An Architecture for the Management of Smart Cards by Mobile Devices Using Java Technologies

Theodore K. Apostolopoulos Ilias S. Kapetanakis George Oikonomou
Computer and Communication Systems Laboratory
Department of Informatics, Athens University of Economics & Business
76, Patission str., 104 34 Athens, Greece
Tel.: +30 210 8203234 Tel.: +30 210 8203158 Tel.: +30 210 8203155
tca@aueb.gr ikapet@aueb.gr g.oikonomou@aueb.gr

Abstract

Smart Cards are cards with memory and a processor, ideal for authentication and secure applications. Mobile devices have an extraordinary spread; People carry them along at all times. The ability to combine the features of those two devices is quite compelling. We used technologies of the Java family in order to achieve this.

In our work, we designed an architecture for the management of smart cards by mobile devices. The system is built up from three components, the Java Card applet, the mobile application and the Java Card-Mobile Device Gateway.

For the purposes of our work and as proof of concept, we developed a sample electronic wallet application. Finally, we extended the proposed system's functionality and described a business case, where the system could be used in the future.

1. Introduction.

Smart Cards are cards with memory and a processor. They are ideal for authentication and for transferring sensitive personal information, so they can guarantee security in our transactions. Mobile devices have an extraordinary spread; People carry them along at all times and use them more and more. Thus, mobile devices can offer ease of use and simplicity forcing the owner to bring along an extra device.

The ability to combine the features of those two devices is quite compelling, to say the least. Unfortunately, no standard way of direct connection has been made available until now, so we decided to develop a gateway, which would have the role of the intermediary between a smart card and the mobile device and could solve the problem of direct communication.

Sun Microsystems has developed two technologies for the Java Family, one for smart cards and one for mobile devices. These are Java Card [1], [3], [9], [12] and Java 2 Platform Micro Edition (J2ME) [2] respectively. Both of them keep many of the features of the Java language such as productivity, security, robustness, tools and portability, allowing the use of the Java technology in smaller devices.

In our work, we developed an application for mobile devices that manages a smart card. The system has the following distinct components:

- A Java Card Applet
- A mobile terminal application that manages the card
- A Java Card – Mobile Device Gateway. Implementing this was the most challenging part of the development process. It will be described in detail further-on in this document.

In section 2 of this document we present the reader with an overview of the design of "A Complete Java Card Application". Later on, in section 3, we provide an end to end description of the system's components. To achieve this we are based on a typical electronic wallet example. In section 4 we present a sample communication scenario between the different parts. Section 5 presents a case where the system may be used in a typical commercial transaction. At last, in section 6, we deduce our conclusions and discuss open issues.

2. A complete java card application.

A complete Java Card application is a system for the management of a smart card, implemented with the Java Card technology. The card resides in a smart card reader and contains one or more different applications, which are called applets. An application, which is

Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05)
1530-1354/05 $20.00 © 2005 **IEEE**

called "terminal application", is responsible for the management of the card and is hosted outside the card reader.

In order to achieve communication between the smart card and the terminal application, a logical data packet is used. The packet is called an Application Protocol Data Unit (APDU) [1],[12]. The Java Card Framework receives and forwards all incoming command APDUs to the appropriate applet. In sequence, the applet processes them and returns a response APDU. Thus, the card has a passive role waiting to serve the APDU commands that arrive. Further explanation of the APDU command and response is out of the scope of this paper and can be found in [8].

There are many communication models between a terminal application (also called host application) and a Java Card applet. All of them use the APDU based message model. Such models are Java Card RMI (JCRMI) [4], the Security And Trust Services API (SATSA) [5],[13] and the Open Card Framework (OCF) [3], [6], [7]. SATSA is a security Application Programming Interface (API) for mobile devices but, at the same time, it offers methods for direct APDU communication between a mobile device and a terminal application. Unfortunately at the time of development of the system, SATSA was under review and was not officially released until September 2004. The solution was the Java Card-Mobile Device Gateway we mentioned above.

The Open Card Framework (OCF) is a standard framework that provides an architecture and a set of APIs that enable the development of a system, independent of card vendor and terminal application. The only things that practically need be done, for developing an OCF-based solution, are as follows. Firstly, an OCF driver for the smart card and card reader we are using must be provided by their vendors. Then we have to implement three components according to our applets and their functions: the CardServiceFactory for each Java Card, the CardService for each applet and an Opencard.properties file for the association between the CardServiceFactory and the card reader. Thus, the OCF can communicate with the card reader by using the appropriate driver and is aware of the different services that have been implemented for a specific card.

