Intégration d'un Gestionnaire de Transaction dans les cartes à microprocesseur

Sylvain Lecomte* **et Didier Donsez†**

* *Recherche et Développement du Dossier Portable (RD2P)*
Laboratoire d'Informatique Fondamentale de Lille (LIFL)
Université de Lille 1, Avenue du Pr Leclercq, 59037 Lille Cedex, France
e-mail:lecomte@lifl.fr
† *Laboratoire d'Informatique et de Mathématiques Appliquées de Valenciennes (LIMAV)*
Université de Valenciennes, Le Mont Houy BP 311 59304 Valenciennes Cedex France,
e-mail:donsez@univ-valenciennes.fr

RÉSUMÉ — Les domaines d'application de la carte à microprocesseur sont de plus en plus vastes et distribués. Notre papier propose des solutions pour mieux intégrer la carte aux systèmes existants. Après avoir constaté les contraintes technologiques imposées par le monde de la carte à microprocesseur en terme de puissance de calcul et de taille de mémoire, nous étudions les limites logicielles des systèmes actuels, pour finalement définir une nouvelle architecture du système d'exploitation de la carte. Elle est basée sur l'implémentation de transactions et d'un moniteur transactionnel à l'intérieur de la carte, et sur l'intégration des cartes dans un environnement distribué CORBA.
MOT-CLÉS : *Carte à microprocesseur, Transactions, CORBA, OTS, Intégration*

ABSTRACT — The application's scopes of the smart card are wider and wider and more and more distributed. Our paper suggests solutions to integrate the card more properly into the existing systems. After establishing the technological constraints due to computing power and memory size, we study the software limits of the actual systems to define a new architecture of the card's Operating System. This architecture is based on the implementation of both transactions and transactional monitor inside the card itself, and on the cards integration through a distributed environment CORBA.
KEY WORDS: *Smart Card, Transaction, CORBA, OTS, Integration*

1. INTRODUCTION

Depuis sa création, la carte à microprocesseur était limitée à des applications de sécurisation dans les accès (GSM, badges d'accès), et de sécurisation dans les paiements (Carte Bleue). Malgré sa faible capacité de stockage, la carte a été aussi utilisée comme dossier portable véhiculant des données personnelles

(Carte de santé du patient **[Sesam Vitale 96]**). Actuellement la carte à microprocesseur prend son essor grâce au développement de Internet commercial, comme solution de sécurisation des paiements et des accès **[EMV 96]**.

Nous allons étudier les nouvelles utilisations possibles de la carte dans plusieurs applications distribuées, telles que le vote, la réservation de billet, l'interaction Carte Patient-Carte Professionnel de santé, requiérant des mécanismes de sécurisation.

Cette étude nous permet de définir les nouvelles fonctionnalités, comme les transactions, qui devront être offertes dans les systèmes d'exploitation des cartes. Ceci nous oblige à prendre en compte les contraintes des plateformes de communication, comme par exemple CORBA. Cette étude nous conduira à décrire une nouvelle génération de système d'exploitation, contenant des outils dédiés à la gestion des transactions.

Après une étude des cartes actuelles dans la section I (caractéristiques physiques et logicielles), nous présenterons une taxinomie des nouvelles applications pour les cartes à microprocesseur dans la section II. Nous verrons dans la section III, l'intérêt du modèle transactionnel pour ces applications. Dans la section IV, nous proposerons à la fois, une nouvelle architecture du système d'exploitation carte (incluant la gestion de transactions), et une intégration de ces nouvelles cartes dans CORBA. Enfin, nous concluerons sur les perspectives de ces travaux.

2. LES CARTES A MICROPROCESSEUR

Les cartes à microprocesseur actuelles, sont très limitées par leurs caractéristiques matérielles et logicielles. Malgré cela, le monde des cartes est en pleine évolution, et la carte à microprocesseur est de plus en plus présente dans les systèmes distribués.

2.1. Les contraintes technologiques

Une carte à microprocesseur est en fait un micro-ordinateur qui tient sur une surface de moins de 30 mm2, et sur une épaisseur de moins de 0,28 mm. Cette partie se nomme le micro-module. Dans cette surface, se trouvent le microprocesseur, la RAM, la ROM, la mémoire EEPROM, le bloc de sécurité et le gestionnaire d'entrées/sorties (Figure 1). Le tout dans un bloc monolithique. Cela implique un grand nombre de contraintes technologiques.

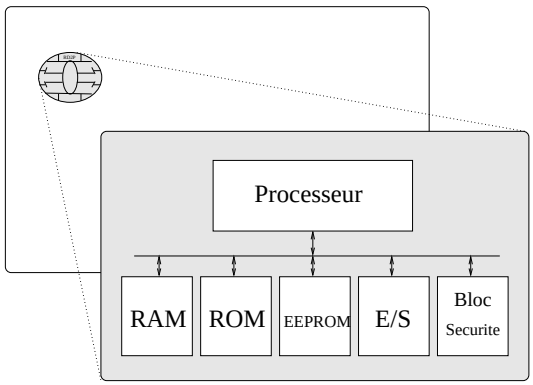

Figure 1. *Module de Carte*

- Le microprocesseur est généralement un processeur 8 bits du type Motorola 6805,

- La mémoire ROM, qui contient le système d'exploitation, a une capacité d'environ 8 KO,
- La mémoire RAM a une capacité de 512 octets maximum,
- l'EEPROM, qui est le support de stockage, a une capacité totale maximale de 16 KO,
- Les Entrées/Sorties se font via un port à 19200 BDS,
- Le bloc de sécurité doit assurer les contrôles relatifs à la sécurité physique de la carte.

Ces éléments sont reliés à un bus de 8 bits.

Malgré ces évidentes restrictions physiques, les nouvelles cartes sont de plus en plus utilisées dans le monde des systèmes distribués intégrant les technologies des bases de données (grâce à des cartes comme la CQL), ou bien encore dans les ORB comme CORBA.

2.2. Cartes et architectures existantes

2.2.1. La carte CQL

La carte CQL (Card Query Language), qui est basée sur des travaux menés à RD2P **[Gordons 92]** adopte les concepts de SGBD **[Ozsu 91]** et utilise des commandes SQL **[ISO 9075]**. Cette carte a été améliorée et industrialisée par la société Gemplus en 1993 **[Gemplus 93]**.

