Outil d'installation d'applications sur une grille de cartes à puce de type Java à API respectant GlobalPlatform - Partie 1

Rapport projet

Responsables scientifiques :
Serge CHAUMETTE
Achraf KARRAY
Responsable pédagogique :
Mohamed MOSBAH

ABDELKHALEK Rached BEN MBARKA Moez
DIYAB Rachid ESSABIR Ayoub
GATTI Christophe Renteria Morales Eder
TRIKI Hamza

27 avril 2006

Table des matières

Chapitre 1

Introduction

Dans le domaine des cartes à puce, la spécification et le développement des interfaces communicantes sont toujours des phases délicates qui nécessitent la mise en commun de compétences fortes en sécurité informatique, en électronique mais également en cryptologie.

Ce projet de fin d'année nous invite donc dans le cadre d'une étude sur grille de 32 cartes à puces, d'implémenter une API permettant de déployer des applets Java sur l'ensemble de ces cartes, et ce, conformément aux spécifications GlobalPlatform. Cette API mise en oeuvre devra donc répondre à un certain nombre de critères en matière de besoins fonctionnels définis par les clients.

Pour ce faire, nous utiliserons le langage Java, s'adaptant aux JavaCards que nous utiliserons, qui offre une grande portabilité comme étant un langage interprété. Nous disposons de stations de travail et d'outils spécifiques nous permettant de mettre en oeuvre le bon fonctionnement de notre projet.

Dans un premier temps, nous listerons les besoins que devra satisfaire notre API, puis nous montrerons l'organisation de notre travail, et enfin nous nous pencherons sur l'aspect implémentation proprement dit de notre API, a savoir les paquetages crées, le contrôle d'erreur fourni, et les tests unitaires répondants à nos méthodes.

Chapitre 2

Contexte du projet

La carte à puce devient de nos jours de plus en plus populaire et se découvre chaque jour de nouvelles applications. Devant cet état des choses, plusieurs industriels et acteurs du domaine des cartes à puce, regroupés en consortiums, fournissent un effort de standardisation, le but étant de pouvoir loger plusieurs applications (de différents fournisseurs) sur une même carte avec des contraintes de sécurité fortes.

2.1 La carte à puce

Une carte à puce est une carte plastique intégrant un circuit électronique capable de manipuler des informations de manière sécurisée. Il existe plusieurs types de cartes selon qu'elles intègrent ou pas un microprocesseur, qu'elle soient à contact ou pas ...

2.1.1 Architecture

Nous nous intéressons dans cette section particulièrement à l'architecture de la carte à microprocesseur et à contact (c'est celle que nous utilisons dans notre projet).

Communication

Ce type de carte communique via un micro contact relié à la puce (microchip) par des fils d'or. Cet assemblage se nomme micromodule (cf Fig.2.1).

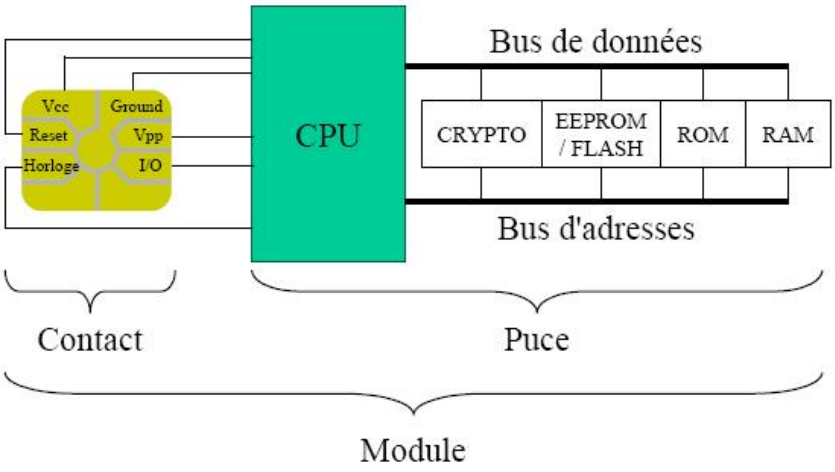


FIG. 2.1 – Le micromodule

Le micromodule est placé dans le corps de carte et correspond à la partie intelligente de la carte. Ces cartes utilisent une communication série via les huit contacts définis dans le standard ISO 7816. En pratique, seulement cinq de ces huit contacts sont réellement utilisés pour réaliser la communication série. Le contact Vpp n'est quasiment plus utilisé puisqu'il servait juste à fournir une alimentation suffisante lors de la programmation de mémoires utilisant des technologies anciennes et donc consommant beaucoup d'énergie. En revanche, les deux derniers contacts qui étaient jusqu'alors réservés à une utilisation future sont aujourd'hui utilisés pour communiquer en USB (Universal Serial Bus). Il s'agit donc toujours d'une communication série, mais universelle et beaucoup plus rapide que le mode de communication classique. Les cartes utilisant cette technologie intègrent donc directement la gestion de l'USB reléguant ainsi le lecteur à un simple adaptateur entre la carte et le PC. Cela permet d'avoir des lecteurs de faible coût.

Microprocesseur et mémoire

Les puces des cartes considérées embarquent un microprocesseur(cf Fig. 2.2) de dimensions extrêmement réduites travaillant à des fréquences comprises entre 5 et 40 Mhz.

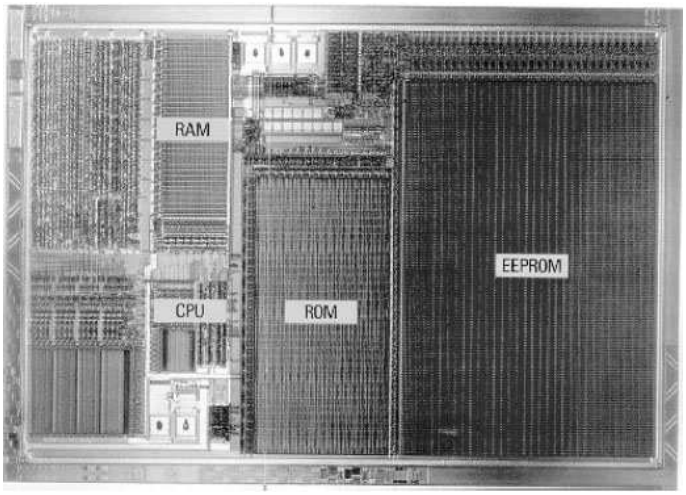


FIG. 2.2 – Le microprocesseur d'une carte à puce

Ces processeurs intègrent 3 types de mémoires :
- La ROM (Read Only Memory) : Programmée en usine, persistante et figée. Taille : 32 ou 64 Ko.
- La RAM (Random Access Memory) : Mémoire volatile, temps d'accès rapide. Taille : de 1 à 4 Ko.
- L'EEPROM (Electrical Ereasable Programmable Read Only Memory) : Mémoire persistante mais dont le contenu est modifiable. Taille : varie de 16 à 64 voire 256 Ko. D'autres types de mémoires possédant les mêmes propriétés existent et commencent à être utilisées pour les cartes à puce (Flash, FeRAM, MRAM...).

Outre la mémoire, ces puces peuvent possèder également un coprocesseur cryptographique qui réalise les opérations de chiffrement et de déchiffrement de manière câblée améliorant ainsi grandement les performances. Ce coprocesseur cryptographique est associé à un générateur de nombres aléatoires RNG (Random Number Generator).

2.1.2 Applications

Les principaux secteurs qui utilisent actuellement la carte à puce sont :
- l'industrie des télécommunications avec, par exemple, les cartes téléphoniques prépayées (comme par exemple les cartes téléphoniques classiques qui sont des cartes à mémoires avec contact) ou bien avec les cartes SIMs insérées dans les téléphones GSM ;
- l'industrie bancaire et monétaire avec les cartes de crédits (e.g. Europay, MasterCard,Visa) et les porte-monnaie électroniques (e.g. Monéo) ;
- le secteur de la santé (e.g. la carte Vitale) ;
- l'industrie audiovisuelle avec la télévision à péage ;
- l'industrie du transport avec les cartes sans contact pour les transports en commun, ou pour le transport routier le remplacement des chronotachygraphes par des cartes à puce ;
- l'industrie du contrôle d'accès physique des personnes aux locaux utilise de plus en plus la carte sans contact ;

Mais de nouvelles applications pour la carte à puce sont aujourd'hui à l'étude pour :
- l'authentification (e.g. des sites sur l'Internet) ;

- les applications de fidélisation (e.g. l'accumulation de miles de voyage en avion gratuits à chaque transaction) ;
- les jeux comme le pocket-gaming (qui consiste en des terminaux de poche comme le deviennent les téléphones ou les consoles portables pour jouer par exemple à des jeux de casino).

Le domaine principal qui permettra à la carte à puce de s'imposer est l'Internet qui au travers des e-services exigent de plus en plus fréquemment une identification et une sécurisation des transactions. Ainsi, la carte peut être utilisée pour ces services en stockant par exemple les clefs servant à l'authentification et à la sécurisation des communications. Quelques exemples de e-services sont :

- le e-commerce ;
- la banque distante ;
- le e-courrier ;
- le télétravail.

2.2 La technologie Java Card

2.2.1 Présentation

La technologie Java Card permet aux cartes à puce et à d'autres périphériques à mémoire limitée de faire fonctionner des applications écrites en langage Java. Une Java Card est donc une carte à puce sur laquelle il est possible de charger et d'exécuter des programmes Java, appelés applets. Contrairement aux cartes à puce traditionnelles, les programmes exécutés par la carte ne sont pas forcément fournis par l'émetteur de la carte.

De manière synthétique, on peut dire que la technologie Java Card, telle qu'elle est vendue par Sun microsystems, définit une plate-forme sécurisée pour cartes à puce, portable et multi-application qui intègre beaucoup des avantages du langage Java.

2.2.2 Les avantages de Java Card

Les avantages de la technologie Java Card pour les développeurs d'applications pour cartes à puce sont pour l'essentiel les mêmes que ceux que Java a pu apporter par rapport aux langages de programmation classiques.
Évidemment un des buts premiers de la technologie Java Card a été d'apporter l'indépendance des applications par rapport au matériel en s'inspirant des principes du langage Java où une application est exécutable sur n'importe quel système puisque son code est exécuté par la machine virtuelle Java.

2.2.3 Les spécifités de Java Card

Dans une station de travail, la JVM (Java Virtual Machine) se comporte comme un processeur virtuel et les données et les objets dont elle a besoin sont créés dans la RAM. Sur la plateforme Java Card, les informations de la machine virtuelle sont préservées au travers des sessions de communication avec le lecteur grâce à la mémoire persistante. Donc, contrairement à la machine virtuelle java classique qui voit son cycle limité à une exécution, la JCVM possède un cycle de vie identique à celui de la carte.

2.3 GlobalPlatform

2.3.1 Présentation

Le but des spécifications GlobalPlatform était d'apporter un standard qui manquait jusqu'alors au monde des cartes : une spécification pour gérer les cartes de façon indépendante du matériel, des vendeurs et des applications.

En d'autres termes, GlobalPlatform définit des spécifications flexibles et efficaces pour que des émetteurs de cartes puissent créer des systèmes utilisant des cartes multi-applicatives. Ainsi, ces spécifications leur permettent de choisir la technologie de cartes multi-applicatives correspondant le plus à leurs besoins actuels tout en s'assurant qu'ils pourront également utiliser une technologie de cartes différente dans l'avenir.

2.3.2 Architecture GlobalPlatform

L'architecture GlobalPlatform comprend un certain nombre de composants permettant de s'abstraire du matériel et du vendeur grâce à des interfaces pour les applications et pour le système de gestion hors carte.
Afin d'être portables toutes les applications doivent être implantés au sein d'un environnement d'exécution sécurisé utilisant une API indépendante du matériel. Néanmoins, GlobalPlatform n'impose aucune technologie particulière quant à l'environnement d'exécution. GlobalPlatform peut donc être utilisé pour n'importe quelle technologie de carte multi-applicative.
Les APIs GlobalPlatform fournissent des services aux applications (vérification, personnalisation, services sécuritaires). Elles offrent aussi aux applications des services de gestion du contenu de la carte.

2.3.3 Mécanismes de sécurité

La plupart des mécanismes de sécurité que propose GlobalPlatform sont d'ordre cryptographique. Ils spécifient des méthodes :

- pour sécuriser les communications ;
- pour s'assurer que les applications chargées sont officiellement signées ;
- pour vérifier l'identité du porteur de carte.

Pour sécuriser les échanges entre la carte et une entité extérieure, GlobalPlatform spécifie les mécanismes :

- d'authentification mutuelle ;
- de canal sécurisé.

Chapitre 3

Outils et organisation

Nous abordons dans cette partie les outils mis en oeuvre au cours de ce projet, ainsi que les aspects formels et organisationnels du déroulement et de la réalisation du projet.

3.1 Outils utilisés

3.1.1 Eclipse

Eclipse est une plate-forme java open source dédiée pour le développement d'outils. Elle supporte déjà de nombreux outils de développement de haut niveau tres complets : un IDE complet Java (JDT) et C/C++ (CDT), un environnement de création de plug-in (PDE) ...

La plate-forme est facilement étendue par de multiples plugs-ins astucieusement intégrés. La plate-forme Eclipse s'est développée initialement autour du Projet Eclipse et de l'IDE JDT, mais aujourd'hui de nouveaux projets connexes voient régulièrement le jour.

