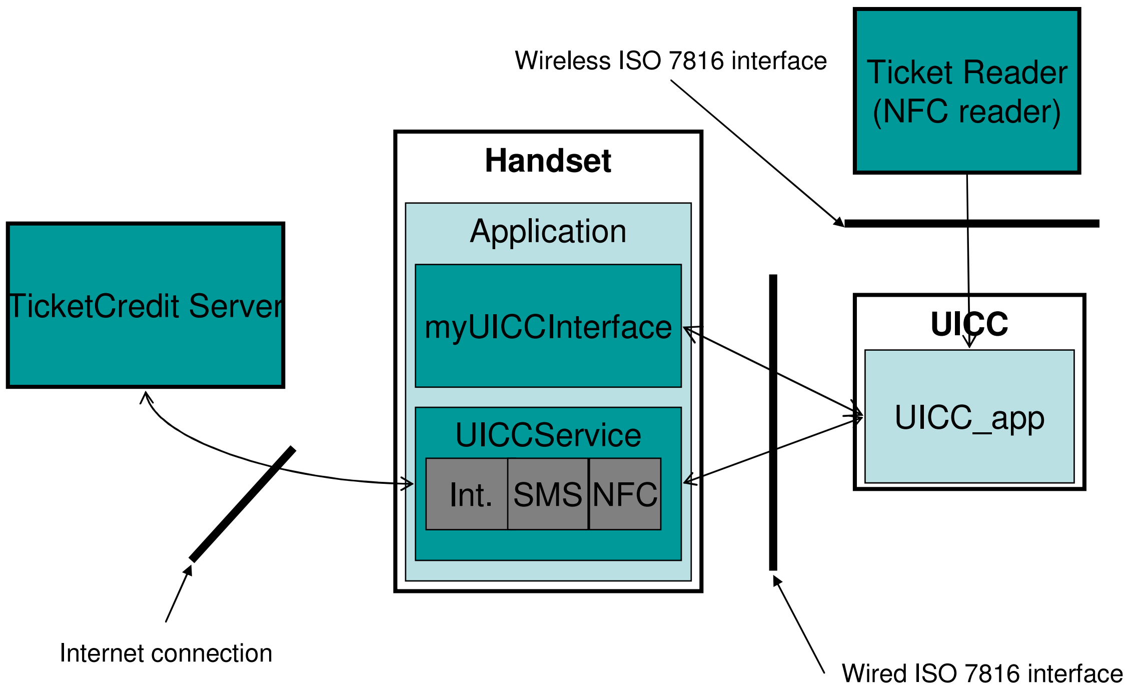

Figure 29: Overview of the implementation

With these limitations revealed, we can give an overview of our implementation (Figure 29). We have implemented:

- Ticketing application on the UICC
    - File: ***UICC_app.java***
- Ticketing application on the handset
    - File: ***Application.java***
- UICC service
    - File: ***Application.java***
- Network operator simulation
    - File: ***Application.java***
- Simulated Ticket SMS Server
    - File: ***Application.java***
- Ticket Credit Server
    - File: ***Server.c***
- Ticket Reader (NFC reader)
    - File: ***ticket.jcsh***

We will now take a brief look at what the different parts contain. The full Java-documentation of UICC_app.java and Application.java can be viewed in:

- …/src/<Application | UICC_app>/doc/

5.2.1 Application.java

On the handset we have one Java implementation called "Application.java". This midlet (applications are commonly referred to as midlets when implemented in Java ME) contains one main class also called "Application". The "Application" class implements all UI events that are initiated by the user. UI events are divided in two, both logically and while executing.

One part is UI used for the ticketing application itself, e.g. operations such as “use ticket” etc (Figure 30).

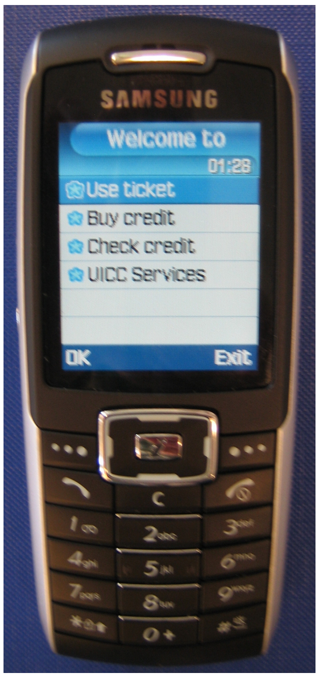

Figure 30: Ticket application UI events

Another part is for testing the functionality of the UICC service and network operator’s role in the system (Figure 31). Because of the limitations we discovered regarding receiving SMS’, we chose to implement the network operator’s messages as UI locally on the handset rather than actually receiving SMS’.

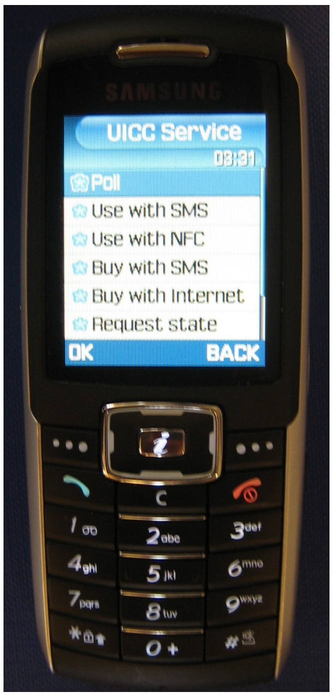

Figure 31: UICC service/network operator UI events

We implemented two classes (Figure 32) that communicate with the UICC application to keep the two groups of UI events logically apart.

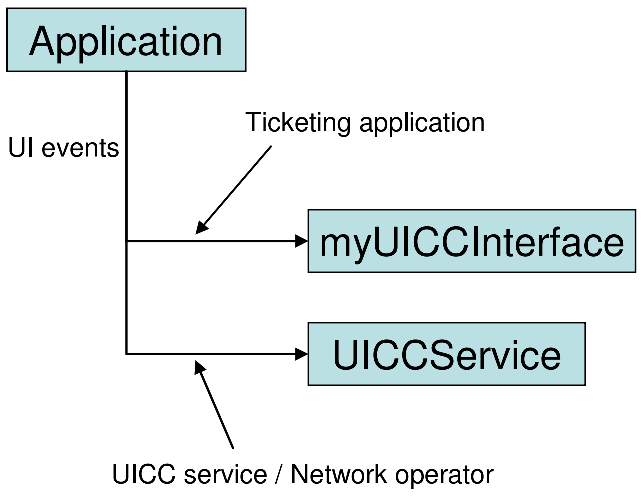

Figure 32: Classes on the handset

The class "myUICCInterface" implements the communication that the ticketing application on the handset uses to communicate with the UICC application. "UICCService" implements the UICC service functionality. In addition, the simulated network operator locally generated commands are sent trough this class. Both classes communicate with the UICC application through the ISO 7816 interface.

5.2.2 UICC_app.java

On the simulated UICC (the Java Card connected to the NFC on the handset) we have a ticketing application implemented for the Java Card platform. This application has a both a wired and wireless ISO 7816 communication interface. The wired ISO 7816 interface is used towards the "myUICCInterface" and "UICCService" classes on the handset, while the wireless interface is used towards the NFC reader we use as the TicketReader (Figure 33).

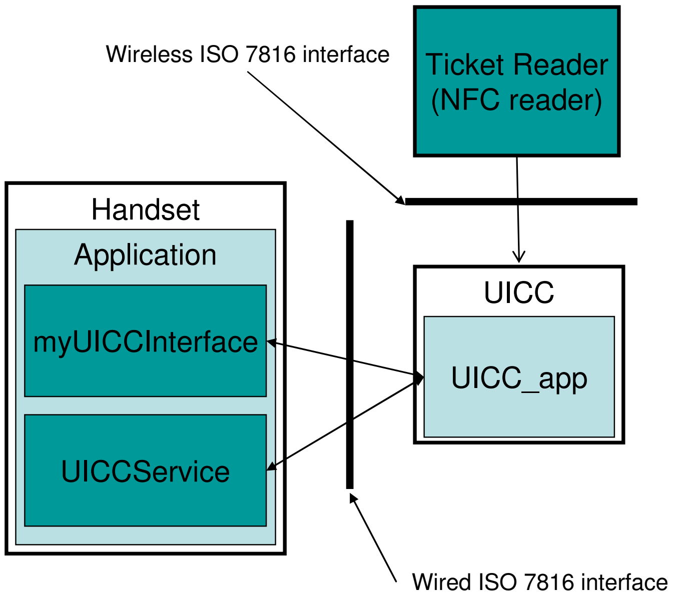

Figure 33: UICC application

5.2.3 server.c

We have implemented a simple remote server in C that simulates a payment server for the ticketing application. The UICC service on the handset can access this server on behalf of the UICC application through an Internet connection (Figure 34).

