Smart Cards

A comprehensive tutorial

Michel Kœnig

University of Nice-Sophia Antipolis - ESSI

Presentation objectives

• Introducing the concepts and the technology of the smart cards
• Describing the protocols between cards and terminals
• Describing how to program the Java Cards
• Exploring the tools and the environments provided by the manufacturers to develop solutions with smart cards
Presentation content

- Introduction
- ISO7816 Protocol
- Security
- Java Card
- Client side programming
- Tools and environments
- Cyphering
- Conclusion
- Ap 1: Remote Method Invocation
- Ap 2: SIM Card

Introduction

History, technology, standards
Objectives

• In this chapter, we'll see
  – A brief history of the smart cards
  – What is a smart card
  – What are the applications available from the smart cards
  – What kind of prerequisites are needed to attend this tutorial

Brief history

• Early seventies, first patents
  – Dr Arimura, R Moreno, M Ugon
• Early eighties, first field testing for a memory card
  – Phone card in France
• Mid eighties, large scale introduction of smart cards in banking system
• Mid nineties, SIM card introduced in mobile telephony
What is a smart card

• A plastic card like a credit card with an embedded micro chip
  – With or without visible contacts
    • Maybe contactless

• Standardized
  – ISO 7816
    • Mechanical properties
    • Electrical behavior
    • Communication protocol

• Contains a software which
  – Protects internal data
  – Give access to these data in a secure way

For what applications ...

• Payment
• Loyalty systems
• Access systems
• Telephony
  – Mobile (GSM …)
• File system
  – Health
  – Education
  – …
Standards

- ISO 7816
  - Mechanical level
  - Electrical and communication protocol
- GSM 11.11 V6.1.0
  - SIM specs
- GSM 11.14 V7.1.0
  - SIM Toolkit specs
- GSM 03.19 V1.0.0
  - SIM API for Javacard
- Java Card
  - Java Card Forum
- EMV
  - Europay, Mastercard, Visa
- Open Platform

Content of this tutorial

- Exploring how a terminal can communicate with a smart card
- Discussing about security
- Understanding the organization of a Java Card
- Learning how to program a Java Card
- Discovering the tools available to program, test and deploy Java Cards
Conclusion

- In this chapter, we have seen
  - A brief history of the smart cards
  - What is a smart card
  - What are the applications available from the smart cards
  - What kind of prerequisites are needed to attend this tutorial

ISO7816 Protocol

*Physical description, communication layer, file system*
Objectives

- In this chapter, we'll see
  - An introduction to the ISO 7816 Protocol
  - Some mechanical and physical aspects of the cards standardized by ISO 7816
  - An extract of the protocol about the data communication

Mecanical and Electrical Aspects

- ISO 7816 standard describes
  - The physical organisation of the plastic card
  - Indicates the various zones
- It specifies also the purpose and the organisation of the contacts
  - For a smart contactful card
- Possible power voltage
  - 3V or 5V
  - Lower maybe in the future
Half-duplex serial protocol

- Due to the unique pin dedicated to input/output, the protocol is
  - Serial
  - Half-duplex

- Com characteristics:
  - Data: 8 bits
  - Parity: even
  - Stop: 1 bit

- Speed starting at 9600 Bps

Terminology

- The smart card reader powered by
  - a PC
  - A cash register
  - a mobile phone

is called a **terminal**

- In the standard ISO 7816 it is called:
  - The **Card Acceptance Device**
  - Or CAD
**Answer to Reset**

- When a card is inserted into the reader, a micro-switch signals this event to the terminal.
- The terminal powers up the card
  - Using a particular protocol
- When it is properly powered, the card sends back to the terminal a message called "Answer to Reset"

---

**General protocol**

- After sending *Answer to Reset*, the card waits until the terminal starts a communication
- The card never starts a communication
- The card answers to a demand coming from the terminal and waits for the next demand
**Application Protocol Data Unit**

- The APDUs are the commands sent by the terminal to the smart card.
- The APDU can:
  - carry parameters to the card
  - Expect results from the card
- Card and terminal must synchronize to:
  - the number of bytes to exchange
  - The direction of the exchange
  - This is done by the software embedded in each device.

