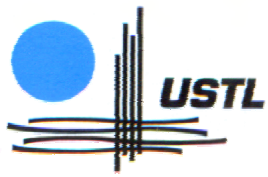
Université des Sciences et Technologies de Lille
Master Informatique Spécialité TIIR

Projets de veille Technologique



JavaCard

Xavier Perrin
Julien Janier
Emmanuel De Castro
Emile Gourlay

Janvier 2005

Sommaire

Introduction

La carte à puce est aujourd'hui un support très répandu pour stocker des informations. Ces exemples les plus courants sont les cartes bancaires, les cartes téléphoniques et les cartes SIM contenues dans les GSM. Il s'agit en fait d'une simple carte de plastique dans laquelle est intégrée une puce électronique.

Il en existe différents types dont la smartcard et plus particulièrement la javacard.

Toutes les cartes à puces possèdent des ressources très limitées disponibles pour l'exécution d'applications. Aujourd'hui, le moyen le plus sûr pour assurer un niveau de sécurité satisfaisant reste la carte à puce. Cependant, le développement d'applications pour carte à puce a toujours été difficile et réservé à des experts.

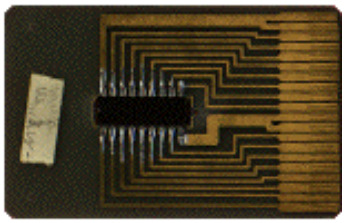
Il a donc fallut développer un langage qui soit à la fois fiable, robuste, peu gourmand en ressources et bien sûr simple. C'est en 1996 que Sun Microsystems Inc propose une solution après des essais menés par Schlumberger : le JavaCard.

Le JavaCard est un système de programmation de cartes à puces basé sur une version allégée du langage Java.

Généralités sur les cartes à puces

Nous allons commencer par voir des généralités sur les cartes à puces. Dans un premier temps nous verrons l'historique des cartes à puces et leur évolution depuis leur création. Puis nous verrons quelques applications qui utilisent les cartes à puces. Ensuite nous verrons les trois types principaux de cartes à puces. Enfin nous verrons un peu plus précisément le type de cartes qui nous intéresse : les SmartCard.

• *Historique*



Voyons d'abord l'historique. En 1974, Roland Moreno qui est chef d'une équipe de recherche pour l'entreprise Innovatron, va créer la première des cartes à puces. Cette carte est une carte à mémoire, nous verrons plus tard ce que signifie une carte à mémoire.

Puis en 1977, la carte à mémoire évolue vers un autre type de carte : la carte à micro-processeur. Bull, une entreprise française, crée le CP8 la première de ces cartes. La carte à microprocesseur servira en 1980 à mettre en place les premiers essais de carte bancaire. Puis en 1983 on verra apparaître toujours sur ce modèle de carte, la première carte santé.

En 1984, un autre type de carte à puce voit le jour, c'est une carte ayant une utilité différente. Elle est à micromodules et permettra de créer les premières télécartes, ou cartes téléphonique à utiliser dans des télécabines France Telecom.

• *Réalisations industrielles majeures*

Passons maintenant à la description de quelques réalisations industrielles majeures utilisant les cartes à puce. Tout d'abord la carte bleue, c'est l'une des premières cartes à microprocesseur.

Elle est mise en place en France et l'on compte en 2001, 40 M d'exemplaires de cette carte en circulation.

Une autre réalisation est la carte vitale : carte de santé basée elle aussi sur un microprocesseur. Elle est à ce jour distribuée à 40 M d'exemplaires.

Voyons maintenant le porte-monnaie électronique, c'est une carte sans contact qui est utilisé dans de nombreux pays : elle se décline en deux modèles. On compte 80 M de geldKarte en Allemagne / Luxembourg, et 80 M de cartes Proton en Belgique,

Suisse, Malaisie.

Enfin une carte qui a pénétré le marché de façon sans précédent : la carte SIM pour la téléphonie mobile. On dénombre 1.5 Md de cartes SIM vendues à travers le monde.

On pourrait aussi parler des cartes multi-services qui sert par exemple au sein d'une commune à la fois de pièce d'identité et de porte-monnaie pour faciliter l'utilisation des services de la ville aux habitants. On peut aussi parler des cartes sans contact comme les cartes de Restaurants Universitaires qui contiennent une puce interne.

• *Types de cartes*

Voyons maintenant les trois types principaux de carte à puce.

• 1- Carte à mémoire

Tout d'abord la carte à mémoire, c'est une mémoire simple sans processeur. Elle peut être lue sans protection, mais l'écriture peut-être rendue impossible. Sa programmation n'est pas possible puisqu'elle ne contient aucun processeur. On a comme exemple d'utilisation de ce type

de carte la carte "porte jeton" utilisée pour des applications de prépaiement (carte téléphonique par exemple).