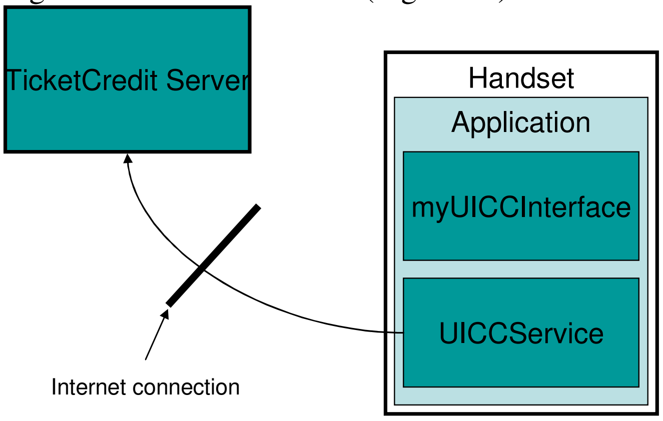

Figure 34: Server.c

5.2.4 ticket.jcsh

ticket.jcsh is a JCOP command script running through Eclipse and a SDI010 reader on a desktop computer. The script uses an APDU command implemented on the simulated UICC to subtract the credit value with a given value.

5.3 Ticketing application –Application.java

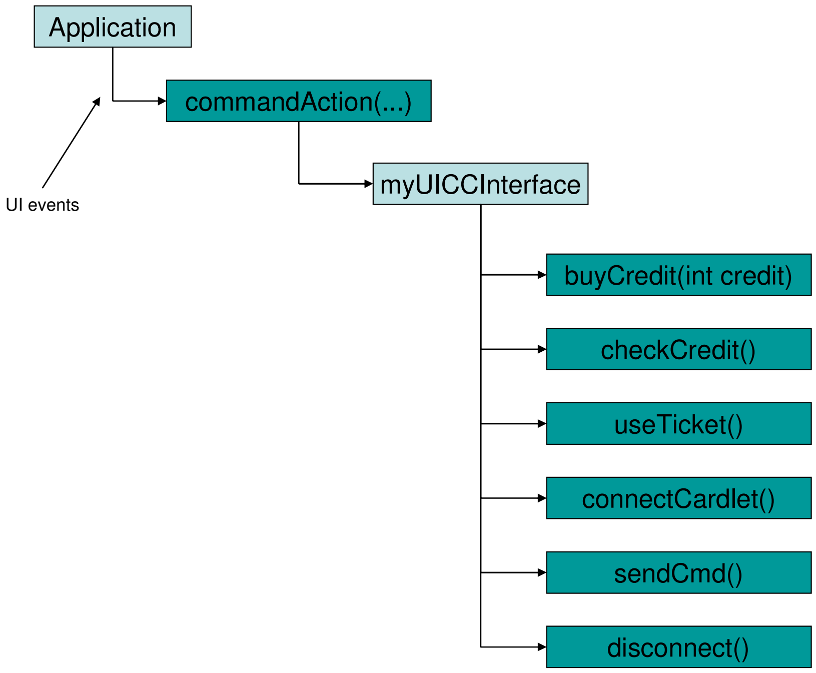

Figure 35: Classes (light green) and methods (dark green) used by the ticketing application

The ticketing-application implemented in the Application-class has a simple UI initiated functionality. We have three functionalities; “Use ticket”, “Buy credit” and “Check credit”. When the user activates any of these commands, the UI event handler ”commandAction” uses an instance of the class ”myUICCInterface”. This class has a method for every command:

"useTicket()" for the "Use ticket" command, "buyCredit(int)" for "Buy credit"-command, and "checkCredit()" for "Check credit"-command.

The myUICCInterface methods do all send an APDU command to the corresponding UICC application according to the protocol shown in the design (section 4.3.1). The myUICCInterface contain the methods "connectCardlet", "sendCmd()" and "disconnect()" for activating the UICC, send APDU commands and de-activate the UICC.

In order for the handset application to give the APDU command to the corresponding UICC application, it needs to know the AID (application id) of the UICC application and select it. We have chosen to solve this by also assigning an AID to the handset application, equal to the UICC application's AID.

After selection, the ticketing application simply relays user input to the UICC application through the methods in the "myUICCInterface"-class. This user input tells the UICC application that the user wants to use a ticket, buy credit with the value X or to check the remaining credit. The UICC application responds by accepting or discarding the command, and from this point the UICC application decides what action is to be made in order to complete the user request.

5.4 *Ticketing application –UICC_app.java*

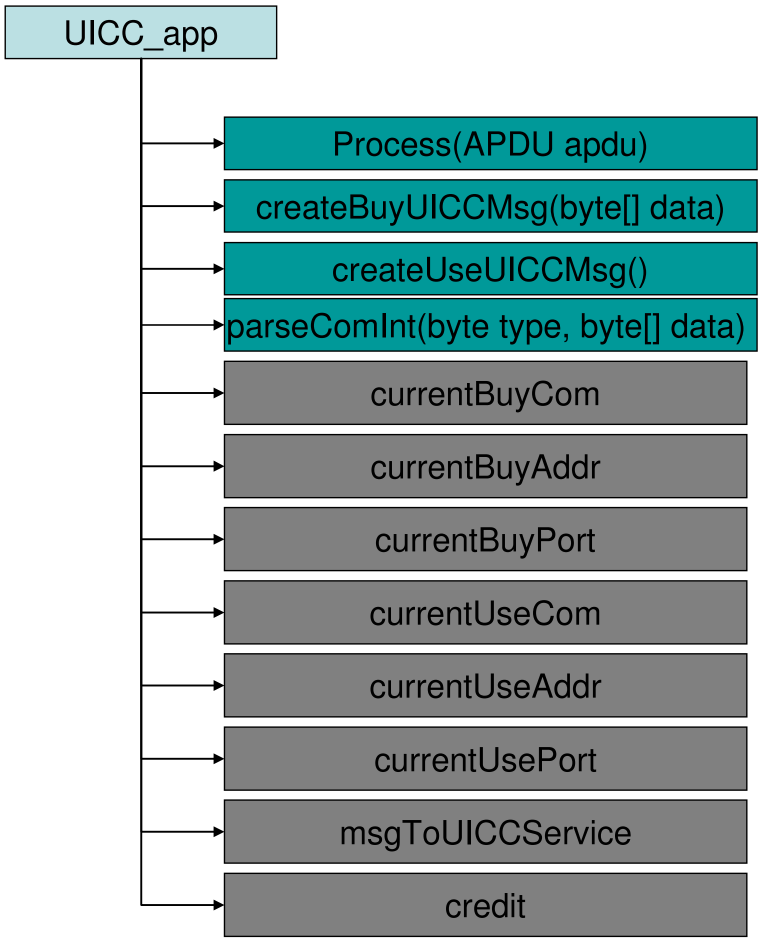

Figure 36: Methods (green) and values (grey) in the UICC_app class

The ticketing application on the UICC is the "brains" of the ticketing service. It contains state about the amount of credit the user has left, which communication interfaces and address information it should use, and the application standard (protocol etc.) to use. Since the ticketing application has two communication operations, buy credit and use ticket, we have two communication interface and address states. All state is stored in variables in the application.

When the UICC application receives an APDU command, it is processed in the method "Process(APDU apdu)". Depending on the received command (CLA and INS fields in the APDU command), this method decides what action it should take.

5.4.1 Commands from the handset application

The UICC application can receive three commands from the handset application: "Buy credit", "Check credit" and "Use ticket". The "Check credit"-command simply returns the value of the credit-variable, while the other two requires communication with services outside the handset.

When the UICC application receives either a "Buy credit" or "Use ticket" command, it uses its current communication interface –state to create a command to the UICC Service on the handset. This state is stored in the values: currentBuyCom, currentBuyAddr, currentUsePort etc. All values are shown in Figure 36. These values are information earlier received from the network operator about change of communication technology (see next section).

The command for the UICC service is created by calling the methods createBuyUICCMsg(byte[] data) or createUseUICCMsg(), depending on if it is a buy or use command. This command contains the communication interface it wants to use, the address for the data (currentBuyCom, currentBuyAddr, currentUsePort etc.), and a data-message parsed for a given application standard. The application standard is defined within the UICC application itself, and this application has to be replaced by the network operator in order to change the application standard. The command created by either createBuyUICCMsg() or createUseUICCMsg() is stored in the buffer "msgToUICCService", which the UICC service will receive after poll and fetch commands.

5.4.2 Commands from the UICC service

The UICC application can also receive commands from the UICC Service on the handset in the "Process(APDU apdu)"-method. We have implemented three commands: "Poll", "Fetch" and "Message from OP". On "Poll" the application checks if it has pending commands for the UICC service and notifies the UICC service about the result. If it has a command, a "Poll" command is followed by a "Fetch" command. On "Fetch" the response buffer "msgToUICCService" is returned. This buffer is already parsed and configured correctly for the UICC service.