Cette carte peut être vue comme un serveur individuel portable de données, très sécurisé, dont la première caractéristique est de contenir plusieurs tables SQL et un moteur de SGBD résident en ROM. Les requêtes qui sont effectuées sur une des tables sont *SELECT, CREATE TABLE, DROP TABLE, INSERT, DELETE, UPDATE, DECLARE CURSOR, FETCH*. Il est aussi possible de définir différentes vues sur une table (*CREATE VIEW , DROP VIEW*), ainsi qu'un dictionnaire qui contient la structure de la carte.

La carte CQL est également munie de fonctions de contrôle d'accès et de maintien de l'atomicité.

- Contrôle d'accès : les accès des utilisateurs à la carte peuvent être sécurisés à 2 niveaux différents. On peut soit demander simplement un mot de passe, soit utiliser une double authentification, qui repose sur l'algorithme à clé secrète DES **[Schneier 96]**.
- Le maintien de l'atomicité : pour la première fois dans une carte à microprocesseur, il est possible de rendre indivisibles des actions exécutées dans le cadre d'une transaction. Ceci a nécessité l'implémentation de 3 ordres à l'intérieur de la carte (*BEGIN TRANSACTION, COMMIT, ROLLBACK*). De part la taille du buffer utilisé, la taille maximale de la transaction CQL est de 5 modifications (insert, erase ou update). Ces ordres implémentent le protocole de validation à une phase.

La carte CQL a marqué l'entrée de la carte à microprocesseur dans le monde des systèmes distribués. Depuis, d'autres applications, que nous décrirons plus loin dans ce papier, sur la base de cette carte, ont rendu la carte à microprocesseur partenaire des systèmes distribués.

Une nouvelle évolution de la carte CQL est en cours de développement **[Noclercq 97]**. Le but de ces nouveaux travaux est de faire effectuer automatiquement des traitements à la carte, sans que le porteur en ait pleinement conscience. Les exemples d'applications de cette nouvelle technique concerne notamment le Porte Monnaie Electronique (calcul automatique dans différentes devises), ou l'aide médicale (interactions médicamenteuses et contre indications).

2.2.2. Intégration des cartes dans les systèmes d'information

Des travaux ont été menés à RD2P sur l'intégration des cartes à microprocesseur dans les Systèmes d'Information Communicationnels [1] **[Van Hoecke 96]**. Ce travail définit une architecture distribuée comprenant le réseau fixe (constitué des serveurs classiques), les machines de connexion (qui sont reliées au réseau), et les cartes (qui viennent se connecter dans les machines de connexion).

Pour la première fois, la carte est directement intégrée dans les systèmes d'information classiques, et une des conclusions de ce travail est que la conception d'un système d'information carte ne doit pas fondamentalement différer d'un système "traditionnel".

2.2.3. La carte dans l'environnement CORBA

CORBA est un bus de communication entre objets répartis **[OMG 97]**. Ces objets sont des applications ou des services accessibles suivant le modèle client/serveurs. Les applications construites sur une architecture Corba sont facilement portables et interopérables. Il était donc tentant d'intégrer la carte à microprocesseur dans cet environnement. Cette intégration a fait l'objet de récents travaux menés à RD2P avec le projet OSMOSE (Operating System and Mobile Object SErvices) **[Vandewalle 97]**. Les objectifs de ces travaux sont de définir une carte à microprocesseur générique qui puisse charger des services au cours de son utilisation, et d'étudier les services de mobilité (des utilisateurs et des machines) intégrant des cartes.

Le principal problème de cette intégration a été d'encapsuler la carte, dont le protocole de communication est défini par la norme 7816-4 de l'ISO **[ISO 7816]** : qui impose l'utilisation des ordres APDU (Application Protocol Data Unit) pour communiquer avec l'extérieur. L'intégration de la carte dans cet environnement se fait par le développement d'une passerelle (un objet d'adaptation), qui propose des services offerts par la carte : cet objet dialogue suivant le protocole imposé par l'ISO. L'architecture d'intégration de la carte dans CORBA est donnée par la figure 2.

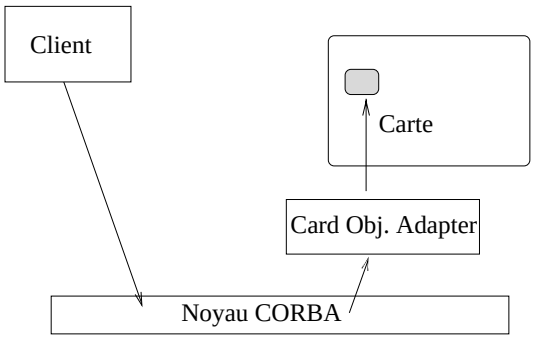

Figure 2. *Intégration de la Carte dans un ORB*

2.3. Limites actuelles

Grâce à des techniques comme la carte CQL, ou l'intégration des cartes dans un environnement distribué comme CORBA, la carte à microprocesseur participe de plus en plus à des applications distribuées. Ce phénomène devrait s'amplifier dans les années qui viennent, notamment avec le développement de l'Internet

[1] système auquel est dévolue la prise en charge de la gestion des dialogues entre des grandes organisations et des particuliers

commercial, qui requiert une sécurisation des paiements (comme nous allons le voir dans le paragraphe suivant).

Cependant, la carte à microprocesseur n'est prise en compte actuellement que dans des architectures client/serveurs classiques. Elle est uniquement connectée à son terminal. Il lui manque la possibilité de participer à des transactions distribuées. l'intégration dans la carte d'un mécanisme de gestion des transactions, ouvrirait beaucoup plus le champ de ses applications. La carte doit collaborer plus étroitement avec les autres participants (ce qui est impossible actuellement de part son mode de fonctionnement le plus souvent déconnecté), et on pourrait même envisager des collaborations entre plusieurs cartes.

3. NOUVELLES APPLICATIONS CARTE

Les applications opérationnelles engageant des cartes à microprocesseur se situent principalement dans les domaines de la santé (Carte patient, Carte Médecin), du monétaire (Carte Bancaire, Porte Monnaie Electronique), des télécommunications (Carte GSM, Carte de pré-paiement), de fidélisation de la clientèle. Les limitations des cartes dédiées ont conduit le monde de la carte à microprocesseur à considérer l'existence de plusieurs services à l'intérieur d'une seule et même carte, ainsi que des applications pendant lesquelles plusieurs cartes sont utilisées.

