# Extension des capacités de traitement des cartes à puce bases de données

Sébastien JEAN+, Didier DONSEZ*

Email : jean@lifl.fr, donsez@univ-valenciennes.fr

+LIFL/RD2P, Université de Lille, Bâtiment M3, Cité Scientifique, 59625 Villeneuve d'Ascq Cedex, France

*LAMIH/ROI, Université de Valenciennes - Le Mont Houy BP 311, 59304 Valenciennes Cedex, France

## Résumé

Les cartes à puce bases de données offrent une manière élégante d'implanter des dossiers portables sécurisés comme le dossier médical de la carte patient. Ces dossiers portables se présentent sous la forme d'un ensemble de tables relationnelles manipulées à l'aide d'un langage de requêtes qui est une version adaptée carte de SQL. D'un autre côté, les cartes multi-services hébergent plusieurs services qui coexistent. Le principal inconvénient de ces cartes est la dépendance des données stockées vis à vis des traitements. Cet article s'intéresse au rapprochement de ces deux modèles indépendants de cartes pour conduire à la définition de la carte hybride. La carte hybride offre l'indépendance entre les données , les traitements et leur contrôle d'accès. Cette étude débouche sur des propositions de mise en œuvre de carte hybride à partir des produits disponibles chez les fabricants de cartes.

## Mots-Clés

Cartes à microprocesseur, Bases de Données Portables, Sécurité.

# 1 Introduction

Depuis son apparition, il y a une vingtaine d' années, la carte à microprocesseur s' est déclinée en plusieurs types fonctionnels divers.

La carte d' accès a pour vocation de sécuriser l' accès soit à des lieux, soit à des systèmes : on parle alors respectivement de sécurité physique ou logique. Dans la plupart des cas, la sécurisation d' accès repose sur l' identification/authentification du porteur. Cette identification peut être simplement la vérification d' un PIN code (Personal Identification Number) ou la mise en œuvre de mécanismes plus complexes tels que la reconnaissance biométrique **[A95]**. Le paiement est également une des applications types, avec des cartes telles que les cartes de prépaiement (comme le Porte-Monnaie Electronique) ou encore les cartes de crédit ou de débit.

Les deux derniers principaux types de cartes à microprocesseur, que nous détaillerons, sont les dossiers portables et les cartes multi-services. Les dossiers portables, orientés données, ont pour vocation de fournir un support sécurisé et portable de données personnelles. On dénombre plusieurs types de dossiers portables, allant de la carte "fichiers" à la carte base de données. Les cartes multi-services, orientées traitement, sont quant à elles plutôt destinées à offrir un support sécurisé de services et proposent de nouveaux concepts : polyvalence et évolutivité.

L' objectif de cet article est d' étudier la convergence entre l' orientation données et l' orientation traitement au sein de la carte à microprocesseur, dans le cadre de l' extension de la capacité de traitement des cartes bases de données. La section suivante présente les mécanismes des cartes à microprocesseur multi-services et des cartes bases de données. La troisième section énonce ensuite les concepts de la carte hybride qui réunit ces deux modèles. Enfin, en nous basant sur des cartes disponibles chez les fabricants, nous proposons une mise en œuvre des concepts de la carte hybride. Enfin, la section 5 conclut et présente quelques perspectives de recherche.

# 2 Etat de l'art carte

La carte à microprocesseur est un objet extrêmement normalisé, des caractéristiques physiques à la gestion de fichiers **[ISO87]**. Le protocole de communication entre la carte et le terminal est entrant, la carte fonctionne uniquement en mode serveur. Une commande normalisée nommée APDU (Application Protocol Data Unit) est envoyée, via le terminal, à la carte qui la traite et qui renvoie éventuellement des résultats. Une application encartée n' émet pas de commandes vers l' extérieur, mais peut simuler l' envoi d' APDU à la carte. L' accès à la carte (données et application) se fait via le terminal. Ce terminal joue le rôle de proxy tantôt pour le porteur qui a inséré la carte dans le terminal et tantôt pour un système distant qui envoie ses commandes via un réseau. Dans ce dernier cas, la commande peut être une commande d'installation d' une nouvelle application par un prestataire de service ou bien de mise à jour (cas de la Java Card / SIM du GSM). Pendant la connexion de la carte au terminal-proxy, les commandes successives du porteur et des systèmes distants peuvent s' entrelacer, le terminal-proxy gère alors un canal différent pour chaque utilisateur local (porteur) ou distant.

Parmi les cartes "évoluées", c' est à dire les cartes qui ne se limitent pas à la sécurisation d' accès, il existe 2 familles indépendantes que nous allons présenter dans les sous-sections suivantes : Les dossiers portables, en particulier les cartes bases de données, et les cartes multi-services.

## *2.1 Cartes multi-services*

### 2.1.1 Concepts et modèle logique des cartes multi-services

Les cartes multi-services sont articulées autour de trois concepts fondamentaux que nous expliciterons dans la suite: la coexistence de plusieurs services, coopérants ou non **[V97]**, sur la même carte, l' évolution dynamique du contenu applicatif de la carte, et enfin, la coopération entre les services.