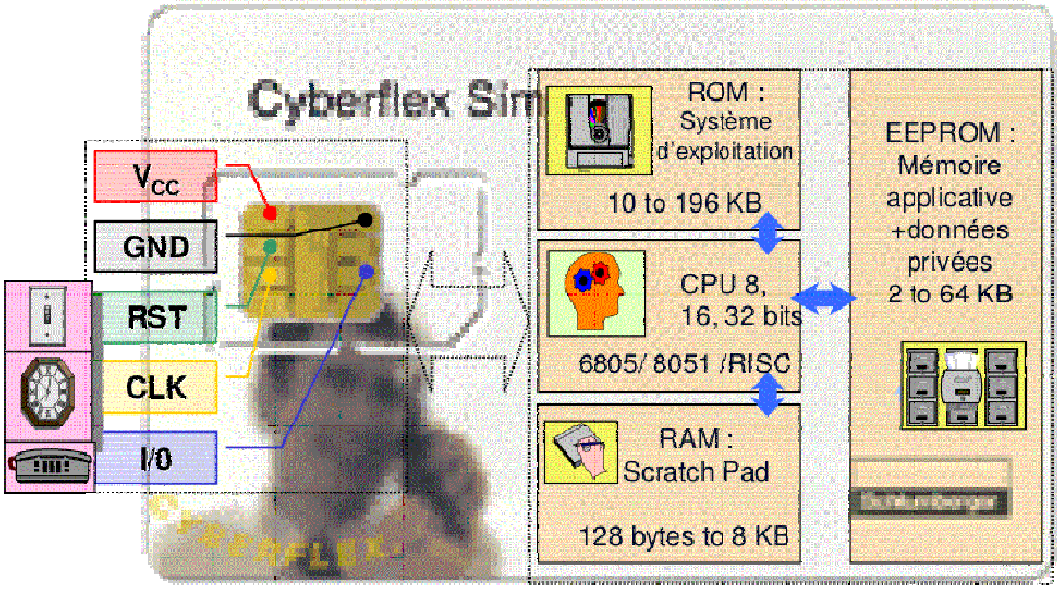
- **2- Carte à logique câblée**

Passons au second type de carte à puce : la carte à logique câblée. Sa mémoire est accessible via des circuits préprogrammés et figés pour une application particulière. Par exemple pour une carte "sécuritaire" pouvant effectuer des calculs figés (accès à un local, à un garage).

- **3- Smart Card**

Voyons maintenant le dernier type de carte à puce, les SmartCard. Les Javacard dont nous allons vous parler font partie des SmartCard. Ce type de puce contient un processeur et de la mémoire, le tout dans un microcontrôleur encarté. Cela permet à la carte d'être programmée pour effectuer un ou plusieurs types d'applications. Elle possède soit une interface électronique par contact, soit elle est sans contact est fonctionne par fréquence radio.

Architecture d'une smart card



Voyons donc l'architecture globale d'une SmartCard et les contraintes qu'elle provoque dans la mise en place d'applications. La SmartCard contient tout d'abord :

- une ROM qui contient le JCRE (Java Card Runtime Environment) et le Système d'exploitation, elle peut atteindre 2 à 4 Mega-octets.

- Il y a aussi le processeur (CPU) qui est très faible comparé aux processeurs de PC, il ne contient qu'un jeu d'instructions limité.

- La RAM (effacée à chaque nouvelle utilisation de la carte) est elle aussi peu élevée, elle atteint les 8 kilo-octets. Elle stocke les données sensibles (clés de cryptographie ...).

- Enfin l'EEPROM qui stocke les applets à executer sur la carte SmartCard. Elle n'est pas effacée et les données sont donc réutilisables dans le même état à chaque utilisation de la carte.

II JavaCard

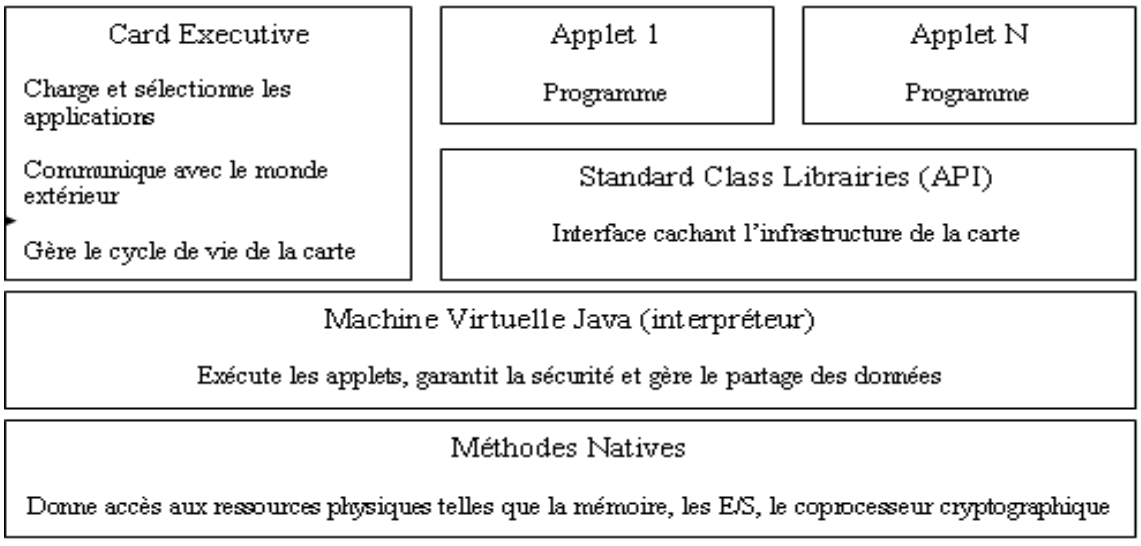
• *Qu'est ce que la JavaCard ?*

Toutes les cartes à puces sont caractérisées par des ressources très limitées disponibles pour l'exécution d'applications. En outre, l'explosion des réseaux de télécommunication et des transactions électroniques ont augmenté le besoin en applications sans négliger la sécurité. Aujourd'hui, le moyen le plus sûr pour assurer un niveau de sécurité satisfaisant reste la carte à puce. Cependant, le développement d'applications pour carte à puce a toujours été difficile et réservé à des experts.

Il a donc fallut développer un langage qui soit à la fois fiable, robuste, peu gourmand en ressources et bien sûr simple. C'est en 1996 que Sun Microsystems Inc propose une solution après des essais menés par Schlumberger : le JavaCard.

Le JavaCard est un système de programmation de cartes à puces basé sur le langage Java et la Java Virtual Machine. JavaCard est une version épurée du langage Java s'adaptant aux environnements réduits des cartes à puces. L'environnement minimum nécessaire au fonctionnement de l'API JavaCard est un processeur 300 KIP, 12 KO ROM, 4 KO EEPROM, et 512 Octets de RAM. Un certain nombre de composants, par exemple les tableaux multidimensionnels, les Threads et le garbage collector ..., ont été enlevés.

• *Architecture de JavaCard.*



L'architecture de la JavaCard est composée de plusieurs couches.

- Les « Native Methods » contiennent un ensemble de fonctions bas niveau qui permettent au Java Card Runtime Environnent (JCRE) de gérer les ressources physiques de la carte telles que les entrées-sorties, la gestion de la mémoire ou le coprocesseur cryptographique. Ces méthodes sont compilées en code exécutable dédié au processeur de la carte.

