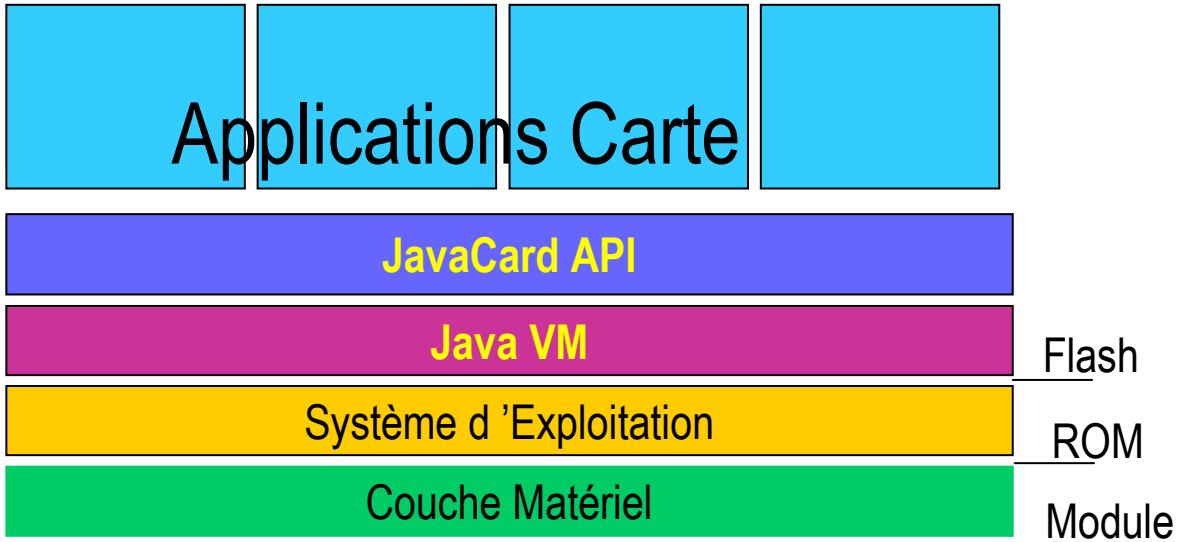
Programmation JavaCard

Didier Donsez (Univ. Joseph Fourier, Grenoble 1)
prenom.nom@imag.fr

en collaboration avec
Gilles Grimaud (Univ. Lille 1)
Sylvain Lecomte (Univ. Valenciennes)
Sébastien Jean (IUT de Valence)

