République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
Département d'Electronique

PROJET DE FIN D'ETUDE POUR L'OBTENTION DU DIPLÔME D'INGENIEUR D'ETAT EN ELECTRONIQUE

THÈME

LES CARTES A PUCE JAVA CARD

DEVELOPPEMENT D'UNE APPLICATION DE PORTE-MONNAIE ELECTRONIQUE

Proposé et dirigé par :

Mr ABDELOUEL Lahcène
Madame HAMAMI Latifa

Réalisé par :

TAZAIRT Réda

Année Universitaire 2001/2002

IV API Java Card

V Sécurité de la plate-forme Java Card

VI Outils de développement Java Card

Deuxième partie: Construction de l'application carte

VII Guide de développement

Chapitre 1
Etat de l'art de la carte à puce

1.1 Introduction

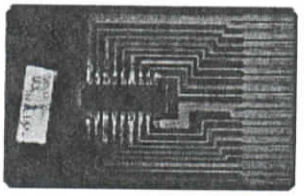
Durant ces vingt dernières années, l'informatique a pris une place prépondérante dans notre vie de tous les jours. Elle est utilisée partout : pour la gestion des capteurs dans nos voitures, pour aider les chirurgiens à opérer, pour les télécommunications, pour retirer de l'argent d'un guichet automatique avec une carte à puce, etc. Cette formidable évolution fut possible grâce au développement des technologies (aux progrès fulgurants de l'électronique) qui a permis, entre autre, la réduction de la taille des éléments de stockage d'information dans les composants. Cette miniaturisation a permis d'augmenter la taille des mémoires et donc de passer d'une programmation proche de la machine (assembleur) à une programmation haut niveau avec, par exemple, des langages orientés objet (Java, C++).

Ce phénomène est en train d'apparaître dans le domaine très fermé des cartes à puces intelligentes (les smart cards), très populaires en Europe et au Japon ,mais encore peu développées aux Etats-Unis, où l'on préfère utiliser les cartes magnétiques. Mais tout cela est en train de changer grâce à l'apparition de nouvelles puces plus puissantes permettant d'installer plusieurs applications sur le même composant. Par exemple, elles pourront contenir des données aussi disparates que de la monnaie électronique, des crédits téléphoniques, des informations médicales utiles en cas d'accident, etc....

Il y a un intérêt grandissant pour les cartes à puce (la demande augmente de 40% par an). En effet, ses possibilités d'authentification et de stockage d'information permettent des accès sécurisés à l'Internet. Gemplus (leader mondial des cartes à puce) estime qu'il en circulait environ 3 milliards à la fin de 2001.

1.2 Naissance de la carte à circuit intégré

Tout commence en 1974 quand Roland Moreno, dirigeant alors la société Innovatron, met au point la carte à circuit intégré. En fait, son projet n'était pas celui que tout le monde a en tête quand on nous parle de carte à puce, mais plutôt celui d'un porte-monnaie électronique logé sur une bague.



De là naît peu à peu le projet "Electronic Purse" qui revient avec la nouvelle technologie du "sans-contact". Mais l'idée principale est retenue : c'est la création d'une carte à mémoire. Les premières à être créées arrivent en septembre 1974 .

Un autre fait marquant de l'évolution de la carte à puce survient en 1977 avec la création de Bull CP8. Au sein de sa direction technique, Michel Ugon s'intéresse sérieusement à la

carte à mémoire. Ainsi en 1979, parait la première carte à microprocesseur (Bull CP8 : voir ci-contre).

Toujours la même année, Schlumberger entre dans le capital d'Innovatron et commence alors ses recherches sur la carte à mémoire : la division "Cartes à Mémoires & Systèmes" est créée au sein même de l'entreprise Schlumberger.

En 1980, le Groupement Carte à Mémoire est créé par un regroupement de banques françaises afin d'utiliser la carte à puce comme un nouveau moyen de paiement : ce qui deviendra la future carte bancaire. Bull, Schlumberger et Philips se lancent alors dans la conception de ces cartes, et c'est réellement en 1984 que la commercialisation des premières cartes bancaires à mémoire a lieu.

En 1983, on voit dans la carte à puce, une aide dans le secteur sanitaire et social, dans ce qui touche à la santé : il s'agit de doter les patients d'une carte santé qui renseignerait de manière plus précise le docteur à propos du malade.

La même année, la Direction générale des Télécommunications (futur France Télécom) présente la "carte télécommunication" : une carte d'abonnement qui a pour objectif de prélever chaque communication sur la facture téléphonique de l'abonné.
Mais c'est en 1984 qu'on note l'apparition de la télécarte avec la "carte pyjama" (appelée ainsi en raison de ses rayures blanches et bleues) créée par Schlumberger pour France Télécom (cartes équipées de micromodules).
En 1985, Bull livre ses premières cartes bancaires "CB" dotées de microprocesseurs.

Ce sont ces cartes qui nous viennent à l'esprit quand on parle de "carte à puce", car effectivement, ce sont celles qui vont connaître la plus grosse production. Peu à peu, les publiphones sont remplacés pars des cabines utilisant les cartes à puce. Les cabines ainsi que les cartes connaissent alors une croissance fulgurante : de 2.000.000 de cartes vendues par an en 1986, on atteint vite 6.000.000 d'exemplaires par mois en 1991. La carte à puce connaît alors un franc succès qui sort d'Europe pour atteindre le niveau mondial.

1.3 La carte à puce

Dans les sociétés occidentales, il est devenu presque impensable de faire ses courses sans utiliser de cartes de crédit. Celles-ci sont devenues indispensables aux économies modernes. Mais aussi familières que sont devenues ces cartes en plastique, elles sont en train d'être rattrapées par quelque chose qui possède nettement plus de puissance et de flexibilité: la carte à puce ou carte intelligente

Une carte à puce est une carte plastique similaire à une carte de crédit, de mêmes forme et dimension, mais avec une différence fondamentale: une carte à puce possède un ordinateur intégré. Ce microprocesseur est capable d'effectuer les tâches des ordinateurs traditionnels comme exécuter des programmes, traiter les entrées-sorties et sauvegarder des données.

Cependant, contrairement à un ordinateur classique, une telle carte ne dispose ni d'écran ni de clavier. En fait, elle ne dispose même pas d'une source d'alimentation.

Parce qu'elle a besoin d'obtenir des données et d'afficher des résultats, une carte intelligente travaille toujours en tandem avec un CAD (Card Acceptance Device = Périphérique d'Acceptation de Cartes), c'est à dire un lecteur de cartes ou bien un terminal. Chacun de ces deux périphériques possède un slot pour y introduire la carte. Les lecteurs de cartes sont connectés à des ordinateurs tandis que les terminaux sont eux-mêmes des ordinateurs.

Cette carte peut-être programmée et de ce fait, elle ouvre la voie à des applications impossibles à réaliser avec la carte magnétique traditionnelle. Par exemple, elle peut être employée pour "les tarifications forfaitaires" .Une application en vogue en ce moment est le paiement des droits de passages sur les autoroutes américaines à péages: on achète une carte prépayée et à chaque passage, le scanner (lecteur de carte) réduit la valeur des crédits disponibles sur la carte. Lorsque tous les crédits ont été épuisés, il suffit de recharger la carte. C'est le même principe qui est utilisé dans le téléphone mobile prépayé.

Une autre application possible serait le stockage des données personnelles, par exemple les bilans de santé .A chaque visite médicale, le personnel médical met à jour les données relatives à votre état de santé: groupe sanguin, allergies, etc.. Cela pourrait être particulièrement utile dans le cas d'un accident où la victime serait incapable de communiquer ces renseignements aux secouristes.

Les cas présentés ne sont qu'un infime échantillon des applications des cartes à puce. Non seulement peuvent-elles être programmées pour différentes applications, mais mieux encore, il est possible de charger sur la carte plus d'une application (applet). Si ces applications sont mises au point en utilisant la technologie JavaCard, les données d'une application sont protégés des autres applications. Dans la terminologie Java, on dit que les applets évoluent dans leur "propre sablier".

En plus, elles peuvent être reprogrammées. Ceci représente un avantage considérable pour les fournisseurs de services à base de ces cartes. Par exemple, supposons qu'une banque ait mis en service 500.000 cartes de Débit/Crédit. Si elle a besoin de mettre à jour le(s) programme(s) de celles-ci, il est inutile de demander à les récupérer à cette fin. Il suffit juste de rendre le programme disponible sur l'ordinateur connecté au lecteur de cartes. A chaque fois qu'une carte sera introduite par l'utilisateur, la mise à jour est automatiquement chargée sur la carte.

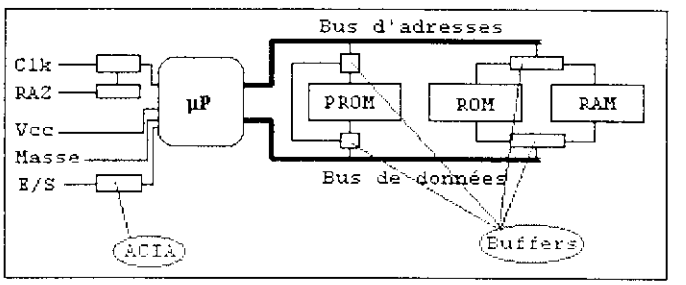
Les Smart Cards les plus connues actuellement sont au format Carte Bancaire. Cependant, il est à noter que certains fabricants proposent d'autres types. On citera par exemple le "Ibuttons" de Dallas Semiconductor ,qui est une smart card implantée dans une bague, communiquant avec le lecteur par radio-fréquences. De même, Sun a récemment introduit le "JavaRing" (anneau Java).

1.4 Types de cartes à puce :

Il y a deux sortes de cartes à puce :

- la carte à mémoire servant seulement au stockage d'informations utilisant la logique pour accéder à l'information (ex : une carte de téléphone avec 50 crédits) .

- la carte à puce intelligente (celle qui fait l'objet de notre étude) qui est un micro-ordinateur. Elle contient un micro-contrôleur 8 bits (les plus connus étant les composants 6805 de Motorola et le 8051 d'Intel) qui a des propriétés de sécurité. Sa mémoire est composée de RAM où sont stockées les données temporaires (i.e. ces données sont perdues lors de la perte de courant), d'EEPROM où sont stockées les données sur le possesseur de la carte (ex : le numéro de compte bancaire, la clé privée de cryptage, etc.) et de la ROM où résident les programmes de la carte.



Structure d'une carte à puce

Les cartes à puce sont bien sûr de plus en plus évoluées. Avec les progrès de la miniaturisation, on peut également stocker de plus en plus d'informations dans leurs mémoires, qui peuvent aller jusqu'à 256 Ko. Notre projet consistera donc à étudier les cartes à puce en général, les cartes à puce Java (JavaCards) ainsi que les différents outils de développement d'applications à base de cartes à puce. On fera l'analyse d'un problème actuel (porte monnaie électronique) que nous essayerons de résoudre . Pour cela, nous utiliserons le kit JCOP(JavaCard Open Platform) d'IBM.

1.5 Normes

1. Contraintes physiques ISO 7816
   Surface <= 25 mm^2 Epaisseur <= 0,3 mm.

- Contraintes mécaniques :

La surface de la carte ainsi que les composants intégrés dans la carte doivent demeurer intacts durant l'utilisation et le stockage .
La surface (et les contacts) doivent supporter sans dommage la pression produite par une balle en acier de 1.5 mm de diamètre sur laquelle est appliquée une force de 1.5 N.

- Résistance électrique:

La résistance mesurée entre deux contacts quelconques ne doit pas être supérieure à 0.5 Ω, pour une valeur de courant comprise entre 50 µA à 300 mA.

- Champ magnétique:

La puce doit supporter sans dommage un champ magnétique statique d'intensité 79500 A.tr/m

- Electricité statique:

La carte doit supporter la décharge d'un condensateur de capacité 100 pF porté à un potentiel de 1500 V à travers une résistance de 1.5 K Ω.

- Torsion maximale de la carte:

Sur la longueur - déformation : 2 cm - périodicité : 30 torsions par minute

Sur la largeur - déformation : 1 cm - périodicité : 30 torsions par minute

La carte doit résister au moins à 1000 torsions avant de craquer.

2. 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).

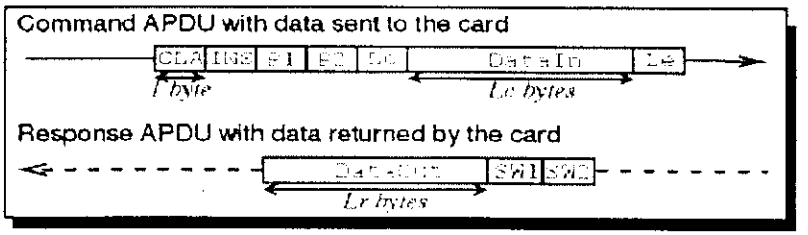
3. Peut être «reprogrammée» par l'écriture de programmes en mémoire non volatile.

4. Eléments de sécurité
   Composant inaccessible

5. Normalisation de commandes
   - ISO 7816-4 : Manipulation des données au travers d'une structure hiérarchique de fichiers
   - ISO 7816-5 : Identification des applications
   - 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

6. Normalisation du format des commandes ISO 7816-4

- Définition du format des «paquets de données» échangés entre un lecteur et une carte : APDUs (Application Protocol Data Units) de commande et de réponse.



Format de commande et de réponse

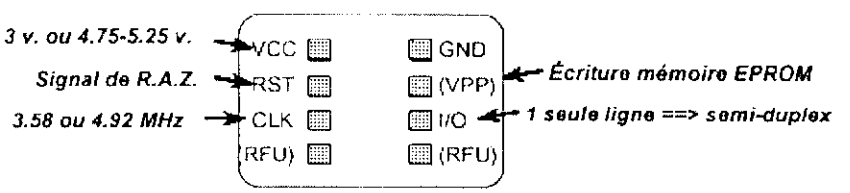
- Mots d'état SW1 et SW2 standards (OK = **0x9000**)

6. Normalisation de la communication ISO7816-3 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
   - Mode maître-esclave : la carte répond à des commandes
     T=0 : transmission de caractères (le plus utilisé)
     T=1 : transmission de blocs de caractères

7. 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



Taille minimale des contacts : 1.7 mm * 2 mm

Position des contacts:

C1 : Vcc = 5V	C5 : Gnd
C2 : Reset	C6 : Vpp
C3 : Clock	C7 : I/O
C4 : RFU	C8 : RFU

1.6 Cycle de vie de la carte

1. 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.

2. Initialisation:
   Inscription en EEPROM des données communes à l'application

3. Personnalisation:
   Inscription en EEPROM des données relatives à chaque porteur.

4. Utilisation:
   Envoi de commandes à la carte et 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

5. Mort
   Par saturation de la mémoire, bris, perte, vol, etc....

1.7 Applications

En dehors de son aspect le plus répandu à travers la télécarte à puce, le système de la puce elle-même est la source de nombreuses autres applications. Que ce soit dans les télécommunications, dans les transports, dans le visuel, dans les finances, dans le marketing, la puce électronique devient vite une technologie indispensable.
Intéressons-nous à différents domaines :

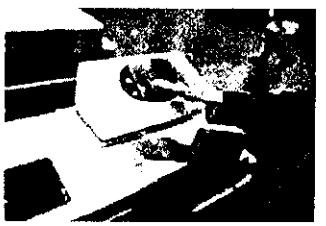
- **Les télécommunications :**

C'est un des secteurs dans lequel la puce est la plus répandue. En effet, tout ce domaine est centré autour de la téléphonie. Celle-ci peut être d'ordre privé avec l'utilisation des GSM :

Global System for Mobile Communications (téléphones portables), avec des cartes comme la carte Pastel (France Télécom) qui permet à l'utilisateur d'être débité en différé sur son compte mais de pouvoir utiliser le réseau téléphonique publique (cabines...).

Elle peut également être d'ordre publique à travers l'utilisation de la télécarte dans des cabines extérieures ou Publiphones.

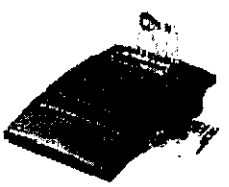
- **Les transports :**



Dès que vous vous déplacez, la puce devient inévitable : à travers les transports urbains, en commun ou bien même au sein de votre véhicule.

Par exemple, la **Transcarte** n'est autre qu'une carte à puce correspondant à un titre de transport :Attribuée à la rentrée scolaire, ce titre donne droit à 1 Aller/Retour gratuit par jour selon les modalités du Conseil Général.

- **La santé :**



Dans ce domaine là, la carte à puce commence à bien se développer et notamment à travers la **carte Sesame** ou la **biocarte** qui laissent croire à une ouverture du marché dans ce secteur. Grâce à cette carte, le médecin peut accéder à des renseignements très précis sur son patient mais surtout de manière très rapide. On y voit une grande facilité d'utilisation et un contrôle de la sécurité des données.

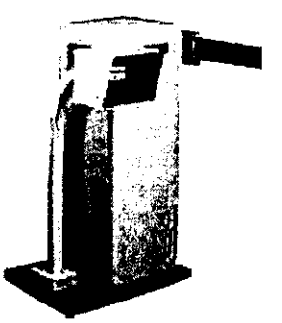
- **Les banques :**

Naturellement, quand vous retirez de l'argent, c'est votre carte bleue qui vous le permet, et si jamais vous ne faisiez pas le rapprochement avec la carte à puce, alors veuillez vous reporter aux sections précédentes !

En effet, la carte à mémoire sert ici de support pour flux monétaire .

- **Les contrôles d'accès logique ou physique :**

Les contrôles d'accès logique sont des garanties de sécurité au niveau informatique, c'est à dire qu'ils contrôlent des données. Les contrôles d'accès physique contrôlent, eux, la sécurité de lieux gardés.
Ce sont les applications auxquelles nous sommes régulièrement confrontés dans la vie courante : parkings, immeubles...