- L' « interpreter » est le coeur de l'architecture JavaCard sur la carte. C'est le moteur d'exécution des Applets ou Cardlets. Il exécute les programmes en convertissant le code pré-compilé en appel aux méthodes natives. Il fournit une couche d'abstraction entre les réalités physiques de chaque carte et le code fourni par le « Converter » (compilateur JavaCard), ce qui rend les programmes JavaCard complètement indépendant de l'environnement utilisé, à condition d'utiliser une carte de type JavaCard.

- Les Standard Class Libraries ou encore le Javacard Framework est un ensemble d'API. L'API contient l'ensemble de classes qui sont les outils de bases pour créer une Applet. Cette partie sera détaillée au prochain chapitre.

Cet ensemble (Native Methods, Interpreter et les Standard Class Libraries) est appelé JCRE. Il est contenu dans le masque des JavaCard et est donc inamovible.

• *Les avantages apportés par JavaCard.*

Voici les principaux avantages du langage qui justifie le choix de ce standard en ce qui concerne la programmation des cartes à puces.

• Langage de haut niveau orienté objets :

Il possède donc tous les avantages de ce type de langage (encapsulation des données, héritage...).

Il permet une structuration plus naturelle du code et donc plus compréhensible. Le concept d'héritage permet la réutilisation du code.

La création de librairies est simplifiées : les packages.

Auparavant, le développement de programmation carte était réalisé en assembleur ou en C. Cela était moins souple et plus technique. La mise au point de programmes sera plus rapide et les temps de développement seront sensiblement réduits.

Ce langage accroît la sécurité : gestion stricte de la mémoire, pas d'arithmétique de pointeur, « access scope » (vérification des accès) des membres d'un objet, langage fortement typé.

- **Write Once, Run Anywhere :**

Le code d'un programme JavaCard est universel au sens où il est exécutable sur n'importe quelle machine virtuelle JavaCard. Le JavaCard n'introduit rien de spécifique et respecte la norme ISO-7816 dans le dialogue avec un lecteur de carte.

- **Plate-forme multi applicative :**

JavaCard possède un modèle de sécurité qui permet à plusieurs applications de coexister en sécurité sur la même carte. Chaque applet possède son espace de mémoire propre (sandbox). Une carte peut contenir une application bancaire, un accès pour télévision satellite, un carnet de santé...

- **Partage de données entre applications :**

Les différentes applications présentes sur une carte peuvent accéder à des données d'une autre application si le programme a été envisagé via le firewall. Seul le JCRE peut accéder à toutes les données. Une applet peut contenir les identifiants de sécurité sociale qui peuvent être utiles pour une applet mis par la mutuelle du détenteur.

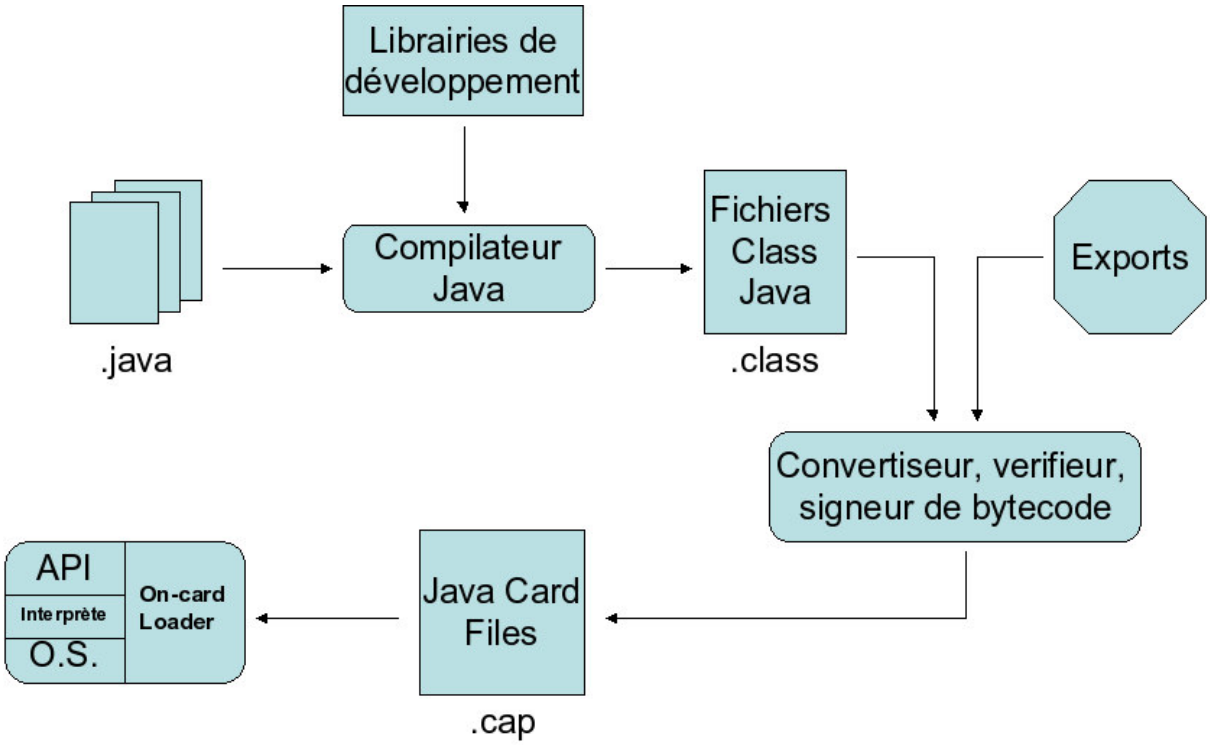
- **Sécurité des données :**

Comme nous l'avons vu, le JCRE gère strictement la mémoire de chaque Applet de façon étanche. L'API JavaCard intègre les outils de base pour des applications d'authentification et de signature. Les données sensibles sont détenues par le JCRE : certificat, code PIN... Nul ne peut les lire directement.

- **Souplesse :**

Le fait que les Applets soient chargées dans une EEPROM, permet de les mettre à jour en effaçant l'ancienne version et en chargeant la nouvelle. Cette évolutivité rend une carte pleinement réutilisable, alors qu'auparavant les applications étaient directement inclues dans le masque et donc non modifiables. Les modifications et chargement d'Applet font appel à des mécanismes sécurisés par le « Card Manager ».

Voici un schéma représentant les différentes étapes pour créer une applet et l'exécuter.