Ces nouvelles applications demandent aux cartes à puce des fonctionnalités issues des systèmes à objets répartis et des systèmes transactionnels.

3.1. Les cartes multi-applicatives

Dans tous les domaines décrits précédemment, la carte est actuellement dédiée à une application unique. Ceci impose aux porteurs d'avoir beaucoup de cartes sur eux (une par application).

Une solution à ce problème est la carte multi-applicative, pour que le porteur n'ait plus, au maximum, qu'une carte par domaine d'utilisation. Par exemple, une carte monétaire pourra comporter un bouquet de services : Porte Monnaie Electronique, Réseau Bancaire national, Réseau bancaire international et services propres à la banque du porteur. De même, une carte à vocation santé regrouperait les services de la sécurité sociale, de la mutuelle et le carnet de santé de l'assuré porteur.

Cependant, la définition d'une carte multi-applicative pose plusieurs problèmes :

- le problème de la gestion du contrôle des accès aux données internes de la carte. Quand les applications sont installées dans la carte par des prestataires différents, chaque application dispose de ses propres données, qui ne doivent pas être connues par les autres applications (sauf en cas de partage explicite et contrôlé de ces données). En reprenant l'exemple de la carte santé, le contenu du carnet de santé ne doit pas être lisible de l'application installée par l'organisme de mutuelle.

- Le problème des accès concurrents et simultanés aux données partagées. Prenons l'exemple d'une carte monétaire qui regrouperait un Porte Monnaie Electronique et une Carte Bancaire. Lorsque le montant du Porte Monnaie Electronique est insuffisant pour effectuer un achat, celui-ci est crédité par une application de virement du compte bancaire. Cela est actuellement impossible : dans les divers propositions de cartes multi-applicatives, les services sont cloisonnés, entre autres, pour des besoins de sécurité. Donc aucune coopération n'est prévue entre les différents services installés sur la carte **[Vandewalle 97]**.

3.2. Les applications multi-cartes

Jusqu'à présent, une application n'engageait qu'une seule carte. On voit apparaître maintenant des applications qui mettent en jeu plusieurs cartes à microprocesseur. Ce sont ce que l'on appelle les applications multi-cartes.

Dans le cadre de ces applications, il y a deux variantes :

- Dans le premier cas, un même porteur dispose de plusieurs cartes. Nous sommes donc dans le cas où plusieurs cartes doivent coopérer. Ce cas n'est pas très éloigné du problème des cartes multi-applicatives. Là aussi, plusieurs services cartes doivent coopérer pour fournir un résultat. La différence réside dans le fait que les services sont sur plusieurs cartes. Un exemple d'application concerne les droits d'accès à des données. Par exemple, la carte d'un médecin donne des droits sur la carte d'un patient. Il y a donc bien coopération entre les deux cartes.

- Dans le deuxième cas, plusieurs porteurs se donnent rendez-vous pour coopérer. Cela ne signifie pas que tous les porteurs doivent se connecter en même temps. Prenons le cas d'une application de vote électronique. Cette application pose de nombreux problèmes, comme le besoin d'anonymat du vote, la non-répudiation, ou encore le droit de voter une seule fois **[Schneier 96]**. De plus, chaque tour peut durer plusieurs jours, et c'est pendant ce laps de temps, que chaque porteur doit venir se connecter pour voter. Le problème dans ce cas, est la validation du vote. Comment le porteur d'une carte de vote va savoir si son action a été validée ou non? La validation ou l'invalidation d'un vote peut conduire l'incohérence de certaines données internes à la carte (par exemple, la durée de validité de la carte). Ce type d'applications, qui durent dans le temps, est repris dans la suite, car il nécessite un contexte global persistant.

3.3. Les applications persistantes

La durée de connexion de la carte est relativement courte. Mais, certaines applications peuvent durer dans le temps, et nécessiter plusieurs connexions. Dans l'exemple de l'application de vote citée précédemment, les votants utilisent leur carte à chaque tour d'une élection.

Une autre application persistante est une application de réservation de billet (transport, spectacles, ...) au moyen d'une carte monétaire. Cette réservation se fait en deux étapes.

- 1ère étape : Le porteur de la carte réserve son billet à distance à l'aide de sa carte. La carte signe alors la demande de billet, et une provision est faite, soit depuis le Porte Monnaie Electronique, soit depuis le Compte Bancaire.

- 2ème étape : Le porteur se reconnecte sur un terminal de la société dans laquelle il a effectué sa réservation. La carte redonne sa signature, et l'application peut continuer. Le compte où l'argent a été bloqué, est alors réellement débité, et le billet est édité.

Dans ces deux exemples, l'application est persistante durant la déconnexion de la carte.

Actuellement, une carte ne sait pas gérer ce type d'application. Dans le cas de l'application de réservation, l'utilisateur doit pouvoir continuer à utiliser sa carte, même après la première phase de l'application (après la provision depuis le compte). D'autre part, on doit s'assurer qu'il reste suffisamment d'argent sur le compte pour valider la transaction et éditer le billet, lors de la deuxième phase de la transaction.

Nous ne traiterons pas les problèmes liés à ce type d'application dans cet article.

4. BESION TRANSACTIONNEL

Ces nouvelles applications requièrent des fonctionnalités bien connues des systèmes d'exploitation et des systèmes transactionnels mais qui restent nouvelles pour le monde des cartes à microprocesseur.

4.1. Besoins Multi-Applications

Le développement de telles applications est grandement facilité par les fonctionnalités offertes par les cartes génériques. Ces cartes permettent à des prestataires différents de charger et de décharger [2] des bouquets de services de façon temporaire : le porteur d'une carte monétaire peut demander à un prestataire le chargement temporaire d'un service de paiement adapté au pays dans lequel il se rend pendant la durée de son séjour. Les cartes génériques offrent des mécanismes de sécurité pour le chargement et pour l'exécution des services chargés. Ces mécanismes reposent actuellement sur l'interprétation du code des services en attendant des mécanismes matériels [3]. Ainsi, les membres du JavaCard Forum, ont défini une carte basée sur l'interpréteur Java de Sun **[Guthery 97]**, dont les limites actuelles sont le manque de Thread, le manque de gestion des exceptions, ou encore le jeu d'instructions, sous ensemble de Java, relativement réduit.