Le JDT dEclipse gère ce qu'on appelle un compilateur java incrémental. Une sorte de pré-compilation permanente. Ainsi toute faute de syntaxe ou autre est détectée et signalée au moment de lécriture du code. Des corrections sont alors proposées si lon le désire.

Parmi les plug-ins étendant les possibilités d'Eclipse, l'environnement de développement ciblant les JavaCards proposé par IBM : JCOP (JavaCard Open Platform)

3.1.2 JCOP (JavaCard Open Platform)

Les outils JCOP sont un plug-in pour éclipse et son environment de développement intégré (IDE) Java Development Tooling (JDT) conçu pour faciliter le développement d'applets JavaCard. Les outils JCOP offrent :

- un convertisseur de byte code Java Card.
- Un simulateur de cartes Java Card.
- Le JCOP Shell, une ligne de commande interactive pour communiquer avec le simulateur ou les cartes à puce.
- ...

3.1.3 Wiki en ligne

Dans le but de faciliter la communication entre notre équipe, le responsable pédagogique et le client, Un wiki est mis en ligne. Il présente, en plus de divers

liens et documents utiles pour notre projet, les différents rapports d'avancement (rapports hebdomadaires) et des réunions : avec les clients, avec le responsable pédagogique et avec le groupe développant l'interface. Il contient aussi la répartition du travail pour les différentes phases du projet.

Suite à la demande des clients, le wiki a été protégé par des comptes d'accès. Le lien vers le wiki : http ://pfa.fisoft1.com

3.1.4 CVS

La gestion concourante des versions a été effectuée à l'aide de CVS.

3.2 Organisation générale du travail

On a essayé de mettre en place une méthodologie de travail basée sur une répartition tournante des rôles :

- Un secrétaire : Un secrétaire est fixé pour chaque semaine. Le secrétaire est responsable de :
  - Rédiger le compte rendu pour chaque réunion ; item Rédiger le compte rendu hebdomadaire ;
  - Noter les absences aux réunions.
- Coordinateur : Parallèlement un coordinateur est fixé pour chaque semaine. Le coordinateur est responsable de la répartition et le suivi du travail réalisé. Il fait le lien entre les différents binômes.
- Responsables communication et sources : Un responsable communication permanent est choisi.Il est responsable de communiquer avec les responsables scientifiques et pédagogique(fixer des rdv, ...). De même un responsable codes sources permanent, chargé de la gestion des sources a été choisi.

L'organisation du travail, s'est voulu, par ailleurs, fidèle autant que possible à l'approche de l'extreme programming.

3.2.1 Extreme Programing

[1]

L'extreme programming (ou encore l'XP) est une approche réfléchie et disciplinée du développement logiciel.

Elle s'appuie sur :

- une forte réactivité au changement des besoins du client (ou changements technologiques) au plus tôt dans le cycle de vie du logiciel ;
- un travail d'équipe : combinaison de l'effort des développeurs, de l'encadrement et du client ;
- la qualité du code : elle garantit un gain d'argent pour le client ;
- la qualité des tests effectués au plus tôt ; une intégration permanente.

L'XP est préconisée dans un milieu où les besoins changent, dès lors, les fonctionnalités du système sont en butte à des changements réguliers.Cette méthodologie est adaptée aux petits groupes de projet comprenant de 2 à 10 personnes (les petites équipes sont plus efficaces que les grosses équipes quand le risque est grand et les besoins dynamiques).

Notre projet, de part la façon dont il est présenté (évolution quasi quotidienne des fonctionnalités et des besoins), la taille de l'équipe de développement (7 per-

[1] Cettez définition est inspirée de celle présentée dasn www.idealix.org

sonnes) et l'utilisation intensive de tests est particulièrement adapté à l'usage de cette méthode.

Chapitre 4

Rappel des besoins

4.1 Objectif généraux

Le but du projet est le développement d'une API respectant les spécifications GlobalPlatform et permettant de réaliser les opérations élémentaires sur une carte.

Actuellement, des outils spécifiques à chaque type de carte sont utilisés. Le but de respecter les normes GlobalPlatform est d'avoir une API valide pour tous les cartes à puce les supportant.

L'API à réaliser devra offrir toutes les fonctionnalités de GlobalPlatform à l'exception des routines de sécurité.

Les principaux services que devra offrir notre API :

- Initialiser un canal de communication avec la carte.
- Installer une application.
- Supprimer une application.
- Sélectionner une application.
- lister les applications disponibles sur la carte.

4.2 Diagramme cas d'utilisation

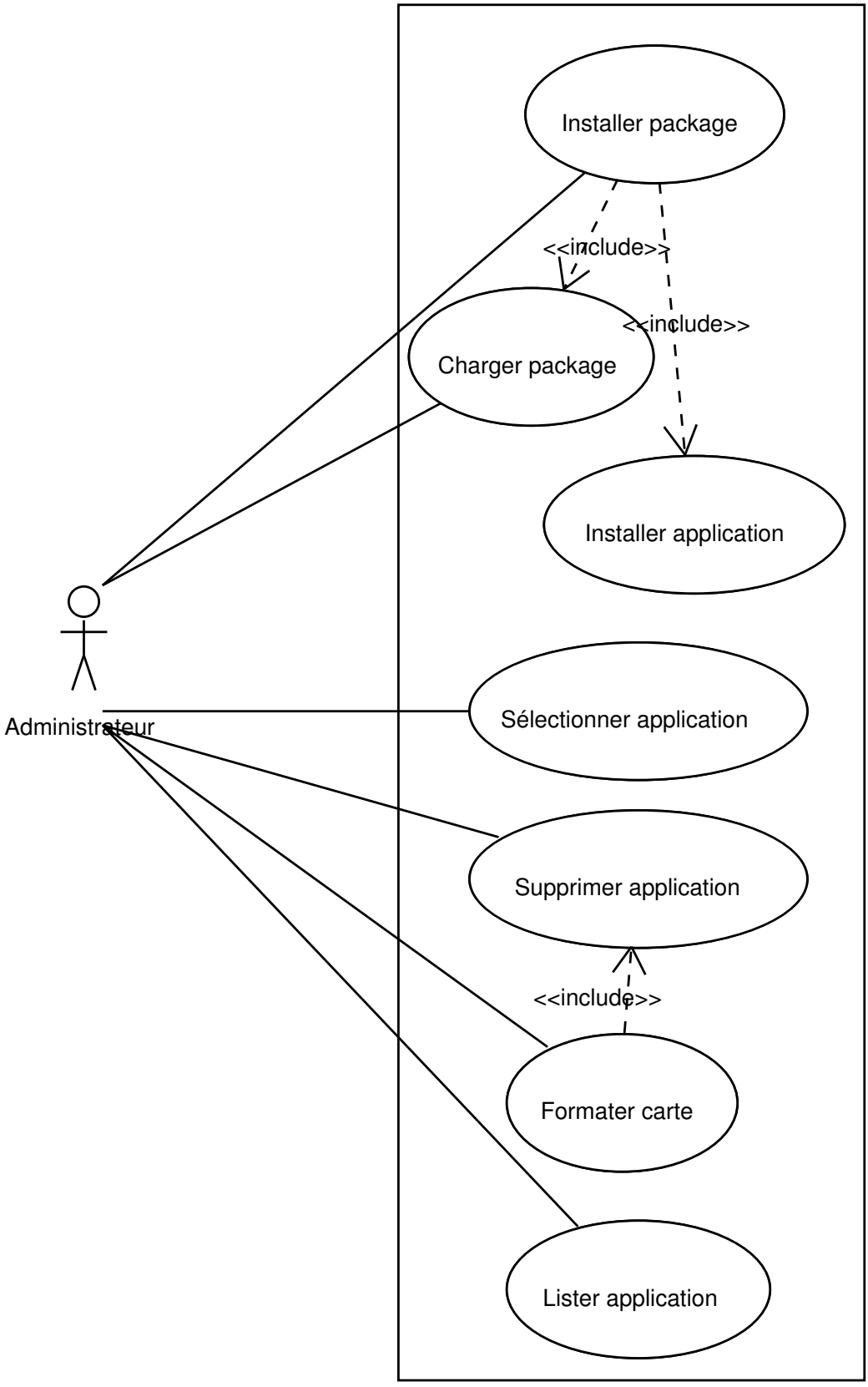


FIG. 4.1 – Diagramme cas d'utilisation

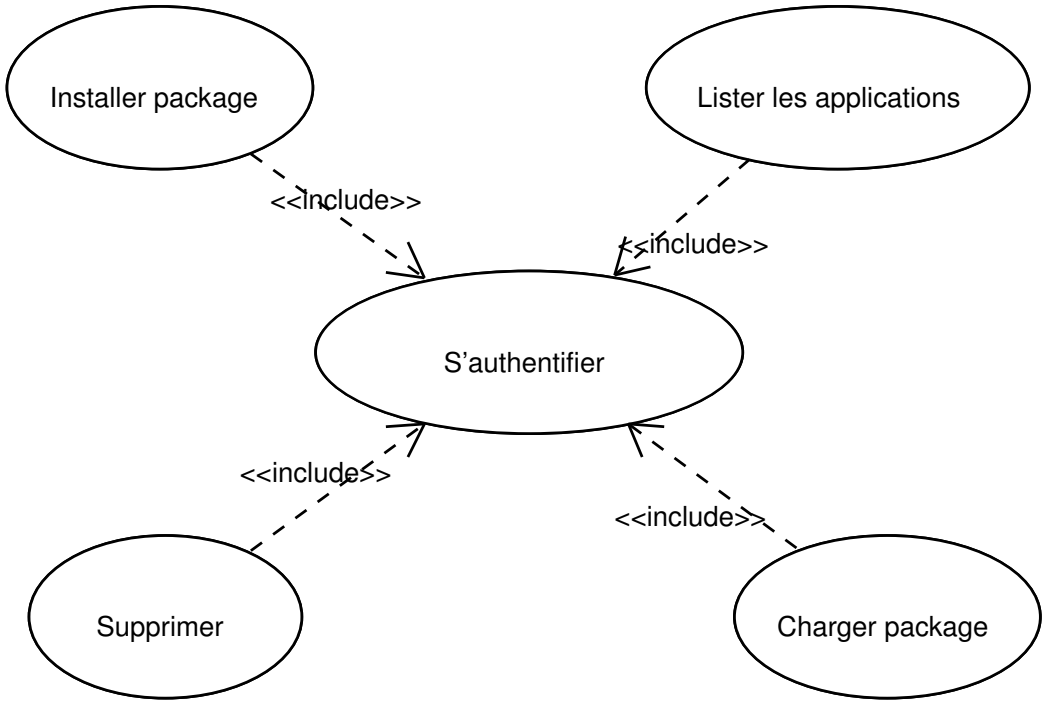


FIG. 4.2 – Suite diagramme cas d'utilisation

Description des cas d'utilisation :

Titre	**Charger package**
Résumé	Charger un package à partir d'un fichier cap sur la carte.
Acteurs	L'administrateur et *cardmanager*.
Précondition	Le *cardmanager* est sélectionné.L'authentification mutuelle avec la carte validée.
Scénario nominal	Envoyer la commande Installer pour charger($InstallforLoad$) avec l'AID du package. Envoyer les blocs de données constituant le package

Titre	**Installer package**
Résumé	Installer les applications contenues dans un package sur la carte.
Acteurs	L'administrateur et le *cardmanager*.
Précondition	Le *cardmanager* est sélectionné.L'authentification mutuelle avec la carte validée. Le package a été précédemment chargé.
Scénario nominal	Envoyer la commande Install pour installer (Install For Install) pour chaque application contenue dans le package.

Titre	**Supprimer package**
Résumé	Supprimer un package de la carte
Acteurs	L'administrateur et le *cardmanager*.
Précondition	Le *cardmanager* est sélectionné. L'authentification mutuelle avec la carte validée. Le package est présent sur la carte.
Scénario nominal	Supprimer toutes les applications associées au package. Supprimer le package.

Titre	**Sélectionner une application**
Résumé	Sélectionner une application sur la carte.
Acteurs	L'administrateur et le *cardmanager*
Précondition	L'application est présente sur la carte.
Scénario nominal	Envoyer la commande Sélectionner avec l'AID de l'application à sélectionner.

Titre	**S'authentifier**
Résumé	Établir l'authentification mutuelle avec la carte.
Acteurs	L'administrateur et le *cardmanager*
Précondition	Le *cardmanager* est sélectionné. La carte et l'administrateur partagent un ensemble de clés communes.
Scénario nominal	- L'administrateur envoie à la carte un nombre aléatoire. et choisie les clés à utiliser. - La carte génère aussi un nombre aléatoire qu'elle utilise avec un algorithme de cryptage pour générer un *cardcryptogram* et envoie ces données à la carte - La carte utilisant le même algorithme vérifie la validité du *cardcryptogram* puis calcule et envoie son *hostcryptogram*. - La carte vérifie le *hostcryptogram* et valide le processus d'authentification.

Chapitre 5

Description des données

5.1 Le standard APDU

5.1.1 Communication entre la carte et l'environnement extérieur

La carte à puce peut être assimilée à un ordinateur. Pour interagir avec l'environnement extérieur, elle a besoin de communiquer.

