École d'été IMAG, INRIA, LIFL - Autrans, août 1998
«Construction d'applications réparties»

Construction d'applications avec la carte à puce

Jean-Jacques Vandewalle
Gemplus Research Group

Introduction

Qu'est-ce donc que ce cours ?

Pourquoi la carte à puce ?

- **La carte à puce est un système informatique**
  - Dispose de processeur, mémoires, interface de communication
- **Une application avec la carte à puce est une application répartie**
  - «Application carte» = (serveurs +) terminaux + cartes
  - Traitements et données présents à la fois dans le terminal, (le lecteur,) et la carte
  - Nécessité de communiquer entre le terminal et la carte
- **Rôle de la carte à puce de plus en plus important**
  - Commerce (électronique), fidélité, sécurité (physique, logique), dossier portable, ...

3

Sujet du cours

- **La construction d'applications carte**
  - En fonction des spécificités de la carte à puce
    - Support matériel, normalisation
    - Interfaces matérielle (terminal - *lecteur* - carte) et logicielle (protocoles de communication)
    - Mode de programmation hérité des composants embarqués
  - En fonction des besoins des applications
    - Souplesse de développement, haut niveau de sécurité, évolutivité des applications
  - Illustrée avec la plate-forme *Java Card*™ et l'environnement de développement *GemXpresso*™

4

Auditoire du cours

- **Concepteurs et développeurs d'applications réparties**
  - Curieux de la carte à puce
  - Intéressés par la sécurisation de données/d'accès à base de cartes
  - Intéressés par la distribution de données sur support individuel
- **Concepteurs et développeurs d'applications carte**
  - Curieux du langage *Java*™
  - Concernés par l'évolution récente des environnements carte
  - Intéressés par les techniques issues des environnements répartis

5

Programme du cours

- **Première partie**
  - De Roland Moreno à la Java Card (30 mn.)
  - La technologie Java Card (40 mn.)
    - Illustré par un exemple
- **Deuxième partie**
  - Discussion (10 mn.)
  - Application des principes de programmation des applications réparties à la Java Card (40 mn.)
    - Illustré par un exemple
  - Sujets à appronfondir et perspectives de travail (10 mn.)
  - Conclusion et discussion (15mn.)

6

Objectif du cours

- **Hypothèse :** *La carte à puce est un élément des systèmes d'information*
- **Problème :** *Elle est difficile à intégrer dans les applications*
- **Lemme :** *Montrer que Java Card offre aux développeurs «non carte» la possibilité de programmer facilement des cartes à puce*
- **Théorème :** *Montrer qu'une application carte peut être construite comme une application répartie*
- **Corollaire :** *La construction d'applications avec la carte à puce devient plus simple*

7

Cheminement de la démonstration

- **Expliciter ce qu'est une carte à puce**
  - Comment ça fonctionne
  - Qu'est-ce qu'on en fait
- **Expliciter la plate-forme Java Card**
  - Plate-forme Java appliquée à la carte puce
  - Utilisation du langage Java et de l'API Java Card
- **Expliciter comment construire des applications carte**
  - En appliquant les techniques des systèmes répartis à la Java Card
- **Ouvrir des pistes pour aller plus loin**

8

De Roland Moreno à la Java Card

État de l'art de la carte à puce

Historique (1/2)

- **1974 : Dépots de brevets par *Roland Moreno***
- **1978 : *Michel Ugon* (Bull CP8) invente le M.A.M.**
- **1980 : Création du G.I.E. «carte à mémoire»**
- **1981 : Début de la normalisation AFNOR**
- **1982-1984 : Expérimentation de paiement par cartes sur 3 sites. La technologie Bull est retenue pour les «cartes bancaires» (CB)**
- **1983 : Lancement de la «télécarte» par la D.G.T.**
  Début de la normalisation ISO
- **1988 : Création de Gemplus**

10

Historique (2/2)

- **1990 : «Télécarte» entre dans le dictionnaire**
- **1992-1998 : Essor des applications avec des cartes à puce**
  - Généralisation des CB
  - Téléphonie mobile (GSM) utilise une carte SIM
  - Premières expériences de carte santé (Sésame, Vitale, All.)
  - Premiers porte-monnaie électroniques (Proton, La Poste)
- **1992-1998 : Développement de la technologie carte**
  - Augmentation des capacités matérielles
  - Systèmes d'exploitation de plus en plus ouverts
  - Système Java Card

11

Différents types de cartes

- **Carte à mémoire**
  - Mémoire simple (sans processeur) accessible en lecture sans protection, mais l'écriture peut être rendue impossible
  - Carte «porte-jetons» pour applications de pré-paiement
- **Carte à logique câblée**
  - Mémoire accessible via des circuits pré-programmés et figés pour une application particulière
  - Carte «sécuritaire» pouvant effectuer des calculs figés
- **Carte à puce**
  - Microcontrôleur encarté (processeur + mémoires)
  - Carte «programmable» pouvant effectuer tout type de traitements

12

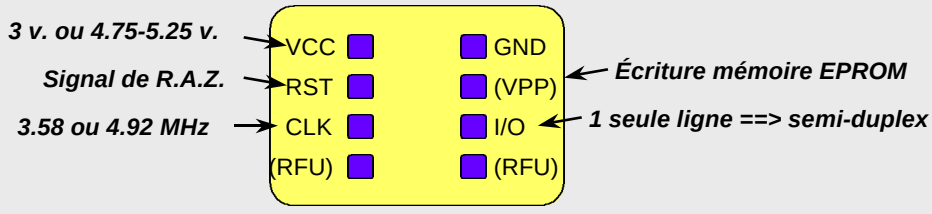
Caractéristiques des cartes à puce

- **La normalisation**
  - Caractéristiques physiques
  - Protocoles de communication
  - Commandes applicatives
- **Le microcontrôleur**
  - Technologie M.A.M. et élements de sécurité
  - Types de microprocesseurs
  - Types des mémoires
- **Les fonctionnalités**
  - Cycle de vie de la carte
  - Fonctions applicatives de la carte

13

Normalisation du matériel

- **ISO 7816**
  - Partie 1 : caractéristiques physiques
    - Format carte de crédit (85 * 54 * 0.76 mm.)
    - Définition des contraintes que doit supporter une carte
  - Partie 2 : dimensions et positions des contacts



  - Partie 3 : caractéristiques électriques

14

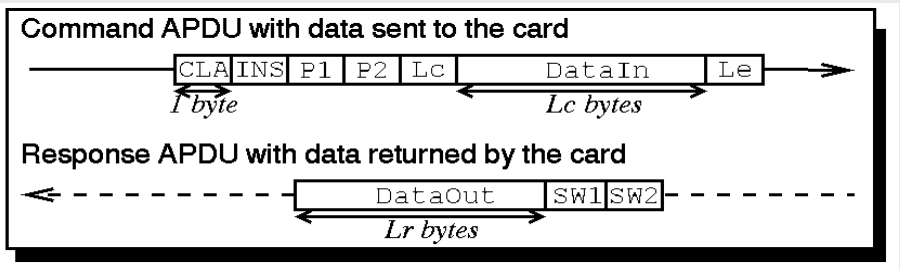
Normalisation de la communication

- **ISO 7816-3 pour protocoles lecteur-carte**
  - Transmission d'un caractère
    - 1 bit démarrage, 8 bits données, 1 bit de parité
    - Définition d'un temps de garde entre 2 caractères
  - Réponse de la carte à la R.A.Z. : séquence d'octets décrivant les caractéristiques de la carte
  - Sélection du type de protocole
  - Protocoles de communication (asynchrones et semi-duplex)
    - Mode maître-esclave : la carte répond à des commandes
    - T=0 : transmission de caractères (le plus utilisé)
    - T=1 : transmission de blocs de caractères

15

Normalisation du format des commandes

- **ISO 7816-4**
  - Définition du format des «paquets de données» échangés entre un lecteur et une carte : APDUs (Application Programming Data Units) de commande et de réponse
  - Mots d'état SW1 et SW2 standardisé (OK = `0x9000`)



16

Normalisation de commandes

- **ISO 7816-4 : Manipulation des données au travers d'une structure hiérarchique de fichiers**
- **ISO 7816-5 : Identification des applications**
- **ISO 7816-6 : Éléments de données référencées (accès direct)**
- **ISO 1816-7 : Manipulation des données au travers d'un schéma relationnel**
- **ETSI GSM 11.11 : Commandes des cartes S.I.M.**
- **E.M.V. : Commandes de paiement**
- ***etc.***

17

Microcontrôleur pour carte (1/2)