4.2. Besoins Multi-Contextes

Les cartes génériques offrent principalement de la flexibilité pour le développement d'applications à base de cartes à microprocesseur. Cependant, les utilisateurs réclament la possibilité d'utiliser simultanément plusieurs services d'une carte : par exemple, il existe des projets d'intégration d'un porte-monnaie électronique sur une carte d'abonné GSM pour des télé-paiements. Les applications de ce type sont actuellement "bricolées au couteau" : la gestion de plusieurs contextes d'exécution est une fonctionnalité à offrir dans les prochains systèmes d'exploitation carte. Actuellement, la carte est mono-tâche : donc un seul contexte s'exécute à la fois (car actuellement, de part la taille de la RAM, un changement de contexte dans une carte implique un swap complet de la RAM en EEPROM). La gestion de plusieurs contextes requiert un partage collaboratif ou préemptif [4] des ressources par les tâches simultanées.

4.3. Besoins systèmes

Le support multi-contexte est nécessaire mais pas toujours suffisant. En effet, la plupart des applications font intervenir plusieurs composants (dont au moins un dans la carte) s'exécutant de manière distribuée dans un contexte sujet aux fautes (de l'encombrement du réseau à l'arrachement de la carte du lecteur par l'utilisateur). Dans ce contexte, le traitement de ces fautes par le développeur d'applications se révèle être un vrai casse-tête. Le modèle des Transactions définit un modèle d'exécution d'applications simultanées qui garantit les propriétés ACID [5] des exécutions par des mécanismes de reprise sur panne **[Haerder 83]** et de contrôle de concurrence **[Bernstein 87]**. Ce modèle simplifie largement le travail du développeur.

Le modèle transactionnel offre également un bon modèle de cohérence lors d'accès simultanés sur des données partagées entre 2 applications (transactions). Cependant, la cohérence forte (ou sérielle) apportée

[2] installer et désintaller

[3] i.e. processeur avec modes superviseur/utilisateur, MMU (Memory Management Unit) protégées

[4] par décrémentation de quantum du processeur à chaque instruction interprétée

[5] Atomicité (l'ensemble des modifications sont toutes validées ou toutes abandonnées par l'application), Cohérence (une transaction prend des données cohérentes, effectue des actions et rend les données dans un autre état cohérent), Isolation (les modifications non validées ne sont pas accessibles aux applications), Durabilité (les modifications validées sont accessibles même après une panne)

par ce modèle en cas de partage, peut conduire à retarder la réponse d'une des applications simultanées.

Ce modèle reste un bon point de départ pour implanter des méthodes de transactions avancées **[Elmagarmid 91]** comme les transactions emboîtées, les SAGAS, les transactions coopératives **[Chrysanthis 91] [Biliris 94]**, ou les transactions longues (notamment pour gérer les applications persistantes).

Un autre argument en faveur du modèle transactionel pour le contrôle de l'exécution d'applications distribuées, est qu'il est supporté par des outils standardisés comme les moniteurs transactionnels **[XOpen 93] [ISO 10026]** permettant la validation ou l'abandon de tous les effets d'une application distribuée.

Rappelons que dans le contexte applicatif carte, la carte doit piloter la validation distribuée et sécurisée de l'ensemble des transactions composant l'application et donc couvre une partie des fonctions de moniteur transactionnel.

4.4. Sécurisation des Terminaisons de Transactions Distribuées

L'application distribuée se modélise par une transaction distribuée constituée d'une transaction principale, qui a été lancée par le site initiateur de l'application, et des sous-transactions invoquées sur les autres sites, par la transaction principale ou d'autres sous-transactions.

Le site initiateur peut être une des cartes engagées dans la transaction distribuée ou bien un site "non-carte". Ces invocations forment un graphe de transactions qui doivent être, soit toutes validées ou soit toutes abandonnées à la terminaison de la transaction distribuée : cette règle permet de maintenir les propriétés ACID. La validation et l'abandon de la transaction sont pilotées par des protocoles comme la Validation à 2 phases (2PC) ou comme la Validation à 3 phases (3PC) **[Skeen 81]**. Ces protocoles sont pilotés par un site central de coordination.

Dans les nouvelles applications carte, la transaction distribuée s'exécute par fragments sur les machines (sites) des participants qui ne sont pas forcément confiants les uns dans les autres : ceux-ci peuvent potentiellement se dédire des opérations effectuées.

Prenons le classique exemple d'une transaction inter-bancaire de débit-crédit. La banque X débite un compte A pour créditer un compte B de la banque Y : que se passe-t-il si un coordinateur Z se comporte comme un général Byzantin, et demande à Y d'abandonner la sous-transaction crédit et demande à X de valider la sous-transaction débit?

Dans ce contexte de sécurité, il est nécessaire que le coordinateur suive scrupuleusement l'algorithme du protocole de validation utilisé, et signe (authentification, certification) les ordres qu'il enchaîne pour la réalisation du protocole (2PC ou 3PC) : en effet, le coordinateur pourrait forcer certaines parties de l'application à valider quand d'autres abandonneraient (attaque byzantine). En plus de la signature, certains participants de la transaction distribuée peuvent exiger que le coordinateur soit exécuté par un site de confiance. Pour une application résidente sur une carte, le site de confiance peut être une machine identifiée du prestataire qui a chargé l'application résidente dans la carte, ou bien la carte elle-même quand la connexion avec le serveur est coûteuse ou hasardeuse. Dans la transaction inter-bancaire de débit-crédit entre X et Y, la banque X (le débiteur) peut exiger qu'une de ces machines soit le coordinateur. Cette machine peut être la carte émise par la banque X dans les cas ne nécessitant pas de connexion. Remarquons enfin que la banque Y dans le cas d'une sous-transaction de crédit n'a pas d'exigence en terme de validation. Dans le cas où plus d'un participant exige la coordination de la validation, il est nécessaire de passer par un tiers de confiance offrant un service de coordination signé et daté comme ce que proposent les "notaires électroniques" (certifying authorities) du commerce sur Internet.

