ÉCOLE NATIONALE SUPÉRIEURE
ÉLECTRONIQUE, INFORMATIQUE & RADIOCOMMUNICATIONS
BORDEAUX

Fourth Semester Project

Javacard Technology

TEACHER : M. Kadionik
GROUPE 1 : Maury P. ; Nguyen R. ; Sabri Y. ; Grolleau J.

Summary

I Presentation of the project

1 Context

Within the framework of our S4 project framed by Mr. Kadionik, we dealt with a new technology which simplifies the programmation of smart cards.
What Java programming technology gives to smart cards may make their current uses look downright simple-minded. JavaCard technology -- the implementation of Java technology for smart cards -- makes it possible to create and download new functions and services to cards, although they are already in the hands of consumers. The JavaCard API (Application Programming Interface) gives smart cards the ability to perform multiple functions, including thoses not even thought of when the cards are issued. Also significant, Java Card technology maintains the inherent security of Java technology. Therefore, Javacards are a safe way to conduct consumer transactions over insecure networks like the Internet and public telephone systems.

2 Abstract

The project was to study what Javacard technology could offer in terms of functions or services for customers and firms.
The development kit of cards applications and interface between terminal host and card (of the Telecommunication Department) is based on a subset of Java, JavaCard.
The project consisted in :

- Installing the development kit
- Getting acustomed with its use
- developing batch files converting the Java code into a format compatible with Javacard
- developing some applications

Progressing in this project, the reasons why Javacard is such a promising technology were understood.

II Introduction

Invented in 1974 by a French man, Roland Moreno, smart cards have now become a widespread support to store information. The most common examples of this type of card are credit cards, phone cards and cell phone SIM cards. It is no more than a plastic card with a single chip. There are various types of chips: microprocessors with internal storage, programmable chips and JavaCard which is easy to use. The connection with the terminal can be done by a direct physical contact or by an electromagnetic field.

Smart cards are of course becoming increasingly technologically advanced and today, some can execute a program on their microchips. The communication protocol between the card and the terminal has been studied. In order to carry out the project, a Gemplus development kit was used. This kit called gemxpresso includes a special card onto which executable codes can be loaded and tested.

III What is a Javacard

1 Advantages of Javacard

Javacard is one of the latest additions to the world of information technology.
Javacards provide :

-data portability : . Although similar to current credit cards, Javacards can have many different uses.

-security : - security between users (confidentiality): As an access-control device, javacards make personal and business data available only to the appropriate users.

- security between applications.

-flexibility

-compatibility with existing standards (Europay/MasterCard/Visa/SIM cards)

2 A single multifunction card

Not only can a card be programmed for many different types of applications, but more than one application can be loaded onto a card. In javacards, each application code and its data is protected and cannot be accessed by other applications. Thus, a javacard can be used for commercial transactions as well as for storing personal records (for instance: one's medical records), and much more. This is all the more valuable in an emergency situation, when one might not be physically able to communicate with a medical personnel : a single mutlifunction card is necessary.

In Java terminology these applications are said to run in their own "sandbox".

3 A modifiable/upgradeable card

In addition, javacards can be reprogrammed. This is an important advantage for the further development of smart cards as well as for industry. Hence, such an advance may allow smart cards to be omnipresent in people's future lives.

4 Not only a card

In Javacards, the microprocessor can run programs, process incomming and outgoing data (I/O), and store data as a computer does. But, because unlike computers, it has no user friendly Input/Output system, a Javacard works in tandem with a Card Acceptance Device (CAD), namely, a card reader or terminal. Both of these device types have slots into which a card is inserted. Card readers are connected to computers; terminals are themselves computers.

