gen 2-2176

50376
1998
289

Thèse de Doctorat

Présentée à
L'Université des Sciences et Technologies de Lille

Pour l'obtention du titre de
Docteur en Informatique
par

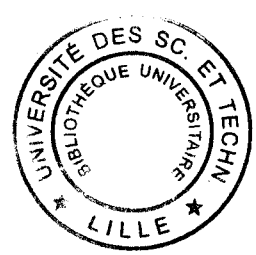
Sylvain Lecomte

COST-STIC
Cartes Orientées Services Transactionnels et Systèmes Transactionnels Intégrant des Cartes

Soutenue le Jeudi 26 novembre 1998 devant le jury :

Président :	Jean-Marc Geib, Prof. LIFL
Rapporteurs :	Jean Ferrié, Prof. ISIM
	Michel Riveill, ENSIMAG
Examinateurs :	Vincent Cordonnier, Prof. LIFL
	Didier Donsez, MdC LIMAV
	Pierre Paradinas, Gemplus
	Philippe Pucheral, MdC, Université de Versailles
	Simone Sédillot, INRIA Rocquencourt

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât. M3 - 59655 VILLENEUVE D'ASCQ CEDEX
Tél. (+33) 3 20 43 47 24 - Télécopie (+33) 3 20 43 65 66



Remerciements

Les travaux présentés dans ce mémoire ont été réalisés au sein de l'équipe de Recherche et Développement sur le Dossier Portable (*RD2P*), du Laboratoire d'Informatique Fondamentale de Lille (*LIFL*). C'est pourquoi, je tiens avant tout à présenter mes remerciements au Professeur Vincent Cordonnier, qui dirige cette équipe, pour ses précieux conseils et la relecture de ce document.

Je tiens aussi à remercier vivement Pierre Paradinas, responsable de l'équipe de recherche avancée de Gemplus, pour son aide, ses encouragements, et sa confiance, tout au long de ces années.

Je remercie aussi le Centre Nationale de la Recherche Scientifique (*CNRS*), et la société Gemplus qui ont bien voulu m'apporter leur appui financier par l'octroi d'une bourse doctorale ingénieur.

J'exprime à présent ma gratitude aux membres du jury:

- A son président, le professeur Jean Marc Geib de l'université de Lille 1,

- Aux rapporteurs, qui ont bien voulu examiner mon travail, le professeur Jean Ferrié de l'université de Montpellier, et le professeur Michel Riveil de l'ENSIMAG,

- Aux examinateurs, le professeur Vincent Cordonnier, de l'université de Lille 1, Didier Donsez de l'université de Valenciennes, Pierre Paradinas de la société Gemplus, Philippe Pucheral de l'université de Versailles, et Simone Sédillot de l'INRIA.

J'ai eu la chance de travailler dans un groupe convivial, énergique et passionné. J'aimerais en remercier David Carlier et Patrick Trane pour les travaux effectués ensemble, ainsi que Didier Donsez et Gilles Grimaud, pour leurs conseils multiples et précieux, ainsi que leur patience et leur rigueur dans la mise au point de ce document.

De la même manière, je tiens à remercier Isabelle Cuignies et Sergiy Nemchenko pour l'aide qu'ils m'ont apportée au cours de leurs stages. Je souhaite aussi chaleureusement féliciter et remercier Lamia Aouni et Jean Jacques Vandewalle, qui ont activement participé à la réalisation du prototype carte.

Il me tient à coeur de présenter ma gratitude aux anciens et actuels membres de l'équipe RD2P, et de l'équipe de recherche de Gemplus, pour leur compétence et leur amitié, et je souhaite bon courage aux nouveaux arrivants.

Ces remerciements ne seraient pas complets, si je n'associais pas aussi à ce travail, mes amis proches, mes parents qui me soutiennent depuis toujours, ainsi que Valérie, qui m'a toujours encouragé, et qui a supporté mes humeurs durant ces 3 années.

Table des matières

Liste des tableaux

Table des figures

I. Etat de l'Art

II. Problématique

Chapitre .1

Introduction

1.1 Les cartes à microprocesseur

La carte à microprocesseur est un composant informatique portable, qui connait un essor important depuis dix ans et qui, de par ses caractéristiques économique et informatique, devrait continuer à évoluer dans les années à venir [**C92**].

Cependant, les cartes à microprocesseur exécutent des services dans un contexte très sujet aux pannes et aux interruptions de service : la carte est très souvent déconnectée, et elle peut être retirée du terminal dans lequel on la connecte à tout moment.

Le premier but de ce mémoire est donc de rendre les cartes à microprocesseur, ainsi que leurs partenaires d'exécution, tolérantes aux pannes. Pour cela, nous étudierons la problématique des cartes à microprocesseur, ainsi que les différents modèles de panne rencontrés.

1.2 Cartes et Systèmes Distribués

Après l'étude des applications simples pour carte à microprocesseur et des problèmes que ces applications posent, nous positionnerons la carte dans les systèmes distribués. Le cadre général de ce projet de thèse sera alors celui des grands systèmes distribués, et des problèmes de sécurité que pose le maniement de données confidentielles ou sensibles sur ces systèmes. Un exemple des nouveaux problèmes de manipulation des données dans les systèmes distribués est le développement de

l'Internet commercial. Les solutions envisagées pour sécuriser les mécanismes de paiement sur Internet peuvent mettre en jeu des cartes à microprocesseur.

Simultanément, les cartes à microprocesseur ont évolué et sont maintenant capables de traitements évolués et sécurisés, tout en devenant plus facilement accessibles aux programmeurs traditionnels. De même, les terminaux pour carte se sont répandus. Le plus courant est le téléphone mobile GSM[1]. L'ensemble de ces avancées rendent la carte plus présente dans les utilisations courantes, et plus intégrable au systèmes informatiques actuels [**C96**].

La carte se présente donc comme un bon moyen pour sécuriser les accès et les transactions financières sur Internet, ou dans tout autre environnement nomade. Cela implique l'intégration des cartes dans des applications de plus en plus distribuées et sujettes aux pannes (et donc à la fraude par simulation, ou provocation volontaire, de pannes). Les systèmes d'exploitation des cartes à microprocesseur, et leur interface ne sont pas actuellement compatibles avec une telle intégration. Il faut donc que les cartes soient capables de gérer efficacement leurs ressources, les reprises sur panne, et la validation des opérations dans un contexte distribué. Elles ne doivent pas pour autant perdre la facilité de programmation qu'elles viennent tout juste de gagner, par l'adoption de langage de programmation commun avec les systèmes traditionnels, comme Java [**GUT97**].

Cependant, il est impossible de modifier l'architecture de base des systèmes distribués existants, ainsi que les normes ou standards de fait qui les régissent. Ceci nous oblige à réfléchir à des composants d'adaptation pour les cartes. En effet, les cartes à microprocesseur sont des composants informatiques très spécifiques, tant par leur déconnexion fréquente, que par leur protocole de communication, ou leur capacité.

1.3 Orientation du projet

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 [**SES96**]). Actuellement la carte à microprocesseur prend son essor grâce au développement de l'Internet commercial, comme solution de sécurisation des paiements et des accès [**SET96**].

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, requérant 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[2] de l'OMG[3][**OMG97**]. Cette étude nous conduira

1. Global System for Mobile communication
2. CORBA : Common Object Request Broker Architecture
3. OMG : Object Management Group

à décrire une nouvelle génération de système d'exploitation, contenant des outils dédiés à la gestion des transactions.

Ainsi, les principales directions suivies dans ce mémoire sont les suivantes :

- La définition d'un système d'exploitation transactionnel pour les cartes à microprocesseur : le modèle transactionnel, de par ses propriétés, offre un bon modèle de cohérence pour les nouvelles applications carte, et retire au programmeur tout souci de tolérance aux pannes, par la définition d'un modèle de reprise sur panne **[Gem94]**. L'intégration du modèle transactionnel dans les cartes à microprocesseur nous permettra de spécifier une Carte Offrant des Services Transactionnels (COST), qui pourra être utilisée tant en mode point-à-point (liaison simple Carte / Terminal, pour les applications les plus simples), que dans les systèmes distribués (comme Internet).

- La définition d'un composant d'adaptation pour systèmes distribués : les cartes sont spécifiques, de part leurs caractéristiques techniques, mais aussi de part leur protocole de communication. Or, l'intégration de ces cartes devra se faire sans modification des systèmes existants. Les Systèmes Transactionnels Intégrants des Cartes à microprocesseur (STIC) devront donc utiliser un composant d'adaptation (un traducteur de requête) pour communiquer avec les cartes à microprocesseur.

1.4 Plan de ce mémoire

La suite de ce mémoire de thèse s'organise en 5 grandes parties, chacune étant subdivisée en chapitres :

- La première partie nous donne un état de l'art, à la fois des cartes à microprocesseur (premier chapitre), mais aussi de la solution que nous envisageons d'utiliser pour répondre à notre problématique : le modèle transactionnel (deuxième chapitre).

- La seconde partie présente dans un premier temps les problèmes et les besoins des nouvelles applications mettant en jeu des cartes à microprocesseur (premier chapitre). Puis, comment ces problèmes se répercutent sur le modèle d'exécution des cartes à microprocesseur (deuxième chapitre).

- La troisième partie propose la définition d'un système d'exploitation pour cartes à microprocesseur, basé sur le modèle transactionnel, qui répond aux problèmes posés dans la seconde partie (premier chapitre). Ensuite, dans cette partie, nous étudierons l'intégration de cette nouvelle carte dans les systèmes distribués existants (deuxième chapitre). Enfin, nous décrierons la maquette réalisée sur les bases de ces spécifications (troisième chapitre).

- La quatrième partie résume les travaux menés et conclut sur les perspectives de ce projet.

- La dernière partie contient les annexes de ce document, ainsi que les références bibliographiques.

Première partie

Etat de l'Art

Chapitre I.1

Les cartes à microprocesseur : Principes et Evolutions

1.1 Introduction

Depuis la création des cartes à microprocesseur en 1974, suivant les concepts énoncés par Roland Moréno, les technologies, tant logicielles que matérielles, ont beaucoup évoluées. Si bien que tout un chacun dispose maintenant d'une ou plusieurs cartes dans son portefeuille (on peut notamment citer les cartes bancaires, les cartes pré-payées de téléphonie, ou les cartes de santé).

Ces cartes n'utilisent pas toutes la même technologie : il existe plusieurs types de cartes, des plus simples aux plus performantes. On trouve notamment :

- Les cartes à mémoire : ce sont les cartes les plus simples. Il s'agit en fait de « porte-jetons ». Ces cartes ne sont pas capables de traitement. L'exemple le plus répandu est la carte téléphonique de France Télécom, qui contient des unités téléphoniques.

- Les cartes à logique câblée : ces cartes sont une évolution des précédentes. Elles ont comme principale caractéristique d'être rechargeables. Ce sont des cartes contenant de la mémoire non volatile (de type EEPROM[1]). Tout comme le premier modèle, ces cartes ne sont pas capables d'effectuer des traitements.

1. EEPROM : Electrically Erasable PROgrammable Memory

- Les cartes à microprocesseur : ces cartes contiennent un microprocesseur, et sont donc capables de traitements sur les données qu'elles contiennent. Les applications les plus connues de ces cartes sont la carte bancaire, ou la carte GSM.

- Les cartes sans contact : ces cartes sont une variante des cartes à microprocesseur, ou des cartes à logique câblée. Leur principale caractéristique est de ne pas nécessiter de contact physique avec un lecteur pour communiquer avec l'extérieur. Ces cartes sont de plus en plus utilisées soit pour marquer des objets (étiquette électronique contre le vol par exemple) dans le cas d'une carte sans contact à logique câblée, ou pour des traitements nécessitants une avancée rapide du porteur (paiement du Métro), dans le cas d'une carte à microprocesseur.

Dans ce mémoire, nous nous intéresserons principalement aux cartes à microprocesseur, en étudiant dans un premier temps leurs contraintes technologiques, puis leur moyen de communication. Nous terminerons cette étude par les caractéristiques logicielles de ces cartes, et par une étude des dernières cartes apparues sur le marché.

1.2 Caractéristiques Matérielles

Malgré leur constante évolution, les cartes à microprocesseur sont très en retrait des systèmes informatiques actuels. Le but de ce paragraphe traitant des caractéristiques matérielles des cartes à microprocesseur est de fixer les ordres de grandeur des possibilités des cartes. Il faut cependant garder à l'esprit que la technologie avance très rapidement dans ce domaine, et que les performances annoncées ici (en terme de taille de mémoire, ou de puissance de calcul) ne sont données qu'à titre indicatif.

1.2.1 Architecture carte classique

Une carte à microprocesseur est constituée de 3 parties. La première est le support plastique. Ce support est normalisé, tant en taille qu'en résistance [**Iso87**]. Il est de peu d'intérêt pour notre étude. Ensuite, on trouve la pastille de contact, qui permet les échanges entre le monde extérieur et la carte à microprocesseur. La position de cette pastille de contact est, elle aussi, normalisée [**Iso88**]. Enfin, on trouve, sous la pastille de contact, le coeur de la carte à microprocesseur : un micromodule monolithique, qui est en fait un micro-ordinateur qui tient sur une surface de moins de 30 mm^2, et sur une épaisseur de moins de 0,28 mm. Ce micromodule est décrit plus en détail par la figure I.1.1.

Les microprocesseurs les plus répandus dans les cartes à microprocesseur sont actuellement les processeurs 8 bits de type Motorola 6805, ou équivalents (Intel 8051, SGS-Thomson 8048 ou Hitachi H8). L'avantage principal de ces processeurs est de prendre peu de place sur le micromodule. Cependant, des solutions qui visaient à intégrer dans les cartes des processeurs de type RISC[2] 32 bits [**C94**] [**Esp93**], sont maintenant commercialisées.