Messages from the network operator should be encrypted and unreadable for the UICC service because the UICC service is a vulnerable part of our architecture. Incoming messages from the OP is nested in the UICC service command as we designed in the previous chapter. The UICC application extracts the OP-command from the UICC service command's data field, and handles the OP-command as an independent APDU command (Code example 1).

```
        byte[] buf = apdu.getBuffer();
        switch (buf[ISO7816.OFFSET_CLA]) {
               //From the UICC_service
               case (byte) 0xB0:
               :
               :
              //Incomming message from OP
               case (byte) 0x04:
                      byte lenOfOPmsg = buf[ISO7816.OFFSET_LC];
                      byte[] OPmsg = new byte[lenOfOPmsg];
                      short i;
                      for(i = 0; i < lenOfOPmsg; i++)
                             OPmsg[i] = buf[ISO7816.OFFSET_CDATA + i];
                      //Check message from OP
                      switch (OPmsg[ISO7816.OFFSET_CLA]) {
                      //New COM
                      case (byte) 0x01:
                             switch (OPmsg[ISO7816.OFFSET_INS]) {
                             //New buy COM
                             case (byte) 0x01:
                                    parseComInt((byte)1, OPmsg);
                             break;
                             //New use COM
                             case (byte) 0x02:
                                    parseComInt((byte)2, OPmsg);
                                    break;
                             default:
                                    break;
                             }
                             break;
                             :
                             :
```
Code example 1: Extraction of network operators message

We have implemented four commands that the UICC application can receive from the network operator: “Change buy com.”, “Change use com.”, “Request for state” and “Old state”. When “Change buy com.” and “Change use com.” –commands are received, the application replaces the current communication interface –state (Figure 36: currentBuyCom, currentBuyAddr, currentUsePort etc.) with the newly received information. Next time the application receives a request from the handset application to buy or use the ticketing credit, these new communication interfaces and addresses will be used.

Before the network operator decides to replace the application on the UICC, it needs to preserve some state currently placed on the UICC. The “Request for state”-command forces the UICC application to pack and return its current state to the network operator. After this, the network operator can replace the application and send the preserved state to the new application. This way the credit and network interface information is kept concurrent after the UICC application is replaced.

5.5 UICC service –Application.java

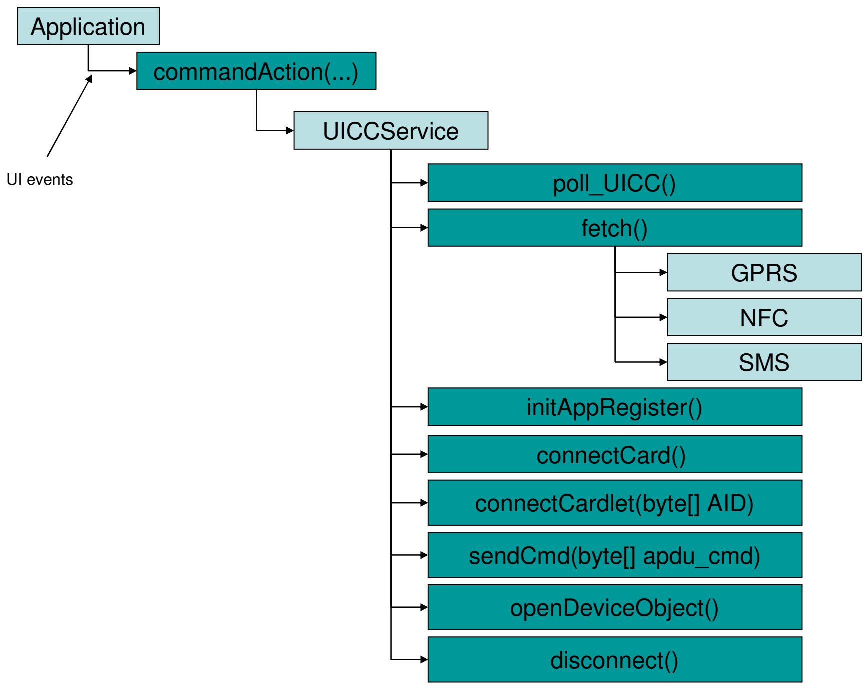

Figure 37: UICCService class and method

The UICC service is the UICC applications link to services and processes outside the handset, and also provides the link to the network operator. According to our design the UICC service can also provide simple UI to the UICC applications, but we have chosen to not implement this in our solution in order to limit the implementation. The implemented functionality in the UICC service is “Poll” and “Fetch” commands and methods for communicating with the UICC, and in addition nested classes for providing SMS, NFC and GPRS communication.

In our implementation we have chosen to implement the “Poll” functionality as a user driven event (UI) in contrary to a background-process that polls the UICC within a given time-period. There are two reasons for this: The Java ME implementation on this handset has limited support for threading. But the best argument for doing polling on request from the user is to test the UICC service functionality while getting output from the actions taken on behalf of the UICC application.

Since the UICC can contain more than one application, the UICC service needs to poll all the UICC applications on the UICC. The UICC service keeps a register of the AID of the applications on the UICC and polls them successively. If an application has a command to the UICC service, the service sends a fetch command in order to receive the command.

The command received from a “fetch” operation is processed to determine what action is to be made. For example if the UICC application gives a command to send data through GPRS, the UICC service extracts the address and port of the destination, and the data field. Then an instance of the GPRS class is loaded and used to send the data to the given address and port. The same is done for SMS connections, while the NFC command only requires that the UICC service put the smart card in wireless mode. This is because the smart card can only be connected either to the handset (wired mode) or to the NFC radio (wireless virtual mode, also called virtual mode).

Since we had problems with receiving SMS' on the Samsung handset, we implemented the network operator's commands as user-initiated operations. These commands are sent through the instance of the "UICCService" class.

5.6 Network operator operations –Application.java

All network operations are implemented as UI events, and the menu is shown in Figure 31. The commands are equal to the designed commands, but instead by being created on a server and relayed by the UICC service, they are created locally on the handset and sent to the UICC. The commands implemented are:

- Use with SMS
- Use with NFC
- Buy with SMS
- Buy with Internet
- Request state
- Send new application
- Send old state

Every command follows the designed commands from the design (section 28) except the "Send new application" command. Since we had troubles receiving SMS' we chose to install a new application through Eclipse with a JCOP plug-in. When we use Eclipse and JCOP, a NFC reader connected to the computer is used to install the application. Hence, the "Send new application" command sets the smart card in wireless mode so it can accept commands from the NFC reader.

The commands "Use with SMS" and "Use with NFC" gives the UICC application commands to change the communication interface and address for using tickets. "Buy with SMS" and "Buy with Internet" changes the communication interface for buying credit. "Request state" sends a command to the UICC application to return its current state. This state is stored in the midlet and returned to the application on a "Send old state" command.

5.7 TicketCredit Server –server.c

TicketCredit Server is a small server application on a remote computer that is implemented to test the "Buy credit with Internet" command. When the UICC application receives this command from the simulated network operator, it will send a buy-credit request to this server, and the server returns the desired credit. The address and port of the server is included in the command received from the simulated network operator.

5.8 Ticket reader –ticket.jcsh

The ticket.jcsh script communicates with the ticketing application on the UICC through a NFC reader on a computer. When the UICC application is put in wireless mode, the handset placed over the reader and the script is run, the script accesses the UICC application based on the AID. Then it sends a command for subtracting the credit-value in the ticketing application. The ticketing application subtracts the given value if it has enough credit and returns an OK response (0x90). If there is not enough credit, it returns a "Not OK" response. The result of the operation is viewable in the JCOP command shell.

5.9 Summary

In this chapter we implemented the design from chapter 4. Because of limitations of not being a network operator, not having an open 3G UICC, a handset without CAT and a faulty Java implementation on the Samsung SGH-X700N, we had to simplify our implementation to some extend compared to our design.

6 Tests

In this chapter we test our implementation from chapter 5.

6.1 Test scenarios

In chapter 2.3 we explained four groups of challenges regarding services on handsets. These were:

- Lack of interoperability amongst handsets
- Different communication technologies within the same type of services
- Different application standards within the same type of services
- Loss of state when users change handsets

We will set up four scenarios that reflect these four challenges.

6.1.1 Lack of interoperability amongst handsets

To test the first test scenario, we need to check how our system handles different handsets with different configurations. Since we only are in possession of one type of handset with a Smart Card and NFC radio, we have to simplify the test. The Smart Card Java API on the Samsung handset is a proprietary API; hence it will not work on any other handset. The test fails already. But by making some assumptions, we are still able to test this scenario. Handsets equipped with support for Java UICCs and CAT will provide a standardized API towards the UICC and a service similar to our UICC service. With this knowledge we can make a test application without the Smart Card API and test it on other handsets. We have chosen to test it on:

- Motorola RAZR V3i
- Nokia E65
- Nokia 6680
- Sony Ericson K700i

These four handsets will be compared to the Samsung SGH-X700N handset, which the implementation was intended for.

6.1.2 Different communication technologies within the same type of service

The next test focuses on how our system handles change of technology for a given type of service. We have implemented a ticketing application on the handset and the simulated UICC, and a UICC service for this purpose. The user can buy credit and use tickets. The UICC application can receive information about which communication interface it should use for these two functions from a simulated network operator.

By changing network interfaces with a service running in the other end while letting the user use the ticketing functions, we can test if the network interfaces are changed correctly and transparently to the user. We will test these ticketing functions:

- Use ticket with SMS
- Use ticket with NFC
- Buy credit with SMS
- Buy credit with Internet

6.1.3 Different application standards within the same type of services

To test different application standards within the same type of services we chose to alter our ticketing application to use a different protocol for buying credit through Internet. For this to be transparent for the user, the application-state needs to be stored at our simulated network operator, a new application transferred to the simulated UICC, and finally the state must be returned to the new application.