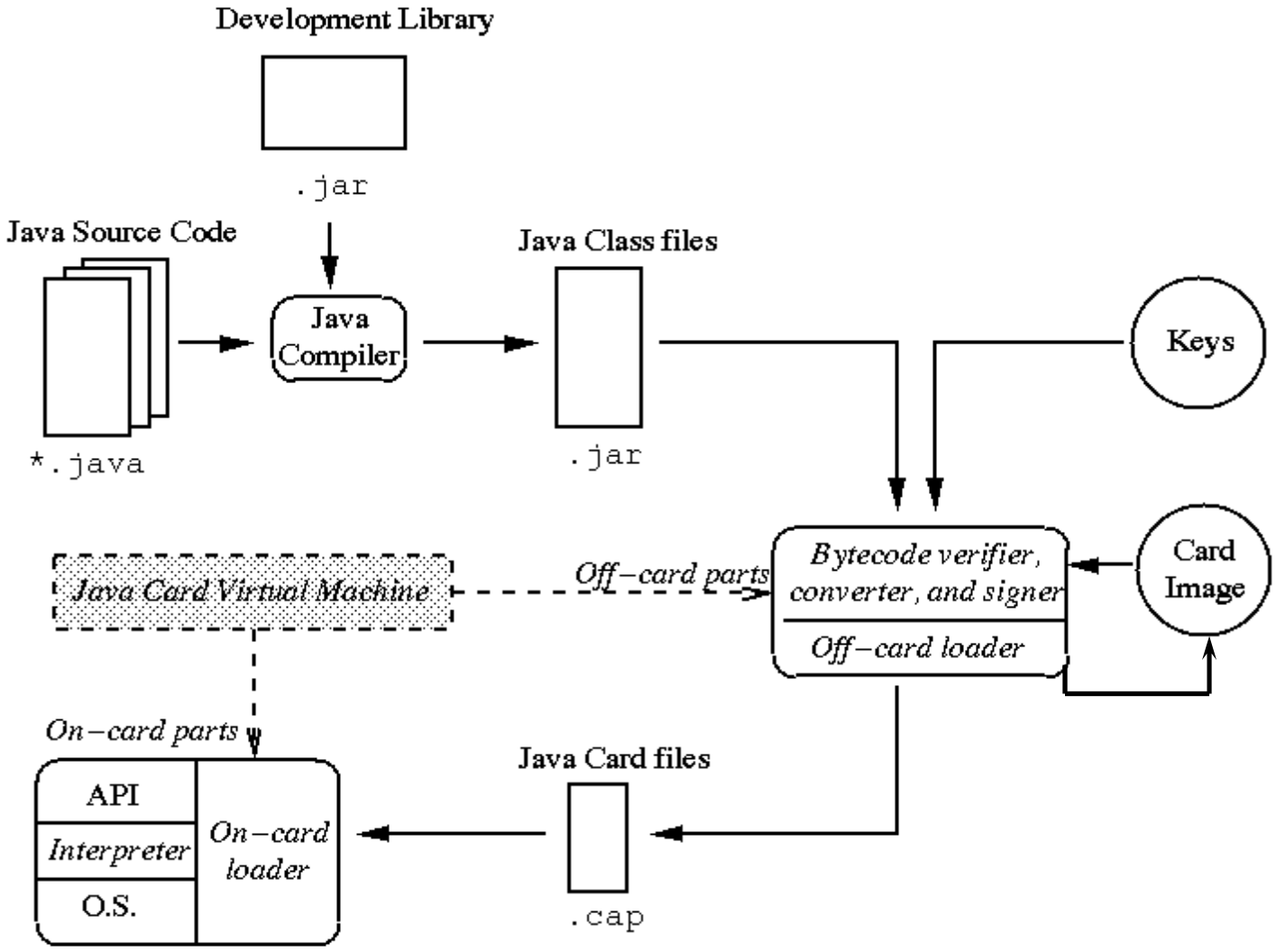
La JavaCard

- Carte basée sur un interpréteur de bytecode Java



- Chaîne de production

Machine virtuelle JC : architecture



JavaCard : Historique

- Carte ’96 (CNIT-Paris)
  - Schlumberger présente la CyberFlex 1.0
  - Parallèlement d ’autres projets commun sur base de :
    - Langage C (Multos)
    - Langage Forth (Projet Gemplus)
  - Accord entre les principaux participants et création du javaCard Forum
- 1998 : le vrai départ
  - CyberFlex 2.0, GemXpresso, ...

Le JavaCard Forum

- Consortium de fabriquant :
  - Carte : Delarue, Gemplus, Oberthur, Schlumberger
  - informatique : IBM, SUN
  - Matériel : DEC, Motorola
  - Utilisateurs : banques
- But :
  - promouvoir la solution de la Javacard
  - Faire des choix communs (définition de standards)
- Solutions :
  - Un comité technique
  - Un comité « business »
  - plus d ’information : http://www.javacardforum.org

Java Card par rapport à Java (1/4)

- Pas de chargement dynamique de classes

- Objets : Allocation dynamique d'objets supportée (**new**)

Mais

Pas de ramasse-miettes (gc)

Pas de désallocation explicite non plus

==> mémoire allouée ne peut pas être récupérée

Pas de méthode **`finalize()`**

Java Card par rapport à Java (2/4)

- Types de base
  - Entiers signés, complément à 2
    - **`byte` (8 bits), , `short` (16 bits), `int` (32 bits)** *optionnel*
    - **`boolean`**
    - Pas de types **`char`** (pas de classe **`String`**), **`double`**, **`float`** et **`long`**
  - Pas de classes **`Boolean`**, **`Byte`**, **`Class`**, *etc.*
- Objets
  - **`java.lang.Object`**
  - **`java.lang.Throwable`**
- Tableaux à une dimension :
  - Eléments : types de base
  - Maximun : 2^15 éléments

Java Card par rapport à Java (3/4)

- Mécanisme d'héritage identique à Java
  - Surcharge de méthodes, méthodes abstraites et interfaces
  - Invocation de méthodes virtuelles
  - Mots-clés **`instanceof`**, **`super`** et **`this`**
- Pas de threads
  - Pas de classe **`Thread`**, pas de mots-clés **`synchronized`**
- Sécurité
  - Notion de paquetage et modifieurs **`public`**, **`protected`** et **`private`** identiques à Java
  - Pas de classe **`SecurityManager`** : politique de sécurité implémentée dans la machine virtuelle

Java Card par rapport à Java (4/4)

- Mécanismes d'exception supportés
  - Peuvent être définies (**`extends Throwable`**), propagées (**`throw`**) et interceptées (**`catch`**)
  - Classes **`Throwable`**, **`Exception`** et **`Error`** supportées et certaines de leurs sous-classes (dans **`java.lang`**)
- Méthodes natives (**`native`**)
- Atomicité
  - Mise à jour de champs d'objets doit être atomique
  - Modèle transactionnel : **`beginTransaction()`**, **`commitTransaction()`** et **`abortTransaction()`**

Résumé Java Card p/r à Java (1/2)

- Supportés:
  - `boolean`, `byte`, `short`, `int`
  - `Object`
  - Tableau à une dimension
  - Méthodes virtuelles
  - Allocation dynamique
  - Paquetages
  - Exceptions
  - Interface
  - Méthodes natives
- Non supportés :
  - `float`, `double`, `long`
  - `char`, `String`
  - Tableau à n dimensions
  - `Class` et `ClassLoader`
  - Ramasse-miettes
  - `SecurityManager`
  - Threads

Résumé Java Card p/r à Java (2/2)

- Mots-clés non disponibles :
  - **`char`**, **`double`**, **`float`**, **`long`**, **`synchronized`**
- API **`java.lang`** de Java Card réduite à :
  - **`Object { public Object();`**
    `public boolean equals(Object obj); }`
  - **`Throwable { public Throwable(); }`**
    `-- Exception`
    `-- RuntimeException`
    `-- ArithmeticException`
    `-- ClassCastException`
    `-- NullPointerException`
    `-- SecurityException`
    `-- ArrayStoreException`
    `-- NegativeArraySizeException`
    `-- IndexOutOfBoundsException`
    `-- ArrayIndexOutOfBoundsException`

Construction d'applications Java Card

- Une application carte
  - Code dans la carte (application serveur = applet Java Card)
  - Code dans le terminal (application cliente)