2. RISC : Reduced Instruction Set Computer

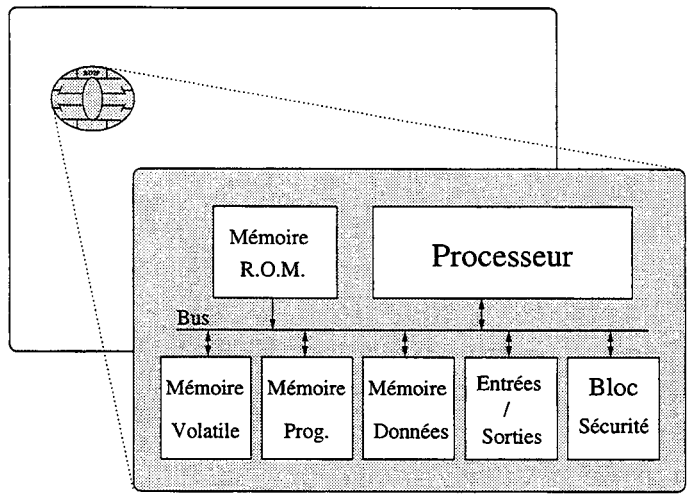


FIG. I.1.1 - *Composants d'un micromodule carte*

En ce qui concerne le port d'Entrées / Sorties, il est actuellement, dans la majorité des cartes, de 9600 BPS[3]. Il s'agit d'un port de communication série bidirectionnel, qui peut recevoir des commandes du monde extérieur, et envoyer les réponses en alternance.

La mémoire de travail (qui est la mémoire volatile) est de type RAM[4] statique, sa taille varie de 128 octets à 1 Ko[5]. Cette taille est fonction de la place disponible sur le module, et du coût du composant. Cependant, on peut espérer des tailles de mémoire de travail de l'ordre de 2 Ko dans l'année à venir, principalement grâce à l'utilisation de nouvelles mémoires non volatiles, prenant moins de place sur le module (cf. paragraphe 1.2.2).

La mémoire ROM[6], contient le système d'exploitation de la carte. Sa capacité varie, selon les besoins de 2 à 64 Ko.

La mémoire non volatile est actuellement en pleine évolution, et fera l'objet d'une étude séparée dans le paragraphe 1.2.2. D'une manière générale, cette mémoire contient tout ce qui est chargé dans la carte au cours de son utilisation (données et applications pour certaines cartes).

Le dernier composant présent dans tous les micromodules des cartes à microprocesseur est le bloc de sécurité. En effet, de manière à résister aux attaques physiques, les cartes à microprocesseur sont munies de détecteurs physiques (lumière, tension, fréquences, etc...) qui provoquent un blocage complet de la carte en cas de condition anormale de fonctionnement. Ce bloc de sécurité participe (avec les tests d'accès logique à la carte, et l'aspect monolithique[7] du microcontrôleur) à la sécurité inégalée des cartes à microprocesseur.

3. BPS : Bits Par Seconde
4. RAM : Random Access Memory
5. Ko : Kilo-Octets
6. ROM : Read Only Memory
7. Il est ainsi impossible d'espionner les données qui circulent sur le bus de communication

1.2.2 Les mémoires non volatiles

Le type et la quantité de mémoire non volatile disponible pour les données et les programmes de la carte, sont en pleine évolution. Ainsi, l'EEPROM, qui était utilisée dans la plupart des cartes, est progressivement remplacée par de la mémoire FLASH. Une autre technologie, La FeRAM pour RAM Ferro Electrique, est en train d'apparaître. Elle est en cours d'évaluation technique. Les avantages et les inconvénients de ces trois techniques sont explicités dans le tableau I.1.1

	EEPROM	**FLASH**	**FeRAM**
Tps Accès Lecture	150ns	150ns	100ns
Tps Prog Ecriture	5ms	10 μs	400ns
Tps Effacement	5ms	100ms	Sans
Granulatité Ecriture	1 à 4 Octets	64 / 128 Octets	Sans
Nb de cycle garanti	10^5 (Ecrit.)	10^5 (Ecrit.)	10^{10} (Lect./Ecrit.)
Taille point Mémoire	$> 30\mu m^2$	$< 10\mu m^2$	$< 10\mu m^2$

TAB. I.1.1 - *Comparatif EEPROM/FLASH/FeRAM*

Le principal défaut de la mémoire EEPROM est la taille du point mémoire. En effet, sur une même surface, on met trois fois plus de mémoire Flash, ou de FeRAM, que de EEPROM. A l'inverse, la principale qualité de la mémoire de type Flash est donc d'occuper moins de place que la mémoire EEPROM. Ce qui a permis de doubler la taille de la mémoire non volatile, et de donner plus de place pour étendre la RAM. Cependant, la mémoire Flash pose des problèmes de granularité d'accès en écriture. Ainsi, pour écrire 1 octet en mémoire Flash, il faut effacer, puis réécrire tout le banc mémoire contenant l'octet (à savoir 64 ou 128 octets suivant les types d'architectures).

Les nouvelles mémoires à base de composants Ferro-électriques marquent, elles aussi, une avancée non négligeable. En effet, en plus d'une taille de point mémoire sensiblement identique à celle de la mémoire Flash, ces mémoires offrent une granularité d'accès excellente (identique à l'EEPROM), et une rapidité d'écriture sensiblement équivalente à de la RAM classique (elle ne nécessite pas d'effacement avant l'écriture de nouveaux mots). Le principal défaut de ce type de mémoire, est que la lecture d'une zone provoque l'effacement de cette zone (ce qui impose une réécriture immédiate après la lecture). Cette technique est encore en phase de test, mais est un bon compromis pour augmenter la taille des mémoires cartes, avec moins de contraintes que les mémoires Flash. Cependant, cette mémoire ne peut pas être utilisée comme mémoire de programme, car l'exécution du code stocké dans cette mémoire entraîne son effacement.

1.3 Dialogue Carte / Terminal

Lors de sa phase d'utilisation, une carte à microprocesseur subit des cycles de trois étapes (cf. figure I.1.2) : l'insertion de la carte, l'exécution de commandes et la déconnexion. Ces trois étapes forment une *connexion.*

Lorsque la carte est insérée, elle se trouve de nouveau alimentée en énergie, et est réinitialisée. La seconde phase consiste en la réception d'une commande sur le port d'entrée/sortie, et à l'exécution de celle-ci. Cette phase peut être renouvelée autant de fois que nécessaire pour la bonne fin de l'application. La troisième phase correspond à une coupure franche de l'alimentation électrique de la carte. Elle se retrouve alors dans l'impossibilité de faire quoi que ce soit.

Seules les deux premières phases de la connexion permettent d'agir sur le contenu de la carte.

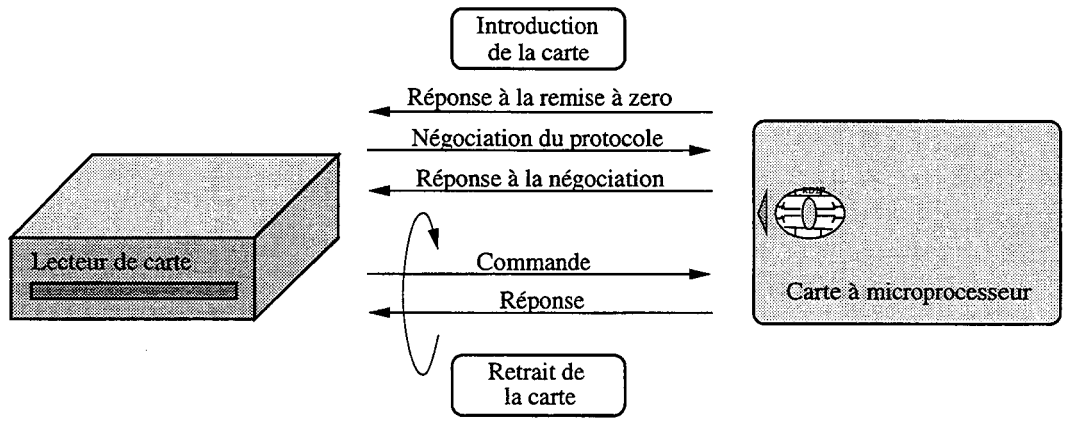


FIG. I.1.2 - *Protocole de communication Carte / Terminal*

1.3.1 La Connexion

La connexion d'une carte dans le terminal représente toutes les actions effectuées entre le moment ou la carte est insérée dans le terminal, et le moment où elle en est retirée.

Lorsque la carte est connectée dans un terminal, elle effectue une suite d'actions pour tester son état, puis elle émet une suite d'octets, appelée «Réponse à la remise à zéro» (aussi appelée *ATR*, pour Answer To Reset). Cette réponse est normalisée **[Iso95]**. L'ATR est composée de 33 octets maximum, qui sont répartis en cinq champs. La signification de tous ces octets est peu importante, et peut être obtenue dans la norme, mais seuls les deux premiers caractères sont obligatoires. Le premier spécifie les conventions de codage des octets de données dans tous les caractères ultérieurs, et le second indique le protocole de communication utilisé par la carte (cf. le paragraphe 1.3.2), ainsi que la taille de l'ATR.

La fin de la connexion carte / terminal est marquée par le retrait de la carte du lecteur (appelé *arrachement*). Dans un usage normal, le porteur est informé par l'interface du lecteur que la connexion est terminée et qu'il peut retirer sa carte. Il arrive également que la carte soit complètement «avalée» par le lecteur et restituée par ce dernier au terme de la connexion. Dans les deux cas, l'arrachement de la carte ne doit pas poser de problème. Cependant, un arrachement intempestif par erreur (distraction, impatience), ou volontaire (tentative de fraude) représente une

situation suffisamment fréquente pour être prise en compte en tant qu'évènement habituel.

1.3.2 Le Dialogue

La communication entre la carte et le terminal s'effectue selon des protocoles de communication normalisés [**Iso93**]. Comme nous l'avons vu précédemment, le choix de ce protocole se fait lors de l'ATR, et est donc défini par la carte à microprocesseur.

Les différents protocoles varient peu sur leur principe de fonctionnement, c'est pour cela que nous ne les distinguerons pas ici. Ces protocoles définissent les ordres que l'on peut envoyer à la carte (ordres entrants), ainsi que les ordres qui demandent des données de la carte (ordres sortants). Toutes les communications entre la carte et son lecteur se terminent par l'envoi d'un mot d'état, soit pour signaler que l'ordre a bien été exécuté (ordre entrant), soit pour dire que toutes les données ont été transmises (ordre sortant). La demande provient toujours du lecteur : la carte ne peut pas émettre de requête, elle peut juste répondre à un ordre du lecteur. Elle est donc considérée comme passive.

Le format des ordres est lui aussi normalisé par la norme [**Iso93**]. Ces commandes sont appelées commandes APDU[8].

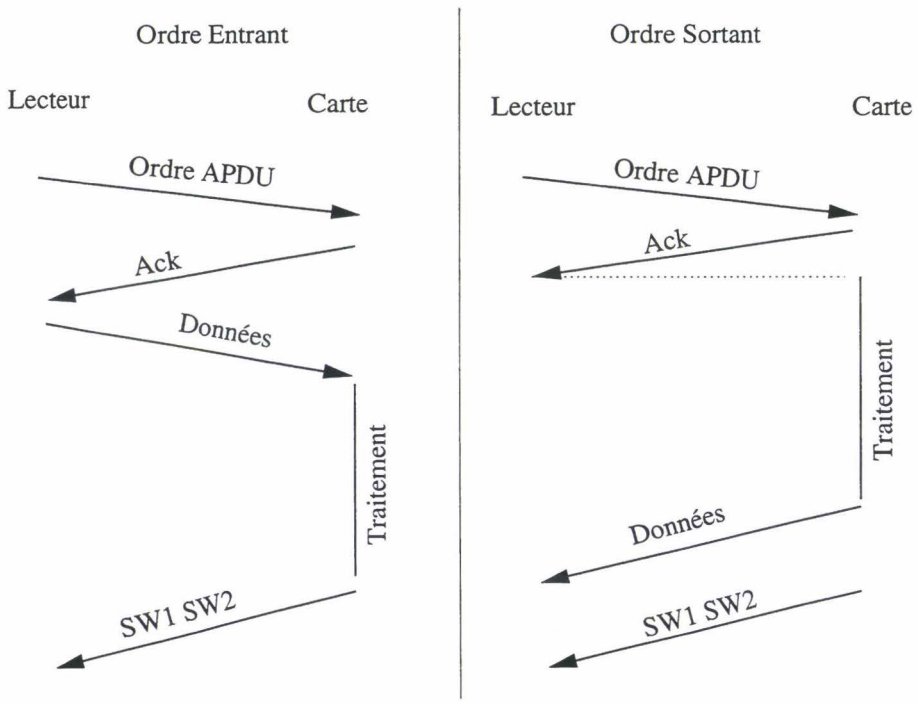


FIG. I.1.3 - *Echange d'ordres APDU*

Le fonctionnement des ordres entrants et sortants, détaillé dans la figure I.1.3,

8. APDU : Application Protocol Data Unit

est le suivant :

- Les ordres entrants : lors d'un ordre entrant, la carte commence par accuser réception de l'ordre reçu, puis elle se met en attente des données. Une fois les données reçues, la carte commence l'exécution de l'ordre. A la fin du travail, la carte envoie le mot d'état au terminal[9] (si tout s'est bien déroulé, ces mots valent 90 00).

- Les ordres sortants : lors d'un ordre sortant, la carte accuse réception de l'ordre reçu, puis elle commence aussitôt l'exécution de la méthode correspondante (l'exécution commence dès que la routine d 'E/S de communication est achevée). A la fin de cette exécution, la carte envoie le résultat de son travail au terminal. Après cela, la carte envoie ses mots d'état.

Ces protocoles sont donc relativement limités, et montrent que la carte à microprocesseur a été conçue comme un serveur de données. Cependant, ces protocoles de communication ne doivent être vus que comme la couche de transport des requêtes du terminal vers la carte. Si l'on se réfère à un réseau développé, les protocoles de communication carte / terminal sont au même niveau que le protocole TCP/IP[10] **[S96]** pour les communications sur le réseau Internet. Dans le reste de ce document, nous chercherons donc à nous abstraire au maximum des protocoles de communication.

1.3.3 La Session

La notion de session représente l'ensemble des ordres APDU nécessaire au traitement d'un service rendu par la carte. Jusqu'à il y a peu de temps, les cartes étaient monoservices. La session était donc synonyme de durée de connexion. Comme maintenant les nouvelles cartes peuvent rendre plusieurs services (cf. paragraphe 1.6) sur une même connexion, une connexion peut être constituée de plusieurs sessions, elles-mêmes constituées de plusieurs ordres APDU (cf. figure I.1.4).

Actuellement, les cartes, même les plus évoluées ne peuvent participer qu'à une seule session active à la fois.

1.4 Architecture Carte : Historique et Futur

Le but de cette partie est de bien situer l'état d'avancement des «logiciels» embarqués dans les cartes à microprocesseur.