La carte ne communique pas directement avec l'ordinateur, mais avec un lecteur de carte qui lui fournit l'alimentation nécessaire pour son fonctionnement.

La communication entre la carte et le lecteur se fait en mode $half - duplex$. A un instant donné seule la carte ou le terminal envoie des données sur le support, mais pas les deux en même temps.

Ce mécanisme nécessite une synchronisation, pour savoir qui est autorisé à lire et qui est autorisé à écrire sur le support. Pour ce faire, la communication suit un modèle de communication de type $maitre - esclave$, le lecteur étant le maître.

5.1.2 Pile protocolaire

la communication lecteur/carte s'effectue au travers d'une pile protocolaire. Cette pile s'aligne sur le modèle OSI (Open Systems Interconnexion), elle est représentée dans la figure suivante.

Couche application : Application Protocole Data Unit (APDU)
Couche transport : Transport Protocole Data Unit (TPDU)
Couche Physique

TAB. 5.1 – Pile protocolaire

La couche physique est normalisée par ISO 7816-3 qui décrit la fréquence d'horloge, la vitesse de communication...
C'est aussi dans l'ISO 7816-3 que sont décrit les échanges de données pour les protocoles de transmission.

En-tête obligatoire				Corps optionnel		
CLA	INS	P1	P2	Lc	Champ données	Le

TAB. 5.2 – Format de la commande APDU

Champ optionnel	En-queue obligatoire	
Champ données	SW1	SW2

TAB. 5.3 – Format de la réponse APDU

5.1.3 La couche application : APDU

Il existe deux types d'échange de données pour cette couche :

C-APDU : La commande APDU, qui est émise par le lecteur en direction de la carte.

R-APDU : La réponse APDU, qui, elle transite de la carte vers le lecteur.

Rappelons que la carte est toujours en attente d'une commande APDU. Une réponse APDU aura donc lieu en retour d'une commande APDU.

La commande APDU

La commande APDU est constituée par :

- Une en-tête obligatoire de quatre octets :

  CLA : L'octet classe identifie la catégorie de la commande et de la réponse APDUs.

  INS : L'octet instruction spécifie l'instruction de commande.

  P1,P2 : Deux octets utilisées pour paramétrer l'instruction.

- Un corps optionnel de taille variable :

  Lc : L'octet qui spécifie la taille de champ de données

  Champ de données : Contient les données à envoyer à la carte pour exécuter l'instruction spécifiée dans l'en-tête.

  Le : L'octet qui spécifie le nombre d'octets attendus par le lecteur pour le champ de données dans la réponse APDU de la carte.

La réponse APDU

- Un corps optionnel de taille variable :

  Champ données : de taille **Le** déterminée dans la commande APDU correspondante

- Une en-queue obligatoire de deux octets :

  SW1 et SW2 : Status word, deux champs d'un octet précisant l'état de la carte après l'exécution de la commande APDU.

5.2 Les fichiers CAP

5.2.1 La technologie Java Card

La technologie Java permet aux cartes de charger et d'exécuter des applications écrites en Java. Les programmes exécutés sur la carte ne sont pas forcément fournis

par l'émetteur de la carte.

Un des buts de la technologie Java Card a été d'apporter l'indépendance des applications par rapport au matériel. Le modèle est le langage Java où une application est exécutable sur n'importe quel système, puisque son code est précompilé en *bytecode* puis exécuté sur une machine virtuelle.

Un des plus grands défis de la technologie Java Card a été de s'adapter aux contraintes liées à l'espace mémoire pour conserver assez de place afin d'embarquer les applications.

La solution choisie a été de supporter seulement un sous-ensemble des caractéristiques du langage Java, et de découper la machine virtuelle Java (JCVM), en deux parties :

- Une partie qui s'exécute sur la carte, elle comprend principalement l'interprète de *bytecode*
- Une partie qui s'exécute en dehors de la carte.

Ainsi, depuis la version 2.1 des spécifications JavaCard, le modèle de la Java Card suit le modèle suivant :

Applications
Java Card APIs : Présentes sur toutes les cartes
Java Card Virtual Machine Interpréteur de bytecode
Hardware de la carte et son système natif

TAB. 5.4 – Architecture de la Java Card

5.2.2 La machine virtuelle Java Card JCVM

La machine virtuelle Java Card a une architecture comparable a celle de la machine virtuelle Java ; la différence est que la JCVM est découpée en deux parties :

- Une partie embarquée sur la carte qui inclue l'interpréteur de bytecode
- Une partie hors carte qui tourne sur une station de travail et qui comprend le convertisseur

Le convertisseur charge et traite les fichiers *.class* qui composent le package Java à convertir et produit en sortie un fichier **CAP**, qui est une représentation binaire exécutable des classes. Le convertisseur génère également un fichier **export** représentant les APIs publiques du package installé.

Le fichier CAP peut alors être chargé sur la carte à puce pour y être exécuté.

5.2.3 Fichier CAP

Le fichier CAP est une archive utilisant le format JAR pour stocker un ensemble de composants. Chacun de ces composants est stocké comme un fichier individuel $file.cap$. Le format de chacun de ces composant suit le format suivant :

Étiquette	Taille	Données

TAB. 5.5 – Format d'un composant du package

Chaque composant décrit un aspect du contenu d'un fichier CAP. Avec les spécifications actuelles, il y a 12 composants avec 3 qui sont optionnels, avec la possibilité de créer de nouveaux composants pour des besoins spécifiques.

Voici la liste des composants officiels avec leur étiquette et une brève description :

1. **header** : contient l'AID du package et les numéros de versions de la machine virtuelle pouvant supporter le package.
2. **directory** : contient la liste des autres composants avec leurs taille respectives
3. **applet, optionnel** : contient la liste des applications avec leurs AID et une référence sur la méthode *install*(), la première méthode appelée par l' interpréteur pour installer l'application.
4. **import** : contient la liste des packages référencés par ce package.
5. **constant pool** : contient la table de références aux classes, aux méthodes et aux champs, respectivement, dans les composants de classes, de méthodes et de champs.
6. **class** : contient la table des classes et interfaces.
7. **method** : contient la code des méthodes de classes.
8. **static field** : contient les valeurs initiales de tous les champs statics du package.
9. **reference location** : contient la liste des références relatives dans le code des méthodes à transformer en références absolues au chargement.
10. **export : optionnel** : contient la liste des éléments de ce package que les autres peuvent référencer directement.
11. **descriptor** : contient des informations de typage.
12. **debug : optionnel** : contient toutes les méta-données nécessaires pour déboguer le package.

Chapitre 6

Package globalPlatform

A la demande du client, un package spécial, se voulant aussi fidèle que possible aux normes GlobalPlatform, permet la construction des APDUs à envoyer aux lecteurs et implémente les méthodes cryptologiques nécessaires à la connexion avec la carte.

6.1 Diagramme des classes

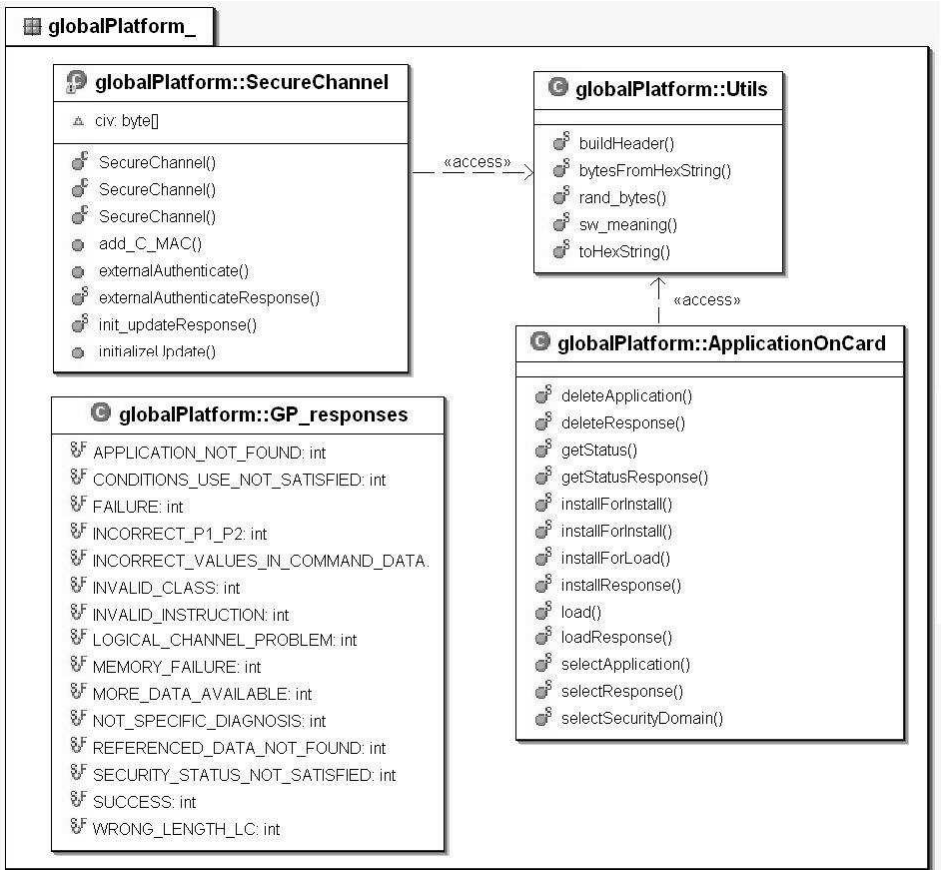


FIG. 6.1 – Package globalPlatform

6.2 Les commandes GlobalPlatForm

Les spécifications GlobalPlatform définissent de manière non ambigu la construction des différentes commandes GlobalPlatform en spécifiant le contenu des différents champs de la commande APDU. Selon ces spécifications certaines commandes supportent plusieurs valeurs possibles pour les champs de paramètrage $P1$ et $P2$, d'autres fixent les valeurs de ces champs. Les méthodes implémentées prennent en paramètres seulement les champs paramétrables.

D'autre part, les méthodes de la classe *ApplicationOnCard* ne font que construire l'APDU. L'envoie à la carte et la réception de la réponse, est fait par les méthodes du package $offCard$.

6.2.1 selectApplication

Cette méthode permet de construire l' APDU qui va servir à selectionner une application présente sur la carte à partir de son AID.

Prototype de la fonction :

```
public static byte[] selectApplication(byte[] aid, byte p2)
```

selectApplication prend en paramètre un tableau de byte pour l'AID de l'application à selectionner sur la carte ainsi que le paramètre $P2$. Ce paramètre peut avoir deux valeurs se distinguant par le deuxième bit :
- '0' : Sélectionner la première occurrence.
- '1' : Sélectionner l'occurrence suivante.

La méthode renvoie un tableau de byte qui contient l'APDU correspondante à la selection.

A noter que cette commande ne nécessite pas une authentification préalable avec la carte mais simplement que le *cardmanager* soit préalablement sélectionné.

6.2.2 La commande deleteApplication

Cette commande permet de construire l' APDU qui va servir à supprimer une application présente sur la carte à partir de son AID.

Prototype de la fonction :

```
public static byte[] deleteApplication(byte[] aid, byte p2)
```

deleteApplication prend en paramètre un tableau de byte présentant l' AID de l'application à supprimer sur la carte ainsi qu'un paramètre $P2$. Celui-ci peut avoir deux valeurs se distinguant par le dernier bit :
- '0' : Supprimer seulement l'objet ayant l'AID envoyé.
- '1' : Supprimer en plus les objets liés.

Contrairement à la commande de sélection, cette commande ne peut pas être envoyé à la carte sans une authentification préalable. Dans le cas contraire, Le code correspondant à l'erreur : "conditions of use not satisfied"sera renvoyée par la carte.

6.2.3 La commande installForLoad

Cette commande permet de construire l' APDU qui va servir à préparer le chargement d'un package sur la carte à partir de son AID.

Prototype de la fonction :

```
public static byte[] installForLoad(byte[] loadFileAID,
 byte[] securityDomainAID,
 byte[] loadFileDatablockHash,
 byte[] loadParametersField,
 byte[] loadToken) ;
```

Le paramètre $securityDomainAID$ correspond à l'AID du *cardmanager* .

Les autres paramètres caractéristiques du package à installer sont récupérés dans la méthode appelante à partir du fichier .cap(voir 5.2).

Comme toutes les fonctions de gestion de contenu, cette commande suppose que l'authentification mutuelle avec la carte à été réalisée avec succès, ainsi que la sélection du *cardmanager*.

La commande *installForLoad* permet de préparer la carte au chargement d'un package. Elle est généralement suivie d'une suite de commandes *LOAD* pour le chargement des données.

6.2.4 La commande Load

Cette commande permet de construire l'APDU qui va servir à transférer un package sur la carte, en envoyant les morceaux successifs de données.

Prototype de la fonction :

```
public static byte[] load(byte P1, byte P2, int Lc, byte[]loadData)
```

LOAD suppose, que la commande précédente envoyée à la carte est $InstallForLoad$. Chaque bloc load doit être envoyé avec son numéro d'ordre dans la totalité des blocs. Le comptage commence par '00' et l'incrémentation se fait nécessairement par un. Ce numéro d'ordre est codé dans $P2$.

Le paramètre $P1$ code, sur son dernier bit, s'il s'agit du dernier bloc ('1')ou si la carte doit attendre un autre bloc('0').