- Construction d'une application Java Card
  - Construction de l'application serveur (applet)
    - Implémentation de services
  - Installation de l'applet dans les cartes
    - Initialisation de services
  - Construction de l'application cliente
    - Invocation de services

Construction de l'application serveur

- Installation de l'applet Java Card
  - Compilation, conversion et chargement sécurisé de l'applet dans les cartes (Java Card IDE)
  - Appel à la méthode `install(APDU apdu)` des applets (non standardisé)
    - L'APDU contient les paramètres d'initialisation de l'applet

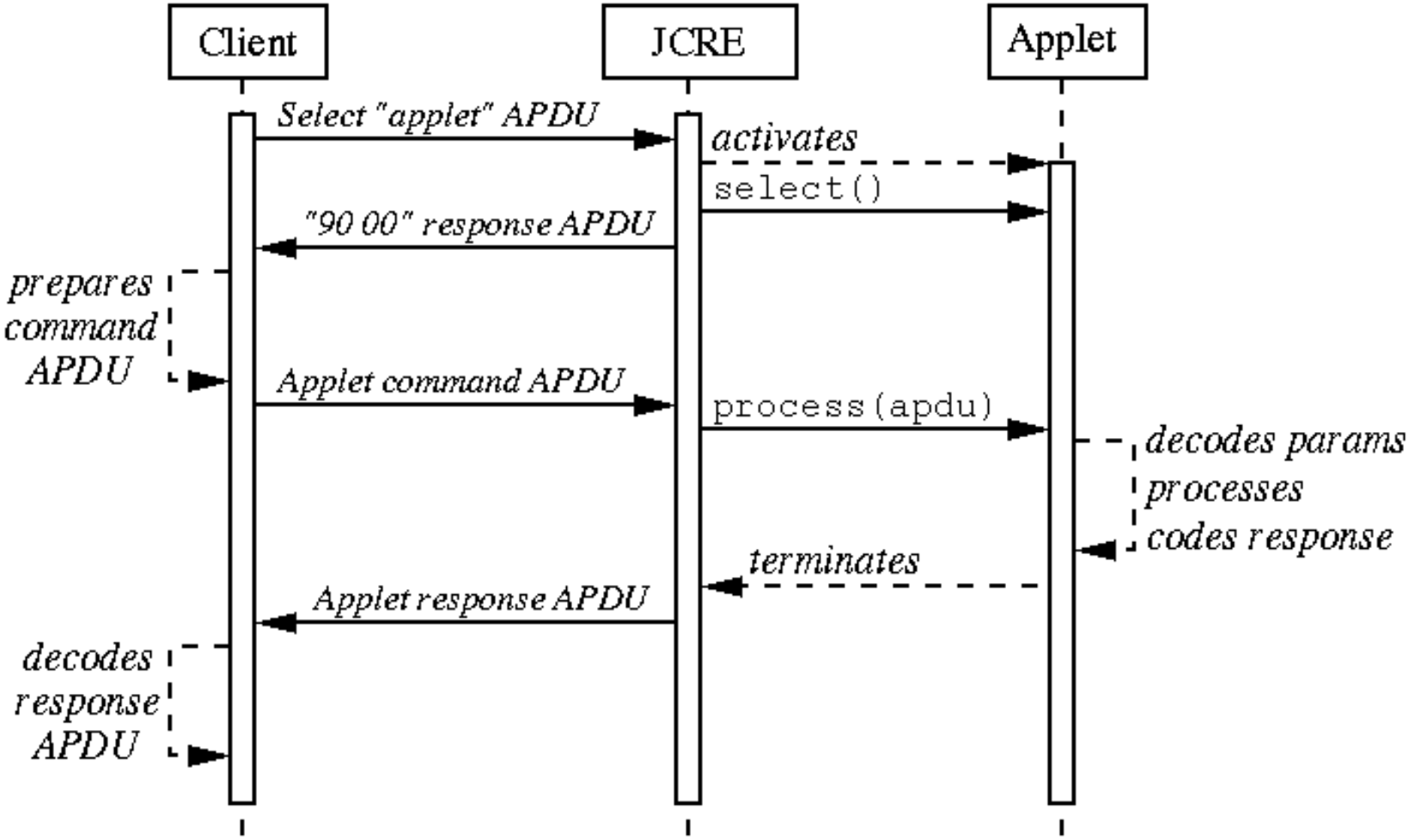
Construction de l'application serveur

- Construction de l'applet Java Card
  - Implémentation des classes de l'applet avec l'API Java Card
  - Définition des APDUs de commande traités par l'applet et des APDUs de réponse renvoyés par l'applet (données ou erreurs)
  - → implémentation de la méthode **`process(APDU apdu)`**
  - Le JCRE fournit l'environnement d'exécution et la couche de communication

Construction de l'application cliente (1/2)

- Construction de l'application terminal
  - Implémentation des classes du terminal (avec JDK)
  - Communication avec le serveur (applet carte)
    - Établissement de la liaison : envoi d'un APDU de sélection avec l'AID de l'applet (standardisé)
    - Invocation de services de l'applet :
      - codage et envoi d'APDUs de commande conformes à ceux traités par l'applet
      - réception et décodage des APDUs de réponse retournés par l'applet
  - Pas d'API standard de communication avec la carte

Construction de l'application cliente (2/2)