- **Contraintes physiques ISO 7816-1 ==>**
  - Surface <= 25 mm^2
  - Épaisseur <= 0,3 mm.
- **Fondé sur la technologie M.A.M.**
  - Microprocesseur + bus + mémoires réunis sur un même substrat de silicium (technologie de 0,7 à 0,35 microns)
  - Peut être «re-programmé» par l'écriture de programmes en mémoire non-volatile
- **Éléments de sécurité**
  - Composant inacessible
  - Détecteurs de conditions anormales

18

Microcontrôleur pour carte (2/2)

- **Types de microprocesseur**
  - 8-16-32 bits (+ coprocesseur cryptographique)
  - SGS-Thomson, Siemens, Motorola, Hitachi, NEC, *etc.*
- **Types des mémoires**
  - ROM (Read-Only Memory) : mémoire non-volatile à lecture seule (jusqu'à 64 Ko)
  - RAM (Random Access Memory) : mémoire volatile à accès rapide (jusqu'à 2 Ko)
  - EEPROM (Electrical Erasable Programmable ROM) : mémoire non-volatile réinscriptible (jusqu'à 32 Ko)
    - l'EPROM devenue obsolète
    - Nouveaux types : Flash EEPROM

19

Cycle de vie de la carte (1/2)

- **Fabrication**
  - Inscription d'un programme en mémoire ROM définissant les fonctionnalités de base de la carte : «*masque*» figé sachant traiter un nombre limité de commandes pré-définies
- **Initialisation**
  - Inscription en EEPROM des données communes à l'application
  - Possibilités pour certaines cartes d'ajouter des «*filtres*»
- **Personnalisation**
  - Inscription en EEPROM des données relatives à chaque porteur

20

Cycle de vie de la carte (2/2)

- **Utilisation**
  - Envoi d'APDUs de commande à la carte
  - Traitement de ces commandes par le masque de la carte
    - Si commande reconnue
      - Traitement en interne de la commande ==> lecture/écriture de données en EEPROM
      - Renvoi d'un APDU de réponse
    - Si commande inconnue
      - Renvoi d'un code d'erreur
- **Mort**
  - Par invalidation logique, saturation de la mémoire, bris, perte, vol, *etc.*

21

Fonctionnalités des masques carte

- **Caractéristique commune : définition d'un jeu *figé* de commandes que la carte sait traiter**
- **Types de masques**
  - Masque spécifique à une application (B0', Proton, SIM) : Lecture/écriture de données figées avec règles de sécurité figées
  - Masque spécifique à un contexte applicatif (GPK, CryptoFlex) : Lecture/écriture de données formatées, fonctions de sécurité adaptées au contexte
  - Masque générique (MCOS, MFC, PocketBase) : Lecture/écriture de données, algorithmes cryptographiques

22

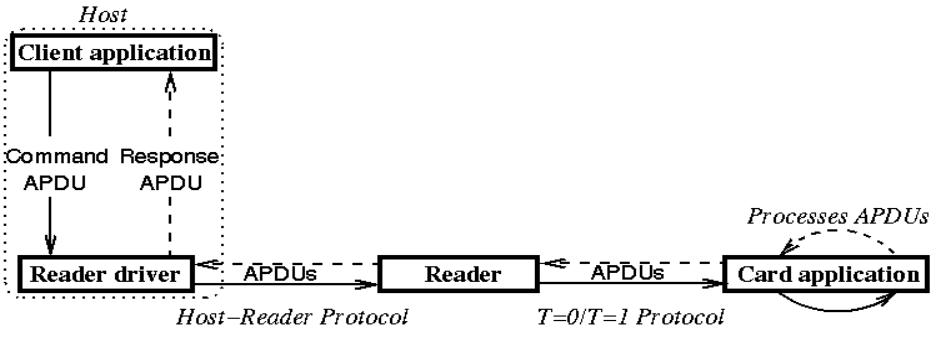
Développement d'applications carte

- **La carte**
  - Accessible via un jeu de commandes figé gravé lors de la fabrication du composant
  - Maintient des données pouvant évoluer
- **Le lecteur et la communication avec la carte**
  - Nécessite un lecteur de cartes (et donc un «driver» pour le piloter)
  - Messages échangés définis par des APDUS de commande
- **Le terminal et l'application cliente**
  - Communique avec la carte via le driver du lecteur
  - Envoie des requêtes APDUs conformes au jeu de commandes de la carte

23

Architecture d'application carte

- **Schéma général**
  - Le terminal contrôle, la carte est passive
  - Dialogue terminal-carte de type requête/réponse
  - Format de messages standard : APDUs



24

Éléments importants

- **Le code applicatif de la carte est gravé en ROM au moment de la fabrication**
  - La carte est un serveur figé en terme de fonctionnalités
  - Le développement du code nécessite des compétences carte
    - Le code est généralement développé par les fabricants
  - Pas de possibilité d'évolution et d'adaptation du code
    - Pas de chargement dynamique de nouveaux programmes en EEPROM
- **Pas de protocole standard de communication entre le système hôte et le lecteur**
  - Pas d'API standard d'accès aux drivers des lecteurs
  - Drivers offrent uniquement une API de transport des APDUs

25

Difficultés à la construction d'applications

- **Si l'application requiert de nouvelles fonctions carte**
  - Nécessite le développement d'un nouveau masque (coûteux)
- **Si les fonctionnalités de la carte doivent pouvoir évoluer**
  - Nécessite un masque qui accepte d'exécuter des programmes en EEPROM (rare et pas standardisé)
- **Intégration dans les systèmes d'informations**
  - Pas d'interface standard de communication avec les différents lecteurs (travaux en cours)
  - Communication *via* APDUs (très bas niveau)

26

Résumé de l'état de l'art

- **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**

27

Vers des cartes plus ouvertes (1/2)

- **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

28

Vers des cartes plus ouvertes (2/2)

- **Éléments de solutions**
  - Java Card
    - Utiliser le langage Java pour programmer les cartes
      - Bénéficie d'un langage orienté objet
    - Utiliser la plate-forme Java pour charger et exécuter des applications dynamiquement
      - Bénéficie d'une architecture sécuritaire
  - GemXpresso
    - Utiliser les concepts de la programmation d'applications réparties pour développer des applications carte avec Java Card
      - Bénéficie des concepts du client-serveur orienté objet

29

La technologie Java Card

Comprendre Java Card

Comprendre Java Card : sommaire

- **Les concepts**
  - Qu'est-ce que Java Card ?
  - Statut du standard Java Card
  - Sous-ensemble du langage Java
  - La machine virtuelle
  - Le modèle mémoire
- **La pratique**
  - Concepts de programmation
  - Les APIs de programmation
  - Développement d'une applet Java Card
  - Construire une application avec la Java Card

31

Qu'est-ce que Java Card ?

- **Une Java Card est une carte à puce qui peut exécuter des programmes Java (applets carte)**
  - Utilisation du langage Java pour programmer des applications carte
    - Basée sur un «standard», programmation orientée-objet
  - Java Card définit un sous-ensemble de Java (1.0.2, *sic!*) dédié pour la carte à puce :
    - Sous-ensemble du langage de programmation Java
    - Sous-ensemble du paquetage `java.lang`
    - Découpage de la machine virtuelle Java
    - Modèle mémoire adapté à la carte
    - APIs spécifiques à la carte

32

Statut du standard Java Card (1/2)

- **10/96 : Spécification Java Card 1.0**
  - Poussée par Schlumberger (produit CyberFlex)
  - Très limitée (4 pages) et proche de la carte
- **02/97 : Création du «Java Card Forum»**
  - Regroupe les fabricants de cartes et Sun pour établir les spécifications + des utilisateurs pour promouvoir Java Card
- **10/97 : Spécification Java Card 2.0**
  - Sous-ensemble du langage et de la machine virtuelle Java
  - Concepts de programmation et APIs
- **10/98(?) : Spécification Java Card 2.1**
  - Modifications API, normalisation Bytecode, *etc?*

33

Statut du standard Java Card (2/2)

- **Licenciés**
  Bull, Dallas Semiconductor, De La Rue, Gemplus, Giesecke & Devrient, Inside Technologies, Keycorp, Lucent, Motorola, NatWest, Oberthur, Schlumberger, Toshiba, TL Technologies, ...
- **Produits**
  - Schlumberger : *CyberFlex Core 2.0* (JC 1.0)
  - Gemplus : *GemXpresso RAD 1.0* (JC 2.0)
  - Bull SA : *Odyssey* (JC 2.0)
  - Giesecke & Devrient : *C@ppucino* (JC 2.0)
  - Dallas Semiconductor : *iButton* (bague Java!)