*a) Coexistence de services et partage d'information*

Le "business model" des cartes à microprocesseur est en plein changement. Alors qu' auparavant le fabricant développait un "monolithe" système/application embarqué sur la carte, l' apparition des cartes multi-services et des cartes "blanches" **[V95]** remet en question ce modèle dans la mesure où l' applicatif est dissocié du système. Avec l' arrivée de cartes telles que la JavaCard **[JCard]**, la conception d' application carte est à la portée de développeurs non spécialistes du domaine. Désormais, les fabricants se concentrent sur la production de cartes physiques et sur la réalisation de systèmes d' exploitation modulaires et sécurisés. Puisque plusieurs applications développées par des organisations différentes peuvent être installées sur la même carte, ceci oblige les concepteurs des systèmes d' exploitation carte à mettre en œuvre des mécanismes de cloisonnement fort entre les applications afin de les isoler les unes des autres. En d' autres termes, la carte multi-services doit être vue comme un ensemble de cartes mono-service.

Le partage d'information entre les applications installées dans la carte est réalisé soit par partage de données soit par partage de code manipulant ces données. Cependant, le partage d'information soulève de nombreux problèmes de sécurité. Un exemple de mise en œuvre du partage est la carte Windows SC, récemment annoncée par MicroSoft, qui propose un partage des données via un partage de fichiers ISO et un partage par le code en simulant de l' intérieur l' envoi de commande APDU de l'application qui possède l'information dans son espace privé. Dans l'exemple de la Figure 1a, l'application A2 accède aux données privées de A3 via une APDU A3::C1.

*b) Evolution dynamique du contenu applicatif*

L' intérêt supplémentaire de ce type de cartes est que leur contenu applicatif peut évoluer au cours de la phase d' utilisation. L' éventail des services offerts aux utilisateurs est de plus en plus large. Cependant, les contraintes de stockage[1] de la carte à microprocesseur ne permettent d' embarquer qu' un nombre réduit de ces services. Cette évolutivité de service confère une souplesse au porteur qui peut décider de retirer ou d' ajouter, au travers de mécanismes offerts par le système carte, un nouveau service en fonction de ses besoins et de ses moyens. Par exemple, un client peut demander à sa banque de charger dans sa carte un service spécifique à un pays étranger pour la durée de son séjour. Cependant, il faut mettre en place une gestion très sécurisée d' importation/retrait de services surtout si ces opérations se font en ligne, par Internet ou par GSM. Ainsi, dans le système MULTOS, des mécanismes de certification sont mis en œuvre pour sécuriser l' importation **[Multos]**.

*c) Contrôle d' accès sur les applications*

Au niveau système, il n' existe pas dans la plupart des cas de contrôle d' accès fin sur les applications. Bien que les cartes multi-services se basant sur la norme ISO 7816-4, ont à leur disposition un contrôle d' accès à 7 niveaux de droits (qui permet un contrôle par classe d' utilisateurs), la gestion du contrôle d' accès est souvent reléguée au niveau applicatif (donc, à la charge du développeur).

[1] 512 octets à 1Ko de RAM et 16Ko à 32Ko de mémoire non volatile (FlashRAM, EEPROM, …).

### 2.1.2 Exemple de cartes multi-services : Java Card

Dans les spécifications Java Card, plusieurs "Card Applets" Java coexistent. L' exécution de ces applets est assurée par le couple API/machine virtuelle (voir Figure 1a) **[Sun99b]**. En exploitant les possibilités du langage Java en matière de sécurité **[FM96]**, la Java Card cloisonne les applications à l'aide de "sandboxes" **[Sun99]**. L' installation/désinstallation soit par l'émetteur d' application soit par des prestataires auxquels ce dernier délègue l'opération. Cet entité étant "de confiance", il n' est pas nécessaire d' effectuer des contrôles stricts tels que ceux évoqués précédemment.

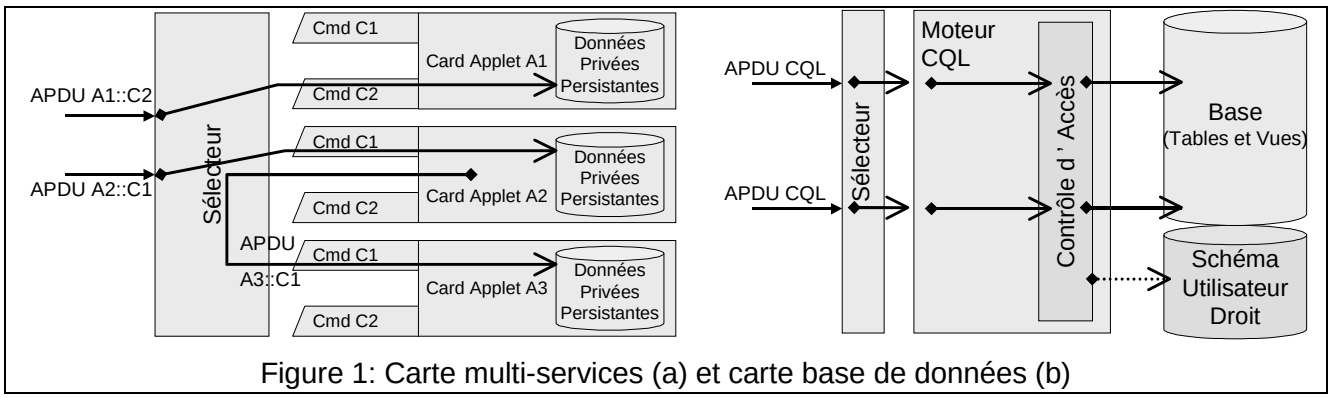


Figure 1: Carte multi-services (a) et carte base de données (b)