• **Schéma d'une applet :**

Pour développer une application JavaCard, il faut tout d'abord importer ses bibliothèques. Les bibliothèques sous Java s'appellent des packages. Le programmeur a la charge de déclarer les bibliothèques aux quelles il souhaite accéder pour son programme. La bibliothèque de base pour l'API JavaCard est le package javacard.Framework. Ce Package contient les classes indispensables à la réalisation telles que :

- APDU (Application Protocol Data Unit), qui est le format de communication entre la carte et le monde extérieur.
- Applet, c'est de cette classe que doivent étendre toutes les Applets.
- AID (Application Identifier), c'est un identifiant qui est associé à une Applet.
- Les Exceptions, qui permettent d'envoyer des Status Word d'erreur.

L'interface ISO-7816, qui contient les Status Word de base et les positions des éléments d'une APDU. Il existe aussi un package sécurité (javacard.security) qui contient les fonctions de chiffrement usuelles (MD5, SHA, DES ...) et de signature. Lorsque le JCRE reçoit une APDU de sélection, il cherche à activer une applet correspondante. Une applet étant une instance de la classe javacard.Framework.Applet, toute applet devra étendre de cette classe. Ce mécanisme d'extension permet aussi d'utiliser les méthodes de la classe Applet ou bien de les surcharger. La surcharge d'une méthode consiste à réécrire le code de celle-ci, afin de modifier son comportement. Les méthodes à implémenter obligatoirement sont pour ne pas avoir le comportement par défaut de la classe Applet :

- install()
- process()
- select()
- deselect()

La fonction **install()** est appelée par le JCRE pour créer et enregistrer (par la méthode **register()**) une instance d'applet auprès de celui-ci.

Les AID servent d'identificateur pour le JCRE. Chaque AID d'une applet présent sur la carte doit être unique et correspondre à la norme ISO-7816.

Une applet est sélectionnable seulement après avoir été enregistrée par le JCRE.

La fonction **process()** est la fonction principale de l'applet, c'est celle qui va collecter toutes les APDU entrantes. Elle envoie ces APDU vers les méthodes qui correspondent à l'octet d'instruction (INS). Le principal travail d'un programmeur se situe dans la gestion du dialogue entre la carte et le lecteur. C'est dans cette méthode qu'il doit être géré.

La fonction **select()** est appelée par le JCRE pour rendre active et initialiser une instance d'applet. Comme défini dans la norme ISO-7816, toute Applet doit rester sélectionnable. Il ne faut pas bloquer la sélection.

La fonction **deselect()** est appelée par le JCRE pour désactiver l'applet en-cours et faire les possibles opérations nécessaires avant la désélection (Ex : Clôture d'une transaction).

- **La Compilation:**

La compilation des programmes JavaCard commence par l'utilisation d'un compilateur Java du JDK standard. Il n'y pas de compilation pour la plate-forme JavaCard mais une traduction du bytecode vers un autre format dédié à JavaCard.

La compilation et la conversion se font en deux étapes. La compilation est laissée à la charge au compilateur java classique, seule la conversion est prise en charge par les outils du JDK JavaCard.

Les classes de l'API JavaCard doivent être importées et définies dans le ClassPath du compilateur

Pour pouvoir faire l'édition des liens. Le compilateur javac ne tient pas en compte le fait que

JavaCard soit un sous ensemble de Java et ne renvoie pas de message d'erreurs lors de la compilation.

La vérification est faite lors de la conversion. Lors de la conversion, voici les taches accomplies

Par le converter :

- Vérifier que les images chargées des classes Java sont correctes.
- Vérifier la conformité par rapport au langage JavaCard (vu ci-dessus).
- Initialiser les variables statiques.
- Résoudre les références symboliques des classes, méthodes et champs vers une forme plus compacte et mieux adaptée à la carte.
- Optimiser le bytecode en tirant avantage obtenu au chargement et à l'édition des liens.
- Allouer l'espace nécessaire et créer les structures de données pour représenter les classes auprès de la machine virtuelle Le fichier issu de la conversion (bytecode JavaCard) est appelé « cap file ».

- **Machine Virtuelle**

Une fois le code compilé, on se retrouve avec un fichier .class. Une étape supplémentaire est nécessaire pour que l'applet passe du monde extérieur vers la carte à puce.

La machine virtuelle

JavaCard est implémentée en 2 parties : la off-card et la on-card. La partie Off-card est appelée Java Card Converter. Elle permet le chargement des classes et de la résolution des références.

Elle vérifie et convertit le byte-code. La partie On-card qui a pour tâche d'interpréter le byte-code comme nous l'avons vu précédemment.

Après un reset, le JCRE entre dans une boucle d'attente. L'applet ne peut pas communiquer directement avec le monde extérieur. Tout ce dialogue passe par le JCRE. De plus, le JCRE peut interrompre l'exécution d'une applet lors du lancement d'une exception. Les applets ne lancent pas les exceptions mais demandent au JCRE de le faire pour elles. On peut parler d'une prééminence du JCRE sur les applets.

• Exemple simple d'applet JavaCard

Voici un exemple simple d'applet java.
Dans un premier temps à l'instanciation de la class l'applet s'enregistre auprès du JCRE grâce à la méthode register().
Ensuite elle se contentera de répondre au APDU, en renvoyant ceux-ci.

```
public class HelloWorld extends Applet
{
    private byte[] echoBytes;
    private static final short LENGTH_ECHO_BYTES = 256;

    /**
     * Only this class's install method should create the applet object.
     */
    protected HelloWorld()
    {
        echoBytes = new byte[LENGTH_ECHO_BYTES];
        register();
    }
    /**
     * Installs this applet.
     * @param bArray the array containing installation parameters
     * @param bOffset the starting offset in bArray
     * @param bLength the length in bytes of the parameter data in bArray
     */
    public static void install(byte[] bArray, short bOffset, byte bLength)
    {
        new HelloWorld();
    }
    /**
     * Processes an incoming APDU.
     * @see APDU
     * @param apdu the incoming APDU
     * @exception ISOException with the response bytes per ISO 7816-4
     */
    public void process(APDU apdu)
    {
        byte buffer[] = apdu.getBuffer();
        short bytesRead = apdu.setIncomingAndReceive();
        short echoOffset = (short)0;

        while ( bytesRead > 0 ) {
            Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA,
                    echoBytes, echoOffset, bytesRead);
            echoOffset += bytesRead;
            bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
        }
        apdu.setOutgoing();
        apdu.setOutgoingLength( (short) (echoOffset + 5) );

        // echo header
        apdu.sendBytes( (short)0, (short) 5);
        // echo data
        apdu.sendBytesLong( echoBytes, (short) 0, echoOffset );
     }
}
```

• *Installation des applets*

Une JavaCard à l'origine contient le JCRE et tout le système du constructeur de la carte. Le JCRE est « écrit » dans la ROM de la carte : c'est le masque de la carte. Il est donc inviolable. Le code des applets est stocké sur la carte généralement en EEPROM.