Nous allons donc commencer par étudier les systèmes carte existants, et principalement leurs limites. Après cela, nous étudierons une carte un peu marginale, la carte CQL[11], qui est la première carte contenant une architecture logicielle semblable aux bases de données classiques. Nous clôturerons cette étude par la dernière génération de système carte, représentée par les cartes embarquant une Machine Virtuelle.

9. Les deux mots d'état sont notés SW1 SW2 pour Status Word
10. TCP/IP : Transmission Control Protocol / Internet Protocol
11. CQL : Card Query Language

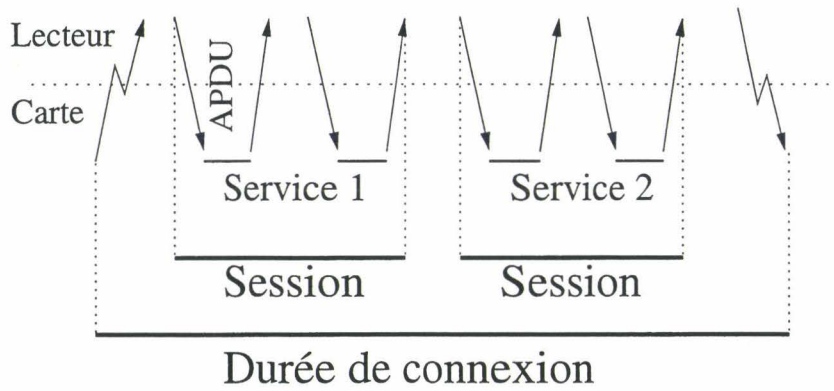


FIG. I.1.4 - *Sessions durant une connexion carte*

1.4.1 Systèmes Carte existants

Pour bien comprendre les systèmes carte, il est important d'étudier leur cycle de fabrication et d'utilisation (figure I.1.5), appelé cycle de vie de la carte.

Les cartes à microprocesseur actuelles sont dédiées à une seule application. En fait, système d'exploitation et application ne font qu'un et forment ce que l'on appelle le *masque*. L'inscription du masque dans la carte fait partie de la phase de fabrication, car ce masque est gravé en ROM. Ainsi, une fois le masque installé, la carte est opérationnelle, et a presque toutes ses fonctionnalités. Il ne reste plus à l'émetteur d'application qu'à *personnaliser* la carte. Cette phase permet de définir, et de charger, les données stockées dans la carte, ainsi que les droits d'utilisation qui leur sont appliqués. Une fois la personnalisation effectuée, la carte entre en phase d'utilisation. Il est alors impossible de modifier les programmes embarqués, et de la faire évoluer autrement que par la valeur des données qu'elle contient.

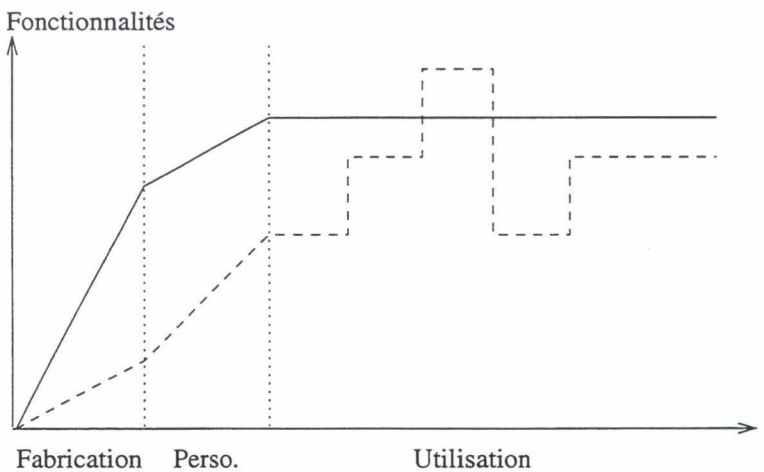


FIG. I.1.5 - *Cycle de vie des Cartes*

Une nouvelle approche consiste à définir des *cartes génériques*, qui disposent d'un

système d'exploitation «classique» [**P95**]. Dans ces cartes, le cycle de vie évolue différemment : à la fabrication, seul le système d'exploitation est chargé dans la ROM. Il permet juste un accès aux ressources matérielles de la carte. On peut donc considérer qu'à la fin de sa fabrication, la carte offre moins de fonctionnalités que les cartes classiques. En fait, à ce stade, elle est dépourvue d'application. Vient alors la phase de personnalisation, où les premières applications sont installées dans la carte. La distribution des rôles change donc : le fabricant prend un rôle moins important, alors que l'émetteur d'application renforce le sien.

Un autre point important dans l'approche carte générique, est la dynamicité des applications installées dans la carte. En effet, l'utilisateur de la carte va pouvoir charger et décharger les applications qu'il désire pendant la durée d'utilisation de la carte à microprocesseur, sans que cela ne pose de problème de sécurité (grâce, par exemple, à un mécanisme de type bac à sable[12] [**Gem96**]). Un exemple de carte générique est la CyberFlex de chez Schlumberger, qui a été commercialisée en 1996.

Ces cartes génériques sont le point de départ des travaux sur les cartes à microprocesseur multi-applicatives, dont nous reparlerons dans la partie II.1 de ce mémoire.

1.4.2 Une autre norme : La carte CQL

La carte CQL, qui est basée sur des travaux menés à RD2P [**GG92**] adopte les concepts de SGBD [**OV91**] et utilise des commandes SQL [**Iso92**]. Cette carte a été améliorée et industrialisée par la société Gemplus en 1993 [**Gem93**].

Cette carte peut être vue comme un serveur individuel portable de données, dont la première caractéristique est de contenir plusieurs tables SQL et un moteur de SGBD résidant 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*). Enfin, on dispose de tables systèmes, qui comprennent un dictionnaire, qui contient la structure de la carte.

De plus, la carte CQL est un serveur de données très sécurisé. En effet, en plus de la sécurité intrinsèque des cartes à microprocesseur, la carte CQL dispose d'un schéma de sécurité très évolué, avec 3 niveaux utilisateurs (l'émetteur, les gestionnaires d'applications, et les utilisateurs). Ainsi, un type utilisateur créant un objet, en est l'unique propriétaire et peut seul effectuer les traitements sur cet objet.

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 sur deux niveaux différents. On peut soit demander simplement un mot de passe, soit utiliser une double authentification, qui repose sur un algorithme de crytographie à clé secrète (par exemple, l'algorithme DES [**NBS77**]).

- 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

12. Le bac à sable est une image qui signifie que chaque application ne peut pas sortir de la zone mémoire lui est allouée

d'une transaction. Ceci a nécessité l'implémentation de trois ordres à l'intérieur de la carte (*BEGIN TRANSACTION, COMMIT, ROLLBACK*). De par la taille du buffer utilisé, la taille maximale de la transaction CQL est de cinq modifications (insert, erase ou update). Ces ordres implémentent un 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 document, 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 [**NP97**]. Le but de ces nouveaux travaux est de faire effectuer automatiquement des traitements à la carte, sans que le porteur ne le décide explicitement (par réaction à une donnée modifiée par exemple). Les exemples d'applications de cette nouvelle technique concernent notamment le Porte Monnaie Electronique (calcul automatique dans différentes devises), ou l'aide médicale (interactions médicamenteuses et contre indications).

1.4.3 Intégration des cartes dans les systèmes distribués

Les cartes à microprocesseur se rapprochent de plus en plus, en conception, des systèmes informatiques classiques. Il était donc logique de vouloir faire coopérer ces deux mondes.

Dans un premier temps, des travaux ont été menés à RD2P sur l'intégration des cartes à microprocesseur dans les Systèmes d'Information Communicationnels[13] [**VaH96**]. 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 sur 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 de celle d'un système «traditionnel».

L'étape suivante dans l'intégration des cartes à microprocesseur dans les systèmes distribués a été la construction d'outils et de méthodes qui assurent les services requis par les conclusions des premiers travaux. Cela a abouti à la mise à disposition des services offerts par la carte sur un bus de communication logiciel (projet OSMOSE[14], mené à RD2P en 1997 [**V97**]). Le bus de communication utilisé était CORBA, de l'OMG[15]. CORBA est un bus de communication entre objets répartis [**OMG97**]. Ces objets sont des applications ou des services accessibles suivant le modèle client/serveur. Les applications construites sur une architecture Corba devraient être facilement portables et interopérables [**GGM97**], car le standard de l'OMG est le fruit de discussions entre tous les participants du consortium.

13. système auquel est dévolue la prise en charge de la gestion des dialogues entre des grandes organisations et des particuliers

14. OSMOSE : Operating System for Mobile Object SErvices

15. OMG : Object Management Group, est un consortium regroupant plus de 800 fabriquants, utilisateurs, vendeurs

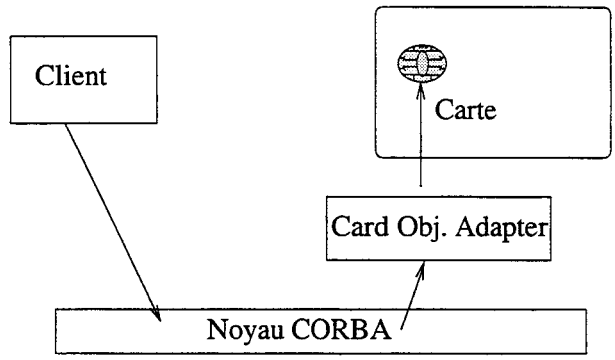


FIG. I.1.6 - *Intégration de la carte dans CORBA*

Il était donc souhaitable d'intégrer la carte à microprocesseur dans cet environnement. Le principal obstacle à cette intégration était le protocole de communication de la carte à microprocesseur (cf. paragraphe 1.3.2). Ce problème a été contourné en utilisant un composant d'adaptation (le COA pour Card Object Adaptor), qui transcrit les requêtes CORBA en requêtes carte à microprocesseur (cf. figure I.1.6).

Le dernier exemple d'intégration de la carte à microprocesseur dans un environnement distribué est donné par l'application Carte-Web-Santé [**MVD96**]. Dans cette application, une carte santé contient des liens vers des examens et des radiographies d'un patient. A partir de cette carte, le serveur Web va reconstituer une page HTML[16], pour ensuite l'envoyer au butineur du client. Ainsi, le patient se trouve toujours en possession des liens permettant d'accéder à ses examens médicaux. Depuis, d'autres projets poursuivent le même but : étendre les capacités des cartes aux systèmes distribués environnants [**CLT96**][**B98b**].

Comme on peut le voir, les cartes à microprocesseur sont de plus en plus intégrées dans des applications distribuées. L'accroissement des possibilités des cartes, ainsi que le développement des systèmes distribués va faire se rencontrer encore plus souvent ces deux mondes.

1.4.4 Une programmation plus facile...

Parallèlement aux travaux d'intégration, de nouveaux systèmes d'exploitation pour carte à microprocesseur voient le jour. Le développement des applications pour cartes en est grandement facilité.

En effet, jusque récemment, une application carte était développée de manière confondue avec le système d'exploitation. Cela nécessitait de très bonnes connaissances carte, et ne pouvait être fait que par le fabriquant de la carte (car l'ensemble application / système d'exploitation était embarqué dans la ROM de la carte à microprocesseur).

Dorénavant, de nouveaux systèmes de développement voient le jour, basés sur des interpréteurs intégrés dans la carte [**BGV96**]. Le dernier exemple de système d'exploitation carte muni d'un interpréteur, est la JavaCard [**JC97**] (cf figure I.1.7),

16. HTML : HyperText Markup Language

qui est basée sur un sous-ensemble du langage Java (langage de programmation orienté objet de SUN).

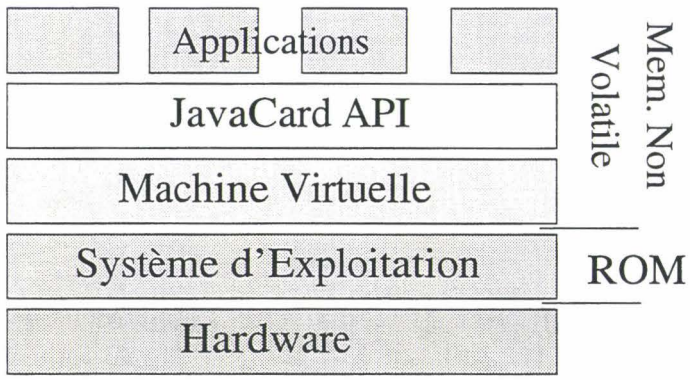


FIG. I.1.7 - *Architecture de la JavaCard*

Le but d'un tel système est d'ouvrir la programmation des cartes à microprocesseur au monde des programmeurs traditionnels. C'est pour cela que les applications de la JavaCard sont directement écrites en Java.

En plus de la facilité d'utilisation du langage de programmation, ces nouvelles cartes permettent de charger des applications de manière dynamique (les applications sont maintenant en mémoire non volatile réinscriptible de type Flash ou EEPROM, suivant les produits). Ce qui permet à tout un chacun de s'initier à la programmation des cartes à microprocesseur.

Il y a actuellement plusieurs produits commercialisés, sur la base de l'API JavaCard 2.0 de SUN. On peut notamment citer la CyberFlex de Schlumberger et la GemXpresso de Gemplus.

1.5 Les Applications Carte

Après avoir étudié la carte à microprocesseur sous ses aspects techniques, il est intéressant d'étudier les applications actuelles de ces cartes. Dans cette partie, nous allons donc tenter de classer les domaines d'application les plus répandus des cartes.

1.5.1 La Sécurisation des accès

Les cartes sont souvent utilisées pour offrir à des lieux (sécurité physique), ou à des systèmes (sécurité logique), la sécurité qui leur manque. Ces cartes de contrôle offrent aussi des possibilités d'identification du porteur par le système. L'exemple le plus couramment répandu est celui de la carte SIM[17], qui permet la sécurisation des accès au terminal GSM d'une part, et l'identification du porteur de la carte par le prestataire de services de téléphonie d'autre part [**ETSI95**].

17. SIM : Subscriber Identity Module

La carte est un support parfaitement adapté à ce type d'application, car le secret qu'elle porte est parfaitement protégé par sa sécurité intrinsèque. De plus la carte à microprocesseur est capable de traitement sur les données qu'elle porte. Cela permet, par exemple, d'imaginer des nouveaux systèmes d'identification et de contrôle d'accès à base de biométrie [**A95**], pour remplacer la solution actuelle du code NIP[18].