## *2.2* Cartes bases de données

Le principal intérêt des dossiers portables est le regroupement d' informations liées à leur porteur sur un support portable. L' avantage est que l' obtention de l' information, embarquée dans la carte, ne nécessite pas de communication externe du terminal avec un réseau de service (mode Off-line).

Le dossier portable se caractérise par un ensemble de fonctionnalités : une structuration globale des données indépendante des applications actuelles et futures, un support pour leur manipulation, un mécanisme de sécurisation afin de permettre le partage des données ou d' en limiter l' accès. Les cartes bases de données, de part les mécanismes qu'elles proposent, réalisent de manière efficace ces objectifs. Une carte base de données peut être vue à la fois comme une base de données portable sécurisée, et même comme un des sites d' une base de données répartie **[GG92]**.

### 2.2.1 Manipulation, sécurité et partage de données structurées

Les cartes bases de données offrent une structuration forte et globale, les données étant organisées en tables. De plus leur système d'exploitation offre un langage de requêtes comme outil puissant de manipulation de données. Les cartes bases de données offrent également des mécanismes de sécurisation et de partage des données dont l' objectif est que les différents intervenants puissent coopérer à travers les informations embarquées dans la carte. La sécurité des données est assurée par un système de contrôle d' accès géré par le système d' exploitation. Le contrôle d' accès est de granularité fine (utilisateur, table et vue), ce qui est un avantage par rapport aux cartes « fichiers ». Chaque utilisateur se voit accorder des privilèges de consultation (SELECT) et/ou de modification (INSERT, DELETE, UPDATE) sur tout ou partie de l' ensemble des données encartées. Ces privilèges peuvent dans certains cas être transmis à d' autres utilisateurs (GRANTEABLE).

### 2.2.2 Modèle de fonctionnement

La carte base de données est, à l' heure actuelle, utilisée uniquement comme serveur de données. Les utilisateurs envoient des ordres au système d' exploitation carte pour agir sur les données. Ces ordres sont traités par un moteur de manipulation qui charge un gestionnaire de droits de vérifier l' habilitation de

l' intervenant. Si l' ordre est conforme, le moteur de manipulation effectue l' accès aux données structurées. Dans cette configuration, il est clair que l' applicatif se situe sur le système hôte de l' utilisateur et enchaîne les ordres de manipulation puisque la carte ne fait pas de traitement pour cet intervenant.

### 2.2.3 Exemple de carte base de données : CQL

*a) Concepts généraux de la carte CQL*

La carte CQL **[G92]**, industrialisée par Gemplus **[Gem93]**, est issue de recherches menées au laboratoire RD2P. Le système d' exploitation de cette carte implante un certain nombre des concepts des SGBD relationnels. Le langage de requête CQL est un sous ensemble de SQL **[ISO92]**. Les ordres CQL envoyés sous forme de commandes APDU CQL, permettent de manipuler les tables, de gérer des droits et des curseurs sur les tables, mais n'assume pas la notion de jointure. La carte CQL a notamment été utilisée comme support pour l' intégration de la carte à microprocesseur dans les environnements distribués **[MVD96]**.

*b) Mécanismes de sécurité*

Le système d' exploitation CQL définit trois classes d' utilisateurs (émetteur d'application, gestionnaire d'application et utilisateur d'application) qui possèdent chacun des privilèges différents . De plus, outre les privilèges sur les données, il existe deux niveaux de contrôle d' accès : un mécanisme classique de vérification de mot de passe, et une double authentification optionnelle basée sur l' algorithme à clé secrète DES **[NBS77]**. La carte CQL fournit également la traçabilité sur les modifications, et un mécanisme transactionnel simple qui permet de défaire des ordres exécutés en cas de rupture de la connexion (par exemple l'arrachement de la carte).

## *2.3 Cartes bases de données et cartes multi-services : avantages et inconvénients*

L' avantage des cartes bases de données est la structuration globale des données. Ces dernières offrent un outil de manipulation de données puissant et des mécanismes de sécurité et de partage fins et évolués. Cependant, le point faible de ces cartes est la quasi inexistence d' applicatif encarté. L' applicatif, comme nous l' avons dit auparavant, doit être situé sur le système hôte de chaque intervenant. L' absence de traitements encartés limite fortement l' extension d' utilisation des dossiers portables à d' autres types d' applications plus complexes. Enfin, le second point faible de ce type de systèmes d' exploitation carte est l' absence de langage de manipulation de données procédural.