• L'installation

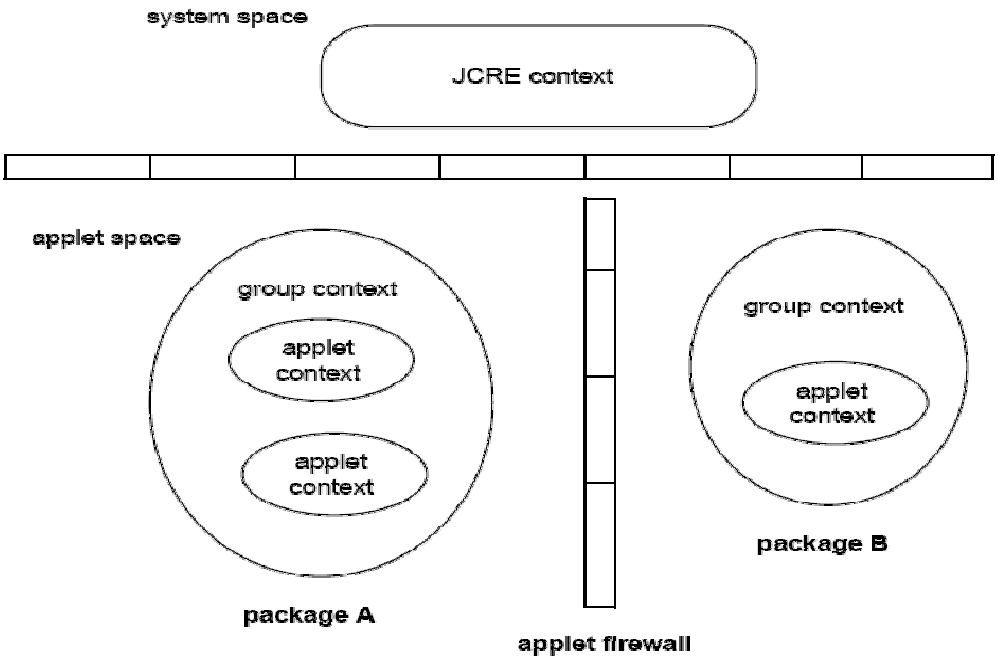
L'installation d'une applet se fait alors que le JCRE tourne. Cela se passe en deux phases.

1. chargement du code de l'applet (cap file) sur la carte
2. création d'une instance de l'applet

Pour installer l'applet, l'installeur prend le fichier .cap et télécharge le contenu du fichier sur la carte. L'installeur écrit le contenu dans l'EEPROM et fait les liens entre les classes dans le fichier .cap et les classes résidant sur la carte. Il crée et initialise les données qui seront utilisées par le JCRE en interne pour se servir de l'applet. Si l'applet nécessite plusieurs package, chaque fichier .cap correspondant à un package sera chargé sur la carte.

L'édition de liens se fait statiquement à l'installation et donc pour faire cette édition on a besoin de toutes les dépendances.

• *Coté sécurité : L'applet Firewall*



Les applications JavaCard peuvent provenir de fabricants différents, et ne doivent pas pouvoir communiquer entre elles. Ces applications sont appelées des applets identifiés uniquement par leur AID (Application Identifier) isolées entre elles par un applet firewall.

Celui ci partage le système d'objet javacard en espaces d'objets protégés et séparés appelés contextes. Le firewall est la barrière entre les contextes. Quand un applet est instancié, la JCRE lui assigne un contexte mais aussi un groupe de contexte dans lequel peut se trouver plusieurs applets.

Les applets dans un même groupe peuvent se voir. Le firewall isole également le contexte du JCRE

- ***RPC et systèmes à objets répartis***

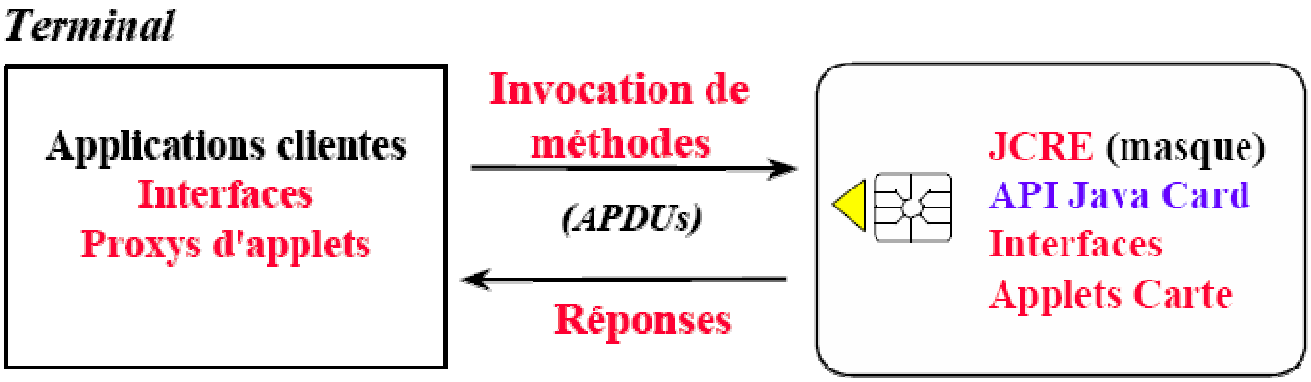
L'approche de l'aspect développement de JavaCard est de construire les applications avec la carte comme des applications réparties.

