

Java Card™

Damien Sauveron
damien.sauveron@unilim.fr
<http://damien.sauveron.fr/>

1

Intervenants dans le module

Damien Sauveron (Cours, TD et TP) :

- Membre de l'équipe « Smart Secure Devices » (Périphériques Sécurisés Intelligents)
- Thème de recherche : sécurité des cartes à puce et des réseaux ad hoc

Pour me contacter :



Damien Sauveron
XLIM UMR 6172 CNRS -- Université de Limoges
123 avenue Albert Thomas
87060 Limoges Cedex, FRANCE

Email: Damien.Sauveron@unilim.fr
Web: <http://damien.sauveron.fr/>
Phone: +33 (0) 5 87 50 67 93

MCF à l'Université de Limoges depuis septembre 2006

Visite postdoctorale au Smart Card Centre de l'Information Security Group du Royal Holloway University of London

Thèse au LaBRI – Université Bordeaux 1 sur la Sécurité de la Technologie Java Card

Ingénieur R&D dans le CESTI de SERMA Technologies (Pessac)

Organisateur de WISTP2007 (Héraklion) et WISTP2008 (Séville)

Vice président du groupe de travail de l'IFIP : Pervasive System Security

2

Plan

Les concepts

- Qu'est-ce que Java Card ?
- Statut du standard Java Card
- Sous-ensemble du langage Java
- La machine virtuelle
- Le modèle mémoire

Les sécurités

Les attaques

La pratique

- Concepts de programmation
- Les APIs de programmation
- Développement d'une applet Java Card
- Construire une application avec la Java Card

3

État de l'art des cartes avant Java Card

La carte à puce est un véritable ordinateur utilisé comme serveur portable et sécurisé de données personnelles

Elle est programmée comme un composant embarqué avec un code applicatif figé

Elle est difficile à intégrer dans les systèmes d'informations

- Car si l'application requiert de nouvelles fonctions carte il faut développer un nouveau masque (=> coûteux)
=> Sinon il faut prévoir un masque qui accepte d'exécuter des programmes en EEPROM (rare et pas standardisé)
- Car pas d'interface standard de communication avec les différents lecteurs (travaux en cours – problème connexe)
 - Communication via APDUS (très bas niveau)

Problèmes à résoudre et/ou besoins à satisfaire

- Permettre le développement de programmes pour la carte sans avoir besoin de graver un nouveau masque
 - Faciliter et accélérer les développements de codes dans la carte
- Faire de la carte un environnement d'exécution de programmes ouvert (chargement dynamique de code)
 - Rendre plus souples et plus évolutives les applications carte
- Faciliter l'intégration des cartes dans les applications
 - Faciliter et accélérer les développements d'applications clientes des cartes

Besoins :

- Diversification des sources cartes et des terminaux
- Pérennité des développements

Procédé de construction des applications

- Permettre le développement «rapide» d'applications carte «évoluées» par des «non spécialistes»
- But : de nouvelles applications pour de nouveaux marchés !

4

Qu'est ce que Java Card ?

Technologie permettant de faire fonctionner des applications écrites en langage Java pour :
carte à puce

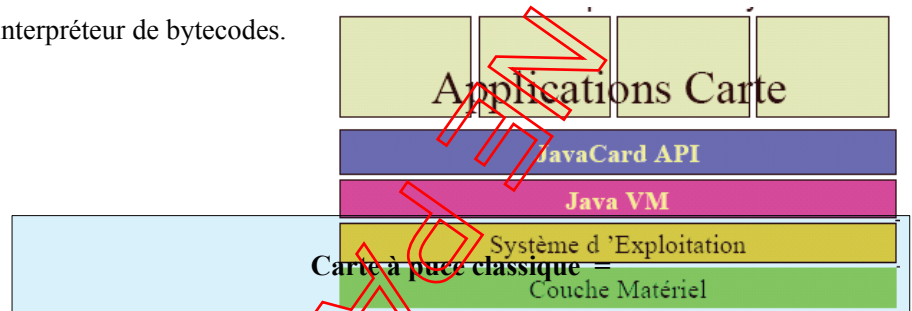


autres périphériques à mémoire limitée (par exemple iButton)



De manière intuitive :

Il s'agit d'une carte basée sur un interpréteur de bytecodes.



La technologie Java Card définit une plate-forme pour cartes à puce sécurisée, portable et multi-applicative qui incorpore beaucoup des avantages du langage Java.

5

Historique 1/2

En Novembre 1996, un groupe d'ingénieurs de Schlumberger cherche à simplifier la programmation des cartes à puces tout en préservant la sécurité (Cyberflex 1.0 au CNIT – Cartes'96).

==> la spécification Java Card 1.0

En Février 1997, Bull et Gemplus se joignent à Schlumberger pour cofonder le « Java Card Forum ».

Consortium de fabricants :

- Cartes (Gemplus, Schlumberger, etc)
- Informatique : IBM, Sun
- Matériel : DEC, Motorola
- Utilisateurs : Banques (CitiBank, etc.)

Buts :

- promouvoir Java Card
- Faire des choix technologiques communs
- Contre pouvoir à Sun (officieusement)

Solutions :

- Un comité technique
- Un comité business

<http://www.javacardforum.org/>

En Novembre 1997, Sun présente les spécifications Java Card 2.0.

1998 : Cyberflex 2.0, GemXpresso, ...

Historique 2/2

En Mars 1999 sort la version 2.1 des spécifications Java Card. Elles consistent en trois spécifications :

- The Java Card 2.1 API Specification.
 - The Java Card 2.1 Runtime Environment Specification.
 - The Java Card 2.1 Virtual Machine Specification.
- Contribution la plus significative :
- Définition explicite de la machine virtuelle de la Java Card.
 - Le format de chargement des applets.

En Mai 2000, sort une petite correction ==> version 2.1.1

En Octobre 2000, plus de 40 entreprises ont acquis la licence d'exploitation de la technologie Java Card.

En juin 2002, spécifications Java Card 2.2.

Introduction de RMI, des canaux logiques, de nouveaux algorithmes cryptographiques, ...

En octobre 2003, spécifications Java Card 2.2.1.

En novembre 2005, la version 2.2.2 est Public Review Draft

En décembre 2005, les licenciés sont :

ARM, Aspects, Axalto, CCL/ITRI, Fujitsu, Gemplus, Giesecke & Devrient GmbH, GoldKey Technology, HiSmarTech, I'M Technologies Ltd., IBM, incard, KEBTechnology, Logos Smart Card, Microelectrónica Española, Oberthur Card Systems, ORGA Kartensysteme, SAGEM, Sermepa, Setec, Sharp, Smart Card Laboratory Inc., SSP Solutions, Inc., STMicroelectronics, Toppan Printing, Trusted Logic. + d'autres qui préfèrent rester anonymes.

7

JavaCard Forum Members (as of 10th Jan 2005)



8

Les avantages de la technologie Java Card

La facilité de développement des applications} grâce :

- à la programmation orientée objet offerte par Java
- à l'utilisation des environnements de développement existants pour Java.
- à une plate-forme ouverte qui définit des APIs et un environnement d'exécution standard.
- à l'encapsulation de la complexité sous-jacente du système des cartes à puce.

L'indépendance au hardware réalisée grâce au langage Java

==> « Write Once, Run Anywhere »

La gestion dynamique de multiples applications

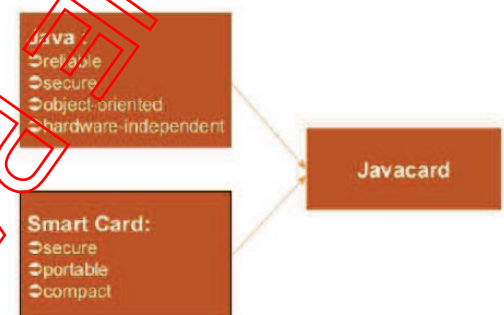
- ==> possibilité de mises à jour des applications de la Java Card sans avoir besoin de changer de cartes
- ==> possibilité d'offrir des nouveaux services

La compatibilité avec les standards existants pour les cartes à puce

ISO7816 :

- Protocole de communication
- Protocole de haut niveau pour la gestion de fichiers, etc

Java Card = Java + smart Card



On verra les inconvénients plus tard.



La démocratisation



Carte à puce standard

- Application, OS et hardware sont liés
- L'application est développée uniquement par le propriétaire de l'OS
- L'application est développée dans un langage de bas niveau (C, Assembleur)
- Cycle de développement = 5 mois
- Pas de vraie multi-application (données uniquement)



Plate-forme Java Card

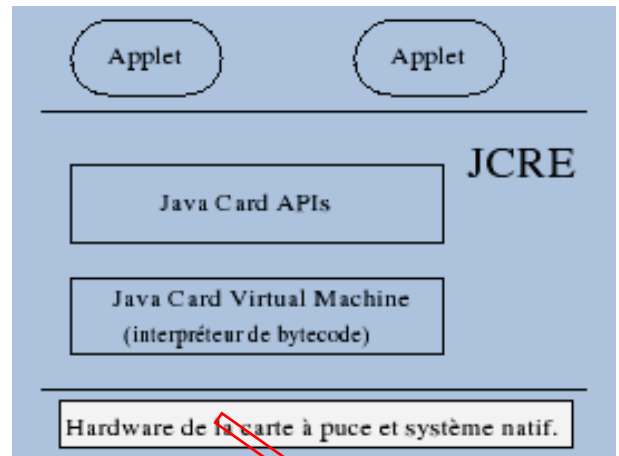
- Application, OS et hardware sont indépendants
- L'application est développée par n'importe quel programmeur Java
- L'application est développée dans un langage standard (de haut niveau)
- Cycle de développement = 2 mois
- Carte multi-application (données + code)

Son architecture

Problème : contraintes mémoires et faible ressource de calculs

Solutions :

- un sous-ensemble des caractéristiques du langage Java
 - Au niveau du langage
 - Au niveau des APIs
- découper la machine virtuelle Java en deux parties
 - Une partie interprète embarquée
 - Une partie vérification hors carte