Chapitre 2
Le Langage de Programmation Java

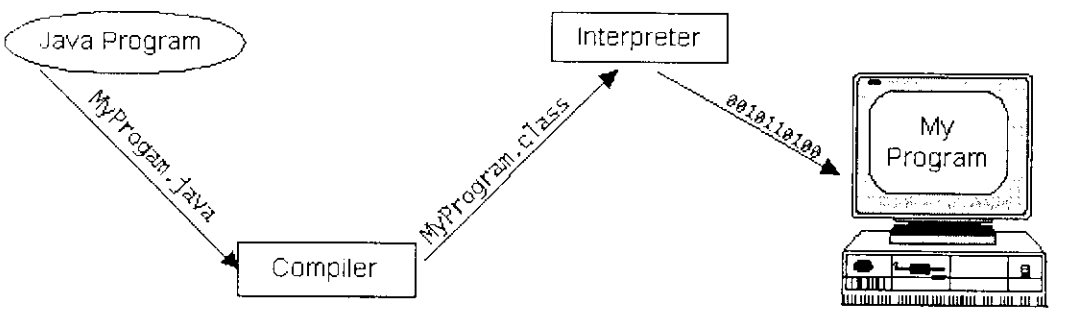
2.1 Qu'est ce que Java?

Java est plus qu'un simple langage de programmation ,c'est aussi une plate-forme .
Java est un langage de programmation haut niveau à la fois:

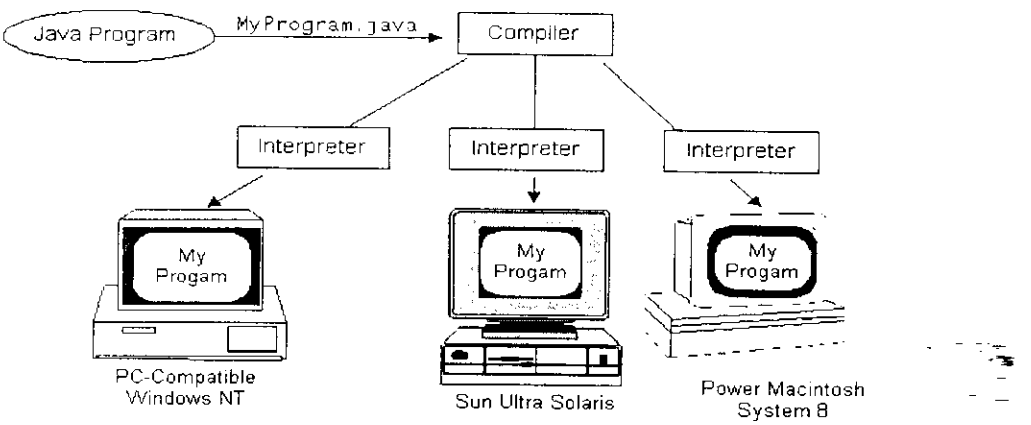
- Simple
- Orienté Objet
- Portable
- Cross plate-forme.
- Multitâche
- Dynamique
- Sécurisé

Le mode de fonctionnement des programmes écrits en Java est aussi inhabituel : ils sont à la fois compilés et interprétés . Avec un compilateur , les programmes sources sont transcrits en un langage intermédiaire appelé Java Bytecodes . Ce langage est plate-forme indépendant .Ces codes binaires sont ensuite interprétés par l'interpréteur Java, qui exécute chaque octet sur la machine. La compilation n'a lieu qu'une seule fois.

L'interprétation a lieu à chaque exécution du programme. La figure ci-dessous illustre le principe de fonctionnement d'un programme Java.[16]



L'utilisation des Bytecodes permet de rendre possible le WORA "Write Once Run Anywhere".Les programmes sont compilés en Code Octets puis interprétés par toute implémentation de la machine Virtuelle Java (JAVA VM). Un programme écrit pour un P.C. pourra être exécuté sur un PC, Mac, Station Sun Solaris.

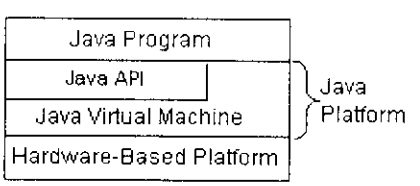


2.2 Plate-forme Java:

Une plate-forme est la partie matérielle ou logicielle qui permet l'exécution de programmes . La plate-forme Java diffère de toutes les autres plates-formes car elle est uniquement logicielle et s'exécute au-dessus de toutes les autres plates-formes matérielles. Les autres plates-formes sont des combinaisons de matériel et de système d'exploitation.

La plate-forme java se compose de deux parties:

- la machine virtuelle Java



- interface de programmation d'application (Java API)

L'API java est une collection de composants prêts à l'emploi qui offrent beaucoup d'avantages tels que l'interface graphique. Ils sont regroupés en librairies. [18]

2.3 Avantages de Java:

La programmation en java fournit certains avantages: meilleurs programmes et efforts de programmation réduits .De plus, elle permet:

- **Un démarrage rapide:** les programmeurs objet, déjà habitués au langage C++, trouveront de grandes similitudes entre les deux langages.
- **Code réduit :** la comparaison entre les programmes Java et C++ démontrent que pour une même tâche ,les programmes java sont quatre fois moins volumineux que ceux écrits en C++.
- **Développement plus rapide**: Le processus de développement peut être réduit de moitié comparé au temps nécessaire pour développer les programmes en C++, car le nombre de lignes de code est nettement inférieur.
- **Eviter la dépendance des plates-formes :** Contrairement aux programmes écrits en C++, ceux écrits en pur Java sont portables à 100%. Cette caractéristique est certainement la plus importante de toutes, et c'est elle qui a fait de Java le langage de l'Internet ! !

2.4 Java, une alternative...

Pour comprendre la particularité de Java, il faut commencer par présenter un historique de ce nouveau langage.

A ses débuts, Java a été présenté aux industriels de la part de la société Sun MicroSystems comme une tentative d'alternative à Windows de Microsoft. En effet, ces dernières quelques années, Microsoft avait acquis et conserve toujours une place prépondérante sur le marché de l'informatique et surtout sur celui de la Bureautique. Avec Intel, le fabricant de microprocesseurs, ils disposent d'un pouvoir quasi omnipotent lorsqu'il s'agit de monter ou de démonter des normes. Ils dictent le futur de l'informatique à l'échelle mondiale. Dans ce contexte, l'idée de Sun fut de créer une alternative. C'était le moyen de donner une « couleur » à ce langage dans un marché nouveau alors qu'il avait subi quelques soubresauts dans sa genèse et qu'il ne s'est retrouvé propulsé là que par un concours de circonstances.

Dans ce contexte, ils se devaient, pour réussir, d'être adaptables à tous les systèmes afin de ne plus imposer une nouveauté axée sur le matériel mais bien de créer un système qui soit « pluriel », comme on dit aujourd'hui. Il fallait que ce soit les systèmes logiciels qui adaptent les applications (ce que l'on va appeler la « machine virtuelle Java ») pour que tout le monde informatique « non PC », puisse adhérer au projet. De plus, il fallait à tout prix éviter de se retrouver directement en concurrence sur le marché de prédilection de Microsoft et ce, en créant une nouvelle alternative et ne plus faire du PC, comme la tendance le voulait, un passage obligé.

Avec une démarche commerciale performante et nouvelle, les gens de Sun ont su trouver leur place et mettre au point un outil intéressant en tirant profit des circonstances

et des marchés. Seulement, le projet seul n'est pas suffisant : Apple avec le Macintosh est bien placé pour le savoir. Il était nécessaire que des éditeurs les suivent dans leurs choix.

L'importance du projet, ses perspectives de développement et ses possibilités l'ont imposé à tous. Ainsi, Netscape, le premier puis Oracle, Borland, IBM, Adobe, Symantec, Novell et même Microsoft (business oblige), ont signé des partenariats afin d'obtenir des licences d'utilisation et d'intégration de la technologie Java dans leurs machines et logiciels. Sun MicroSystems avait assuré le développement de Java. Entre Mai 1995 et Mars 1996, ils avaient réussi à persuader le monde de l'informatique de la nécessité et de l'importance de leur projet. Ils s'étaient associés avec les plus grands éditeurs mondiaux et avaient créé un avenir à cette nouvelle technologie .

2.5 L'histoire

C'est en 1990 que commence le développement de Java à Mountain View en Californie, dans les laboratoires de Sun MicroSystems. A cette époque, l'équipe de James Gosling se consacre à l'écriture d'un nouveau langage de programmation, destiné plus particulièrement au pilotage des appareils électroménagers.

Le groupe avait un objectif assez ambitieux : il s'agissait de mettre au point un langage qui s'appliquerait de la même façon à tous les dispositifs de commande électronique actuels et futurs ! Le projet portait le nom de Oak, « Chêne » en anglais. La finalité était d'éliminer la compilation spécifique à chaque système que réclame tous autres langages. C'est à dire de supprimer cette étape de l'utilisation d'un programme qui consiste à transformer un code informatique du « langage évolué » au « langage machine » propre à chaque système.

Au bout d'un certain temps, le projet changea de nom et d'orientation. Compte-tenu de l'évolution fulgurante des techniques de communication de ces dernières années, on compléta les spécifications du langage. Il se plaça alors dans un cadre beaucoup plus large. Le projet entamait un nouvelle vie sous le nom de Java. En argot américain, « Java », c'est la pause café, celle que s'accordèrent les chercheurs ne parvenant pas à trouver un nom convenant au projet.

Le domaine d'application du projet avait donc évolué. Il s'agissait maintenant de s'attaquer au marché de l'électronique grand public. C'est à dire tous les appareils susceptibles de contenir une commande électronique. Java devait permettre « d'insuffler la vie aux téléviseurs, téléphones, magnétoscopes, cafetières et autres appareils courants ». La première des préoccupations était dans le comportement demandé à l'utilisateur face à la machine. Il ne s'agissait pas seulement de mettre en œuvre des algorithmes de pilotage et de régulation prédéfinis, mais bien de poser les nouvelles bases de l'interaction homme/machine. L'équipe réussit ainsi à monter un prototype, le Star Seven (*7). Le *7 était une machine programmable dans un langage « presque Java » qui se commandait au travers d'une interface graphique: l'utilisateur développait, aidé par de petites figurines animées, en particulier la mascotte actuelle de Java: Tumbling Duke.

Le concept Java était lancé et pouvait encore être étendu. Un mot nouveau apparut VOD (Video On Demand), et les développeurs y virent très vite un autre domaine de débouchés pour Java. La VOD apparaissait comme une innovation majeure pour les temps à venir et Sun s'employa à faire percer les développements dans ce domaine.

L'idée de base était claire et novatrice: Java devait rendre possible l'interaction avec les services de vidéo sur demande. Les programmes pouvaient être transmis sans problème au travers du réseau câblé de télévision. On remarquait ainsi que la diversité des plates-formes en service ne posaient pas de problème à Java. Mais ce fut un « flop ».

2.6 Le développement grâce à l'Internet :

La carrière de Java ne connaissant guère de succès, Sun se mit en quête d'un nouveau domaine d'application au printemps 1994. C'est à ce même moment que le réseau Internet, jusque là réservé aux Universités et aux Centres de Recherche, entra dans sa grande phase de développement vis à vis du grand public. La baisse des prix des matériels et la diffusion croissante des ordinateurs personnels firent véritablement exploser l'Internet. Le réseau devint le thème de prédilection des télécommunications. Sun reconnut très vite les capacités de son projet à fonctionner dans des réseaux hétérogènes de type Internet.

La structure du réseau jouait un rôle de premier plan. En effet, Internet se résume finalement à la mise en relation d'un grand nombre de réseaux décentralisés dont les voies permettent l'échange global d'informations. La notion de village planétaire était née, avec, pour la première fois, la vision d'un système de communication mondial. L'éloignement géographique n'était plus une barrière pour les communications. Se lancer dans une conversation avec un partenaire à l'autre bout de la Terre n'était pas plus compliqué que de transférer un fichier vers une autre ville.

Mais les capacités techniques d'Internet furent rapidement dépassées en raison de son acceptation croissante comme moyen universel de communication. Le réseau manqua de s'effondrer sous les intérêts économiques. Les taux de transfert médiocres et l'affichage de graphisme mettaient les nerfs à rude épreuve et la volonté de rendre malgré tout l'Internet dynamique demandait de nouvelles solutions. De plus, l'interaction avec l'utilisateur était quasiment impossible.

Java affichait déjà des qualités capables de combler les lacunes du réseau. Ainsi ces qualités pouvaient sûrement servir à développer des applications dédiées au réseau. [10]

2.7 L'indépendance vis à vis de la plate-forme...

La neutralité en matière de plate-forme des projets Java ouvrait de nouvelles perspectives: un seul code pour une application qui tourne sur n'importe quel système. Car toutes les applications écrites en Java sont théoriquement opérationnelles sur n'importe quel système, sans compilation spécifique. Dès lors, Sun MicroSystems réoriente son action vers les réseaux, et commence à développer des applications en Java.

Dès Mai 1995, la société présente publiquement à San Francisco la version Bêta du programme de navigation HotJAVA, entièrement développée en Java.

Le succès est considérable. Les éditeurs signent des licences avec Sun et en particulier Netscape. Celui-ci intègre dans la version 3 de son célèbre navigateur Internet (Browser en anglais), Navigator les attributs (« plug-in ») Java qui rendent compatible la visualisation d'applications au travers des pages Internet. Peu à peu, on appelle ces programmes intégrés au code de description (code en langage Html) des pages WWW des applets. Ce sont des programmes qui se lancent sur les machines de visualisation.

Le mécanisme est simple, la source code Html et Java est chargé sur le serveur. A partir du browser et du plug-in Java, le code est interprété sur le client sans qu'il soit nécessaire au serveur d'intervenir sauf si une nouvelle requête est faite. Ainsi, quelle que soit la machine, le même code source est chargé pour tous les clients quelque soit leur nature. C'est après l'interpréteur, dans ce cas le browser, qui se charge de transformer ce code source en un résultat alliant tous les moyens de communication moderne: textes, images, animations, sons...

La position dominante sur le marché de Netscape à cette période permet très rapidement de convaincre les autres éditeurs : la voie du succès est tracée pour Java. [12]

2.8 Une machine « virtuelle »

Java est alors présenté comme un langage évolué dont la syntaxe est proche de celle de C++ avec qui il présente un certain nombre de similitudes.
Java est avant tout un langage interprété, c'est à dire que l'on ne dispose pas d'un code machine directement. Le code source est d'abord compilé en pseudo-code, qui, bien que différent du langage machine dans la syntaxe, s'en rapproche dans la structure. Ceci permet une certaine rapidité dans l'exécution par la « machine virtuelle ». Ce pseudo-code compilé est ensuite « interprété » par ce que l'on appelle un « interpréteur Java ».

Ce dernier peut prendre plusieurs formes: browsers comme Navigator ou HotJAVA, JDK ou encore système d'exploitation intégrant Java comme JavaOS de Sun. C'est là que s'explique la « portabilité » de Java. En effet, quelque soit la couche matérielle, la couche logicielle qui contient l'interpréteur Java s'accapare les ressources du système pour les gérer toujours de la même façon. Ainsi, quelle que soit la machine si l'on se place au niveau du pseudo-code Java, on verra toujours la même chose: la fameuse « machine virtuelle ». C'est lui qui va s'occuper de gérer par exemple la mémoire nécessaire au programme mais qui, aussi, la libérera quand un objet sera détruit.
On ne se retrouve plus dans un environnement fixé par la partie« hard » du système mais bien dans un environnement « soft ».

On comprend alors aisément comment Java permet de se libérer des différentes plates-formes. A l'heure actuelle, une nouvelle solution est envisagée pour rendre les machines compatibles Java à un niveau inférieur et ne plus avoir recours à un interpréteur. En effet, Sun ferait des recherches qui consisteraient à faire contenir la « machine virtuelle Java », non plus dans un fonctionnement logiciel, mais directement dans la couche matérielle à

travers une puce qui pourrait être intégrée dans les ordinateurs de tous « formats ». Avec cette solution, l'interpréteur Java serait intégré au hard et les possibilités offertes par la technologie Java seraient encore accrues.

Ainsi, un nouveau projet se développe chez les grands éditeurs : les **Network Computers**(NC).

2.9 Les futurs NC...

Si cette solution venait à exister, elle pourrait révolutionner le monde l'informatique. Ce projet consiste à dire qu'aujourd'hui, la majorité des machines n'exploitent pas la totalité de leur puissance matérielle et que cela engendre des dépenses non nécessaires, en particulier dans les grandes entreprises où l'on se sert de micro-ordinateurs plutôt comme de machines à écrire. Dès lors, on a imaginé un projet qui, à partir d'un gros serveur « distribuerait » des données à des machines qui ne contiendraient qu'un écran, un clavier, une RAM, et un processeur, les fameux NC. Les programmes et toutes les données seraient stockés sur le serveur et on accéderait au serveur par cette machine. On chargerait uniquement le programme avec lequel on travaille que ce soit un traitement de texte, un tableur, un compilateur, un browser...

Plusieurs avantages à ce système: moins de maintenance lourde des réseaux, moins d'incidents sur le réseau, moins de dépenses matérielles. Les machines connectées ne sont plus que des clients simples sans unité de stockage. Ainsi, les utilisateurs ne se servent que de ce dont ils ont besoin. L'utilisation du réseau est optimisée !

Dans ce contexte, Java trouve tout à fait sa place. Si le processeur intégré au NC est compatible Java, plus besoin d'autres systèmes d'exploitation sur la machine et fini les multiples licences. On se contente de licences réseaux qui permettent à tous d'utiliser un logiciel sur la machine virtuelle Java et de ne pas prendre les ressources du serveur. Il charge sur le NC les programmes utiles, le NC est autonome jusqu'à la fin de la session, puis le serveur récupère le travail de la session. C'est la fin du règne du Personnal Computer dans les entreprises.