34

Sous-ensemble du langage Java

- **Pourquoi un sous-ensemble ?**
  - Limitations carte : puissance de calcul, tailles des mémoires
    - JVM doit pouvoir s'exécuter sur composant 8 bits avec 512 octets RAM, 16 Ko EEPROM, et 24 Ko ROM
  - Contexte applicatif carte : petites applications de type serveur
- **Définition d'une application Java Card**
  - Applets Java Card (`javacard.framework.Applet`)
  - À la différence du JDK, pas de notions :
    - d'applets : `java.applet.Applet`
    - d'applications : `public static void main( String[] args )`

35

Java Card par rapport à Java (1/4)

- **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()`

36

Java Card par rapport à Java (2/4)

- **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`
  - Pas de classes `Boolean`, `Byte`, `Class`, *etc.*
- **Tableaux à une dimension avec élements :**
  - Types de base
  - Objets (les tableaux sont eux-mêmes des objets)

37

Java Card par rapport à Java (3/4)

- **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

38

Java Card par rapport à Java (4/4)

- **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()`

39

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

40

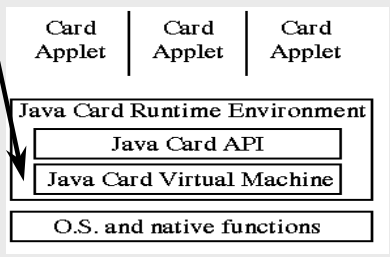
Résumé Java Card p/r à Java (2/2)

- **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
```

41

Machine virtuelle JC : partie carte

- **Définition de la machine virtuelle Java**
  - Vérifieur de Bytecode
  - Chargeur dynamique de classes
  - Interpréteur de Bytecode
- **Dans la carte, la JCVM n'implémente que :**
  - Interpréteur de byte-code
  - Gestion des classes et objets
  - Isolation des applets
    - Par paquetage



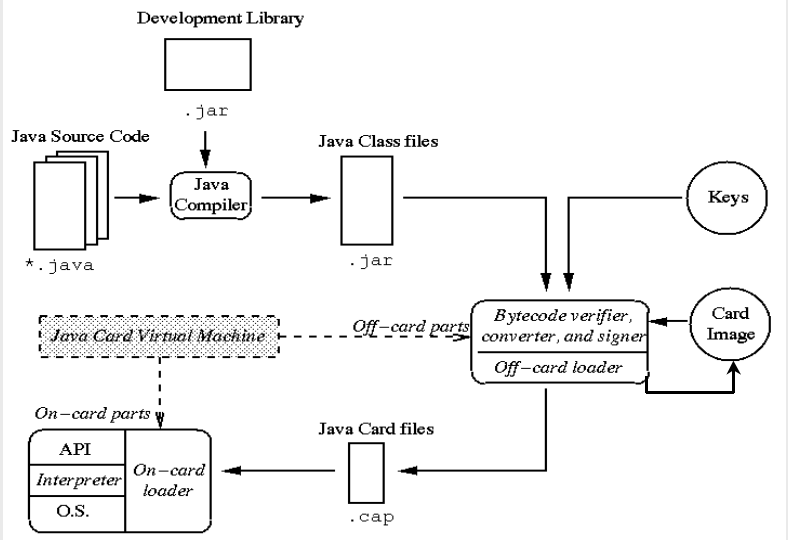
42

Machine virtuelle JC : découpage

- **Pourquoi la JCVM ne contient pas le vérifieur ?**
  - Trop lourd pour être stocké et/ou exécuté dans la carte
- **Pourquoi la JCVM ne contient pas le chargement dynamique de classes ?**
  - Pas d'accès à l'endroit où sont stockés les fichiers de classes depuis la carte
  - Pas de vérifieur dans la carte permettant de vérifier dynamiquement la validité d'une classe chargée
- **Architecture JCVM p/r à la JVM**
  - «Identique» mais découpée en une partie dans la carte et une partie hors carte («Java Card Converter»)

43

Machine virtuelle JC : architecture



44

Machine Virtuelle JC : partie hors-carte

- **Vérifieur de Bytecode**
  - Utilise le vérifieur de Bytecode Java «classique»
  - Contrôle le sous-ensemble Java Card (langage + API)
- **Convertisseur («early binding»)**
  - Préparation : initialise les structures de données de la JCVM
  - Optimisation : ajuste l'espace mémoire, remplace certains `InvokeVirtual` par des `InvokeStatic`, *etc.*
  - Édition de liens : résout les références symboliques à des classes déjà présentes dans la carte (*via* «image» de la carte)
- **Signeur**
  - Valide le passage par le vérifieur et le convertisseur par une signature vérifiée par la carte au moment du chargement

45

Modèle mémoire Java Card

- **La JCVM est toujours active même quand la carte est déconnectée**
  - Elle est automatiquement remise en route à la (re-)connexion
- **Les objets sont stockés de manière persistente**
  - Stockage en EEPROM sans ramasse-miettes
  - Attention ! aux clauses `throw new Exception();`
    - Créer l'objet une fois (patron "singleton"), utiliser des méthodes statiques `Exception.throwIt(...);`
    - Détruire les objets locaux non assignés en fin d'exécution
- **L'objet applet**
  - Créé une seule fois (avec un identifiant AID unique)
  - Toujours une applet active par défaut

46

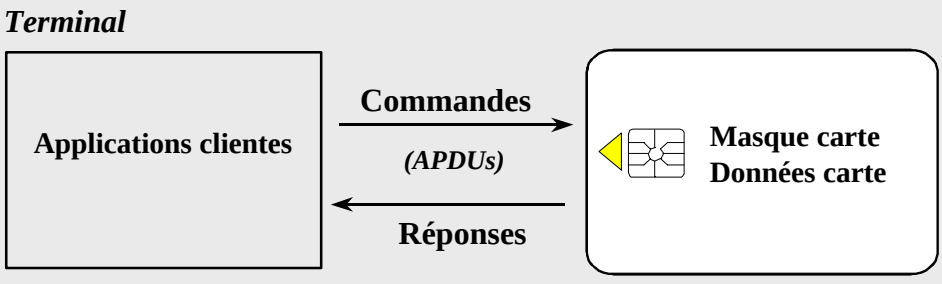
Comprendre Java Card : sommaire

- **Les concepts**
  - Qu'est-ce que Java Card ?
  - Statut du standard Java Card
  - Sous-ensemble du langage Java
  - La machine virtuelle
  - Le modèle mémoire
- **La pratique**
  - Concepts de programmation
  - Les APIs de programmation
  - Développement d'une applet Java Card (+ exemples)
  - Construire une application avec la Java Card

47

Concepts de programmation (1/2)

- **Approche traditionnelle**



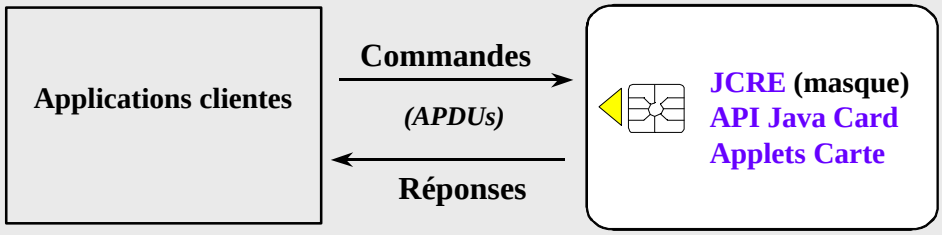
  - Masque carte traite les APDUs de commande
  - Masque carte lit/écrit/met à jour les données carte
  - Application cliente envoie des APDUs de commande

48

Concepts de programmation (2/2)

Approche Java Card

Terminal



- Environnement d'exécution Java Card (JCRE) = support de l'exécution d'applets carte grâce à l'API Java Card
- Applets carte traitent des APDUs de commande
- Application cliente envoie des APDUs de commande

49

Les APIs de programmation Java Card

Paquetage `javacard.framework`

- Principale API pour programmer une applet carte
- Définit les classes :
  - `AID`, `APDU`, `Applet`, `ISO`, `PIN`, `System`, `Util`
  - Plus des classes d'exceptions

Extensions

- `javacardx.framework` : gestion de fichiers conforme à la norme ISO 7816-4
- `javacardx.crypto` : gestion de clés publiques et privées, générateur de nombres aléatoires, fonction de hashage
- `javacardx.cryptoEnc` : algorithme de chiffrement DES

50

Les APIs utilitaires de `javacard.framework` (1/2)

- **`public final class System`**
  - Méthodes statiques (natives) pour interagir avec le JCRE
    - Gestion des transactions (1 seul niveau)
    - Gestion du partage d'objets entre applets (en cours de redéfinition)
- **`public class Util`**
  - Méthodes statiques (natives) utiles pour performance carte
    - Copie, comparaison de tableaux de bytes
    - Création de **`short`** à partir de **`byte`**
- **`public final class AID`**
  - Encapsule des identifiants d'applications carte conformes à la norme ISO 7816-5

51

Les API utilitaires de `javacard.framework` (2/2)

- **`public class ISO`**
  - Champs statiques de constantes conformes aux normes ISO 7816-3 et 4
  - **`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
    - **`ProxyPIN`** : représentant d'un code secret

52

Applet carte

- **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()`

53

Méthodes publiques d'une applet

- `public void install( APDU apdu )`
  - 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 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

54

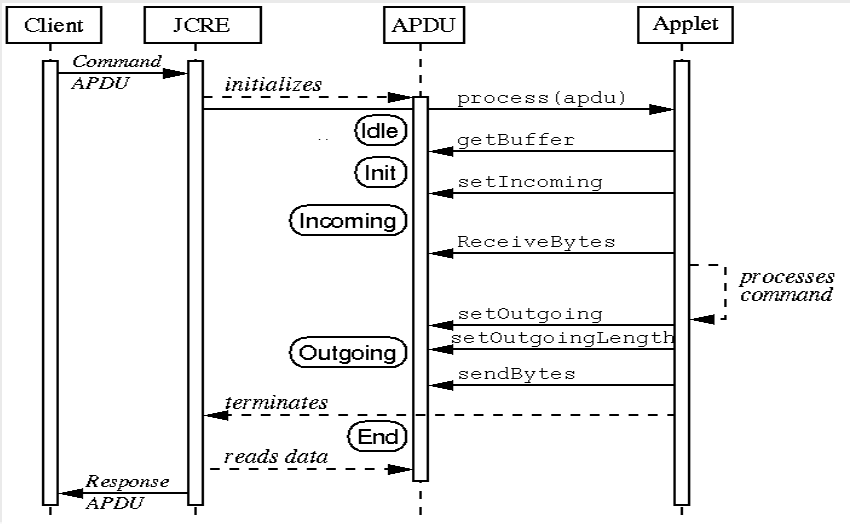
Gestion des APDUs par une applet (1/2)

- **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

55

Gestion des APDUs par une applet (2/2)

- **Lecture/écriture dans le buffer d'APDU**



56

Résumé des caractéristiques du JCRE

- **Classes de l'API Java Card**
- **Machine virtuelle Java Card**
  - Interpréteur de Bytecode
  - Gestion des classes et des objets
- **Gestion des applets**
  - Installation, enregistrement et initialisation
  - Sélection et désélection
  - Transmission des APDUs
  - Récupération des exceptions
- **Méthodes natives**
  - Entrées/sorties, transactions, cryptographie

57

Conception 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**
  - 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
  - Définir les traitements à la sélection et à la désélection

58

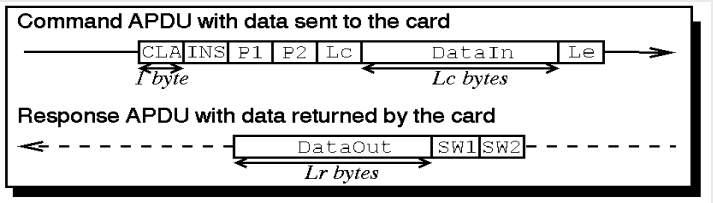
Exemple «Compteur» : Conception

- **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 JCRE
- **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

59

Exemple «Compteur» : APDUs

- **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`

60

Applet «Compteur» : Classe `Applet`