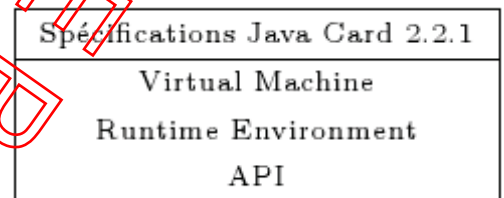
Problème : pas de vérificateur embarqué

Solution :

fournir des mécanismes sécuritaires avec l'environnement d'exécution

Java Card est un sur-ensemble d'un sous-ensemble du langage Java

- Nouveau modèle mémoire utilisant la persistance
- APIs spécifiques à la carte



11

Caractéristiques du langage 1/2

Pas de chargement dynamique de classes

- Classes de base présentes dans la carte (avec le masque)
- Nouvelles classes chargées dans la carte ne doivent référencer que des classes «connues»

Objets

- Instances de classes ou de tableaux à une dimension
- Allocation dynamique d'objets supportée (new)
- Pas de clonage d'objets (méthode equals() supportée)

Pas de ramasse-miettes (gc)

- Pas de désallocation explicite non plus ==> mémoire allouée peut ne pas être récupérée
- Pas de méthode finalize()

Types de base (nombres signés, complément à 2) :

- byte, short, boolean (8 bits), int (optionnel)
- En l'absence du type int, les calculs intermédiaires ou non assignés doivent toujours être «castés» en byte ou short
- Pas de types char (pas de classe String), double, float et long
- Pas de modifieur transient (utilisé pour la sérialisation)
- Pas de classes Boolean, Byte, Class, etc.

Tableaux à une dimension avec éléments :

- Types de base
- Objets (les tableaux sont eux-mêmes des objets)
- Maximum 2^{15} éléments

12

Caractéristiques du langage 2/2

Pas de threads

Pas de classe `Thread`, pas de mots-clés `synchronized` ni `volatile`

Mécanisme d'héritage identique à Java

Surcharge de méthodes, méthodes abstraites et interfaces
Invocation de méthodes virtuelles
Mots-clés `instanceof`, `super` et `this`

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

Mécanisme d'exceptions supporté

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`)

Supportées pour les classes de base (masquées)
Optionnelles pour les nouvelles classes chargées

Atomicité

Mise à jour de champs d'objets doit être atomique
Modèle transactionnel : `beginTransaction()`, `commitTransaction()` et `abortTransaction()`

13

Résumé des caractéristiques du langage

Caractéristiques Java supportées

Type simple de donnée de petite taille : `boolean`, `byte`, `short`
Tableaux à une dimension
Paquetages Java, classes, interfaces et exceptions
Caractéristiques orientées objets : héritage, méthodes virtuelles, surcharge et création dynamique d'objets, contrôle d'accès
Le mot clé `int` et le support des entiers sur 32 bits sont optionnels

Caractéristiques Java ~~non~~ supportées

Type simple de donnée de grosse taille : `long`, `double`, `float`
Tableaux à plusieurs dimensions
Caractères et chaînes
Chargement dynamique des classes
`Security Manager`
Ramasse-miettes et finalisation
Threads
Sérialisation d'objet
Clonage d'objet

14

Résumé des caractéristiques du langage

Mots-clés non disponibles :

char, double, float, long, synchronized, transient, volatile

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

15

La machine virtuelle Java Card : JCVM

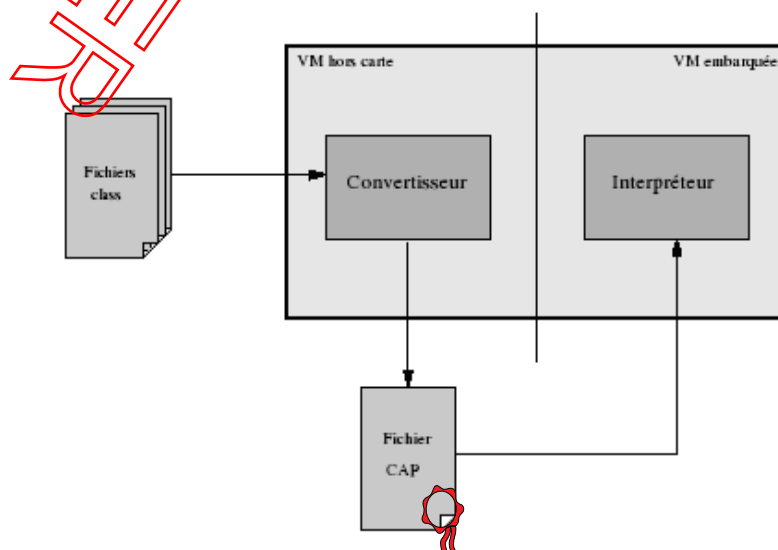
Définition de la machine virtuelle Java

Vérifieur de Bytecode

Chargeur dynamique de classes

Interpréteur de Bytecode

Les deux parties implémentent toutes les fonctions d'une machine virtuelle.



À cause du découpage de la JCVM, la plate-forme est distribuée dans le temps et dans l'espace.

Il faudra des mécanismes pour garantir que ce qui est produit par la partie hors carte n'est modifiée avant d'arriver à la partie embarquée (mécanisme de DAP – Data Authentication Pattern -- et signature) : cf. GlobalPlatform¹⁶

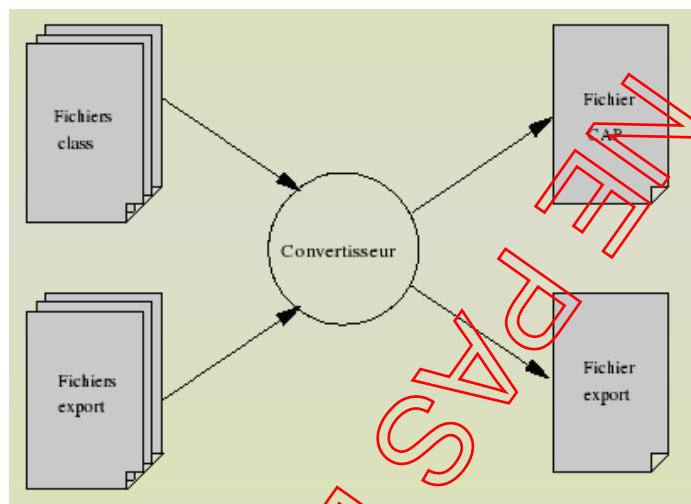
La machine virtuelle Java Card (1/2)

Le convertisseur Java Card

Le convertisseur réalise les tâches que la machine virtuelle Java sur une station de travail doit réaliser au chargement des classes :

- Vérifie que le chargement des images des classes Java sont bien formées.
- Contrôle des violations du langage Java Card.
- Réalise des initialisations de variables `static`.
- Résout les références symboliques aux classes, méthodes et champs dans la forme la plus compact qui peut être traitée efficacement sur la carte.
- Optimise le bytecode en tirant avantage des informations obtenues au chargement des classes et au « linkage ».
- Alloue l'espace et crée les structures de données de la machine virtuelle pour représenter les classes.

Il produit un fichier CAP et un fichier export pour le packaging converti



17

La machine virtuelle Java Card (2/2)

Fichier CAP

Le fichier CAP est le format standard de fichier pour la compatibilité binaire de la plate-forme Java Card.

Un fichier CAP contient une représentation binaire exécutable des classes d'un package Java.

Un fichier CAP est un fichier JAR qui contient un ensemble de composants.

Chaque composant décrit un aspect du contenu d'un fichier CAP comme les informations sur les classes, les bytecodes exécutables, les informations de « linkage », les informations de vérifications, etc.

Fichier export

Un fichier export contient les informations publiques sur les API pour un package de classes complet.

Interpréteur Java Card

L'interpréteur Java Card fournit le support d'exécution du modèle du langage Java qui autorise une indépendance au hardware du code de l'applet.

L'interpréteur réalise les tâches suivantes :

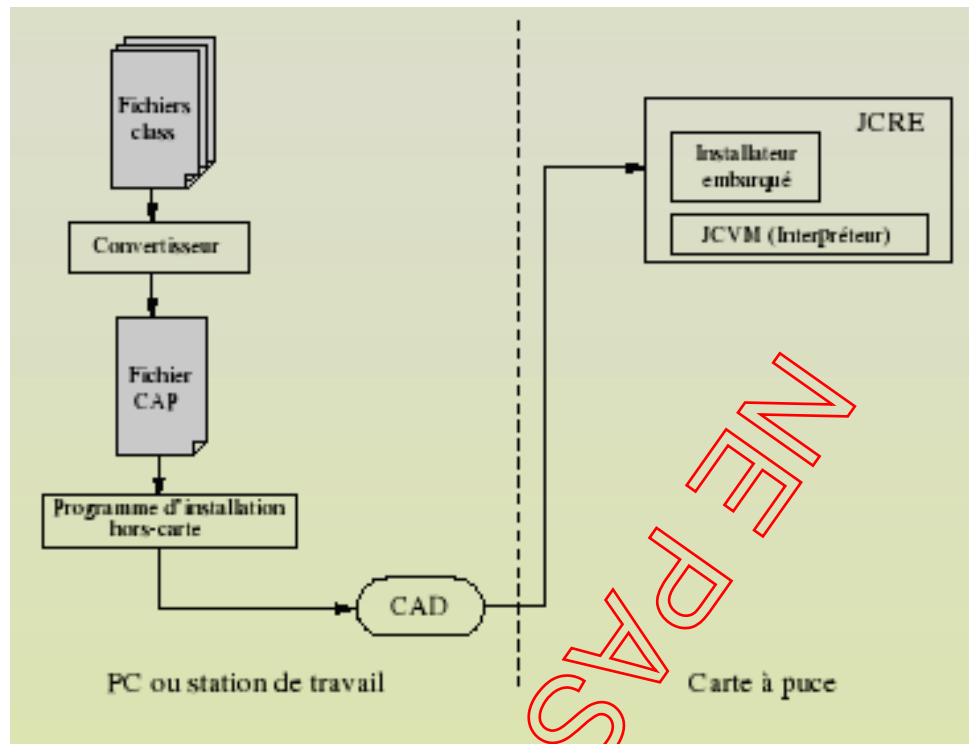
- Exécute les instructions du bytecode et ultimement les exécutions des applets.
- Contrôle les allocations de mémoire et les créations d'objets.
- Joue un rôle crucial pour s'assurer de la sécurité lors de l'exécution.