5 Data storage

As a computing device, smart cards currently have limited resources. A typical javacard has an 8-bit processor, 8 KB of ROM (Read-only Memory), 8 KB of EEPROM (Electrical Erasable Programmable Read Only Memory), and 128 bytes of RAM (Random Access Memory). A JavaCard application is called applet and is identified by a single identifier (AID = Application IDentifier) definied by the ISO7816 standard. It creates the objects in a EEPROM to keep information in a persistent way. Thus, data is saved and not lost when the card leaves the terminal (reader). Safety is directly integrated into the level of the operating

system. Each application has its own memory zone, which ensures the safety between the various applications and delimits the applications (delimitation necessary for maintenance because each applications can concern a different company). Thus, the different applications onto the card can't take data of an other application.

6 Javacard : a subset of Java

There are differences between Java itself and Java Card. Smart card applications, for instance, contain fairly basic functions, so, supporting multiple threads is not essential and would take up a lot of unnecessarily space (the subset of Java : Javacard does not need to contain the thread package). Current implementations of the JCVM (JavaCard Virtual Machine) (see appendix 3) use specialized bytecode instructions. The specific bytecodes are not yet specified so there is currently only source-code compatibility. This is likely to change in the near future. The JCVM bytecode set contains full support for smaller, primitive types, such as shorts. However, it does not support char, double, float, long, or arrays of more than one dimension. Thus, there are some programming constraints: using constants instead of variables (static,final), re-using the variables as far as possible, keeping a simple hierarchy of classes. (see appendix 2).

IV Communication between card and terminal

1 A communication protocol for every smart card

A smart card is inserted into a Card Acceptance Device (CAD), which may connect to another computer. It provides a basic function, namely to supply the card with power and to establish a data-carrying connection.

When two computers communicate with each other, they exchange data packages, which are constructed following a set of protocols. Similarly, smart cards communicate with the outside world using their own data packages -- called *APDU* (Application Protocol Data Units).

Thus, APDUs are packets of data that are exchanged between the CAD and a smart card. APDUs are the standard means of communication for smart cards. There are two types: command APDUs which specify an operation to be performed by a smart card; and response APDUs which contain the smart card's reply (status and, optionally, data) to an operational request. Java Card technology is modeled on a smart card specification standard, ISO7816. This standard specifies that communication between a host application and a smart card is done through APDUs. An APDU is a packet of data that follows a specific format :

- A command APDU starts with a header and is optionally followed by a body. The header contains fields that specify the operation to be performed by a smart card. The body includes any data that accompaniates the request; it also indicates the maximum number of data bytes expected in response to the command.
- A response APDU optionally begins with a body that contains any data returned in response. The response APDU ends with two mandatory bytes that specify the processing state of the card.

2 Use of APDU for Javacard

In Javacard technology, the host application sends a command APDU, and a Java Card applet responds with a response APDU. In fact, a Java Card applet remains idle until it receives a command APDU. However the communication is not directly 'host application' to 'Java Card applet'. Instead the JCRE acts as a stub. The command APDU is transmitted to the JCRE, which sends it to the appropriate Java Card applet for processing. After processing the APDU, the Java Card applet transmits a response APDU to the JCRE, which sends it to the host application.

3 Format of APDUs

The following tables illustrate command and response APDU formats respectively. APDU structure is described in ISO 7816, part 4.

Command APDU						
Mandatory Header				Conditional Body		
CLA	INS	P1	P2	Lc	Data field	Le

Figure 1 : Command APDU

The header codes the selected command. It consists of four fields: class (CLA), instruction (INS), and parameters 1 and 2 (P1 and P2). Each field contains 1 byte:

- CLA: Class byte. In many smart cards, this byte is used to identify an application.
- INS: Instruction byte. This byte indicates the instruction code.
- P1-P2: Parameter bytes. These provide further information for the APDU command.

Lc denotes the number of bytes in the data field of the command APDU; Le denotes the maximum number of bytes expected in the data field of the following response APDU.

Response APDU		
Conditional Body	Mandatory Trailer	
Data field	SW1	SW2

Figure 2 : APDU response

Status bytes SW1 and SW2 denote the processing status of the command APDU in a card.