Les avantages des cartes multi-services sont évidents. La polyvalence confère plus d' ergonomie à l' utilisateur et l' évolutivité du contenu permet d'autre part d'« oublier » la limite de stockage de la carte. L' inconvénient réside dans la gestion des données. En effet, il n' existe pas à proprement parler de structuration globale. Dans la plupart de cas, les données sont indissociables des traitements. Il n' existe donc pas non plus de support de manipulation simple et global de données. Enfin, il n' existe pas non plus de gestion fine des droits sur les données en vue d' en effectuer le partage.

# 3 Concept de Carte Hybride

Après avoir présenté les cartes bases de données et les cartes multi-services, nous allons énoncer dans la suite les concepts de la carte hybride qui réunit les deux orientations.

## *3.1 Les trois modes logiques de fonctionnement*

Selon le type d'utilisation, la carte hybride doit pouvoir être vue de 3 manières différentes : carte base de données, carte multi-services ou carte multi-services avec données structurées. Ces trois vues logiques sont présentées sur la figure 2 qui illustre une carte hybride basée sur CQL et Java Card. Dans le premier cas, l'utilisateur (i.e. le porteur ou serveur distant) utilise la carte en tant que carte base de données en envoyant des APDU CQL qui sont traitées par le moteur CQL interne. Dans le second cas, les APDU envoyées

correspondent à des appels de services (cas de l'APDU A0::C1) utilisant uniquement des données persistantes privées (situées dans l'espace cloisonné de l'application et géré par le support d'éxécution d'applets). La carte est utilisée en tant que carte multi-services « classique ». Enfin, dans le troisième cas , les APDU envoyées correspondent toujours à des appels de service (cas des APDUs A1::C1 et A2::C2), mais ces services sont conçus pour utiliser des données au travers de la base de données. Un autre cas, qui se ramène au précédent, est celui d'un service qui manipule la base de données au travers d'un autre service.

## *3.2 Mode multi-services avec accès à la base de données*

Dans le dernier cas présenté dans la sous-section précédente, les données du service appelé font partie intégrante de la base de données. Une même manipulation sur une donnée opérée soit directement de l'extérieur, soit par un service, doit être traitée de manière uniforme et avoir les mêmes effets. Dans les deux cas, ces données doivent être manipulées par le moteur de la base de données afin de maintenir leur cohérence et de contrôler les droits d'accès.

Lorsqu'un service manipule une donnée de la base, nous proposons que cela se traduise par une APDU base de données simulant ainsi un envoi de l'extérieur (voir figure 2). Cela implique qu'il faille fournir un moyen d'exprimer dans le code des services cartes l'envoi d'APDU base de données.

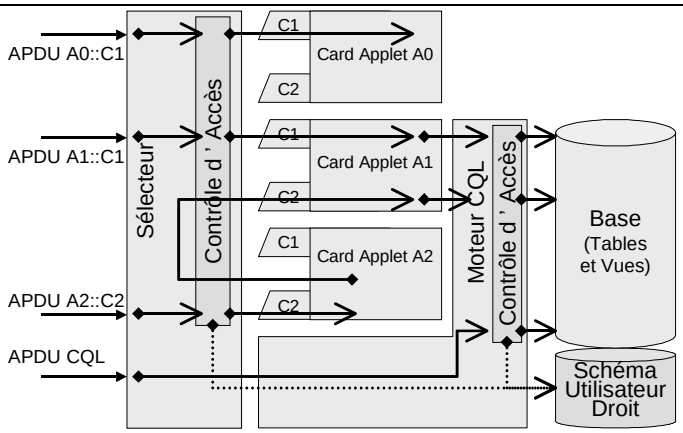


Figure 2: La carte hybride

## *3.3 Contrôle d'accès*

La carte hybride doit offrir à la fois un contrôle d' accès aux données compatible avec celui de CQL et respecter le "business model" des cartes multi-services en laissant la possibilité à l' émetteur de la carte d' autoriser des prestataires de services carte à y installer des applications carte et d' autoriser son porteur à invoquer les services installés. Le moteur de la base de données gère une liste d'utilisateur qu'il est capable d'identifier et leurs privilèges sur les objets de la bases afin de pouvoir en contrôler les accès. Nous proposons alors d'étendre le contrôle d'accès aux commandes des services encartés.

### 3.3.1 Utilisateurs et identification

Le contrôleur d' accès de la carte hybride doit pouvoir identifier les utilisateurs locaux ou distants par des moyens d' identification classiquement mis en œuvre dans les cartes : saisie d' un PIN code avec ratification (i.e. blocage de la carte au $3^{ème}$ essai infructueux) pour le porteur et authentification des commandes des systèmes distants par les méthodes cryptographiques présentes dans la carte **[S96]**. Dans certains cas, l'utilisateur n'est pas identifié lors de l'envoi de la commande. Ceci est vrai notamment lorsque la commande est publique ou lorsqu'elle dispose de son propre mode d' identification (cas de la commande A0::C1 de la figure 2). L'utilisateur étant anonyme, il est nécessaire pour des raisons de sécurité de distinguer si ce dernier est distant ou local. L'information de provenance doit être fournie par le terminal.