18

Installateur Java Card

L'installation se fait grâce à deux entités :

l'installateur Java Card et un programme d'installation hors carte



19

L'environnement d'exécution de la Java Card

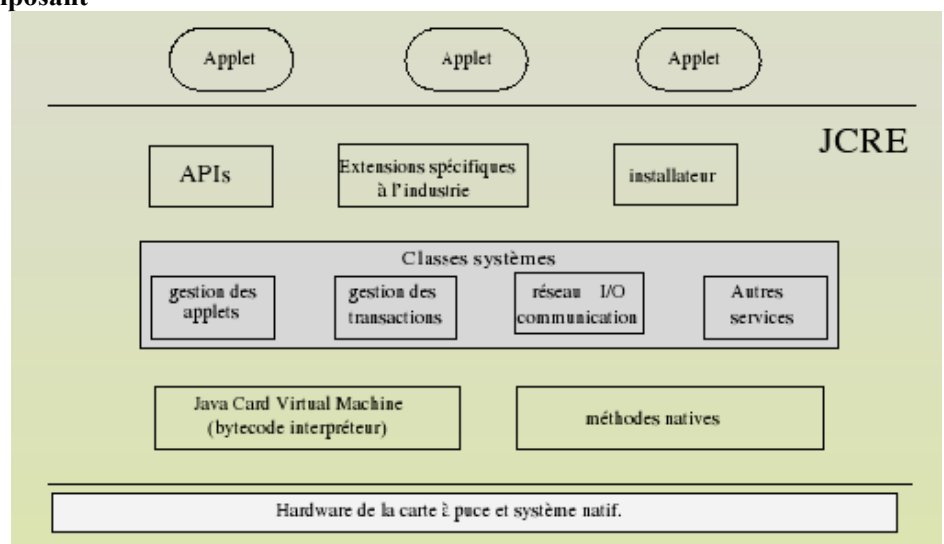
Le JCRE est responsable de la gestion des ressources de la carte, communication réseau, exécution des applets, du système de la carte et de la sécurité des applets.

Gestion des applets

- Installation, enregistrement et initialisation
- Sélection et désélection
- Transmission des APDUs
- Récupération des exceptions

Méthodes natives d'appel au composant

- Entrées/sorties
- Transactions
- Cryptographie



Le cycle de vie du JCRE

Le JCRE est initialisé à la phase d'initialisation de la carte (seulement une fois dans le cycle de vie de la carte).

Le cycle de vie du JCRE est le même que le cycle de vie de la carte.

Quand la tension est enlevée, la machine virtuelle est seulement suspendue.

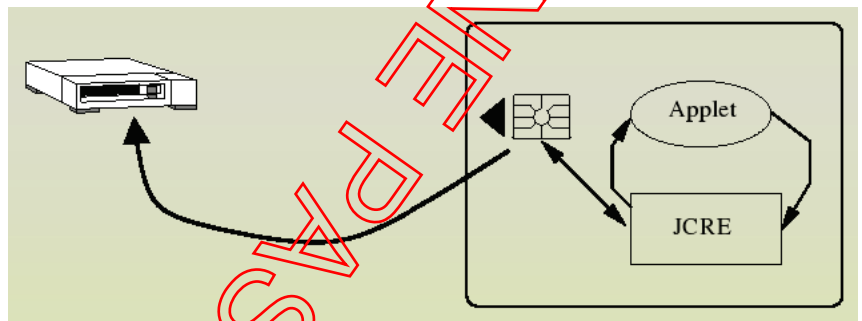
L'état du JCRE et des objets créés sur la carte est préservée.

À la prochaine remise sous tension, le JCRE relance l'exécution de la machine virtuelle en chargeant les données depuis la mémoire persistante.

Durant le reset, si une transaction n'a pas été terminée, le JCRE réalise tous les nettoyages nécessaires pour remettre le JCRE dans un état cohérent.

Comment le JCRE opère durant les sessions CAD ?

le JCRE opère comme une carte à puce typique.



(À comparer à une carte classique)

21

Les caractéristiques du JCRE

Objets persistants et temporaires

Objets persistants existent à travers les sessions CAD.

Objets temporaires contiennent des données temporaires qui ne persistent pas au travers des sessions CAD.

Atomicité et transactions

Chaque opération d'écriture de la JCVm est atomique.

Le champ mis à jour prend soit la nouvelle valeur soit il est restauré à la valeur précédente.

Une transaction est un bloc d'opérations atomiques.

`` Firewall `` des applets et les mécanismes de partage

Le `` firewall `` isole les applets à l'intérieur de leur espace (contexte).

Si les applets ont besoin d'accéder à des données ou à des services du JCRE, la machine virtuelle autorise de telles possibilités à travers des mécanismes sécurisés de partage.

Les APIs Java Card

Ensemble de paquetages optimisés pour la programmation des cartes à puce en accord avec le modèle ISO7816.

java.lang : un sous ensemble strict de son équivalent sur la plate-forme Java

javacard.framework : définit les classes et interfaces essentielles pour le fonctionnement des applets Java Card. **Principale API pour programmer une applet carte**

Définit les classes / interfaces :

- AID, APDU, Applet, ISO7816, PIN, OwnerPIN, JCSysSystem, Util
- Plus des classes d'exceptions

javacard.security : Définit les fonctions cryptographiques (algorithmes et types de clés) supportées sur la plate-forme Java Card

Extensions:

javacardx.framework : gestion de fichiers conforme à la norme ISO 7816-4

javacardx.crypto : un paquetage d'extension pour des algorithmes cryptographiques qui serait interdit à l'exportation. Générations de clés publiques et privées, générateur de nombres aléatoires, fonction de chiffrement et de hashage...

23

Cycle de développement d'une applet (application Java Card)

Une applet est un programme serveur de la Java Card

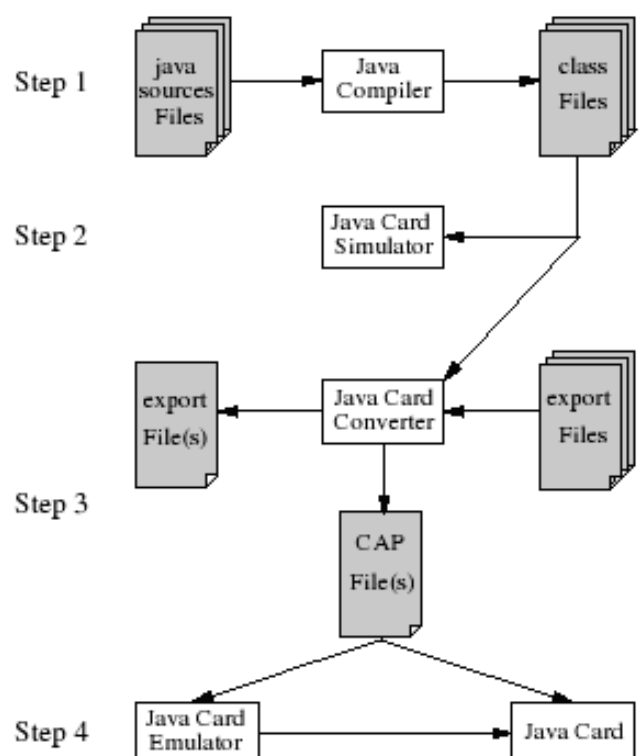
Le fichier CAP est chargé sur la carte

- Il possède un AID
- Ensuite une ou plusieurs applets peuvent être installées
- Si l'AID n'est pas précisé à l'installation de l'applet elle prend celui par défaut préciser lors de la fabrication du paquetage

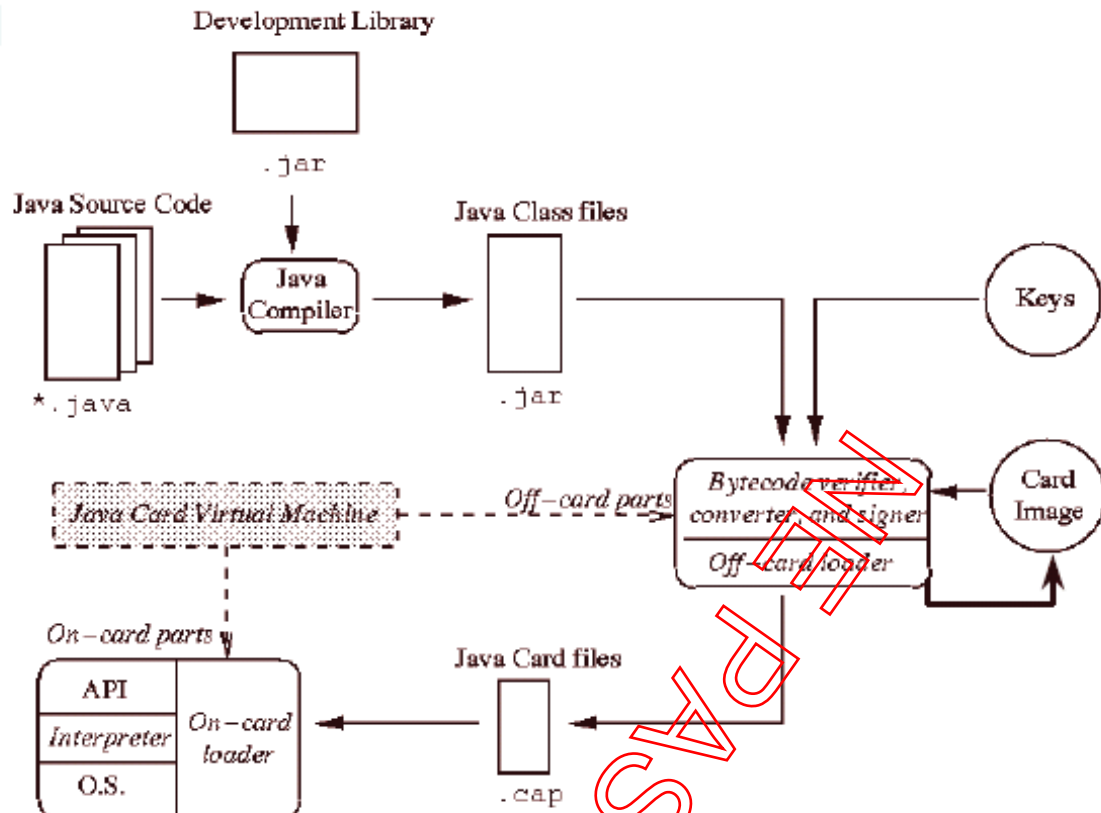
Un AID :

RID (5 octets)	PIX (0-11 octets)
----------------	-------------------

Un fichier CAP = un paquetage



Architecture globale



Évolutions futures ?