Exemple d'application JavaCard

- Un simple Compteur
  - Carte de fidélité, Porte Monnaie Electronique, …
- APDUs traités par l'applet :
  - **`int lire()`**
    - Commande : **`AA 01 XX XX 00 04`**
    - Réponse : **`RV3 RV2 RV1 RV0 90 00`**
  - **`int incrementer( int )`**
    - Commande : **`AA 02 XX XX 04 AM3 AM2 AM1 AM0 04`**
    - Réponse : **`RV3 RV2 RV1 RV0 90 00`**
  - **`int decrementer( int )`**
    - Commande : **`AA 03 XX XX 04 AM3 AM2 AM1 AM0 04`**
    - Réponse : **`RV3 RV2 RV1 RV0 90 00`**

Applet «Compteur» : Classe `Applet`

```
package org.carte.compteur ;

import javacard.framework.* ;

public class Compteur extends Applet {
  private int valeur;

  public Compteur() { valeur = 0; register(); }
  public static void install( APDU apdu ) { new Compteur(); }

  public void process( APDU apdu ) {
    byte[] buffer = apdu.getBuffer();
    if ( buffer[ISO.OFFSET_CLA] != 0xAA )
      ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
    switch ( buffer[ISO.OFFSET_INS] ) {
      case 0x01: ... // Opération de lecture
      case 0x02: ... // Opération d'incrémentation
      case 0x03: ... // Opération de décrémentation
      default:
        ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
  }
}
```

Applet «Compteur» : décrémentation

```
case 0x03: // Opération de décrémentation
{
  // Réception des donnnées
  byte octetsLus = apdu.setIncomingAndReceive();
  if ( octetsLus != 4 )
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);
  int montant = (buffer[ISO.OFFSET_CDATA]<<24) |
    (buffer[ISO.OFFSET_CDATA+1]<<16) |
    (buffer[ISO.OFFSET_CDATA+2]<<8) |
    buffer[ISO.OFFSET_CDATA+3];
  // Traitement
  if ( montant<0 || valeur-montant<0 )
    ISOException.throwIt((short)0x6910);
  valeur = valeur - montant;
  // Envoie de la réponse
  buffer[0] = (byte)(valeur>>24);
  buffer[1] = (byte)(valeur>>16);
  buffer[2] = (byte)(valeur>>8);
  buffer[3] = (byte)(valeur);
  apdu.setOutgoingAndSend((short)0, (short)4);
  return;
}
```

Exemple 2 : PME (1/4)

```
package com.banque ;

import javacard.framework.*;

public class Pme extends Applet {
  final static byte Pme_CLA       = (byte)0xB0;
  final static byte Crediter_INS  = (byte)0x10;
  final static byte Debiter_INS   = (byte)0x20;
  final static byte Lire_INS      = (byte)0x30;
  final static byte Valider_INS   = (byte)0x40;
  final static byte MaxEssai_PIN  = (byte)0x03;
  final static byte MaxLg_PIN     = (byte)0x08;
  final static short BalanceNegative_SW = (short)0x6910;

  OwnerPin pin;
  byte balance;
  byte[] buffer;

  private Pme() {
    pin = new OwnerPIN(MaxEssai_PIN, MaxLg_PIN);
    balance = 0;
    register() ;
  }
```

Exemple 2 : PME (2/4)

```
public static void install(byte[] bArray,
                           short bOffset, byte bLength) {
  Pme p=new Pme();
  pin.updateAndUnblock(bArray, bOffset, bLength);
}

public boolean select() { pin.reset(); return true; }

public void process( APDU apdu ) {
  buffer = apdu.getBuffer();
  if ( buffer[ISO.OFFSET_CLA] != Pme_CLA )
    ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
  switch ( buffer[ISO.OFFSET_INS] ) {
    case Crediter_INS : crediter(apdu); return;
    case Debiter_INS : debiter(apdu); return;
    case Lire_INS : lire(apdu); return;
    case Valider_INS : valider(apdu); return;
    default:
      ISOEXception.throwIt(ISO.SW_INS_NOT_SUPPORTED);
  }
}
```

Exemple 2 : PME (3/4)

```
// Réception de données
private void crediter( APDU apdu ) {
  if ( !pin.isValidated() )
    ISOException.throwIt(ISO.SW_PIN_RIQUIRED);
  byte octetsLus = apdu.setIncomingAndReceive();
  if ( octetsLus != 1 )
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);
  balance = (byte)(balance + buffer[ISO.OFFSET_CDATA]);
}

// Réception de données
private void debiter( APDU apdu ) {
  if ( !pin.isValidated() )
    ISOException.throwIt(ISO.SW_PIN_RIQUIRED);
  byte octetsLus = apdu.setIncomingAndReceive();
  if ( octetsLus != 1 )
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);
  if ( (balance - buffer[ISO.OFFSET_CDATA]) < 0 )
    ISOException.throwIt(BalanceNegative_SW);
  balance = (byte)(balance - buffer[ISO.OFFSET_CDATA]);
}
```

Exemple 2 : PME (4/4)

```
    // Émission de données
    private void lire( APDU apdu ) {
      if ( !pin.isValidated() )
        ISOException.throwIt(ISO.SW_PIN_RIQUIRED);
      apdu.setOutgoing();
      apdu.setOutgoingLength((byte)1);
      buffer[0] = balance;
      apdu.sendBytes((short)0, (short)1) ;
    }

    // Manipulation du code secret
    private void valider( APDU apdu ) {
      byte octetsLus = apdu.setIncomingAndReceive();
      pin.check(buffer, ISO.OFFSET_CDATA, octetsLus);
    }

}
```

Les APIs de programmation Java Card

- Paquetages
  - **`java.lang`**
  - **`javacard.framework`**
    - Principale API pour programmer une applet carte
    - Définit les classes :
      - **`AID`**, **`APDU`**, **`Applet`**, **`ISO`**, **`PIN`**, **`JCSystem`**, **`Util`**
      - Plus des classes d'exceptions
  - **`javacard.security`**
    - gestion de clés publiques et privées, générateur de nombres aléatoires
- Extensions
  - **`javacardx.framework`**
  - **`javacardx.crypto`** : fonction de chiffrement et de hashage...