6.1.4 Loss of state when users change handsets

The last scenario is not possible to test because state is stored in the simulated UICC, and this Smart Card cannot be removed from the handset. If we had used a proper UICC we would have been able to bring the UICC, thus the state, to a new handset. Even so, to prove that the state is not stored in the handset-application, we can delete this application, install a new, and check the state.

6.2 Test execution

6.2.1 Lack of interoperability amongst handsets

To test the lack of interoperability, we stripped our handset implementation of all classes and functions that was related to communication, and we were left with a UI implementation. We installed this implementation on all four handsets in addition to the Samsung handset. We tried all the UI functionality to check if the user events produced the same output on all handset, that is: We tested our program flow up to the point where the proprietary Smart Card API was called on the Samsung handset.

6.2.2 Different communication technologies within the same type of service

To test the two ticketing functions with two different communication technologies, we need to make sure that the communication state on the UICC application does not contain the correct address or interface at start-up. By re-installing the application without state, this is achieved. Then we used the menu from Figure 31 to send commands to the UICC application about change of address and interface. After we have sent these commands and tried to either use tickets or buy credit, we use the Poll command from the menu in Figure 31 to do the actual polling and check the action taken by the UICC service.

Use ticket with SMS
When using the “Use ticket” function with SMS, we chose to implement a simulated SMS service. To check the correctness of the UICC application we could not rely on a service on the other end to receive SMS. Instead we checked that the destination number and port, and the payload-data were correct. In addition, we check that the credit value on the UICC is correctly subtracted.

Use ticket with NFC
The NFC interface on the Samsung handset is connected to the Smart Card we used to simulate the UICC, hence the simulated UICC communicates directly with the ticketing service when its supposed to use NFC. But because of the wiring on the NFC Smart Card, the handset needs to set the Smart Card in “virtual” mode (also called wireless mode). The UICC service puts the UICC in this mode and tells the user to swipe the handset over the NFC reader. The correctness is controlled by the output from the JCOP command script and by checking that the credit value is subtracted correctly.

Buy credit with SMS
We test this function the same way as “Use ticket with SMS”, but with different destination number and port.

Buy credit with Internet
We have a remote ticket server that prints out all data received on a given port, and returns a message according to the given protocol. To check the correctness of the Internet communication, we check that the server receives the data and that this data is sent back to the simulated UICC through the UICC service. If the interface, address, port and data is correct, the credit value in the UICC application will be incremented with the same amount as the user requested.

6.2.3 Different application standards within the same type of services

To test this scenario we make sure that the UICC application has state about both use and buy communication interface, and a given value of credit. Then we retrieve the state from the application, install a new application, and return the state. The new application has an altered communication protocol towards the credit server, but the address and port should be the same. We then check that the credit value is the same as before the new application was installed. In addition we test both the use and buy functions, and check that the communication with the credit server is correct according to the new protocol. If all these elements are correct, the test was executed successfully.

6.2.4 Loss of state when users change handsets

In the final test we make sure that the UICC application has a certain state before we delete the handset application. If we install the application and the state is the same as before the deletion, we have preserved the state.

6.3 Test results

6.3.1 Lack of interoperability amongst handsets

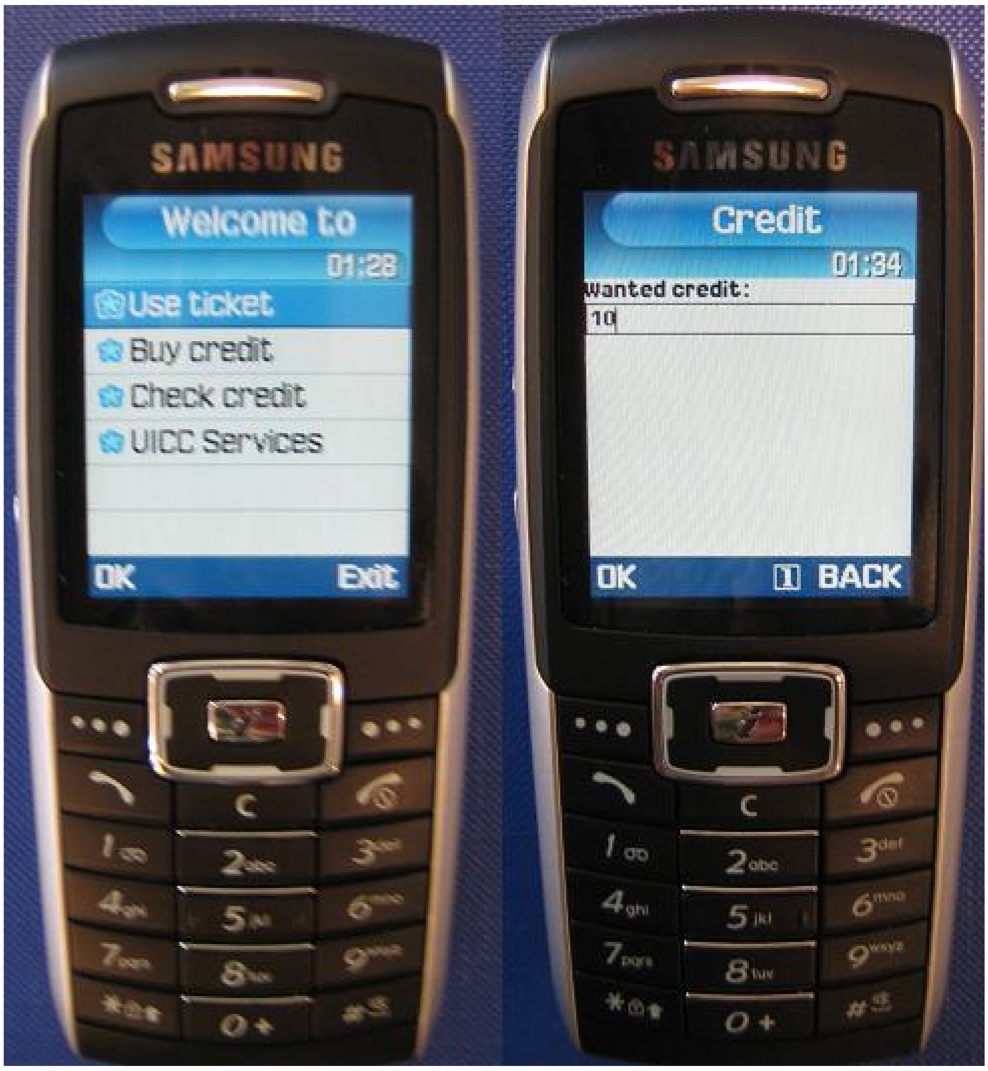

Figure 38: Samsung SGH-X700N

In Figure 38 we can see the UI as it was on the Samsung handset, the handset that the midlet was developed for. The "OK" button is placed on the left side, while the "Exit" button is placed on the right side. In the "Buy credit" menu the credit is entered in a rectangle.

The next handset we tested was the Motorola RAZR V3i (Figure 39). The midlet runs on this handset, though with minor differences from the Samsung handset. The "OK" and "Exit" buttons have changed position, and credit input is entered in an input-field that opens by default when the user tries to change the credit field.

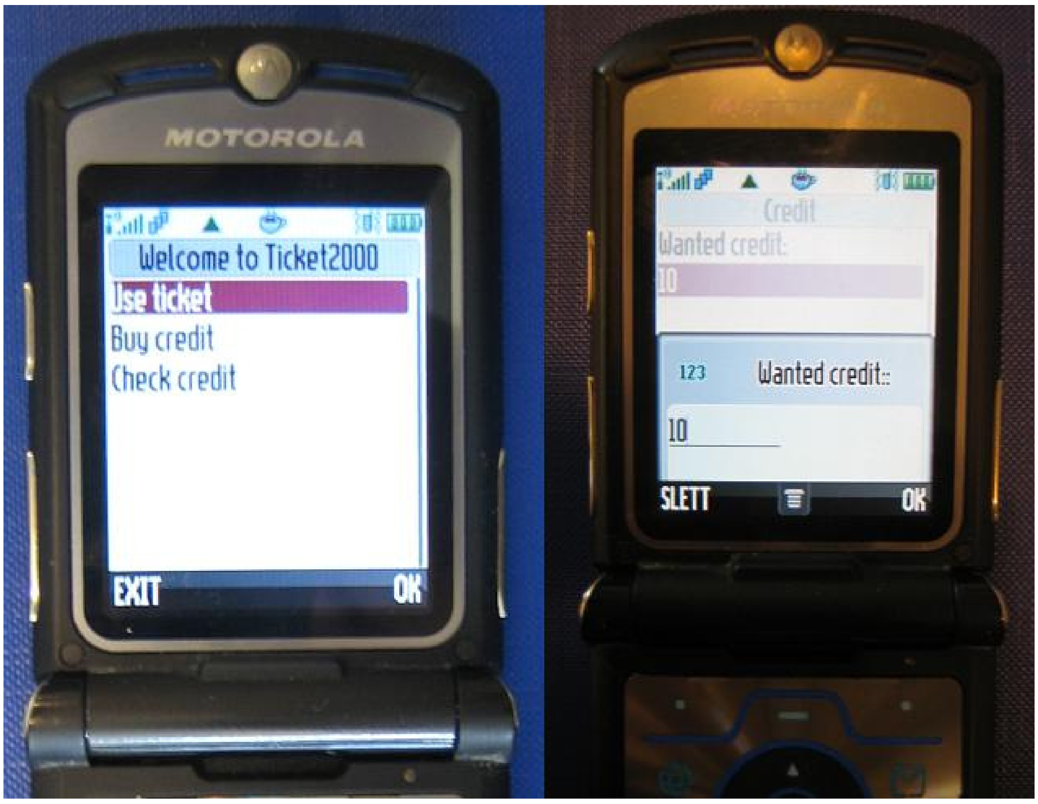

Figure 39: Motorola RAZR V3i

The Java implementation on the Nokia 6680 handset puts the "OK" and "Exit" commands in a separate menu called "Options" on the lower left side of the screen (Figure 40). In addition they add a second default "Exit" command on the lower right side of the screen. The same is done in the "Buy credit" menu, where the "Exit" command is replaced with "Back".