Il est donc nécessaire d'implanter dans la carte transactionnelle les fonctions de coordination que proposent les moniteurs transactionnels **[ISO 10026]] [XOpen 93] [OMG 96]** mais également des mécanismes d'autorisation pour les coordinateurs des validations. Ces mécanismes utilisent largement les certificats X509.

4.5. Besoins des applications décrites

Pour toutes les applications définies précédemment, nous avons examiné les besoins en terme de chargement de code dynamique, d'utilisation simultanée de plusieurs services (aspect multi-contexte), de possibilité de valider ou d'annuler la transaction (Commit et Rollback), de contrôle de concurrence, et de présence d'un coordinateur de validation (figure 3).

	Chargement	Multi-Contexte	Refaire / Defaire	Ctrl Concurrence	Coordination 2PC
Carte Actuelle	Non	Non	Non	Non	Non
C.Q.L.	Non	Non	Oui	Oui	Non
JavaCard	Oui	Non	Non	Non	Non
Appli. Monetaire	Opt.	Oui	Oui	Oui	Oui
Reserv. Billet	Opt.	Oui	Non	Oui	Oui
Vote	Opt.	Oui	Oui	Non	Oui

Caracteristiques cartes actuelles

Besoins des nouvelles applications

Opt = Option

Figure 3. *Besoins des applications carte*

Dans la figure 3, nous avons tout d'abord étudié les cartes existantes (C.Q.L., JavaCard, ou la carte utilisée actuellement pour les applications les plus courantes). Comme nous l'avons vu dans la section I, la carte CQL est la seule actuellement, à implémenter un mécanisme d'annulation de transaction, et de contrôle de concurrence sur ses données. Par contre, à l'inverse d'une carte comme la JavaCard, elle ne permet pas le chargement dynamique de code. Il faut aussi remarquer qu'aucune des 3 cartes décrites ici, ne gère le multi-contexte, ou un mécanisme de coordination de validation (2PC).

Après l'étude des cartes, nous avons repris les applications décrites dans la section III, pour examiner leurs besoins. Prenons le cas de l'application monétaire décrite dans la section II.1, plusieurs applications s'exécutent simultanément (l'application bancaire et l'application P.M.E.). Cela nécessite la gestion de plusieurs contextes dans la carte. De plus, ces applications agissent sur des données communes, ce qui implique une gestion des accès concurrents. Enfin, ces 2 applications doivent décider de valider ou d'abandonner simultanément leurs travaux, ce qui implique un protocole de validation. En ce qui concerne la première colonne du tableau (chargement de code dynamique), pour chacune des applications, cela est optionnel. En effet, cela n'influence pas l'exécution de l'application. Le seul intérêt est pour le prestataire qui commercialise la carte : il peut utiliser un même type de carte pour plusieurs applications.

Il faut noter que l'utilisation de plusieurs services est obligatoire dans le cas d'une coordination de la validation de l'application distribuée. Donc, si on prend l'application de réservation de billet, cette application n'est pas multi-contexte. Cependant, un contexte supplémentaire est nécessaire pour exécuter le gestionnaire de transaction.

5. NOUVEAU SYSTEME D'EXPLOITATION CARTE ET INTEGRATION DANS CORBA

Nous proposons dans cette section, à la fois, une nouvelle architecture du système d'exploitation carte (incluant la gestion de transactions), et une intégration de ces nouvelles cartes dans CORBA. Cette nouvelle architecture de Système d'Exploitation intègre la gestion des transactions, des verrous et des journaux, afin de garantir les propriétés ACID. L'intégration de ces cartes dans CORBA se fait via des composants d'adaptation, comme pour la carte OSMOSE (figure 2).

5.1. Support Transactionnel de la carte

Jusqu'à maintenant, nous avons étudié les contraintes technologiques de la carte. Mais comme nous l'avons vu précédemment, on peut être obligé d'implémenter dans la carte un gestionnaire de transaction.

5.1.1. Les systèmes d'exploitation carte actuels

Jusqu'à maintenant, les cartes étaient mono-applicatives (il n'y avait pas de distinction entre Système d'Exploitation et application). De nouvelles générations de cartes, multi-applicatives, disposent d'un Système d'Exploitation, qui les rend de plus en plus proche des architectures logicielles classiques.

Un Système d'exploitation Carte est composé de 3 entités distinctes (Figure 4).

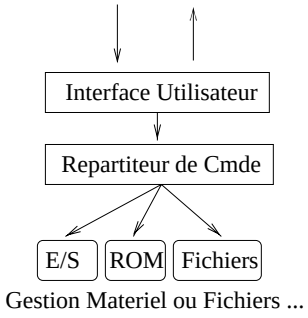

Figure 4. *Composants d'un Système d'exploitation Carte*

- L'interface Utilisateur permet de rentrer des ordres à la carte. Ces ordres sont normalisés (c'est ce que nous avons appelé "ordre APDU" précédemment), de manière à ce qu'une carte puisse se connecter sur n'importe quel terminal, dans le monde entier. Les ordres entrants sont transmis au dispatcher de commandes,
- Le répartiteur de commandes peut être vu comme un interpréteur de commande. Il reçoit les ordres de l'interface utilisateur, et lance les services correspondants, ou accède aux ressources matérielles de la carte,
- Les gestionnaires du matériel gèrent la mémoire, les entrées/sorties et tous les autres composants de la carte (permet les lectures et écritures en EEPROM, par le biais d'un code APDU normalisé).

Dans les nouvelles cartes multi-applicatives, telles que celles définies dans le projet OSMOSE, le dispatcher de commandes est un interpréteur de "bytecode" Forth **[Biget 96]**.

5.1.2. Le système transactionnel pour carte

A partir des remarques formulées dans la partie concernant les besoins transactionnels, nous avons spécifié un Système d'Exploitation intégrant un support d'exécution à des transactions et un gestionnaire de transactions, pour coordonner celles-ci, et entre des transactions externes à la carte.

Un service transactionnel doit par principe toujours garantir les propriétés ACID **[Haerder 83]**. Cependant, il est possible que le service transactionnel carte ne garantisse qu'une partie des propriétés des transactions : par exemple, dans le cas d'un PME (Porte Monnaie Electronique), les données ne sont présentes que dans la carte et sont perdues en cas de destruction de cette carte puisqu'il n'y a pas de réplication (la propriété de Durabilité n'est donc pas garantie).