D'autre part, la taille maximale pour un bloc est caractéristique de la carte, et dans tous les cas elle ne doit pas dépasser 255 octets, étant donnée que la taille du champ *Data* est codé sur un seul octet.

6.2.5 La commande installForInstall

Cette commande permet de construire l' APDU qui va servir à installer une application sur la carte à partir de son AID .

Prototype de la fonction

```
public static byte[] installForInstall(byte[] executableLoadFileAID,
byte[] executableModuleAID,
byte[] applicationAID,
byte applicationPrivileges,
byte[] installParameters)
```

installForInstall nécessite, en plus qu'une authentification préalable, que l'application soit précédemment chargée sur la carte.

Elle prend en paramètre l'AID du package précédemment chargé ainsi que l'AID de l'application à installer.

Le parmètre *applicationPrivileges* code les privilèges de l'application à installer. Ce codage est spécifié dans les spécifications Global Platform. Quand au paramètre *installParameters*, il est optionnel.

6.2.6 La commande getStatus

Cette commande permet de construire l' APDU qui va servir à obtenir des informations sur la carte, le CardManager ou les applications présentes sur la carte.

Prototype de la fonction

```
public static byte[] getStatus(byte p1, byte p2, byte[] data)
```

Le tableau d'octets *data* code le critère de recherche à utiliser. Par exemple pour récupérer des informations sur une application sur la carte, on code sur ce paramètre l'AID. Par contre si on souhaite recevoir des informations sur toutes les applications de la carte il suffit de ne spécifier aucun critère de recherche.

Le parmètre $P1$ code le domaine sur lequel la recherche doit être effectué sur la carte. Exemples :

- '0x40' : On inclue seulement les applications et le domaine de sécurité.
- '0x80' : On retourne seulement des informations sur le domaine de sécurité. Avec cette valeur, le critère de recherche est ignoré.

Quand à $P2$:
- Le premier bit code si on attend la première occurrence de la réponse ou la suivante. Le résultat de la recherche dans la carte peut ne pas tenir sur une seule réponse.
- Sur Le deuxième bit on code la forme souhaitée de la réponse. Les spécifications Global Platforme proposent plusieurs formes de réponses dépendant aussi du domaine de la recherche codé en $P1$.

A noter enfin que cette commande ne peut être envoyée a la carte que si l'authentification mutuelle ainsi que la sélection du *cardmanager* ont été réalisés.

6.3 Initialisation du canal sécurisé(Secure Channel Protocol 01-SCP01)

Trois niveaux de sécurité sont supportés par SCP01 :

1. **Authentification mutuelle :** La carte et l'utilisateur vérifient qu'ils partagent les même clés secrêtes.
2. **Intégrité des données :** La carte vérifie que le bloc des données reçu est exactement celui envoyé par la carte qui a fait l'authentification. La carte vérifie aussi l'enchaînement des commandes.
3. **Confidentialité :** Les données sont passés chiffrés à la carte.

Seulement les deux premiers niveaux ont été implémenté.
En préalable à toute authentification, les deux entités doivent partager une même information secrète, ici un jeu de clés statiques : S_ENC et S_MAC

6.3.1 La commande INITIALIZE UPDATE

La première étape d'authentification est l'envoie par l'entité extérieur vers la carte d'une commande *APDU* sous le nom de *InitializeUpdate* contenant un nombre aléatoire appelé *HostChallenge*. Cette commande spécifie aussi à la carte

un index et une version correspondants aux clés qui vont être utilisés au cours de l'authentification.

A la réception de cette commande, la carte génère un autre nombre aléatoire appelé $CardCHallenge$. En utilisant ces deux nombre aléatoires($cardChallenge$ et $HostChallenge$) ainsi que les clefs statiques communes, elle produit de nouvelles clés appelées clés de sessions suivant un algorithme commun à la carte et au PC.
Les clés de sessions ainsi créés sont ensuite utilisées par un algorithme cryptographique -lui aussi connu de la carte et du PC- pour produire un valeur appelée $CardCryptogram$.
Les algorithmes utilisés pour le calcul de cette valeurs ainsi que les clés de session seront décrits en détail un peu plus bas(Le PC utilise des algorithmes symétriques pour réaliser sa part de l'authentification).

Le $CardCryptogram$, le $cardChallenge$ ainsi que d'autres informations sont ensuite envoyées dans une commande APDU au PC.

6.3.2 La commande EXTERNAL AUTHENTICATE

Avant de se lancer dans la formation de l'APDU de $EXTERNALAUTHENTICATE$, le PC doit vérifier à partir du $CardCryptogram$ envoyé par la carte que celle ci possède bien les clés secrètes dont l'index a été envoyé dans la commande précédente. Etant donnée que le PC partage maintenant : Les clés statiques, le $HostChallenge$, le $CardChallenge$ ainsi que l'algorithme servi pour le calcule du $CardCryptogram$, il est capable de calculer la même information et donc la comparer avec la valeur reçue.// Une fois cette vérification réussit, on commence la création de l'APDU qui permettra de prouver à la carte qu'on partage les mêmes clés secretes.

Les algorithmes implémentés sont les mêmes utilisés par la carte. La première étape est le calcule des clés de sessions : S_ENC et S_MAC. Ce sont des clé DES de taille 16 octets.
Pour le calcul de ces clé, on besoin d'une clé de dérivation (key derivation data). Celle ci est calculée de la façon suivante :

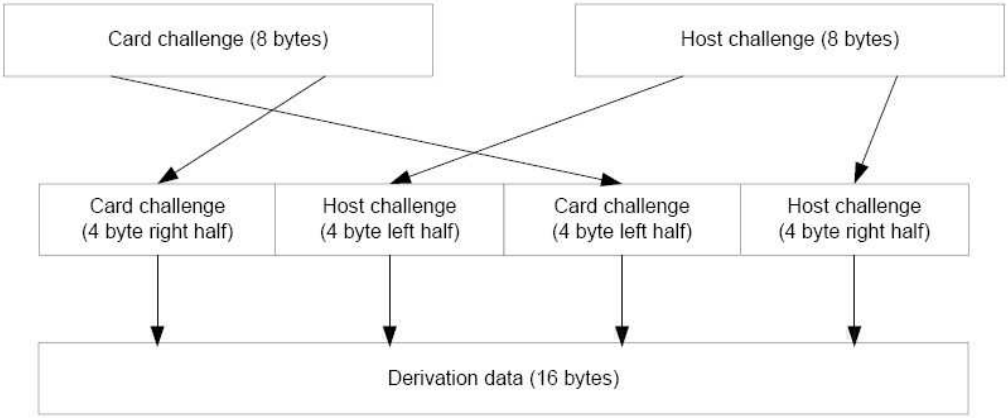


FIG. 6.2 – Création des données de dérivation

Ensuite, les clés DES de session sont générés en appliquant l'algorithme de cryp-

tage triple DES en mode ECB :

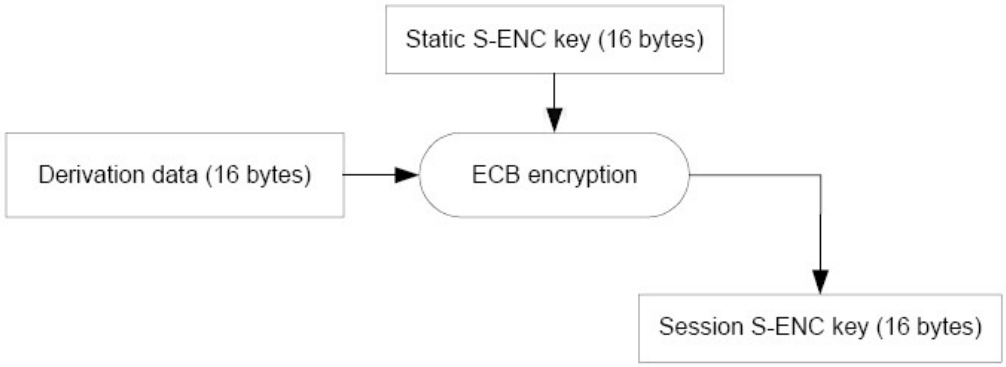


FIG. 6.3 – Création de la clé de session S_ENC

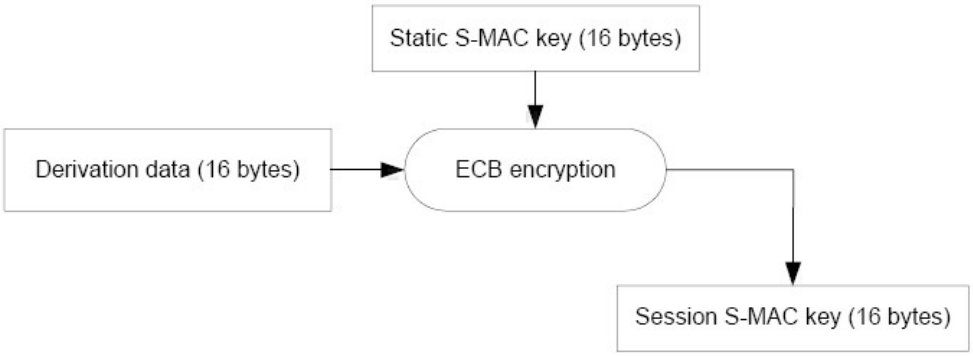


FIG. 6.4 – Création de la clé de session S_MAC

A noter que ces clés de session doivent être générés de nouveau à chaque initialisation de canal sécurisé et qu'elles peuvent être utilisées ultérieurement pour assurer les niveaux de sécurité deux et trois.

L'étape suivante consiste à calculer le *HostCryptogramme*.
Celui-ci est généré de la façon suivante :

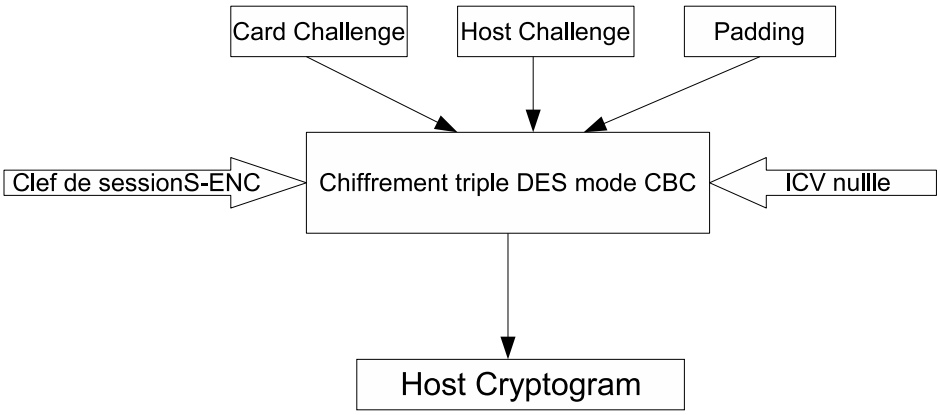


FIG. 6.5 – Génération du Host Cryptogram

Le padding ajouté est le DES Padding. Il est réalisé sur un bloc d'octets de la façon suivante :

1. ajouter '80' à la fin du bloc
2. ajouter des '0' à la fin jusqu'à ce que la taille totale du bloc devient un multiple de 8.

Une fois, le $HostCryptogram$ calculé, l'étape suivante correspond à assurer l'intégrité de la commande APDU de $EXTERNALAUTHENTICATE$ à envoyer à la carte. En effet, cette commande nécessite le deuxième niveau de sécurité. Pour cela, un C_MAC est généré. Celui-ci est le résultat de l'application de l'algorithme de cryptage triple DES en mode CBC sur la totalité de la commande APDU et en utilisant la clé de session S_MAC et un ICV nulle.

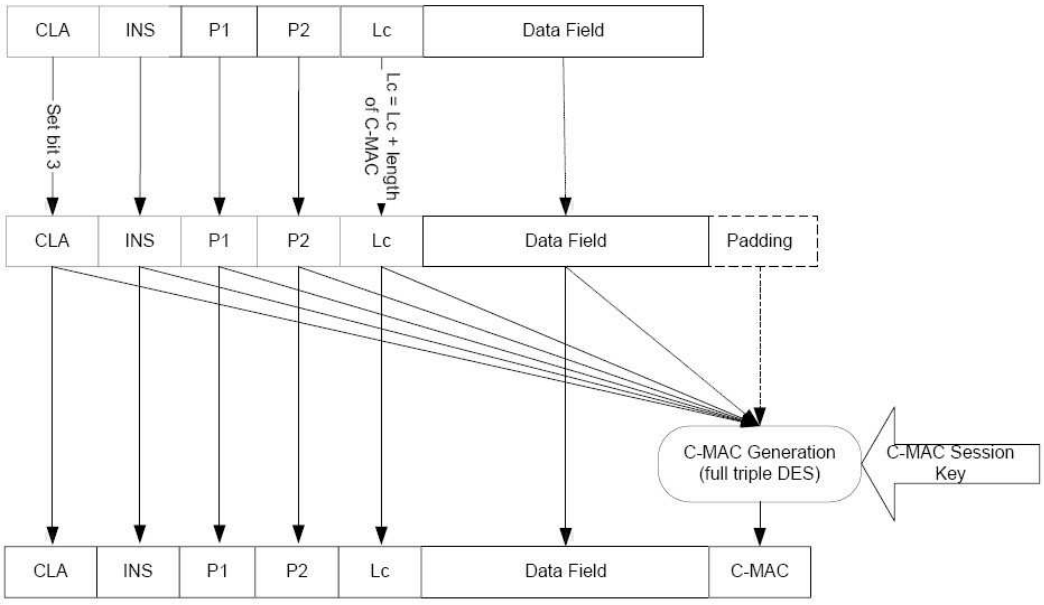


FIG. 6.6 – Génération du C_MAC

D'autre part, pour indiquer au *cardmanager* sur la carte que la commande contient un C_MAC, il faut modifier l'entête de la commande en mettant le troisième bit du champ CLA à 1 et en ajoutant la taille du C-MAC dans le champ Lc.