Le principe est simple, le principe de Java RMI est retenu !

La notion d'interface objet est utilisée pour décrire les objets distants.

Il y a ensuite un pré-compilateur qui génère les couches (skell stub)

Voici un schéma qui explique le principe de fonctionnement.



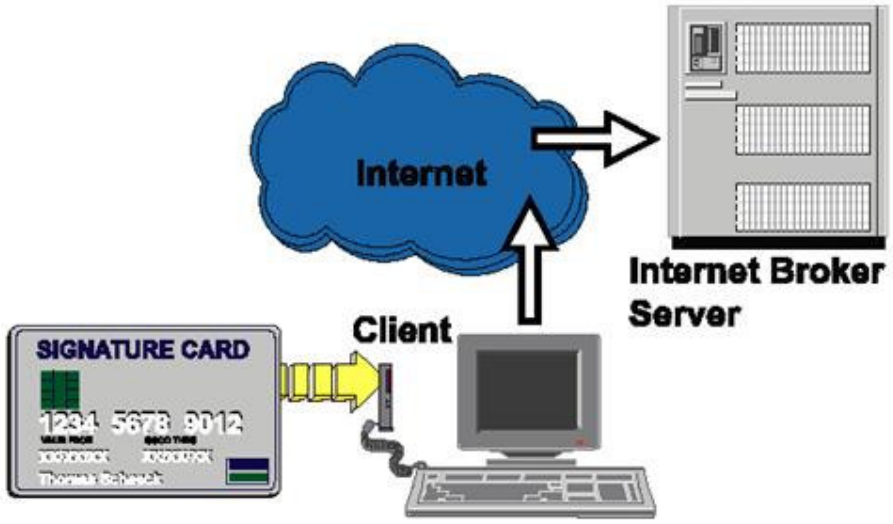
III Coté terminal

• *Généralités sur les terminaux*

• Qu'est ce qu'un terminal?

Nous appelons terminal, l'appareil permettant de relier la carte à puce à un service informatique. Ils en existent pour les cartes à contact et sans contact mais selon l'utilisation recherchée, différentes formes et fonctions peuvent être utilisées.

Par exemple, sur le schéma qui suit, nous prenons l'exemple d'un service bancaire.



Dans tous les cas, le terminal sert de relais entre la carte a puce et la banque mais selon que l'on désire retirer de l'argent ou régler un achat, le terminal (l'appareil ou l'on introduit la carte) diffère.

D'autre différences, moins visible, peuvent aussi exister. La taille de la carte, les protocoles de communications avec le service distant sont autant de points qui peuvent varier.

La norme ISO 7816

La norme ISO 7816 se décompose en une dizaine de sous parties (dénommée 7816-X, où X est le numéro de la partie). Nous n'en verrons en détails que quatre, celles qui nous intéresse le plus.

La partie 1 définit caractéristiques physiques

La partie 2 impose la dimension et la position des contacts

La partie 3 normalise les caractéristiques électriques et les protocoles de communication

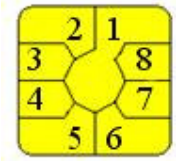
La partie 4 définit le format des paquets de données échangés entre un lecteur et une carte

ISO 7816-1

Comme nous l'avons dit précédemment, cette partie définit les caractéristiques physiques comme le célèbre format "carte de crédit " (85 x 54 x 0.76 mm). Elle définit également les contraintes physiques auxquelles la carte doit résister, par exemple résistance à la flexion et à la torsion mais aussi aux rayons ultraviolets, aux rayons X, et résistance électrique des contacts du microcircuit.

ISO 7816-2

Cette deuxième partie définit la dimension et la position des contacts des microprocesseurs qui sont implantés dans les cartes.



1 : GND
2 : VCC
3 : RESET
4 : CLK
5 : RFU
6 : RFU
7 : I/O asynchrone
8 : VPP

ISO 7816-3

Cette partie définit le protocole de communication qui doit être utilisé par les cartes afin de dialoguer avec le monde extérieur. Cette partie décrit les signaux électriques et les informations qui sont échangées lors de toute communication. Les courants, les tensions, les fréquences des signaux sont normalisés ainsi que le format des données échangées lors d'un dialogue entre une carte et un lecteur de carte.

Quelques caractéristiques données par la norme 7816-3 :

La carte ne dispose pas de sa propre horloge, elle est fournie par le terminal.

La transmission est du type asynchrone.

La communication est du type half-duplex, c'est à dire que les entrées et les sorties s'effectuent par alternance et en se servant de la même ligne physique (On s'en rends bien compte car il n'existe qu'un connecteur pour les entrées/sorties sur la puce).

Note:

La carte ne prend jamais l'initiative de l'échange d'informations. Elle ne fait que répondre à des commandes.

ISO 7816-4

La norme ISO 7816-4 a pour objectif de définir le format des «paquets APDU» (Application Programming Data Units) échangés entre un lecteur et une carte.

- **Le standard PC/SC.**

Les normes modernes, telle que la norme ISO 7816, ont pallié à l'incompatibilité des lecteurs, cartes et applications, qui justifiait la faible adoption des cartes en dehors de l'Europe. En ajoutant à cela les indéniables cotés sécuritaire que la carte à puce offrent, on comprends bien l'attrait que celle-ci a dans le monde informatique. Mais, il subsistait toujours un certain nombre de problèmes dus à de sérieuses lacunes d'interopérabilité :

D'abord, il n'existait pas encore de standards pour l'interface PC aux lecteurs de cartes, ce qui rendait difficile toute application et créait invariablement des surcoûts de développement et de maintenance.

Ensuite, il n'existait pas non plus d'interface de programmation haut niveau encapsulant les fonctionnalités communes des cartes. Un tel logiciel aurait permis de diminuer les coûts et de simplifier le développement des logiciels.

Enfin, les mécanismes qui permettaient d'exploiter réellement les cartes multiplicatives n'étaient pas définis.