Java Card 3.0 (spécifications prévues pour 2007) :

- Multi-threads
- TCP/IP, SSL, ...
- Nouveau modèle qui permet à la carte d'être client

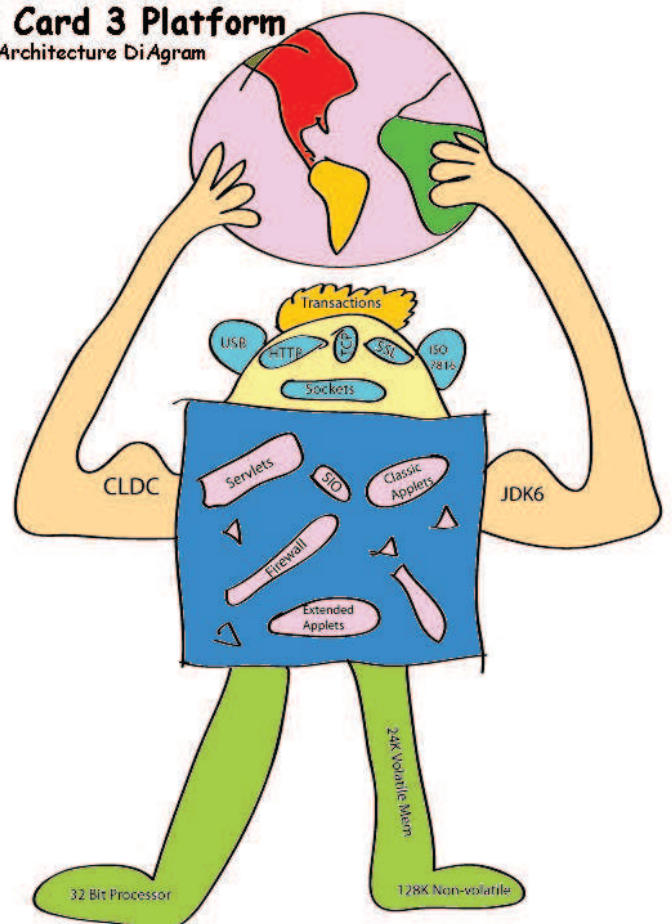
Futurs domaine d'applications :

- Sécurité end to end : protection de contenu.
- Sécurité des Réseaux
- Broadcast Video
- Téléphone 4G
- DRM

A little explanation. At the hardware level, Java Card stands on 32 bit processor and 24K of Volatile and 128K of No-Volatile. The strengths are JDK 6 and CLDC. The interfaces to and from the card are, USB, HTTP, SSL, Sockets all latest protocols. The application models are Servlets/Applets/SIOs

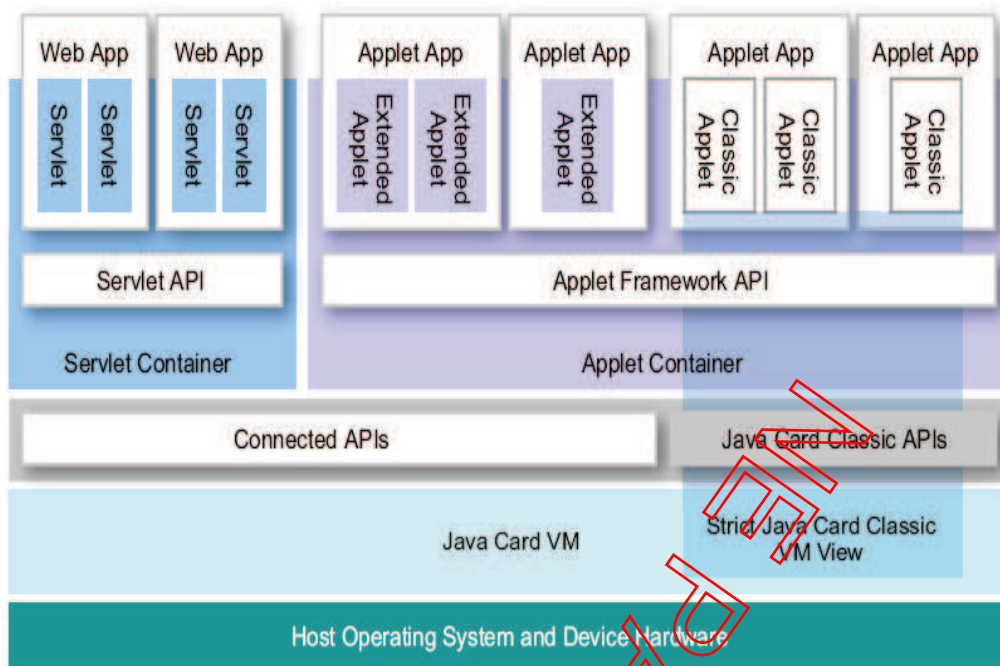
Java Card 3 Platform

Not An Architecture Diagram



Java Card 3.0

Cf slides TL



27

Java Card : Sa sécurité

28

Les sécurités

fournies par le langage

- plusieurs niveaux de contrôle d'accès aux méthodes et aux variables (public, package, protected, private)
- un langage fortement typé
- l'impossibilité de construire des pointeurs

fournies par le runtime

Elles sont de différentes natures :

- purement logicielle :
 - un `` firewall ''
- purement matérielles
 - les transients
 - les transactions

Assurées par l'utilisation de méthodes formelles, d'outils d'aide au développement, ...

29

La sécurité du langage

Convertisseur Java Card

Réalise les tâches que la machine virtuelle Java sur une station de travail doit réaliser au chargement des classes :

- Vérifie que le chargement des images des classes Java sont bien formées.
- Contrôle des violations du langage Java Card.
- Réalise des initialisations de variables `static`.
- Résout les références symboliques aux classes, méthodes et champs

Il embarque la plupart du temps un vérifieur de bytecode.

L'interpréteur de bytecodes

Réalise les tâches suivantes :

- Exécute les instructions du bytecode et ultimement les exécutions des applets.
- Contrôle les allocations de mémoire et les créations d'objets.
- Joue un rôle crucial pour s'assurer de la sécurité lors de l'exécution.

Il existe des machines virtuelles :

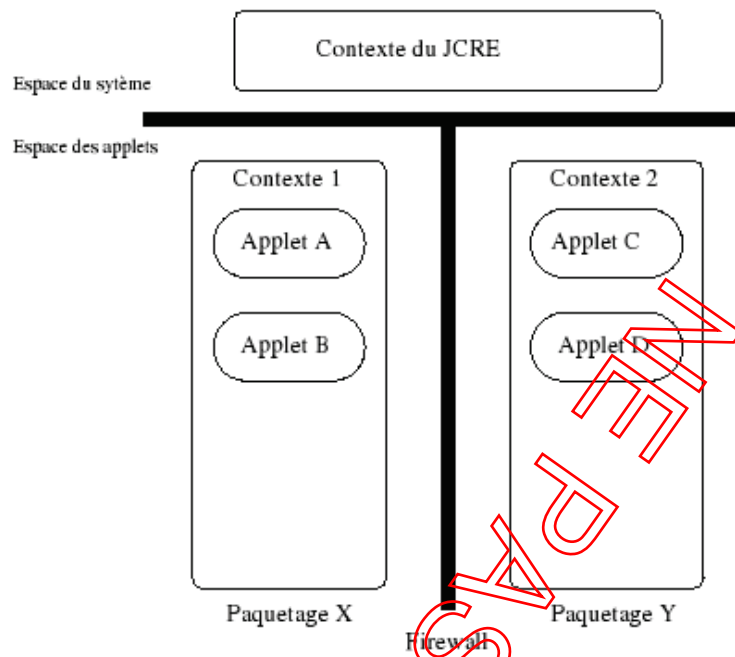
- défensives (beaucoup de vérification à l'exécution)
- offensives (aucune vérification à l'exécution) => fait confiance au vérifieur

30

“ Firewall” des applets et les mécanismes de partage

Isole les applets à l'intérieur de leur espace (contexte).

Si les applets ont besoin d'accéder à des données ou à des services du JCRE, il existe des mécanismes sécurisés de partage.



31

Java Card : les attaques

32

Les attaques

Les attaques externes :

=> attaques identiques à celles des cartes classiques.

Les attaques internes :

détection de services ;
scan d'applets / usurpation d'identité (spoofing) ;
identification physique de service ou de bytecodes.