---

**Application Protocol Data Unit**

- **CLA**: Class of the APDU: one byte which is characteristic of the APDU of the application.
- **INS**: Instruction: this is the command.
- **P1, P2**: two parameters which can be combined to form a short integer.
- **LC**: length of parameters which will be exchanged between the terminal and the card (from the terminal to the card, or from the card to the terminal).
No parameters exchanged

- **LC** = 0
- The card receives the APDU
- It processes it
- It returns a status word
  - Two bytes

Parameters sent by the terminal

- **LC** ≠ 0
- LC indicates the length of the data in bytes
- The software in the terminal and the software in the card must agree on the direction of the exchange
- The card acknowledges by sending back the **INS** byte
  - Simpliest case
Data expected by the terminal

- **LE ≠ 0**
  - The 5th byte is called LE in this case
- The card acknowledges the APDU by sending back the **INS** byte
  - Simplest case
Data are returned by the card, followed by the status word

Status word

- Status report of the internal operation done by the card
- **0x9000** means success!
- When different, could indicate
  - Denied access
  - File not found
  - No such CLA or INS expected
  - ...
Conclusion

• In this chapter, we have seen
  – An introduction to the ISO 7816 Protocol
  – Some mecanical and physical aspects of the cards standardized by ISO 7816
  – An extract of the protocol about the data communication

Java Card

Java Card Forum, history of the versions, programming aspects
Objectives

• In this chapter, we'll see
  – The various operating systems available for the smart cards
  – An introduction to the Java Card system
  – How the Applet are working
  – Some classes and methods provided by the Java Card API

Operating systems

• Beginning: proprietary systems
  – Only the applications were standardized
    • B0' for french banking system
• Now: multi-application systems
  – MULTOS
  – Windows for Smart Card
    • Dead
  – Java Card
Java Card History

- Early 1996
  - First development
    - Schlumberger, Bull CP8, GemPlus, Sun
  - Schlumberger’s Cyberflex
  - Java Card Forum
    - Most of the smart cards manufacturers
    - Sun
      - As a Java guru

Why Java in a smart card

- Java is an interpreted language
  - Need a Java Virtual Machine to run
- Applications could be portable from one smart card to another
- Applications run securely in a "sand box"
- Byte code is small
Is Java for Java Card pure Java?

• No!
• Roughly:
  – Basic types restricted to
    • Boolean
    • Small integers
      – Byte
      – Short
      – Int (optionnal)
  – Arrays restricted to one-dimensional arrays
  – Limited libraries
    • Including java.lang

Available libraries

• Basically, javacard and javacardx contain the smart card API
  – framework, security and crypto
• java.lang is reduced mainly to the exception definitions
• java.io and java.rmi was introduced in the last version
  – java.io to manage channels
  – java.rmi to manage remote method invocation
How Java works in a smart card

- A Java Virtual Machine is embedded
  - 4 K bytes
  - Basic library
- **Java Card Runtime Environment**
  - In charge of
    - Activation of applications
    - Low level communication protocol
    - Application downloading

Roles of the JCRE

- Downloading a package
- Creating an instance of an applet
- Selecting an applet
- Transmitting an APDU to a selected applet
- Managing the communication protocol with the CAD
Downloading a package

- Applets must be encapsulated in a package
- External processes
  - Compile the applets
  - Verify the bytecode
  - Create a jar-like container
    - CAP file
  - Will be seen later
- Package and applets are associated an identifier for future selection

What is a Java Card Applet

- A java object which is
  - Running using the JVM
  - Controlled by the JCRE
- The class of this object must extend the class `javacard.framework.Applet`
- The class must overload several methods

```java
package ePurse;
import javacard.framework.*;

class EPurse extends Applet {
  short balance;
  public EPurse() {
    ...
  }
  public static void install(...) {
    ...
  }
  public boolean select() {
    ...
  }
  public void process(APDU apdu) {
    ...
  }
}
```
Class APDU

- This class provides the basic features needed to handle the ISO7816 protocol from the applet point of view
- It gives access to the internal buffer dedicated to the communication
- This buffer can be
  - Retrieved by the applet
  - Filled up by the applet and sent to the CAD

Main methods of the APDU

```java
byte buffer[] = apdu.getBuffer();
apdu.setIncomingAndReceive();
short le = apdu.setOutgoing();
apdu.setOutgoingLength(le);
apdu.sendBytes(ISO7816.OFFSET_CDATA, le);
apdu.setOutgoingAndSend();
```

- These methods help to
  - Get the internal buffer
  - Start receiving data
    - Acknowledgement
    - Start transmitting data
- Utilities help to
  - Transform 2 bytes in a short and vice versa
  - Copy buffers
  - Compare buffers
Class ISO7816

This class encapsulates most of the ISO7816 constants needed to program the applets.