### 3.3.2 Privilèges et objets

La carte CQL dispose déjà d' un mécanisme de contrôle d' accès sur deux types d'objets (TABLE, VIEW) qui se base sur celui de SQL. La carte hybride complète ce contrôle d' accès sur les données par un contrôle d' accès sur un troisième type d'objet: les commandes des applications (PROCEDURE). Ce complément s' inspire lui aussi des SGBDs car les commandes des applications cartes installées sont considérées par la carte hybride comme les procédures stockées le sont dans ces SGBDs. Les services sont alors vus comme un regroupement de procédures stockées.

- Ainsi une application (avec ses commandes) est installée par un utilisateur identifié par la carte si l' utilisateur émetteur de la carte lui en a donné (GRANT) le privilège (CREATE PROCEDURE),
- et une commande d' une application peut invoquée par un utilisateur identifié ou anonyme par la carte si l' utilisateur installateur lui en a donné (GRANT) le privilège (EXECUTE).

Dans ce contexte, la carte hybride dispose d' un administrateur qui sera vraisemblablement l' émetteur de la carte (plutôt que son fabricant). L' émetteur définit la liste des utilisateurs porteurs ou distants avec leur mode d' identification (PIN Code, Certificat, …) et spécifie les privilèges pour chacun d' eux : par exemple, un prestataire peut installer une application carte. A son tour, le prestataire peut spécifier des privilèges sur les objets qu' il crée (TABLE, VIEW et PROCEDURE) : par exemple, il peut autoriser le porteur à invoquer une commande d' un service. La délégation de privilège (GRANTEABLE) peut aussi être envisageable mais elle devra être configurée avec précaution par l' administrateur et les prestataires.

### 3.3.3 Changement d' identité pour l'exécution

Un utilisateur I propriétaire d' une application installée peut accorder une partie des privilèges à un autre utilisateur U durant l'exécution d'une commande de cette application. Cet accord de privilège est différent de la délégation de privilèges (GRANTEABLE). En effet, l'accord de privilège se base sur le changement d'identité de l'utilisateur U pour l'identité de l'utilisateur I propriétaire/installateur de l'application. La principale utilisation du changement d'identité est de permettre à une commande d'accéder à des objets (i.e. tables, vues et commandes/procédures) pour lesquels l'utilisateur U n'a pas les privilèges requis. Dans l'exemple de l'application Loyalty de la figure 4, l'utilisateur U a le privilège de consulter les lignes de la table Service pour déterminer le total de points accumulés soit par une commande APDU CQL, soit par la commande Loyalty::getBalance(). Cependant, il ne possède pas le privilège de modification (UPDATE) des points accumulés qui est détenu par l'utilisateur I. La modification sera néanmoins possible via la commande Loyalty::addPoint() qui utilise l'identité de l'utilisateur I propriétaire de la table Service pour effectuer la modification. Ce changement d'identité pour accéder à des données est utilisé notamment par Unix **[T89].**

# 4 Mise en œuvre de la Carte Hybride

La mise en œuvre de la carte hybride doit reposer sur la réutilisation des composants logiciels existants et prouvés sûrs compte tenu de leurs domaines d' application. Ces deux composants sont d' une part un système carte multi-services (Java Card, Windows SC) offrant un langage général (Java, VBScript) et d' autre part un moteur du gestionnaire de bases de données comme celui de la carte CQL ou celui annoncé par Centura. Dans la suite de l'article, nous utiliserons la Java Card et le moteur CQL pour décrire la mise en œuvre.

## *4.1 Accès à la base par les applications carte*

Nous discutons dans cette section de l' intérêt des trois propositions suivantes pour permettre aux applications carte de requêter la base de données :

- L' intégration d' un middleware SQL/CLI dans une carte multi-services,
- La définition d' une API de construction APDU CQL évitant l' analyse de la requête par la carte,

- Le précompilateur "Embedded CQL for X" offrant une syntaxe plus concise et une analyse statique des requêtes.

### 4.1.1 Middleware SQL/CLI

Les middlewares SQL/CLI sont largement connus des développeurs d' applications client/serveur. Ces middlewares sont par exemple ODBC pour les applications clientes sur plate-forme Windows ou JDBC pour les applications ou les applets Java. Ces middlewares offrent un accès uniforme à des sources de données locales (bases MS Access ou Dbase) et à des serveurs bases de données distants (Oracle, Informix, …). L' indépendance de l' application au format de la source donnée et à l' interface native du SGBD, réside dans l' utilisation de pilotes adaptés. On peut remarquer qu' il existe déjà des pilotes ODBC et JDBC pour requêter une carte CQL insérée dans le lecteur du terminal (PC, NC). Du point de vue du développeur, l' ordre SQL est exprimé sous la forme d'une chaîne de caractères qui devra être analysée dynamiquement par le moteur de gestion de base de données lors de l' exécution. On parle alors d' invocation dynamique de requêtes.