33

Identification physique (1/2)

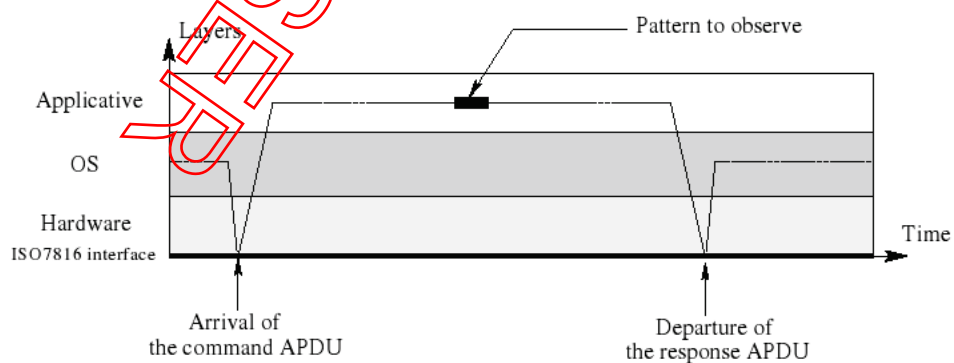


FIG. 1 – Trace d'une exécution normale.

34

Identification physique (2/2)

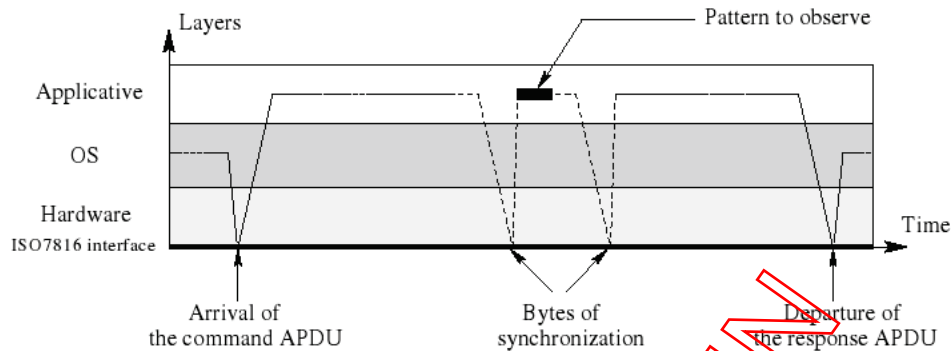


FIG. 2 – Trace d'une exécution glitchée.

35

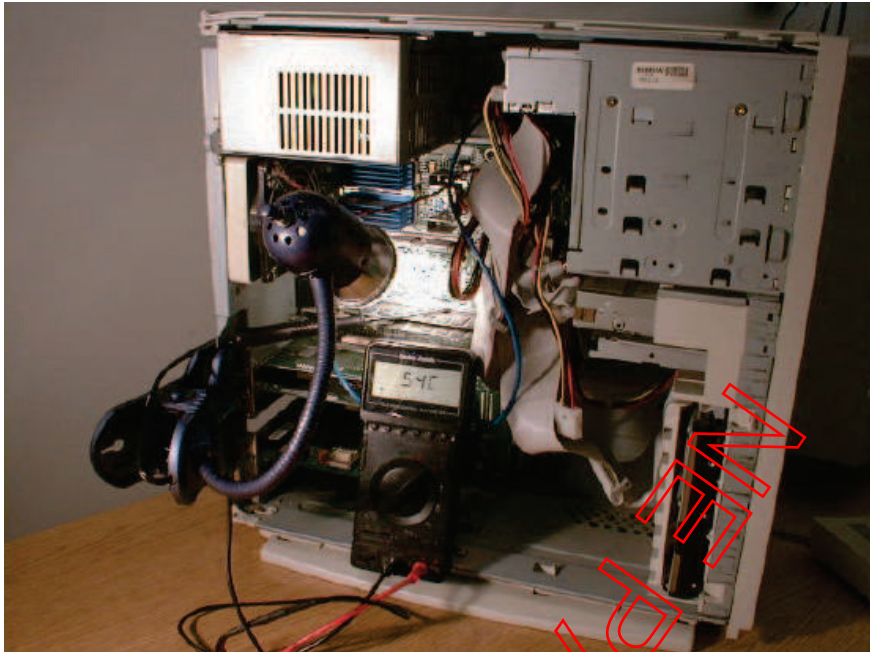
Si absence de typage sur la pile (VM offensive)

```
short readShortAtAddress(short Address) {  
  sload_1 // Get the Address local variable. We hope this short will be considered as a reference  
  sconst_0 // Represent the index 0 in the false array starting at Address  
  saload // Try to get one short and push it on the stack. If this operation succeeds the stack  
          // is not well implemented since the saload bytecode should only accept that the first  
          // argument is of reference type (and not a short type as Address) and that the  
          // object referenced is also a short array  
  sreturn  
}
```

36

Attaques software/hardware

Pour forger des pointeurs et ensuite ...



<http://www.cs.princeton.edu/~sudhakar/papers/memerr-slashdot-commentary.html>

37

Développement Java Card

38

Applet Java Card

Une applet carte est un programme serveur de la Java Card

APDU de sélection depuis le terminal

Sélection par AID (chaque applet doit avoir un AID unique)

Une fois installée dans la carte, est toujours disponible

Classe qui hérite de `javacard.framework.Applet`

Doit implémenter les méthodes qui interagissent avec le JCRE :

`install()`, `select()`, `deselect()` et `process()`

```
public static void install( byte[] bArray, short bOffset, byte bLength )
```

Appelée (une fois) par le JCRE quand l'applet est chargée dans la carte

Doit s'enregistrer auprès du JCRE (méthode `register()`)

```
public boolean select()
```

Appelée par le JCRE quand un APDU de sélection est reçu et désigne cette applet

Rend l'applet active

```
public abstract void process( APDU apdu )
```

Appelée par le JCRE quand un APDU de commande est reçu pour cette applet (doit être active)

```
public void deselect()
```

Appelée par le JCRE pour désélectionner l'applet courante

39

APDU

L'unité de traitement de base d'une applet est un objet de type `javacard.framework.APDU`

Transmis par le JCRE à la réception d'un APDU de commande par la carte

- Appel à la méthode `process()` de l'applet courante

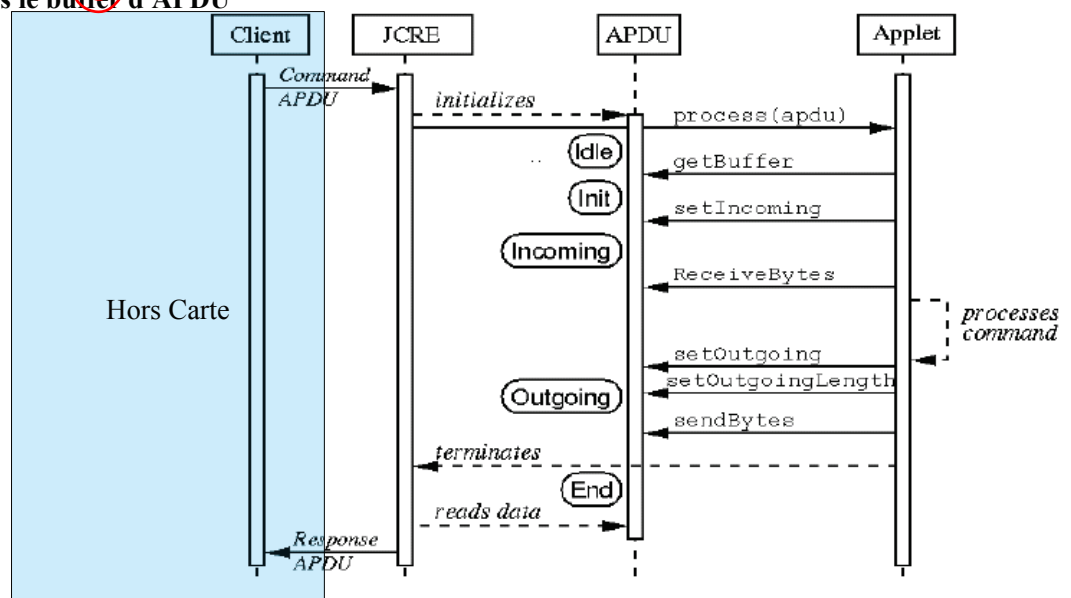
Classe `javacard.framework.APDU`

Compatible avec le format de messages ISO 7816-4

Cache les caractéristiques des protocoles de communication bas niveau (T=0 ou T=1)

Encapsule les échanges de messages APDUs (commandes et réponses) dans un buffer d'entrées/sorties

Lecture/écriture dans le buffer d'APDU



Les API utilitaires de `javacard.framework`

`public final class JCSys`

Méthodes statiques (natives) pour interagir avec le JCRE
Gestion des transactions (1 seul niveau)
Gestion du partage d'objets entre applets

`public class Util`

Méthodes statiques (natives) utiles pour performance carte
Copie, comparaison de tableaux de bytes
Création de short à partir de byte

`public class AID`

Encapsule des identifiants d'applications carte conformes à la norme ISO 7816-5

`public interface ISO7816`

Champs statiques de constantes conformes aux normes ISO 7816-3 et 4

`ISOException.throwIt(short reason)`

Renvoie la «raison»

`public interface PIN`

Représentation d'un code secret (tableau d'octets)
`class OwnerPIN`: code secret pouvant être mis à jour

41

Nature des objets

Objets persistants

Un objet persistant est stocké en mémoire persistante : entête + champs.

Un objet persistant **devra** être créé lorsque :

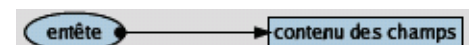
- la méthode `Applet.register` est appelée (enregistrement d'une application) ;
- il a été créé par le bytecode `new` avant de devenir persistant ;

Il conserve ses valeurs ~~au travers~~ des sessions avec le lecteur.

Objets transitoires

Un objet transient est un objet dont le contenu est temporaire :

- l'entête persiste et l'espace alloué au contenu également.



Un objet transient est associé à un événement dont l'occurrence déclenche la réinitialisation de ses champs :

- `CLEAR_ON_DESELECT` ;
- `CLEAR_ON_RESET`.

Les seuls types d'objets transients jusque dans les spécifications JC 2.2.1 sont les tableaux de types simples.