The midlet runs, but the default extra buttons are unnecessary and unwanted. It might seem confusing to have two "Exit" buttons and two "Back" buttons.

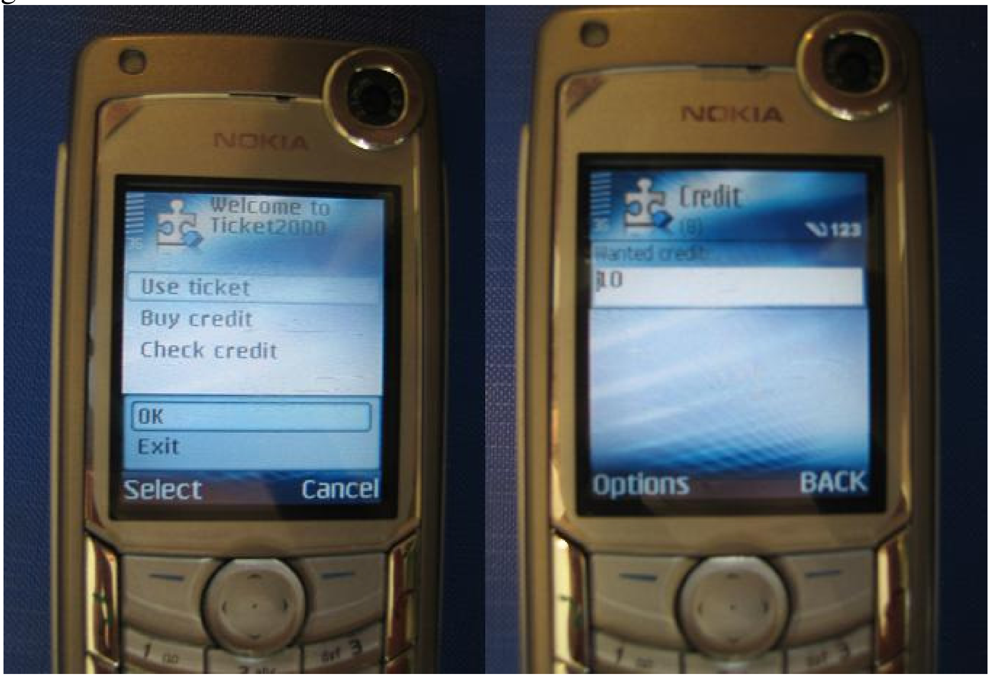

Figure 40: Nokia 6680

On the newer Nokia E65 the UI is almost identical to the Samsung handset (Figure 41). This might seem strange since both the Nokia 6680 and E65 runs a Symbian OS, but they are of different versions, thus the Java implementation is different. The 6680 handset runs the Symbian v8.0 S60 while the E65 runs v9.1 S60.

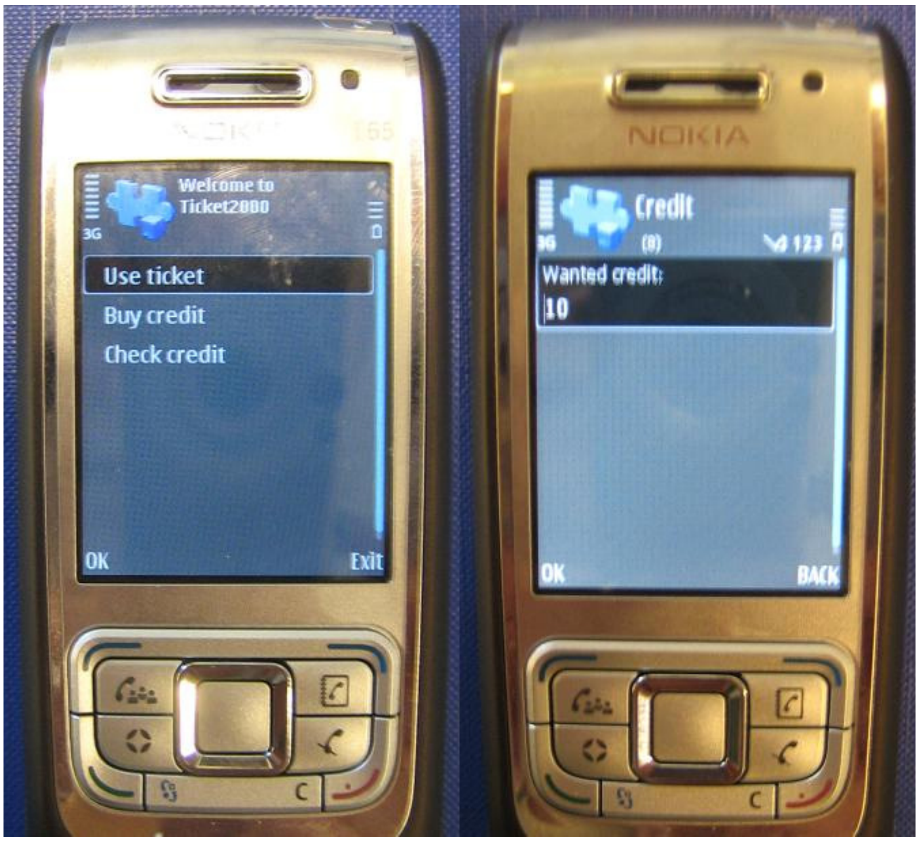

Figure 41: Nokia E65

The last handset we tested was the Sony Ericsson K700i. The UI on this handset (Figure 42) was similar to the Nokia 6680 where the "OK" and "Exit" buttons are placed in a menu on the lower left side of the screen.

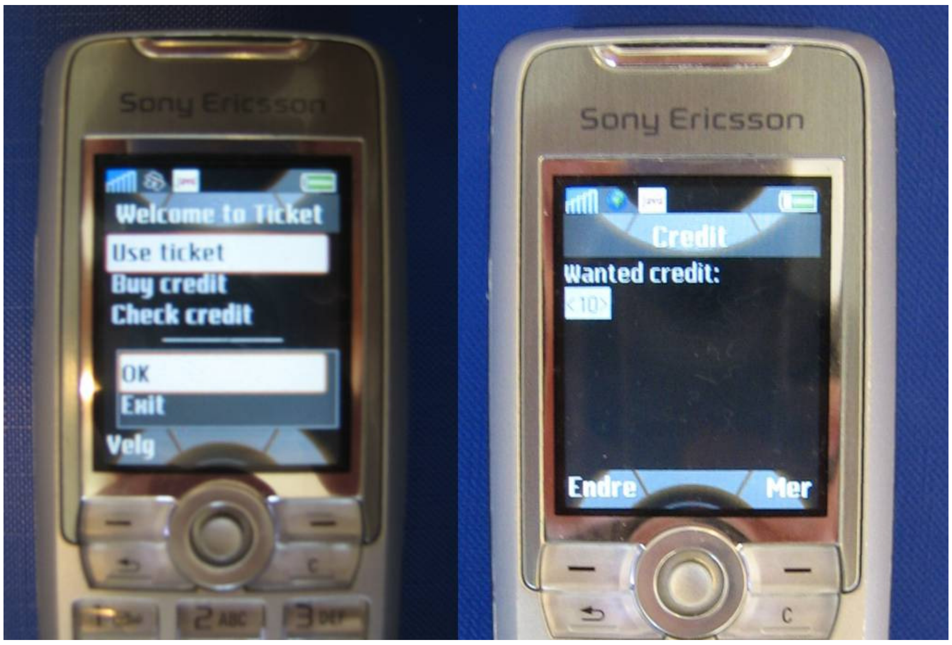

Figure 42: SonyEricsson K700i

Even though all handsets produced a different UI from the same midlet, they all worked and were functional.

6.3.2 Different communication technologies within the same type of service

Use ticket with SMS
The “Use ticket” function worked as expected after the command from the simulated network operator, and “poll” resulted in calling the simulated SMS function in the UICC service. It received the correct interface information, number, port and payload data, and the credit value was subtracted correctly.

Use ticket with NFC
After giving the UICC application a message to use the NFC, and the “Use ticket” function was used, the Smart Card was put in wireless mode on “poll” and ready to be read by the desktop NFC reader and the JCOP command script. The display on the handset asked the user to swipe the handset over a reader, and the credit value was correctly subtracted and the script returned the correct command (0x90).