Les APIs utilitaires de javacard.framework (1/3)

- **`public final class AID`**
  - Encapsule des identifiants d'applications carte conformes à la norme ISO 7816-5
- **`public class Util`**
  - Méthodes statiques (natives) utiles pour performance carte
    - `arrayCopy()` Copie atomique/non atomiques de tableaux de bytes
    - `arrayCopyNonAtomic()` Copie non atomiques de tableaux de bytes,
    - arrayCompare() : Comparaison de tableaux de bytes
    - `arrayFillNonAtomic() : Remplissage d'un tableau avec une valeur entière`
    - `makeShort()` : Création de **`short`** à partir de **`byte`**
- **`public final class JCSystem`**
  - Méthodes statiques (natives) pour interagir avec le JCRE
    - Gestion des transactions (1 seul niveau)
    - Gestion du partage d'objets entre applets
    - Création d'objets transients

Les APIs utilitaires de javacard.framework (3/3)

- Transactions
  - Rendre attomique une section de code.
  - beginTransaction(), commitTransaction(), abortTransaction()
    - Attention : Nombre d'instructions recouverables limité par la RAM et l'EEPROM
- Tableaux transients
  - makeTransientXXXArray(lenght,event)
  - Crée un tableau de XXX réinitialisé
    au Reset : event=`CLEAR_ON_RESET`
    à la désélection : event=`CLEAR_ON_DESELECT`
  - Ex : Authentification, Etat de l'automate, Clé de session, …
- Partage d'objets
  - L'applet crée un objet d'une classe héritant de Shareable
  - L'applet partage l'objet en fournissant sa référence avec
    - getAppletShareableInterfaceObject(AID, parameter)

Les APIs utilitaires de javacard.framework (3/3)

- **`public class ISO`**
  - Champs statiques de constantes conformes aux normes ISO 7816-3 et 4
    - `public static final short SW_NO_ERROR = (short) 0x9000`
    - `public static final short SW_FILE_NOT_FOUND = (short) 0x6A82`
    - `public static final short SW_RECORD_NOT_FOUND= (short) 0x6A83`
    - `public static final short SW_INCORRECT_P1P2 = (short) 0x6A86`
    - `public static final short SW_WRONG_P1P2 = (short) 0x6B00`
    - `public static final short SW_CLA_NOT_SUPPORTED = (short) 0x6E00`
    - `...`
    - `public static final byte CLA_ISO7816= 0x00`
    - `public static final byte INS_SELECT= 0xA4`
    - `public static final byte INS_EXTERNAL_AUTHENTICATE= 0x82`
  - **`ISOException.throwIt(short reason)`**
    - Renvoie la «raison»
- **`public abstract class PIN`**
  - Représentation d'un code secret (tableau d'octets)
    - **`OwnerPIN`** : code secret pouvant être mis à jour

Les APIs Cryptographique de javacard.security et javacardx.crypto

- But: indépendance fonction / algorithmes
  - Gestion de clés publiques et privées
    - **KeyBuilder** est une fabrique de clés
    - Key, DESKey, DSAPrivateKey, DSAPublicKey, PrivateKey, PublicKey, RSAPrivateCrtKey, RSAPrivateKey, RSAPublicKey, SecretKey
  - Générateur de nombres aléatoires : **RandomData**
  - Hachage : **MessageDigest**
  - Signature : **Signature**
  - Chiffrage/Déchiffrage : **Cypher**

Exemple d'usage de la cryptographique

```
private DESKey myDESKey;

public static void install(byte[] bArray,
                           short bOffset, byte bLength) {
    new Encryption ();
    pin.updateAndUnblock(bArray, bOffset, bLength);
 }

  public boolean select() { pin.reset(); return true; }

  public void process( APDU apdu ) {
    buffer = apdu.getBuffer();
    if ( buffer[ISO.OFFSET_CLA] != 0x00 )
      ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
    switch ( buffer[ISO.OFFSET_INS] ) {
      case ENCRYPT_INS : encrypt(apdu); return;
      case PINCHECK_INS : pinCheck(apdu); return;
     default:
        ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
  }
```

Applet Carte (Card Applet)

- Classe dérivant de **`javacard.framework.Applet`**
- Une applet carte
  est un programme serveur de la Java Card
  - APDU de sélection depuis le terminal (select)
  - Sélection par AID (chaque applet doit avoir un AID unique)
    - AID
      - 5 octets identifiant le propriétaire
      - 0-11 octets dépendant du propriétaire

Applet Carte (Card Applet)

- Cycle de vie : Méthodes appelées par JCRE
  - **`static void install(bArray,bOffset,bLength)`**
    - Crée une instance de la classe avec les paramètres passés dans bArray
    - Puis l'enregistre (register()) auprès du JCRE
  - **`boolean select()`**
    - Appelé à la sélection
    - peut retourner false si l'initialisation est incomplète (liaison impossible vers des objets partagés, …)
  - **`void deselect()`**
    - Appelé à la désélection
  - **`void process(APDU apdu)`**
    - Méthodes appelées par JCRE