```
package javacard.carte.autrans.fr ;

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);
    }
  }
}
```

61

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;
}
```

62

Exemple 2 (1/4)

```
package banque.com ;

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( byte[] valeurPin ) {
    pin = new OwnerPIN(MaxEssai_PIN, MaxLg_PIN);
    balance = 0;
    register() ;
  }
```

63

Exemple 2 (2/4)

```
  public static void install( APDU apdu ) {
    new Pme();
    buffer = apdu.getBuffer();
    byte octetsLus = (byte)apdu.setIncomingAndReceive();
    if ( octetsLus <= MaxLg_PIN )
      pin.updateAndUnblock(buffer, ISO.OFFSET_CDATA ,
                           octetsLus);
  }

  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);
    }
  }
```

64

Exemple 2 (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]);
  }
```

65

Exemple 2 (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);
  }

}
```

66

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

67

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()`
  - Le JCRE fournit l'environnement d'exécution et la couche de communication
- **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()` des applets (non standardisé)

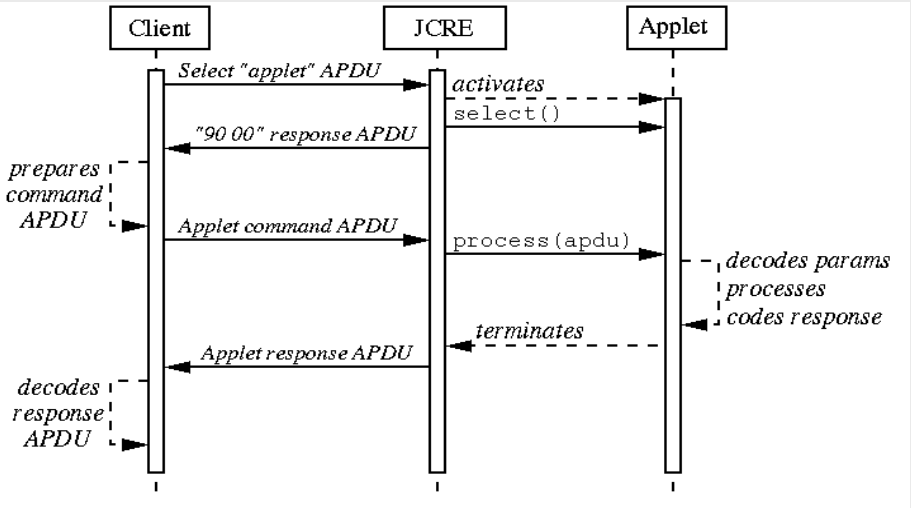
68

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

69

Construction de l'application cliente (2/2)



70

API de communication avec la carte

- **Open Card Framework (OCF) : API Java en cours**
  - `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)
  - API Cliente du Gemplus SDK
    - Tout lecteur Gemplus
    - Java VM (JDK 1.1 ou MS 2.0) sous Windows et Solaris

71

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

72

Client «Compteur» : liaison carte

```
package javacard.client.autrans.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());
}
```