2.10 Java et Internet :

Internet est l'évolution du premier réseau américain ARPA (Advanced Research Project Agency), fondé en 1969. Son but était au départ de disposer d'un moyen de communication opérationnel en cas d'attaque nucléaire et plus généralement d'accélérer les échanges de données nationales. Ce réseau reliait plusieurs ordinateurs au travers les Etats Unis. Le chemin que devait emprunter une information pour aller entre deux points n'était pas prédéterminé mais dépendait de la charge et de l'activité des différents noeuds. Il utilisait des programmes appelés « routeurs » dont la fonction était de rechercher le chemin le plus rapide à travers un réseau.

ARPAnet est resté homogène pendant quelques années, puis il s'est ouvert à d'autres réseaux, c'était la naissance de l'Internet. Ouvert prioritairement aux Universités et aux Centres de Recherche, la structure du réseau a alors subi des modifications radicales. Pour

permettre la communication se met alors en place un protocole commun HTTP (Hyper Text Transfert Protocol), il est toujours en vigueur aujourd'hui.

En 1993, Internet connaît un essor énorme. Le grand public s'intéresse à cet outil. Les chercheurs du CERN en Suisse, travaillent sur un nouveau langage pour l'Internet afin de publier leurs résultats. C'est à partir de leur technologie, le langage HTML (Hyper Text Makeup Language), que l'on trouve le fondement du World Wide Web, c'est à dire la partie graphique et maintenant multimédia de l'Internet, celle qui est ouverte au grand public. La norme HTML toujours en vigueur permet de crypter des mises en pages à l'aide d'un simple fichier ASCII. On peut en outre y intégrer des images, du sons et même des séquences vidéos. On peut aussi y inclure des liens vers d'autres sites du WWW ce qui a donné naissance à l'expression « surfer sur le Web », puisque l'on passe d'un site à l'autre par un simple click de souris.

Maintenant, l'Internet est devenu un lieu d'échange, grâce en particulier à la baisse du coût du matériel, et les entreprises y voient maintenant un enjeu économique. Java y joue un rôle important. En effet, son indépendance vis à vis des plates-formes lui permet d'être lu partout sur le réseau, et en intégrant un appel dans le code Html, le browser disposant du « plug-in » Java va interpréter les données de l'applet comme une image ou une séquence vidéo classique et cela dans une zone de l'écran que l'on aura définie. Ainsi, on obtient une véritable façon d'interagir entre le client et le serveur puisque une applet va fonctionner comme un programme en étant toutefois enfermée dans une page Html. De plus, des applets Java n'encombrent pas les réseaux puisque elles sont chargées en même temps que le code Html et sont interprétées en local ensuite par la « machine virtuelle Java» , d'où un gain en rapidité sur le réseau où ne circule que du texte.

Toutefois, on ne peut limiter le domaine d'application de Java, Internet, WWW, les futurs NC. Java est ainsi, la plus récente, la plus vivante et la plus intéressante invention pour rendre le réseau encore plus vivant.[10]

Chapitre 3
Java Card

Le grand nombre de fabricants de cartes à puce offre une grande diversité de produits. Cette diversité se traduit par des processeurs, des systèmes d'exploitation et des langages de programmation différents. Toutes ces différences impliquent que les outils de développement sont spécifiques à chaque carte (ou du moins à chaque fabriquant de carte) et rendent la portabilité d'une application impossible.

Un groupe d'entreprises regroupées autour de Sun ont formé le Java Card Forum. Le but de ce forum est de développer une technologie permettant de créer des applications qui soient indépendantes du matériel utilisé. Cette technologie se nomme Java Card.

3. Qu'est ce que Java Card ?

JavaCard est une technologie permettant de faire fonctionner des applications écrites en langage Java pour des cartes à puce. Elle définit une plate forme sécurisée, portable et multi-application.

3.1 Naissance de Java Card

En mars 1996, un petit groupe d'ingénieurs de Schlumberger au Texas ont souhaité simplifier la programmation des cartes à puce tout en préservant la sécurité des données. Presque immédiatement, ils ont analysé que ce problème avait déjà été posé par le chargement de code sur le World Wide Web. La solution avait été Java. Malheureusement, vu la taille de la mémoire disponible sur une carte à puce, seulement un sous-ensemble de Java pouvait être utilisé (la taille de la JVM (Java Virtual Machine) et du système de runtime ne devant pas dépasser 12 Ko). C'est ainsi qu'est né Java Card, le premier langage à objets adapté aux cartes à puce. Schlumberger et Gemplus sont les co-fondateurs du Java Card Forum, qui recommande des spécifications à JavaSoft (la division de Sun à qui appartient Java Card) en vue d'obtenir une standardisation du langage et qui s'occupe aussi de la promotion des APIs (Application Programming Interface) Java Card pour qu'il devienne la plate-forme standard de développement des applications pour les smart cards.

Java présente trois avantages majeurs pour la programmation des cartes à puce :

1. C'est un langage haut niveau, à objets permettant l'encapsulation des données et la réutilisation de code. Ainsi, les programmeurs n'auront plus besoin d'apprendre les langages assembleur 6805 ou 8051 pour réaliser une application pour une carte. La mise au point de programmes sera plus rapide et les temps de développement seront beaucoup plus courts, d'où l'intérêt économique. De plus, c'est un langage sûr : pas de manipulation de pointeurs, pas d'allocation de mémoire, ... C'est un langage fortement typé, avec des niveaux de contrôle d'accès (public, protected ou private) de ses membres (classes, méthodes, variables).

2. Il est exécutable sur n'importe quel système d'exploitation (Motorola ou Intel) car son code pré-compilé en byte-code est exécuté par la machine virtuelle Java (JVM). Donc cette portabilité permettra d'écrire du code qui fonctionnera sur n'importe quel microprocesseur de cartes à puce (" Write Once, Run Anywhere ").

3. Il possède un modèle de sécurité qui permet à plusieurs applications de coexister en sécurité sur la même carte.

3.2 Présentation de Java Card

C'est un sous-ensemble de Java adapté aux cartes à puce. Sa syntaxe est légèrement moins riche à cause de la faible capacité mémoire d'une puce. La réalisation de ce sous-ensemble implique que Java Card ne supporte pas les tableaux multidimensionnels, les types float, double, long, les chaînes, le chargement dynamique des classes, Security Manager, le Garbage Collector, les Threads, la sérialisation d'objets et le clonage d'objet.

Par contre, ce sous ensemble supporte les types short, byte, boolean, les tableaux à une dimension, les paquetages Java, Classes, Interfaces et exceptions, les caractéristiques orientées objets (héritage, surcharge, etc...). Ses APIs sont basées sur les standards 7816 et ont été adoptées par 95% des constructeurs de cartes à puce. Les applications Java Card sont appelées des applets, identifiées uniquement par leur AID (Application IDentifier). Elles possèdent 5 méthodes : la méthode install() qui permet de créer l'applet, la méthode register() qui permet d'enregistrer l'applet au JCRE (Java Card Runtime Environment), les méthodes select() et deselect() qui permettent la sélection ou la desélection d'une applet et la méthode process() qui traite les commandes. Plusieurs applets pourront cohabiter sur une carte même si elles appartiennent à des fabriquants différents. Les applets d'une carte sont gérées par le JCRE. Les minima requis pour faire tourner un programme Java Card sont 16Ko de ROM, 8Ko d'EEPROM et 256 octets de RAM.

Java Card, bien qu'il soit un sous-ensemble de Java, possède des mécanismes spéciaux liés aux spécificités des cartes à puce. Ainsi, il a 2 types d'objets : les objets transitoires (i.e. stockés en RAM qui perdront leurs valeurs lors du retrait de la carte, d'une coupure de courant, de la desélection de l'applet ou après chaque transaction) et les objets persistants stockés en EEPROM. Il permet aussi de partager des objets entre applets sous certaines conditions. Les lignes de sécurité sont implémentées par la machine virtuelle.

Le gros avantage de Java Card est la possibilité de charger dynamiquement, n'importe quand, une nouvelle applet sur la carte. Cela signifie que le code des applets pourra être mis à jour.

3.3 Présentation du problème

L'usage de Java Card au lieu des assembleurs propriétaires utilisés actuellement pour coder des applications pour les cartes à puce permettra de réduire énormément le temps de réalisation. De plus, il est possible d'utiliser les environnements de développement existants pour Java (par exemple Symantec Visual Café, Microsoft Visual J++, etc.) pour Java Card.

Malheureusement, les erreurs de programmation ne sont pas exclues même avec un langage de haut niveau. C'est pourquoi les développeurs ont besoin d'outils pour les aider à vérifier le comportement de leurs applets Java Card avant leur mise en mémoire dans une carte à puce.

3.4 Architecture

Les cartes à puce représentent la plus petite plate-forme informatique actuellement utilisée. La configuration mémoire peut être de l'ordre de 1K de RAM, 16K de EEPROM et 24 K de ROM. Le plus grand défi de la technologie Java Card est de faire tenir le système Java sur la carte tout en conservant assez d'espace pour les applications. La solution, comme déjà mentionné à la section 2.2 est de n'implémenter qu'une partie de Java et de la machine virtuelle Java (JVM).

La Java Card Virtual Machine (JCVM) est décomposée en deux parties: une s'exécute sur la carte(on-card) et l'autre en dehors de celle-ci (off-card). Beaucoup d'applications qui ne sont pas destinées à fonctionner en temps réel, par exemple la vérification du code, l'édition des liens ainsi que l'optimisation ,sont à la charge de la JCVM off-card.

En plus de fournir un support du langage Java, Java Card définit un environnement d'exécution qui gère la mémoire, les entrées-sorties ainsi que la sécurité. Cet environnement est défini par la norme ISO7816. Ainsi, pour une application, il suffit de déposer des requêtes à travers cet interface de haut niveau.

En conséquences, la technologie Java Card définit essentiellement une plate-forme qui permet aux applications écrites en Java d'être exécutées sur carte à puce et périphériques à contrainte en mémoire. Grâce à cette décomposition, la charge est répartie entre kit de développement et carte. Trois acteurs participent à cela:

• The Java Card 2.1 Virtual Machine (JCVM) : définit un sous-ensemble du langage Java adéquat aux cartes

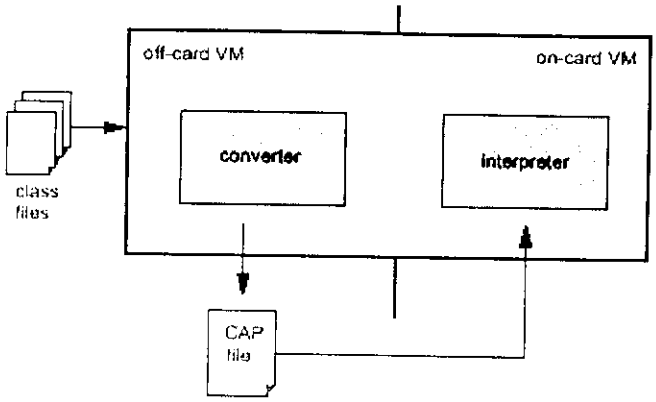
• The Java Card 2.1 Runtime Environment (JCRE) : décrit précisément la gestion de la mémoire, accès aux applets, sécurité.

• The Java Card 2.1 Application Programming Interface (API) :fournit un ensemble de classes et de paquets d'extensions utiles pour les applications.

A cause de la contrainte mémoire, la plate-forme Java Card ne présente qu'un échantillon bien adapté à ce support, soigneusement puisé dans le langage Java , tout en conservant les possibilités de programmation orientée objet.

3.4.1 Java Card Virtual Machine

La différence la plus importante entre la JVM et la JCVM est que la JCVM est implémentée en deux parties: une partie on-card et une autre off-card. La partie on-card contient l'interpréteur de Bytecodes, tandis que le convertisseur tourne sur un Pc ou station de travail. Ce dernier charge les fichiers .class et les convertit en fichier .CAP (converted applet).C'est ce fichier CAP qui est chargé et interprété sur la carte.



3.4.2 Fichier CAP et fichier d'exportation

Java Card introduit deux nouveaux formats binaires qui permettent d'étendre les possibilités d'indépendance plate-forme. Le fichier CAP qui contient une représentation binaire exécutable (interprétable) ainsi que le fichier JAR (Java Archive) employé comme contenant du fichier CAP. Ce dernier contient un ensemble de composants, chacun d'eux décrivant un aspect de ce fichier: infos sur la classe, les liens, infos de vérification....Ce fichier est optimisé au maximum pour avoir une taille réduite.

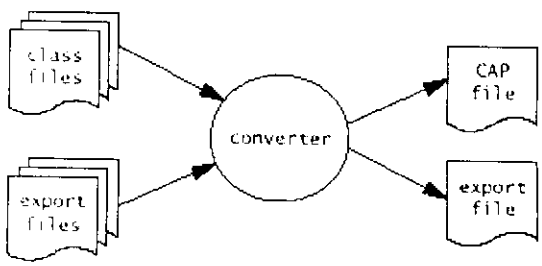
Un fichier d'exportation contient les informations relatives aux APIs. Il définit la portée des variables ainsi que la signature des méthodes. Il contient aussi les informations nécessaires aux référencements interpaquets.

3.4.3 Convertisseur Java Card

Contrairement à la JVM, qui ne traite qu'une seule classe à la fois, l'unité de conversion de ce convertisseur est le paquet. Un compilateur Java transforme les fichiers .java en fichiers .class. C'est ensuite que le convertisseur regroupe les fichiers en un paquet qu'il convertit en fichier CAP et génère les fichiers d'exportation.

Durant cette opération, il effectue:

- L'initialisation des variables statiques.
- L'optimisation des bytecodes
- L'allocation de la mémoire



Convertisseur Java Card

3.4.4 Interpréteur Java Card

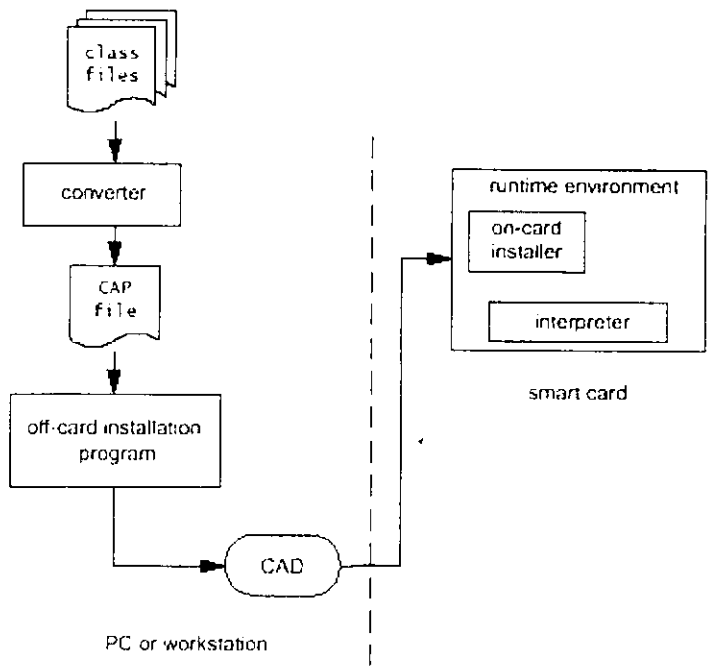
L'interpréteur Java Card a pour missions:

- Exécution des applets
- Contrôle de l'allocation mémoire et de la création d'objets
- Assurer la sécurité

3.4.5 Programme d'installation

L'interpréteur ne charge pas les fichiers CAP en mémoire. Il ne fait que leur interprétation. Les mécanismes pour charger les applets converties sont pris en charge par une unité appelée: l'installeur.

Celui-ci réside sur la carte. Il travaille en coopération avec un programme d'installation off-card. Ce dernier transmet le fichier binaire CAP à l'installeur à travers le CAD. L'installeur met le fichier en mémoire, et génère les liens avec les autres classes.



3.4.6 JCRE

Le JCRE est la partie réalisant le lien entre la machine virtuelle et les ressources physiques. Le JCRE est responsable de la gestion des ressources de la carte, de la communication réseau, de l'exécution et de la sécurité des applets. C'est pourquoi il fait office de système d'exploitation de la carte.

Le JCRE fournit une API standard. En conséquences, les applets sont plus faciles à mettre sur pied et sont transportables sur différentes plates-formes.

L'installeur permet un chargement sécurisé des applets sur la carte, après la fabrication et la mise en service de celle-ci. Bien sûr, il y a coopération avec le programme d'installation.

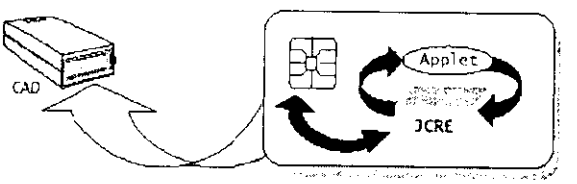
Les applets peuvent être chargées via l'installeur sur la carte après sa fabrication. Cependant , l'installeur est un composant optionnel du JCRE. Sans lui, tous les logiciels devront être écrits sur la carte lors de sa fabrication.

3.4.7 Session CAD

La durée durant laquelle la carte se trouve insérée dans le lecteur est appelée Session CAD. Durant une session CAD, le JCRE assure les communications E/S(commandes et réponses) . Lorsque la commande est reçue, le JCRE l'analyse et:

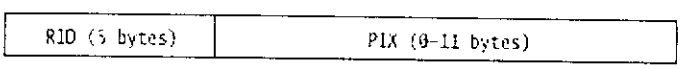
- Soit la fait suivre à l'applet actuellement sélectionnée si elle lui est destinée,
- Soit déselectionne l'applet en cours, charge l'applet visée et lui fait suivre la commande.

A la fin, la carte renvoie une réponse au JCRE qui se charge de le faire suivre à l'hôte.