Maintenant, notre APDU est prête à être envoyer. Le champ $DATA$ contient le $Hostcryptogram$ et le C_MAC.

Elle définit dans son paramètre $P1$ le niveau de sécurité pour les commandes ultérieures que recevra la carte.
Les deux niveaux qui sont supportés :

- '00' : Aucun traitement cryptologique n'est nécessaire pour les commandes suivantes.
- '01' : Les commandes suivantes doivent contenir un $C-MAC$.

La dernière étape du process de l'authentification est effectuée par la carte. Celle-ci calcule de la même façon un $HostCryptogram$ et le compare à celui qui a été reçu.

Pour valider complètement le processus, la carte informera le PC de la réussite ou non du processus.

6.3.3 Intégrité des données

Le deuxième niveau de sécurité permet à la carte de vérifier l'intégrité des commandes APDU envoyées par le PC, ainsi que l'enchaînement de ces commandes.

- L'intégrité est assuré par l'ajout à toute commande un C-MAC. Le calcule de ce dernier est présentée sur la figure 6.6.

- L'enchaînement des commandes est assuré par la valeur de l'ICV utilisé pour le calcule du C-MAC. L'ICV prend pour chaque commande la valeur du dernier C-MAC envoyée et vérifiée par la carte.

6.3.4 Cryptage triple DES

Le calcule des clés de session a nécessité l'utilisation des algorithmes triple DES en modes CBC et ECB.
On utilise les bibliothèques java suivantes :

```
import javax.crypto.Cipher;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;
import javax.crypto.spec.IvParameterSpec;
```

Deux méthodes privées ont été implémentées pour les deux modes :

```
private byte[] des3_ECB_encryption(byte[] keyBuffer, byte[] buffer)
throws Exception

private byte[] des3_CBC_encryption(byte[] keyBuffer, byte[] buffer, IvParameterSpec iv)
throws Exception
```

Le principe étant de générer une clé DES à partir du tableau d'octets $keyBuffer$ puis utiliser cette clé pour crypter le tableau $buffer$.

```
private byte[] des3_ECB_encryption(byte[] keyBuffer, byte[] buffer)
throws Exception{
  DESedeKeySpec key= new DESedeKeySpec(keyBuffer) ;
  SecretKeyFactory kf = SecretKeyFactory.getInstance(ENCRYPTION_ALGO) ;
  Key sk= kf.generateSecret(key) ;
  Cipher c=Cipher.getInstance(ENCRYPTION_ALGO + "/ECB/NoPadding") ;
  c.init(Cipher.ENCRYPT_MODE, sk) ;
  return c.doFinal(buffer) ;
}
```

Chapitre 7

Package cardGridCom

7.1 API JPC/SC

PC/SC est un standard de communication entre un PC et un lecteur de cartes à puce. Ce standard est développé pour assurer la compatibilité entre les cartes à puce, les lecteurs/encodeurs de cartes et les ordinateurs produits par différents constructeurs. Cette nouvelle technologie intégrant carte à puce et PC est construite dans la norme standard ISO 7816, et supporte les applications spécifiques telles que EMV (Europay, MasterCard, Visa) et GSM (Global Standard for Mobile Communication). Elle permet aux développeurs de tirer des avantages sur la portabilité et la sécurité des appareils qui sont deux facteurs importants pour le développement des applications sécurisées.

Les deux implémentations connues de PC/SC sont Microsoft PC/SC et PC/SC-lite. Le dernier est développé sous licence libre et donc plus adapté pour notre projet. Dès le début, les clients nous ont proposé l'utilisation de JPC/SC un wrapper java utilisant JNI(Java Native Interface) pour renvoyer les appels de méthodes et les arguments Java vers la bibliothèque client fournie avec Microsoft PC/SC ou pcsc-lite(dans notre cas pcsc-lite).

D'autres solutions supportant le standard PC/SC sont disponibles en Java(comme OCFPCSC). Pour garantir l'indépendance de notre API par rapport à la couche communicant avec les lecteurs, une classe $CardGridUser$ faisant le rôle d'une couche intermédiaire a été fournie par les clients et adopté pour notre projet.
Cette classe offre deux fonctionalités principales :

- La connexion au lecteur.
- l'envoie d'un tableau d'octets à la carte et le retour de la réponse.

7.2 Connexion au lecteur

La connexion au lecteur s'effectue à l'instanciation de la classe CardGridUser avec le nom du lecteur.

```
public CardGridUser(String cardName){
  initContext() ;
  card = this.connect(cardName) ;
}
```

7.3 Envoie d'une commande à la carte

L'envoie se fait avec la méthode suivante :

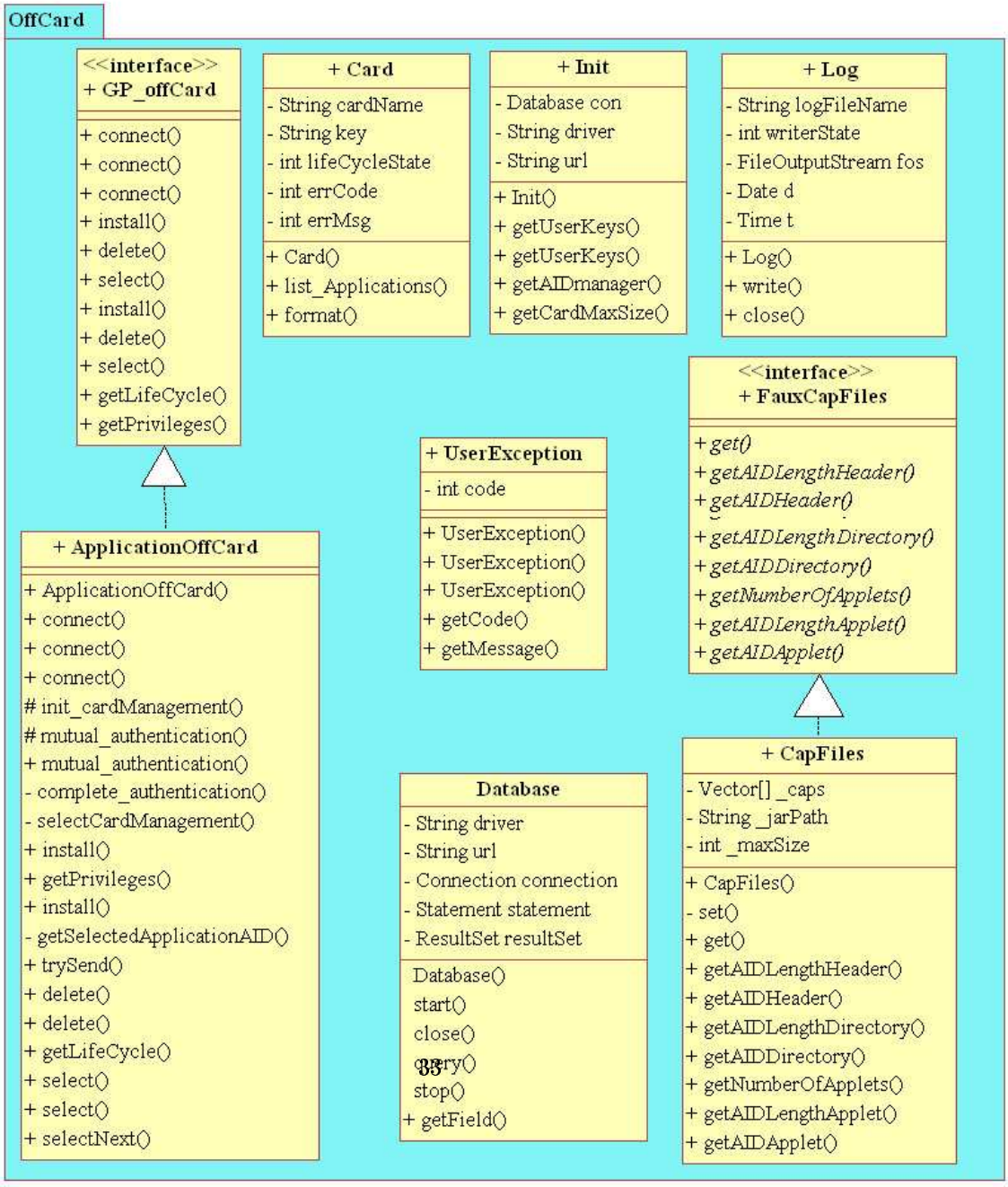
```
public byte[] sendAPDU(byte[] ba) throws Exception
```

Cette méthode envoie le tableau d'octets ba au lecteur et retourne le tableau d'octets reçu.

Chapitre 8

Package offCard

8.1 Diagramme des classes



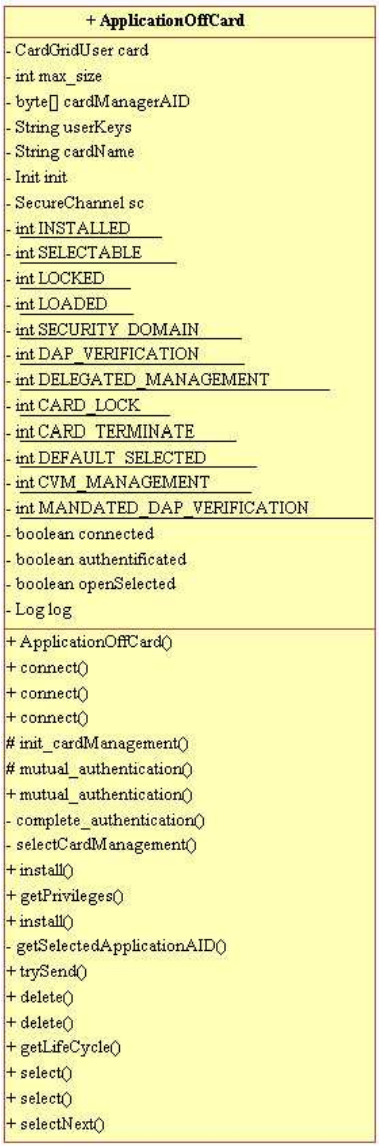


FIG. 8.2 – Classe ApplicationOnCard

8.2 Initialisation de la communication avec la carte

L'initialisation de la communication avec la carte se fait en trois étapes :

1. La connexion au lecteur.
2. La sélection du card manager.
3. L'authentification mutuelle.

Deux modes d'utilisation sont possibles :

– Ouverture d'une session juste après l'instanciation de la classe ApplicationOffCard. Ceci en utilisant la méthode *connect* fournie avec trois prototypes :

```
public void connect(String cardName) throws UserException ;
```

```
public void connect(String cardName, String userkeys) throws UserException ;

public void connect(String cardName, int keyVersion, int keyIndexNumber) throws UserExcepti
```

Selon le prototype utilisé, les paramètres non fournis sont lus directement de la base de données en utilisant l'identifiant de la carte.
Cette méthode effectue successivement les trois étapes de l'initialisation de la communication avec la carte.
L'avantage de cette méthode est que l'initialisation n'est faite qu'une seule fois pour la totalité d'une session tant qu'il s'agit de la même carte.

- Initialisation de la communication au début de chaque opération. Avec cette méthode, les appels à ces différentes opérations doivent fournir aussi les paramètres de l'authentification tel que $userKeys$.

8.2.1 La connexion au lecteur

La communication avec la carte est assurée par les fonctions du package $cardGridCom$.
On utilise deux fonctionnalités principales :

1. L'ouverture de la connexion. Ceci se fait en instanciant la classe $CardGridUser$ avec le nom du lecteur désiré.
2. L'envoie d'un tableau d'octets à la carte, et la réception d'une réponse.

Si la connexion réussit l'attribut *connected* est mis à *true*. Pour plus de détails voir le chapitre package $cardGridCom$ (7).

8.2.2 La sélection du card manager

Le card manager(OPEN) est le gestionnaire côté carte qui est responsable de gérer les commandes de gestion de contenu (Sélection, installation, suppression,...).
Il doit être sélectionné avant l'envoie de telles commandes.
La méthode implémentée a le prototype suivant :

```
private void selectCardManagement(String cardName) throws UserException
```

l'AID du *cardmanager* est récupérée à partir de la base de données en utilisant le nom de la carte. Ensuite, la méthode $GlobalPlatform\ selectApplication$ est appelée avec cet AID.
Si la sélection réussit l'attribut *openSelected* est mis à true.

8.2.3 L'authentification mutuelle :

L'authentification mutuelle est une phase durant laquelle une carte et une entité extérieure vérifient l'identité de leur correspondant.

Cette étape a pour objectif de vérifier que chacun est celui qu'il prétend être. L'authentification mutuelle est nécessaire avant l'envoie de toute commande de gestion de contenu.
La figure suivante montre les étapes d'une authentification mutuelle entre la carte et le PC.