- Constants are prefixed by
  - CLA for class related constants
  - INS for instruction related constants
  - OFFSET for offsets in the buffer
  - SW for status word related constants

Lifecycle of an applet

- The JCRE downloads the package containing the Applet
- It calls the static method `install` on the Applet
- This method creates an instance
  - Or more
- And `register` this instance using an `AID`
Lifecycle of an Applet

- When the instance is created and registered it can be called
- The JCRE can
  - select
  - deselect
  the instance
- Can call the instance to process an APDU

Example of an Applet
Converting the `.class` file

- The file `EPurse.class` must be converted before downloading.
- The downloading format is called a **CAP File**.
- The tool **converter**
  - Makes the conversion
  - Assigns an AID to the package
  - Assigns an AID to the applet
The converter

- The converter can be called using a configuration file: **EPurse.opt**

```
> converter -config epurse.opt

Java Card 2.2 Class File Converter (version 1.1)
Copyright (c) 2000 Sun Microsystems, Inc. All rights reserved.

conversion completed with 0 errors and 0 warnings.
```

Michel Koenig

Smart cards tutorial
Java Card simulation

The Toolkit provides tools for simulation:

- **jcwe**: Java Card Workstation Development Environment
  - Which simulates the Java Card
- **apdu_tool**: Which simulates the Java Card reader
The **jcwde tool**

- The simulator
  - Uses an applet called "Installer" to download and install another applet
  - Listens on a socket for incoming APDU
    - Port 9025 par défaut
  - Manages the protocol
    - Throws an exception in case of trouble

---

Running the **jcwde tool**

- This is done using a configuration file which indicates
  - The applet Installer AID
  - The AID of the applet to be downloaded

```
> jcwde -p 9025 jcwde.app
```

Java Card 2.2 Workstation Development Environment (version 1.1).
Copyright (c) 2000 Sun Microsystems, Inc. All rights reserved.
jcwde is listening for T=0 Apdu's on TCP/IP port 94025.
File `jcwde.app` content

```java
// applet
com.sun.javacard.installer.InstallerApplet
  0xa0:0x00:0x00:0x62:0x03:0x01:0x08:0x01

EPurse.EPurse
  0xa0:0x00:0x00:0x62:0x03:0x01:0x0c:0x03:0x01
```

**apdutool**

- This tool reads a script which contains APDU and sends the APDU to the card
  - In fact to the *jcwde simulator*
- It displays the result in hexadecimal
  - On the standard output
  - Or in a specified file when using the option `-o`
**Script example**

```
powerup;
// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
// begin installer command
0x80 0xB0 0x00 0x00 0x00 0x7F;
// create EPurse
0x80 0xB8 0x00 0x00 0x0c 0xa0 0xa0 0x00 0x00 0x62 0x03 0x01 0x0c 0x03 0x1
0x00 0x7F;
// end installer command
0x80 0xBA 0x00 0x00 0x00 0x7F;
// Select EPurse
0x00 0xa4 0x04 0x00 0xa0 0xa0 0x00 0x00 0x62 0x03 0x01 0x0c 0x03 0x1
powerdown;
```
Other Java Card features

• Many features available
  – PIN code management
  – Transaction handling using **JCS**ystem
  • Possibility to group together a certain number of actions into a transaction
  • Possibility to **abort** or **commit** the transaction
  – Shareable applets
  – Possibility to have several applets selected at the same time

OwnerPIN

• This class helps the developer to protect the access to some features of the smart card using a PIN code

```java
private OwnerPIN pinCode;

/** Creates a new instance of EPurse */
public EPurse() {
    balance = (short) 0;
    pinCode = new OwnerPIN(EPURSE_PIN_TRY_LIMIT, EPURSE_PIN_MAX_SIZE);
}
```
OwnerPIN