Quelques caractéristiques :

- il a été créé par `makeTransientBooleanArray`, `makeTransientByteArray`, `makeTransientShortArray`, `makeTransientObjectArray` (de la classe `JCSys`) ;
- il est réinitialisé lors de l'occurrence d'un événement.

42

Atomicité et Transaction

Atomicité

Util

```
static short arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff, short length)
```

```
static short arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short destOff, short length)
```

```
static short arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte bValue)
```

Transaction

JCSysm

```
beginTransaction()
```

```
commitTransaction()
```

```
abortTransaction()
```

- Rendre atomique une section de code.
- `beginTransaction()`, `commitTransaction()`, `abortTransaction()`
 - Attention : Nombre d'instructions recouvrables limité par la RAM et l'EEPROM

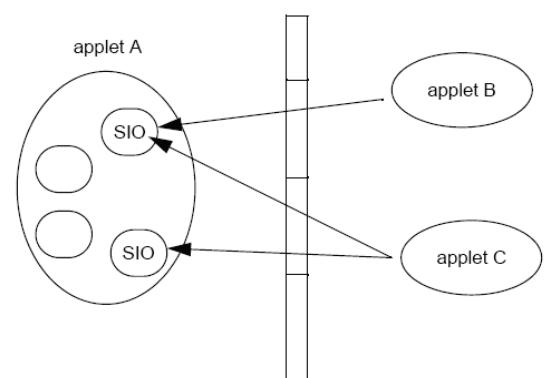
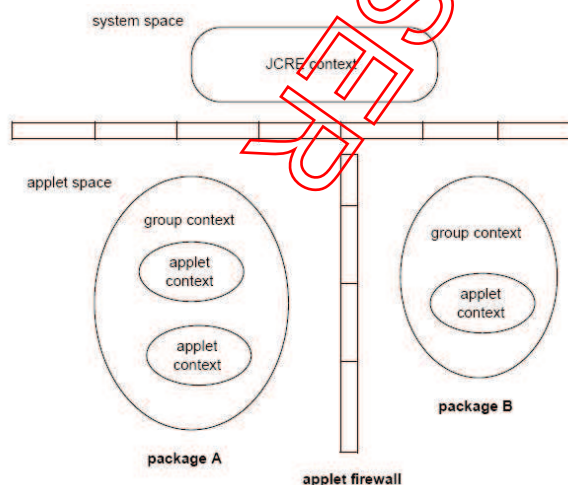
43

Partage d'objets

Le firewall est là pour protéger les données et le code des applets contre des actions malveillantes d'une applet de contexte différents.

À l'intérieur d'un même contexte les règles de visibilité Java prévalent

Les applets au sein d'un même contexte sont développées par la même personne (un contexte = un paquetage)

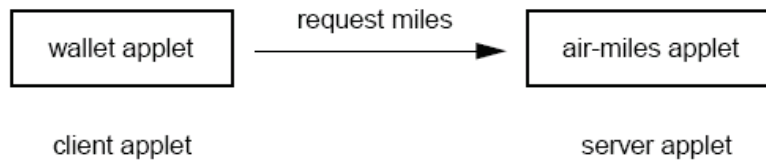


On veut pouvoir partager des « données » entres des applets développées par des personnes différentes.

=> il faut donc un mécanisme de partage d'objets !

On remarquera qu'on ne partage pas de données mais seulement des méthodes

Exemple de partage d'objet



Créer un Shareable Interface Object (SIO)

Définir l'interface partageable (Shareable Interface) qui spécifie l'interface du service

```
package com.fasttravel.airmiles;  
import javacard.framework.Shareable;
```

```
public interface AirMilesInterface extends Shareable {
```

```
    public void grantMiles (short amount);  
}
```

Créer un objet qui implémente cette interface (cela peut être l'applet serveur elle même).
Cet objet fournira le service à l'applet cliente.

Ici le SIO est l'applet !

```
package com.fasttravel.airmiles;  
import javacard.framework.*;  
  
public class AirMilesApp extends Applet  
    implements AirMilesInterface {  
  
    private short miles;  
  
    public void grantMiles(short amount){  
        miles = (short)(miles + amount);  
    }  
}
```

Exemple de partage d'objet

Demander un Shareable Interface Object

Avant de pouvoir faire une requête sur le SIO il doit le récupérer auprès du serveur.
=> Il doit donc connaître l'AID du serveur pour pouvoir lui demander le SIO

Construction de l'AID du serveur (on n'utilise pas de new)

```
public static AID JCSysTem.lookupAID(byte[] buffer, short offset, byte length)
```

Requête auprès du JCRE :

```
public static Shareable getAppletShareableInterfaceObject(AID server_aid, byte  
parameter)
```

Le second paramètre peut être utilisé comme identifiant du SIO désiré. Il peut aussi être utilisé comme paramètre pour l'authentification (hummm !)

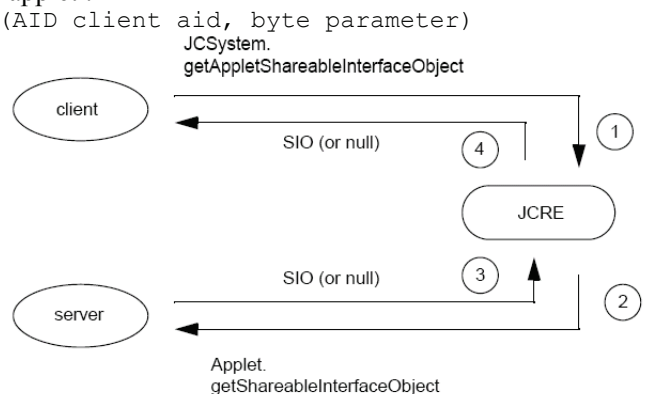
Le JCRE identifie si l'applet est présente à partir de l'AID

Si non, il retourne null si oui il interroge la méthode de l'applet :

```
public Shareable getShareableInterfaceObject(AID client aid, byte parameter)  
    JCSysTem.  
    getAppletShareableInterfaceObject
```

L'implémentation par défaut retourne null.

Si le client est autorisé et qu'il passe les bons paramètres, il reçoit le SIO désiré



Exemple de partage d'objet

```
public class AirMilesApp extends Applet
    implements AirMilesInterface {

    short miles;
    public Shareable getShareableInterfaceObject(AID client_aid,
                                                byte parameter) {

        // authenticate the client -- explained later
        // ...

        // return the shareable interface object
        return this;
    }

    public void grantMiles(short amount){
        miles = (short)(miles + amount);
    }
}
```

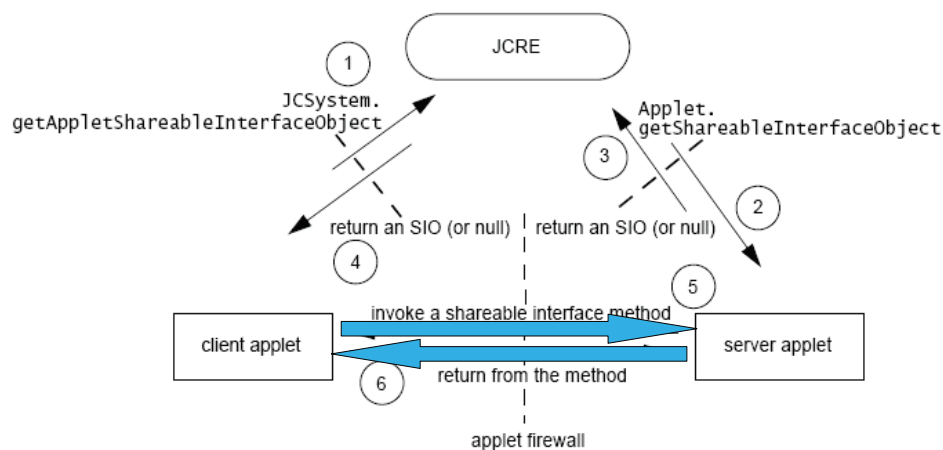
47

Exemple de partage d'objet

Utilisation du SIO

Il faut caster la référence reçue par le JCRE avant de pouvoir l'utiliser (car on reçoit une interface Shareable)

```
AirMilesInterface sio = (AirMilesInterface)
JCSysyem.getAppletShareableInterfaceObject(server_aid, parameter);
```



48

```

package com.smartbank.wallet;

import javacard.framework.*;
import com.fasttravel.airmiles.AirMilesInterface;

public class WalletApp extends Applet {

    private short balance;
    // hardcoded in the applet or assigned at
    // applet personalization
    private byte[] air_miles_aid_bytes = SERVER_AID_BYTES;

    // called by the process method on receiving
    // a DEBIT APDU command
    private void debit(short amount) {

        if ( balance < amount)
            ISOException.throwIt(SW_EXCEED_BALANCE);

        // update the balance
        balance = (short)(balance - amount);

        // ask the server to grant miles
        requestMiles(amount);
    }

    private void requestMiles(short amount) {

        // obtain the server AID object