73

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» */ }
}
```

74

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

75

Discussion : les problèmes des développeurs

Les limites de Java Card

Deux types de problèmes

- **Portabilité des applications**
  - Une applet Java Card peut-elle s'exécuter sur toutes les Java Cards du marché ?
  - Une application cliente Java Card peut-elle s'exécuter avec différents matériels ?
  - 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 !

77

Résumé construction d'applications (1/2)

- **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
    - ***Portabilité des applets s'arrête avec le «Java Card Converter», mais bientôt Java Card 2.1***
  - Exécution par le JCRE
    - Sélection d'applet défini par Java Card
    - Réception et transmission d'APDUs par l'applet définies par Java Card

78

Résumé construction d'applications (2/2)

- **Développement de l'application cliente**
  - Utilisation d'une API de communication carte permettant de transporter des APDUs
    - ***Bientôt une API Java standard (OCF)***
  - 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***

79

Deux types de problèmes (suite et fin)

- **Portabilité des applications**
  - Conversion et installation des applets seront définies par Java Card 2.1, mais nombreuses options encore possibles
  - OCF définit une API Java d'accès aux cartes, mais fournisseurs devront la supporter
- **Procédé de construction des applications**
  - 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

80

Problèmes pour la construction

- **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

81

Les techniques des applications réparties appliquées à la carte

Construire les applications carte mieux et plus facilement

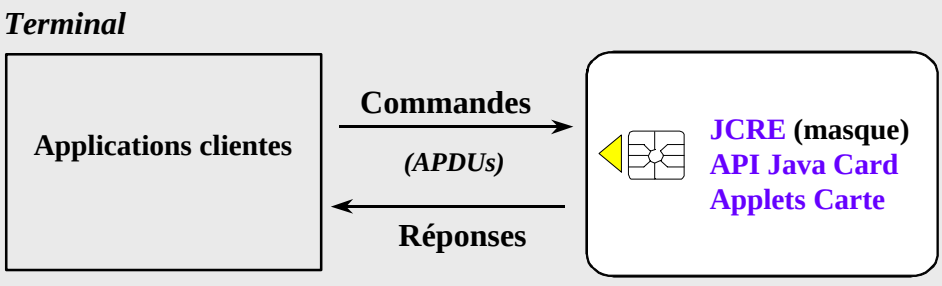
Applications réparties carte : sommaire

- **Les concepts**
  - Approche client-serveur, objets répartis et carte
    - Les RPC
  - Invocation de méthodes dans la carte
    - Protocole DMI
  - Mise en œuvre dans GemXpresso
- **La pratique**
  - Concepts de programmation
  - Développement d'une applet (+ exemple)
  - Construire une application avec GemXpresso

83

Rappel

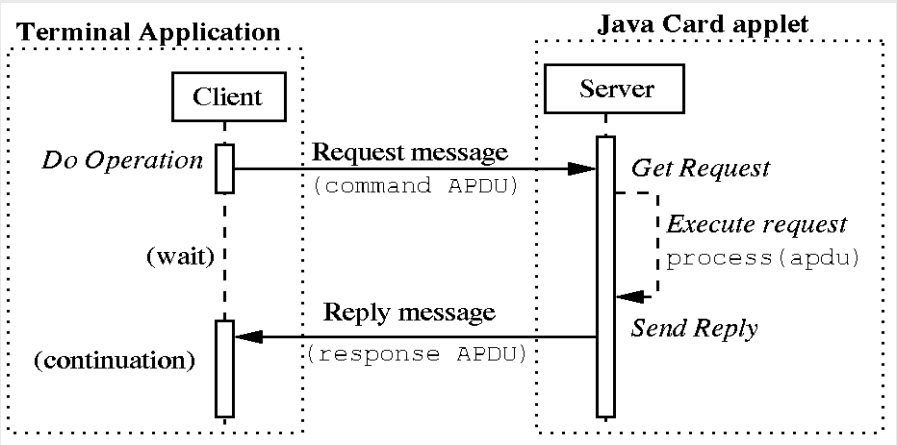
- **Approche Java Card**



  - Environnement d'exécution Java Card (JCRE) = support de l'exécution d'applets carte grâce à l'API Java Card
  - Applets carte traitent des APDUs de commande
  - Application cliente envoie des APDUs de commande

84

Modèle client-serveur de Java Card



- **Codage/décodage des messages à un niveau proche du protocole de communication (APDUs)...**

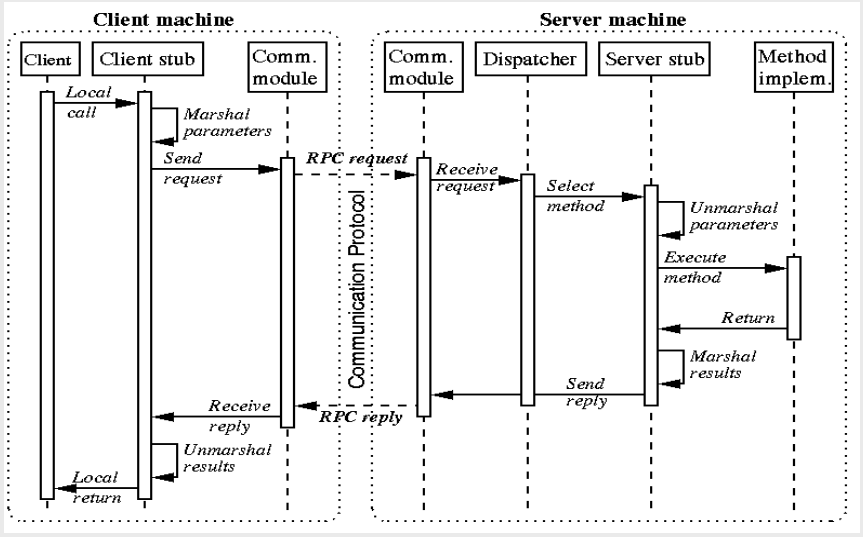
85

Client-serveur et Remote Procedure Call

- **Définition de RPC**
  - Introduit dans les applications réparties pour porter l'échange de messages entre un client et un serveur au niveau de l'appel de procédures
    - Message de requête = appel de procédure par le client
    - Message de réponse = valeur de retour de la procédure exécutée par le serveur
  - Protocole d'appel de procédures sur machine distante indépendant de la couche de communication
    - Définit comment «nommer» une procédure distante
    - Définit comment sont codés les paramètres et les valeurs de retour dans un format «neutre»

86

Architecture RPC



87

Construction d'applications avec RPC

- **Définir les interactions entre clients et serveurs**
  - Liste des procédures serveurs «appelables» depuis un client *via* RPC = interfaces (contractuelles) des services
- **Produire les souches clientes et serveurs**
  - Emballage et déballage des messages RPC
  - Autant de souches différentes que de machines cibles
- **Développer les applications clientes et serveurs**
  - Utilisation des souches clientes pour utiliser les services comme des services locaux
  - Utilisation des souches serveurs pour implémenter les appels aux services comme des appels locaux
  - Utilisation des bibliothèques RPC pour liaison client-serveur et transmission/réception des messages RPC

88

RPC et systèmes à objets répartis

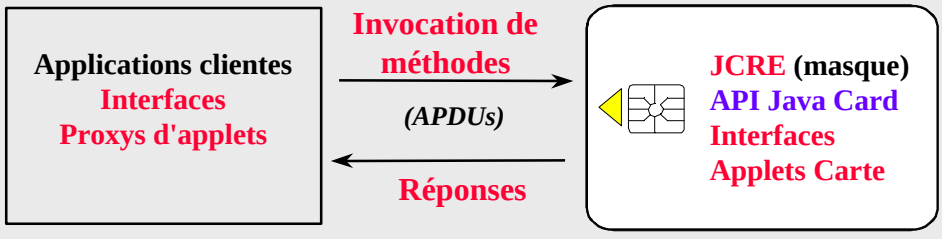
- **Principes**
  - Utilisation de la notion d'interfaces objet pour décrire les objets distants
  - Pré-compilateur pour générer les souches (proxys et squelettes)
  - Protocole d'invocation de méthodes
- **Mises en œuvre**
  - CORBA
    - Interfaces IDL, projections dans langages cibles, protocoles ORB et inter-ORB
  - Java
    - Interfaces Java, `rmic`, protocole RMI dans JVM

89

RPC et carte : approche

- **Construire les applications avec la carte comme des applications réparties à objets**
  - Applet Java Card = objet serveur distant
  - Client invoque des méthodes de l'applet

Terminal



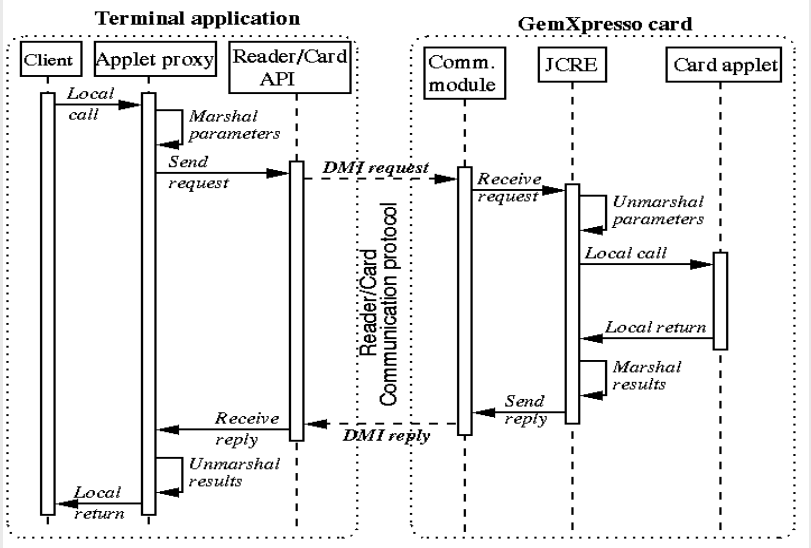
90

RPC et carte : mécanismes

- **Invocation de méthodes de l'applet carte prises en charge par un protocole «à la» RPC : *Direct Method Invocation (DMI)***
  - APDUs de commande et réponse pour la sélection d'applet
  - APDUs de commande et réponse pour l'invocation de méthodes
- **Génération des souches cliente et serveur pour l'emballage et le déballage des messages DMI**
  - Proxy pour le client (*Card Applet Bridge*)
  - Description d'interface pour le JCRE
- **Description des interfaces d'applets**
  - Interface Java avec restrictions

91

RPC et carte : architecture



92

Protocole DMI : généralités

- **Protocole applicatif d'échange de messages pour :**
  - La sélection de services carte (`select()`)
  - L'invocation de méthodes du service sélectionné (`invoke()`)
- **Les messages-réponses DMI sont spécifiés par des APDUs de commande et de réponse**
  - Compatibilité avec format de messages ISO 7816-4
    - Peuvent être transportés par les APIs lecteur
  - Nécessite l'interprétation de ces commandes par le masque de la carte (JCRE)
    - Commandes non standard mais pourraient l'être...
    - Pas interdit par Java Card
- **Non limité à Java Card**

93

Protocole DMI : `select` (1/2)

- **Sélection d'une applet Java Card par son AID**
  - Compatibilité ISO 7816-5 et Java Card
  - AID sur 16 octets
- **Message**
  - `CLA INS P1 P2 Lc DataIn Le`
    `A8 A4 04 XX 10 «applet AID» 00`
- **Interprétation**
  - *Si* l'applet existe (AID référencé dans la carte) et accepte d'être sélectionnée (méthode `Applet.select()` renvoie `true`), *alors* les prochains APDUs de commande seront destinés à cette applet (l'applet courante est désélectionnée)
  - *Sinon*, l'applet courante reste active

94

Protocole DMI : `select` (2/2)

- **Réponse**
  - Pas de données renvoyées
  - `90 00` si la sélection a réussi
  - Sinon, code d'erreurs
    - `6A 82` : applet non trouvé
    - `6F A0` : exception/erreur renvoyée par la JCVM

95

Protocole DMI : `invoke` (1/2)

- **Invocation d'une méthode d'applet Java Card avec la signature de la méthode et ses paramètres**
  - Identifiant de méthode (numéro de `01` à `7F`)
  - Types et valeurs des arguments
    - Chaque type a un identifiant (signature)
    - Les paramètres sont passés par valeurs
  - Type de la valeur de retour
- **Message**
  -
```
CLA INS P1    P2      Lc DataIn          Le
A8  36  #meth TypeRet ?? Valeur params   00
```

96

Protocole DMI : `invoke` (2/2)

- **Interprétation**
  - *Si* la méthode existe dans l'applet courante, *alors* elle est exécutée avec les paramètres fournis *et* le résultat de l'exécution (valeur de retour ou exception) est retournée
  - *Sinon*, un code d'erreur est retournée
- **Réponse**
  - `DataOut` `SW1 SW2`
    `Valeur Retour ou Exception` `90 00`
  - Sinon, code d'erreurs
    - `6F 00` : méthode inconnue
    - `61 03` : erreur renvoyée par la JCVM en cours d'exécution

97

Mise en œuvre dans GemXpresso (1/2)

- **GemXpresso**
  - Implémentation de Java Card 2.0 (32 bits, 8/512/32)
  - Ajout du mécanisme DMI
- **Applet Java Card doit aussi implémenter une interface**
  - ```
    class MonApplet extends Applet
      implements MonInterface
    ```
  - Types transportés par DMI (acceptés dans les interfaces Java d'applets)
    - (`void`), `byte`, `boolean`, `short`, et `int`
    - Tableaux à une dimension des types de base
    - Exceptions définies par les spécifications Java Card