1.5.2 Les Cartes de paiement

Une autre utilisation très répandue des cartes à microprocesseur est le paiement électronique. Les cartes de paiement peuvent être divisées en plusieurs catégories :

- Les cartes de pré-paiement : la carte peut être vendue préchargée d'unités (c'est le cas par exemple, de la carte téléphonique de France Télécom). Il se peut aussi que la carte soit rechargée au cours du cycle d'utilisation de la carte à microprocesseur (c'est le cas des PME[19], ou des cartes de communication GSM sans abonnement).

- Les cartes de crédit et de débit : ces cartes sont celles communément nommées «Cartes Bancaires». Avec les cartes de crédit, les achats sont débités par mensualités avec un taux d'intérêt, et avec les cartes de débit, le compte bancaire du titulaire est débité quelques jours après l'achat (un bon exemple de carte de débit est la «carte bleue», ou CB du GIE Carte Bancaire).

L'intérêt de la monnaie électronique est important. En effet, cette monnaie permet une baisse des liquidités. Ce qui facilite les traitements, autant pour les banques que pour les commerçants. En résumé, les avantages pour les divers intervenants sont les suivants :

- Pour la banque : moins de fraude, baisse des liquidités, meilleur contrôle des crédits,

- Pour le commerçant : garantie de paiement par la banque, rapidité d'encaissement sans passer par sa propre banque, absence de liquidité,

- Pour l'acheteur : facilité et rapidité de paiement, protection contre le vol (grâce aux divers codes).

De plus, la monnaie électronique, de par sa simplicité d'utilisation et sa sécurité, permet le développement rapide de la vente par correspondance, ainsi que du commerce électronique (Minitel avec lecteur de carte, ou Internet).

1.5.3 Les Cartes «Dossier Portable»

On peut parler de dossier portable pour un objet ou pour une personne.

En effet, pour répondre à des soucis de qualité croissants, ou pour faciliter les transactions, ou déplacements, des objets, la traçabilité de ceux-ci est de plus en

18. NIP : Numéro d'Identification Personnel
19. PME : Porte Monnaie Electronique

plus souvent assurée par des Tags, qui sont en fait des marqueurs informatiques. Ces marqueurs sont maintenant de plus en plus sophistiqués, et deviennent capables d'émettre et de recevoir des requêtes [**B98**].

En ce qui concerne les cartes de dossier portable pour les personnes, on peut citer les cartes d'assurance maladie comme la SESAME Vitale en France [**SES96**], ou encore les dossiers médicaux [**P88**], et les cartes étudiants. Ces cartes regroupent des informations sur leur porteur, et les systèmes d'informations qui utilisent ces cartes, consultent ou mettent à jour les informations.

1.6 Carte Multi-services

Dans certains secteurs applicatifs (comme les banques ou les télécommunications), la concurrence devient très importante. La différence ne se fait plus forcément sur les tarifs, mais surtout sur les services offerts. Dans cette optique, les cartes à microprocesseur vendues aux clients doivent offrir de plus en plus de possibilités. Une même carte doit donc offrir le maximum de services. Par exemple, les nouvelles cartes GSM devront offrir des PME intégrés. La réalisation des applications seules n'est pas un problème (on sait actuellement comment faire une carte GSM, ou un PME). Ce qui pose problème, c'est la présence de ces applications sur une même carte.

Les principaux problèmes de ces cartes multi-services sont :

- 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).

- 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 diverses 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.

Le montage de ces applications cartes est donc actuellement «bricolé au couteau», pour répondre à la demande des émetteurs d'applications. Nous verrons par la suite que ces nouvelles cartes demandent des mécanismes de gestion des données complexes.

Cette évolution va se poursuivre et s'amplifier, pour déboucher progressivement vers une intégration plus complète de la carte dans des environnements plus évolués (cf. chapitre II.1).

1.7 Conclusion

Le monde de la carte à microprocesseur qui était jusqu'à maintenant très hermétique, tant au niveau matériel, qu'au niveau applicatif, est en train de s'ouvrir au monde extérieur. Cette ouverture est rendue possible aussi aux progrès des technologies utilisées dans la carte (processeurs, mémoires, systèmes d'exploitation). Le changement le plus marquant est le passage d'un masque (comprenant le système et l'application) situé en ROM, à un véritable système d'exploitation (toujours en ROM) avec lequel on peut utiliser une ou plusieurs applications (en mémoire non volatile réinscriptible). La preuve est faite par la JavaCard, qui permet à tout un chacun d'écrire et d'installer ses propres applications dans les cartes à microprocesseur. La carte y gagne en facilité de programmation, et l'arrivée de nouveaux programmeurs que cela implique, va sans aucun doute doper le développement des applications pour cartes.

De plus, la demande de service par les émetteurs traditionnels de cartes à microprocesseur est de plus en plus grande, et les fabricants de cartes ont du mal à répondre à temps à ces demandes (notamment en terme de carte multi-applications). De ce point de vue, les cartes ont marqué un net progrès ces dernières années (cartes génériques plus proches des architectures logicielles classiques, et début d'intégration dans les systèmes distribués).

Cependant, il reste de nombreux problèmes en suspens pour réellement ouvrir les cartes à de nouveaux services et au monde extérieur (problème de déconnexion de la carte, protocoles de sécurité à respecter pour le chargement des applications).

De plus, des problèmes de coopération entre les applications cartes, ou d'intégration complète de ces applications dans les systèmes distribués restent posés. Nous allons donc tenter, dans ce mémoire, d'étudier ces problèmes, et de proposer un modèle pour les résoudre.

Chapitre I.2

Le modèle Transactionnel

2.1 Introduction

Comme nous l'avons vu dans le premier chapitre de ce document, les cartes à microprocesseur s'ouvrent de plus en plus aux systèmes distribués. Nous nous retrouvons donc dans un contexte d'exécution distribué, et sujet aux fautes (encombrement du réseau, serveur non disponible, ou non présence de la carte à microprocesseur). Le traitement de ces fautes par le programmeur de l'application se révèle comme étant un véritable casse-tête.

Nous allons donc étudier dans cette partie, en quoi le modèle transactionnel représente une bonne solution à ces problèmes. Une transaction peut être définie comme étant une suite d'actions commençant par un «Début_Transaction» (Begin_Transaction en anglais), et se terminant par un «Validation_Transaction» (Commit_Transaction en Anglais), si tout se passe bien, ou par un «Annulation_Transaction» (Rollback ou Abort en Anglais), s'il y a eu un problème.

Le modèle transactionnel définit un modèle d'exécution pour applications simultanées, qui garantissent un certain nombre de propriétés. Dans un premier temps, nous étudierons en détail ces propriétés (appelées propriétés ACID[1] **[HR83]**), pour ensuite, nous intéresser aux différents mécanismes existants nécessaires à l'implantation du modèle transactionnel sur des systèmes distribués (mécanismes de contrôle de concurrence et de reprise sur panne).

Nous terminerons cette partie par l'étude des mécanismes de validation distribuée

1. ACID : Atomicité, Cohérence, Isolation et Durabilité

des actions des transactions, ainsi que des implantations existantes.

2.2 Les propriétés ACID

Pour éclaircir les propos sur les propriétés des transactions, nous allons utiliser un exemple très simple de transaction : le débit d'un compte A, pour créditer un compte B.

```
BeginTransaction
  CompteA.débit(100);
  CompteB.crédit(100);
CommitTransaction
```

2.2.1 Atomicité

L'atomicité d'une transaction peut être définie comme étant la règle du «tout ou rien». En effet, lorsque l'on exécute une transaction, on doit être en mesure d'assurer que, soit toutes les actions de la transaction ont été validées, soit aucune de ces actions n'est conservée. Si nous prenons l'exemple de notre virement bancaire, cela signifie que s'il y a une panne après le débit du compte A, ce compte doit être recrédité, car la transaction n'a pas été validée.

En pratique, cela signifie qu'aucun effet d'une transaction annulée ne doit apparaître dans le système.

Comme nous le verrons plus en avant dans ce document, l'atomicité est du ressort de la tolérance aux pannes.

2.2.2 Cohérence

La propriété de maintien en cohérence des données dit qu'une transaction doit prendre des données dans un état cohérent, et les rendre dans un autre état cohérent.

La cohérence est obligatoirement violée pendant la transaction. Si on prend l'exemple de transaction du versement bancaire, la cohérence des données n'est pas respectée après le débit (et avant le crédit), mais tout rentre dans l'ordre à la fin de la transaction.

Le maintien de la propriété de cohérence est donc principalement du ressort du programmeur. La propriété d'Atomicité veille à ce que les règles de travail sur les données définies par le programmeur, soient respectées, en garantissant l'exécution de toutes les actions prévues par le programmeur.

2.2.3 Isolation

Cette propriété dit que les modifications d'une transaction sont invisibles aux transactions concurrentes. Ce qui signifie, que lors de l'exécution concurrente de plusieurs transactions, chaque transaction doit apparaître comme si elle s'exécutait seule.

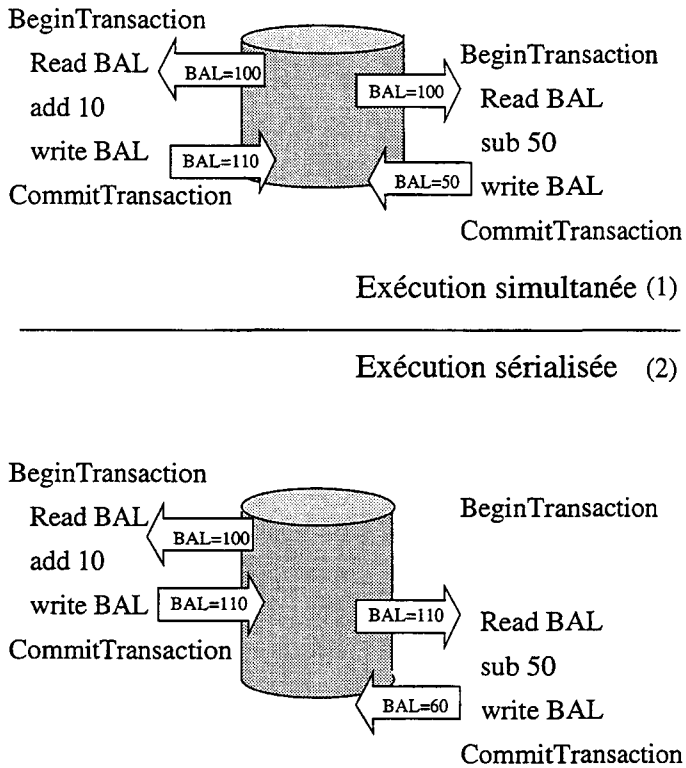


FIG. I.2.1 - *La propriété d'Isolation*

Prenons comme exemple, la figure I.2.1. Dans le premier cas, les deux transactions s'exécutent simultanément, mais le résultat final (c'est à dire après les deux transactions) ne serait pas le même si ces transactions s'étaient exécutées individuellement. Il faut donc utiliser des mécanismes de *sérialisation*[2], qui ordonnent les actions des transactions, pour respecter la propriété d'isolation (exemple (2) dans la figure I.2.1). Ces mécanismes seront décrits plus en détail dans le paragraphe 2.3.

2.2.4 Durabilité

La propriété de durabilité implique que les actions d'une transaction ne peuvent pas être perdues si la transaction est validée. Cela signifie que les résultats d'une transaction doivent être conservés, même après une panne.

Il est évident que la durabilité absolue n'existe pas, mais on peut définir plusieurs niveaux de protection des résultats (possibilité de refaire les actions perdues, ou support de sauvegarde).

La durabilité est du ressort des mécanismes de reprise sur panne.

2. Une exécution de transactions est dite sérialisable si ses effets sont les mêmes qu'en exécutant les transactions les unes après les autres

2.2.5 Implantation des propriétés ACID

Comme nous avons pu le voir au cours de l'explication des diverses propriétés, seuls deux mécanismes sont nécessaires pour implanter les propriétés ACID dans un système : il faut un mécanisme de sérialisation et un mécanisme de reprise sur panne.

Avant d'étudier plus en détail ces mécanismes, nous allons examiner des modèles de transactions plus évolués, qui répondent à des problèmes, à la fois plus précis et plus complexes.

2.3 Le contrôle de concurrence

Nous allons présenter ici les méthodes garantissant la propriété d'isolation. Comme nous l'avons vu dans la partie 2.2, la propriété d'Isolation s'obtient en appliquant le principe de la sérialisation. Les techniques employées pour implanter ce principe sont les techniques de contrôle de concurrence [**BHG87**].

Nous allons ici présenter les diverses techniques existantes de contrôle de concurrence. On peut tout d'abord différencier deux grandes familles de protocoles de contrôle de concurrence :

- Les protocoles pessimistes : Aussi appelés contrôle continu. Le contrôle détecte les conflits au fur et à mesure de l'exécution. Si l'exécution d'une instruction entre en conflit avec les règles de sérialisabilité établies, l'instruction est différée, ou même rejetée (ce qui provoque un abandon de la transaction)

- Les protocoles optimistes : les opérations sont exécutées librement jusqu'à la fin de la transaction, où celle-ci transaction est soumise à une phase de *certification* (toutes les dépendances sont contrôlées). Alors, la transaction est soit validée, soit abandonnée (dans le cas où la sérialisabilité n'est pas respectée).

Il est à noter que la première méthode à un coût même si la transaction se déroule sans problème. Toutefois, ce coût reste modéré dans tous les cas. La seconde méthode a l'avantage d'être sans coût dans le cas où tout se passe bien, mais elle est très pénalisante en cas de conflit.

Dans un premier temps, nous allons étudier deux méthodes de contrôle de concurrence pessimistes (le verrouillage à deux phases et l'estampillage) puis nous étudierons une méthode optimiste (la certification).

2.3.1 Le verrouillage à deux phases

Il s'agit d'une technique dérivée du principe d'exclusion mutuelle. Chaque transaction pose un verrou avant toute opération de lecture ou d'écriture sur une granule[3].

Il existe deux modes d'accès à une donnée : la consultation et la modification. Par rapport à cela, il existe deux types de verrous : les verrous partagés (utilisés

3. On appelle granule tout élément de base du système. Par exemple, un octet, ou bien une donnée, ou un objet (suivant la granularité choisie)

	Lire	Ecrire
Lire	OUI	NON
Ecrire	NON	NON

TAB. I.2.1 - *Compatibilité des instructions*

par les lectures pour interdire les écritures) et les verrous exclusifs, utilisés par les écritures pour exclure les autres écritures et les lectures. La règle logique découle de la table de compatibilité I.2.1.