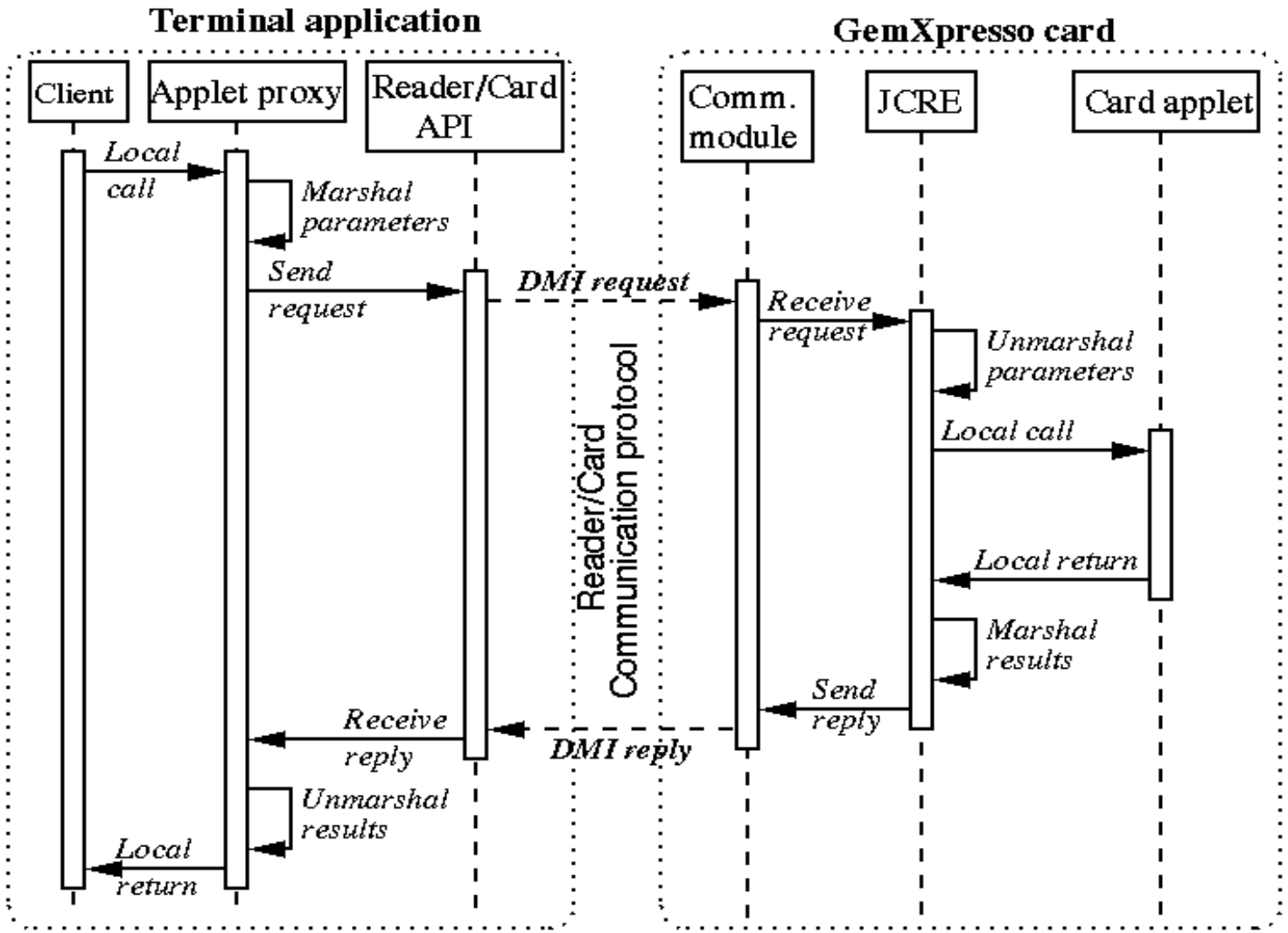
Java Card 2.1

- Format des fichiers de chargement d'applet carte :
  - Format normalisé !!!
  - Fichier .cap identique à .class sauf :
    - 1seul fichier .cap par paquetage Java
- « Firewall » entre applets
  - la VM doit vérifier à l'exécution que le code d'une applet ne sort jamais de son contexte
  - Mécanismes de changement de contexte
    - Objets points d'entrée et tableaux globaux du JCRE peuvent être accédés par les applets (e.g., APDU)
    - Le JCRE peut accéder à n'importe quel objet
    - Interactions entre applets via interfaces partageables Suppresion de la méthode System.share(Object ...)

Java Card 2.2

- Chargement spécifié des .cap
  - l'interopérabilité de JC 2.1 s'arrêtait juste avant la carte...
- Suppression d'objets
- JC-RMI
  - Introduit par la première GemXpresso (ex-DMI)
  - Générateur de Souche-Talon « à la RMI » masquant le encodage/decodage des invocations de méthodes en APDU
    - Rapidité de développement (mise au point, maintenance, …)
  - Souche CardService OCF ??