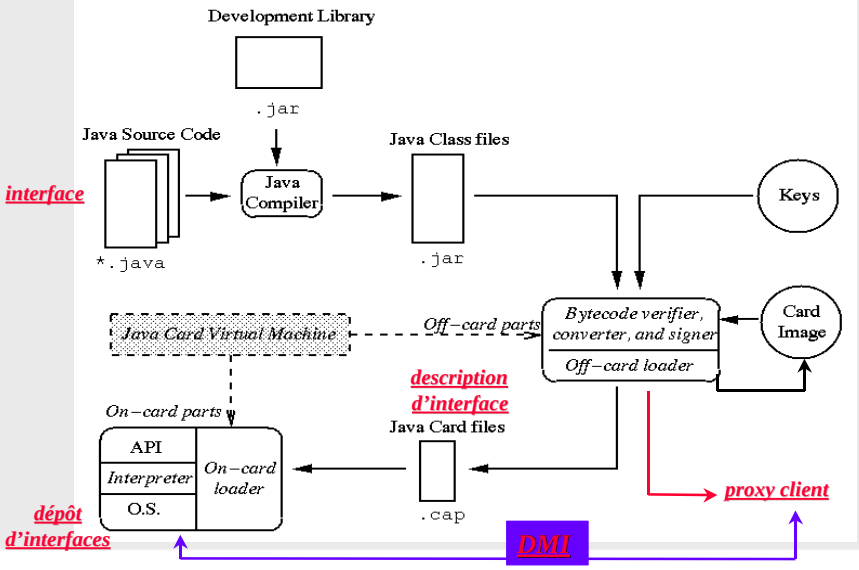
98
    ```

Mise en œuvre dans GemXpresso (2/2)

- **Génération de proxies clients**
  - Dans les langages Java/C++/C/VisualBasic
  - Utilise l'API cliente du Gemplus SDK (Java) ou l'API 4 de Gemplus (C++/C/VB)
  - Sera étendue aux APIs OCF (Java) et PC/SC (C++/C/VB)
- **Souches serveurs**
  - Gérées directement par le JCRE de la carte GemXpresso
  - Fourniture à l'installation de l'applet de la carte de la description de l'interface au JCRE
    - «Dépôt d'interfaces» dans la carte
    - Vérification et décodage des messages `invoke` de DMI

99

Architecture GemXpresso



100

Autres caractéristiques GemXpresso (1/3)

- **En plus par rapport à Java Card 2.0**
  - Type `int` et mot-clé `transient`
  - Destruction des objets locaux non assignés
  - Politique de sécurité : paquetage public ou privé
  - Exceptions Java Card 2.1
- **En moins par rapport à Java Card 2.0**
  - Pas de gestion des transactions
  - Partage d'objets non implémenté car sera complétement revu dans Java Card 2.1
  - Chargement sécurisé par *un* mot de passe
  - Algorithmes cryptographiques non implémentés pour des raisons d'export

101

Autres caractéristiques GemXpresso (2/3)

- **APIs** `com.gemplus.gemxpresso.library`
  - Interface `IApplet` avec méthodes de `javacard.framework.Applet`

```
interface MonInterface extends IApplet
```

  - Notification : `Observer` et `Observable`
  - Streams : `CardByteArrayInputStream` et `CardByteArrayOutputStream`
  - Arithmetique : `Byte`, `Short`, `Int` avec uniquement `MIN_VALUE`, `MAX_VALUE`
  - Mathématique : `Math` avec uniquement `abs`, `min` et `max`
  - Manipulation de champs de bits : `BitSet`
  - Utilitaire : `Util32` pour construire des `int` depuis des `byte`

102

Autres caractéristiques GemXpresso (3/3)

- **Spécifités DMI**
  - Invocation du `clinit` des classes au chargement d'un paquetage
    - Numéro de méthode `00` pour le `invoke` de DMI
    - Pas de paramètres
  - Invocation du constructeur de l'applet au chargement
    ==> *Une applet peut avoir plusieurs constructeurs*
    - Numéro de méthode `FE` à `80` pour le `invoke` de DMI
    - Paramètres du constructeur
    - Enregistre automatiquement l'applet si pas d'exceptions/d'erreurs

103

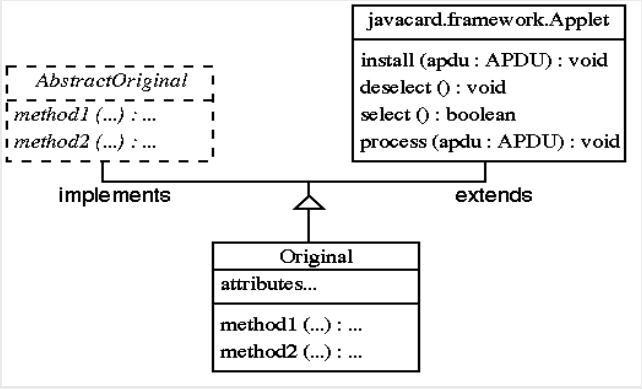
Applications réparties carte : sommaire

- Les concepts
  - Approche client-serveur, objets répartis et carte
    - Les RPC
  - Invocation de méthodes dans la carte
    - Protocole DMI
  - Mise en œuvre dans GemXpresso
- **La pratique**
  - Concepts de programmation
  - Développement d'une applet (+ exemple)
  - Construire une application avec GemXpresso

104

Conception d'une applet GemXpresso

- **Compatible avec Java Card**
  - Mode Java Card seul possible
  - Mode GemXpresso = sur-ensemble Java Card



105

Applet carte GemXpresso et JCRE (1/2)

- **Installation**
  - Java Card
    - Appel à `install(apdu)`
  - GemXpresso
    - Appel à un constructeur de l'applet *ou*
    - Appel à `install(apdu)`
- **Sélection**
  - Mécanisme identique pour Java Card et GemXpresso
    - APDU de commande `select` de Java Card et DMI identiques
    - Utilisation de `deselect()` et `select()`

106

Applet carte GemXpresso et JCRE (2/2)