`Règle : N Lecteurs XOR 1 Rédacteur`

Le verrouillage à deux phases comporte, comme son nom l'indique, deux étapes :

- La phase de verrouillage : Lorsqu'une opération d'écriture ou de lecture arrive au niveau du système d'exploitation, le gestionnaire de verrous est contacté. Il vérifie alors que la variable accédée n'est pas verrouillée, ou que son verrouillage est compatible (cf. table I.2.1). Si elle est libre, il la verrouille. Si la demande de verrouillage n'est pas compatible, l'opération est soit annulée, soit reportée.
- La phase de relâchement : Cette phase consiste à libérer une variable précédemment verrouillée. Si des transactions ont été mises en attente à cause du verrouillage sur cette donnée, elles sont «réveillées». Dans le cas d'un «verrouillage rigoureux», les verrous sont levés après la validation ou l'annulation d'une transaction (tous les verrous posés par cette transaction sont libérés simultanément). Le verrouillage rigoureux a comme principal avantage d'être plus simple d'utilisation, mais il limite encore plus la concurrence de transaction.

Le principal problème des techniques à base de verrouillage est le risque d'interblocage[4] des transactions, car elles sont basées sur l'attente par une transaction de la libération des variables qu'elle tente d'accéder. Un cycle dans le graphe d'attente peut donc se former. Cependant, il existe plusieurs techniques pour remédier à ce problème :

- On peut affecter à chaque transaction un temps maximal d'exécution. Si ce temps est dépassé, on peut soit annuler la transaction, soit la redémarrer.
- Une autre solution consiste à rendre impossible les interblocages, en affectant à chaque transaction une estampille (méthode de prévention dynamique). En cas de conflit, la moins prioritaire des transactions, généralement la plus récente, est abandonnée [**RLS78**].

4. Deadlocks en Anglais

- Enfin, il existe des méthodes de détection des interblocages. Dans ces méthodes, le système tient à jour le graphe d'attente, et la recherche de cycle est lancée périodiquement, ou lorsqu'une transaction se bloque.

Les avantages de la technique du verrouillage à deux phases sont principalement la simplicité de l'algorithme pour poser les verrous, et pour tester si une variable peut être utilisée ou non, par une transaction. Il s'agit généralement d'une table système pour poser les verrous et de l'utilisation d'un masque pour tester la variable.

Le problème de cette méthode est la libération des variables à la validation, ou à l'abandon, d'une transaction. Il faut alors parcourir tous les verrous pour trouver les variables verrouillées par une transaction donnée.

2.3.2 L'estampillage

Il s'agit d'un ordonnancement initial des transactions: une valeur numérique (ou estampille) est associée à chaque transaction (par exemple, la date de début de transaction, ou un compteur incrémenté au fur et à mesure). Il suffit que la génération de ces estampilles vérifie une relation d'ordre total stricte et qu'elle soit croissante. Après cela, il suffit de vérifier que les granules soient accédés par des transactions ayant des estampilles croissantes (car la granule conserve l'estampille de la dernière transaction qui l'a accédée) (cf. Tableau I.2.2). Si ce n'est pas le cas, la transaction est entièrement rejouée et pour cela, elle reçoit une nouvelle estampille.

Transaction 1 Estampille = 1	Transaction 2 Estampille = 2	Valeur Estampille	Observation
Lire (A)		Estampille = 1	OK
	Lire (A)	Estampille = 2	OK
	Ecrire (A)	Estampille = 2	OK
Ecrire (A)		Estampille = 2	Impossible

TAB. I.2.2 - *Exemple d'Estampillage*

Plusieurs techniques d'optimisation existent pour cet algorithme, comme la distinction des estampilles de lecture et d'écriture.

Avec cette technique, il n'y a pas de problème d'interblocage. En effet, si une transaction a une estampille qui lui permet de travailler, elle accède aux données

dont elle a besoin. Mais si elle ne satisfaisait pas les conditions et qu'elle avait encore besoin d'accéder à une donnée, elle devrait être entièrement rejouée. Elle n'est donc pas bloquée et une variable ne peut pas être prise dans un cycle de blocage.

2.3.3 Contrôle Optimiste

Ces méthodes de contrôle de concurrence, aussi appelées contrôle de concurrence par certification, sont plus performantes si il n'y a pas de risque fort de conflit sur les variables [**KR81**]. Généralement, ces techniques sont très coûteuses en cas de conflit.

Ces méthodes laissent les dépendances s'installer entre les transactions, et repousse le contrôle de la sérialisabilité d'une transaction jusqu'à sa terminaison.

L'exécution d'une transaction comporte plusieurs étapes :

- une étape pour les traitements et les lectures des données dans l'espace de travail,

- puis, après une phase de certification, une étape d'écriture et de validation des modifications dans la base de données.

La méthode présentée ici consiste à ordonnancer les transactions selon leur ordre d'arrivée à la fin de l'étape d'exécution des opérations dans l'espace de travail. Si la transaction ne satisfait pas les critères de sérialisabilité contrôlé lors de la certification, elle est abandonnée (et elle devra être réexécutée depuis le début).

Etant donné que le contrôle s'effectue après l'exécution de la transaction, son efficacité mise donc sur un faible taux de conflit, car dans ce cas, le déroulement de la transaction consomme inutilement des ressources.

De plus, cette méthode est mal adaptée à un mode de mise à jour immédiate, car les transactions peuvent utiliser librement les effets d'une transaction pas encore validée. Dans ce cas, il y a donc de forts risques d'abandons en cascade.

Le principal avantage de ces méthodes est de permettre un plus haut niveau de concurrence.

2.3.4 Prise en compte de la sémantique

Les techniques de contrôle de concurrence strict (basées uniquement sur les opérations de lecture et d'écriture) ont comme principal défaut de trop limiter la possibilité d'exécution simultanée des transactions. La prise en compte de la sémantique des opérations typées[5] permet de reconnaître comme sérialisables des exécutions qui ne l'auraient pas été avec un contrôle de concurrence strict. Ainsi, si deux opérations agissant sur un même objet sont commutables, alors il n'y a pas de conflit [**CFR89**].

5. Les opérations typées sont définies sur des objets typés. Un objet typé est constitué de données ayant une caractéristique commune et d'un ensemble d'opérations pour les manipuler (par exemple un compteur contient les opérations «incrémenter» et «décrémenter»).

2.3.5 Choix de la méthode

Le choix du contrôle de concurrence s'effectue d'après le degré de contention[6] des données et aussi en fonction du type de mise à jour des données choisi (immédiate ou différée).

Si on a un fort degré de contention entre les données, il est préférable d'utiliser un mode de contrôle de concurrence pessimiste, car le mode optimiste obligerait souvent à refaire les transactions.

De plus, si on utilise un mode de mise à jour de la base de données immédiat, le contrôle optimiste n'est pas adapté. En effet, ce dernier permet l'utilisation de données déjà modifiées par une autre transaction (pas encore validée). Les risques d'abandon en cascade des transactions deviennent donc très importants.

2.4 Reprise sur panne

Sur un système informatique, nous pouvons définir plusieurs types de «pannes», qui impliquent plusieurs problèmes différents :

- Abandon d'une transaction : cela peut impliquer de défaire les actions de la transaction abandonnée. Ce cas de «panne», dans le cas de systèmes distribués traditionnels, être par exemple dû à un problème d'accès concurrent sur une donnée, ou à l'utilisateur [**GR93**].

- Panne du système : lorsque le système tombe complètement en panne, le contenu de la RAM est perdu, et le contenu du support de mémoire stable n'est pas affecté par ce type de panne. Dans ce cas, il faut complètement défaire toutes les transactions en cours (cf. cas précédent), ou éventuellement les transactions validées (dont les effets ne sont pas présents sur le disque).

- Panne de journal[7] : dans ce cas là, toutes les modifications effectuées sont perdues. Cela implique de tout refaire depuis le début ou d'utiliser un support de sauvegarde.

De manière à implanter les propriétés d'Atomicité et de Durabilité, nous devons utiliser des mécanismes de reprise sur panne [**BHG87**] [**GR93**].

Dans cette partie, nous allons tout d'abord étudier les différentes gestions d'écriture possibles (mise à jour des données directement sur le support ou non) puis, nous étudierons différentes techniques de reprise sur panne (utilisation de journaux ou de mécanismes de répliquât).

6. On appelle degré de contention, le niveau de risque d'accès concurrent sur une donnée. Plus le risque est fort, plus le degré de contention est dit élevé.

7. le journal est un fichier séquentiel utilisé par certains mécanismes de reprise sur panne. Il sera présenté dans le paragraphe 2.4.2

2.4.1 Influence de la gestion du cache sur la reprise

Il s'agit de la gestion du cache et de la façon dont il est recopié sur le support stable. Il existe quatre méthodes de mise à jour des données [**GV89**] :

- Soit les objets présents dans le cache sont recopié avant la validation de la transaction. Il se peut alors que l'on soit obligé de *défaire*[8] des transactions, mais on a jamais à *refaire*[9] une transaction.

- Soit les données sont mises à jour uniquement après la validation (dans ce cas là, la transaction n'est jamais défaite, mais il peut être nécessaire de la refaire).

- Soit la mise à jour des données sur le support est sans contrainte, ce qui est le cas le plus fréquent. On peut alors avoir à défaire et à refaire des actions de transactions.

- Enfin, la dernière méthode consiste à mettre à jour les données non pas en remplaçant les données initiales, mais en faisant la mise à jour «ailleurs» sur le support stable. Cette méthode un peu à part sera détaillée dans la partie 2.4.3.

Les mises à jour des données sont appelées des *points de reprise*. Ce sont en fait des points du journal, où on est assuré que toutes les actions antérieures ont leurs effets présents sur le support stable.

2.4.2 Méthodes de Journalisation

Les méthodes de journalisation («Logging» en anglais)sont basées sur le principe qu'une transaction est en fait une suite d'actions. A chaque action, il suffit de laisser une trace dans un journal. Alors, pour refaire ou défaire une transaction, on parcourt le journal.

Il existe plusieurs types de journaux [**R92**] pour assurer le recouvrement des actions d'une transaction :

- Les Journaux de valeurs : Dans ces journaux, on stocke la valeur des objets modifiés par une transaction. Il existe deux types de journaux de valeurs, les *journaux des images avant* et *les journaux des images après*. Dans le premier cas, on sauvegarde une image des objets modifiés par la transaction avant que celle ci ne les change ; alors que dans le second cas, on sauvegarde la nouvelle valeur de ces objets.

- Les journaux différentiels : Dans ces journaux, on effectue un XOR (OU exclusif) entre ancienne et nouvelle valeur de l'objet. Alors, le journal ne contient pas de réelles valeurs des données, ce qui permet de ne pas sortir de secret dans le journal.

8. On appelle «défaire» une transaction, éliminer les actions d'une transaction sur le disque
9. On appelle «refaire» une transaction, rejouer une transaction pour intégrer des actions perdues

- Les journaux d'opérations : Ce journal sauvegarde toutes les opérations effectuées sur les objets avec leurs opérandes et, nécessairement les opérations inverses correspondantes.

Quel que soit le type de journal, l'implémentation est en fait un fichier séquentiel (logiquement, et physiquement pour accélérer le débit des écritures) où est enregistré une suite d'actions. Ce fichier doit obligatoirement être stocké sur un support stable, car son contenu ne doit pas être perdu.

Pour défaire une transaction, nous avons besoin de conserver dans les journaux les *images avant* des données (c'est-à-dire la valeur des données avant modification par la transaction). Pour pouvoir défaire une transaction, il faut que l'image avant de la donnée soit écrite dans le journal avant que la donnée ne soit modifiée (donc avant que la donnée migre du cache). A l'inverse, pour refaire une transaction, il faut conserver les *images après* des données, et dans ce cas, les informations doivent être stockées dans le journal avant l'ordre de validation.

2.4.3 Méthode des Pages Ombres

La méthode des pages ombres («Shadow Pages» en anglais) est une méthode de mise à jour «ailleurs» (cf. figure I.2.2) qui peut se faire au moment de la validation de la transaction (ainsi, nous n'avons pas besoin de journaux pour refaire ou défaire les transactions).

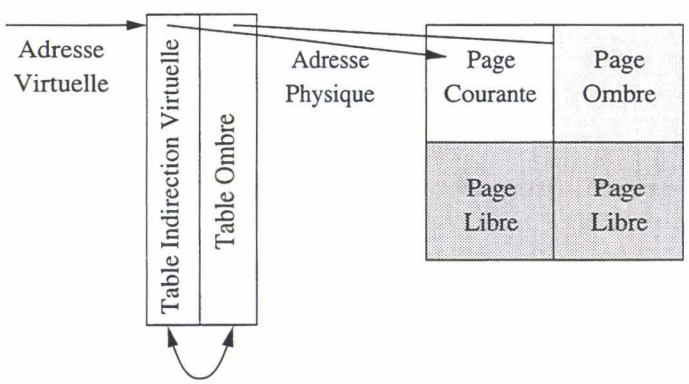


FIG. I.2.2 - *Méthode des pages ombres*

Le principe de fonctionnement des pages ombres est le suivant :

- Avant la transaction, la table d'indirection virtuelle est divisée en deux partie : une table valide qui pointe sur les pages courantes, et une table ombre.

- Pendant la transaction, lors de la modification d'une donnée, la nouvelle valeur de la donnée va être écrite «ailleurs» sur le support stable, et va être pointée par la partie ombre correspondante de la table d'indirection. Ainsi, les données de la page courante ne sont pas modifiées.

- A la validation, la nouvelle table valide est reconstituée : les parties de la table ombre pointant sur des données modifiées par la transaction font dorénavant parties de la table courante, et les parties contenant les données initiales sont maintenant dans table ombre.
- En cas d'annulation, la table d'indirection virtuelle valide n'est pas modifiée (les données pointées par la page ombre sont donc perdues).

Ce mécanisme est très simple d'utilisation et d'implantation pour un système exécutant très peu de transactions simultanément. Il devient moins efficace que la journalisation pour des systèmes transactionnels plus évolués.

2.5 Transactions distribuées

Le modèle transactionnel est un modèle de tolérance aux pannes dans les environnements distribués. Dans cette partie, nous allons voir au travers d'un exemple ce qu'implique la distribution pour le modèle transactionnel.