La première proposition de carte hybride consiste à porter le middleware dans la carte et de développer un pilote carte pour la CQL. Dans le cas d' une Java Card, la carte doit contenir les classes JDBC ainsi que le pilote JDBC pour CQL. Cette proposition permet au développeur carte d' utiliser l' API JDBC qu' il connaît bien. Cependant son principal inconvénient est l' importante occupation en mémoire. En effet, du fait des invocations dynamiques de requêtes, le middleware carte contient au minimum un analyseur qui transforme la chaîne de caractères dynamique en ordre CQL. Dans le cadre des applications carte, cette dynamicité offre une facilité relative d' écriture du code mais il déporte dans la carte un gros travail qui pourrait être réalisé à l' extérieur de la carte avant l' installation de l' application.

### 4.1.2 API APDU CQL

La seconde proposition consiste à définir une interface (API) de construction des requêtes évitant une phase d' analyse syntaxique. Cette interface permet de construire les APDU CQL dans un langage de programmation d' application carte (C, Java, VBScript). Dans le cas de la Java Card, cette interface repose sur les JNI (Java Native Interface) pour invoquer le code objet du moteur CQL. La construction de l' ordre APDU se fait en plusieurs étapes en spécifiant les tables sources, les colonnes de projection et les conditions de restriction (rappelons que l' opérateur de jointure n' existe pas dans la CQL). La récupération des lignes du résultat se fait au moyen d' un curseur. Cependant, cette proposition a deux inconvénients. Premièrement, l' invocation de la requête reste dynamique et donc l' existence des tables, des vues et des colonnes n' est vérifiée qu' au moment de l' exécution; ainsi le développeur n' a pas d' autre moyen que d' installer et de tester son application pour vérifier que toutes les APDU construites ne conduisent pas à une erreur sémantique. D' autre part, la construction des APDU reste une opération fastidieuse pour le développeur.

### 4.1.3 Précompilateur "Embedded CQL for X"

La troisième proposition pallie aux deux inconvénients de la proposition précédente. Cette proposition consiste à utiliser un précompilateur pour insérer les ordres de construction des APDUs spécifiées dans l' interface présente à partir d' ordres CQL nichés (embedded) dans le programme écrit en C, Java ou VB. La figure 3 présente la chaîne de compilation et d' installation d' une application carte. Le précompilateur contrôle aussi l' existence des colonnes et des tables ainsi que la correspondance des types des colonnes et des variables du programme, à partir du schéma de la base contenue dans la carte dans laquelle l' installation aura lieu. Ce contrôle peut être optionnel quand le schéma n' est pas connu à l' avance. De plus, la syntaxe des ordres CQL nichés est plus concise que dans les deux propositions précédentes et soulage le travail du développeur.

Dans le cas de la Java Card, le précompilateur que nous baptisons CQLJ s' inspire sur le principe et sur la syntaxe du précompilateur SQLJ proposé par Oracle, IBM, Sybase, Tandem à l 'ISO/ANSI pour nicher des ordres SQL dans des applications et des applets Java (i.e. "Embedded SQL for Java"). Pour mémoire,

SQLJ génère un code Java contenant des appels à JDBC. L' exemple de la figure 4 présente une application de fidélité nichant des ordres CQLJ.