- **Communication**
  - Java Card
    - Classe qui hérite de `javacard.framework.Applet`
    - APDUs traités par `process(apdu)` de l'applet
  - GemXpresso
    - Classe qui implémente une interface et hérite de `javacard.framework.Applet`
    - *Si* commande APDU = `invoke` de DMI
      alors décodage par le JCRE et invocation d'une méthode de l'interface de l'applet
    - *Sinon*, méthode `process(apdu)` appelée par le JCRE
      - Si `process()` non redéfinie, ne fait rien et renvoie `90 00`

107

Applet «Compteur» : interface `ICompteur`

```
package gemxpresso.carte.autrans.fr ;

import javacard.framework.* ;

public interface ICompteur
{
  public static final short VALEUR_NEGATIVE = 1;

  public int lire();

  public int incrementer( int montant )
    throws UserException;

  public int decrementer( int montant )
    throws UserException;
}
```

108

Applet «Compteur» : classe `Compteur`

```
package gemxpresso.carte.autrans.fr ;

import javacard.framework.* ;

public class Compteur extends Applet implements Icompteur {
  private int valeur;

  public Compteur() { valeur = 0; }

  public int lire() { return value; }

  public int incrementer(int montant) throws UserException {
    if (montant<0) throw new UserException(VALEUR_NEGATIVE);
    valeur += montant;
    return valeur;
  }

  public int decrementer(int montant) throws UserException {
    if (montant<0 || valeur-montant<0)
      throw new UserException(VALEUR_NEGATIVE);
    valeur -= montant;
    return valeur;
  }
}
```

109

Applet «Compteur» : extensions

- **Ajout d'un autre constructeur**

```
public Compteur( int valeur ) { this.valeur = valeur; }
```

- **Compatibilité avec applet «Compteur» Java Card**
  - Ajout des méthodes `install()` et `process()` identiques à l'applet Java Card
  - Permet à des terminaux travaillant avec l'applet Java Card (communiquant donc avec les APDUs traités par celle-ci) de continuer de travailler avec l'applet GemXpresso
    - Java Card utile pour reprogrammer des applications existantes
    - GemXpresso permet de développer facilement de nouvelles applications

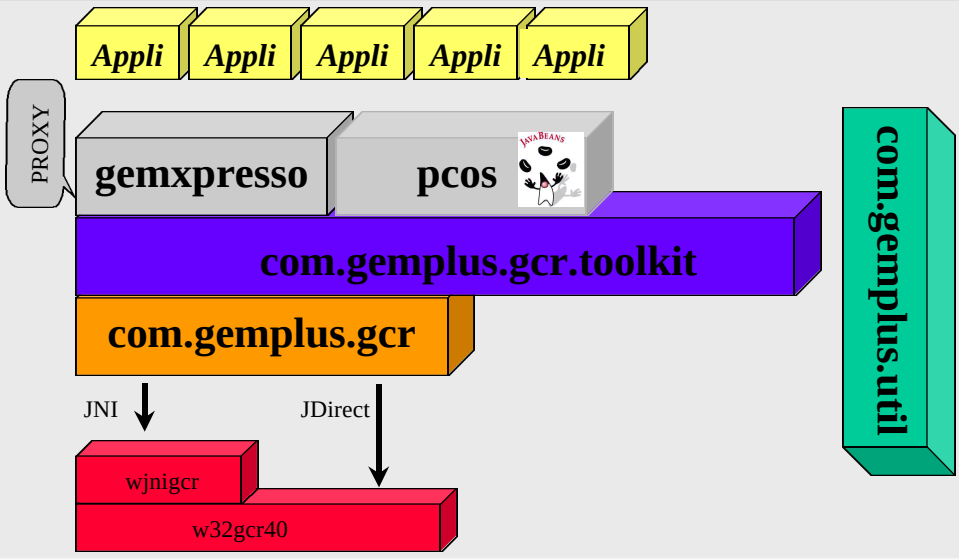
110

Construction de l'application cliente

- **Construction de l'application terminal**
  - Implémentation des classes du terminal avec JDK
  - Communication avec le serveur (applet carte)
    - Utilise l'interface de l'applet via le proxy généré
    - Établissement de la liaison : envoi d'un APDU de selection avec l'AID de l'applet (standardisé)
      - Méthode `select()` du proxy
    - Invocation de méthodes de l'applet par l'utilisation du proxy
  - API de communication avec la carte
    - API cliente du Gemplus SDK
    - Paquetage `com.gemplus.gcr.toolkit.gemxpresso`

111

API cliente du Gemplus SDK



112

API Cliente du Gemplus SDK

- **Paquetage** `com.gemplus.gcr.toolkit`
  - «Framework» pour les applications clientes de cartes
- **Paquetage** `com.gemplus.gcr.toolkit.gemxpresso`
  - `class GxCard extends JavaCard` (`extends Icc`)
    - Représente la carte GemXpresso
    - Gère le protocole DMI : `invokeConstructor(...)` et `invokeMethod(...)`
  - Classes `DmiOutputStream` et `DmiInputStream` pour codage et décodage des messages DMI
  - Classes `ConnectionException`, `CommunicationException` et `CardError` pour les erreurs de communication ou carte
  - Classe `CardVMUnexpectedException` pour une exception retournée par la JCVM

113

Client «Compteur» : proxy (entête)

```
import gemxpresso.carte.autrans.fr.ICompteur;
import com.gemplus.gcr.toolkit.gemxpresso.*;

public class GxCabCompteur
  extends GxCab
  implements ICompteur
{
  public static final GxAID GXAID = GxAID.create(...);

  public GxAID __getAID() { return GXAID; }

  public GxCabXCompteur( GxCard carte ) { super(carte); }

  // Installation de l'applet
  public void Compteur()
  {
    this.gemXpresso.invokeConstructor(-2, // #contructeur
      null); // Valeur des paramètres
  }

  // Autres constructeurs

  // Méthodes
}
```

114

Client «Compteur» : proxy (méthode)

```
public int decrementer( int montant ) throws UserException {
  try
  {
    // Construit les paramètres DMI
    DmiOutputStream __dmiParameters = new DmiOutputStream() ;
    __dmiParameters.write( montant ) ;
    // Invoque la méthode
    byte[] __b = this.gemXpresso.invokeMethod(3, // #Méthode
      DMITYPE.INT, // Type de la valeur de retour
      __dmiParameters.toByteArray()); // Valeur paramètres
    __dmiParameters.close() ;
    // Retourne la valeur de retour
    DmiInputStream __dmiRV = new DmiInputStream( __b ) ;
    return __dmiRV.readInt() ;
  }
  // Propage seulement les exceptions déclarées et sous-types
  catch ( CardVMUnexpectedException __exc )
  {
    if (__exc.getCardVMException() instanceof UserException)
      throw (UserException)__exc.getCardVMException() ;
    // Les autres exceptions restent encapsulées
    throw __exc ;
  }
}
```

115

Client «Compteur»

```
CardReader lecteur = new new Gcr410(SERIALPORT.G_COM1);
GxCard carte = new GxCard(lecteur);

try {
  // Connexion à la carte via lecteur GCR410
  reader.connect();
  AnswerToReset atr = carte.connect();

  // Communication avec l'applet via le proxy
  GxCabCompteur compteur = new GxCabCompteur(carte);
  compteur.select();
  System.out.println(''Montant du compteur = '' +
    compteur.decrementer( 100 ) );
  /* etc */

  // Fin de la connexion
  carte.dispose();
  lecteur.dispose();

} catch ( UserException e1 ) {
   System.out.println(''UserException : '' + e1.getReason());
} catch ( Exception e2 ) {
  System.out.println(''Problème : '' + e2.getMessage());
}
```

116

Conclusion GemXpresso

- **Méthodologie de développement d'applications carte**
  - Basée sur Java Card pour programmer la carte
  - Basée sur RPC pour communication client-applet
- **Points positifs**
  - Environnement ouvert
  - Pas de codes spécifiques à la carte
  - Intégration dans les systèmes d'informations

117

Sujets de recherche

Encore mieux intégrer la carte dans les applications réparties (des pistes)

Améliorations

- **Java Card**
  - Normalisation format de Bytecode et procèdure de chargement
  - Politiques de sécurité et de partage d'objets
  - Ramasse-miettes
  - APIs : `String`, *etc.*
- **DMI**
  - «Normalisation» du concept
  - Passage d'objets par référence ou valeur
  - Appel de méthodes depuis la carte vers le client
  - Sécurisation du protocole