- The CAD must validate the PIN code prior to access the other features

```
case EPURSE_ADD:
apdu.setIncomingAndReceive();
if(!pinCode.isValidated())
    ISOException.throwIt(
        ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

case EPURSE_PIN:
apdu.setIncomingAndReceive();
if(!pinCode.check(buffer,
            ISO7816.OFFSET_CDATA, EPURSE_PIN_MAX_SIZE))
    ISOException.throwIt(
        ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
break;
```

OwnerPIN

- The OwnerPIN proposes a method to unblock a blocked PIN code (after a TRY_LIMIT unsuccessful attempts)

```
case EPURSE_UNBLOCK:
    pinCode.resetAndUnblock();
```
OwnerPIN

- The OwnerPIN proposes a method to reset the validated flag

```java
public boolean select(){
    pinCode.reset();
}
```

Cryptography

- An entire package is dedicated to cryptography
  - Including creation and management of keys
    - Public and private
  - Using several algorithms
    - AES, DES, DES3, RSA
- Two main packages
  - javacard.framework.security
  - javacardx.crypto
Cryptography

Class Summary

<table>
<thead>
<tr>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>PublicKey</td>
</tr>
<tr>
<td>PrivateKey</td>
</tr>
<tr>
<td>ShamirSecretKey</td>
</tr>
<tr>
<td>RSAKeyPair</td>
</tr>
<tr>
<td>RSAHelper</td>
</tr>
<tr>
<td>PKCS10Certificate</td>
</tr>
<tr>
<td>PKCS10CertificatePolicyInfo</td>
</tr>
</tbody>
</table>

In this chapter, we have seen:

- The various operating systems available for the smart cards
- An introduction to the Java Card system
- How the Applet are working
- Some classes and methods provided by the Java Card API
Security

Hardware and software aspects

Objectives

• In this chapter, we’ll see
  – An introduction about the security aspects of the smart cards
    • From a hardware point of view
    • From a software point of view
Hardware security

- A smart card contains important data
  - It could contain money
    - Electronic purses
- It must be tamper resistant

"If you know the attack you can build the shield"

The attacks

- X-raying the micro-chip
- Measuring the power consumption variation during critical APDU
  - When the PIN code is transmitted for example
- Measuring the answer delay
  - To try to predict what branches in the program are completed
The shields

- The micro-chip uses an internal shield to protect itself against an X-Ray scanning
- It guarantees the same delay for both branches of an alternative statement
- It guarantees the same power consumption in all cases

Software attacks and shields

- Data are protected using cryptography
  - Various techniques
    - DES, DES3
    - RSA
    - SHA
  - Cryptography is based on
    - A public algorithm
    - A key
      - Private (DES, DES3)
      - Public (RSA)
Symmetric Enciphering

Bob

SAME KEY

Alice

Asymmetric enciphering

Bob’s private key

Bob

Bob’s Public Key

Alice
Signing using asymmetric keys

Bob’s private key

Bob

Bob’s Public Key

Alice

Certify public key

X509 Certificate

• Subject (name, company, e-mail …)
• Start Date
• End Date
• Issuer’s subject
• Public Key
**Certification Authority**

- Subject (name, company, e-mail ...)
- Start Date
- End Date
- Issuer's subject
- Public Key

**Certificate**

- Subject (name, company, e-mail ...)
- Start Date
- End Date
- Issuer's subject
- Public Key

**Authentication**

**Authorization**

**Privacy**

- Integrity
- Non-repudiation
Protect private key With Smart cards

- The Private key born, live and die inside the card
  - Key pair generation
  - Secure access
  - Cryptographic algorithm process inside the card

- Physically secure
  - No Hard drive storage of the private key

- Portable
  - No multi-key
  - Multiple Device

- Enciphering is done inside the card
  - Computer Independent

Hashing (a.k.a Fingerprint)

- Modifying one bit completely changes the Hash
- Hash result is completely unpredictable
- Usual algorithms are MD5 (used for linux Password storage) or SHA-1
Digital Signature (E mail)

Sender

Sender’s private key & X.509 certificate
Kps Sender’s PK
Ksa Sender’s SK

Receiver

Letter

8365923334 Hash

Digital Signature (8A!G@3&04)

Certificate Authority

CA Public key (certificate checked)

Identification/ Authentication of the content of the letter

Hash

= ?