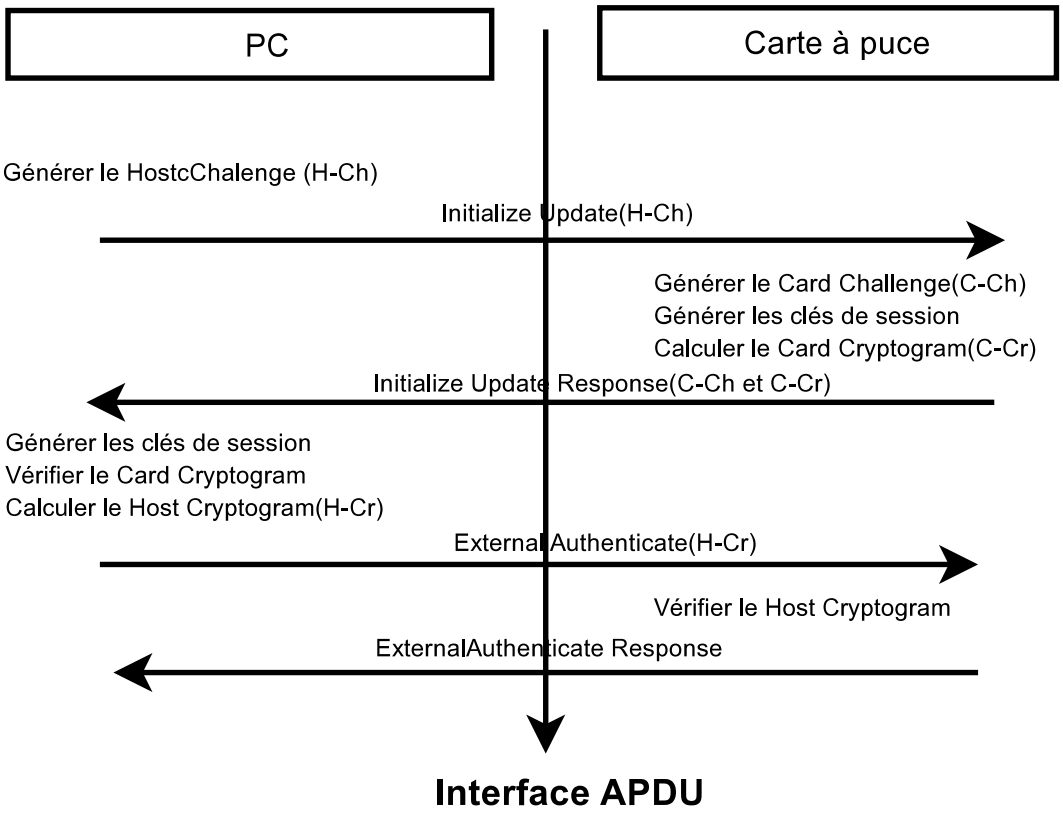


FIG. 8.3 – Authentification mutuelle

Comme le montre la figure ci-dessus, l'authentification se fait en deux étapes :

1. **Initialiser l'authentification :**
   Il s'agit d'envoyer la commande $initialize - update$ en appelant la méthode initializeUpdate du package $globalPlatform$(voir 6).
   Deux prototypes sont fournis :

```
public void mutual_authentication(String userKeys) throws UserException

protected void mutual_authentication(String userKeys, int keyVersion,
 int keyIndexNumber) throws UserException
```

   Le premier prototype prend simplement les clés statiques servant à l'authentification. Le deuxième prend en plus la version de la clé et son index. En effet, une carte peut contenir un ensemble de clés définies par des indexs. Par défaut les deux paramètres : keyVersion et keyIndexNumber sont égaux à 0.
   Si cette étape réussit on passe la réponse de la carte à l'étape suivante qui termine l'authentification.

2. **Compléter l'authentification :**
   La deuxième phase de l'authentification se fait avec la méthode $externalAuthenticate$ du package globalPlatform en utilisant la réponse de l'étape précédente.
   Si cette étape réussit, l'attribut *authenticated* est mis à *true* marquant que

la carte est maintenant prête pour reçevoir les commandes nécessitant une authentification.
Le prototype fourni est le suivant :

```
private void complete_authentication(byte[] initResp) throws UserException
```

8.2.4 Modification du niveau de sécurité

Par défaut, la valeur initiale du niveau de sécurité est 0. La méthode suivante permet de modifier ce niveau de sécurité :

```
public void set_securityLevel(int level) throws UserException
```

Dans le cas où une authentification précédente a été établie avec un niveau de sécurité différent, L'authentification est réinitialisée.

8.3 Chargement et installation d'un package

Il s'agit d'enregistrer sur la carte un package. Un package est constitué d'un ou plusieurs applets. Cette fonction nécessite que $authenticated = true$ et $openSelected = true$.
L'installation se fait en plusieurs étapes comme le montre la figure suivante :

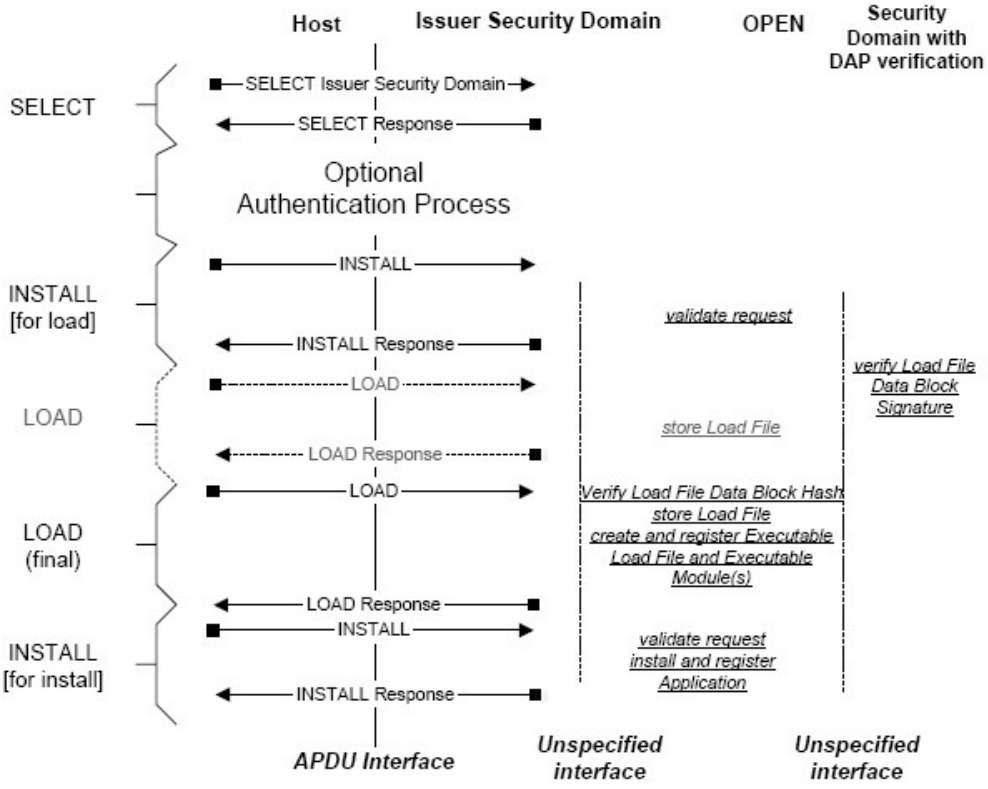


FIG. 8.4 – Chargement et installtion d'un package

1. **Préparer la carte à la réception des données :**
   La première commande global platform à envoyer à la carte est *InstallForLoad*.
   L'envoie de cette commande suppose que le *cardmanager* a été sélectionné et que l'authentification mutuelle a réussi. Dans le cas contraire, une exception 'Conditions d'utilisation non satisfaites' sera levée.
   Au cours de cette étape, on envoie l'AID du fichier à charger.
2. **Envoie des données :**
   Cette étape est réalisée avec la commande *LOAD*. Il s'agit d'envoyer les applets sous forme de blocs de données successifs. La taille maximale d'un blocs est spécifique à la carte et elle est lue à partir de la base de données.
   Le champ P1 de la commande permet de spécifier si la carte doit attendre un nouveau bloc ou il s'agit du dernier.

3. **Installation du package :**
   A cette étape, on utilise la commande *InstallForInstall*. Un package, peut contenir plusieurs applets. Il s'agit donc d'envoyer cette commande successivement avec les AIDs de ces différents applets.

Pour l'installation deux signatures sont disponibles :
- La première suppose qu'un appel à *connect* a été fait et donc une connexion avec la carte a été initialisée. Ceci est vérifiée en testant que les variables *connected*, *authenticated* et *openselected* sont toutes à *true*.

  ```
  public void install(String packagePath) throws UserException ;
  ```

- La deuxième prend tous les parmètres nécessaires pour initialiser une connexion avec la carte et installer le package.
  ```
  public void install(String card, String userKey, String packagePath)
    throws UserException;
  ```

8.4 Suppression d'un package

Il s'agit de supprimer un package se trouvant sur la carte. Cette fonction nécessite une authentification préalable et aussi la sélection du *cardmanager*($authenticated = true$ et $openSelected = true$). La figure suivante montre les échanges avec la carte pour supprimer une application :

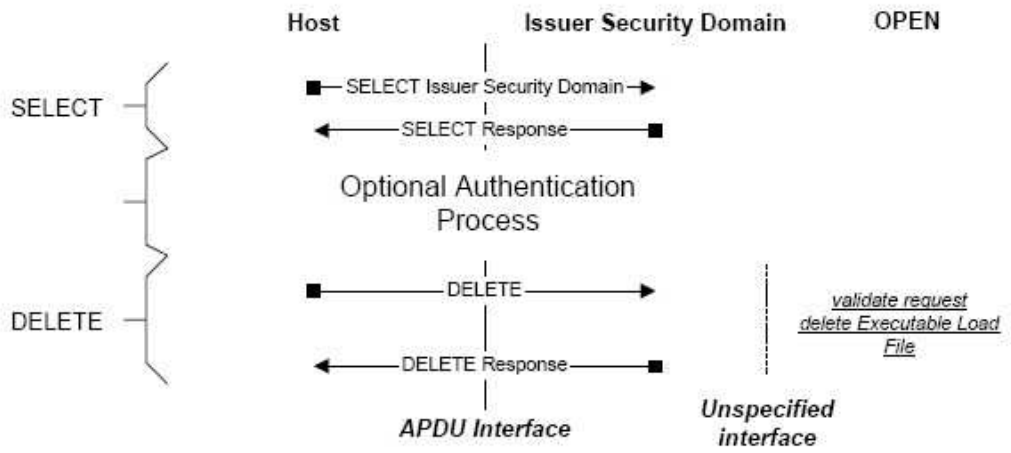


FIG. 8.5 – Suppression d'une application sur la carte

Deux signatures sont disponibles :

```
public void delete(String packagePath) throws UserException;

public void delete(String card, String userKey, String packagePath)
  throws UserException;
```

La première signature suppose qu'une connexion a été initialisé avec *connect*.

8.5 Sélection d'une application

Il s'agit de sélectionner une application disponible sur la carte dans l'état *Selectable*. La sélection d'une application sur la carte signifie que c'est cette application qui va reçevoir les commandes suivantes. Pour cette raison on met à la fin *openSelected* à $false$. Ceci permettra de refaire la sélection du *cardmanager* à l'appel suivant. Comme pour les méthodes précédentes, deux signatures sont également disponibles :

```
public void select(String packagePath) throws UserException;

public void select(String card, String userKey, String packagePath )
throws UserException;
```

La première signature suppose également qu'une connexion a été initialisée avec *connect*.

A ces méthodes s'ajoute une autre, permettant de sélectionner l'occurence suivante de l'application spécifié :

```
public void selectNext(String card, String userKey, String packagePath)
 throws UserException;
```

8.6 Privilèges et états d'une application sur la carte

Les privilèges associés à une applications sont codés sur 8 bits. Ils sont divers et variés et concernent la sécurité, le blocage d'une carte, la terminaison d'une carte, la vérification DAP, l'opération par défaut etc...

La méthode suivante retourne le privilège associé à une application :

```
public String getPrivileges(byte [] AID)
```

Tout comme la méthode *getLifeCycle*, cette méthode initialise une connexion, utilise le critère de recherche lc égal à 4F et l'envoie à l'APDU en utilisant la fonction *getStatus*.

La réponse retournée est analysée et une chaîne de caractère contenant le privilège associé est renvoyée.

8.7 Listage des applications de la carte

Il s'agit de demander à la carte la liste des applications qu'elle contient.
Comme toutes les commandes de gestion de contenu, cette fonction nécessite aussi l'authentification et la sélection du *cardmanager*.
La commande $GlobalPlatform$ utilisée est $GETSTATUS$. la commande APDU construite fournie à la carte contient un critère de recherche vide pour avoir les informations sur tout le contenu.

Le paramètre $P1$ définie le domaine de la recherche sur la carte. Deux commandes type de commandes $GETSTATUS$ sont envoyés :

- $P1 = 20$: Pour avoir les AID des modules chargés.
- $P1 = 40$: Pour avoir les AID des applications.