3. A system for managing a smart card from a mobile application.

In our work, we developed an application for mobile devices that manages a smart card. The system is built up from three components, the Java Card applet, the mobile application and the Java Card-Mobile Device Gateway. In this section we analyze each one of the distinct components. We describe the system's architecture based on the typical example of an electronic wallet. The functions that it includes are: PIN verification, PIN update, credit amount, debit amount and get balance. Our main goal is to describe the entire system giving emphasis to the communication between the different parts and not the smart card applet itself. This is the reason we have chosen to use such a simple example.

The mobile application is a J2ME application that uses a simple user interface for the navigation through the different functions. The mobile application implements all the functions defined inside the Java Card and it controls errors that might show up, before sending the APDU to the card.

The development and emulation of the Java Card and the mobile device application were made with the Java Card Development Kit 2.1.1 (JCDK) [10] and the J2ME Wireless Toolkit 2.0 [11] respectively.

As for the communication between the mobile device application and the gateway, we decided to use a simple protocol, over UDP sockets. The J2ME application, via GPRS or some other way, dispatches messages to the Gateway. In sequence, the Gateway provides for the communication with the Java Card. The reverse process is followed for transferring responses from the card to the mobile device.

The message of the mobile application consists of two parts, as depicted in Figure 1. The first part is the first byte of the message, which defines the function that the whole message refers to. The second part is the data field, if any data exists. For example, in the 'Verify PIN' function, the first byte is "0" and the data field carries the PIN that's going to be verified.

HEADER	**DATA FIELD** (Optional)
1 Byte	0..n bytes

Figure 1 – Structure of the mobile application message

The gateway must receive messages from the Mobile Device Application, convert them to an appropriate format and forward them to the Java Card, while it must also be able to do the reverse process. The messages from the application contain APDU commands for certain functions of the card and the messages from the card contain APDU responses to previous commands. In addition we want the gateway to be independent of the device manufacturers.

So, firstly we need a socket server that will accept the messages from the mobile device application and will forward them to the card. That is the way the

gateway communicates with the mobile application. The other thing we want to achieve is the communication between the Java Card and the gateway. The solution to this is the Open Card Framework.

Practically, we implemented just a Card Service Factory and a Card Service for our wallet Java Card. In the Card Service Factory, we declare the Card Services that exist on the card. Particularly in our system, only the walletCardService exists. In the walletCardService we constructed a method for every function of the card, so that when we wish to communicate with the card all we have to do is call the corresponding method of the OCF's walletCardService. The communication with the card reader is OCF's responsibility and the whole process is transparent to the end user.

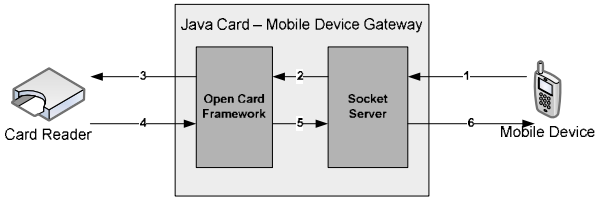

Figure 2 – Steps of communication between the mobile device and the java card

The following steps, depicted in Figure 2 above, take place for a communication between the mobile device application and the wallet Java Card:

1. The mobile device application sends a request to the gateway as a message.
2. The server socket at the gateway receives the request, extracts the first byte and according to it, calls the corresponding method of the walletCardService.
3. The OCF takes control and communicates with the card by sending the appropriate APDU command.
4. The card processes the APDU response and sends an APDU response to OCF.
5. The OCF forwards this response to the socket server, which first constructs a message based on the response and then sends it to the mobile device.
6. The mobile device receives the response, which shows up in the mobile device's screen.

4. A sample scenario.

We will show how point-to-point communication is achieved. Let's assume we want to grant access to our wallet so we will be able to perform debit and credit operations. Entering the mobile application we will be asked to enter our PIN. After entering the PIN "1234" and pressing the send button, the application will construct the message "01234". This consists of the first byte "0" that refers to the 'Verify PIN' function. The rest of the message is the data bytes. The Gateway will receive the message and the socket server will extract the first byte to find out which function it refers to. Then it will call the 'Verify PIN' method from the OCF by passing it as argument the data field of the message. Finally, the OCF takes control and constructs the appropriate APDU command to send to the card.

The reverse procedure is followed in order for the card to send a response to the mobile application. If the PIN has been correctly inserted, the Java Card constructs an APDU response without a data field but with the status word that means that the processing of a command has been completed successfully. The OCF takes the response and hands it over to the server. In turn the server creates an appropriate response and sends it to the mobile application. The mobile application informs the user. The same process is followed every time the mobile device sends a message to the card. The socket server at the gateway keeps a log with the messages that the mobile device and the card are exchanging.

5. A business scenario.

In the following paragraphs we present a typical commercial transaction scenario, in which the system might be used in the future.