S/MIME Encryption

Sender

Message

$@!&@#

- generate “symmetric document key” (PC)
- encrypt message with symmetric key (PC)

Encrypted message

- get certificate of receiver, verify certificate and extract public key

- encrypt “sym document key” with receiver’s public key

Trust Centre

Kpr

- unwrap document key with the receiver’s private key
- decrypt message with “sym document key”

Receiver

Message
Conclusion

• In this chapter, we have seen
  – An introduction about the security aspects of the smart cards
    • From a hardware point of view
    • From a software point of view

Client side programming

ISO7816-3 protocol
Objectives

- In this chapter, we'll see
  - The ISO7816 protocol between the card reader and the smart card
  - The protocols between the smart card reader and the PC
    - PC/SC
    - TLP224
    - GBP
    - Open Card

Reader-smart card protocol

- According to the manufacturer, smart card can support
  - Two different power voltages
    - 3V and 5V
  - Two different ways to transmit characters
    - Direct or inverted
  - Two different protocols
    - T0, T1
Detecting the voltage

- A complex algorithm is needed to detect the voltage of the smart card
  - Try 3V
    - If ATR detected
      - Is it good?
      - ...
**T0 protocol**

- **T0 is a character oriented protocol**
  - One character is transmitted after the other
  - Acknowledgement, if needed, is done after the transmission of the 5 bytes of the APDU
- **T0 limits the length of the data transmitted**
  - 32 bytes
  - Possibility to chain APDUs

**T1 protocol**

- **T1 is a block oriented protocol**
  - The entire APDU, including the extra data, is transmitted all at once
  - Possibility to have sequences of messages for long data
Classes of APDU and Commands

- It is virtually possible to use any value for CLA and INS in an APDU.
- Nevertheless, some values are reserved by ISO7816:
  - The constant CLA_ISO7816 is the value of CLA reserved by ISO.
- The next page displays some values reserved for INS.

Standard ISO commands

<table>
<thead>
<tr>
<th>Value</th>
<th>Command</th>
<th>Value</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0E</td>
<td>Erase binary</td>
<td>0xC0</td>
<td>Get response</td>
</tr>
<tr>
<td>0x20</td>
<td>Verify</td>
<td>0xC2</td>
<td>Envelope</td>
</tr>
<tr>
<td>0x70</td>
<td>Manage channel</td>
<td>0xCA</td>
<td>Get data</td>
</tr>
<tr>
<td>0x82</td>
<td>External authenticate</td>
<td>0xD0</td>
<td>Write binary</td>
</tr>
<tr>
<td>0x84</td>
<td>Get challenge</td>
<td>0xD2</td>
<td>Write record</td>
</tr>
<tr>
<td>0x88</td>
<td>Internal authenticate</td>
<td>0xD6</td>
<td>Update binary</td>
</tr>
<tr>
<td>0xA4</td>
<td>Select file</td>
<td>0xDA</td>
<td>Put data</td>
</tr>
<tr>
<td>0xB0</td>
<td>Read binary</td>
<td>0xDC</td>
<td>Update record</td>
</tr>
<tr>
<td>0xB2</td>
<td>Read record</td>
<td>0xE2</td>
<td>Append record</td>
</tr>
</tbody>
</table>
**PC – Reader protocols**

- Most of the smart card readers are connected to a PC through a serial link
  - A new generation use a USB link
- The most common protocols used are
  - TLP224
    - Characters oriented
  - GBP
    - Blocks oriented
- Microsoft had introduced recently a new protocol: PC/SC
- Open Card had introduced a protocol based on Java: OCF

---

**TLP224**

- Introduced by Bull CP8
- Encode commands to the smart card reader
  - Power on
  - Power off
  - Send APDU
  - Resend message
  - Error

- The message is encapsulated between
  - ACK (0x60)
  - LN length of message
  - LRC
- Bytes are split in quartets and encoded in ASCII
GBP

- Introduced by GemPlus
- Similar to the protocol T=1
  - Simplified
- It is a transport layer which allows the PC to send commands to the reader

PC/SC

- **PC to Smart Cards**
- Introduced by Microsoft
  - Helped by smart cards manufacturers
OpenCard

- Offer a portable platform to develop client-side application
- This application could work
  - With Java Cards
  - Or with other cards
- The technique used is based on the use of a proxy
  - Based on the Remote Method Invocation technique
  - We'll see more about OCF in the next chapter