3.4.8 Identification des applets

Alors que les fichiers et programmes Java sont identifiés par des chaines de caractères, les Java Card applets sont identifiées de manière univoque par un AID (Application IDentifier). La norme ISO7816 spécifie les AIDs à utiliser pour l'identification des applets ainsi que de certains types de fichiers systèmes. l'AID est un vecteur d'octets qui peut être décomposé en deux parties:



Le RID(Resource Identifier) a une longueur fixe de 5 octets tandis que le PIX(Proprietary Identifier Extension) peut aller de 0 à 11 octets. De ce fait, la longueur de l'AID peut aller de 5 à 16 octets. Chaque compagnie possède son propre RID, délivré par l'Organisation Internationale de Normalisation ISO.

3.5 Procédure de développement d'applet

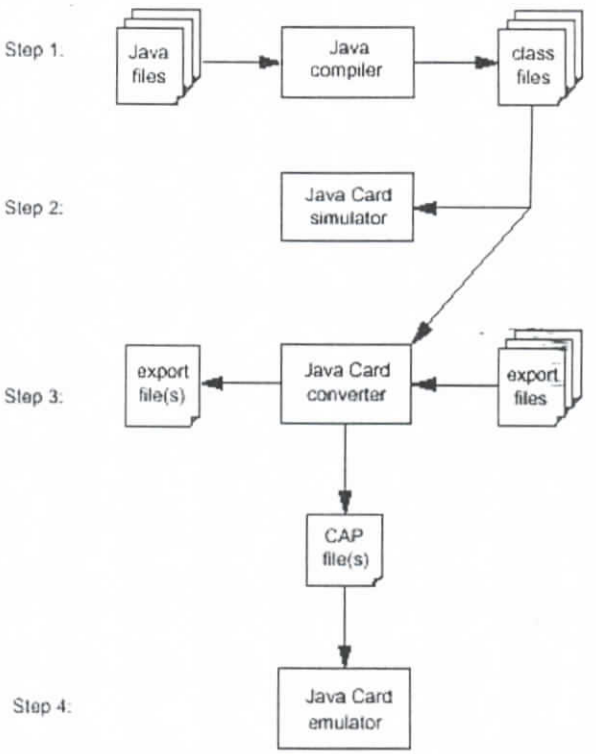
Le développement d'une applet commence comme avec n'importe quel programme Java: le programmeur écrit une ou plusieurs classes et les compile en fichiers .class avec n'importe quel compilateur Java.

Ensuite, l'applet est exécutée et débogguée dans un environnement de simulation, qui reproduit le JCRE sur PC ou station de travail; ceci permet de tester le comportement de l'applet sans la convertir.

Le(s) fichier(s) .class est (sont) converti(s) en fichier CAP via le Java Card Converter.

Enfin, le fichier CAP représentant l'applet est chargé et testé dans un environnement d'émulation, qui simule lui aussi le JCRE. Cependant, il est beaucoup plus puissant que le simulateur, car il comprend une implémentation du JCRE.

L'émulateur permet d'observer le comportement de l'applet comme si elle s'exécutait sur la carte. Il ne s'agit pas seulement de la tester , mais aussi d'apprécier les performances telles la justesse des résultats ou le temps d'exécution.



Finalement, une fois l'applet testée et prête à être transférée sur carte, le ficher CAP qui la représente est transféré via le CAD à la Java Card.

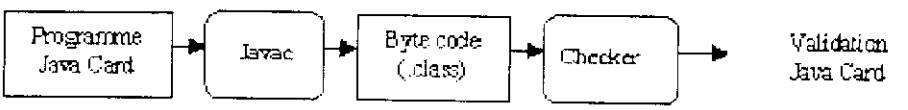
- **Compiler en Java Card**

Toute applet Java Card est une instance d'une classe qui hérite de la classe *javacard.framework.Applet*. Cette classe contient quelques méthodes qui doivent être surchargées durant la conception de l'applet. La création d'une applet Java Card se fait exactement comme une applet Java standard (à l'aide de Java Development Toolkit par exemple), à la différence près que cette applet doit respecter rigoureusement les spécifications de Java Card et non de Java.

Comme les ressources disponibles sur la Smart Card sont très limitées, Java Card ne contient qu'un petit sous ensemble de l'ensemble des fonctions de Java, qui est défini dans les API au niveau du *Java Card Framework*.

Pour compiler des programmes Java, il faut utiliser le compilateur de Java, *javac*, en précisant dans le classpath les librairies spécifiques à Java Card : throwable.jar, jc21api.jar,jcre.jar et jcsimulation.jar . Tout emploi de méthodes, exceptions ou classes n'appartenant pas à Java Card est détecté lors de cette phase de compilation.

Le compilateur *javac*, qui a un analyseur syntaxique adapté à Java et non à Java Card, ne peut garantir que le programme compilé est bien écrit en Java Card car certaines erreurs syntaxiques ne pourront pas être détectées. Pour cette raison, Sun fournit en plus des librairies Java Card, un utilitaire appelé *checker* qui permet de vérifier si les paquetages/applets sont écrits en Java Card et non en Java. Son rôle est de détecter l'usage de types non supportés, d'opérations arithmétiques non autorisées et de classes, méthodes ou exceptions n'appartenant pas à Java Card. La figure 5, ci-dessous montre les étapes conduisant à l'obtention d'un code valide pour Java Card.



L'outil checker peut être utilisé de deux manières, sur un programme Java Card pour tester la validité des applications Java Card et sur des programmes Java pour connaître les modifications à apporter pour le transformer en programme Java Card (le checker indiquera les classes, méthodes et exceptions non supportées).

3.6 Installation d'applets

Quand une Java Card est fabriquée, le JCRE, la JCVM ainsi que l'API sont gravés en ROM Ce procédé d'écriture définitive en mémoire est appelé "Masking" (masquage).Les techniques du masking sont propres à chaque fabricant de cartes et leur étude est en dehors des limites de notre projet.

Les applets de type ROM sont les applets par défaut et sont produites par les fabricants de cartes intelligentes. Parce que leur contenu est contrôlé par le fabriquant, la technologie Java Card permet aux applets ROM de déclarer des méthodes natives, c'est à dire qui sont écrites dans un langage autre que Java, généralement C ou assembleur.

Pour qu'une applet soit exécutable, il faut qu'elle soit connue dans le JCRE. C'est la procédure d'installation qui rend une applet connue du JCRE. L'installation de l'applet se fait généralement à l'endroit de fabrication de la carte ou ultérieurement par un terminal sécurisé. Lors de l'installation, une fois l'applet chargée de façon définitive dans la mémoire persistante de la carte (soit la ROM ou l'EEPROM, dépendant si l'applet est installée à la fabrication de la carte ou ultérieurement), le JCRE fait un appel automatique de la méthode *install ( )*. Cette méthode n'est exécutée qu'à une seule reprise, celle-ci, et plus jamais. Elle doit être déclarée public static parce qu'elle est appelée avant que l'applet ne soit instanciée. Cette méthode doit être surchargée par le programmeur qui doit y mettre les déclarations qui doivent exister durant toute la durée de vie de l'applet. En particulier, une méthode très importante doit être appelée à partir de *install ( )* : c'est la méthode *register ( )*. Elle sert à enregistrer l'applet dans

l'environnement d'exécution Java Card. A partir de cet instant, l'applet sera connue tout le temps par le JCRE.

- **Sélection, activation et désactivation d'une applet**

Une applet sur la Smart Card reste inactive aussi longtemps qu'elle n'est pas explicitement choisie pour être exécutée. La sélection est décidée en fonction de la demande provenant du CAD. Chaque applet est identifiée par un identifiant unique. Lorsque le CAD envoie un message de sélection formé par un APDU contenant l'AID de l'applet à exécuter au JCRE, ce dernier suspend l'exécution de l'applet active en faisant un appel à sa méthode *deselect ( )* pour la rendre inactive. Puis, il active l'applet dont l'AID est indiqué dans l'APDU de sélection en appelant sa méthode *select ( )*. La méthode *select ( )* prépare l'applet à recevoir des APDUs provenant du CAD. Par la suite, le JCRE redirigera tous les paquets entrants vers l'applet active jusqu'à ce qu'une autre demande de sélection soit reçue.

- **Communication avec les applets**

La classe Applet contient une méthode *process ( )* qui représente la seule façon pour une applet de recevoir de l'information en provenance du CAD. L'information est passée à l'aide d'un APDU de sélection ou de commande.

3.7 Contraintes lors de l'installation

Il est indispensable de comprendre que l'installation d'une applet est différente du chargement dynamique de classes, qui lui est assuré par la JVM. L'installation des applets signifie simplement son chargement dans la carte après que celle-ci ait été fabriquée.
C'est pourquoi l'installation des applets présente deux contraintes:

1. Une applet s'exécutant sur la carte ne devrait référencer que les classes déjà présentes sur la carte, puisqu'il n'y a aucun moyen de charger des classes pendant l'exécution.

2. Tout paquet nouvellement chargé ne devra faire référence qu'aux paquets déjà présents sur la carte. Par exemple, pour installer une applet, le paquetage *javacard.framework* dont hérite toute applet devra déjà être présent. De même, il est strictement interdit d'effectuer des références circulaires.

Chapitre 4

API Java Card

Nous avons vu à travers les chapitres précédents qu'une Java Card pouvait exécuter des programmes écrits en Java. Pour cette nouvelle plate-forme, JavaSoft, filiale de Sun MicroSystems, a mis à la disposition des programmeurs l'API JavaCard. La version la plus récente est la version 2.

A travers ce chapitre, nous allons explorer ces APIs plus en détails, ce qui nous permettra par la suite d'écrire notre propre applet.

4.1 Protocole de communication entre le terminal et la carte (applet)

- **La classe APDU**

Le protocole de communication entre la carte à puce et le terminal peut être assimilé au modèle client-serveur où le serveur est la carte à puce et le client l'application cliente (terminal ou host). L'applet située sur la carte à puce attend les requêtes émanant de l'utilisateur. Le dialogue entre la carte et l'application cliente s'effectue avec des trames de données appelées APDU (Application Protocol Data Units) et commence toujours par une trame de type APDU de commande en provenance du terminal. L'applet située sur la carte exécute alors l'action indiquée dans la APDU de commande et répond au terminal avec une trame APDU de réponse. Ainsi, on observe toujours le schéma de communication suivant entre une carte et un terminal: un APDU de commande suivi d'une APDU de réponse. Les tables suivantes illustrent respectivement les formats des APDUs de commande et des APDUs de réponse.

- **APDU de commande**

Mandatory Header (entête obligatoire)				Conditional Body (Corps facultatif)		
CLA	INS	P1	P2	Lc	Data Field	Le

L'entête qualifie la commande et est constituée de quatre champs de 1 octet chacun. Quant au corps, il est constitué de trois champs de 1 octet chacun.
CLA: Octet utilisé pour identifier l'application
INS: Code instruction
P1-P2: Octets paramètres
Lc: Représente le nombre d'octets de la commande
Le: Représente le nombre maximal d'octets attendus dans la réponse à cette commande.

- **APDU de réponse**

Conditional Body (corps facultatif)	Mandatory Trailer	
Data Field	SW1	SW2

Les octets SW1 et SW2 sont des mots d'état indiquant le succès ou l'échec d'une commande.

L'Application Protocol Data Unit (APDU) est défini dans l'ISO 7816-4. C'est une classe du paquetage *javacard.framework*.Elle transfère les données à travers un buffer (APDU buffer) .

En effet avec sa méthode *getBuffer( )*, elle récupère l'APDU du buffer de transfert des données. Avec cet objet ,il est facile de positionner le sens du transfert avec la méthode *setOutgoing*( ) pour envoyer les données ou la méthode *setIncomingandReceive*( ) pour recevoir les données.

Pour envoyer des données, on utilise une des commandes *sendBytes*( ) ou bien *setOutgoingandSend*( ).

La taille des données peut être plus grande que celle du buffer de l'APDU. Dans ce cas, la méthode *sendBytesLong( )* est un tableau a partir duquel les données sont envoyées. Ainsi les données peuvent être divisées en plusieurs blocs qui seront envoyés l'un à la suite de l'autre.

Pour analyser le contenu du buffer, l'interface ISO7816 fournit un ensemble de constantes qui identifient les différents champs:

Constante	Signification	Valeur
OFFSET_CLA	Offset du champ CLA	Offset_CLA=0
OFFSET_INS	Offset du champ INS	OFFSET_INS=1
OFFSET_P1	Offset du champ P1	OFFSET_P1=2
OFFSET_P2	Offset du champ P2	OFFSET_P2=3

Exemple d'utilisation:

```
byte[ ] apdu_buffer=apdu.getbuffer();
byte CLA=apdu_buffer[ISO7816.OFFSET_CLA] ;
```

Pour une bonne communication, il est bon de respecter la norme ISO 7816-4 décrite ci-dessous.

4.2 La Norme ISO 7816-4

Cette norme définit comment il faut positionner les bits des octets pour utiliser les APDUs .

Code	Nom	Taille en octet(s)	Description
CLA	Class	1	Classe d'instruction
INS	Instruction	1	Code instruction
P1	Paramètre 1	1	Paramètre instruction 1
P2	Parametre2	1	Paramètre instruction 2
Lc field	Longueur	Variable 1 à 3	Nombre d'octets présents dans le champ de donnée
Data Field	Donnée	Variable =Lc	Donnée
Le field	Taille	Variable 1 à 3	Nombre maximum d'octets attendus dans le champ de donnée de la réponse.

Ce tableau représente de manière générale la convention de codage des APDU de commande.

4.3 Java Card API

La plupart des cartes à puce sont compatibles avec les normes ISO7816 et EMV. Nous avons déjà expliqué la première et nous allons aborder la seconde. EMV a été introduit par Europay, MasterCard et Visa. Il s'agit d'un standard basé sur la norme ISO7816 et qui fournit des extensions propres à l'industrie financière.

Le squelette JavaCard (JavaCard Framework) est destiné à faciliter la tâche des programmeurs, en proposant une interface de programmation qui encapsule les spécificités des différentes cartes. Cela permet de concentrer les efforts sur le design et l'optimisation du code, plutôt que sur la manière dont les APDU sont construits et transmis.

L'API contient trois paquetages fondamentaux ainsi qu'un paquetage d'extentions. Les trois fondamentaux sont: java.lang, javacard.framework , and javacard.security. Ils sont à la base de toute applet. Le paquetage d'extension est javacardx.

Un programmeur familier avec le langage Java notera que beaucoup de classes de la plate-forme Java ne sont pas présentes dans l'API JavaCard. On citera par à titre illustratif: les interfaces GUI (Graphical User Interface), communication réseau.....La raison étant que les cartes ne disposent pas de périphérique d'affichage et font appel à des protocoles autres que le TCP/IP, UDP et FTP. De même, beaucoup de classes utilitaires ne sont pas présentes et ce pour se plier aux contraintes liées à la mémoire réduite. Cependant, il serait erroné de croire que l'API JavaCard est un sous-ensemble de l'API JAVA au sens strict. En effet, JavaCard possède des classes propres (relatives au standard ISO 7816) qui n'existent pas dans l'API Java.

4.3.1 Paquetage java.lang

Ce paquetage est un sous-paquetage de son équivalent sur la plate-forme Java. Il ne supporte que les classes : Object, ainsi que quelques type d'exceptions.

Pour les classes implémentées, seules quelques méthodes sont fournies. Par exemple, la classe Object ne possède que deux méthodes : un constructeur de classe ainsi que la méthode equal (pour la comparaison de deux objets).Cette classe demeure néanmoins la racine de toute la hiérarchie des classes, de même que la classe Throwable est la racine de toutes les exceptions.

4.3.2 Paquetage javacard.framework

Ce paquetage est essentiel. Il définit la classe mère Applet dont hérite toute application destinée à la carte. Une autre classe tout aussi importante est la classe APDU. Les APDUs sont transmis par le protocole. Il définit aussi les classes : AID, APDU, ISO, PIN et Util

PIN est l'acronyme de Personal Identification Number. Il représente la forme la plus courante de mots de passe utilisés par les cartes à puce pour identifier les porteurs.

public class Util
Méthodes statiques utiles pour performance carte
Copie, comparaison de tableaux de bytes.
Création de short à partir de byte.

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

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

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

public class Applet
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()

- 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 reçu désigne cette applet
Rend l'applet active

public void deselect()
Appelée par le JCRE pour désélectionner l'applet courante

public void process(APDU apdu)
Appelée par le JCRE quand un APDU (de commande ou de sélection) est reçu.
S'il s'agit d'un APDU de commande dont l'AID correspond à celui de l'applet actuelle, le JCRE le transmet. Sinon, il s'agit d'un APDU de sélection ou de commande d'une autre applet: le JCRE désélectionne l'applet en cours(appel de la

méthode deselect() qui effectue le nettoyage nécessaire), puis sélectionne l'applet dont l'AID est indiqué dans le champs Data Field et appelle se méthode process().

public class APDU

L'unité de traitement de base d'une applet est un objet de type APDU.
Transmis par le JCRE à la réception d'un APDU de commande .
Encapsule les échanges de messages APDUs (commandes et réponses) dans un buffer d'entrées/sorties.

4.3.3 Paquetage javacard.security

Ce paquetage fournit des foncionnalités cryptographiques supportées par la JCVM. Il implémente les clé utilisées dans le cryptage symétrique (DES) et asymétrique (DSA et RSA) .De plus, il propose d'autres fonctions telles les signatures numériques , données aléatoires, ainsi que les fonctions de hashage (hash function) pour générer les *message digest.*

Les applets créent des objets qui manipulent l'information. Tout objet appartient à la classe qui l'a créé. Bien que celle-ci puisse avoir la référence d'un objet , elle ne peut invoquer les méthodes de ce dernier, à moins que l'objet lui appartienne ou qu'il soit explicitement partagé. Bien sûr, une applet peut partager de manière explicite n'importe lequel de ses objets avec une applet particulière ou avec toutes les applets.

4.3.4 Paquetage d'extensions

javacardx.crypto : gestion de clés publiques et privées, fonction de hashage.
javacardx.cryptoEnc : algorithme de chiffrement DES.