Afin de répondre à ces lacunes les grands fabricants de cartes, d'ordinateurs et d'éditeurs de logiciels (Groupe Bull, Hewlett-Pakard, Microsoft, Schlumberger, et Siemens-Nixdorf) se sont regroupés en mai 1996 et ont crée le Personal Computer/Smart Card (PC/SC) Workgroup. Ce premier ensemble s'est vu rejoint ultérieurement par de nouveaux membres tels que Gemplus, IBM Corporation, Sun Microsystems, Toshiba, et Verifone. Le groupe cherche à développer un certain nombre de spécifications dans le but d'améliorer l'interopérabilité nécessaire à l'utilisation des cartes en environnement informatique. En plus de ces spécifications, il a développé à la fois les procédés matériels, et les logiciels nécessaires à la validation de leur effort.

L'architecture définie par ces spécifications, en terme de logiciel et de matériel, se décomposent en plusieurs couches dont voici le détail:

La carte format carte de crédit, qui respecte la norme ISO 7816.

- **Le lecteur de carte.**

Le lecteur handler, qui respecte la norme ISO 7816 et implémente principalement les protocoles de communication.

La carte ressource manager est un composant majeur de l'architecture du PC/SC Workgroup. Il est responsable de trois points importants:

D'abord de l'identification et recherche des ressources (type de lecteur, de carte, et éventuel insertion, retrait de carte).

Ensuite il est prend en charge l'allocation du lecteur et des ressources.

Et enfin, il gère l'accès aux services offerts par une carte.

Le service provider encapsule les fonctionnalités offertes par une carte spécifique et la rend accessible grâce à l'interface de programmation haut niveau.

Le standard PC/SC offre donc une interface entre les drivers des lecteurs et la programmation. Ainsi, le lecteur devient transparent lors du développement. Il suffit alors d'appeler les fonctions identiques pour tous les drivers respectant ce standard. Tout logiciel développé autour du PC/SC devient portable sur n'importe quel lecteur compatible.

• *Opencard*

- **Qu'est ce qu'Opencard?**

Sous le nom d'Opencard, on englobe, le framework et le consortium qui est l'initiateur du projet et qui s'occupe de le maintenir.

Le framework OpenCard est un framework de développement de smart card orienté objet (Java)

- **Pourquoi un tel framework?**

Actuellement, l'industrie de la carte à puce, plus précisément des smartcard, est en plein essor et entre dans une phase de standardisation pour réduire les frais de développement de logiciels. Opencard Framework est apparu dans le but de standardiser cela. En effet, jusqu'à présent, le standard PC/SC est utilisé. Celui-ci, bien que facilitant le développement d'applications pour carte à puce, requiert que le fabricant de la carte et du terminal suivent ce standard. Opencard propose une autre solution, basé sur deux couches d'abstractions:

Card Terminal Layer: Il s'agit du driver du terminal. Il est fournit par le fabricant du lecteur. Il existe plusieurs implémentations:

Driver natif (sous linux ou dll windows)

Driver Java (pour les ports séries ou parallèles)

Passerelle vers un driver PC/SC

Card Service Layer:Il s'agit d'une interface de haut niveau permettant de dialoguer avec une carte en particulier. Ce driver est fourni par l'émetteur de la carte (ou par le développeur de l'applet). Cette couche masque l'encodage et le décodage des APDU: elle accepte en entrées du bytecode Java et le converti en requetes APDU. Les requêtes APDU retourné sont converties en bytecode Java. Il est important de noter la conversion des erreurs en exceptions.

On peut faire une similitude du framework Opencard avec le monde des bases de données. En base de donnée, nous utilisons JDBC sans nous demander le type de bases de données utilisées. Les accès à celle ci se fait de la même manière que l'on soit en présence d'une base My-SQL ou d'une base Oracle.

- **Quels sont les avantages d'Opencard?**

Au niveau du développement d'application

L'utilisation du framework Opencard facilite le développement et permet une forte portabilité du code. Il en résulte ainsi une réduction du coût de production d'une applet, celle ci pouvant être utilisé sur tous les supports.

Au niveau du fabricant de lecteur et de carte

Pour que ces acteurs proposent des drivers pour leurs terminaux ou pour leurs cartes, il est important qu'ils aient eux aussi des avantages. Avec Opencard, leurs produits sont compatibles avec tous les programmes existants pour peu que les drivers soient fournis. De ce fait, ils sont successibles de toucher de plus gros marché.