Un service non transactionnel effectue des opérations sur lesquelles il ne pourra revenir (impossible d'utiliser le ROLLBACK).

Une architecture nouvelle

Une des particularités d'une application distribuée, dans laquelle intervient une carte à microprocesseur, est que l'application ou le service exécuté par la carte, peut exiger que le contrôle de la transaction distribuée soit réalisé par un site de confiance.

Ce site de confiance peut être soit la carte en question, soit une des machines reconnues par la carte. Par exemple, dans le cas d'une carte bancaire, le site de confiance peut être la carte elle-même ou bien le serveur de la banque qui a émis cette carte, et qui n'est pas toujours contacté lors d'une application (car pour des faibles montants, le coût de la vérification serait trop élevé).

Il y a toujours risque de tromperie, et donc de fraude si le site de contrôle n'effectue pas correctement la validation ou l'abandon d'une transaction distribuée. Pour éviter toute fraude, la carte peut donc être amenée à héberger le gestionnaire de transaction (qui est notamment chargé de la validation de la transaction distribuée).

La carte doit donc offrir un support pour gérer les transactions, mais aussi pour permettre le partage des données. A partir de ces observations, nous avons complété l'architecture carte existante, avec des éléments prennant en charge la validation (ou non) d'une transaction, la reprise sur panne, et le contrôle de concurrence (Figure 5).

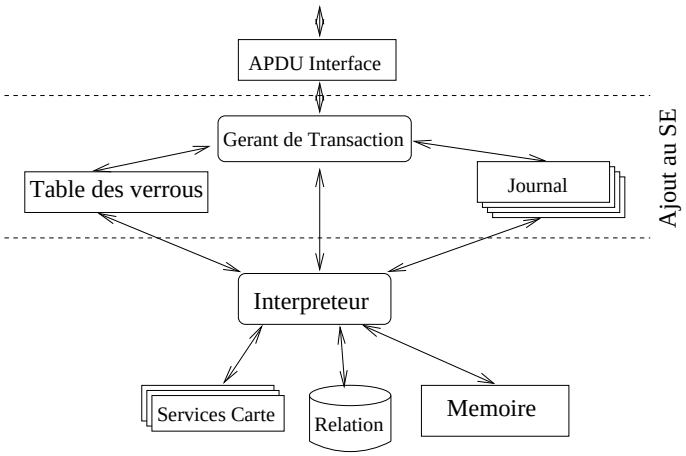

Figure 5. *Architecture OS Transactionnel*

Cette nouvelle architecture système comprend donc en plus du Système d'Exploitation classique définit précédemment, les éléments suivants :

- Le Gestionnaire de Transaction. Comme toute transaction, les transactions distribuées dans une carte à microprocesseur peuvent être annulées ou validées **[Traverson 91]**. Cet élément de l'architecture est notamment chargé de la validation de la transaction à l'intérieur de la carte (si la carte est esclave), et pour tous les partenaires (si la carte est maître). En plus de cela, il doit gérer la table des verrous et des journaux. Pour cela, nous avons décidé de placer le gérant de transaction entre l'interface utilisateur et l'interpréteur.

- Les Journaux des images avant. Il existe plusieurs types journaux **[Haerder 83]**. Dans la carte à microprocesseur, chaque écriture prend énormément de temps, car elle se fait en EEPROM (une écriture de 4 octets prend environ 100ms). De manière à ne pas perdre de temps lorsque la transaction se déroule correctement, nous modifions directement les valeurs des données sur le support stable. De manière à pouvoir adandonner une transaction, il faut conserver les anciennes valeurs des données. Nous utilisons donc un journal des images avant. Nous avons décidé de créer un Journal par transaction. Ainsi, il est beaucoup plus facile de gérer l'annulation.

- La Table des Verrous : l'intégration des cartes dans un environnement distribué rend possible les accès simultanés sur les données. Pour prendre en compte cela, une table des verrous permet de mémoriser les accès en cours sur une donnée. Il y a deux types de verrous : les verrous en écriture et ceux en lecture. Nous sommes donc obligés de maintenir à jour une table des verrous des diverses transactions. Il existe 2 modes de pose des verrous : le mode implicite et le mode explicite. Dans le premier cas, le programmeur n'a pas à utiliser les instructions de programmation Lock et Unlock. Ce qui évite les erreurs de programmation. Dans le second cas, à chaque accès à une donnée, le programmeur doit explicitement la verrouiller, puis explicitement la libérer. L'interprétation du code dans la carte, permet la pose de verrous implicites sur les données partagées en transactionnel.

5.2. La carte transactionnelle dans un contexte CORBA

Dans le cadre de ce papier, nous nous placerons dans l'environnement applicatif CORBA, et plus particulièrement son service de Transaction OTS (Object Transaction Service) **[OMG 96]**.

Cependant, ces concepts que nous utilisons ici, peuvent s'appliquer dans d'autres environnements, comme :

- L'environnement DCOM (Distributed Component Object Model), l'ORB de Microsoft, muni de MTS (Microsoft Transaction Service), le serveur de transaction de Microsoft **[Bernstein 97]**,

- ou avec les RMI (Remote Method Invocation) [6] **[Sun 97]** dans le cas d'utilisation d'objets Java [7] .

L'environnement CORBA a été choisi, car il est de plus en plus utilisé dans les communications client/seveurs, et de nombreux outils sont maintenants disponibles. Il est à noter aussi que les nouveaux systèmes d'exploitation carte, ainsi que les nouvelles applications sont développés avec une méthodologie orientée objet. De plus, le travail d'intégration de la carte avait déja été mené au sein du laboratoire. Cela facilite donc le développement d'une maquette utilisant les concepts définis dans cet article.