Conclusion

- In this chapter, we have seen
  - The ISO7816 protocol between the card reader and the smart card
  - The protocols between the smart card reader and the PC
    - PC/SC
    - TLP224
    - GBP
    - Open Card
Remote Method Invocation

*Principles and programming techniques*

**Objectives**

- In this chapter, we'll see
  - What is the Remote Method Invocation technique
  - How this technique was introduced for smart card programming
  - How to develop services using the Open Card Framework
Principles of RMI - intro

• In a Java program located in a single machine
  – Classes and objects lay in the same memory storage
  – All of them are powered by the same Java Virtual Machine

• During the call of a method the control is passed from one object to the other
  – The JVM does the job

Principles of RMI - intro

• In the case of a distributed program
  – Classes and objects do not lay in the same memory storage
  – They are powered by two different Java Virtual Machines

• The direct call of one method of Object2 by Object1 is no longer possible
Michel Koenig

Smart cards tutorial

Java Virtual Machine

RMI Agent

Class Proxy

Object1

Class1

Interface2

RMI Compiler

Class2

Interface2

Object2

Registration

Access demand to Object2

Michel Koenig

Smart cards tutorial

Java Virtual Machine

RMI Agent

Class Proxy

Object1

Class1

Interface2

RMI Compiler

Class2

Interface2

Object2

Registration

Access demand to Object2

Michel Koenig

Smart cards tutorial
RMI for the Java Card

- The application classes reuse the same production environment as in Java Standard case
  - Use base class CardRemoteObject for implementation
  - Use base interface Remote for remote interface

Extra software needed

- Java Card RMI needs extra software to work
  - One part on the card itself
  - One part on the terminal
    - Open Card Framework was chosen by Sun as a reference implementation
Example: Applet-side interface Purse

```java
package com.sun.javacard.samples.RMIDemo;

import java.rmi.*;
import javacard.framework.*;

public interface Purse extends Remote{
    public short getBalance() throws RemoteException;
    public void debit(short m) throws RemoteException, UserException;
    public void credit(short m) throws RemoteException, UserException;
}
```

Michel Koenig

Example: Applet-side (1) implementation EPurseImpl

```java
package com.sun.javacard.samples.RMIDemo;

import javacard.framework.UserException;
import javacard.framework.Util;
import javacard.framework.service.CardRemoteObject;
import java.rmi.RemoteException;

public class PurseImpl extends CardRemoteObject implements Purse {
    private short balance = 0;

    public PurseImpl() {
        super(); // export it
    }
}
```

Michel Koenig
Example: Applet-side (2)
implementation EPurseImpl

```java
public void credit(short m) throws RemoteException, UserException {
    if (m <= 0) UserException.throwIt(BAD_ARGUMENT);
    balance += m;
}

public void debit(short m) throws RemoteException, UserException {
    if (m <= 0) UserException.throwIt(BAD_ARGUMENT);
    balance -= m;
}

public short getBalance() throws RemoteException {
    return balance;
}
```

Example: Applet-side (1)
installing applet

```java
import java.rmi.*;
import javacard.framework.APDU;
import javacard.framework.ISOException;
import javacard.framework.UserException;
import javacard.framework.Util;
import javacard.framework.service.*;

public class PurseApplet extends javacard.framework.Applet {
    private Dispatcher disp;
    private RemoteService serv;
    private Remote purse;
}
```

A dispatcher glues together all the services and dispatches APDU to services

A service knows how to process all incoming APDU
Example: Applet-side (2) installing applet

```java
public PurseApplet() {
    purse = new PurseImpl();
    disp = new Dispatcher((short) 1);
    serv = new RMIService(purse);
    disp.addService(serv, Dispatcher.PROCESS_COMMAND);
    register();
}
public static void install(byte[] aid, short s, byte b) {
    new PurseApplet();
}
public void process(APDU apdu) throws ISOException {
    disp.process(apdu);
}
```

Only one service will be created and attached to the dispatcher.
Add the service which was just created as a command processor.
Delegate the process of the apdu to the dispatcher (then to the service).