Comme nous l'avons déjà dit, ce paquetage est un paquetage d'extensions. Il contient des classes et des interfaces de cryptographie, dont le contenu est rigoureusement contrôlé par les lois en vigueur aux USA. Ce paquetage proposé la classe Cipher qui supporte les fonctions de cryptage/décryptage. Il définit aussi des interfaces auxquelles les applets font appel lorsqu'elles requièrent des services cryptographiques.

En général, on trouve sur la carte un coprocesseur dédié aux opérations de cryptage - décryptage.

Chapitre 5
Sécurité de la plate-forme Java Card

La sécurité des traitements est la raison fondamentale qui est derrière l'emploi de la carte intelligente. C'est pour cette raison que l'aspect sécurité est l'un des plus grands défis auxquels les développeurs doivent faire face. Ce chapitre traite de la sécurité sur la plate-forme JavaCard. Il est constitué de trois sections. La section (1) décrit les caractéristiques sécuritaires JavaCard. La section (2) montre comment ces caractéristiques sont renforcées à travers un ensemble de mécanismes. La section (3) met en relief les points essentiels qu'il faut avoir en tête lors de la conception d'applets.

5.1.Caractéristiques sécuritaires Java Card:

Les caractéristiques sécuritaires de la plate-forme JavaCard sont une combinaison de celles du langage Java et d'autres caractéristiques propres au support carte à puce.

a)Caractéristiques issues de Java

La plate-forme JavaCard est un sous-ensemble du langage Java approprié au support "carte intelligente". De ce fait, la plate-forme JavaCard a hérité de la sécurité construite autour de Java:

- Java est un langage fortement typé. Transtypage très contrôlé : interdiction de convertir des entiers en pointeurs.
- Absence d'arithmétique de pointeurs. En fait, la structure de pointeurs est inexistante.
- Initialisation obligatoire avant utilisation.
- Contrôle des accès aux classes, méthodes et champs. Par exemple, une méthode privée (*private*) ne peut être invoquée qu'à partir de sa classe de définition.

b) Caractéristiques liées à la carte:

La possibilité de coexistence de plusieurs applets sur la même carte impose des restrictions supplémentaires quant aux variables et méthodes de chacune d'elle.

- *Modèles d'objets persistants et transitoires:*

Par défaut, les objets sont stockés en mémoire persistante (EEPROM).Pour des raisons de performance (l'écriture en RAM est 1000 fois plus rapide que l'écriture en EEPROM), il est aussi possible de créer des données volatiles (ex résultats intermédiaires) stockés sous forme d'objets temporaires en RAM.La durée de vie de ces objets peut être soit CLEAR_ON_RESET ou CLEAR_ON_DESELECT. La valeur d'un objet volatile est remise à sa valeur par défaut (zero, false, null), soit quand la session CAD est terminée, soit quand l'applet est desélectionnée.

- *Atomicité:*

Lors d'une opération d'écriture en mémoire persistante, il est possible qu'une panne ou une coupure de courant. survienne. Afin d'assurer l'intégrité des données, deux mécanismes sont définis:

1. La plate-forme JavaCard garantit que toute opération de mise à jour sur un champ unique sera atomique, c.à.d. que si une erreur survient pendant la mise à jour, la plate-forme se charge de restituer au champ sa valeur initiale.

2. La méthode arrayCopy garantit l'atomicité des mises à jour sur les blocs de données. Dans ce cas, cela signifie que si tous les octets du vecteur de départ ne sont pas entièrement copiés, alors le vecteur d'arrivée sera restauré à sa valeur initiale.

- *Applet firewall:*

La sécurité et l'intégrité du système (JCRE) et de chaque applet sont assurées par l'applet firewall (mur de feu) .L'applet firewall renforce l'isolation de chaque applet aussi bien du système que des autres applets. Chaque applet est confinée dans son propre espace appelé contexte. Une applet ne peut pas accéder aux variables et méthodes d'une autre applet à moins d'être définie dans le même paquetage (donc dans le même contexte) ou à travers une interface de partage d'objets rigoureusement contrôlée par le JCRE.

- *Partage d'objets:*

Le partage d'objets dans le système JavaCard est régi par les règles suivantes:

1. Le JCRE est un membre privilégié qui a tous les droits d'accéder aux applets et aux objets qu'elles créent.

2. Les applets dans un même contexte peuvent accéder librement aux méthodes et variables des applets du même contexte.

3. Les applets qui résident dans différents contextes peuvent accéder aux objets d'autres contextes à travers une interface de partage.

- *Méthodes natives:*

Les méthodes natives ne sont pas exécutées par la JCVM et ne sont donc pas sujettes aux règles de sécurité déjà citées. En conséquences, les applets POSTFABRICATION n'ont pas le droit de contenir des méthodes natives. Les applets ROM (PREFABRICATION) qui sont contrôlées par les fabricants ont le droit de contenir de telles méthodes.

5.2.Mécanismes sécuritaires de la plate-forme:

La sécurité est une chaîne, et le moindre point de rupture pourrait tout compromettre. Donc, les mécanismes de sécurité doivent être déployés à chaque étape du développement de l'applet et de son installation.

1. Vérification du code au chargement:

Le code source Java est compilé en utilisant n'importe quel environnement, que ce soit Sun's JDK ou Symantec Café. Les fichiers générés par le compilateur sont appelés fichiers class (extension : *. class).

Le compilateur effectue des vérifications intensives durant la compilation afin de détecter le maximum d'erreurs. Du fait que Java est très typé:

- Les objets ne peuvent pas être transtypés (convertis) vers une sous-classe sans vérification.
- Toutes les références aux méthodes sont vérifiées afin de s'assurer que les objets sont du type correct.
- Le compilateur vérifie que les droits d'accès (ex: référence d'une variable privée) sont respectés.
- Les entiers ne peuvent pas être convertis en objets ni les objets en entiers.
- Le compilateur s'assure que le programme n'accède pas à la valeur d'une variable non initialisée.

2. Vérification des fichiers class:

Bien que les compilateurs sont en général conformes aux restrictions imposées, un fichier class pourrait provenir d'un réseau qui n'est pas forcément sûr. Dans l'environnement Java, tous les fichiers class chargés sont contrôlés par un vérificateur. Ce dernier :

- Renforce les droits d'accès: par exemple, les méthodes et champs privés ne sont pas accessibles en dehors de leurs classes de définitions
- Vérifie les types des paramètres de fonctions (signature).
- S'assure qu'il n'y a pas de conversions illégales

Contrairement à la JVM, la JCVM est décomposée en deux parties: un converter off-card s'exécutant sur PC ou station de travail et un interpréteur on-card .Le converter prend le(s) fichier(s) class d'applet comme argument. Durant la conversion, la JCVM soumet ces derniers aux mêmes règles que la JVM avec les fichiers class.

Comme JavaCard n'inclut qu'un sous-ensemble de Java , la vérification doit se poursuivre afin de vérifier que seules ces fonctions sont implémentées. Cette étape se nomme *Subset checking* (vérification de sous-ensemble) ; les vérifications suivantes y sont réalisées:

- Pas de types de données non supportés (char, long,double ou float).Certaines JCVM supportent l'emploi du type int.
- Pas de caractéristiques propres à Java (multi-threading et tableaux à plusieurs dimensions)
- Vérification des limites de certaines opérations. Par exemple : un paquetage ne peut pas contenir plus de 255 classes ni un vecteur plus de 32,762 éléments.

5.3.Vérification des fichiers CAP:

Les classes sont rangées sous forme d'un ou plusieurs paquetages. Le "converter" prend toutes les classes d'un paquetage et les convertit en fichier CAP. Ce dernier est ensuite chargé sur la carte intelligente et exécuté par l'interpréteur. En plus de créer un fichier CAP, le converter génère un "Export File" (fichier d'exportation) représentant les APIs utilisées par le paquetage en conversion. L'information contenue dans ce fichier n'est pas directement utilisée par l'interpréteur. Elle sert plutôt à l'édition des liens.

En pratique, il n'y a aucune garantie qu'un fichier CAP généré à partir de fichiers class vérifiés soit chargé immédiatement sur la carte. Son intégrité est vérifiée par la "CAP file verifier". A cause de la restriction de mémoire, ce vérificateur réside hors carte. Les vérifications effectuées sont:

- Renforce les droits d'accès: par exemple, les méthodes et champs privés ne sont pas accessibles en dehors de leurs classes de définitions
- Vérifie les types des paramètres de fonctions.
- S'assure qu'il n'y a pas de conversions illégales

Remarquons que toutes ces vérifications sont identiques à celles effectuées par le vérificateur de classe. En plus, les vérifications suivantes propres aux fichiers CAP, sont effectuées :

- Le paquetage et chaque applet qui y est définie doivent avoir un AID valide, dont la longueur varie entre 5 et 16 octets. En plus , les AIDs doivent avoir le même RID.
- Une applet doit implémenter la méthode install() avec une signature correcte afin de pouvoir créer des instances(objets) de cette classe.
- La définition des classes et des interfaces doit respecter la règle suivante: les interfaces doivent apparaître avant les classes et les super-classes avant les sous-classes (dérivées).
- L'indicateur int est activé si le type int est utilisé dans l'applet. Cela permet, comme déjà mentionné, à une version JavaCard n'implémentant pas le type int de rejeter le fichier CAP par simple vérification de l'indicateur int.

5.4.Vérification à l'installation:

L'installation d'un fichier CAP est accomplie grâce à la coopération du programme d'installation off-card et de l'installeur on-card. La sécurité à l'installation est assurée à deux niveaux: au premier niveau, on trouve la sécurité standard renforcée par l'installeur et le JCRE. Au second plan, on trouve les règles dictées par les fabriquants. Ensemble, ils protègent, contre:

- La corruption des données
- L'incompatibilité entre le fichier CAP est les ressources de la carte
- L'insuffisance des ressources durant l'installation et l'initialisation
- Etat indéterminé de la carte suite à une panne ou un retrait durant la procédure d'installation.

Avant de pouvoir écrire les données sur la carte, l'installeur vérifie en premier lieu si le fichier CAP peut entrer dans la carte. Par exemple, il vérifie si il y a assez de mémoire ; si elle ne supporte pas le type int, il vérifie l'indicateur .Il prend ensuite en charge l'édition des liens, c'est à dire les références inter-paquetages. Il vérifie aussi que l'applet ne fait référence qu'aux classes déjà présentes sur la carte, étant donné que le chargement dynamique de classes n'est pas possible. La méthode *install()* est ensuite exécutée pour créer des instances de classe, assigner le contexte, affilier au *group context.*

L'installation est atomique, c'est à dire qu'une erreur engendre la destruction du fichier CAP et de toute applet créée par ce dernier, afin de recouvrer l'espace mémoire.
Au delà de ces caractéristiques minimales définies par l'installeur et le JCRE, les fabriquants sont libres de configurer la carte afin de personnaliser la procédure d'installation, imposant des niveaux de sécurité différents en fonction des différentes sources.
La forme de protection la plus simple est d'authentifier l'installeur via un PIN. D'autres schémas plus élaborés font appel à la cryptographie et à la signature numérique. La section suivante l'illustre.

5.5.Protection par cryptographie:

Durant le processus d'installation, plusieurs parties conjuguent leurs efforts: développeurs, fabriquants de cartes, de terminaux...Afin de rendre la procédure la plus sure possible, JavaCard propose un modèle de "chaîne de confiance", au travers duquel l'identité de chacune des parties concernées est authentifiée et l'intégrité des données assurée.

La cryptographie moderne offre des outils puissants pour assurer l'identité et la confidentialité des données. Tout fichier JavaCard (code source, fichier classe..) peut être crypté durant son transfert et son installation. L'utilisation d'une signature numérique permet d'authentifier son fournisseur.

Avant tout, il est nécessaire que le host (terminal hôte) et la carte s'authentifient mutuellement. Après quoi, la carte peut accepter des données en toute sécurité.

5.6.Sécurité à l'exécution:

La sécurité au niveau de l'exécution concerne l'isolation des applets. Afin de renforcer l'applet firewall, quand un objet est accédé, l'interpréteur vérifie si l'accès peut-être accordé. Cela permet aux applets d'un même paquetage d'accéder aux objets de chacune d'elle. Dans le cas où un travail coopératif inter-paquetage est nécessaire, le JCRE le permet grâce aux objets d'interface partagée (shareable interface objects).

En plus, les règles suivantes doivent être respectées:

- Afin d'éviter des fuites de données, le tampon d'APDU(APDU buffer) doit être remis à zéro à chaque sélection d'applet, avant d'accepter des APDUs de commande.
- Les erreurs d'exécution doivent être prises en charge. Celles qui ne sont pas fatales seront gérées par des exceptions. Les erreur fatales causeront un arrêt de la JCVM, éventuellement un blocage de la carte pour éviter les utilisations futures.

5.7.Sécurité des applets:

La sécurité d'une applet est déterminée par la sécurité de la plate-forme sur laquelle elle s'exécute, ainsi que par la sécurité qu'elle implémente elle-même. La plate-forme JavaCard est conçue de sorte à permettre la création, l'installation, le déploiement ainsi que l'exécution de façon sécurisée.

La sécurité au niveau de l'application est celle qui doit être programmée dans une applet (ex authentification PIN).Parce que la sécurité fait partie intégrante de JavaCard, les programmeurs concentrent leurs efforts sur la stratégie de protection plutôt que sur la compensation des lacunes de la plate-forme.

Cette stratégie est définie afin de parer aux intrusions les plus fréquentes sur les applets. Une bonne stratégie permet d'atteindre les objectifs suivants:

- L'authentification: avant d'accepter des APDUs de commande, l'hôte doit s'authentifier au niveau de l'applet. Cette dernière devra faire de même avant de concéder un service.
- La confidentialité: il est nécessaire de protéger des données sensibles comme le numéro de compte bancaire, le solde... Les données secrètes, telles le PIN et les clés privées ne devraient jamais quitter la carte.
- L'intégrité des données: toute modification de données doit faire l'objet d'un test contre les erreurs afin de s'assurer qu'elles sont correctes. Comme illustration, le solde ne doit jamais devenir négatif .

Nul besoin de rappeler que le niveau de sécurité requis dépend de l'applet. Il est bon aussi de savoir que la sécurité est assurée aux dépends des performances. En conséquence, le développement d'une applet commence par l'évaluation du degré de sécurisation nécessaire, afin de trouver le bon compromis avec les ressources de calcul disponibles.

Chapitre 6
Outils de développement Java Card

6.1 Java Card™ 2.1.2 Development Kit

Le Java Card Development Kit est un environnement de développement complet, gratuit, permettant la création et le test des applets. Il permet aux développeurs de tirer plein profit de l'API Java Card 2.1. afin de créer des applications portables et sécurisées.

Ce kit de développement est disponible aussi bien pour la plate-forme Windows NT4 que pour la plate-forme Solaris, sous forme d'archive zip à partir du site Sun (www.sun.com). Il est fourni avec une série de documents (au format pdf) facilitant la prise en main du kit; on citera à titre d'exemple: *Java Card 2.1.2 Development Kit User's Guide.*Le défaut majeur est l'absence d'une interface graphique.[6]

6.2 Les outils commerciaux adaptés à Java Card

En plus du kit de développement offert par Sun, il existe actuellement quatre outils de création d'Applets Java Card : *Odyssey-Lab* de Bull, *GemXpresso RAD* de Gemplus, JCOP d'IBM et *Cyberflex* de Schlumberger. Ces outils offrent toute une infrastructure permettant d'écrire des applications Java Card et de les tester. Les seules informations disponibles sont des documentations commerciales donc non objectives.

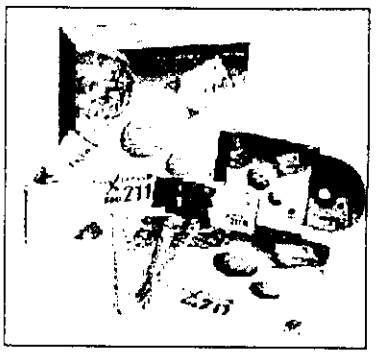
1. Odyssey-Lab de Bull[BUL98]

C'est un environnement de développement fournissant les outils nécessaires pour optimiser et charger les applets (compilées avec un compilateur quelconque). Odyssey I (nom de la carte) est conforme aux spécifications Java Card API 2.0 et sa JVM est l'une des plus performantes du marché selon Bull. Son composant, développé par ST (auparavant SGS Thomson), a été conçu pour les besoins des plates-formes ouvertes et pour être adapté à de nouveaux mécanismes de sécurité (accès sécurisé aux secteurs partitionnés de la mémoire permettant une séparation étanche des applications, authentification ; intégrité et confidentialité assurées lors du chargement des applets ; cryptographie). La carte Odyssey offre en standard l'application de crédit/débit CCPS de Visa et 8 ko d'EEPROM pour des applications propriétaires (ex : programmes de fidélité, porte-monnaie électronique,...). [8]

2. GemXpresso Rapid Applet Development (RAD) de Gemplus

Il est le premier membre d'une famille de produits de carte à puce supportant les APIs Java Card 2.0 pour 8 ou 32 bits. Son kit inclut 2 cartes pour le prototypage d'applets, un lecteur de carte et un environnement de développement. Cette plate-forme de développement est basée sur les spécifications de Java Card 2.0 et sur un processeur 32 bits. Selon les informations commerciales données par Gemplus , elle permettrait d'accélérer le

développement et les tests des applets pour la programmation d'un prototype ou d'un pilote car elle contiendrait un simulateur intégré.