5.2.1. Les Transactions dans l'environnement CORBA

CORBA **[OMG 97]** est un environnement de communication permettant l'interopérabilité entre des composants développés dans des langages différents par des fournisseurs différents. L'architecture de

[6] Les RMI ne supportent pas les transactions dans leur définition actuelle
[7] Il existe JavaIDL, qui permet d'écrire des applications et des services CORBA en Java

CORBA repose sur un bus de communication ORB (Object Request Broker) par lequel une application invoque des requêtes (méthodes) auprès de services (objets) selon le principe du modèle client-serveur. L'ORB réalise la correspondance entre un service et le ou les serveurs qui le réalisent. Les services disponibles sur l'ORB sont des services communs nécessaires à la gestion de l'ORB et des services applicatifs constituant l'application client-serveur.

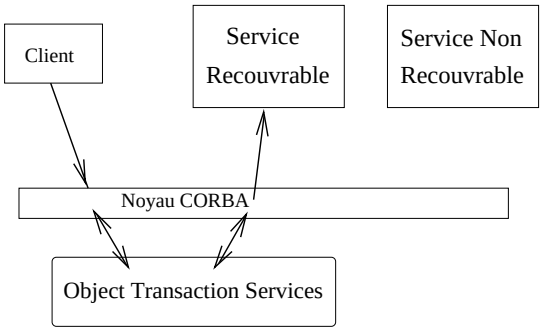

Figure 6. *OTS : Object Transaction Service*

Le Service Transaction OTS est un service commun de CORBA qui permet à une application d'invoquer des requêtes à des services dans le contexte d'une transaction (figure 6). Pour définir une transaction, l'application demande le contexte d'une nouvelle transaction à l'OTS, qu'elle communiquera (implicitement ou explicitement) à chaque invocation de requêtes auprès de services applicatifs. De leur côté, chaque service applicatif qui reçoit un contexte de transaction associé à une requête, s'enregistre la première fois auprès d'OTS pour signaler qu'il participe à la transaction et utilise ce contexte pour les requêtes suivantes de l'application. A la terminaison de l'application (i.e. la transaction distribuée), le service OTS coordonne la validation ou l'abandon en dialoguant avec l'application et les serveurs applicatifs qui se sont enregistrés. Un service applicatif qui accepte des contextes de transactions doit implanter l'interface de type "service recouvrable" (recoverable) garantissant les propriétés ACID des transactions au sein même du service avec des mécanismes de reprise sur panne et de contrôle de concurrence qui sont en dehors de la spécification de l'OTS.

Le service OTS peut être implanté par des composants centralisés ou distribués coopérants. OTS peut également coopérer (en tant que coordinateur ou comme participant) avec d'autres OTS sur d'autres ORB **[OMG 96]**. Le service OTS peut également importer ou exporter des transactions vers d'autres moniteurs transactionnels très largement diffusés comme Tuxedo ou Encina au moyen du modèle DTP définit par l'X/Open **[XOpen 93]**.

5.2.2. La carte et OTS

Les travaux d'intégration de la carte dans un environnement CORBA, et la description des caractéristiques de OTS, nous ont amené à vouloir utiliser OTS pour gérer le contexte des transactions, ainsi que la validation, dans le cas de la carte.

L'architecture la plus simple, permettant à 2 O.T.S. de collaborer, est celle mettant en jeu 2 ORBs. Cette architecture, appliquée à la carte est celle décrite dans la figure 7. Mais, de part les caractéristiques techniques, et de communication de la carte, une telle architecture est inconcevable.

Nous proposons donc une architecture issue de la première, où la carte est intégrée sur un ORB via un objet d'adaptation. L'OTS de la carte est particulier, car il doit être capable de communiquer à la fois

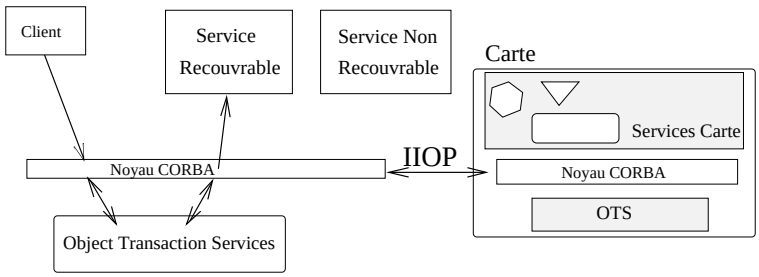

Figure 7. *OTS intégrant une carte*

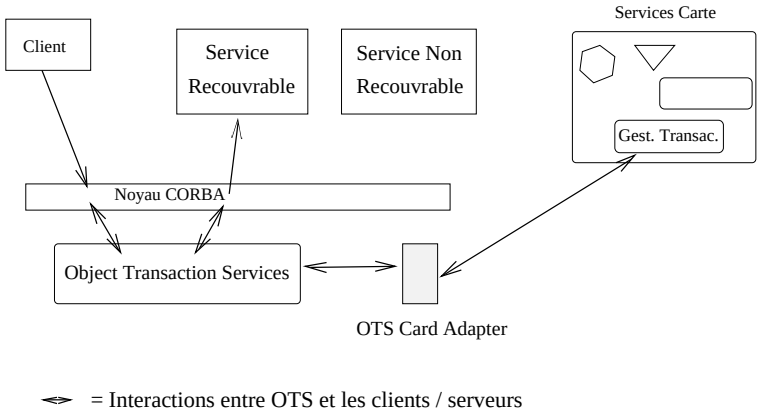

Figure 8. *OTS intégrant une carte*

avec la carte et avec d'autres OTS. Il doit donc reprendre une partie des caractéristiques du COA défini dans le projet OSMOSE, et offrir une partie des services de OTS. L'architecture utilisée est celle décrite dans la figure 8. Lorsque la carte est serveur d'une application distribuée classique, l'application démarre comme décrit dans le paragraphe précédent. Mais lorsque le service applicatif doit s'enregistrer auprès de OTS, il est obligé de passer par un objet d'adaptation (le OTS Card Adapter), qui doit remplir le rôle de passerelle entre la carte, dialoguant suivant le protocole ISO 7816-4, et l'environnement CORBA. L'objet d'adaptation est directement relié au Coordinateur intégré dans la carte. Une carte qui est serveur d'une application distribuée, doit disposer de services recouvrables.

6. CONCLUSION

Les cartes à microprocesseur se rencontrent maintenant dans les relations de tous les jours entre un porteur et un commerçant, un médecin, un opérateur téléphonique, etc. Parallèlement, les caractéristiques des cartes évoluent vers l'offre d'un bouquet de services au sein d'une seule carte pour le même domaine d'utilisation (carte monétaire avec des applications de débit du compte bancaire, de porte-monnaie électronique, etc.). Cependant, ces cartes multi-services ne proposent l'utilisation que d'un seul service à la fois.