GemXpresso DMI Architecture



JavaCard 3.0

http://java.sun.com/products/javacard/3.0/

- Sortie en Mars 2008
- Spécification JC séparée en 2 éditions
- *Classic Edition*
  - Compatible avec 2.2.2
  - Architecture matérielle semblable à celle requise par 2.2.2
  - Essentiellement une légère évolution de la spécification 2.2.2
- *Connected Edition*
  - Architecture matérielle requise
    - → Rapprochement de J2ME/CLDC
    - Processor 32b, MMU/noMMU, 40KB RAM, 256KB FlashNOR
  - Modèles d'applications
    - Compatibilité avec les applications dévelopées en *Classic Edition*
    - Applets
    - Servlets pour des interactions Web (HTML, SOAP, XMLRPC, RESTful, …)
  - Runtime
    - Multithreading
    - Objets volatiles (Garbage Collection)
    - TCP/IP HTTP
    - Registre dynamique de « services » entre applications
    - Modèle de partage d'objets basé sur un registre dynamique de services
    - Card Management (déploiement sous la forme de Jar files)
    - ClassLoading delegation
    - …

Outils JavaCard

- JavaCard Kit de Sun
  - converter
  - verifycap
  - installer
  - jcre (emulateur accessible via une socket IP)
  - +maskgen, apdutool, capgen, …
- Kits Fabricants
  - GUI + Plugins pour AGL
    - Eclipse, NetBeans, Visual Studio, …
  - Simulateurs Carte (+ ou - contraints, + ou - realistes)
- Taches Apache Ant

Conclusion Java Card

- Méthodologie de développement d'applets cartes
  - Basée sur Java pour programmer la carte
  - Basée sur APDUs pour communication client-applet
- Points positifs
  - Carte ouverte
  - Langage Java
  - API standard
- Nouveautés arrivent...

Global Platform

- A l'origine VOP (Visa Open Platform)

- Ensemble de commandes APDU pour
  - Authentifier de l'installateur
  - Lister des applications et des bibliothèques installées et initialisées
  - Installer/désinstaller des applications et des bibliothèques
  - Sélectionner d'une application

API de communication avec la carte

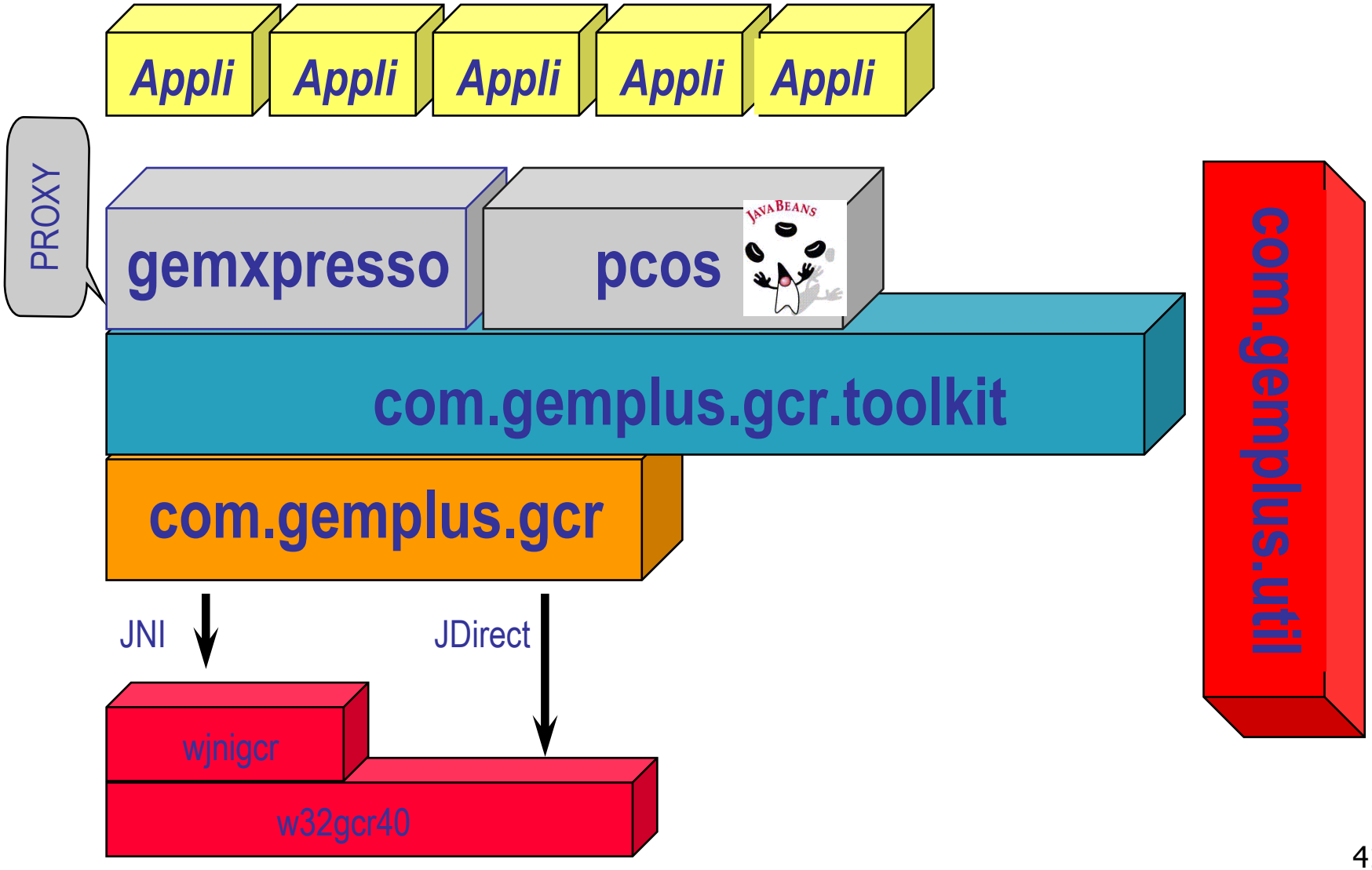
- Open Card Framework (OCF)
  - API Java désormais obsolete
  - `opencard.core.terminal` : abstractions pour les lecteurs, les modes de communication, les connexions/déconnexions avec la carte
  - `opencard.core.service` : «framework» pour la définition de services carte
- Existant
  - PC/SC : API C/C++ Microsoft pour accéder aux cartes sur les plates-formes Windows 32 bits (98 et NT4 et 5)
    - USB CCID 1.0 (Chip/Smart Card Device Interface)
  - pcsc-lite, JPC/SC
  - JSR 268 désormais dans le JRE 1.6