De cette façon, les tests des programmes se feraient directement sur une station de travail. Il ne serait pas nécessaire de les charger sur une carte au préalable pour cette phase de mise au point car le simulateur permettrait de représenter ce qui se passe dans la carte. Malheureusement Gemplus ne donne pas plus d'informations techniques sur ce simulateur. [8]

3. Cyberflex 2.0 Multi8K de Schlumberger

Cyberflex est une carte intelligente Java (appelée J-Card), 8 bits développée par Schlumberger qui peut exécuter des applications Java Card (les Cardlets). Elle est conforme aux spécifications de Java Card 2.0. Le kit vendu pour PC comprend à peu près les mêmes éléments que celui de Gemplus mais apparemment n'a pas d'environnement de développement car il serait très facile de l'intégrer dans un environnement de développement Java existant (comme par exemple Microsoft Visual J++).

Cet outil posséderait lui aussi un **simulateur** qui vérifierait le code d'une cardlet après la phase de compilation et d'édition de lien faite dans un environnement de développement Java quelconque. Pour l'utiliser, il suffirait juste de le lancer en cliquant dans le menu Tool de l'environnement utilisé, puis d'exécuter la cardlet. Il suffirait ensuite d'interagir avec la carte par l'interface graphique du simulateur. Il détecterait les erreurs fonctionnelles possibles du code avant que celui ne soit chargé sur la carte. Après cette simulation le byte code de la cardlet est modifié par MakeSolo (un outil du kit qui teste l'absence de caractéristiques non supportées et qui groupe les fichiers *.class* en un seul fichier) puis peut être chargé sur la carte Schlumberger par l'utilitaire LoadSolo.[8]

4. IBM Java Card Open Platform (JCOP 2002)

Cet kit intégré se compose d'un environnement de développement complet associé au compilateur Java version 2. Il contient aussi un debuggueur ainsi qu'un gestionnaire de carte. Ce dernier permet de lire le contenu d'une carte et de gérer les applets et paquetages présents .

En outre, il permet de visualiser les APDUs de commande et de réponse .A mon sens, c'est un kit assez complet, bien que la documentation laisse quelque peu à désirer. [8]

Chapitre 7
Guide de développement d'une applet Java Card

Ce chapitre est un guide détaillé de la procédure à suivre pour développer une applet Java Card. Chaque étape, aussi bien le design, l'implémentation et l'exécution est amplement détaillée. [4]

7.1. Design de l'applet:

Comme pour le développement de n'importe quel logiciel, avant d'écrire le code, il est indispensable de passer par une phase de design. Durant cette phase, l'architecture de l'applet est définie en quatre points:

- Spécifier les fonctions de l'applet ,
- Assigner les AIDs à l'applet et au paquetage qui va la contenir ,
- Créer les classes,
- Définir l'interface entre l'applet et l'application terminal (spécifier le jeu d'instructions)

a)Spécifier les fonctions de l'applet:

L'applet "porte-monnaie électronique" sauvegarde sur la carte de la monnaie électronique et supporte les opérations bancaires suivantes:

1. Crédit (ajouter de la monnaie)
2. Débit (retirer de la monnaie pour les dépenses)
3. Solde (réserve disponible)

Afin d'éviter l'utilisation frauduleuse de la carte, l'applet utilise un algorithme de sécurité. Cet algorithme nécessite l'utilisation d'un PIN(Personal Identification Number) d'une longueur maximale de 4 digits. Le porteur de la carte introduit le PIN grâce à un clavier connecté au terminal. L'algorithme bloque automatiquement la carte après trois tentatives non fructueuses.

Note: Une application réelle ferait appel à une procédure d'identification bien plus élaborée.

Pour simplifier la tâche, nous ferons les hypothèses suivantes:

- La réserve maximale est de 10.000,00 DA
- Nulle transaction ne peut dépasser 100,00 DA

b)Assigner les AIDs:

Les classes de l'applet "porte-monnaie électronique" sont définies dans un seul paquetage. Les différents AIDs sont tabulés ci-dessous.

On rappelle qu'un AID (Applet Identifier) est constitué de deux champs:

1. Un RID (Resource Identifier : identificateur d'origine) de longueur fixe égale à 5 octets
2. Un PIX (Proprietary Identifier Extension) dont la longueur varie de 0 à 11 octets.

Chaque organisation possède son propre RID qui est délivré par ISO. Celui qui est utilisé dans notre cas est celui de IBM (avec son autorisation).Quant aux PIX, chaque compagnie les attribue à sa guise aux applets qu'elle développe. La seule condition est que les applets d'un même paquetage (donc d'un même fournisseur) doivent partager le même RID.

Champ	Valeur	Longueur en Octets
RID	0x01 ,0x02 ,0x03 ,0x04 ,0x05	5
PIX de paquetage	0x06	1
PIX de l'applet	0x06,0x01	2

Note: 0x signifie hexadécimal $0x01 = 01_h$

c)Définir les classes:

Nous avons déjà vu qu'une applet était une instance dérivée (héritage) de la classe *Java Card.framework..Applet*, dont les méthodes sont listées ci-dessous. L'applet ré-implémente une ou plusieurs de ces méthodes pour obtenir le comportement désiré.
Une applet doit définir et implémenter la méthode statique *install()* pour créer un objet de type applet et en obtenir une référence auprès du JCRE, par appel de la méthode *register(param).*

public static void	install (byte[ ] bArray, short bOffset, byte bLenght)
public boolean	Select( )
public void	Deselect()
public abstract void	Process(APDU apdu)
protected void	Register( )
protected void	Register(byte [ ] bArray, short bOffset, byte bLength)
protected boolean	SelectingApplet( )

La métohde process() (dans la classe de base)est abstraite: l'applet **doit l'implémenter.**Dans cette méthode, chaque APDU est décodé puis interprété. En général, une applet sait interpréter un jeu d'APDUs défini à l'avance, comme nous le verrons dans le paragraphe suivant.

Les méthodes select( ) et deselect( ) sont invoquées par le JCRE quand l'applet est sélectionnée ou déselectionnée. Elles peuvent être ré-implémentées dans le cas où une initialisation particulière est requise. En général, elles sont utilisées telles qu'elles.

Deux méthodes register( ) et une méthode selectingApplet( ) sont respectivement utilisées pour enregistrer l'instance d'applet avec le JCRE et pour détecter les APDUs de selection (distinguer les selectAPDUs des commandAPDUs).

d)Définir l'interface applet-terminal:

Une applet communique avec le terminal via le CAD en échangeant un ensemble d'APDUs. L'interface entre l'applet et et le programme hôte est un ensemble d'APDUs de commande partagés et compréhensibles par les deux parties. Ce jeu est constitué d'un APDU de sélection et d'APDUs de commande:

- La commande SELECT informe le JCRE de sélectionner l'applet sur la carte.
- Les APDUs de commande qui déterminent le comportement de l'applet.

Les formats des APDUs sont définis par le standard ISO7816.Les APDUs forment des paires APDU commande/APDU réponse.

Pour chaque APDU de commande, l'applet doit en premier lieu décoder les champs de l'en-tête afin de déterminer la commande à exécuter. En second lieu, il faudra déterminer si la commande prend des paramètres. Le cas échéant, il faudra aussi décoder les champs de données et vérifier leur format.

Pour les APDUs de réponse, l'applet devra définir des mots d'état pour indiquer le résultat d'exécution de la commande spécifiée par l'APDU de commande correspondant. Si aucune erreur ne se produit, le mot d'état retourné est 0x9000 (ISO7816).Si une erreur survient durant l'exécution, un mot d'état autre que 0x9000 doit être délivré pat l'applet pour indiquer son état et éventuellement un diagnostic de l'erreur.[4]

L'applet "porte-monnaie électronique" supporte le crédit, débit, solde ainsi que l'identification du porteur par PIN. Les APDUs sont listés ci-dessous:

1. Sélection:
   - Commande:

CLA	INS	P1	P2	Lc	Data	Le
0x0	0xA1	0x04	0x0	0x07	0x01,0x02,0x03,0x04,0x05, 0x06,0x01	ND

L'en-tête (CLA,INS,P1,P2) doit être codée comme indiqué afin que le JCRE puisse identifier l'APDU de sélection(commande *selectingApplet( )*).Le champ Data contient l'AID de notre applet. Le JCRE le recherche dans son registre interne contenant les différents AIDs des applets présentes sur la carte, puis il charge l'applet et fait suivre l'APDU de sélection à la méthode *process()*.

   - Réponse:

Data	Status Word (mot d'état)	Signification du SW
Pas de données	0x9000	Traitement réussi
	0x6999 (choix arbitraire)	Echec de traitement: aucun AID correspondant n'a été trouvé dans le registre. .L'applet n'a pu être trouvée ou sélectionnée.

2. Vérification:

- Commande:

CLA	INS	P1	P2	Lc	Data	Le
0xB0	0x20	0x0	0x0	Longueur du PIN	PIN	ND

L'octet INS(0x20) indique le code de l'instruction VERIFY
P1 et P2 ne sont pas utilisés et fixés à 0
Le champ Data indique le PIN.

- Réponse:

Data	Status Word (mot d'état)	Signification du SW
Pas de données	0x9000	Traitement réussi
	0x6300 (choix arbitraire)	Echec de vérification

3. Crédit:

- Commande:

CLA	INS	P1	P2	Lc	Data	Le
0xB0	0x30	0x0	0x0	1	Montant du crédit	ND

- Réponse:

Data	Status Word (mot d'état)	Signification du SW
Pas de données	0x9000	Traitement réussi
	0x6301	Vérification de PIN requise
	0x6A83	Montant invalide
	0x6A84	Crédit maximum dépassé

4. Débit:

- Commande:

CLA	INS	P1	P2	Lc	Data	Le
0xB0	0x40	0x0	0x0	1	Montant du débit	ND

- Réponse:

Data	Status Word (mot d'état)	Signification du SW
Pas de données	0x9000	Traitement réussi
	0x6301	Vérification de PIN requise
	0x6A83	Montant invalide
	0x6A85	Solde négatif

5. Solde:

- Commande:

CLA	INS	P1	P2	Lc	Data	Le
0xB0	0x50	0x0	0x0	ND	ND	2

- Réponse:

Data	Status Word (mot d'état)	Signification du SW
Montant du solde	0x9000	Traitement réussi

7.2. Programme :

Une fois la phase de design achevée, l'étape suivante dans le développement est l'écriture du code. Cette section présente le code détaillé et commenté de l'applet porte-monnaie électronique:

```
package wallet;

import javacard.framework.*;


public class WalletApp extends Applet{

// déclaration des constantes
// code de l'octet CLA dans l'APDU de commande
final static byte Wallet_CLA =(byte)0xB0;

// le second octet est INS.Il identifie une instruction spécifique.Le choix est purement arbitraire.
// Pour nos différentes commandes( vérification de PIN, crédit, débit, solde, ,changement du pin) nous utiliserons les codes 0x20, 0x30, 0x40, 0x50 et 0x60

// codes de l'octet INS dans l'APDU de commande

final static byte VERIFY = (byte) 0x20;
final static byte CREDIT = (byte) 0x30;
final static byte DEBIT = (byte) 0x40;
final static byte GET_BALANCE = (byte) 0x50;

// Il est utile de définir d'autres constantes pour limiter les montants de transactions et de solde
// La constante MAX_BALANCE limite la valeur du solde à 32,767 unités (0x7FFF),
// La constante  constant MAX_TRANSACTION_AMOUNT limite le montant des transactions à 127 unités (0x7F).
// Deux autres constantes caractérisent la longueur de PIN(8)et le nombre maximal de tentatives avant blocage

// solde maximal
final static short MAX_BALANCE = 0x7FFF;

// montant maximal de transaction
final static byte MAX_TRANSACTION_AMOUNT = 127;

// nombre d'essais avant blocage du PIN
final static byte PIN_TRY_LIMIT =(byte)0x03;

// Taille du PIN
final static byte MAX_PIN_SIZE =(byte)0x04;

// Définition des mots d'état qui sont retournés par l'applet après exécution d'une commande ( ex vérification du PIN).Ils transitent par le JCRE qui les transmet à l'hôte a travers le tampon(APdU buffer)

// Echec de vérification du PIN
final static short SW_VERIFICATION_FAILED = 0x6300;

// Vérification de PIN obligatoire
// pour effectuer une transaction
final static short SW_PIN_VERIFICATION_REQUIRED = 0x6301;

// Montant de transaction invalide
// montant > MAX_TRANSACTION_AMOUNT ou montant < 0
final static short SW_INVALID_TRANSACTION_AMOUNT = 0x6A83;

// dépassement de capacité pour le solde
final static short SW_EXCEED_MAXIMUM_BALANCE = 0x6A84;

// solde négatif
final static short SW_NEGATIVE_BALANCE = 0x6A85;
```

```
// l'applet définit deux variables
// pin est un objet de type OwnerPin qui encapsule les opérations sur le PIN ( vérification, mise à jour)
// La variable balance représente le solde actuel de l'applet.

OwnerPIN pin;
short balance;
short offset=0x0000;
byte[] pinvalue={0x01,0x09,0x07,0x09};  // PIN = 1979

// constructeur de classe
private WalletApp (byte[] bArray,short bOffset,byte bLength){


pin = new OwnerPIN(PIN_TRY_LIMIT, MAX_PIN_SIZE);

// Les paramètres d'installation contiennent la valeur initiale du PIN.
// C'est une bonne habitude d'initialiser les membres de classe à l'intérieur du constructeur, cela permet
d'éviter d'être à court de mémoire pendant l'exécution

pin.update( pinvalue, offset, MAX_PIN_SIZE);

//enregistrement avec le JCRE
register();

 } // fin du constructeur

// installation de l'applet

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

   // création d'une nouvelle instance

new WalletApp(bArray, bOffset, bLength);
// bArray : vecteur paramètres d'installation
 }

// sélection de l'applet
public boolean select()
{
// L'applet décline la demande si le nombre d'essais >3
 if ( pin.getTriesRemaining() == 0 )
 return false;

 return true;

}

//Désélection de l'applet ( réception d'un autre APDU de sélection)
//L'applet effectue des opération de nettoyage mémoire

public void deselect() {
}

//traitement des APDUs
public void process(APDU apdu) {

//Obtention d'une référence de l'APDU
byte buffer[] = apdu.getBuffer();
```

```
// analyse de l'en-tête
// si CLA=0 et INS =0xA4 alors c'est un APDU de sélection

if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
  (buffer[ISO7816.OFFSET_INS] == (byte)
                    (0xA4)) )
 return;

//--------------------------------------------------------------------
// Il est possible d'utiliser la méthode selectingApplet() qui effectue ce test
// if (selectingApplet())
//  return;
 //--------------------------------------------------------------------


//Vérification de l'octet CLA afin de s'assurer de l'intégrité des commandes
// La valeur indiquant les APDUs de commande est 0xB0. Si telle n'est pas sa valeur, il ne sera pas
reconnu par la méthode process() ie il ne représente pas une commande exécutable

if (buffer[ISO7816.OFFSET_CLA] != Wallet_CLA) // Wallet_CLA=0xB0
  ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);


//exécution de la méthode appropriée
// il suffit de lire la valeur de l'octet INS pour choisir entre crédit, débit, vérification et solde

switch (buffer[ISO7816.OFFSET_INS])
{
   case GET_BALANCE:  getBalance(apdu);
                      return;
   case DEBIT:       debit(apdu);
                      return;
   case CREDIT:       credit(apdu);
                      return;
   case VERIFY:       verify(apdu);
                      return;
        default:     ISOException.throwIt
       (ISO7816.SW_INS_NOT_SUPPORTED);
 }

}

// opération de crédit : il s'agit de diminuer le crédit au solde (balance)
// Mais avant, il faut vérifier le PIN

private void credit(APDU apdu) {

if ( ! pin.isValidated() ) // authentification d'accès
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);

// référence de l'APDU
byte buffer[] = apdu.getBuffer();

// Lc : nombre d'octets à lire ( taille du champ de données transmis)

byte numBytes = buffer[ISO7816.OFFSET_LC];

// il est nécessaire de vérifier l'intégrité des données reçues avant de créditer le compte
// la méthode setIncomingAndReceive() renvoie le nombre d'octets reçus dans le champ de données
```

```
byte byteRead =(byte)(apdu.setIncomingAndReceive());

// il est nécessaire que les deux champs aient une taille de 1 octet car nous avons défini la limite de
transaction comme une variable byte

if ( ( numBytes != 1 ) || (byteRead != 1) )
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

// montant du crédit
byte creditAmount = buffer[ISO7816.OFFSET_CDATA];

// vérification du crédit
if ( ( creditAmount > MAX_TRANSACTION_AMOUNT) || ( creditAmount < 0 ) )
   ISOException.throwIt(SW_INVALID_TRANSACTION_AMOUNT);

// vérifier que le nouveau solde tient sur une variable short (2 octets)
if ( (short)( balance + creditAmount) > MAX_BALANCE )
  ISOException.throwIt(SW_EXCEED_MAXIMUM_BALANCE);

// En l'absence d'erreur, valider la transaction
balance = (short)(balance + creditAmount);
return;
} //fin de la méthode de crédit

// cette méthode achevée avec succès, l'applet rend le contrôle au JCRE qui renvoie un APDU de réponse
à l'application hôte contenant le mot d'état 0x9000;indiquant l'exécution sans erreur

private void debit(APDU apdu) {

if ( ! pin.isValidated() ) // authentification d'accès
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);

// référence de l'APDU
byte buffer[] = apdu.getBuffer();

// Lc : nombre d'octets à lire ( taille du champ de données transmis)

byte numBytes = buffer[ISO7816.OFFSET_LC];

// il est nécessaire de vérifier l'intégrité des données reçues avant de débiter le compte
// la méthode setIncomingAndReceive() renvoie le nombre d'octets reçus dans lec champ de données

byte byteRead =(byte)(apdu.setIncomingAndReceive());

// il est nécessaire que les deux champs aient une taille de 1 octet car nous avons défini la limite de
transaction comme une variable byte

if ( ( numBytes != 1 ) || (byteRead != 1) )
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

// montant du débit
byte debitAmount = buffer[ISO7816.OFFSET_CDATA];

// vérification du débit
if ( ( debitAmount > MAX_TRANSACTION_AMOUNT) || ( debitAmount < 0 ) )
   ISOException.throwIt(SW_INVALID_TRANSACTION_AMOUNT);

// vérifier que le nouveau solde est positif
if ( (short)( balance - debitAmount)< (short)0 )
  ISOException.throwIt(SW_NEGATIVE_BALANCE);
```

```
// En l'absence d'erreur, valider la transaction
balance = (short)(balance -debitAmount);
return;
} //fin de la méthode de débit

private void getBalance(APDU apdu) {

if ( ! pin.isValidated() ) // authentification d'accès
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);

// référence de l'APDU
byte buffer[] = apdu.getBuffer();

//informer le JCRE que l'applet doit envoyer des données
short le=apdu.setOutgoing();

// fixer la taille du champ de données
apdu.setOutgoingLength((byte)2);

//transférer le champ au tampon APDU avec décalage 0
Util.setShort(buffer,(short)0,balance);

// commander l'envoi de la réponse
apdu.sendBytes((short)0,(short)2);
}

// vérification du PIN
private void verify(APDU apdu) {

// référence de l'APDU
byte buffer[] = apdu.getBuffer();

// réception du PIN
byte byteRead=(byte)(apdu.setIncomingAndReceive());

// vérifier le PIN
// Le PIN est lu dans le tampon APDU avec l'offset ISO7816.OFFSET_CDATA
//Longueur du PIN = byteRead

if ( pin.check(buffer, ISO7816.OFFSET_CDATA,byteRead)==false)
ISOException.throwIt(SW_VERIFICATION_FAILED);

}//fin de la méthode
}// fin de l'applet
```