119

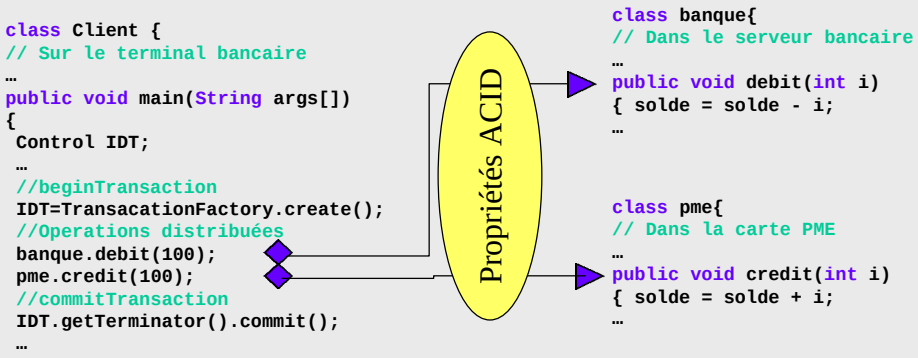
Intégration dans les systèmes

- **Voir la carte comme...**
  - Un serveur RMI ou CORBA ou DCOM
- **Services des applications réparties et carte**
  - Nommage et AIDs carte
  - Sécurité et accès à la carte
    ou carte participant au service de sécurité
  - Transactions distribuées (voir après)
  - *etc.*
- **Méthodes formelles pour certification des applets carte**
  - Preuve formelle du convertisseur (voir après)

120

Transactions avec des cartes

- **Permettre d'intégrer les cartes dans les schémas transactionnels distribués (OTS)**



121

Preuve formelle du convertisseur (1/2)

- **Prouver que le Bytecode Java Card converti présente les mêmes garanties de sécurité que le Bytecode Java original**
  - Augmente la confiance dans le convertisseur
  - Permet une certification des applets à partir du Bytecode Java original et du convertisseur (prouvé)
- **Propriétés du Bytecode Java Card à vérifier**
  - Propriétés vérifiées statiquement par le vérifieur Java
    - Typage, pile, initialisations, confinement, *etc.*
  - Propriétés relatives au modèle de sécurité Java Card
    - À définir...

122

Preuve formelle du convertisseur (2/2)

- **Deux parties**
  - Preuve de la validité des transformations effectuées
  - Preuve de la conformité du convertisseur par rapport à ces transformations
- **Utilisation de la méthode B pour :**
  - Modéliser les interpréteurs de Bytecode Java et Java Card, ainsi que les transformations effectuées sur le Bytecode
  - Faire la preuve de l'équivalence entre Bytecode Java et Bytecode Java Card
    - JVM = JCC + JCVM
  - Implémenter un convertisseur en fonction de ces modélisations

123

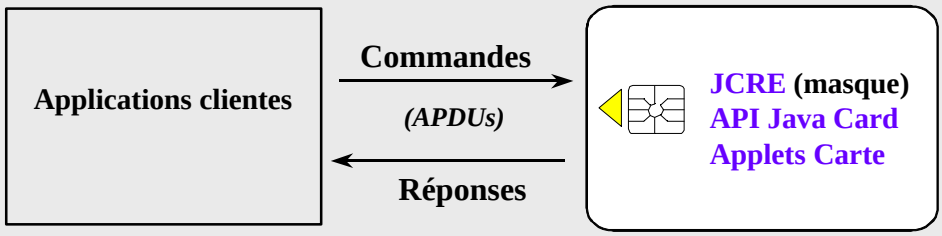
Conclusion

Où en sommes-nous ?

Résumé : applications avec Java Card

- **Langage Java pour programmer la carte**
- **Client-serveur bas niveau (échange d'APDUs)**

Terminal

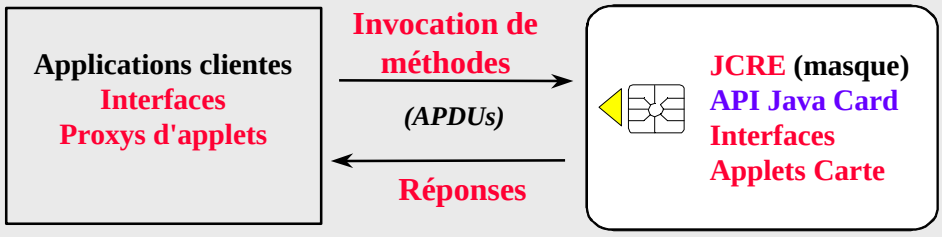


125

Résumé : applications avec GemXpresso

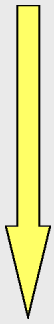
- **Techniques des applications réparties pour carte**
  - Protocole d'invocation de méthodes
  - Méthodologie de développement client-serveur

Terminal



126

Tendances de la carte à puce



Carte

- ISO 7816-4 (APDU)
- ISO 7816-7 (SCQL)
- Java Card (Java + DMI)

Client

- Orienté protocole
- SQL, ODBC, JDBC
- RMI, CORBA, DCOM

- **Carte à puce = élément informatique comme les autres**
  - Programmable
  - Intégrable dans les applications (réparties)
  - Potentialités d'applications carte de plus en plus importantes

127

Discussion

Des idées pour débattre

Cartes à puce et applications réparties

- **La carte devient un élément comme les autres des systèmes d'informations**
  - Environnement d'exécution Java dans la carte
  - Serveur d'objets dans les applications réparties
    - Objets personnels sécurisés
    - Objets distribués à chaque porteur
  - Serveur sécurisé dans les applications réparties
    - Données sensibles (clés cryptographiques) stockées et utilisées (algorithmes cryptographiques) dans un support "sûr"
      - Microcontrôleur encarté (sécurité physique)
      - Accès logique sévèrement contrôlé (sécurité logique)

129

Cartes à puce et applications réparties

- **La construction d'applications carte utilise les même techniques que celles des applications réparties**
  - Description IDL des services carte
  - Génération des souches clientes et serveurs
  - Protocole d'invocation de méthodes distantes au dessus du substrat de communication
  - Services horizontaux
    - Sécurité
    - Transactions distribuées
    - Nommage, *etc.*

130

Cartes à puce et applications réparties

- **Gemplus à l'école d'été d'Autrans**
  - Faire connaître la carte
  - Sensibiliser aux problèmes carte
  - Susciter le développement d'applications originales à base de cartes
- **L'école d'été d'Autrans chez Gemplus**
  - Faire connaître les techniques des applications réparties
  - Mettre en évidence la pertinence des techniques des applications réparties pour la carte
  - Susciter l'utilisation originale de la carte dans des applications réparties

131

Cartes à puce et applications réparties

- **Merci de votre acceuil et de votre compétence**
- **Continuons à "cross-fertiliser"**

Gemplus Research Group

jeanjac@research.gemplus.com

132

Pour obtenir plus d'informations (1/4)

- **Livres carte. Pas vraiment de bons livres mais les plus récents sont :**
  - Dreifus H. et Monk J. T., *Smart cards*, Wiley, 1998.
  - Guthery S. B. et Jurgensen T. M., *Smart Card Developer's Kit*, Macmillan, 1998.
    `http://ww.scdk.com`
  - Rankl W. et Effing W., *Smart Card Handbook*, Wiley, 1997.
- **Java Card**
  - Site Sun :
    `http://java.sun.com/products/javacard`
  - Java Card Forum :
    `http://www.javacardforum.com/`

133

Pour obtenir plus d'informations (2/4)

- **Articles et présentations**
  - Articles dans JavaWorld
    `http://www.javaworld.com/javaworld/`
    - Smart Cards: A Primer
      `jw-12-1997/jw-12-javadev.html`
    - Understanding Java Card 2.0
      `jw-03-1998/jw-03-javadev.html`
  - Présentations à Java One 98
    `http://java.sun.com/javaone/javaone98/sessions/`
    - Java Card Technology: The Java Card Platform and APIs
      `T802/index.html`
    - Java Technology: Using Smart Cards with Java Technology
      `T800/index.html`

134

Pour obtenir plus d'informations (3/4)

- Gemplus: Developing Smart Card-based Java Applications
  `T908/index.html`

- **Produits Java Card**
  - GemXpresso : `http://www.gemplus.com/gemxpresso/`
  - CyberFlex : `http://www.cyberflex.austin.et.slb.com/`
  - Odyssey : `http://www.cp8.bull.net/products/javacara.htm`
  - C@ppuccino : `http://www.gdm.de/products/terminal/software/javacap1.htm`
  - iButton : `http://www.ibutton.com/`

135

Pour obtenir plus d'informations (4/4)

- **API d'accès aux cartes à puce**
  - Open Card Framework (OCF)
    `http://www.opencard.org/`
  - PC/SC
    `http://www.smartcardsys.com/`
    `http://www.microsoft.com/smartcard/`
  - Gemplus Smartcard Development Kit (SDK) :
    - Pas de version publique
    - Livrée avec le kit GemXpresso
- **Recherche**
  - Actes de *Cardis 1996* et *Cardis 1998*

136