Buy credit with SMS
The “Buy credit” function worked as expected after the command from the simulated network operator, and “poll” resulted in calling the simulated SMS function in the UICC service. It received the correct interface information, number, port and payload data, and the credit value was incremented correctly

Buy credit with Internet
The simulated network operator gave the correct address and port to the UICC application, and after executing the “Buy credit” and “poll” functions the UICC service connected to Internet through GPRS and transferred the payload-data from the UICC application. The payload-data was received correctly buy the credit server, and it returned a credit value to the UICC application which was incremented correctly.

6.3.3 Different application standards within the same type of services

This test executed correctly. The credit value was preserved, and both the buy and use functions worked with the same interface, address and port as before re-installation of the UICC application. The new application did also follow the new application standard.

6.3.4 Loss of state when users change handsets

This test executed correctly. State was preserved during re-installation of the handset application, and the application executed identically as before the re-installation.

6.4 Summary

Our tests showed that the implementation worked as expected. All implemented functionalities worked without faults.

7 Evaluation

In order to evaluate this design (chapter 4) and implementation (chapter 5), we compare the test results (chapter 6) with the main goal, secondary objectives and finally the requirements (chapter 3).

7.1 Achieving the goal

Our main goal in this thesis was to design a system that facilitates widespread adoption of services and applications on handsets based on various communication technologies.

The system was designed to let the user use services with as little administration from the user-side as possible. The system was designed so that the users should be able to change handset and location without concerning about handset's software platform, different communication technology and application standards, and not concerning themselves about loss of state when changing handset. When services are provided in a simple fashion to the user, it is likely to believe that this will facilitate widespread adoption of services and applications from the user-side.

To facilitate widespread adoption of services and applications on handsets from a developer's standpoint, developers need to be able to do development in a fast and easy fashion. The main problem for developers today is the number of different platforms that are available on handsets. We have tried to find an interoperable platform so developers can spend as little time as possible on re-developing applications and services to fit different handsets.

7.1.1 Platform interoperability

To achieve platform interoperability we identified and used the only software platform on the handset that we know is standardized and existing on every handset: The UICC. However, the UICC alone is not able to communicate with the user, nor external services without the handset. Hence, we added a service on the handset that offered simple UI and the available communication technologies on the handset to the UICC. In addition, we made it possible to put memory consuming and UI specific parts of service as applications on the handset. This was to supplement to the UICC applications with more memory and more advanced UI.

With our designed system, developers can develop a service as an application on a UICC. These applications are platform independent as long as the handset in use has a UICC service as we designed and implanted, or preferably, the CAT standard implemented. CAT offers generic and standardized UI and communication -interfaces to UICC applications, hence solving the challenges surrounding APIs on different software platforms. Currently we do not know of any handsets with CAT implemented, but as long as network operators and service providers encourage handset manufacturers to implement CAT on their handset, new handsets will most likely be made with CAT in the future.

Implementing services as applications on UICC has two challenges that might pose problems surrounding platform interoperability: First, the software platform on the UICC itself. A UICC can have either a native OS or Java Card. Native OS does not offer an interoperable platform since every native OS needs an independent implementation. Java Card on the other hand is a standardized smart card OS, and applications developed for this platform can run on any smart card with Java Card, independent of the smart card manufacturer. Secondly, only the 3G-type of UICCs can support independent applications. UICCs with the 2G architecture do not include other applications other than the SIM and SAT applications. Hence, our solution requires that network operators equip their costumers with 3G UICCs with Java Card in order for our solution to be platform interoperable. By doing this, the network operator will create a standardized UICC for their costumers.

Another feature with our solution that might contradict the platform interoperability – objective is the possibility to place parts of an application on the handset. In our tests from chapter 6.3.1 we installed the UI specific parts of our test application on different handsets to test the interoperability of our ticketing application. The tests showed that even though we implemented our handset-application for the Java ME platform, there were some differences amongst the handsets. All installations on the handsets provided the desired functionality with different outlays of the UI. This did not pose any problems for our test-application, but with a more complex UI we might have experienced problems. Even so, the possibility to place parts of an application on the handset is an extra feature to offer more complex UI for both the developer and the user, and it is not mandatory. Developers can choose to use CAT for UI interface instead. Furthermore, developers only need to re-develop handset-applications to fit different handsets. UICC-applications and their interface are platform interoperable and do not require re-development. Placing as much as possible of an application on the UICC requires less work with re-development of handset applications.

7.1.2 Communication technology standard

We discovered that services might use different communication technology standards to achieve the same goal. Searching locally on the handset can do discovery of the communication technology in use at a given location. An example of this is the CMAB solution from Microsoft for containing a link to Internet services (section 2.5.2). But searching locally on the handset is a processor and memory consuming operation, hence power consuming. Power is a limited resource on handsets because of their battery limitations.

In addition, a connection to a remote service often requires a destination address (it was only the NFC interface in our implementation that did not require an address). This information has to be obtained somehow. Keeping addresses for different services locally on the UICC would have scaled linearly, and this would have posed problems as the number of providers of a type of service rises. UICCs and handsets have limited storage capacity compared to e.g. a network operator's backend servers.

We know that the network operator can obtain knowledge about handsets locations. Combined with knowledge about locations capability to offer services based on a communication technology, the network operator can do the administrative work of keeping a connection to services. The network operator also has the means of communicating to UICC applications through CAT or a UICC service similar to our solution, hence operators can inform the UICC applications about which communication technology to use at any given time.

The drawback of this solution is that network operators need to obtain this information from service providers. This calls for collaboration between network operators and service providers. The solution is not autonomous and requires maintenance as services are removed, added and updated.

7.1.3 Application standard

Varying application standards is a challenge that arises when service providers lack collaboration in standardization, try to optimize their service compared to other providers, or use a different standard to differentiate themselves from other providers. Our solution to this challenge is to give the network operator an administrative role of transparently replace application standards as they change, as for communication technology standard.

In our solution, the application standards towards services are implemented in the UICC applications (in this sense: typically communication protocols, data-formats etc). By placing application standards on the UICC, the network operator only needs to keep one copy of every application standard since the UICC is a generic platform. If application standards where implemented on the handset, the network operator would have needed to have several copies of one application standard, where every copy was dependent on the handset and it's application platform.

However, this once again requires that the UICC is equipped with Java Card. But as earlier stated: Network operators can choose to create a standard UICC with Java Card for their users by equipping the users with such an UICC. Costumers are bound to follow an UICC standard that the network operator chooses since the UICC is the property of the network operator.

As for the solution to the challenge surrounding communication technology standard, this solution requires collaboration between network operators and service providers. Service providers must be willing to either develop their application standard for Java Card UICCs and give this to network operators, or to reveal their application standard to network operators and permit the use of these.

7.1.4 State

Users of handsets change their handsets relatively often (section 2.4.5), and this poses challenges surrounding the preserving of state for the users and their services. By using the UICC for applications we can store state in these applications, and state can be brought from handset to handset. It would have been possible to store such state at a remote server, but this creates a lot of overhead traffic in the network. Furthermore this could have created privacy issues around personal information stored in the state information.

However, we need to store state outside the UICC when UICC applications are replaced in order to update application standards. We could have stored this state locally on the handset during application replacement, but the handset is not considered to be a safe storage medium with respect to malicious software. In addition CAT does not have support for storing data for the UICC applications, hence temporally storing state on the handset eliminates the possibility for using CAT in our system.

7.2 Requirements

7.2.1 Platform interoperability

We stated three requirements to achieve platform interoperability. The first requirement stated that the whole or parts of a service should be placed on an interoperable platform connected to the handset. We identified the UICC as an interoperable platform since it is always present in the handset architecture. To ensure that applications can be deployed on all UICCs from different producers, they have to run the Java Card OS. If network operators equip their users with Java Card UICCs, we will have an interoperable platform for services, and R1.1 is fulfilled. In addition it is possible to place handset specific parts of an application on the handset in accordance with R1.1.

The UICC is a transparent and mandatory part of the handset architecture, and it does not require any installation or administration to equip a handset with a given UICC. CAT should be a pre-deployed feature on handsets when this standard is incorporated in handset software platforms. Currently, most 2G handsets have SAT and most 3G handsets have USAT. We believe that most handsets produced in the near future will have CAT. If so CAT and UICC will be transparent for the user; hence we fulfil R1.2 that states that the interoperable platform should be transparent for the user.

Our last requirement stated that the interoperable platform should support at least 5 handsets with different OS'. We failed to meet this requirement from the very beginning since we were not in possession of any handset with an accessible UICC with Java Card. Instead we simulated such an UICC with a Java Card-smart card connected to the Samsung SGH-X700N and it's NFC radio. This smart card used the same interface towards the handset as an UICC, the ISO 7816 interface. Since both the platform (Java Card OS) and the interface (ISO 7816) were the same as for a Java Card UICC, this simulation does not cause any implications.

By definition, all handsets support UICC regardless if they have Java Card OS or any other OS since they all use the same generic interface towards the UICC. Hence, this requirement is by definition fulfilled based on the platform itself. But applications does not have any support for communication with the handset, hence external services, since handsets does not have CAT nor have accessible UICCs. On the other hand 3G handsets have support for USAT that offers the same services for USAT-applications on the UICC as CAT offers to independent UICC-applications. But with USAT it is not possible to place handset-specific parts of an application on the handset, since USAT-applications are inaccessible from handset applications.