Example: client-side (1) code: PurseClient

```java
import opencard.core.service.*;
import examples.purse.*;
import com.sun.javacard.javax.smartcard.rmiclient.*;
import com.sun.javacard.ocfrmiclientimpl.*;
import javacard.framework.UserException;

public class PurseClient extends java.lang.Object {
    /** Creates new PurseClient */
    public PurseClient() {
    }

    public static void main(java.lang.String[] argv) {
        // arg[0] contains the debit amount
        short debitAmount = (short) Integer.parseInt(argv[0]);
```
Example: client-side (2)
code: PurseClient

```java
try {
    // initialize OCF
    SmartCard.start();

    // wait for a smartcard
    CardRequest cr = new CardRequest(CardRequest.NEW_CARD,
        null, OCFCardAccessor.class);
    SmartCard myCard = SmartCard.waitForCard(cr);

    // obtain a Java Card RMI Card Accessor CardService
    CardAccessor myCS = (CardAccessor)
        myCard.getCardService(OCFCardAccessor.class, true);

    // create a Java Card RMI connector instance
    JavaCardRMIConnect jcRMI = new JavaCardRMIConnect(myCS);
}
```

Start the Open Card Framework
This class could be considered as the driver of the card

Example: client-side (3)
code: PurseClient

```java
// select the Java Card applet
byte[] appAID = new byte[] {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
jcRMI.selectApplet(appAID);

// obtain the initial reference to the Purse interface
Purse myPurse = (Purse) jcRMI.getInitialReference();

// debit the requested amount
try {
    short balance = myPurse.debit(debitAmount);
} catch (UserException jce) {
    short reasonCode = jce.getReason();
    // process UserException reason information
}
```