Le paramètre $P2$ code la structure du tableau de la réponse. Celui-ci contient la liste des AIDs des applications trouvées ainsi que d'autres informations, comme l'état de l'application et ses privilèges. La liste des AIDs est retournée sous forme d'une collection.
Le prototype est le suivant :

```
public Collection list_Applications ()throws UserException ;
```

8.8 Formatage d'une carte

Il s'agit de supprimer toutes les applications de la carte. Cette liste est récupérée en utilisant la fonctionnalité précédente. Une simple boucle appelle la fonction *deleteApplication* pour chaque AID.
Le prototype est le suivant :

```
public void format()throws UserException
```

Chapitre 9

Gestion des exceptions

La vérification des erreurs est une partie essentielle du développement des Applications. Dans notre projet, nous avons eu besoin de traiter différents types d'erreurs à travers la gestion des exceptions.

Dans cette partie seront expliqués les différents traitements d'exceptions qui ont été implémentés.

9.1 Le module UserException

La classe UserException qui hérite de la classe Exception, appartient au package offCard et est utilisée pour gérer et masquer tous les types d'exceptions qui peuvent être levées par l'API. Cette classe contient 3 différents constructeurs, un par défaut, un autre qui prend un message d'erreur et le dernier qui prend en plus un code d'erreur.

Avec l'utilisation de ce module, nous pouvons traiter les erreurs que peuvent arriver au moment d'initialiser une connexion avec une carte, de l'authentification mutuelle, de l'installation des applications, etc.

9.2 Les exceptions PC/SC

Lors de la connexion et de la communication avec la carte, plusieurs exceptions peuvent être levées. Exemples d'exceptions PCSCException :

`Connection Failure !SCardListReaders: 0x8010002e, PCSC failed with 0x8010002E`

Cette exception indique l'absence de lecteur connecté.

Tous les appels aux méthodes de la couche JPCSC sont encapsulés dans un bloc *try* et *catch*. Si une exception est levée elle est capturée et envoyée sous forme de UserException.

9.3 Les erreurs Global Platform

Les spécifications Global Platform définissent un certains nombre de codes d'erreurs que renvoie la carte comme réponse à une commande erronnée ou dont le contexte n'a pas été préparé. Ce code d'erreur est codé avec les champs $SW1$ et $SW2$ de la réponse(Voir 5.1.3). Le module $GP_responses$ du package Global Platform définit des constantes représentant les différents codes d'erreurs possibles.
Après chaque envoie à la carte, la réponse doit passer à une méthode qui analyse le contenu des champs $SW1$ et $SW2$. Si la commande a été exécuté avec succès la valeur de ces deux champs doiet être égale à $0x9000$ et donc la constante SUCCES est retournée. Sinon, la commande a échouée et dans ce cas, une exception UserException est levée avec le code de l'erreur et un message correspondant.

Exemple d'exception : Essayer d'installer un package sans authentifier.
La réponse de la carte : $0x6982$ (dans les champs $SW1$ et $SW2$).
Message correspondant : "Conditions of use not satisfied".

9.4 Problèmes rencontrés et solutions apportées

Nous avons déjà commenté l'importance de traiter les différents types d'erreurs mais dans le cas particulier du développement du présent projet, cette question prend une importance transcendantale. Une erreur non détectée peut causer que la carte soit bloquée ou que le résultat soit la perte de données stockées dans la carte. Par suite, pour éviter ce genre de problèmes, nous avons eu besoin de recourir à la gestion d'exceptions.

Certains types d'erreurs peuvent se présenter au moment de la communication entre la carte et l'application destinée au management de celle- ci. Pour détecter et traiter ces inconvénients, nous avons proposé l'utilisation des gestionnaires d'exceptions cités ci-dessus. Nous les avons utilisés pour indiquer exactement la partie du code où une exception a été levée, connaître l'origine de cette erreur et postérieurement faire le traitement approprié. Pour cette raison, nous avons crée les traitements d'exceptions par rapport aux besoins propres de notre projet.

Chapitre 10

Tests

Pour tester les différentes fonctionnalité de l'API nous avons eu recours à différents types de tests :

- Les tests unitaires qui consistent à vérifier le contenu des commandes APDU construits.
- Les tests avec JCOP,ce qui consiste à faire des tests sur des cartes ou avec le simulateur JCOP.
- Tests avec des vrais cartes et sans JCOP.
- Les tests de compatibilité et intégration avec l'interface développée en parallèle.

10.1 Tests unitaires

Ces tests sont effectué en utilisant l'interface Junit.
Les tests ont été fait pour la construction des APDU des commandes suivantes :

- *select*
- $installForLOad$
- $installForinstall$
- *delete*

10.2 Tests avec JCOP

Avant que les focntions d'authentifications de notre API soient terminées et testées, nous étions obligés d'utiliser le shell JCOP pour tester les différentes fonctions(install, delete,...). En effet, les commandes shell : $init - update$ et $external - authenticate$ permettent d'établir l'authentification mutuelle avec la carte.

Le principe des différents tests effectués avec JCOP est d'afficher l'APDU construit avec notre API sur la console puis la copier et la lancer avec la commande shell *send*. JCOP se charge d'afficher la réponse de la carte indiquant si la commande envoyée a réussit ou pas.

10.3 Tests avec des vrais cartes et sans JCOP

Une fois les fonctions d'authentification sont implémentées et testés, nous avons pu effectué des tests en communiquant directement avec un lecteur.

Un jeu de programmes principales a été crée utilisant l'ensemble des fonctions implémentées. Plusieurs scénarios de tests ont été implémentés permettant

de vérifier le bon fonctionnement des différentes méthodes ainsi que la gestion des exceptions.

Exemples de scénarios :
- Installation puis sélection puis suppression et enfin réinstallation d'un même package sur une même carte. La réussite de ce test montre le bon fonctionnement des différentes fonctionnalités utilisées
- Installation d'une application sur une liste de carte Ce test n'a pas pu être réalisé sur des vrais cartes, puisqu'on ne disposait que d'un seul lecteur.
- Charger deux fois successives le même package sur une même carte. Une exception indiquant l'existance d'un package avec le même AID est levée.
- Supprimer un package qui n'existe pas sur la carte. Une exception indiquant que le package est introuvable est levée.

10.4 Problèmes rencontrés et solutions apportées

Plusieurs problèmes ont été rencontrés lors des tests d'intégration et de compatibilité avec la partie interface développée en parallèle par un autre groupe, ces problèmes sont surtout dûs aux incompatibilités de prototypes de méthodes que nous développons et des appels qu'ils font ou l'inverse.
Pour y remédier, nous avons fournie des interfaces pour les classes $ApplicationOffCard$ et $Card$

Chapitre 11

Conclusion

D'un point de vue technique, notre projet de fin d'année à été mené à terme dans les temps, et la pérennité de l'API pour applications sur cartes à puces a été testée avec succès, ce qui est satisfaisant.

Après avoir mis en oeuvre des méthodes de travail, nous avons pris part à une véritable étude de projet, suivie par la communication avec notre responsable pédagogique, et les clients. Nous avons fait en sorte que notre API respecte de plus près les besoins évoqués par les clients et avons travaillé paralèllement à l'autre équipe se chargeant de l'interface graphique, et nous sommes rencontrés plusieurs fois afin de rendre nos projets concordants.

Nous avons dû faire face à de nombreuses difficultés tant pour l'implémentation des méthodes qui demandait une lecture et un apprentissage pour la conformité au standard GlobalPlatform, que pour la communication des réels besoins clients. Ainsi notre travail s'est déroulé dans de bonnes conditions, et bien que l'atmosphère n'a pas été toujours comme nous le souhaitions, nous avons su offrir un travail de groupe tant que possible. Par la même occasion, cela nous a permis de partager nos connaissances vues en cours et de nous entraider dans les moments difficiles.

Ce projet aura été très enrichissant, car il nous a permi d'être insérés dans un projet industriel avec toutes ses composantes : spécifications, référentiel qualité, développement, et déploiement. Il s'inscrit donc dans la complétion de notre formation d'ingénieur ENSEIRB, spécialisé en Informatique.

Annexe A

Lexique

TAB. A.1: Terminologie et définitions

Terme	**Définition**
Application	L'exemple d'un module exécutable après lui a été installé et a rendu sélectionnable.
Application Protocol Data Unit (APDU)	Protocole standard de transmission de messages entre une carte qui accepte le dispositif et une carte intelligente.
Application Provider	Entité qui possède une application et est responsable du comportement de l'application.
Application Session	Le lien entre l'Application et le monde externe sur un canal logique commençant par le choix de l'application et la fin quand un autre choix d'Application se produit sur le canal logique, le canal logique est fermé ou la session de carte se termine.
Asymetric Cryptography	Une technique cryptographique qui utilise deux transformations connexes, une transformation publique (définie par la composante clé publique) et une transformation privée (définie par la composante clé privée), ces deux composantes clés ont une propriété de sorte qu'elle soit informatiquement infaisable pour découvrir la clé privée, même si la clé publique est donnée.
Basic Logical Channel	L'interface disponible de manière permanente entre la carte et une entité externe. L'élémentaire canal logique est numéroté zéro.
Card Content	Information de code et application (mais pas les données d'application) contenues dans la carte qui est sous la responsabilité de OPEN e. g. Executable Load Files, instances d'application, etc.
Card Image Number (CIN)	Un identificateur pour une carte espécifique GlobalPlatform.
Card Issuer	Entité qui possède la carte et est finalement responsable du comportement de la carte.
Card Manager	Terme générique pour les 3 entités de la carte GlobalPlatform :OPEN, Issuer Security Domain et le fournisseur Cardholder Verification Method Services.
Card Recognition Data	Information qui indique un système externe, en particulier "Card and Application Management System (CAMS)", comment travailler avec la carte (indiquant que c'est une carte de GlobalPlatform).
Card Session	Le lien entre la carte et le monde externe commençant par l'ATR et la fin avec une remise suivante ou une désactivation de la carte.

TAB. A.1: Terminologie et définitions (suite)

Terme	**Définition**
Card Unique Data	Donnée qui uniquement identifie une carte étant la concaténation entre le nombre d'identification de l' emetteur (Issuer Identifcation Number) et "Card Image Number".
Cardholder	C'est l'utilisateur final de la carte.
Cardholder Verification Method (CVM)	C'est une méthode pour assurer que la personne qui présente la carte est la personne à qui la carte a été émise.
Controlling Authority	Le "Controlling Authority" a le privilège de garder le contrôle du contenu de la carte par l'exigence de "DAP Verification".
DAP Block	Partie du "Load File" utilisée pour assurer la vérification de "Load File Data Block".
DAP Verification	Un mécanisme utilisé par un domaine de securité (Security Domain) pour vérifier que un "Load File Data Block" est authentique.
Delegated Management	Les changements pré autorisés de "Card Content" sont exécutés par un fournisseur d'applications (Application Provider) approuvé.
Digital Signature	Une transformation cryptographique asymétrique de données qui permet à le destinataire de données de prouver l'origine et l'integrité des données.
Executable Load File	Actuel sur carte qui contient le code executable d'une ou plusieurs applications.
GlobalPlatfomr Registry	Un récipient d'information s'est relié à la gestion de contenu de carte.
Host	Un terme logique utilisé pour representer les systèmes arriéres d'en qui soutiennet le systè de "GlobalPlatform"; les "hosts" exécutent des fonctions telles comme l'autorisation et authentification, l'administration, code de l'application"Post-Issuance" et données téléchargeant, et traitement transactionnel.
Immutable Persistent Memory	Mémoire qui seulement peut être lue.
Issuer Security Domain	Entité de sur carte fournissant l'appui pour les requeriments de contrôle, de securité et de communication de l'emetteur de la carte.
Life Cycle	L'existence du contenu de la carte sur une carte de GlobalPlatform et le diverses etapes de cette existence où est applicable.
Life Cycle State	Un état spécifique dans le cycle de vie de la carte ou du contenu de la carte.
Load File	Un fichier transféré à une carte de GlobalPlatform qui contient un "Load File Data Block" ou probablement un ou plusieurs blocs DAP.
Load File Data Block	Partie de "Load File" qui contient une ou plusieurs applications an bibliothèques ou soutenez l'information pour l'application selon les exigences d'une plateforme spécifique.
Load File Data Block Hash	Une valeur fournissant l'integrité pour le "Load File Data Block".
Load file Data Block Signature	Une valeur entourant le "Load File Data Block Hash" et fournissant les deux l'integrité et l'authenticité du "Load File Data Block".
Message Authentication Code (MAC)	Une transformation cryptographique symétrique de données qui fournit l'authentification d'origen des données et l'integrité des données.
Mutable Persistent Memory	Mémoire qui peut être modifié.
OPEN	L'administrateur de sur-carte central qui possède le "GlobalPlatform Registry".