In our test of the implementation we tested our handset-specific implementation on five different handsets with different OS'. The UI was represented to the user in different ways on the 5 handsets with the Java Me platform. But at the same time the test showed that the applications worked with the same desired functionality on all handsets.

7.2.2 Communication technology standard

We implemented a UICC service that was based on CAT since we were not in possession of a handset with CAT. UICC services offered an Internet, NFC and SMS interface (the interfaces possible to implement on the Java ME platform on the Samsun handset) to the UICC application through the generic ISO 7816 interface and standardized APDU commands. We did not implement the Bluetooth interface in UICC service, and came on communication technology short of R2.1. R2.1 stated that all communication technologies on the handset should be accessible to applications on the interoperable platform.

If we were in possession of a handset with CAT, we would have fulfilled R2.1. CAT offers an interface for all communication technologies on a given handset, and CAT is expandable to include new communication interfaces as new technologies are added to handsets. With CAT we would have fulfilled R2.1.

R2.2 stated that any known pair of internal and external communication technology that could offer an obtained service should be available transparently to the user. We designed our system so that the network operator lets the UICC applications know which communication technology to use at any given time and location. This is based on the location of the handset and the locations services availability through communication technologies. This fulfils R2.2.

In the implementation we simulated the network operator to test this design. We sent messages to the UICC application about change of communication technology, and the UICC application changed the communication technology accordingly. Since we did not have network privileges to determine handsets locations, we could not automatically and transparently change network technology. In our handset application we had a menu that sent network operator messages to the UICC application. This menu was separated from the ticketing menu, and the technology change messages were from a ticketing-user standpoint sent transparently. For the UICC it is irrelevant which source these messages came from as long as they arrive in the right format. Hence, we can say that we partially fulfilled R2.2 in our implementation, and only partially because of the fact that we did not possess network operator's privileges. The implementation showed that this would have worked if all practical circumstances were present (i.e. network operator rights for location-based-services and functional receiving of SMS).

7.2.3 Application standard

We had one requirement to handle the challenge surrounding application standard: Application standards implemented on the interoperable platform should be transparently replaced on demand from a remote service. We solved this by designing a system where the network operator (the remote service) replaced the UICC applications (applications on the interoperable platform) based on application standards in use for services on a given location. Before replacement we designed it so that the current application sent its state to the network operator before replacement. After replacement, the network operator sent this state back to the new application. The design is in accordance with R3.1

In the implementation we could not receive SMS properly, and we decided to transfer new applications to the smart card with the help of Eclipse with the JCOP plug-in. The state was transferred to the simulated network operator placed locally on the handset before replacement, the new application transferred through JCOP, and the new state transferred to the new application. From a ticketing-user standpoint, this was done transparently since the ticketing application had the same UI and functionality after change of application standard, and for the UICC application it is irrelevant where these messages originated as long as the packet-format is correct.

7.2.4 State

Our last requirement stated that users should be able to bring state about applications and the user form handset to handset without overhead administration from the user side. We achieve this by using a UICC as the application platform in the design. This way applications and state can be brought from handset to handset by simply inserting the UICC in the new phone, and we fulfil R4.1.

In our implementation this is not achieved since we only simulate a UICC. The smart card we use on the Samsung SGH-X700N is not possible to remove from the handset without picking the handset apart.

8 Conclusion

The use and development of services on mobile handsets are put up against several challenges because of issues surrounding a wide range of actors on all levels of the handset architecture and their number of standards or lack of standards. We identified four challenges surrounding services on mobile handsets:

- Lack of interoperability on software platforms
- Lack of standardization of communication technology for accessing services
- Lack of application standards
- Loss of state about services when users change handsets

The main goal of this thesis was *to design an architecture that facilitated more widespread development and use of services on the mobile handset* to handle these challenges.

We started theses by taking a look at the basics of the mobile handsets features and software platforms. Next we identified the challenges that surface as services are developed and used. These challenges are known challenges amongst actors in the mobile handset industry, and we continued by looking at some of the initiatives these actors have made to conquer some of these challenges. We found that none of these initiatives handled all the challenges; neither was no more than one of these initiatives implemented on any handset at the time this thesis was written.

We designed our own architecture to handle these challenges on the basis of our main goal, secondary objectives and requirements. Our designed architecture handled the lack of interoperability on handsets and loss of state by placing applications on an UICC with Java Card OS. These applications had interfaces to communication technology on the handset through a UICC service on the handset. This UICC service was based on CAT, and can be replaced in the architecture by CAT when CAT is implemented on handsets. In addition we opened up for placing handset-specific parts of an application on the handset. These parts can offer a more complex UI, and relieve the UICC of memory consuming parts of an application.

To handle the challenges surrounding lack of standardization of communication technologies for accessing services and the lack of application standards, we gave an administrative role to the network operator. The network operator uses its knowledge about handsets' locations combined with knowledge about locations' service availability through communication technologies and application standards to change the behaviour of UICC applications. The operator does this by telling the UICC applications what network interfaces and addresses to use as services are available through new communication technologies, and replace UICC applications as application standards change. However, our solution depends on the network operator obtaining knowledge about locations and their service availability through application standards and communication technologies. This calls for collaboration between network operators and service providers.

As we started to implement the design, we re-discovered the challenges earlier identified on the handset that we used for our implementation (see section 5.1). Combined with the fact that we were neither in possession of network operator rights, nor a handset with CAT or an open 3G UICC, our implementation was limited compared to the design. Another feature that we did not implement according to our design was the actions made by the network operator as users changed handsets, and CAT and handset parts of an application were missing. In the design we specified that the network operator should transfer the UICC service and such handset specific applications to the handset when users changed handsets (section 4.2.2). Because of time limitations we did not implement this.

Nevertheless, we were able to show that our design is possible to implement in a real system since the mechanism we did not implement (such as location-based services, UICC with Java Card and SMS receiving) exist and are possible to implement in our designed system in co-operation with a network operator. CAT is not implemented on handsets as of today, but an implementation is believed to be around the corner.

The final conclusion will be that we approached an architectural-design that facilitates more widespread development and use of services on the mobile handset. This was realized through simplified development and use of services. All elements in our design are realizable through current solutions (i.e. Java Card UICC and location based services), standards believed to arrive soon (i.e. CAT), and further standardization among actors (i.e. agreement on protocols between handset and UICC applications). We implemented this design as far as possible given our resource-limitations.

References:

1. Physorg.com, *World's first NFC enabled mobile product for contactless payment.* 2005 Available from: http://www.physorg.com/news2984.html.
2. 3G.co.uk, *World's First Mobile Phone with Zigbee solution.* 2004 Available from: http://www.3g.co.uk/PR/December2004/8787.htm.
3. Wikipedia, *Service.* 2007 Available from: http://en.wikipedia.org/wiki/Service.
4. Princeton, W., *Service.* 2007 Available from: http://wordnet.princeton.edu/perl/webwn?s=service.
5. Alex Varshavsky, M.C., Eyal de Lara, Jon Froehlich, Dirk Haehnel, Jeffrey Hightower, Anthony LaMarca, Fred Potter, Timothy Sohn, Karen Tang, and Ian Smith, *Are GSM phones THE solution for localization.* IEEE Workshop on Mobile Computing Systems and Applications (HotMobile 2006),, 2006 Available from: http://www.cs.toronto.edu/~walex/papers/are_gsm_phones_the_solution_for_localization_wmcsa2006.pdf.
6. Stefan Steiniger, M.N.a.A.E., *Foundations of Location Based Services.* 2006 Available from: http://www.geo.unizh.ch/publications/cartouche/lbs_lecturenotes_steinigeretal2006.pdf.
7. Microsoft, *Windows Mobile.* 2007 Available from: http://www.microsoft.com/windowsmobile/default.mspx.
8. Symbian, *Symbian OS.* 2007 Available from: http://www.symbian.com/.
9. Palm, *Palm OS.* 2007 Available from: http://euro.palm.com/europe/en/index.html.
10. SavaJe, *SavaJe OS.* 2007 Available from: http://www.savaje.com/.
11. LinuxDevices.com, *Linux on a roll in mobile phones.* LinuxDevices.com, 2005 Available from: http://www.linuxdevices.com/articles/AT3908389811.html.
12. Symbian, *Symbian OS user interfaces.* 2005
13. Wikipedia, *Run-time System.* 2007 Available from: http://en.wikipedia.org/wiki/Runtime_system.
14. Microsystems, S., *Java™ Platform, Micro Edition (Java ME) Overview.* 2007 Available from: http://java.sun.com/javame/index.jsp.
15. Qualcomm, *The Brew Solution.* 2007 Available from: http://brew.qualcomm.com/brew/en/about/about_brew.html.
16. Microsystems, S., *The Java ME Platform — the Most Ubiquitous Application Platform for Mobile Devices.* 2007 Available from: http://java.sun.com/javame/index.jsp.
17. Microsystems, S., *Java ME: De-fragmentation.* 2007 Available from: http://developers.sun.com/techtopics/mobility/reference/techart/design_guidelines/.
18. ISO, *ISO 7816.* 1995-2007 Available from: http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=14738&showAllRelItems=y.
19. ETSI, *Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface (GSM 11.11 version 5.10.1 Release 1996).* ETSI Publications Download Area, 1998 Available from: http://webapp.etsi.org/exchangefolder/ets_300977e05p.pdf.
20. ETSI, *Universal Mobile Telecommunications System (UMTS);UICC-terminal interface;Physical and logical characteristics (3GPP TS 31.101 version 6.5.1 Release 6)* ETSI Publications Download Area 2006 Available from: http://pda.etsi.org/pda/home.asp?wki_id=bO'ZXdWpkVACECHAjzI.q.