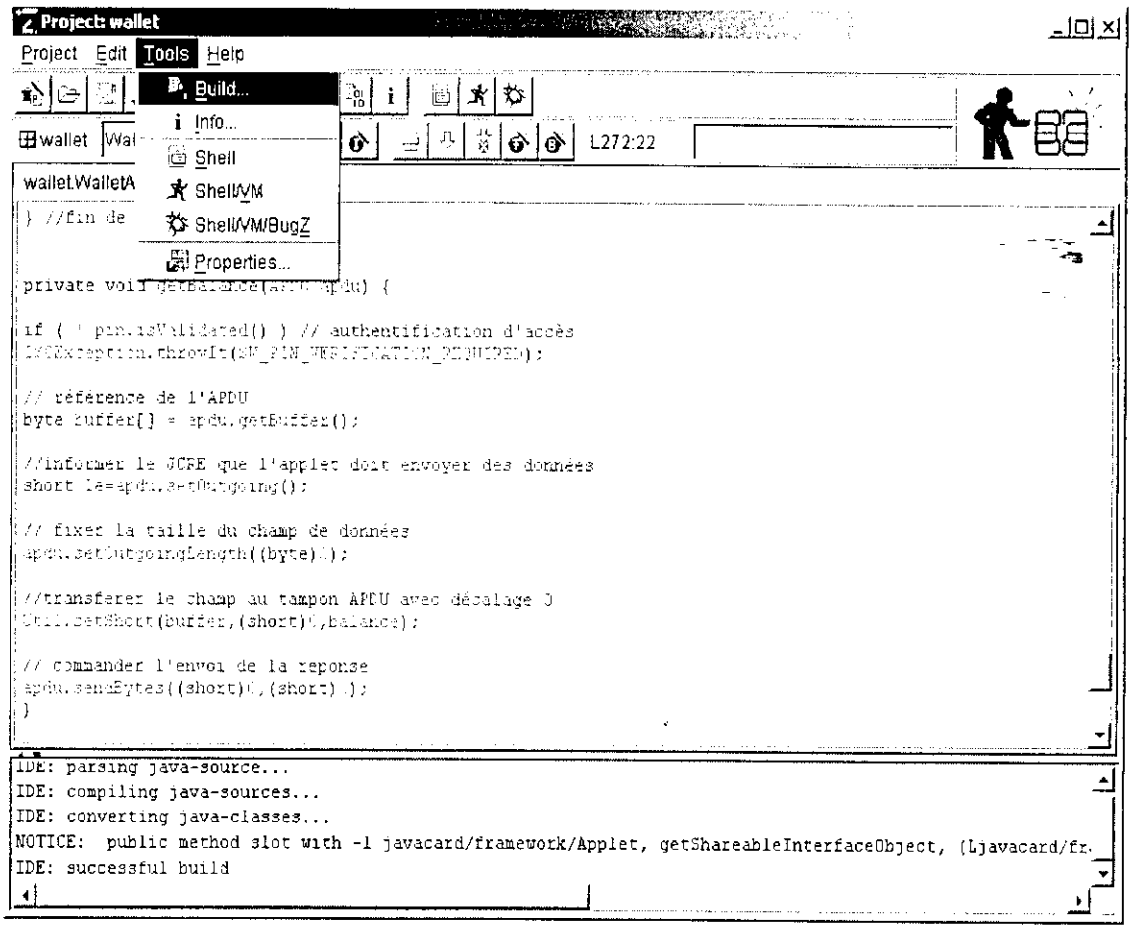
7.3. Compilation et exécution dans JCOP IDE :

Avant toute chose, il est bon de comprendre comment installer et configurer le kit de développement. Bien qu'il s'agisse d'un lecteur Plug&Play, il est nécessaire de redémarrer le système après l'avoir connecté au PC.
L'installation des drivers se fait automatiquement. La seule configuration manuelle qu'il faudra effectuer est la mise à jour de la variable système CLASSPATH.
Comme Windows2000 introduit une nouvelle façon d'accomplir cette tâche, j'ai jugé utile de l'expliquer.

Pour ce faire, aller dans Poste de travail, panneau de configuration, Système puis cliquer sur l'onglet Avancé. Choisir variable d'environnement puis CLASSPATH.
Ajouter le chemin suivant : D:\Program Files\ibm\java13\lib\tools.jar

Ayant écrit et sauvegardé la classe WalletApp dans le fichier WalletApp.java, il est temps de le compiler pour générer le fichier class lequel servira pour créer le fichier Cap.

Cette opération s'effectue facilement via la commande Build dans le menu Tools du JCOP IDE.Si tout se passe bien, on obtient en bas de l'ecran : *Successful build*



L'étape suivante consiste a faire passer l'applet et son paquetage sur la carte. Pour ce faire, il est bon d'employer l'utilitaire JCShell.

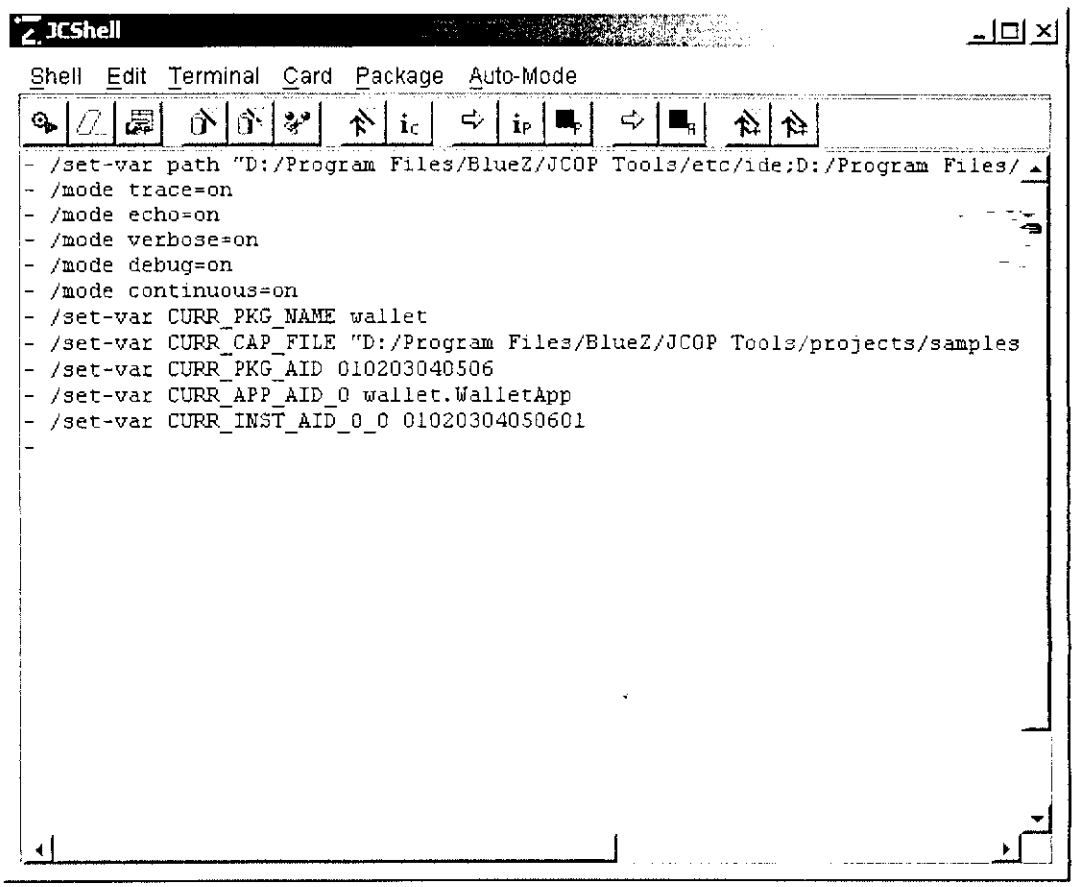
JCShell est un programme qui permet non seulement de charger l'applet sur la carte , mais surtout de tracer les APDUs échangés.

Pour activer JCShell, il suffit d'aller dans le menu Tools puis Shell.

Si tout se passe bien, on obtient une invite de commande. A partir de ce moment, la main est donnée au programmeur pour commander la carte.
La première étape est de créer un lien vers le lecteur et la carte. Ceci est achevé comme suit :

/terminal
/card

Le lecteur s'allume et s'il contient une carte, retourne ses caractéristiques(ex ATR).
L'étape suivante est l'authentification. Il est bon de noter que celle-ci est propre au kit de développement JCOP et qu'elle n 'existe pas sur Schlumberger Cyberflex.
Il suffit d'aller dans le menu *Card* puis *Authenticate*.
A ce stade, la carte est totalement accessible pour le programmeur.



Lisons d'abord le contenu d'une javacard vierge:
cm>/card-info

Card Manager AID : A0000000030000
Card Manager state : OP_READY

Load File : LOADED (--------) A0000000620101 (javacard.framework)
Load File : LOADED (--------) A0000000620001 (java.lang)
Load File : LOADED (--------) A0000000620102 (javacard.security)
Load File : LOADED (--------) A0000000620201 (javacardx.crypto)

```
Load File :    LOADED (--------) A0000000030000  (visa.openplatform)
Load File :    LOADED (--------) "1PAY."         (PSE)
Load File :    LOADED (--------) A00000000310    (VSDC)
Load File :    LOADED (--------) "access"
Load File :    LOADED (--------) A0000000036010  (VisaCash)
Load File :    LOADED (--------) A00000009820
```

Chargeons d'abord le paquetage Wallet (converted package) de l'applet WalletApp

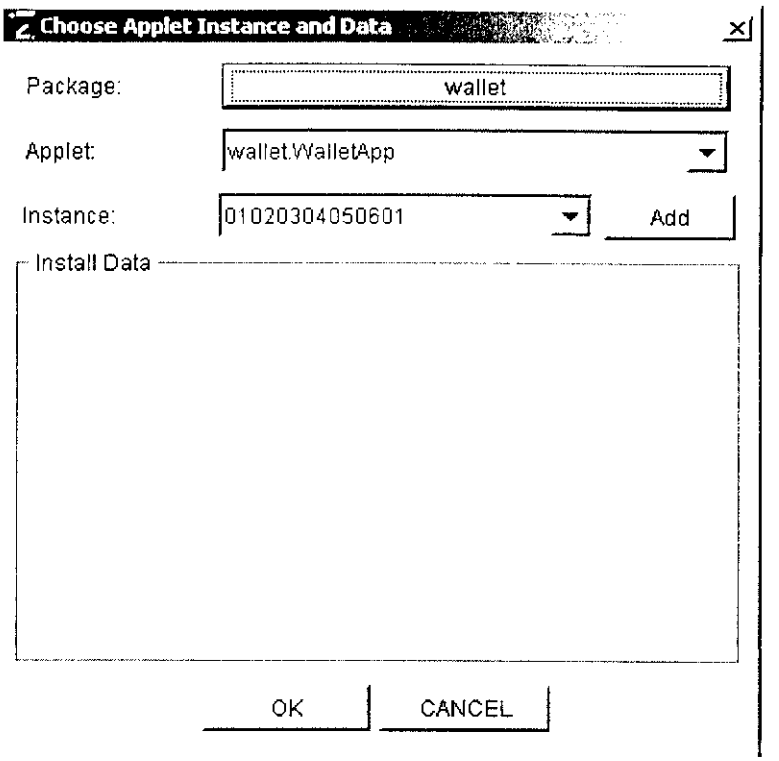
```
cm> upload -p "D:/Program Files/BlueZ/JCOPTools/projects/samples/wallet/javacard/wallet.cap"
 => 80 E6 02 00 0B 06 01 02 03 04 05 06 00 00 00 00  ..... 00
(251 msec)
 <= 00 90 00 (APDU de réponse OK opération réussie)
Status: No Error
 => 80 E8 00 00 FE C4 82 02 E8 01 00 10 DE CA FF ED  .....
   01 02 04 00 01 06 01 02 03 04 05 06 02 00 1F 00  .....
   10 00 1F 00 0B 00 15 00 6A 00 12 01 B8 00 0A 00  ........j.....
   40 00 00 00 92 00 00 00 00 00 00 02 01 00 04 00  @.....
   15 02 00 01 07 A0 00 00 00 62 01 01 00 01 07 A0  .........b.....
   00 00 00 62 00 01 03 00 0B 01 07 01 02 03 04 05  ...b.....
   06 01 00 6E 06 00 12 00 80 03 04 00 02 04 04 00  ...n.....
   00 00 0E FF FF 00 01 00 16 07 01 B8 00 01 10 AD  .....
   00 8B 00 01 61 04 03 78 04 78 01 10 AD 00 8B 00  ....a..x.x.....
   02 7A 02 21 19 8B 00 03 2D 1A 03 25 61 0A 1A 04  .z.!....-..%a...
   25 10 A4 6B 03 7A 1A 03 25 10 B0 6A 08 11 6E 00  %..k.z..%..j..n.
   8D 00 04 1A 04 25 75 00 2D 00 04 00 20 00 27 00  .....%u.-... .'.
   30 00 21 00 40 00 1B 00 50 00 15 18 19 8D 00 05  0.!.@...P.....
   7A 18 19 8D 00 06 7A 18 19 8D 00 07 7A 18 19 8D  z.....z.....z...
   00 08 7A 11 6D 00 8D 00 04 7A 05 30 8F 00 09 3D  ..z.m....z.0...=
   18 1D 1E 8D 00 0A 7A 05 40 18 8D 00 0B 03 B7 0C  ......z.@.....
   18 07 90 00                                      ....
(851 msec)

 <= 90 00 (APDU de réponse OK opération réussie)

Load report:
  744 bytes loaded
effective code size on card:
     + package AID      6
     + applet AIDs      14
     + classes          21
     + methods          443
     + statics          0
     + exports          0

overall          484  bytes
```

Maintenant, chargeons le fichier cap(Applet convertie) en cliquant sur le menu Package-Install Applet. Ceci fait apparaître la fenêtre suivante pour gérer les paramètres d 'installation (ex vecteur d'initialisation).



Comme notre applet ne nécessite aucune initialisation particulière, ce champ reste vide. Cliquons sur OK

Le JCShell répond par la commande :

```
cm> install -i 01020304050601 010203040506 01020304050601
=> 80 E6 0C 00 1B 06 01 02 03 04 05 06 07 01 02 03  ................
   04 05 06 01 07 01 02 03 04 05 06 01 01 00 00 00 00
 (2043 msec)
 <= 00 90 00
Status: No Error
```

Faisons une nouvelle authentification et voyons de nouveau le contenu de la carte

```
Card Manager AID   : A0000000030000
Card Manager state : OP_READY

   Application: SELECTABLE (--------) 01020304050601
   Load File :   LOADED (--------) 010203040506
   Load File :   LOADED (--------) A0000000620101  (javacard.framework)
   Load File :   LOADED (--------) A0000000620001  (java.lang)
   Load File :   LOADED (--------) A0000000620102  (javacard.security)
   Load File :   LOADED (--------) A0000000620201  (javacardx.crypto)
   Load File :   LOADED (--------) A0000000030000  (visa.openplatform)
   Load File :   LOADED (--------) "1PAY."         (PSE)
   Load File :   LOADED (--------) A00000000310    (VSDC)
   Load File :   LOADED (--------) "access"
   Load File :   LOADED (--------) A0000000036010  (VisaCash)
   Load File :   LOADED (--------) A00000009820
```

On voit en gras le paquetage ainsi que l'applet qui viennent d'être transférés.
La dernière étape est de sélectionner l'applet et de lui envoyer quelques APDUs de commande.

```
cm> /select 01020304050601
 => 00 A1 04 00 0A 01 02 03 04 05 06 01 00          ...../APDU de sélection
 (210 msec)
 <= 90 00                                          ../APDU de réponse
Status: No Error
```

Lisons le solde bancaire

```
cm> /send b050000002
 => B0 50 00 00 02                                 .P...
 (1592 msec)
 <= 63 01                                          c.
Status: 0x6301
```

Ce code d'erreur a été défini comme *Vérification de PIN* requise car le PIN n'a pas été introduit. En fait, le premier APDU de commande qui doit envoyé est l'APDU de commande VERIFY.

