La carte à puce

Jean-Philippe Babau

Département Informatique

INSA Lyon

Certains éléments de cette présentation sont issus de documents Gemplus Research Group

jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

jean-philippe.babau@insa-lyon.fr

Introduction

- Carte à puce de plus en plus présente
  - commerce (électronique), identification, …
- La carte à puce est un système enfoui, mobile, embarqué
  - pas d'IHM, portable
- La carte à puce peut être un système critique
  - carte bancaire
- La carte à puce est un système dédié
  - 1 processeur et des mémoires limités, 1 interface de communication spécifique
- Une application avec la carte à puce est une application répartie
  - application « carte» = serveurs + terminaux (lecteurs) + cartes
  - traitements et données présents à la fois dans le terminal et la carte

jean-philippe.babau@insa-lyon.fr

Historique

- 1974 : Dépots de brevets par *Roland Moreno*
- 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
- 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.)

jean-philippe.babau@insa-lyon.fr

Types de cartes

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

jean-philippe.babau@insa-lyon.fr

Avec / sans contacts

- Carte avec contact
  - sécurité
- Carte sans contact
  - Distance lecteur : 0 -10 centimètres
  - Alimentation par le lecteur
  - Plus rapide en manipulation, moins d'usure
  - Transfert lent, coût plus élevé

jean-philippe.babau@insa-lyon.fr

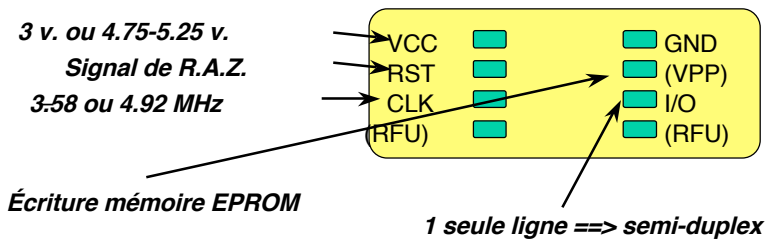
Microcontrôleur pour carte

- 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 inaccessible
  - détecteurs de conditions anormales
- Types de microprocesseur
  - 8-16-32 bits (+ coprocesseur cryptographique)
  - SGS-Thomson, Siemens, Motorola, Hitachi, NEC, *etc.*
- Types des mémoires
  - ROM jusqu'à 64 Ko, RAM jusqu'à 2 Ko
  - EEPROM jusqu'à 32 Ko

jean-philippe.babau@insa-lyon.fr

Normalisation : carte

- 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

jean-philippe.babau@insa-lyon.fr

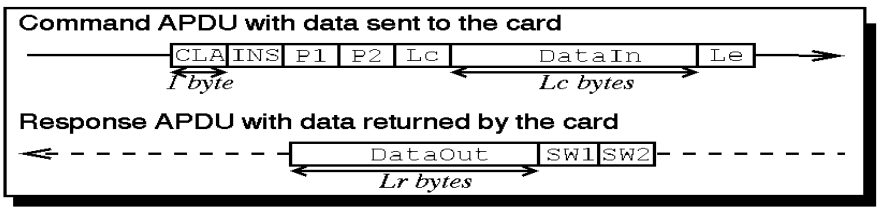
Normalisation : communications

- 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

jean-philippe.babau@insa-lyon.fr

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`)



jean-philippe.babau@insa-lyon.fr

Cycle de vie de la carte

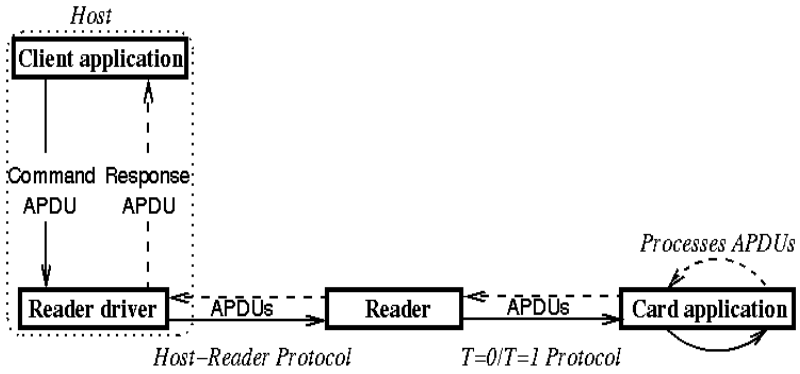
- Fabrication
  - Inscription d'un programme en mémoire ROM définissant les fonctionnalités de base de la carte
- Initialisation
  - Inscription en EEPROM des données communes à l'application
- Personnalisation
  - Inscription en EEPROM des données relatives à chaque porteur

Cycle de vie de la carte

- Utilisation
  - Envoi d'APDUs de commande à la carte
  - Traitement de ces commandes par 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.*