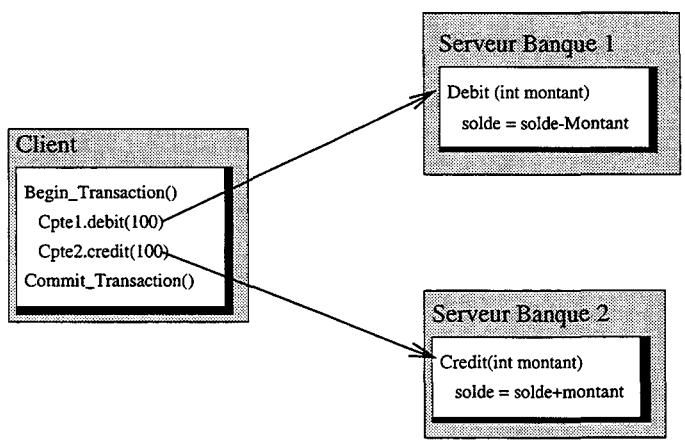


FIG. I.2.3 - *Exemple simple de transaction distribuée*

L'exemple choisit est celui d'un virement bancaire entre deux banques (cf. figure I.2.3). Dans une architecture distribuée, chaque site est responsable du respect des accès à ses données. Cela signifie que chaque site doit implanter un système transactionnel pour assurer le maintien des propriétés ACID.

Cependant, le caractère distribué de l'application a des conséquences sur les mécanismes de maintien des propriétés ACID. Nous allons tout d'abord étudier l'influence de la distribution sur les mécanismes de contrôle de concurrence, puis sur les mécanismes de reprise sur panne. Enfin, nous étudierons les problèmes de validation distribuée des transactions.

2.5.1 Implication sur le contrôle de concurrence

Les mécanismes de contrôle de concurrence appliqués dans le cadre des transactions simples peuvent être utilisés tels quels dans le cas des transactions distribuées

(sous réserve que les sites utilisent tous la même méthode ,et qu'ils ne privilégient pas les transactions locales).

Cependant, les problèmes posés dans le cadre d'une utilisation simple des transactions (c'est-à-dire sur un seul site) sont aggravés par le contexte distribué. Prenons, par exemple, les mécanismes de verrouillage. La principale difficulté liée à leur utilisation est le risque d'interblocage. Ce risque devient beaucoup plus difficile à détecter dans le cadre d'une transaction distribuée sur plusieurs sites.

2.5.2 Implication sur la reprise sur panne

Dans le cas des systèmes distribués, un nouveau type de panne peut apparaître (en plus des trois précédemment cités). Il s'agit d'une panne du réseau de communication entre les différents sites (Remarque : il n'est pas toujours facile de distinguer une panne de site, d'une panne de communication).

Cependant, les techniques de reprise sur panne étudiées dans le cadre des transactions pour une architecture centralisée sont transposables dans le cadre des transactions distribuées.

Chaque site doit implanter son propre mécanisme de reprise sur panne. Le problème est de décider si le site doit valider ou abandonner une transaction distribuée. En effet, il est tout à fait possible qu'un site ait réussi à exécuter correctement une transaction, mais qu'un autre participant ait échoué. Dans ce cas, la propriété d'Atomicité impose un abandon de la transaction par tous les sites. Ce problème est traité dans la partie suivante.

2.5.3 Validation de la Transaction Distribuée

De par la propriété d'Atomicité, à la fin d'une transaction distribuée, tous les participants doivent décider uniformément de valider ou d'abandonner la transaction. Cela oblige à synchroniser la prise de décision, car tous les participants ne terminent pas l'exécution de la transaction simultanément. La solution consiste à utiliser un *Coordinateur*, qui, en suivant un algorithme de validation, va ordonner aux *Participants* de valider, ou bien d'abandonner, la transaction.

Dans un premier temps, nous allons approfondir le rôle de ce coordinateur, pour ensuite étudier divers algorithmes de validation de transaction distribuée (dont le plus couramment utilisé : l'algorithme de validation à deux phases [**G81**]).

2.5.3.1 Coordination de la validation

Les protocoles de validations atomiques sont basés sur le vote des participants pour la validation de la transaction. La transaction ne sera validée que si tous les participants votent pour la validation.

Le coordinateur de la transaction[10] (ou *gestionnaire de transaction*[11]) reçoit tous les votes des participants et prend la décision de validation ou d'abandon de la transaction.

10. Coordinator en Anglais
11. Transaction Manager en Anglais

Le coordinateur peut être un des participants de la transaction, ou un site à part. Certains algorithmes (appelés algorithmes bloquants) sont bloqués si le coordinateur est en panne (c'est le cas notamment du protocole de validation à deux phases). Nous verrons qu'il existe des algorithmes non bloquants, mais qui, la plupart du temps, s'avèrent plus coûteux en nombre de messages.

2.5.3.2 Le protocole de Validation à deux phases

Comme son nom l'indique, le protocole de validation à deux phases se déroule en deux parties (cf. figure I.2.4) :

- Pendant la première phase, le coordinateur récolte l'état de tous les participants mis en oeuvre par la transaction, de manière à pouvoir prendre la décision de validation ou bien d'annulation (c'est la phase de *préparation*).

- La seconde phase consiste à appliquer à tous les participants, soit la décision de COMMIT (validation), soit de ABORT (annulation) de la transaction. Cette décision doit être appliquée de manière fiable (résistante aux pannes).

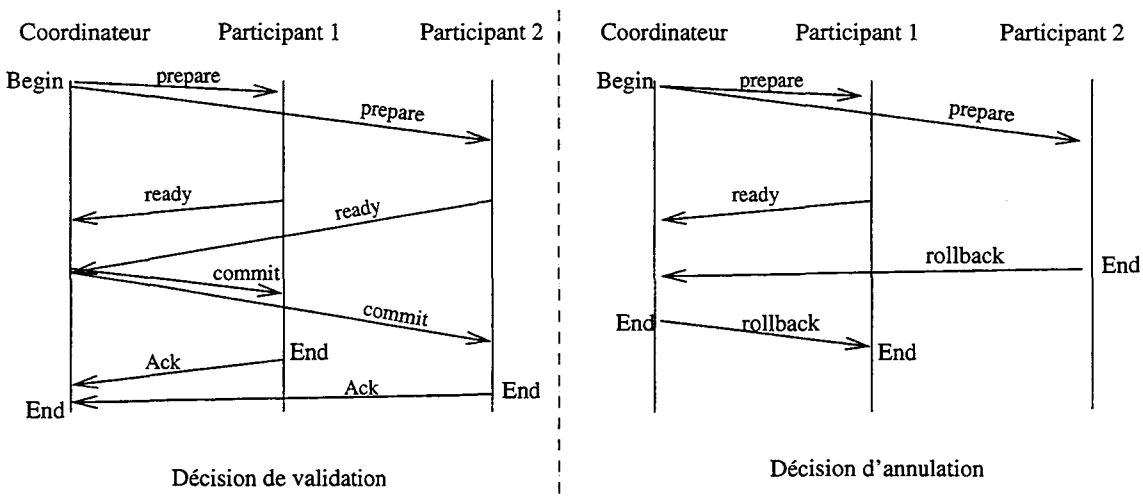


FIG. I.2.4 - *Protocole de Validation à 2 phases*

De nombreuses stratégies d'optimisation en fonction de la topologie de réseau existent **[T91]**. Cependant, le principal problème de cet algorithme est qu'il est bloquant : si le coordinateur tombe en panne entre la phase de vote et de décision, les participants qui ont voté la validation de la transaction se retrouvent bloqués : ils ne savent pas quoi décider (ceux qui ont voté l'annulation savent que la transaction va être abandonnée). Les données manipulées sur les participants par la transaction bloquée restent donc inaccessibles.

Pour éviter cela, il existe d'autres algorithmes, dits «non bloquants», que nous allons maintenant étudier.

2.5.3.3 Les protocoles non bloquants

Le protocole de validation non bloquant le plus répandu est le protocole de validation à trois phases [**S81**], qui en cas de panne du coordinateur, permet l'élection d'un nouveau coordinateur. Un des défauts de cet algorithme est d'être très coûteux en nombre de messages.

D'autre part, cet algorithme ne supporte pas les panne de support de communication. Dans le cas où le coordinateur tombe en panne, le protocole est bien non bloquant. Mais dans le cas où la panne est due à un problème de communication, il faut qu'il soit bloquant pour assurer la cohérence de la validation.

Des optimisations de ce protocole ont été proposées [**GLS96**].

D'autres protocoles non bloquants, basés sur des protocoles de validation à une seule phase, et sur les caractéristiques des systèmes de communications synchrones voient le jour [**AP97**]. Le protocole de validation peut être réalisé en une seule phase par élimination de la phase de vote du protocole de validation à deux phases. Pour cela, ces protocoles partent du principe que les participants utilisent un contrôle de concurrence pessimiste (de type verrouillage à 2 phases rigoureux), ce qui signifie qu'une transaction qui a exécuté ses actions ne peut pas être abandonnée par un problème d'accès concurrent.

2.6 Modèles Avancés de Transactions

A partir du modèle de base explicité précédemment, un certain nombre d'extensions sont apparues, principalement pour rendre moins contraignantes les propriétés ACID [**E92**]. Nous allons étudier, ci-après une partie de ces extensions, ainsi que leurs implications sur les diverses applications transactionnelles.

2.6.1 Les Transactions Emboîtées

Les transactions imbriquées[12] [**M85**] introduisent la notion de sous-transaction. Elles assurentles propriétés ACID pour la transaction globale, et les propriétés d'atomicité et d'isolation pour les sous transactions. Ces sous-transactions répondent à trois propriétés :

- Une transaction fille démarre après le début de la transaction mère, et se termine avant la fin de la transaction mère.

- L'abandon d'une transaction mère entraîne l'abandon des transactions filles (mêmes si elles ont été validées).

- La validation d'une transaction fille est conditionnée par la validation de la transaction racine globale. Les modifications des transactions filles ne deviennent donc définitives (c'est-à-dire durables) qu'à la validation de la transaction mère.

12. Nested Transaction en anglais

Il est important de noter que l'abandon d'une transaction fille n'entraîne pas obligatoirement l'abandon de la transaction mère. La transaction mère décide de la stratégie à adopter (refaire les transactions filles échouées, ignorer leur échec, ou abandonner la transaction au moindre problème). Ainsi, on peut réaliser un contrôle modulaire de l'exécution.

D'autres modèles de transactions emboîtées (multi-niveaux [**BSW88**]) plus ouverts relaxent la propriété d'isolation en permettant que les effets des sous-transactions validées deviennent visibles aux autres transactions concurrentes.

2.6.2 Les Sagas

Le modèle des sagas [**GS87**] est un modèle de transactions avancées, qui permet de libérer la propriété d'Isolation. Ce modèle assure les propriétés d'atomicité, de cohérence et de durabilité pour la transaction globale, et les propriétés d'atomicité, d'isolation et de durabilité pour les sous-transactions. La validation d'une sous-transaction n'est plus conditionnée par la validation de la transaction mère (appelée saga).

A l'inverse, la validation de la transaction racine est toujours conditionnée par la validation de toutes ses sous-transactions. Ainsi, si une sous-transaction est abandonnée, il faut abandonner la transaction mère, et donc défaire toutes les sous-transactions déjà validées.

Pour défaire une sous-transaction, il faut définir des *actions de compensation*[13] [**BGM95**]). Cela implique que pour chaque sous-transaction, il doit exister une transaction de compensation correspondante (ce qui n'est pas toujours possible).

Ce modèle a été conçu pour répondre à la problématique des transactions longues, qui risqueraient de ne jamais valider si on maintient les contraintes d'isolation, contenu des nombreux conflits avec les autres transactions, et les risques d'interblocage qui en découle. Pour cela, ce modèle permet de relâcher des variables qui peuvent servir à d'autres sous-transactions concurrentes.

2.6.3 Modèle à flots de tâches

Ce modèle a lui aussi été conçu pour répondre aux problèmes liés aux transactions de longue durée [**BCF97**]. Ce modèle est basé sur la construction et l'exécution d'un scénario, fixant les enchaînements des transactions. Cela permet de relâcher les propriétés d'Atomicité et d'Isolation de la transaction globale, tout en maintenant la cohérence sémantique de l'application.

Le scénario peut être du type suivant :

```
Si la Transaction T1 est validée
   Alors
     Exécuter la Transaction T2
   Sinon
     Exécuter la Transaction T3
```

13. On appelle actions de compensation, des actions permettant d'annuler une transaction après sa validation

Le modèle à flots de tâches a comme objectifs de :

- Permettre l'arrêt volontaire d'une exécution, et sa reprise, dans un délai non fixé.
- Permettre une visibilité des résultats avant la fin de la transaction globale (ce qui peut nécessiter là aussi des actions de compensation).
- Définir un modèle de reprise permettant de redémarrer l'exécution après une panne (sans avoir annulé les premiers traitements).
- Contrôler et synchroniser l'exécution de la transaction.
- Spécifier les actions à exécuter en cas de conflit ou d'échec.

2.7 Normes et Services Transactionnels existants

Il existe plusieurs standard et normes pour définir les modèles transactionnels. Les standards les plus couramment utilisés sont ceux définis par des consortiums (X/OPEN ou OMG) ou des constructeurs en position de quasi-monopole (Microsoft).

Nous allons donc, dans cette partie, étudier ces différents standards.

2.7.1 Le protocole OSI TP

OSI-TP[14] **[ISO96b]** est un protocole transactionnel conforme au protocole de validation à deux phases dans un système distribué. Ce protocole fournit à la fois des services transactionnels (tels que la validation ou l'annulation de transaction), mais aussi un protocole de communication utilisé pour véhiculer les ordres transactionnels au travers du réseau.

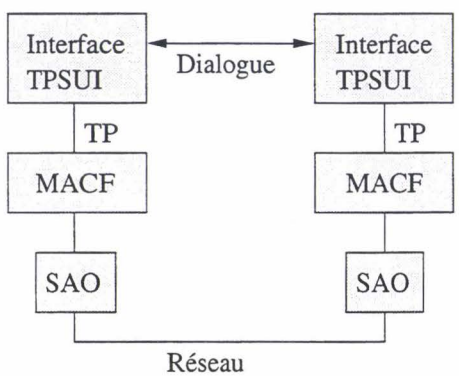


FIG. I.2.5 - *Le modèle OSI-TP*

14. OSI-TP : Open System Interconnection Transaction Processing

Le MACF[15] est le noyau de la machine OSI-TP, qui offre différents services aux utilisateurs TPSUI[16] (ces services sont préfixés de TP : par exemple TP-BeginTransaction). Il coordonne aussi les interactions avec les subordonnées (SAO[17]) qui encodent et décodent les informations transmises entre deux MACF.