Vérifions le PIN :

```
cm> /send b02000000401090709
 => B0 20 00 00 04 01 09 07 09                     . .....
 (201 msec)
 <= 90 00                                          ..
Status: No Error
```

A partir de maintenant, la carte est débloquée et répond aux APDUs. Relisons le solde bancaire :

```
cm> /send b050000002
 => B0 50 00 00 02                                 .P...
 (1242 msec)
 <= 00 00 90 00                                    Solde nul
Status: No Error
```

Introduisons de la monnaie dans la carte.(79 unités)

```
cm> /send b03000000179
 => B0 30 00 00 01 79                              .0...y
 (180 msec)
 <= 90 00                                          ..
Status: No Error
```

Retirons 50 unités :

```
cm> /send b04000000150
 => B0 40 00 00 01 50                              .@...P
 (181 msec)
 <= 90 00                                          ..
Status: No Error
```

Lisons le nouveau solde :

```
cm> /send b050000002
```

=> B0 50 00 00 02 .P...
(170 msec)
*<= **00 29** 90 00 .)..*
Status: No Error

On vérifie bien que 79-50=29.

Essayons de retirer plus que le solde(29 unités).Soit un retrait de 50 unités :
cm> /send b04000000150
*=> B0 40 00 00 01 **50** .@...P*
(200 msec)
*<= **6A 85** .//réponse différente de 0x9000*
Status: 0x6A85

Ce code d'erreur a été défini au début comme *Solde négatif.* L'applet ne permet pas les découverts. Les autres tests peuvent être effectués d'une manière similaire.

On voit donc que le rôle de l'applet (Serveur) est, dans un sens, de générer des APDUs de réponses. Quant aux APDUs de commande, ils seront, soit générés manuellement comme dans notre cas, soit automatiquement par l'application cliente qui réside sur un terminal.

Avant de retirer la carte du lecteur, il est nécessaire de fermer la connexion avec le JCShell (dans un contexte plus général, avec le programme client).

cm>/close
/q

Chapitre 8
Optimisation des applets

Un des problèmes majeurs à considérer lors du design et l'implémentation d'une applet est la limitation des ressources disponibles sur la carte (ressources limitées en terme de calcul comme en mémoire).

Cette section s'intéresse à l'optimisation des applets. Elle présente un ensemble de recommandations à respecter lors de la phase de développement, afin d'assurer un bon compromis entre fonctionnalité et performance.[1]

8.1. Optimisation du design

L'optimisation d'une applet devrait toujours débuter par l'optimisation du design. La raison est que celle-ci donne de meilleurs résultats lorsqu'elle est appliquée à un niveau global.

L'optimisation du design repose sur les caractéristiques orientées objet du langage Java. L'utilisation des classes permet de définir les variables et méthodes communes aux objets d'une même catégorie. De ce fait, elles fournissent les avantages de modularité et de réutilisabilité. Ces dernières peuvent être davantage étendues par l'utilisation de l'héritage. Ceci permet de définir un comportement générique dans une classe mère et d'ajouter des caractères spécifiques dans chaque sous-classe (classe héritière).

L'optimisation est un compromis plutôt qu'une solution. En effet, une applet dont l'architecture maximise la flexibilité par utilisation de l'héritage et de plusieurs classes engendre une baisse conséquente de performance. Par contraste, une applet compacte avec peu de classes et de méthodes consommera nettement moins de ressource mais réduira considérablement les opportunités de travail coopératif entre applets.

Malheureusement, il n y a aucune technique qui permette de choisir l'une ou l'autre des deux approches. Le programmeur devra chercher le compromis entre ressources nécessaires et ressources disponibles. Comme règle générale, une applet devra avoir entre 5 et 10 classes avec une profondeur d'héritage maximale égale à 3 niveaux.

8.2. Temps d'exécution

Les opérations de cryptographie et d'écriture en EEPROM sont les plus gourmandes en calculs. En conséquences, il n'est pas envisageable d'exécuter des fonctions de cryptage sur des données volumineuses. En général, les algorithmes gourmands en temps de calcul sont plutôt employés pour la génération de signatures numériques.

Afin de minimiser les écritures EEPROM, l'utilisation de tableaux transitoires (RAM)est fortement conseillée pour les valeurs intermédiaires et les variables temporaires.(l'écriture RAM est 1.000 fois plus rapide que l'écriture EEPROM). [1]

8.3. Appel de fonctions

Durant l'exécution, les données(paramètres, variables locales, résultats intermédiaires) sont stockées dans une pile appelée *stack* .Dans plusieurs cartes, sa taille ne dépasse pas 200 octets. Donc, afin de réduire l'utilisation du *stack*, il est conseillé de limiter l'imbrication des méthodes, qui pourrait engendrer un *stack overflow*. En particulier, éviter les fonctions récursives.

8.4. Création d'objets

Comme déjà exposé dans le chapitre API JavaCard, une applet est créée après l'exécution de la méthode *install()*.Dans la mesure du possible, une applet devrait créer tous les objets et variables dont elle aura besoin à l'intérieur du constructeur. Cela présente deux avantages:

- L'applet peut gérer l'espace afin d'éviter des dépassements mémoire.
- Etant donné qu'*Install()* est une méthode atomique, si une erreur survient lors de son exécution, l'espace occupé sera automatiquement restitué.

8.5. Elimination du code redondant

L'élimination de la redondance de code est une technique d'optimisation très efficace. En effet, il y a redondance de code lorsque deux ou plusieurs parties d'un programme répètent une fonction identique. L'optimisation peut être obtenue en isolant le code dupliqué dans une méthode à part.

Afin d'accroître la modularité et la réutilisabilité (donc éliminer la redondance),il est souvent intéressant de décomposer un programme en plusieurs méthodes de 10 à 15 lignes chacune. Cependant, cela engendre en général un programme global plus long.

La tâche d'un développeur sera alors d'identifier les parties qu'il est vraiment nécessaire d'isoler. Les méthodes de petite taille seront généralement laissées avec le programme principal, alors que les blocs répétés de grande taille seront isolés, permettant ainsi d'économiser un espace mémoire parfois considérable.

8.6.Réutilisation d'objets

Dans la plate-forme Java, les objets sont créés en fonction du besoin et sont détruits par le Garbage Collector (Ramasse-miettes) .

Sur la plate-forme JavaCard, un objet persistant existe durant toute la durée de vie d'une applet. Il est donc impératif de savoir que l'espace occupé par un objet persistant ne peut être récupéré.
Considérons l'exemple suivant:

```
public void Méthode() {
      Object a= new Object();
}
```

A chaque fois que la méthode est invoquée, il y a création de l'objet a, qui devient inaccessible en dehors de la méthode(variable locale).Tôt ou tard, l'accumulation de ces objets consommera toute la mémoire, rendant la carte inutilisable.

A cet effet, la règle générale est qu'une déclaration unique d'un objet doit servir plusieurs fois, en attribuant au même objet différentes valeurs en fonction de son utilisation.
L'application de cette règle transforme l'exemple précédent comme suit:

```
public class maclasse{

      private static Object a; // déclaration unique de l'objet a

                  public void Méthode_1(var_1) {
                        a= var_1;
                  }

      .....

                  public void Méthode_n(var_n) {
                        a= var_n;
                  }

}
```

8.7. Utilisation de tableaux

Afin d'optimiser l'utilisation de la mémoire, si un même élément d'un tableau est accédé plusieurs fois dans une même méthode, il est préférable de sauvegarder la valeur de cet élément dans une variable locale lors du premier accès et d'utiliser cette référence par la suite. Considérons l'exemple suivant:

```
if (buffer[ISO7816.OFFSET_INS]==VERIFY)
      verifyPIN(apdu);
else if (buffer[ISO7816.OFFSET_INS]==CREDIT)
      credit(apdu);
if (buffer[ISO7816.OFFSET_INS]==DEBIT)
      debit(apdu);
else if (buffer[ISO7816.OFFSET_INS]==CHECKBALANCE)
      checkBalance(apdu);
```

On voit que l'élément dont l'indice est `ISO7816.OFFSET_INS` dans le tableau buffer est accédé à plusieurs reprises afin de déterminer la commande spécifiée par l'APDU de commande. Ce code peut-être optimisé de la manière suivante:

```
Byte ins= buffer[ISO7816.OFFSET_INS];

if (ins==VERIFY)
      verifyPIN(apdu);
else if (ins==CREDIT)
      credit(apdu);
if (ins==DEBIT)
      debit(apdu);
else if (ins==CHECKBALANCE)
      checkBalance(apdu);
```

8.8. Structure switch contre structure if

La structure if précédente peut être remplacée par la structure:

```
Byte ins= buffer[ISO7816.OFFSET_INS];

Switch(ins){
case VERIFY
      verifyPIN(apdu);
      break;
case CREDIT
      credit(apdu);
      break;
case DEBIT
      debit(apdu);
      break;
case CHECKBALANCE
      checkBalance(apdu);
      break;
      }
```

En général, une structure switch s'exécute plus rapidement et prend moins d'espace mémoire qu'un if-else équivalent. Cependant, il est possible que l'inverse se produise.

8.9. Expressions arithmétiques

L'utilisation d'une affectation compacte au lieu de deux affectations consécutives permet d'économiser de l'espace mémoire. En effet, considérons :

```
x=a+b; x=x-c;
x=a+b-c;
```

La seconde façon de coder l'affectation est beaucoup plus économe en mémoire. En effet, des affectations séparées engendrent des opérations supplémentaires (ex sauvegarder la valeur de (a+b) en mémoire puis la recharger lors de l'exécution de la seconde instruction).
Combiner plusieurs expressions en une seule paraît alors être une bonne habitude de programmation; cependant , il faut veiller à ne pas imbriquer trop d'expressions arithmétiques car cela réduit la lisibilité du code source et engendre souvent des erreurs de programmation.

8.10. Optimisation des variables

Une variable locale doit être initialisée avant de pouvoir l'utiliser. En l'absence d'une initialisation explicite, une initialisation implicite est appliquée à toute variable non initialisée. Celle-ci attribue une des valeurs zero (byte, short, int *), false(boolean), null(tableau).C'est pourquoi il est inutile d'initialiser des variables si leur valeur est celle par défaut.[1]

```
public class MonApplet extends Applet
      {
      // initialisation inutile
      static int a=0;
      boolean b=false;
      byte[ ] buffer=null;
      }
```

*sous réserve que le type int soit supporté

Conclusion

Depuis son invention en 1974, la carte à puce a trouvé de nombreux domaines d'applications : cartes de crédit, commerce, santé, télécommunications, etc. L'introduction récente de Java dans la technologie de la carte à puce (JavaCard) ouvre des perspectives intéressantes d'intégration. En effet la possibilité de charger des applets JavaCard modifie notre vision banale d'une carte, qui n'est plus seulement une simple mémoire morte mais, la pastille de silicium collée à la carte est en fait un microprocesseur sans clavier ni écran qui réalise néanmoins toutes les fonctions de stockage et de calcul de l'ordinateur.

A travers ce travail, nous avons étudié l'architecture JavaCard, qui se résume en une machine virtuelle (JCVM), est des API (Application Programming Interface) JavaCard. La JCVM, implantée au-dessus de la puce intégrée, sert de système d'exploitation et de fonctions de bas niveau.

Ainsi, les spécificités liées à la technologie du constructeur sont masquées et ce qui est accessible est une interface commune à tous les constructeurs.

Les API javacard définissent les conventions utilisées par les applets pour accéder aux méthodes de base et à l'environnement de travail. Ceci permet aux développeurs d'écrire des applications sans se soucier des détails de l'infrastructure de la carte. Nous avons aussi étudié les différentes phases du processus de développement et de mise en œuvre d'une application Javacard : de la création à l'exécution de l'applet au sein d'une carte.

A ce niveau, il faut retenir que la procédure d'installation charge l'applet de façon définitive dans la mémoire (soit la ROM ou l'EEPROM) de la carte. Cette applet restera inactive aussi longtemps qu'elle n'est pas explicitement choisie pour être exécutée. Mais une fois sélectionnée, l'applet sera prête à recevoir les APDU (Application Programming Data Unit) provenant du CAD (Card Acceptance Device).

Après la maîtrise des aspects techniques des cartes à puce javacard, nous avons procédé au développement d'une application de porte-monnaie électronique (PME),
permettant le paiement avec une même carte d'un ensemble de biens et de services de natures différentes. L'intérêt du PME vient du fait qu'il a beaucoup de points communs avec les applications bancaires et de commerce électronique, et que tous les réseaux de cartes débit/crédit planifient de migrer de la solution carte à piste magnétique vers les cartes à microprocesseur.

Malgré les difficultés d'implémentation pratiques, nous avons réussi à installer notre applet dans une carte à puce Javacard d'IBM en utilisant le kit de développement JCOP IDE. Et nous avons tracé les APDU's échangés avec l'applet serveur chargée dans la carte. Par ailleurs, JavaCard est complètement indépendante de la plate forme matérielle de la carte, cependant des efforts supplémentaires devront être faits dans le sens de la compatibilité cartes/lecteurs : En effet, malgré tous nos efforts, la carte JCOP d'IBM n'a pas fonctionné avec le kit Cyberflex Schlumberger.
Des améliorations et des extensions peuvent être apportées à notre application. La première serait de développer l'application cliente (à installer sur le terminal) qui se

chargera de générer les APDU's de commande et d'interpréter les APDU's de réponse.

Pour l'aspect sécurité, les données enregistrées sur la carte sont inviolables car leur accès est contrôlé par un code (PIN) de quatre octets. L'une des améliorations possibles serait d'étendre la longueur du PIN a huit octets, ainsi que le recours à un algorithme de cryptage pour sécuriser l'opération d'authentification. D'un autre côté, en plus du PIN et malgré la faible capacité mémoire de la carte à puce, elle est en mesure de stocker d'autres éléments d'authentification (empreinte digitale, image du fond de l'œil,...) destinés à authentifier le porteur.

Enfin, en ce qui concerne les tendances technologiques actuelles, on peut citer deux cas qui changent complètement les données d'implémentation de la carte à puce dans différents secteurs (transports, santé, sécurité, contrôle d'accès etc....) :

1. La carte à puce sans contact où une antenne noyée dans le corps de la carte permet au terminal d'alimenter l'électronique de la carte et de communiquer avec elle.

2. La carte à puce compatible USB (Universal Serial Bus) qui se passe d'un lecteur de carte. Ce qui répond parfaitement à la demande de sécurité dans les réseaux d'information de l'entreprise et dans les applications liées au commerce électronique.

Ces technologies, jumelés avec JavaCard, vont favoriser sans nul doute l'adoption de la smart-card dans beaucoup de secteurs de l'industrie, par la convergence des applets sur des cartes multi-applicatives.

Bibliographie et Webographie :

[1] Java Card Technology for Smart Cards: Architecture and Programmer's Guide
Par Zhiqun Chen_Sun MicroSystems
Edition Addison Wesley Avril 2000

[2] Construction d'applications avec la carte à puce
Jean-Jacques Vandewalle
Gemplus Research Group Août 1998

[3] Java Cryptography : SSL and TLS (Secure Sockets Layer and Transport Layer Security)

[4] · Programmation C++ :Deitel et Deitel édition 1998
Edition Prentice Hall http://www.prenthall.com
Très utile pour les principes de POO ainsi que les techniques d'écriture dans les flots de données.

[5] 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

[6] Site Sun :
http://java.sun.com/products/javacard

[7] Java Card Forum :
http://www.javacardforum.com/

[8] 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

[9] Club java:
www.club-java.com

[10] · The Java Tutorial - http://java.sun.com/nav/read/Tutorial/index.html
Transcription « on-line » d'un livre de présentation de Java et de la Programmation Orientée Objet sur l'Internet. C'est le livre de référence bien qu'il parle essentiellement de la version 1.0.2 .

[11] · Java Series - http://www.aw.com/cp/javaseries.html
Aide en ligne directe sur les évolutions de Java réalisées par Sun Microsystems, mise à jour permanente pour corriger les bugs de Java, liste d'URL pour trouver d'autres compléments d'informations sur Java. (site US)

[12] · Sun Microsystems - http://www.sun.com
Le site officiel des concepteurs de Java. Des informations sur l'avenir de la « plate-forme Java », sur le langage et sur son utilisation. De la documentations et des possibilités de télécharger des versions démos ou/et bêta des logiciels.
Téléchargement gratuit depuis plusieurs adresses FTP du Java Developer's Kit, le fameux « JDK ». (site US)

[13] · Java Soft Home Page - http://java.sun.com & http://www.javasoft.com
Toute l'actualité Java remise à jour quotidiennement. On y trouve les réussites, les bons points et que les bons plans de Java. Cette page est orientée particulièrement sur les solutions Java pour les gros systèmes et pour les réseaux (applets). (site US)

[14] · Java Ressources List - http://www.sun.com/java/list.html
Un florilège de petits programmes pour augmenter les capacités de son JDK ou à trouver des solutions professionnelles à un problème. C'est un peu l'anarchie sur ce site où l'ordre n'est pas à l'ordre du jour mais c'est une façon de découvrir beaucoup d'URL qui ne seraient pas répertoriés. (site US)

[15] · The AWT Home Page - http://www.java.sun.com/products/jdk/awt/
La page dédiée par les développeurs de Java à la classe AWT, Abstract Windowing Toolkit.

[16] · JavaWorld - http://www.javaworld.com
Le magazine on-line de l'évolution d'Internet. Globalement mis à jour comme un mensuel, il publie des articles tous les jours et on peut être avertis des nouveaux articles par e-mail. (site US)

[17] · Java Cyber Club France -
http://www.labri.u-bordeaux.fr/Equipe/ALiENor/membre/chaumett/java/
Le site le plus complet sur java en Français.

[18] · Programmation Java - http://www.fisystem.fr/java/javabook.htm
La transcription « on-line » du livre de Jean-François Macary et Cédric Nicolas. Ils portent sur Java un regard de développeurs enthousiastes, mais avant tout soucieux d'efficacité professionnelle. Dans ce livre, ils enseignent les concepts objet, font le tour des librairies, démêlent les mécanismes des threads et des exceptions,comparent applets et applications, parlent de de la sécurité de Java mais restent toujours sur des considérations générales sur le langage.