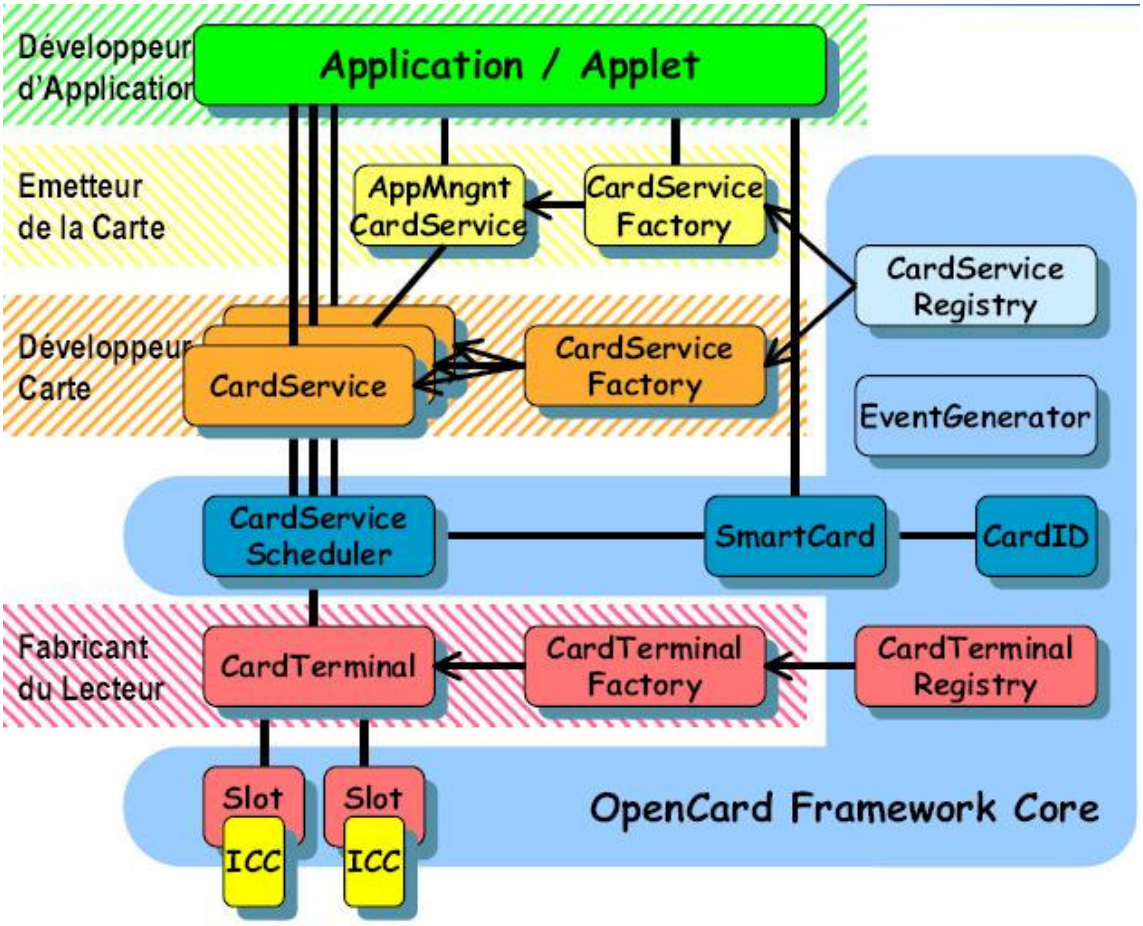
- **Opencard vs PC/SC**

Comme nous l'avons vu, le framework Opencard est beaucoup plus flexible et garantie une plus grande portabilité que le standard PC/SC. On ne voit donc plus l'intérêt d'utiliser Opencard aujourd'hui. Pourtant, nous ne parlons pas de compétition entre les 2 technologie. On parle plutôt d'une complémentarité de l'un par rapport à l'autre. En effet, l'utilisation d'Opencard oblige le développement en Java et la disponibilité des drivers.

En raison de la jeunesse du produit, ce second point est sensible: tous les fabricant ne proposent pas encore de drivers pour leurs produits. De plus, Opencard permet l'utilisation, via des passerelles, de driver PC/SC pour le Card Terminal Layer.

- **Architecture du framework**

Voici un schéma récapitulatif de l'architecture du framework:



IV Javacard en pratique

Voyons maintenant la mise en pratique de Javacard. Nous allons dans un premier temps voir le kit de développements mis à disposition par SUN, puis les différentes documentations accessibles pour développer des applets pour javaCard et OpenCard.

• *Le JavaCard Development Kit*

Pour JavaCard, la société Sun a créé un JDK spécialement conçu pour Javacard. Ont été mis à disposition un environnement de simulation regroupant toutes les fonctionnalités du framework dont nous avons parlé. C'est à dire, la cryptographie, le transport réseau par RMI ainsi qu'un adressage étendu. Plusieurs outils ont été apportés pour développer Javacard :

- **JcWde** : JavaCard Workstation Development Environment. C'est une machine virtuelle (en java) pour PC sur laquelle on peux tester une applet à porter sur JavaCard. On ne peux pas avec cette machine virtuelle utiliser les messages APDU.

- **C-Ref**: Machine virtuelle de référence en C pour executer les applets JavaCard (par simulation de la carte). Grâce à cela on peux utiliser les échanges de messages APDU.

- **ApduTool** : Utilitaire d'envoi et de réception de messages APDU.

- **Converter** : Utilitaire de conversion des fichiers .class en fichiers .cap .

Enfin, coté plateformes de développement, le JDK Javacard support actuellement le JDK 1.4.1 de Java et est utilisable en version Linux.

- ***Documentation du développeur***

Pour programmer javacard vous aurez besoin des Cartes à puce JavaCard et de terminaux pour lire les cartes. Cela peut s'acheter aisément sur Internet. Enfin toute la documentation javaCard 2.1.1 dont vous aurez besoin se trouvera sur le site de Sun où l'on trouve les spécifications :

- de la VM
- du JCRE
- de L'API.

La documentation pour OpenCard 1.2 est un peu moins fournie mais peut se trouver sur le site officiel Opencard ou sur celui de GemPlus le leader actuel sur le marché. On trouvera donc :

- une API
- un guide du programmeur.

Conclusion

Pour conclure, rappelons que les cartes à puces sont de plus en plus présentes autour de nous, qu'elles ont leur place dans la vie de tous les jours. De nombreuses applications pourraient voir le jour sur un tel support.

Pour les programmeurs il est nécessaire de pouvoir utiliser un langage évolué et objet pour remplacer le C ou l'assembleur. C'est pourquoi JavaCard est un standard qui tend à devenir une référence.

Comme vous l'avez appris javaCard propose un SDK complet aux outils performants et fortement documentés. On y trouve du réseau, de la cryptographie. OpenCard quant à lui offre un framework pour standardiser les échanges avec les terminaux.

On peut donc voir en ces deux technologies un avenir prometteur. Cependant on peut se poser la question des capacités actuelles de la carte à puce qui limite fortement ce que l'on peut obtenir comme performances. Dans un avenir plus ou moins proche les cartes à puces devraient voir leurs performances augmenter considérablement, et le framework se rapprocher petit à petit de celui du java actuel.

En attendant cet avenir, il existe une alternative à JavaCard : c'est JITS - Java In The Small développé par l'équipe RD2P du laboratoire de recherche du LIFL. Cette équipe propose le mélange de l'OS et de la JVM pour plus de légèreté et de flexibilité. L'API de JITS est redéfinie à partir de celle de Java et a été nettement moins réduite que l'API de JavaCard. C'est ainsi que certaines fonctionnalités comme le garbage Collector ou encore certains types comme les Strings ont été conservés.

Les sources

- Java Card Platform Specification v 2.2.1
- Java Card RMI Client API
- How to Write a Java Card Applet: A Developer's Guide
- Java Card Security White Paper
- Java Card FAQ
- Java Card Technology for Smart Cards: Architecture and Programmer's Guide
- Smart Card Overview

- http://java.sun.com/products/javacard/reference/docs/
- http://etna.int-evry.fr/~bernard/sarp/projets00-01/CartePuce/fonctionnement.htm
- http://www.opencard.org/
- http://www.gemplus.com/