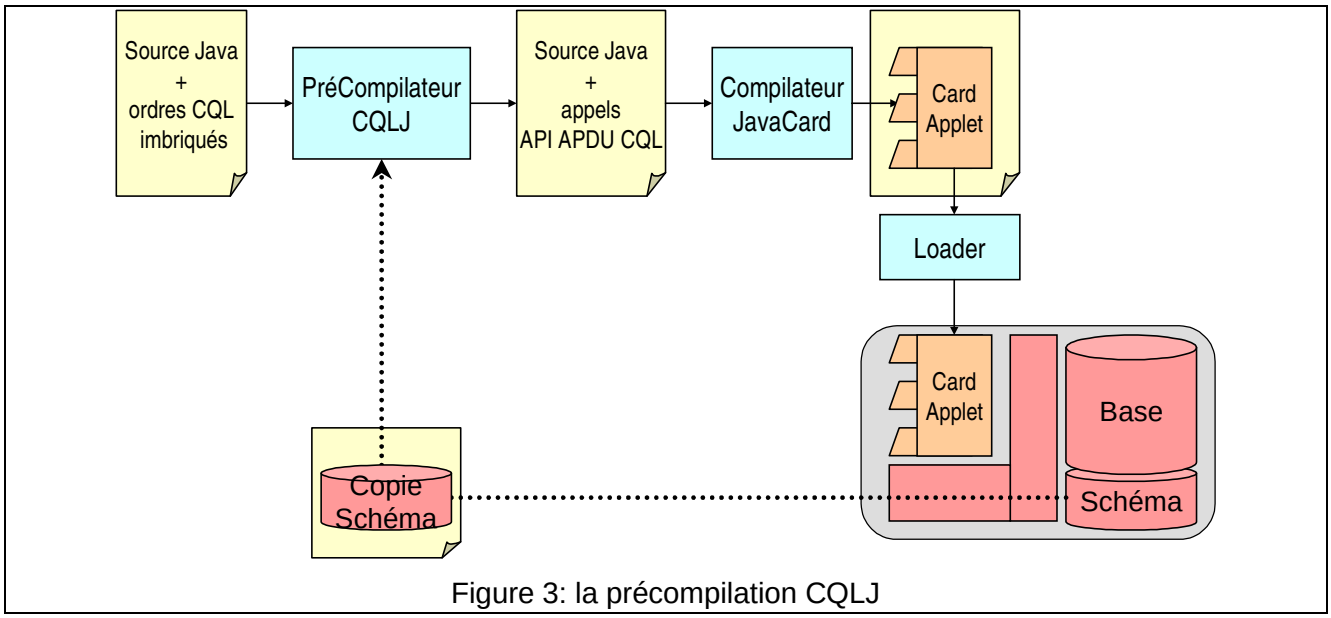


Figure 3: la précompilation CQLJ

## *4.2 Contrôle d'accès*

Dans notre proposition de mise en œuvre de la carte hybride, le mécanisme de contrôle d' accès doit aussi tenir compte de l' existant comme nous l' avons fait dans la section 4.1. D' une part, la CQL possède un mécanisme de contrôle d' accès uniquement sur les données (TABLE). D' autre part, l' installation d' une application carte obéit déjà à un protocole de chargement établi. Pour ces deux raisons, nous avons choisi un compromis permettant à un installateur d' application (i.e. le prestataire de service) de positionner de manière optionnelle des privilèges sur les commandes de l' application installée.

- L' utilisateur installateur spécifie au moyen d' une commande APDU CQL les privilèges des utilisateurs autorisés à accéder à une commande. Ces autorisations sont stockées dans la table système des privilèges des procédures. Ce serait à priori le cas de la commande addPoint() dans l' application Loyalty (figure 4).
- En l' absence d' une entrée dans cette table, la procédure est considérée publique et peut être invoquée par tout utilisateur identifié ou anonyme. Ce serait à priori le cas de la commande getBalance() dans l' application Loyalty (Figure 4).

```
import JavaCard.framework.*;

#cql public iterator IterService (String, int);
    // déclaration d 'une classe d'itérateur

public class Loyalty extends Applet implements ILoyalty {

// Constructeur de l'application invoqué une seule fois
// à l'installation de l'application dans la carte
 Loyalty() throws CQLException {
   // l'application carte utilise deux tables de la base CQL :
   //   Service et HistoryAdd
   #cql { ATOMIC BEGIN
            CREATE TABLE Service(name CHAR(8),point INT);
            CREATE TABLE HistoryAdd( servicename CHAR(8),
                                previouspoint INT, number INT, date DATE);
```

```
          END; }
    };
// Commande APDU pour ajouter de points de fidelité pour un service
  int addPoint(String servicename, int number, JavaCard.cql.Date date)
                    throws CQLException {
    int previouspoint;
    #cql { ATOMIC BEGIN
             SET USER %APPLICATIONOWNER
             SELECT point INTO :previouspoint FROM Service
               WHERE :name = servicename;
             UPDATE Service SET point := point + :number
               WHERE :name = servicename;
             INSERT INTO HistoryAdd
               VALUES (:servicename, :previouspoint, :number, :date);
           END; }
    };

// Commande APDU retournant le total de points de fidelité de la carte
  int getBalance() throws CQLException {
      // SELECT SUM(point) INTO :sumpoint FROM Service
    String name; int point, int sumpoint=0;
    IterService iter; // déclaration d 'un objet itérateur
    #cql iter = { SELECT name, point FROM Service }
    while (true) {
      #cql { FETCH :iter INTO :name, :point }
      if (iter.endFetch()) break;
      sumpoint+=point;
    }
    return sumpoint;
  }
  ...
}
```

Figure 4 : une application carte de fidélité utilisant CQLJ

Comme par défaut une commande d' une application installée est publique tant que des privilèges n' ont pas été spécifiés pour elle, il faudra veiller à rendre atomiques les opérations d' installation et de positionnement des privilèges. En effet il faut éviter toute attaque basée sur l' arrachement de la carte entre ces deux opérations. Or l' atomicité de ces deux opérations n' est actuellement pas envisageable **[L98]**. Une solution réalisable et sûre consiste à réaliser l' opération de positionnement des privilèges avant l' opération d' installation. En cas d' arrachement précoce, cette solution peut entraîner des incohérences entre les privilèges et les applications installées ; cependant ces incohérences n' ont pas d' impact sur la vérification de privilèges cohérents. Par défaut, lors de l' invocation de la commande, tout accès à la base est autorisé si l' utilisateur possède les privilèges requis. Cependant, il est possible d' exécuter un ordre CQL avec l' identité de l' utilisateur installateur. Ceci doit être spécifié dans le code de la commande au moyen d' un ordre SET USER %APPLICATIONOWNER. Cet ordre configure l' identité de l' utilisateur utilisée par la commande pour ouvrir une "connexion" avec la base CQL (voir addPoint() dans la figure 4).