V Stages of the project

1 Configuration of the development kit

The first step was to install the kit Gemxpresso RAD 211 on Windows 95 OS. The kit was made up of a terminal GC410,an installation CD-ROM and two Javacards.
In order to load an applet and communicate with a Javacard, what is needed is :

- The JDK (Java Development Kit) platform : JDK 1.2.2
- The files and libraries in order to convert the.java files
- The JcardManager which enables us to send the APDU command. (see appendix 3)
- The Comm API which provides the communication between the parallel and serie port for the JDK.

- The OpenCard Framework which provides the interface between the customer applications and the card (provided with the CD-ROM)

2 Loading of an applet onto Javacard

There is already an applet in the development kit. This second stage consists in loading this applet onto Javacard. This stage was the most important of the project. Indeed, first of all, the running of the kit has to be understood in order to adapt the batch files which convert the .java files into a compatible format (.jca, .exp, .jar, see figure : loding an applet).

There are four steps involved in the Java Card development process:

1. Applications can be compiled using any standard Java development environment, such as JDK.
2. Because Java Card defines a subset of Java, one must check that only features supported by Java Card are present. This step is called "verification."
3. After verification, the code is in JVM bytecode format, but needs to be converted to JCVM bytecode. This step is called "conversion."
4. After conversion, the code is ready to be loaded onto the card. Nowadays, there is no standardized file format for loading; however, one is in process of being.

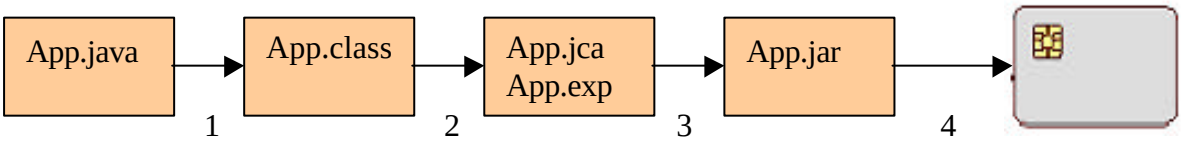


Figure 3 : Loading an applet

1 : compilation by JDK
2 : convertion of the file.class in two files (.jca and .exp)
3 : compression of the file.jca into a file.jar
4 : loading with JCardManager

Loading with Jcard Manager :

Jcard Manager is a Graphic User Interface (GUI). With JcardManager, the Javacard is authenticated with the file card.properties and by choosing the authentification command. Then the command “Upload JAR file into a card” has to be chosen. After words, the AID of the applet and that of the package have to be set. The AID of the security field, which is that of CardManager here, can be specified. Lastly, the Jar file has so that it can be selected to be loaded.

Loading the applet with batch files (RunGSE then UploadGSE) is also possible.

3 Development of two new applets

First of all, every application which has already been developed in smart cards will be adapted to Javacard. That is why we have chosen to develop two kinds of applications: a pointer (Pointeuse program) and a virtual fly ticket (Reservation program). The structure of the OPPurse.java code and some parts of OPPurse applet have been kept : those concerning the authentification, the security, the select and the reset of the applet.
In the pointer applet, the different APDUs made when the user put his card in the reader are :

- APDU 0:credit which saves and modifies the variables in function of at what time the card is inserted in the reader
- APDU 1 : get-hour which gives the engaging hour
- APDU 2 : get-maxnumberhourstodo which gives the number of hours to be done in the week
- APDU 3 : get lastnumberhoursdone which gives how many hours the user has worked in the last day
- APDU 4: get numberhoursdone which gives how many hours the user has worked since the beginning of the week
- APDU 5 : getnumberhourstodo which gives the number of hours that remain to be done in the week

The aim of this reservation applet is to store and then retrieve information about a fly reservation.
With this applet, we can have an access to a lot of data and modify some of it.