  - API Cliente du Gemplus SDK
    - Tout lecteur Gemplus
    - Java VM

API Cliente du Gemplus SDK

- Paquetage **`com.gemplus.gcr`**
  - Classe **`Ifd`** (Interface Device)
    - Représente le lecteur
    - Gère canaux de communication avec le lecteur
    - Sous-classe pour chaque mode de communication
  - Classe **`Icc`** (Integrated Circuit Card)
    - Représente la carte
    - Gère la connexion à la carte
    - Gère l'échange d'APDUs avec la carte par la méthode :
      ```
      ApduResponse exchangeApdu(ApduCommand command)
        throws GcrException
      ```
  - Classe **`GcrException`** (et sous-classes) pour les erreurs de communication

API cliente du Gemplus SDK



Client «Compteur» : liaison carte

```
package compteur.client.gemplus.fr ;

import com.gemplus.gcr.* ;

/* Application terminal */
Ifd lecteur =
  new IfdSerial(IFDTYPE.GCR410, SERIALPORT.G_COM1, 9600);
Icc carte = new Icc() ;
try {
  // Connexion à la carte via lecteur GCR410
  short canal = reader.openChannel();
  SessionParameters atr = carte.openSession(canal);
  // Échange d'APDUs (APDU de selection de l'applet Compteur)
  ApduCommand commande = new ApduCommand( /* paramètres */ );
  ApduResponse reponse = carte.exchangeApdu(commande);
  /* etc */
  // Fin de la connexion
  carte.closeSession();
  lecteur.closeChannel(canal);
} catch ( GcrException e ) {
  // Récupération de l'erreur
  System.out.println(''Problème : '' + e.getMessage());
}
```

Client «Compteur» : décrémentation

```
// Commande = AA 03 XX XX 04 AM3 AM2 AM1 AM0 04
// Reponse = RV3 RV2 RV1 RV0 90 00 ou 69 10
int montant = System.in.read();
byte[] montantApdu = new byte[4];
montantApdu[0] = (byte)(montant >> 24);
montantApdu[1] = (byte)(montant >> 16);
montantApdu[2] = (byte)(montant >> 8);
montantApdu[3] = (byte)(montant);
ApduCommand commande =
  new ApduCommand(0xAA, 0x03, 0, 0, montantApdu, 4);
ApduResponse reponse = carte.exchangeApdu(commande);
if ( reponse.getShortStatus() == 0x9000 ) {
  byte[] apduValeur = reponse.getDataOut();
  int valeur = (apduValeur[0]<<24) |
    (apduValeur[1] <<16) | (apduValeur[2]<<8) |
    apduValeur[3];
  System.out.println(‘‘Valeur compteur : ‘‘ + valeur);
} else {
  if ( reponse.getShortStatus() == 0x6910 )
  { /* Traite l’erreur «Valeur négative» */ }
}
```

OCF 1.1 : Open Card Framework

- Framework standard d'accès à des cartes et des lecteurs depuis un environnement Java
  - Drivers Terminal doit être implémenté et intégré dans le framework par chaque fabricant
    - Accessible depuis la classe CardTerminal
    - Plusieurs possibilités
      - Driver natif accessible depuis JNI
      - Driver PC/SC
      - Driver Java utilisant l'API javax.comm
  - Chaque carte est représentée par une classe CardService
- Voir le cours OCF
  - http://www-adele.imag.fr/users/Didier.Donsez/cours/ocf.pdf

MuscleCard

- Javacard Applet d'identifitation
  - Licence BSD
  - http://www.linuxnet.com/musclecard/index.html

Benchmarks JavaCard

- Voir la thèse de doctorat de Julien Cordry, CNAM Paris, 30/11/2009.

Bibliographie

- JAVACARD
  - Zhiqun Chen , "Java Card Technology for Smart Cards: Architecture and Programmer's Guide (The Java Series) ", 1 edition (June 2, 2000) , Addison-Wesley Pub Co; ISBN: 0201703297,
    - http://java.sun.com/docs/books/javacard/
  - Site Sun : **http://java.sun.com/products/javacard**
  - Java Card Forum : **http://www.javacardforum.com/**
  - Gemplus
    - http://www.gemplus.fr/developers/technologies/javacard/
- Une introduction
  - http://wireless.java.sun.com/javacard/articles/javacard3/

Vos suggestions et vos remarques

- Merci de me les retourner à
  - Didier DONSEZ, didier.donsez@imag.fr
- Avez vous trouvé ce cours instructif ?
  - Est il complet ?
  - Qu 'est qu 'il manque ?
  - Qu 'est que vous auriez aimé voir plus développé ?
  - Est il bien organisé ?
  - ...
- Quels sont votre fonction et votre domaine d'activité ?

Remerciement à

- Jean-Jacques Vandewalle (Gemalto R&D)
- Pierre Paradinas (CNAM/INRIA)
- Alain Rhélimi (Gemalto)