TAB. A.1: Terminologie et définitions (suite)

Terme	**Définition**
Post-Issuance	Phase suivant à l'emission de la carte au "Cardholder".
Pre-Issuance	Phase antërieure à l'emission de la carte au "Cardholder".
Private Key	Le composant privé de la clé paire asymétrique.
Public Key	Le composant publique de la clé paire asymétrique.
Receipt	Une valeur cryptographique fourni par la carte (si ainsi requis par le "Card Issuer") comme preuve qu'une operation "Delegated Manegement" s'est produit.
Retry Counter	Un compteur, employé en même temps que le "Retry Limit", pour determiner tentativement quand une valeur CVM sera interdit.
Retry Limit	Le nombre de maximum de fois où une valeur CVM inadmissible peut être présenté avant le traiteur CVM interdissant plus loin pour présenter une valeur CVM.
Secure Channel	Une mécanisme de communication entre une entité de "off-card" et une carte qui fournit un niveau d'assurance à une ou les deux entités.
Secure Channel Protocol	Un protocole de communication secure et un ensemble de services de securité.
Secure Channel Session	Une session, pendant une "Application Session", commençant avec le déclemchement du "Secure Channel" et la fin avec un arrêt de "Secure Channel" ou l'arrêt soit d'une "Application Session" ou d'un "Card Session".
Security Domain	Entité "On-card" fournissant l'appui pour le contrôle , la securité et les requeriments de communication du "Application Provider".
Supplementary Logical Channel	Jusqu'á 3 interfaces additionnelles (autre que le "Basic Logical Channel" disponible de manière permanente) entre la carte et une entité externe. Chaque "Supplementary Logical Channel" est numeroté par 1, 2 ou 3.
Symmetric Cryptography	Une technique cryptographique qui emploie la même clé secrète pour le créateur et la transfomation du destinataire.
Token	Une valeur cryptographique fourni par un "Card Issuer" comme preuve qu'une operation "Delegated Management" a été autorisé.

TAB. A.2: Abréviation et Notations

Abréviation	**Signification**
AID	Application Identifier.
APDU	Application Protocol Data Unit.
API	Application Programming Interface.
ASCII	American Standard Code for Information Interchange.
ATR	Answer-to-Reset.
BCD	Binary Coded Decimal.
BER	Basic Encoding Rules.
CBC	Cipher Block Chaining.
CIN	Card Image Number / Card Identification Number.
CLA	Class byte of the command message.
CVM	Cardholder Verification Method.
DAP	Data Authentication Pattern.
DEK	Data Encryption Key.

TAB. A.2: Abréviation et Notations (suite)

Abréviation	**Signification**
DES	Data Encryption Standard.
ECB	Electronic Code Book.
EMV	Europay, MasterCard, et Visa ; utilisé par rapport à les specifications d'ICC pour des systèmes de paiement.
ENC	Encryption.
FCI	File Control Information.
HEX	Hexadecimal.
ICC	Integrated Circuit Card.
ICV	Initial Chaining Vector.
IIN	Issuer Identification Number.
INS	Byte d'instruction pour le message de commande.
ISO	International Organization for Standardization.
Lc	Longeur exacte de données dans la commande du cas 3 ou du cas 4.
Le	Longeur maximum de données prévues dans la réponse à une commande du cas 2 ou du cas 4.
LV	Length Value.
MAC	Message Authentication Code.
OID	Object Identifier.
OPEN	GlobalPlatform environment.
P1	Reference au paramètre de contrôle 1.
P2	Reference au paramètre de contrôle 2.
PIN	Personal Identification Number.
RAM	Random Access Memory.
RFU	Reserved for Future Use.
RID	Registered Application Provider Identifier.
ROM	Read-only Memory.
RSA	Algorithme asymétrique Rivest / Shamir / Adleman.
SCP	Secure Channel Protocol.
SW	Status Word.
SW1	Status Word One.
SW2	Status Word Two.
TLV	Tag Length Value.

Annexe B

Comptes rendus

Tous les comptes rendus ainsi que plusieurs d'autres informations concernant le déroulement du projet(Répartition, avancement,...) sont disponibles sur nos pages wiki à l'adresse : http ://pfa.fisoft1.com

B.1 Comptes rendus hebdomadaires :

La semaine du 23/01 au 29/01

Travail accompli

Au cours de cette semaine, on a essayé d'approfondir notre compréhension du sujet en lisant la spécification de GlobalPlatform, la documentation fournie par Java de l'API Java Card ainsi que le code fourni sur SourceForge (code). Cette documentaton étant fort abondante, une répartion en groupe s'est faite.

On a également mis en place une méthodologie de travail en groupe.

Planning

Des questions restent en suspens après la lecture de la doc, ces questions seront abordées avec les responsables scientifiques au cours de la prochaine réunion.

La semaine du 30/01 au 04/02

Travail accompli

Avancement du cahier des charges : Cette semaine a ete consacre pour la redaction du cahier des charges. Un plan a ete mise en place et presente au responsable pedagogique pendant la reunion. Une fois le plan valide, la redaction a commence. Environ 75redige avant la fin de la semaine.

Planning

La finalisation du cahier des charges pour la validation de celui-ci par le client et par le responsable pedagogique.

Semaine du 19/03 au 25/03

L'état de l'avancement du projet

– Package globalPlatform :

- classe ApplicationOnCard : toute les fonctions(demandées) GP sont faites avec les tests unitaires. Les commandes DELETE et SELECT ont été aussi testé avec JCOP et elles fonctionnent.
- classe SecureChannel : Les fonctions init_update et external_authenticate ont été implémenté. On a eu des problèmes lors du test avec JCOP. Les deux cartes sont apperemment bloqués après un certains nombre de tests, même JCOP n'arrive plus à s'y connecter.

- Package offCard :
  - La répartition pour le développement de ce package est sur notre wiki(http ://pfa.fisoft1.com).

  - Les fonctions terminés :
    - install
    - init_cardManager : se connecte à la carte et selectionne le card manager (essentiel pour pouvoir lancer les autres fonctions)
    - mutual_authenticate : etablit l'authentification mutuelle.

    On a prévu de terminer le reste des fonctions de ce package avant le prochain Jeudi.

Problèmes

- Les deux cartes sur lesquels on a fait les tests sont 'bloqués'. Nous souhaitons savoir s'il ya une manipulation à faire pour les rendre de nouveau 'utilisables' avec JCOP.
- On souhaite obtenir un rendez-vous au début de la semaine prochaine(de préférence Lundi après midi après 16h40) pour effectuer les tests, surtout pour la fonction external_authenticate qui utilise des fonctions de cryptage compliqués, et pour laquelle on n'a aucun moyen de vérifier sa validité sans une carte valide.

- pour le package comm : on a pris la liberté de changer la signature des methodes de la classe CardGridUser ? de façon à pouvoir se connecter à une carte donnée avec le nom du lecteur. Mais on a aucune maîtrise sur les exceptions qui peuvent y être levés. les appels aux méthodes de cette classe sont faites dans un bloc try de facon a capturer les exception et lever notre exception UserException. Les interfaces pour le package offCard : GP_card et GP_offCard ainsi que UserException(gestion des exception) et GP_responses(Gestion des codes d'erreurs GP) ont été passé au groupe interface.

- étant donnée qu'on n'a pas un parseur de .cap, on utilise pour le moment une classe faussaire FauxCapParser pour simuler l'accès aux données du fichier .cap

Semaine du 26/01 au 02/04

Toutes les fonctions prévues ont été implémentés. Les fonctions testées : select, delete, init-update et external-authenticate Fonctions non testées : Install, listage des applets de la carte et formatage d'une carte.

L'état de l'avancement du projet

- L'accès à la base de données :

A priori, nous avons besoin des informations suivantes :

- Les clés statiques(ENC et MAC) à partir de l'index de la clé et l'identifiant de la carte.
- L'AID du cardManager à partir de l'identifiant de la carte.
- max_size(La taille maximale d'une commande APDU à envoyer à une carte) à partir de l'identifiant de la carte.

- Les prototypes retenus : (pour install comme exemple)
  - install(String cardId, String userKeys, String path) : on s'authentifie directement avec les valeurs des clés données

  - install(String cardId, int keyVersion, int keyIndexNumber, String path) :On récupère les clés de la base des données avant l'authentification

  Dans les deux cas, on récupère l'AID du card manager de la base de données pour le sélectionner avant l'envoie de toute commande.

- Parsing des fichiers .cap : on utilise temporairement les source d'un ancien projet étudiant.

B.2 Comptes rendus des réunions avec les clients

La réunion du 19/01/2006

Cette réunion, étant la première avec les responsables scientifiques, était essentiellement consacrée pour se mettre dans le contexte général du projet et pour discuter de quelques détails organisationnels. Nous avons aussi obtenu de la documentation générale sur le projet ainsi que sur les normes GlobalPlatform à suivre pour le développement de l'API.

Un deuxième rendez-vous a été fixé : le Mardi 24/01 à 16h30.

la réunion du 24/01/2006

Cette réunion est la deuxième aprés une présentation générale du projet. Elle était consacré à la réponse aux questions soulevées par la lecture de la documentation.

Beaucoup de questions ont été soulevé concernant l'identification et le repérage des cartes dans la grille. Le problème est transparent pour notre équipe car notre visibilité est restreinte à une seule carte.

Les applications à charger/décharger sur une carte sont choisit par l'utlisateur à travers l'interface. Cette dernière transmet alors à un module intérmédiaire la liste des applications à installer, qui nous les transmet alors une à une, en respectant l'ordre dans lequel elles devront être installées.

Nous avons également obtenu de la documentation conceranant les spécifications GlobalPlatform.

La réunion du 30/01/2006 :

Objectif généraux :

* Eclaircir certains points de la documentation fournie la semaine dernière. Préciser les fonctionnalitées à implémenter.

Résumé :

Cette réunion nous a permis d'avoir une idée plus claire sur le travail demandé :

- L'implémentation en Java de toutes les fonctionnalitées offertes par GlobalPlatform sauf celles qui concernent la sécurité, à savoir :
  . Initialiser un canal de communication avec la carte (en s'authentifiant)
  . Installer une application
  . Supprimer une application
  . Selectionner une application
  . Lister les applications contenues dans la carte

- Il est également demandé de fournir un interpréteur de commande pour lancer les différents fonctions ainsi qu'un jeu de tests.
- Plusieurs d'autres fonctionnalitées peuvent être demandées selon l'avancement du projet tels que l'implémentation des routines de gestion de sécurité.

L'idée de réutiliser le travail effectué sur le projet Open Source a été , à priori, abandonné, vue que la totalité de l'implémentation a été faite en C.

Commentaires

- Un rendez vous inter-équipe est prévu pour le Mardi 31/01/2006 à 16h pour commencer concrêtement l'écriture du cahier des charges.
- Selon l'avancement de celui-ci, un autre rendez-vous avec les resp scientifiques peut être prévu au cour de cette semaine.

Réunion du 01/03/2006

Objectif généraux :

- Discuter des derniers détails du cahier des charges.
- Discuter des outils de développement à utiliser.

Résumé :

La dernière version du cahier des charge a été validé par les clients. Cependant quelques détails ont été discuté :

- La méthode init_channel_01 de la classe ApplicationOnCard ? devra être déplacé dans nouvelle classe 'security'.

- Tous les méthodes du package globalPlatform devront être indépendant de l'environnement(Cartes, lecteurs...), elle ne doivent pas prendre en paramètre l'identifiant de la carte ou du lecteur.

Commentaires

- Les clients nous ont fournit un lecteur et deux cartes pour pouvoir effectuer des tests en utilisant le code source du projet developpe dans sourceforge.net

Réunion du 16/03/2006 :

Objectif généraux :

- Discuter de l'avancement du développement.
- Poser plusieurs questions techniques concernants des problèmes rencontrés pendant la semaine.

Décisions importantes prises :

- La gestion du lancement parallèl des installation sera développé par le groupe interface.Les prototypes définis au cours de la semaine seront maintenus définitivement.
- La vérification de la validité des paramètres pour les fonctions GP ne sera pas faite.L'invalidité sera levée éventuellement par la carte.

Réunion du 29/03

Toutes les fonctions prévues ont été implémentés.

-Les fonctions testées : select, delete, init-update et external-authenticate.

-Fonctions non testées : Install, listage des applets de la carte et formatage d'une carte.

B.3 Compte rendus des réunions avec le responsable pédagogique

Réunion du 13/01/2006

Objectifs généraux :

- Présentation générale du déroulement du pfa.
- Conseils sur la méthodologie et l'organisation du travail.
- Présentation des outils mis en place.

Résumé :

Un wiki a été mis en place

Un alias qui regroupe les membres de l'équipe(pfa@fisoft1.com) a été créé.

Réunion du 02/02/2006

Objectif généraux :

- Discuter et valider le plan du cahier des charges.
- Discuter de l'avancement de la redaction du cahier des charges.

Résumé :

La reunion etait consacree pour la discution du plan du cahier des charges.
Plusieurs commentaires ont ete souleves :

- Les diagrammes des cas d'utilisation doivent etre accompagnes par une documentation exhaustive.

- Une partie "Contexte du projet" doit etre ajoutee au cahier des charges pour developper le contexte general du projet.
- Une partie " Extensions possibles" peut etre ajoute pour exposer des fonctionnalites supplementaires eventuelles
- Il faut revoir l'interet et le contenu eventuel de la partie "Gestion des risques et incertitudes" .

Réunion du 20/03

Objectif généraux :

Suivi de l'avancement du projet.

Résumé

Nous avons exposé au responsable pédagogique, l'ensemble des problèmes qui restent à résoudre :

- L'instabilite de ce que veut exactement le client(les prototypes et limites avec le groupe interface)

- Le module pour parser les fichier .cap nécessaire pour le bon fonctionnement des fonctions du package offCard

Une date butoir : le Jeudi 30/03/2006 pour terminer tout le développement.