Nous ne décrierons pas plus ici les services OSI-TP. Cependant, il est à noter que ce protocole a deux grands avantages :

- Il est standardisé : ce système permet à des systèmes hétérogènes d'interopérer pour le bon déroulement de la transaction distribuée,
- Il est ouvert : le composant clé est le MACF, mais le TPSUI et le SAO peuvent être considérés comme des interfaces avec les utilisateurs et le réseau, qui cachent le MACF, et le détachent de l'environnement d'exécution.

2.7.2 Le modèle DTP de X/OPEN

Le modèle DTP[18] de X/OPEN[19] est un standard d'interfaces entre composants transactionnels [**XOpen96**].

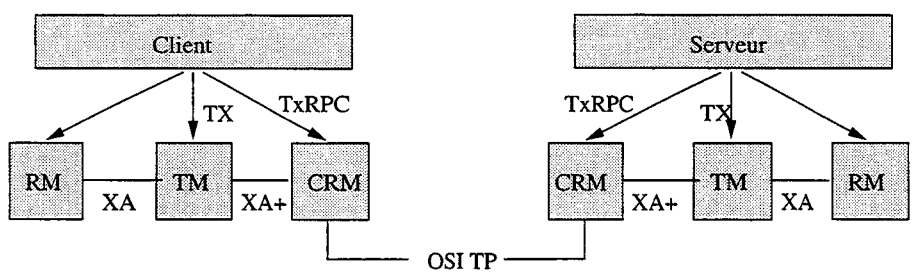


FIG. I.2.6 - *Le modèle DTP de X/OPEN*

Les différents composants du modèle DTP (cf. figure I.2.6) sont :

- l'AP(Application Program) qui est l'application qui définit les actions de la transaction
- Le RM(Ressource Manager) qui est le gestionnaire de ressources de l'environnement de l'application. Il gère l'accès aux données.
- Le TM(Transaction Manager) qui est le gestionnaire de la transaction. Il assure les services de coordination pour les transactions (gestion des identifiants, validation, etc.)

15. Multiple Association Control Function
16. TPSUI : Transaction Processing Service User Invocation
17. SAO : Single Association Object
18. DTP : Distributed Transaction Processing
19. X/OPEN est un consortium d'industriels qui a défini des standards pour les transactions distribuées

- Le CRM(Communication Ressource Manager) qui gère les communications entre des applications distribuées sur plusieurs domaines (c'est-à-dire dépendant de plusieurs gestionnaires de transaction).

Les services de communication utilisés par les applications distribuées sont ceux définis par OSI-TP.

2.7.3 OTS de l'OMG

L'OMG a défini un service Transactionnel (OTS[20])[**OMG97c**] pour les applications distribuées basées sur une architecture CORBA.

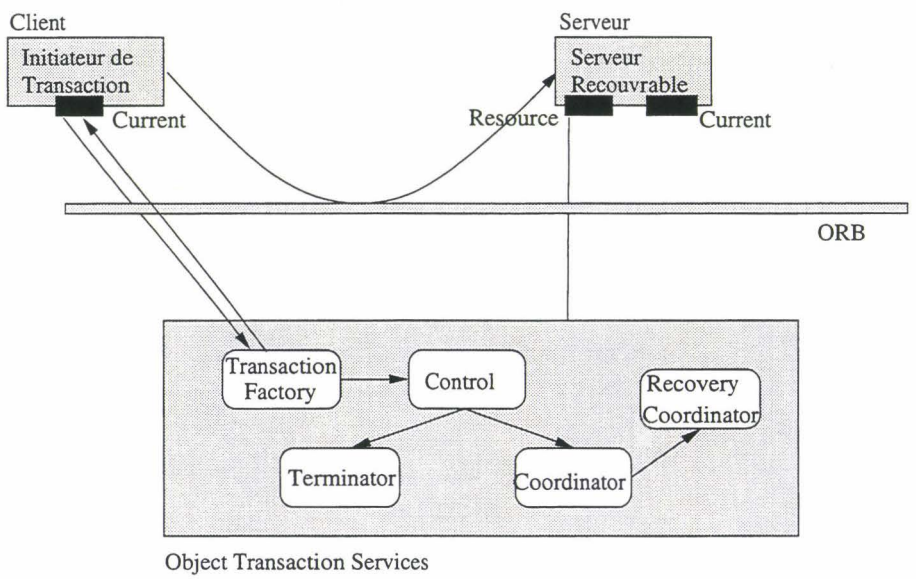


FIG. I.2.7 - *Le système de transaction OTS*

Le principe de fonctionnement est le suivant (cf. figure I.2.7) :

- Le client, qui est l'initiateur de la transaction, s'enregistre auprès de l'OTS, et crée une nouvelle transaction. Pour cela, il a deux solutions. Il peut utiliser l'interface *current* (il utilise alors un mode appelé *indirect*, qui rend transparent la gestion du contexte transactionnel), ou bien il utilise directement l'objet *TransactionFactory* de l'OTS (cette méthode est dite *directe*), qui lui renvoie un identifiant de transaction.

- Lorsque le serveur reçoit une requête de la part du client, il reçoit simultanément le contexte de la transaction (soit de manière *explicite*, c'est-à-dire, en tant que paramètre de la requête, soit de manière implicite, c'est-à-dire véhiculé par l'ORB). Le serveur va alors s'enregistrer auprès de l'OTS.

20. OTS : Object Transaction Services

- A la fin de la transaction, le client demande la validation (ou l'annulation), et l'OTS exécute alors un algorithme de validation à deux phases avec les serveurs.

Plusieurs OTS peuvent collaborer sur plusieurs ORB, via une technique appelée technique d'interposition [**L97**]. Les différentes interfaces de l'OTS peuvent être trouvées en annexe à ce document (Annexe V.1).

2.7.4 MTS de Microsoft

Le service transactionnel MTS[21] est basé sur une architecture 3-tiers [**Mic97**].

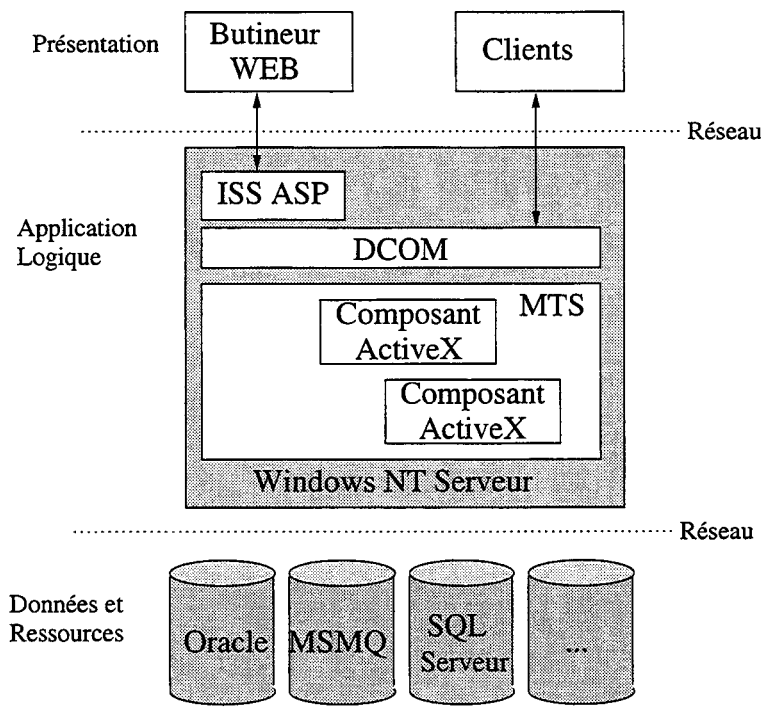


FIG. I.2.8 - *MTS et Les Architectures 3-Tiers*

Dans les architectures 3-tiers, le client, les traitements et les données sont séparés en trois composants qui peuvent être distants sur le réseau (cf. figure I.2.8) :

- Le composant de présentation d'interface, qui permet l'interaction avec l'utilisateur, et qui envoie des requêtes vers les services applicatifs (il peut être comparé à un client dans une architecture client / serveur),
- Le composant applicatif, qui effectue les opérations logiques, ainsi que tous les traitements, et qui envoie des requêtes vers les bases de données,
- Les serveurs de données, qui répondent aux requêtes du composant applicatif.

21. MTS : Microsoft Transaction Server

Le service transactionnel de Microsoft est basé sur les architectures COM[22] et DCOM[23]. Ces architectures peuvent être comparées à CORBA, par les services qu'elles offrent.

Dans le cadre d'une application utilisant MTS, le composant applicatif s'exécute sous le contrôle du serveur transactionnel. Les clients qui invoquent ces services via DCOM, peuvent être soit des applications classiques, soit des scripts ASP[24] s'exécutant au travers d'un serveur Internet IIS[25] [**GLJ97**].

Le principal avantage de MTS est qu'il est une solution adaptée aux mécanismes développés par Microsoft au sein de ses systèmes d'exploitation.

2.7.5 Interopérabilité des Moniteurs Transactionnels

Les moniteurs transactionnels sont utilisés pour assurer aux exécutions les propriétés transactionnelles, et pour garantir une meilleure gestion des ressources du système. Actuellement, la plupart des moniteurs transactionnels répartis se rapproche du modèle DTP (Tuxedo, Encima, etc...).

Cependant, il est à prévoir que des applications désirant accéder à ces bases de données vont être développées sur des bus CORBA. Les gérants de transactions à objets se rapproche donc du service OTS. Il se pose alors le problème d'une transaction essayant d'accéder à un service sur lequel n'a pas été implanté l'interface ressource de OTS (cf. figure I.2.9)

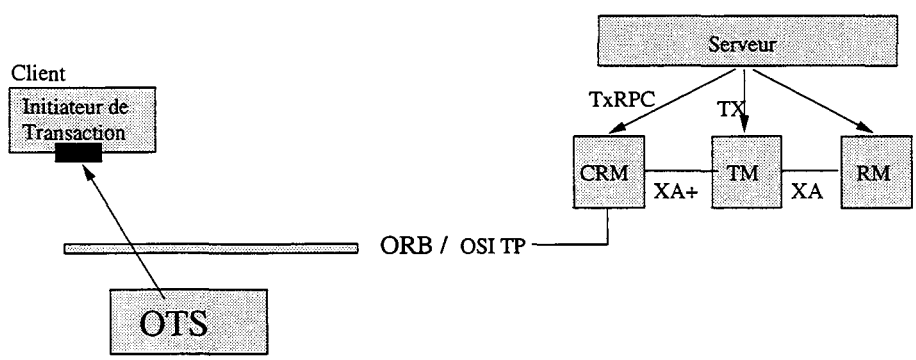


FIG. I.2.9 - *Interopérabilité OTS - X/OPEN*

Des recherches sont en cours pour définir des noyaux transactionnels adaptables, offrant à la fois les interfaces de X/OPEN et de l'OMG [**LS98**].

2.8 Conclusion

Dans ce chapitre, nous avons tout d'abord présenté les propriétés du modèle transactionnel (nommées propriétés ACID), ainsi que les mécanismes nécessaires à

22. COM : Component Object Model
23. DCOM : Distributed Componant Object Model
24. ASP : Active Server Page
25. IIS : Microsoft Internet Information Server

leur implantation. Ces mécanismes sont au nombre de deux :

- Le mécanisme de reprise sur panne assure l'Atomicité et une partie de la Durabilité. Nous avons étudié plusieurs procédés de reprise sur panne (Pages ombres et Journalisation), en faisant ressortir les avantages et inconvénients de chacune des techniques,

- Le mécanisme de contrôle de concurrence assure la propriété d'Isolation des actions entre les transactions. Comme pour la reprise sur panne, nous avons étudié plusieurs stratégies de contrôle de concurrence.

Nous reviendrons sur les mécanismes utiles aux cartes à microprocesseur lors de leur intégration dans les systèmes cartes.

Après cela, nous avons étudié l'influence sur ces mécanismes, de la distribution sur plusieurs sites de l'exécution d'une transaction. La principale caractéristique d'une transaction distribuée, est la nécessité d'implanter (en plus des mécanismes décrit précédemment), un protocole de validation distribuée, qui assure que la même décision, d'abandon ou de validation, soit prise par tous les sites.

Le modèle transactionnel, dans le cadre du déploiement d'applications distribuées, facilite largement la tâche du programmeur. Un autre argument en faveur du modèle transactionnel pour le contrôle de l'exécution d'applications distribuées, est qu'il est supporté par des outils standardisés comme les moniteurs transactionnels.

Nous allons donc maintenant étudier les besoins des cartes à microprocesseur en terme d'exécution transactionnelle, pour ensuite examiner les possibilités d'implantation des mécanismes décrits dans ce chapitre dans les cartes (tant au niveau du système d'exploitation carte, qu'au niveau de l'intégration de ces cartes dans des transactions distribuées).

Deuxième partie

Problématique

Chapitre II.1

Nouvelles Applications et Nouveaux Besoins Carte

1.1 Introduction

Comme nous l'avons vu dans la présentation des cartes à microprocesseur, la programmation d'applications pour carte devient de plus en plus facile. Cela ouvre de nouvelles perspectives au monde des cartes à microprocesseur.

En effet, la durée séparant la décision de réalisation d'une application carte et la commercialisation de cette application[1], va être réduite à quelques semaines (contre quelques mois à l'heure actuelle).

De plus, des secteurs comme la téléphonie mobile demandent de nouveaux services, afin d'offrir une valeur ajoutée, dans un contexte tarifaire équilibré. Le choix du client se fait donc sur la qualité et la diversité des services fournis. Le nombre d'applications portées sur les cartes devraient donc considérablement augmenter dans les années à venir.

Dans ce chapitre, nous allons tout d'abord donner quelques définitions de termes que nous allons ensuite utiliser couramment.

Après cela, nous étudierons les utilisations futures des cartes à microprocesseur, notamment les applications mettant en jeu plusieurs cartes, ou des applications qui se prolongent sur plusieurs connexions.

Ensuite, nous verrons les nouvelles applications des cartes à microprocesseur dans

1. Cette durée est appelée le «Time To Market»

les systèmes distribués (tels que l'Internet) et les problèmes de sécurité inhérents à de telles applications.