```

49

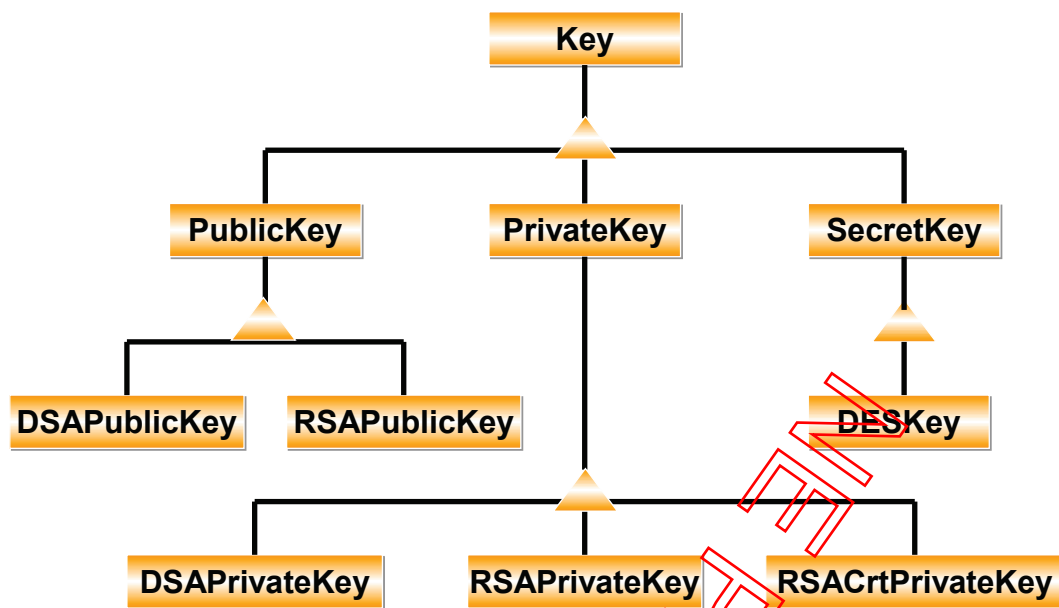
Cryptographie

Cf l'API

- **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**
- Chiffre/Déchiffre : **Cypher**

50

The key hierarchy

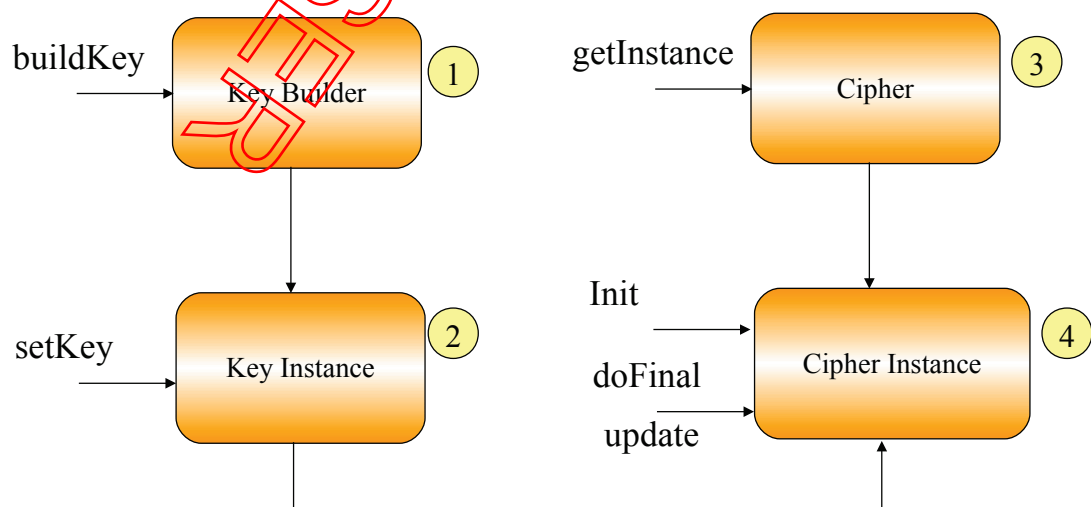


51

Principe

KeyBuilder

- Creates non initialized cryptographic keys for signature and cipher algorithms
- buildKey method create the key and returns a key interface

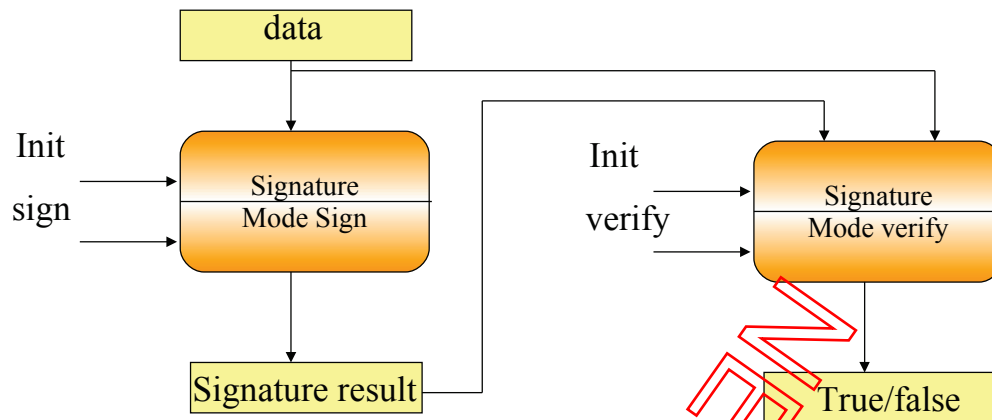


52

Principe

Signature class

- Is an abstract class for signature algorithms
- Implementations must extend this class and implement all the abstract methods,
- Contains all signature constants value for getInstance algorithms parameters specification



53

Cryptography usage

Once inside the applet:

```
DESKey k = KeyBuilder.buildKey( TYPE_DES, LENGTH_DES, false) ;  
Signature s = Signature.getInstance( ALG_DES_MAC_NOPAD, false);  
k.setKey(buffer, offset, length) ;
```

At each usage:

```
s.init(k, MODE_SIGN) ;  
s.update(in_buff, in_ofs, in_len) ;  
s.sign(in_buff, in_ofs, in_len, out_buff, out_ofs) ;  
s.verify(in_buff, in_ofs, in_len, out_buff, out_ofs);
```

54

javacardx.security.cipher

Defines all cipher algorithms methods management

Implementation of Cipher algorithms must extend this class and implement all the abstract methods

Contains all cipher constants value for getInstance algorithm parameter specification

Once inside the applet:

```
DESKey k = KeyBuilder.buildKey( TYPE_DES, LENGTH_DES, false) ;
```

```
c = cipher.getInstance( ALG_DES_MAC_NOPAD, false);
```

```
k.setKey(buffer, offset, length) ;
```

At each usage:

```
c.init(k, MODE_SIGN) ;
```

```
s.update(in_buff, in_ofs, in_len) ;
```

```
s.doFinal(in_buff, in_ofs, in_len, out_buff, out_ofs) ;
```

55

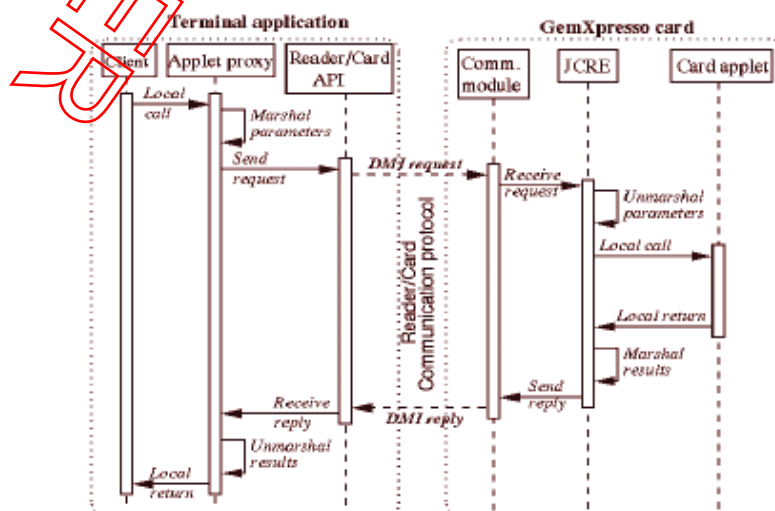
JCRMI

Existe depuis JC 2.2

On peut invoquer depuis l'extérieur des méthodes sur des objets présents dans la carte

Pas l'inverse

Utilise un générateur de souche/talon pour masquer l'encodage et le décodage des invocations de méthode en APDU



Il y a des contraintes fortes.

Mais on en parlera pas plus !

56

Construction d'applications carte

Une application carte

- Code de 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 des services
- Installation de l'applet dans les cartes
 - Initialisation des services
- Construction de l'application cliente
 - Invocation des services

Définir le but de l'application puis spécifier le protocole de communication entre l'application cliente et l'applet Java Card.

- Commandes à traiter
- Réponses possibles
- Cas d'erreurs

57

Construction d'une applet Java Card

Rôle d'une applet

- Maintenir son propre état : gestion des champs de l'applet, créer des objets et les référencer pour travailler
- Répondre à des APDUs de commande (méthode `process()`) et retourner des APDUs de réponse

Conception

- Définir les structures de données nécessaires
- Créer les objets de base à l'installation, initialiser les champs
 - Implémentation de la méthode `install()`
- Définir les APDUs traités par la méthode `process()`
 - Implémentation d'un analyseur de commandes
 - Utilisation des champs et objets de l'applet
 - Utilisation des APIs
- Définir les traitements à la sélection et à la désélection

Fabrication (avec un IDE)

- Compilation
 - Production de fichier class
- Conversion
 - Production du fichier CAP

Initialisation

- Chargement du fichier CAP
- Installation de l'applet (instanciation) via un appel à la méthode `install()` en lui donnant un AID

Utilisation

- Via l'AID défini pour cette applet

58

Construction de l'application cliente

Rôle de l'application cliente

Stimuler l'application serveur et utiliser les résultats pour faire son travail (exemple informer le commerçant que oui l'ordre a bien été accepté)

Conception

Définir les structures de données propre à l'application

Choisir son langage, un compilateur, éventuellement un framework développement (.NET, Java, etc)

Choisir un framework de communication : PC/SC, JPC/SC, OCF, etc

Implémentation en utilisant ce framework du protocole de communication avec le serveur (carte)

- Établissement de la liaison avec le serveur : envoi d'un APDU de sélection avec l'AID de l'applet
- Invocation des 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