Get the remote reference of the Purse (a reference to the proxy)
Example: client-side (4)
code: PurseClient

```java
// display the balance to user
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        SmartCard.shutdown();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Conclusion

- In this chapter, we have seen
  - What is the Remote Method Invocation technique
  - How this technique was introduced for smart card programming
  - How to develop services using Java Card RMI and the Open Card Framework
Conclusion

• In this presentation, we have
  – Introduced the concepts and the technology of the smart cards
  – Described how to program the Java Cards
  – Explored the tools and the environments provided by the manufacturers to develop solutions with smart cards
Appendix 1

SIM cards

SIM Cards

Proactive SIM cards
Objectives

- In this chapter, we'll see
  - The standards around the SIM card
  - What is a proactive SIM card
  - How works a proactive SIM card

SIM cards

- Standardized by ETSI for GSM
  - GSM 11.11 V6.1.0
    - SIM specs
      - Subscriber Identification Module
  - GSM 11.14 V7.1.0
    - SIM Toolkit specs
  - GSM 03.19 V1.0.0
    - Javacard SIM API
Proactives SIM

• Using the SIM Toolkit, possibility to
  – Program the SIM
  – Make the SIM card application driving the phone
    • Access to keyboard, display, ...

Internal organization of the SIM

• The SIM contains a certain number of "files" grouped into "directories"
• Terminology:
  – Element File: file
  – Dedicated file: directory
**Proactive SIM**

- The ISO7816 standard does not permit that the card starts talking first
  - A card is waiting for an APDU and responds when it receives the APDU
- Proactive SIM cards use a specific status word to indicate to the **Mobile Equipment** that they want to talk to it

**Proactive protocol**

- The Mobile Equipment (the phone) send a command (**Envelope**)
  - Containing the menu selection
- The card answers using the status word **91xx**
  - Xx is the length of the command that the SIM wants to send back to the ME
- The ME sends the command **Fetch** to get the command from the SIM
- ...

---

Michel Koenig  Smart cards tutorial  127

Michel Koenig  Smart cards tutorial  128
Allowed commands for the SIM

- The SIM card can
  - Display text on the phone display
  - Input data from the keyboard
  - Play tone
  - Send a SMS
  - Process an incoming SMS
  - …

Conclusion

- In this chapter, we have seen
  - The standards around the SIM card
  - What is a proactive SIM card
  - How works a proactive SIM card
Appendix 2

Logical channels

Objectives of this appendix

• In this appendix, we'll see
  – What is a channel, and what it is for
  – The standard ISO7816-4 about channels
  – How Java Card 2.2 takes in account this standard
  – An example how to use the channels
What is a channel

• Within the smart card, several applets could be installed at the same time
  – An e-purse
  – A loyalty program
  – A credit-debit application
  – An e-ticket application

• For what we have seen, only one can be selected at once

What is a channel (continued)

• Example: buying a transportation ticket using the epurse and getting loyalty points

• Security must be the same during all the operations

• We cannot expect the same OwnerPIN for the three applets
What is a channel (continued)

- It is a virtual link between the CAD and a selected element within the card
- The APDU are redirected to the corresponding element according to the channel number which is specified in the APDU

Channels in ISO7816-4

- ISO7816-4 allows up to 4 channels to be used
  - Channel 0: default (or basic) channel
  - Channels 1, 2, 3
- Two commands are dedicated for channel handling
  - SELECT FILE
  - MANAGE CHANNEL
Selecting a channel

• Channel information can be held only with APDU starting with a CLA byte of the following type
  – 0x0X, 0x8X, 0x9X and 0xA0
  – The X nibble is responsible for
    • Channel encoding
      – Least significant bits
    • Secure message encoding

MANAGE CHANNEL APDU

• This command will not be processed by any applet from the card
• It is processed by the underlying operating system
• It is used to
  – Open a channel
  – Close a channel
Opening a channel

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>Lc</th>
<th>data</th>
<th>Le</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x01</td>
<td></td>
</tr>
</tbody>
</table>

- The new open channel will be **R**
- If \( Q=0 \), the default applet will be selected on channel **R**
- If \( Q\neq 0 \), the selected applet on channel **Q** will become the current applet selected on channel **R**

---

Opening a selected channel

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>Lc</th>
<th>data</th>
<th>Le</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td></td>
</tr>
</tbody>
</table>

- If \( Q=0 \), the default applet will be selected on channel **R**
- If \( Q\neq 0 \), the selected applet on channel **Q** will become the current applet selected on channel **R**
Closing a channel

- Channel R will be closed
  - Channel R must not be the basic channel

Select an applet

- Channel R can be any (opened or unopened) channel, including the basic channel
  - The applet identified by AID will become the selected applet in channel R
  - If channel R is not open, the command open it
  - If the channel is open, the command will change the selected applet in the channel to the one specified
Multiselectable applets

- The same applet can handle APDU on several channels
- The applet must implement the `javacard.framework.Multiselectable` interface
  - Must implement methods `select` and `deselect`
- Classpath must include `apduio.jar`

Memory usage

- If applets A and B from the same package are `multiselected`
  - They share the same `CLEAR_ON_DESELECT` memory segment
  - This segment will be clear **ONLY** when both applet will be deselected
Memory usage (continued)

- The applet can work with an internal two-fold data structure
  - One for the first channel
  - One for the second channel
- This avoid duplicating code in memory
  - Only data is duplicated

Multiselectable interface

- This interface supports two methods:
  - public boolean select(boolean allreadySelected);
    - Indicates if the applet (or one of the same package) has already been selected on different channels
    - Important to know for initialization process
  - public void deselect(boolean stillSelected);
    - Indicates if the applet (or one of the same package) remains selected on different channels
    - Important to know for CLEAR_ON_DESELECT memory usage
**Multiselectable applet facts**

- If one applet within a package is **Multiselectable**
  - All the applets in the same package must be selectable
- Each time an applet is proposed to be selected
  - It must take in account if other applets within the same package are selected or not
- Same thing for deselection
- During the process of an APDU
  - Only one applet is active

**Other features**

- Some methods are provided to help developer to handle correctly the channels
  
```java
static byte APDU.getCLACHannel();
```
- According to the selected channel, the applet can process differently the data
Example

• Suppose a wireless device able to handle several communications at the same time:
  – Phone call
  – Far Internet connection
  – Local internet connection
• This device needs to activate counters for each of the connection started and to stop counters when the connection is held.

Example

• Possible usage:
  • Channel 0 is dedicated to the global clock
    – To increment started counters
  • Channel 1 is for the phone call connection
  • Channel 2 is for the far Internet connection
  • Channel 3 is for the local Internet connection
Example

• We can use a unique applet which
  – Create an array of short counters (3) on selection on channel 0
  – APDU to start, to stop or to get counters differ only on the channel number
  – APDU to increment counters is on channel 0

Conclusion

• In these appendices, we have seen
  – The SIM cards
    • Internal organization
    • The proactive commands
  – The channels
    • What is a channel, and what it is for
    • The standard ISO7816-4 about channels
    • How Java Card 2.2 takes in account this standard
    • An example how to use the channels