21. Egeberg, M., *The mobile phone as a contactless ticket.* 2006 Available from: http://www.diva-portal.org/diva/getDocument?urn_nbn_no_ntnu_diva-1048-1__fulltext.pdf.
22. NXP, *Philips and Samsung join forces for the development of new mobile devices based on Near Field Communication (NFC).* 2004 Available from: http://www.nxp.com/news/content/file_1083.html.
23. Nokia, *Near Field Communication (NFC).* 2007 Available from: http://www.nokia.com/A4305081.
24. Gemalto, *Technical description of the USB proposal.* Gemalto - Standardization & Innovoation,
25. GlobalPlatform, *Card Specification v2.2* 2006 Available from: http://www.globalplatform.org/specificationview.asp?id=card.
26. CardTechnology.com, *A Three Way Race.* 2005
27. Microsystems, S., *Runtime Environment Specification, Java Card™ Platform, Version 2.2.2.* 2006 Available from: http://java.sun.com/products/javacard/specs.html.
28. ETSI, *Smart Cards;UICC Application Programming Interface (UICC API) for Java Card (TM) (Release 7)* ETSI Publications Download Area 2007 Available from: http://pda.etsi.org/pda/home.asp?wki_id=,LNhQ877x9BDGJFCxbbNv.
29. Giesecke&Devrient, *UniverSIM.* Available from: http://www.gi-de.com/pls/portal/maia.display_custom_items.DOWNLOAD_SEEALSO_FILE?p_ID=5531&p_page_id=55292&p_pg_id=42.
30. Scott B. Guthery, M.J.C., *Mobile Application Development with SMS and the SIM Toolkit.* 2002
31. ETSI, *Digital cellular telecommunications system (Phase 2+) (GSM);Specification of the SIM application toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface (GSM 11.14 version 5.9.0 Release 1996)* ETSI Publications Download Area 1998 Available from: http://pda.etsi.org/pda/home.asp?wki_id=Zo.FM6sE57BIEHDN3KSo.
32. ETSI, *Universal Mobile Telecommunications System (UMTS);UICC-terminal interface;Physical and logical characteristics (3GPP TS 31.101 version 6.5.1 Release 6)* ETSI Publications Download Area 2006 Available from: http://pda.etsi.org/pda/home.asp?wki_id=6rfxdu8czurtvtyrWqtvW.
33. Wikipedia, *3G.* 2007 Available from: http://en.wikipedia.org/wiki/3g.
34. ETSI, *Digital cellular telecommunications system (Phase 2+);Universal Mobile Telecommunications System (UMTS);Universal Subscriber Identity Module (USIM) Application Toolkit (USAT) (3GPP TS 31.111 version 7.6.0 Release 7)* ETSI Publications Download Area 2007 Available from: http://pda.etsi.org/pda/home.asp?wki_id=nuY2Xu_obPJLPJRSQRv-O.
35. Telenor, *Lad med mobilen - få ekstra ringetid.* 2003 Available from: http://www.telenor.no/om/presse/aapen_linje/2003_04/aktuelt_ladmed.shtml.
36. FFS, S.C., *MobileTicket.* 2007 Available from: http://mct.sbb.ch/mct/en/reisemarkt/billette/mobile-ticket.htm.
37. Forum, N., *NFC.* 2007 Available from: http://www.nfc-forum.org/home.
38. ZigBeeAlliance, *ZigBeeAlliance, Wireless Control That Simply Works.* 2007 Available from: http://www.zigbee.org/en/index.asp.
39. Chu, S., *How Would You Like to Pay for That? Cash, Card or Phone?* 2006 Available from: http://www.corporate.visa.com/md/nr/press291.jsp.
40. Nokia, *Nokia 6131 NFC phone taps into mobile payment, ticketing and local sharing* 2007 Available from: http://www.nokia.com/A4136001?newsid=1096858.

41. CardTechnology, *Telecom Italia To Launch ZigBee SIM Trial.* CardTechnology, 2007 Available from: http://www.cardtechnology.com/article.html?id=20061107G5FWMUBW.
42. IEEE, *IEEE 802.15 Working Group for WPAN.* IEEE802.org, 2007 Available from: http://ieee802.org/15/.
43. Network, S.D., *Getting Started.* 2007 Available from: http://developer.symbian.com/main/getstarted/.
44. MOBYMEMORY, *Overview of different memory card types for sale.* 2007 Available from: http://www.mobymemory.com/.
45. ETSI, *Want to know about ETSI?* 2007 Available from: http://www.etsi.org/about_etsi/5_minutes/home.htm.
46. Association, G., *About GSM Association.* 2007 Available from: http://www.gsmworld.com/about/index.shtml.
47. OpenMobileAlliance, *About Open Mobile Alliance.* 2007 Available from: http://www.openmobilealliance.org/about_OMA/index.html.
48. OMTP, *OMTP Objective Summary.* 2005
49. OMTP, *Technology Agnostic Core Software Platform.* 2005
50. OpenMobileAlliance, *OMA Service Environment.* 2007 Available from: http://www.openmobilealliance.org/release_program/ose_v1_0.html.
51. ETSI, *Smart Cards;Card Application Toolkit (CAT) (Release 7).* ETSI Publications Download Area 2007 Available from: http://pda.etsi.org/pda/home.asp?wki_id=qLK-gaYyfuVXadZZGBZ3z.
52. Heiss, J.J., *Open Sourcing Java Platform, Micro Edition: A Conversation With Sun's Senior Director of Mobile & Embedded Platforms, Shannon Lynch.* Sun Developer Network, 2006 Available from: http://java.sun.com/developer/technicalArticles/Interviews/lynch_qa.html.
53. Microsoft, *Introducing Windows Mobile 5.0.* 2006 Available from: http://msdn.microsoft.com/mobility/windowsmobile/howto/windowsmobile5/.
54. NXP, *Business News From Philips.* Press Release NXP, 2006 Available from: http://www.nxp.com/news/content/file_1216.html.
55. Eclipse, *Eclipse - an open development platform.* 2007 Available from: http://www.eclipse.org/.
56. IBM, *JCOP Tools.* 2007 Available from: http://www-306.ibm.com/software/wireless/wecos/tools.html.
57. Microsystems, S., *SDI 010 Secure Dual Interface Reader Contact and Contactless.* 2005 Available from: http://www.scbsolutions.com/Brochures/SDI010.pdf.

Figure listings

Table listings

Code Example listings

Appendix A –Samsung SGH-X700N Java-specifications

R&D Team			SUWON 2.5G
Model Name			SGH-X700N
Region		Europe	O
		Americas	X
		Asia Pacific	O
		China	X
		Africa	O
Operator			X
Processor			ARM9
CLDC			1.1
MIDP			2.0
Networking	Protocol	HTTP	O
		HTTPS	O
		SOCKET	O
		SECURE SOCKET	O
Proprietary API	Audio	MMF	O
		MIDI	X
	Vibration		O
	Backlight		O
	SMS		O
	NFC		O
	Acceleration		X
Supported JSRs	JSR-75	FC	X
		PIM	X
	JSR-82 Bluetooth		X
	JSR-120 WMA	SMS	O
		CBS	X
	JSR-135 MMAPI	Audio Type	MIDI
		Image Type	PNG
		Video Type	X
		Audio Playback	O
		Audio Streaming	X
		Video Capture	X
		Video Playback	X
		Video Streaming	X
		Video Resizing	X
		Video Positioning	X
	JSR-184 Mobile 3D		X
	JSR-205 WMA 2.0		X
DRM support			X
Image file formats supported in Java			PNG
Full screen size			176x220
Soft Key Size			23
Soft Icon Size			N/A
Colour Depth		Java	65536
		Display	262144
Heap Size		For running VM	1.1 MB
		For images	540 KB
Max JAR size			300 KB

Total Storage		4 MB
# of Max Midlet		N/A
# of Default Midlet		N/A
# of PreInstalled Midlet		N/A
RMS size		Up to 4 MB
OK/Fire		-5
UP		-1
DOWN		-2
LEFT		-3
RIGHT		-4
LEFT SOFT		-6
RIGHT SOFT		-7
C		-8
*		42
#		35
0		48
1		49
2		50
3		51
4		52
5		53
6		54
7		55
8		56
9		57
Font (Alphabets)	Small	13
	Medium	17
	Large	19
Font (Chinese characters)	Small	N/A
	Medium	N/A
	Large	N/A
T9 input - French		O
T9 input - German		O
T9 input - Russian		O
JAR Download Protocol		HTTP
SDK		
Functionality		GSM