# 5 Conclusion

Les cartes à microprocesseur bases de données offrent une manière élégante d' implanter des dossiers portables sécurisés. Cependant, ce type de cartes a pour principal inconvénient de limiter les traitements à un enchaînement de requêtes. D' un autre côté, les cartes multi-services offrent un support sécurisé de traitement où plusieurs services coexistent. Ces services sont écrits dans des langages généraux comme notamment Java. Le principal inconvénient de ces cartes est la dépendance des données stockées vis à vis des traitements et du contrôle d'accès.

Cet article propose de rapprocher ces deux modèles de cartes pour conduire à la définition d' une carte hybride remplissant ces deux fonctions. Cette carte hybride offre l' indépendance entre les traitements et les données. Les données sont les tables et les vues de la base de données et les traitements peuvent être externes

sous la forme de commandes APDU CQL invoquées via le terminal ou sous la forme de commandes des applications installées. Ces applications peuvent également accéder à la base via des commandes APDU CQL internes. La carte hybride offre un mécanisme de contrôle d'accès à la fois sur les données (tables et vues via les commandes APDU CQL) mais également sur les traitements qui sont considérés comme des procédures stockées sur un serveur base de données. La carte hybride respecte ainsi le « business model » actuel de l'industrie des cartes à microprocesseur comportant plusieurs types d'acteurs identifiés ou anonymes comme le fabricant, l'émetteur, les installateurs d'application et le ou les porteurs.

La mise en œuvre de la carte hybride qui est en cours de réalisation, fait quelques compromis dans ses principes afin que l'implantation ne perturbe pas les mécanismes existants des produits utilisés (contrôle d'accès, installation d'application, …). En perspective, nous envisageons l'extension des principes de la carte hybride à la carte active **[N97]** dans laquelle les déclencheurs (triggers) sont armés sur des objets de la base de données et déclenchent l'exécution de procédures stockées lors de la modification de ces objets.

# 6 Bibliographie


[A95] T. Alexandre. *Manipulation de données multimédias dans la carte à microprocesseur : application à l'identification biométrique et comportementale.* Thèse de Doctorat en Informatique, LIFL, Université de Lille 1, 1995.

[FM96] J.S. Fritzinger, M. Mueller. *Java Security*. Whitepaper, Sun microsystems Inc. 1996.

[Gem93] Gemplus. *CQL card reference manual.*1993

[GG92] G. Grimonprez, E. Gordons. *A card as element of a distributed database*. IFIP WG 8.4 Workshop, Ottawa. 1992.

[G92] G. Grimonprez. *Etude et réalisation d'une carte à microprocesseur intégrée aux systèmes de gestion de bases de données.* Mémoire d'habilitation à l'université de Lille 1, 1992.

[ISO87] International Standard Organisation (ISO). *Cartes d'identification – Cartes à circuits intégré à contacts , Normes ISO 7816-1,2,3,4*. 1987-1995.

[ISO92] International Standard Organisation (ISO). *Information technology – Database languages - SQL, Norme ISO 9075*. 1992.

[JCard] www.javasoft.com/products/Java Card

[L98] Sylvain Lecomte. *COST STIC : Des Cartes Orientés Services Transactionnels et des Systèmes Transactionnels Intégrant des Cartes.* Thèse de Doctorat en Informatique, LIFL, Université de Lille I, 1998.

[Multos] www.multos.com

[MVD96] P. Merle, J.J. Vandewalle, et E. Dufresne. *Intégration d'environnements hétérogènes : World Wide Web, Cartes à microprocesseurs et Corba*. Inforsid 96, Bordeaux. 1996.

[NBS77] National Bureau of Standards. *Data Encryption Standard (DES)*. Federal information processing standards, FIPS-46. 1977.

[N97] Bernard Noclercq. *Carte Active à Microprocesseur à noyau CQL* . Mémoire d'Ingénieur, CNAM, Lille. 1997.

[S96] Bruce Schneier. *Applied Cryptography*. Ed. Wiley, ISBN 0-471-59756-2. 1996.

[Sun99] Sun microsystems Inc. *Java Card 2.1. Runtime Environment (JCRE) Specification*.1999.

[Sun99b] Sun microsystems Inc. *Java Card 2.1. API Specification.* 1999.

[V95] J.J. Vandewalle. *Loading several services into multi-purpose Integrated circuit card : Distribute functions to users, gather data into cards !*. In Proceedings European Research Seminar on Advances in distributed Systems, p317-322, Grenoble. 1995.

[V97] J.J . Vandewalle. *OSMOSE : Modélisation et implémentation pour l'intéropérabilité de services carte à microprocesseur par l'approche orientée objet*. Thèse à l'université de Lille I. 1997.

[T89] Andrew Tanenbaum. *Les systèmes d'exploitation : Conception et Mise en Œuvre*. Ed Interéditions. 1989.