Dans cet article nous proposons d'introduire dans les systèmes d'exploitation carte des mécanismes :

- le multi-contexte permettant à plusieurs applications de s'exécuter simultanément au cours de la connexion de la carte au terminal.
- le modèle transactionnel dans la carte étend le multi-contexte, en offrant en plus, un modèle simple et robuste de coopération et de cohérence entre applications simultanément dans un environnement fortement sujet aux pannes (carte arrachée du lecteur, etc.).

D'autre part, les cartes à microprocesseur deviennent de plus en plus un composant indispensable dans les applications distribuées sécurisées. Ces applications impliquent des participants qui peuvent potentiellement nier des opérations qu'ils avaient pourtant validées entre eux. Dans ce contexte, la carte transactionnelle que nous avons présentée offre un schéma souple de sécurisation des transactions engageant plusieurs partenaires.

Ces nouvelles fonctionnalités cartes (multi-contexte et transactionnelle) doivent s'intégrer dans des environnements de développement standards. Cet article propose l'intégration d'une ou plusieurs cartes dans des applications distribuées CORBA. Cette intégration utilise des composants (objets) d'adaptation qui font passerelle entre les transactions actives dans la carte et les services applicatifs ou communs enregistrés sur l'ORB. Un de ces services communs est plus particulièrement le Service de Transaction OTS qui coordonne la validation d'une transaction distribuée entre la ou les cartes et les autres machines serveurs engagées dans l'application distribuée.

Dans cet article, l'intégration de la carte dans CORBA et OTS passe par des composants d'adaptation de l'ORB. Bien que ces composants soient génériques, il y a une rupture dans les fonctionnalités attendues d'un composant CORBA, comme par exemple l'enregistrement des services cartes auprès de l'ORB auquel elle est connectée. Une des perspectives dans ce domaine seroa peut être d'offrir un ORB complet dans la carte elle-même quand les capacités de celle-ci auront évolué.

7. BIBLIOGRAPHIE

[Bernstein 87] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987

[Bernstein 97] P.A. Bernstein, E. Newcomer, *Principles of Transaction Processing for the systems professional*, M. Kaufmann Publishers, 1997.

[Biget 96] P. Biget, P. George, J.J. Vandewalle, *How Smart Cards Can Take Benefits from Object-Oriented Technologies*, Hartel, Paradinas, Quisquater eds, Septembre 1996.

[Biliris 94] A. Biliris, S. Dar, N. Gehani, H. V. Jagadish, K. Ramamritham, *ASSET: A System for Supporting Extended Transactions*, ACM SIGMOD, 1994.

[Chrysanthis 91] P. Chrysanthis, K. Ramamritham, *Synthesis of Extended Transaction Models using ACTA*, VLDB 1991.

[Elmagarmid 91] A. K. Elmagarmid, *Database Transaction Models for Advanced Applications*, M. Kaufmann Publishers, 1992.

[EMV 96] MaterCard et Visa, *Secure Electronic Transaction, Book 3: Technical Specifications*, Juin 1996.

[Gemplus 93] Gemplus, *CQL-Card Reference Manual*, 1993.

[Gordons 92] E. Gordons, G. Grimonprez, *A Card as element of a distributed database*, IFIP WG 8.4 Workshop, OTTAWA, 1992.

[Guthery 97] S. B. Guthery *Java Card: Internet Computing on a Smart Card*, IEEE Internet Computing Vol 1 No 1, Jan-Fev 1997

[Haerder 83] T. Haerder, A. Reuter, *Principles of Transaction-Oriented Database Recovery*, ACM Computing Surveys, Vol. 15, 1983

[ISO 7816] ISO International Organization for Standardization, P.O. Box 56, 1211 Geneva 20, Switzerland. International Standard ISO/IEC 7816, 1993

[ISO 9075] ISO/IEC 9075, *Information Technology - Database Languages- SQL*, ISO, 1992.

[ISO 10026] ISO/IEC 10026, *Distributed Transaction Processing*, ISO, 1996.

[Moss 85] J. E. B. Moss, *Nested Transaction : an Approach to Reliable Distributed Computing*, MIT Press series in Information systems, 1985.

[Noclercq 97] B. Noclercq, J.M. Place, *A Advanced Card Operating System on the Cascade Platform*, EMMSEC'97 Conference, Novembre 1997.

[OMG 96] Object Management Group, *Object Transaction Services*, 1996

[OMG 97] Object Management Group, *The Common Object Request Broker : Architecture and specification version 2.1*, Septembre 1997.

[Ozsu 91] M. Tamer Ozsu, P. Valduriez, *Principles of Distributed Databases System*, Prentice Hall Intl., ISBN 0-13-715681-2, 1991.

[Schneier 96] B. Schneier, *Applied Cryptography, 2nd edition*, John Wiley & Sons, 1996.

[Sesam Vitale 96] G.I.E. Sesam Vitale, *Cahier des charges SESAM-Vitale 1996*, Version 1.0, Décembre 1996.

[Skeen 81] D. Skeen, *NonBlocking Commit Protocols*, ACM Sigmod, ACM Press, 1981.

[Sun 97] Sun Microsystems *Remote Method Invocation Specification*, Rev 1.4, JDK 1.1, Fevrier 1997.

[Traverson 91] B. Traverson, *Stratégies d'optimisation et évaluation de performance du protocole de Validation en deux phases*, Thèse à l'université de Paris VI, Septembre 1991.

[Van Hoecke 96] M.P. Van Hoecke, *Contribution à la modélisation des Systèmes d'Information Communicationnels intégrant des cartes à microprocesseur*, Thèse à l'université de Lille 1, Janvier 1996.

[Vandewalle 97] J.J. Vandewalle, *OSMOSE : Modélisation et Implémentation pour l'interopérabilité de services carte à microprocesseur par l'approche orientée objet*, Thèse à l'université de Lille 1, Mars 1997.

[XOpen 93] X/OPEN Guide, *Distributed Transaction Processing Reference Model Version 2*, Octobre 1993.