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



jean-philippe.babau@insa-lyon.fr

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
  - les pilotes offrent uniquement une API de transport des APDUs

jean-philippe.babau@insa-lyon.fr

Vers des cartes plus ouvertes

- Problèmes à résoudre et/ou besoins à satisfaire
  - permettre le développement de programmes pour la carte sans avoir besoin de graver un nouveau masque
  - faire de la carte un environnement d'exécution de programmes ouvert (chargement dynamique de code)
  - faciliter l'intégration des cartes dans les applications
- 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

jean-philippe.babau@insa-lyon.fr

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
  - Matériel
    - 16 ko de ROM, 8ko d'EEPROM, 256 o de RAM
  - Java Card définit un sous-ensemble de Java (1.0.2) 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

jean-philippe.babau@insa-lyon.fr

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)**

jean-philippe.babau@insa-lyon.fr

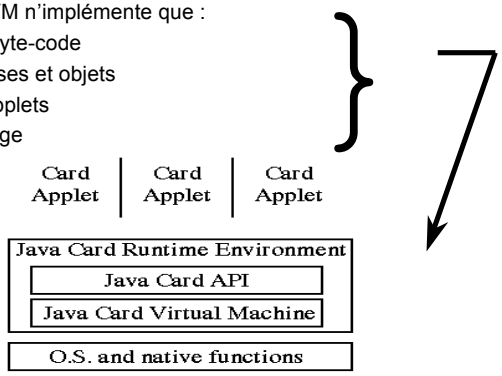
Java Card p/r à Java

- Supportés:
  - `boolean`, `byte`, `short`, `int`(optionnel)
  - `Object`
  - Tableau à une dimension
  - Méthodes virtuelles
  - Mots-clés `instanceof`, `super` et `this`
  - Allocation dynamique
  - Paquetages `public`, `protected` et `private`
  - Exceptions
  - Interface
  - Méthodes natives
  - Surcharge de méthodes, méthodes abstraites et interfaces
- Non supportés :
  - `float`, `double`, `long`, `char`, `String`
  - Tableau à n dimensions
  - Chargement dynamique de classe
  - Ramasse-miettes
  - `SecurityManager`
  - Threads
  - Clonage d'objet

jean-philippe.babau@insa-lyon.fr

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
  - isolement des applets
    - par paquetage



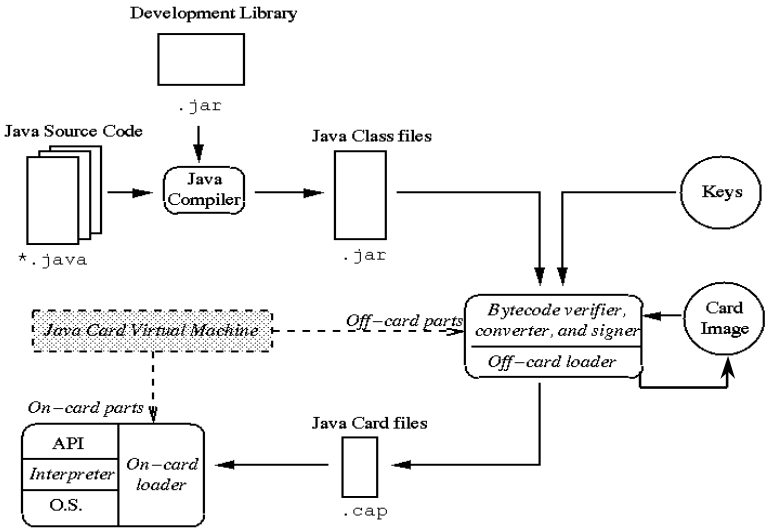
jean-philippe.babau@insa-lyon.fr

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»)

jean-philippe.babau@insa-lyon.fr

Machine virtuelle JC : architecture



jean-philippe.babau@insa-lyon.fr

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
- 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

jean-philippe.babau@insa-lyon.fr

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 persistante
  - 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 JavaCard
  - créé une seule fois (avec un identifiant AID unique)
  - toujours une applet firewall active par défaut
  - Objet partageable "shareable"

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
    - objets personnels sécurisés
    - objets distribués à chaque porteur
  - serveur sécurisé
    - données sensibles (clés cryptographiques) stockées et utilisées (algorithmes cryptographiques) dans un support "sûr"
      - microcontrôleur encarté (sécurité physique)
      - accès logique sévèrement contrôlé (sécurité logique)
- 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

Utilisation de la carte

- Carte à mémoire
  - carte «porte-jetons»
- Carte à logique câblée
  - carte «sécuritaire» pouvant effectuer des contrôle de code
- Carte à puce, JavaCard
  - carte «programmable» pouvant effectuer tout type de traitements
- Domaine
  - Identification, authentification
  - Suivi de produits, capteurs intelligents
    - Ibutton + capteur de température
  - Marquage / trace
  - Paiement
  - Dossiers, contrats (données + adresses)
  - Informations mobiles

Conclusion

- Système dédié
  - Ressources limitées
  - Cartes souvent dédiées à une application
  - Configuration statique
  - Protocole de communication spécifique (half duplex)
- Java Card
  - Adaptation au contexte de l'embarqué
  - Sous-ensemble de Java
    - Pas de thread, …
  - Gestion prédictible de la mémoire
  - Adaptation de la machine virtuelle
- Conception d'une application
  - Portage manuel des éléments embarquables
  - Conception dédiée