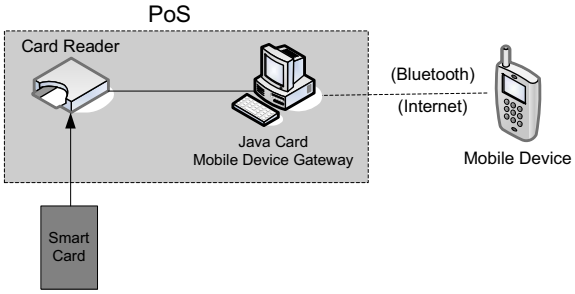

Figure 3 - A business scenario

Imagine a merchant store that accepts smart credit or debit cards (or even electronic wallets). In this case the merchant's Point of Sales (PoS) would consist of a card reader and an application similar to our gateway. Needless to say, the gateway we presented in the previous sections would have to be re-coded in order to provide the required functionality. However, the concept remains essentially the same. On the front-end, all that is needed is a mobile telephone with the card management application. The above scenario is depicted in Figure 3.

The user enters the store and decides to make a purchase with his smart card. He hands the card over to the store's employee who inserts it in the card reader. The user activates the card management application in his mobile telephone and performs the following tasks:

1. Establishes a connection with the gateway at the merchant's PoS.
2. Selects the relevant card product (see below)
3. Enters his PIN.
4. Authorizes the debiting of a specified, merchant-provided amount.
5. The transaction is authorized (or declined) as per the normal procedures that apply for chip cards.

Communication between the mobile node and the gateway may be achieved through a Personal area network, such as Bluetooth or through the internet, via GPRS or UMTS.

Each of the above options has advantages. In the former case, the user activates Bluetooth, scans for devices and then initiates communication with the merchant's PoS. In the later case, the user would have to provide the gateway's hostname or IP address. Furthermore, the user would have to pay the cost for activating the GPRS service for his mobile telephone and he would be charged by the provider with the cost resulting from sending data over the GPRS network. In the case of the proposed business model, moving towards the 'Bluetooth' solution seems more attractive, than the GPRS or UMTS choice. On the other hand, one could name a number of other business models, where communication over the internet would be a far better, or even the only possible choice.

6. Conclusions – open issues.

In this paper we presented a system for managing a smart card from a mobile device. We used technologies from the Java family and we came along with the difficulty of communication between the two devices. For that purpose we developed a Java Card – Mobile Device Gateway that is smart card and mobile device independent.

Inside the Java Card, data are secure but that is not the case outside the card. We examined the technical aspects of the system but we didn't deal with security issues in the communication between the different components. In order to make the system viable as a real-world application, adding security is of the essence. Entering the PIN from a, generally, non-secure device, such as mobile telephone and sending data via a public network, are vulnerabilities we cannot ignore. In addition the mobile device's Operating System may be the target of keystroke logging Trojans and the communication could become the target of various attacks. Therefore, a lot of work should be done on mobile devices, before they can be considered secure enough. Using the standard methods that Java offers, is one approach to the problem. Furthermore, using more sophisticated security techniques, would result in a more robust and bullet-proof system.

In addition, the Security and Trust Services API adds security enhancements to mobile devices and provides for direct communication between mobile devices and Java Cards. Using it, we are able to develop many useful systems such as SIM cards with embedded smart card applets or electronic commerce solutions where, for example, someone might make his transactions from his mobile telephone while he pays with a smart card that is connected to it.

7. References.

[1] C. Enrique Ortiz , "An Introduction to Java Card Technology", May 2003

[2] Sun Microsystems, "Java 2 Platform, Micro Edition Datasheet"

[3] Fodor O. and Hassler V. , "JavaCard and OpenCard framework : A Tutorial", *7th IEEE International Conference on Emerging Technologies and Factory Automation*, 18-22 Oct 1999, pp. 13-22

[4] Sun Microsystems, "Java Card™ 2.2 RMI Client Application Programming Interface", June 2002

[5] Java Community Process, "Security And Trust Services API", May 2004

[6] IBM, "OpenCard Framework General Information Web Document", Open Card Consortium, 1998

[7] IBM , "OpenCard Framework 1.2 Programmer's Guide", Open Card Consortium, 1999

[8] Zhiqun Chen, "How to write a Java Card applet: A developer's guide"

[9] Zhiqun Chen, "Understanding Java Card 2.0"

[10] Sun Microsystems "Development Kit User's Guide, For the Binary Release with Cryptography Extensions, Java Card™ Platform, Version 2.2.1"

[11] Sun Microsystems , "User's Guide, Wireless Toolkit Version 2.0 , Java 2 Platform Micro Edition"

[12] Wolfgang Rankl and Wolfgang Effing, *Smart Card Handbook,* John Wiley & Sons, 2003

[13] Java Community Process, "Security And Trust Services API for Java 2 Platform Micro Edition Developer's Guide", December 2004