- APDU 0: sets the area of departure and arrival
- APDU 1: sets the time of departure and arrival
- APDU 2: gets the passenger flight class
- APDU 3: gets the cost of the flight
- APDU 4: delays the time of departure
- APDU 5: gets the area of departure and arrival
- APDU 6: gets the time of departure and arrival

(see the appendix 1 : an example of code which underlines the structure of APDU)

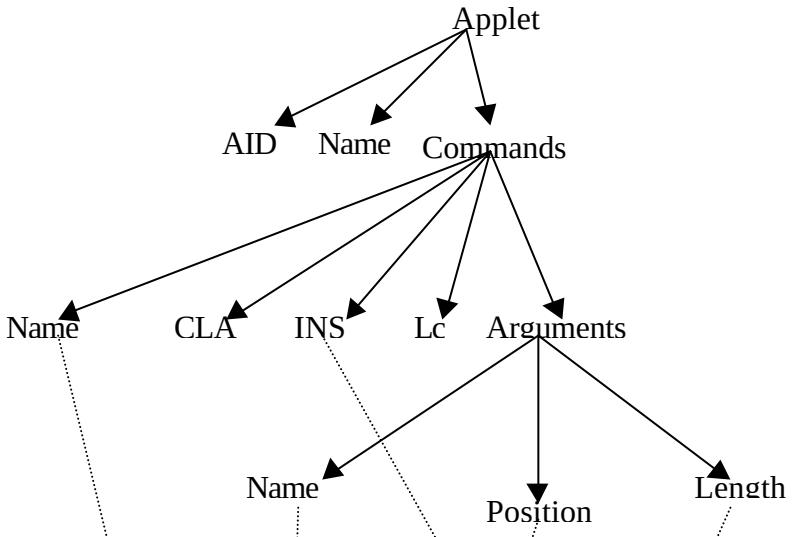
Through these applications, we have seen that developing applications dedicated to Javacards is all the more convenient than the language used is Java (universal language). Whereas the actual development of applications in smart cards requiers accurate knowledge in the proper language of card providers, Javacard technology makes it possible to develop applications on one's own. Moreover, as JCVM interprets the code before being loaded onto the card, the mistakes are detected and can be corrected.

VI Conclusion

Java Card technology is under full development. Today, various firms such as telecommunication operator which develop SIM card, firms which install advertisements in the street, or firms which install soft drink distributors firms plan to use JavaCard technology. Commercially speaking, there are a number of Java Card products on the market, including Schlumberger's Cyberflex (http://www.cyberflex.slb.com) and Gemplus's GemXpresso (http://www.gemplus.com/). Several other smart card vendors -- including Bull, Giesecke & Devrient, Motorola, and DeLaRue -- have announced plans to release Java Card. Furthermore, companies such as IBM, Informix, Oracle, and Visa International have endorsed the Java Card specification. Within 10 years, Java Card is likely to become a systematically used standard. Javacard development sets the tendency for the general use of Java in many fields

(cf *the monthly Programmez mai 2003 n°53* et J2ME : http://java.sun.com/j2me/docs).
However, even if Java has an advantage due to its low consumption of resources, a more sophisticated system still needs to be developed to allow a greater number of possible applications. Whithin the framework of Javacard, developers can design applications either in the terminal but also in the chip of the card for more portability. This is what is likely to propell Javacard technology in up and comming years.