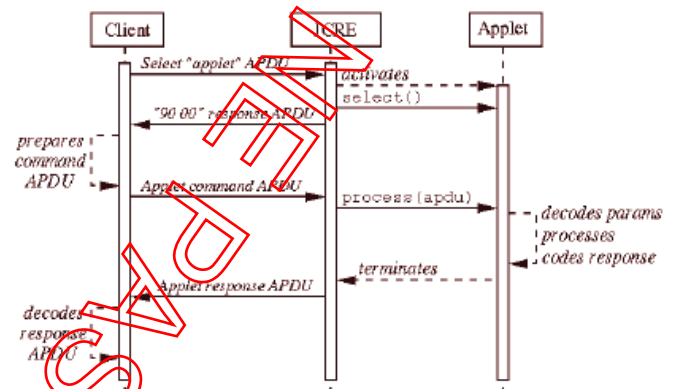
Fabrication

Avec l'IDE choisi

Déploiement

sur les machines où le service sera utilisé

Utilisation



Résumé construction d'applications

Développement de l'applet carte

Utilisation de l'API Java Card

- Essentiellement pour traiter des APDUs
- Conversion et installation dans la carte
 - Format du bytecode et procédure d'installation dans la carte non «encore» définis par Java Card
- Exécution par le JCRE
 - Sélection d'applet définie par Java Card
 - Réception et transmission d'APDUs par l'applet définies par Java Card

Développement de l'application cliente

Utilisation d'une API de communication carte permettant de transporter des APDUs

- Exemple : API OCF pour Java standard
- Sélection de l'applet par son AID
 - Commande définie par Java Card
- Échanges d'APDUs avec l'applet carte
 - Codage d'APDUs de commande et décodage d'APDUs de réponse conformes aux APDUs traités par l'applet

Problèmes pour la construction

Client et applet communiquent par APDUs

Structure de données pauvre (tableau d'octets), peu explicite (pas de typage), et difficile à manipuler

Oblige à définir le contenu et la sémantique des messages échangés : décodage et encodage d'APDUs par le client et l'applet

La spécification des échanges entre client et applet correspond à un protocole

Le format des messages APDUs échangés est utilisé comme spécification commune

Travail sur un protocole plutôt que sur des fonctionnalités

Nécessite une formation carte

L'applet et le client implémentent un protocole

Code «dupliqué» dans les programmes clients et applets

- Dans la carte : toutes les applets décodent des APDUs

Code «sensible»

- Difficile à programmer : prévoir tous les cas, pas d'outils
- Source d'erreurs

61

Exemple «Compteur» : Conception

État de l'applet :

Maintient une valeur entière positive ou nulle (pas d'objets)

Installation : création de l'objet Applet

Initialisation de la valeur du compteur

Enregistrement auprès du JCR

Opérations :

Lecture : retourne la valeur du compteur

Incrémentation/décrémentation : ajoute/soustrait un montant au compteur et retourne la nouvelle valeur du compteur

Sélection et désélection :

Aucun traitement

Rappels :

APDUs traités par l'applet :

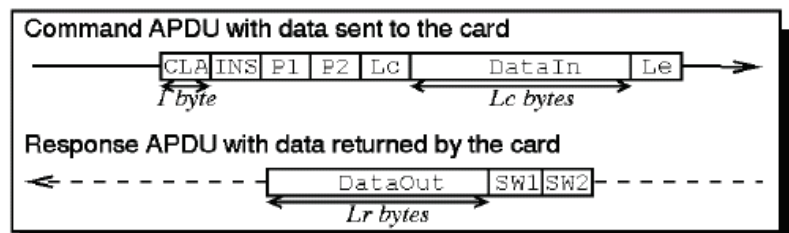
int lire()

- 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) : idem mais INS=03



62

Applet «Compteur» : Classe Applet

```
package compteur.carte.gemplus.fr ;
import javacard.framework.* ;

public class Compteur extends Applet {

    private int valeur;

    public Compteur() { valeur = 0; register(); }

    public static void install( byte[] bArray, short bOffset, byte bLength)
    {new Compteur(); }

    public void process( APDU apdu ) {
        byte[] buffer = apdu.getBuffer();
        if ( buffer[ISO7816.OFFSET_CLA] != 0xAA )
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        switch ( buffer[ISO7816.OFFSET_INS] ) {
            case 0x01: ... // Opération de lecture
            case 0x02: ... // Opération d'incrément
            case 0x03: ... // Opération de décrémentation
            default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }
}
```

63

Applet «Compteur» : décrémentation

```
case 0x03: // Opération de décrémentation
{
    // Réception des données
    byte octetsLus = apdu.setIncomingAndReceive();
    if ( octetsLus != 4 )
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
    int montant = (buffer[ISO7816.OFFSET_CDATA]<<24) |
        (buffer[ISO7816.OFFSET_CDATA+1]<<16) |
        (buffer[ISO7816.OFFSET_CDATA+2]<<8) |
        buffer[ISO7816.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;
}
```

64

GlobalPlatform

Spécifications à destinations des cartes à puce multiapplicatives pour standardiser l'administration de la carte et les procédures de communication sécurisées avec l'extérieur.

Ensemble de commandes APDU pour :

- S'authentifier auprès de l'installateur (pour avoir le droit de charger)
- Lister le contenu de la carte : applications, paquetages et bibliothèques installées
- Installer/désinstaller des applications et des bibliothèques
- Sélection une application
- ...

Ensemble d'APIs pour spécifier des protocoles sécurisés de communication et d'authentification avec l'extérieur :

- Authentification mutuelle
- Secure channel
- ...

Ensemble de mécanismes pour favoriser l'administration et la gestion d'applications sur une carte en définissant des zones appelés Security Domain possédant leur propre politique de sécurité.

- Un acteur possède un SD dans lequel il définit sa politique de sécurité (en terme de communication avec l'extérieur) et toutes les applets qu'il y installera la suivront.
- => Permet aux acteurs de se différencier et aux applications de déléguer la sécurité

65

Vers des cartes plus ouvertes ?

Cartes multiapplicatives ouvertes = tout le monde peu charger sa propre application (même malicieuse)

L'architecture est la même que celle d'une carte multiapplicative fermée avec des mécanismes sécuritaires supplémentaires :

- Soit un verifieur de bytecodes et une VM offensive ;
- Soit une VM défensive

Devraient être disponible sur le marché dans quelques années.

1980 : Cartes monoapplicatives fermées

1996 : Cartes multiapplicatives fermées

2010 : Cartes multiapplicatives ouvertes

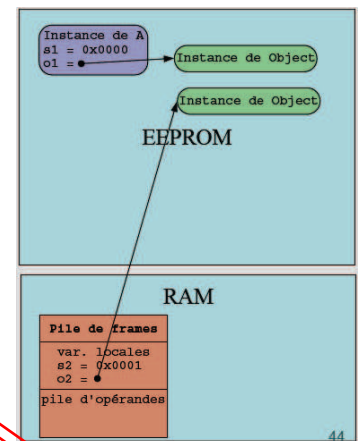
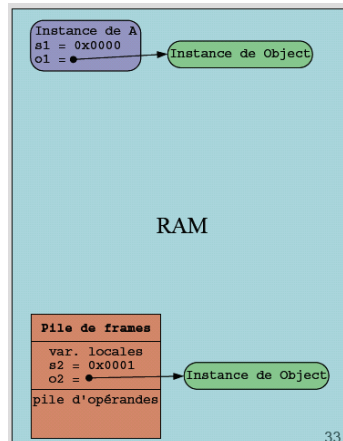
66

Inconvénients (dans la pratique) 1/2

Les problèmes de sémantique par rapport à Java :

Dus à la persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```



- Sémantiques différentes pour les variables locales et d'instance
- Objet inaccessible et non ramassé

Dus au firewall

```
package X;  
  
import Z.*;  
  
public class A extends Applet {  
  
    public void process(APDU apdu) {  
        ...  
        Foo.bar = new Bar();  
        ...  
    }  
    ...  
}
```

```
package Y;  
  
import Z.*;  
  
public class C extends Applet {  
  
    public void process(APDU apdu) {  
        ...  
        Bar bar = Foo.bar;  
        bar.test();  
        ...  
    }  
    ...  
}
```

```
package Z;  
  
public class Foo {  
  
    public static Bar bar;  
}  
  
public class Bar {  
  
    public void test() {  
        ....  
    }  
}
```

Inconvénients (dans la pratique) 2/2

Les problèmes d'intégration

Au niveau des IDE (int par défaut)

Portage d'applications

- CAP File
- Limitations dues aux possibilités physiques (taille du buffer de transaction, protocole de communication, etc.)

Les problèmes de la politique de développement

Via le Java Card Forum (JCF) et non un Java Community Process (JCP)

Outils de développement Java Card

Java Card Development Kit de Sun :

Essentiellement des outils en ligne de commande
Disponible publiquement et gratuitement

JCOP (Java Card Open Platform) de IBM :

Plugins pour l'IDE Eclipse
Disponible (presque) gratuitement

Outils de tiers :

Outils avancés pour le développement, le test et le débogage d'applications Java Cards
Basé sur les outils OEM de Sun
Disponible grâce aux vendeurs de solutions.

Exemple :

- CodeWarrior de Metrowerks

Kits de développement hardware :

Outils pour le développement et le test avec des cartes réelles
Commercialisés par les licenciés Java Card

Exemple :

- GemXpresso de Gemplus
- Sm@rtCafé® Professional Toolkit de Giesecke & Devrient
- ...

69

Références

Site de la technologie Java Card chez Sun microsystems :

<http://java.sun.com/javacard/>

Technology for Smart Cards: Architecture and Programmer's Guide

Zhiquan Chen
18 Septembre 2000

Thèse de Damien Sauveron

<http://damien.sauveron.fr/>

Cours du CNAM (Pierre Paradinas) :

<http://deptinfo.cnam.fr/~paradinas/cours/>

Cours de Didier Donsez :

<http://www-adele.imag.fr/~donsez/cours/#smartcard>



70

Disclaimer

Ce cours a été réalisé grâce :

aux transparents que j'ai pu présenter dans différents séminaires
aux transparents de cours de Samia Bouzefrane, Didier Donsez, Gilles Grimaud, Sylvain Lecomte, Pierre Paradinas
et Jean-Jacques Vandewalle.
à quelques illustrations trouvées sur Internet.

NE PAS DIFFUSER