APPENDIX 1 : an example of code which underlines the APDU structure



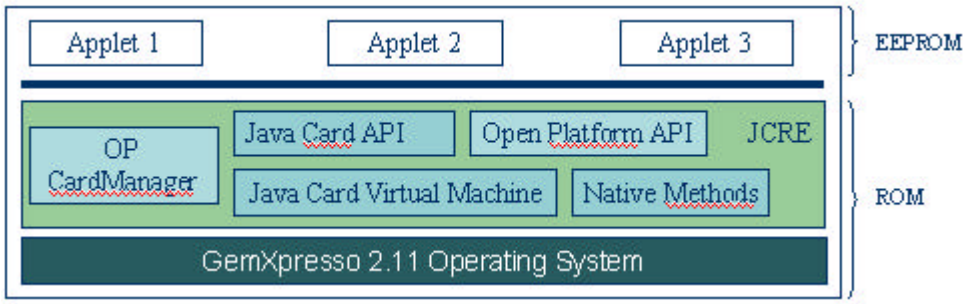

```
public class Pointeuse extends javacard.framework.Applet
{
        private final static byte INS_GET_HOUR = (byte)0x30;
        ....
        public void process(APDU apdu) throws ISOException
        {
                byte[] apduBuffer = apdu.getBuffer();
                switch(apduBuffer[ISO7816.OFFSET_INS])
                {
                        case INS_GET_HOUR :
                        gethour(apdu);
                        .....
                }
        }
        private void gethour(APDU apdu)
        {
        //get the APDU buffer
        byte[] apduBuffer=apdu.getBuffer();
        //write the hour into the apdubuffer after the APDU command part
        apduBuffer[5]=(byte)(hour>>8);
        apduBuffer[6]=(byte)hour;
        apdu.setOutgoing();
        apdu.setOutgoingLength((short)2);
        apdu.sendBytes((short)5,(short)2);
}
```

APPENDIX 2 : Javacard System Architecture

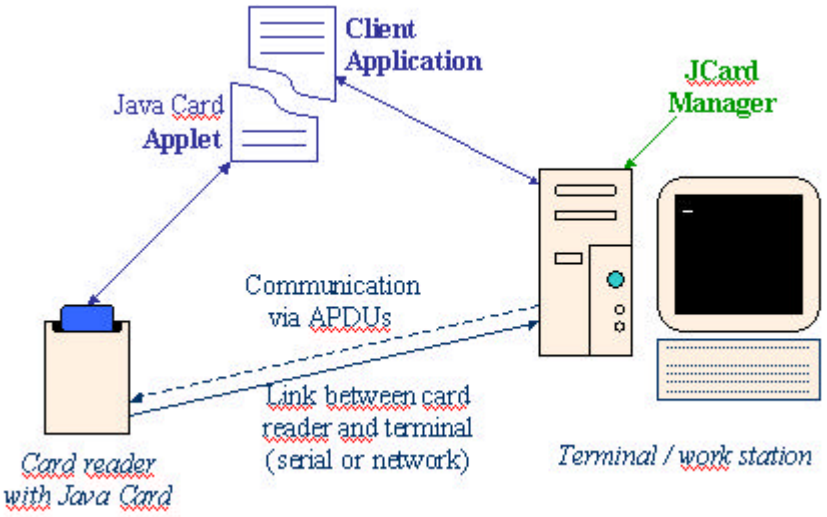
Software components for the smart cards in technology Java :

COS	OS of the Java card
Native service	Provide the service of cryptogrphie, allowance of the memory, inputs/outputs.
Java Card Virtuelle Machine (JCVM)	VM allows the execution of the Java byte-code, and the management of the exceptions
Framework	Together classes constituting the API one. Management of atomicity, selection of the applet, installation and the APDU.
JCRE	The « JavaCard Runtime Environment » : JavaCard Virtuel Machine (JCVM), Framework, natives functions, API.

Javacard System Architecture :



APPENDIX 3 : Communication between Card and CAD - Use of JCManager



Bibliography

Gemplus development Kit Documentation

Developer web sites :
Gemplus web site : http://www.gemplus.com
Schlumberger's Cyberflex web site : http://www.cyberflex.slb.com

Forum :
Java Card Special Interrest Group : http://www.javacard.org
Gemplus javacard team : javacardteam@gemplus.fr

Javacard standard :
http://java.com.sun/product/javacard

Java (J2ME) :
the monthly Programmez mai 2003 n°53
J2ME documentation : http://java.sun.com/j2me/docs