A ces problèmes (propres aux applications), s'ajoutent ceux propres aux cartes. Nous étudierons donc les différentes pannes liées à l'utilisation des cartes à microprocesseur dans les applications, pour en déduire les besoins des futurs systèmes cartes (de manière à pallier à la fois aux problèmes cartes et aux problèmes applicatifs).

1.2 Quelques définitions

Nous allons ici donner un certain nombre de définitions qui permettront de mieux fixer le sens exact de plusieurs termes qui vont être beaucoup utilisés, à la fois dans ce chapitre, mais aussi dans les suivants.

1.2.1 Applications et services

Par le terme *Application*, nous entendons l'exécution de un ou plusieurs services[2], permettant la transformation de données d'un état initial A à un état final B. Ces services pourront être tous situés sur le même site (on parlera alors d'*Application Centralisée*) ou au contraire répartis sur plusieurs sites (on parlera alors d'*Application Distribuée*). Un service carte, est un objet (au sens CORBA) carte, qui définit des données stockées dans la carte (les variables d'instance de l'objet), et des fonctions (les méthodes de l'objet), qui exécute des instructions sur ces objets, ou qui renvoie un résultat à l'utilisateur [**V95**].

Le terme *Application longue*, signifie que les différents services s'exécutent de manière espacée dans le temps, avec déconnexion possible de la carte de son terminal. Le terme *Application Persistante* peut aussi être utilisé à cette fin.

1.2.2 Contexte d'exécution

Le *Contexte d'exécution* d'un service contient l'ensemble des informations nécessaires à l'exécution de ce service. C'est-à-dire, l'identifiant de l'application (ou de la transaction) qui a fait appel à ce service, les droits d'accès à ce service, et bien sûr, les variables systèmes, comme la pile d'exécution.

Le *Contexte Applicatif* est en fait l'état des données utilisées par le service (par exemple le solde d'un compte). Ces données ne doivent pas être perdues en cas de changement de contexte d'exécution.

Par *Système d'Exploitation Multi-Contexte*, on entend un système d'exploitation capable de gérer plusieurs contextes d'exécution.

1.3 Applications futures

Notre propos dans cette partie est de bien identifier les différents rôles que pourra prendre la carte dans le futur. En effet, le développement des capacités de mémoire

2. De manière plus détaillée, on considère qu'une application est consituée de plusieurs services, eux mêmes constitués de plusieurs méthodes, elles mêmes constituées de plusieurs instructions.

et de traitement des cartes et des systèmes informatiques courants a plusieurs conséquences :

- Le nombre de cartes à microprocesseur en possession de chaque utilisateur est en constante augmentation (carte téléphone, carte bancaire, carte santé, PME, fidélité, etc...),
- De plus en plus d'utilisateurs disposent sur eux d'un terminal souvent [3] connecté sur un réseau : leur téléphone GSM,
- Les systèmes de paiement électronique sont de plus en plus nombreux (recharge de carte prépayée pour GSM, paiement sur Internet, PME ou carte bancaire chez les commerçants).

La diversité et le nombre croissant d'applications pour carte à microprocesseur vont avoir plusieurs conséquences. Tout d'abord, il faut chercher à réduire le nombre de cartes en possession d'un porteur, tout en permettant aux services dont il dispose d'interopérer (il s'agit de la problématique des cartes multi-services que nous allons étudier dans la première partie de cette section).

De plus, le nombre de cartes augmentant, on voit apparaître un nouveau type d'application distribuée : les applications mettant en jeu plusieurs cartes (ou applications multi-cartes). Nous étudierons la problématique de ces applications dans la seconde partie de cette section.

Enfin, en partant de la constatation qu'une carte à microprocesseur est plus souvent dans la poche de son porteur qu'en cours d'exécution d'une application, nous étudierons les applications dont l'exécution se déroule sur plusieurs connexions de la carte.

1.3.1 Les cartes multi-services

Nous avons déjà beaucoup parlé de cartes multi-services depuis le début de ce mémoire. Cependant, les réalisations faites jusqu'à ce jour concernaient uniquement des cartes contenant plusieurs applications, mais toutes parfaitement isolées les unes des autres.

Il s'agit ici de définir des modèles d'exécution permettant la coopération entre plusieurs services installés sur la même carte à microprocesseur. Cette coopération peut se faire de deux manières :

- Par communication *Externe* : La carte est alors reliée à un terminal qui fait d'abord appel à un service, puis à un autre (cf. figure II.1.2)
- Par communication *Interne* : Un service carte fait appel à un autre service carte. Il se pose alors des problèmes de communication entre services (c'est à la carte de gérer ce type de problèmes).

Nous allons voir dans cette partie des exemples de cartes multi-services, ainsi que leurs principes de fonctionnement. Nous terminerons par les problèmes encore en suspens dans ce type d'application carte.

3. Mais pouvant être déconnecté de manière impromptue, suite à la sortie de la zone de couverture

1.3.1.1 Des exemples d'applications

De manière à mieux illustrer notre propos dans cette section, nous allons commencer par étudier quelles pourraient être les applications mettant en jeu plusieurs services d'une même carte à microprocesseur.

Parmi les services les plus souvent installés dans les cartes à microprocesseur, on trouve le PME et l'application GSM (Identification, agenda, etc.). Un premier exemple d'application utilisant simultanément plusieurs services d'une même carte est le paiement d'un achat grâce à un PME, pendant une communication GSM (cette communication pouvant être utilisée comme connexion à un réseau sans fil) (cf. figure II.1.1).

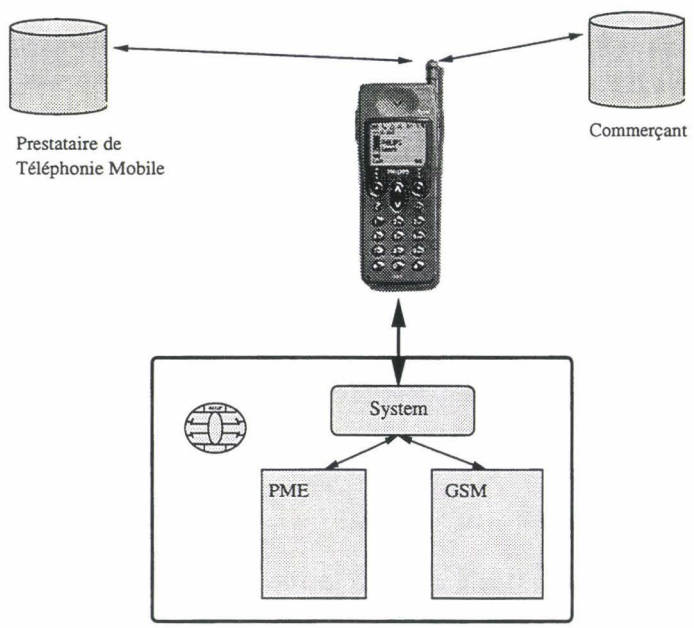


FIG. II.1.1 - *Exemple d'application utilisant plusieurs services carte*

Dans cet exemple, le GSM sert en fait de terminal de paiement pour le commerçant. Le porteur est authentifié auprès de son prestataire de téléphonie et auprès du commerçant grâce à l'application GSM de sa carte à microprocesseur.

Une autre application peut consister dans le rechargement du PME grâce au service «Carte Bancaire» installé sur la même carte (sans que les deux appartiennent nécessairement à la même banque).

Comme on peut le voir, il est facile d'imaginer des applications mettant en jeu plusieurs services d'une seule et même carte (carte santé permettant de payer sa consultation, application de recharge d'un crédit téléphonique à l'aide d'un PME, etc...). Pour que cela soit possible, nous allons maintenant étudier le principe de la coopération de services carte entre eux.

1.3.1.2 Coopération Externe

Le modèle d'application décrit par la figure II.1.1 a de nombreuses conséquences sur le modèle d'exécution des services internes à la carte.

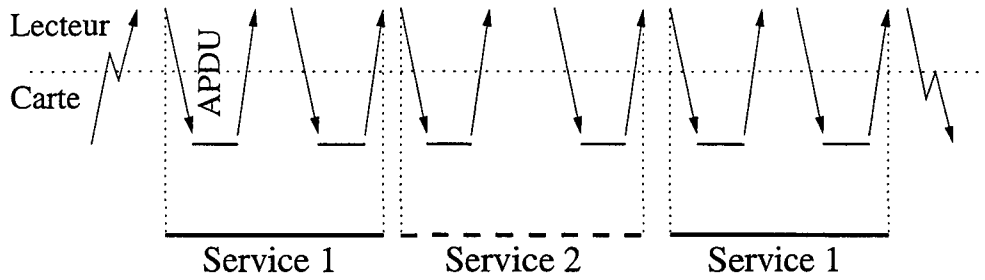


FIG. II.1.2 - *Principe d'utilisation de plusieurs services*

Dans le cadre d'une application utilisant plusieurs services d'une carte à microprocesseur, l'échange carte / terminal sera constitué de plusieurs sessions (cf. figure II.1.2).

En effet, si on reprend l'exemple précédent, à l'ouverture de la communication, la carte va échanger des informations avec le prestataire de téléphonie mobile (Service 1) pour que l'utilisateur se connecte au réseau sans fil. Puis, l'utilisateur va désirer faire un achat grâce à son PME. Alors, la carte effectue des échanges avec le commerçant (Service 2) pour assurer le paiement de l'achat. Pendant le temps de l'achat, le prestataire de téléphonie mobile doit pouvoir intervenir sur la carte (maintien de la connexion, dépassement de temps de communication, etc...). Enfin, après l'achat, le service de téléphonie reprend seul la main.

1.3.1.3 Coopération Interne

Les cartes multi-services existantes ont actuellement des services complètement cloisonnés (les services ne peuvent pas échanger de données ou partager des objets entre eux).

Cependant, la présence de plusieurs services dans une même carte peut inciter à briser cet isolement, pour les faire coopérer. Prenons l'exemple de la figure II.1.3. Dans ce cas, la carte reçoit un ordre APDU de son terminal demandant l'exécution de l'application 1. Le système de la carte lance donc le service correspondant à cette application. L'invocation une méthode de l'objet 2 de cette application 1, provoque l'appel à une application partagée (appelée Application 2). Le but de ce partage d'objets est à la fois d'éviter des duplications de programme à l'intérieur de la carte à microprocesseur, mais aussi de faciliter la mise en commun de données.

Un exemple de partage d'objet entre services est celui des applications de fidélité. De plus en plus de grands groupes s'associent pour distribuer des cartes à microprocesseur de fidélité. Plusieurs prestataires peuvent passer un accord pour grouper leur offre de points de fidélité. Le compteur additionnant ces points doit donc être partagé entre toutes les applications des différents prestataires.

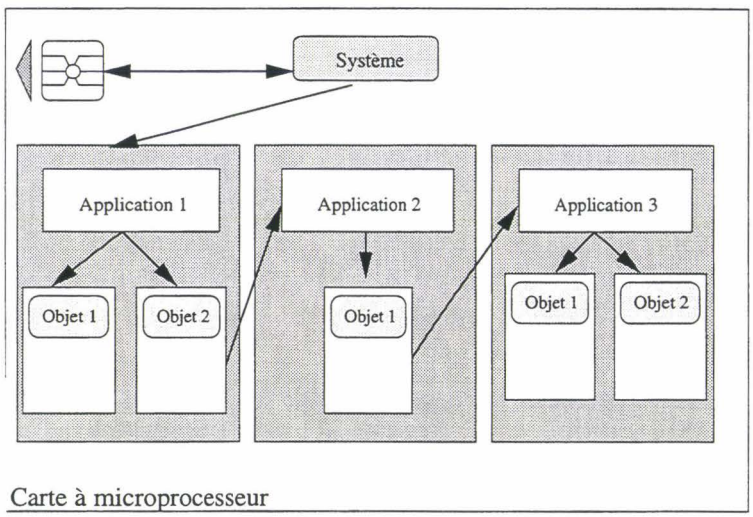


FIG. II.1.3 - *Exemple de partage de données entre applications*

Un tel partage pose des problèmes de droit d'accès aux services : dans la figure II.1.3, le service 2 peut donner le droit au service 1 de l'utiliser, sans que le service 1 ai des droits sur le service 3 (la confiance n'est pas transitive : le service 3 peut faire confiance au service 2, le service 2 au service 1, et le service 3 peut être en désaccord avec le service 1). L'exécution d'une application 1 telle qu'elle est définie dans notre schéma pose donc problème.

Une autre manière de limiter la duplication de code à l'intérieur de la carte à microprocesseur est l'utilisation de bibliothèques, qui viennent enrichir le langage de programmation de la carte à microprocesseur.

Le principal avantage d'une utilisation de bibliothèque, par rapport au partage d'objet, est de limiter la duplication de code, sans pour autant augmenter les problèmes d'accès concurrents à un objet. De plus, la gestion des droits d'accès se trouve facilitée.

Cependant, une telle coopération ne permet pas le partage de données. Par exemple, la mise en commun d'un PME n'est pas possible avec la seule utilisation de bibliothèques.

Dans ce mémoire, nous ne traiterons pas plus en avant les problèmes de sécurité liés aux communications entre des services carte, et nous nous concentrerons sur la définition d'une carte capable de gérer des transactions utilisants plusieurs services. Pour cela, nous considérerons que les problèmes de sécurité liés au partage des objets dans la carte sont résolus (par des accords entre les différents prestataires qui ont installé les services).

1.3.1.4 Problèmes posés

Les nouvelles applications basées sur les cartes multi-services posent de nombreux problèmes.

L'utilisation «simultanée» de plusieurs services dans les cartes à microprocesseur

oblige à effectuer des changements de contexte dans la carte. En effet, la carte à microprocesseur actuelle est mono-tâche : un seul service s'exécute à la fois. En outre, de par la taille de la RAM disponible dans les cartes à microprocesseur, un changement de contexte d'exécution oblige un vidage complet («Flush» en anglais) de la mémoire volatile sur le support non volatile.

Cependant, la gestion de plusieurs contextes applicatifs dans la carte peut s'avérer insuffisante. En effet, l'exécution de plusieurs services dans la carte, pouvant en plus partager des données, peut poser des problèmes d'accès concurrents à ces données. Dans l'exemple proposé (figure II.1.2), les deux prestataires (le commerçant et celui de la téléphonie) accèdent à des services *différents* de la carte à microprocesseur. Cependant, il se peut aussi qu'ils essayent d'accéder au même service (par exemple le PME), ce qui pose des problèmes de contrôle des accès concurrents aux données de ce service.

Le dernier point problématique concerne le partage d'objet entre deux services installés dans une même carte. En effet, le contrôle sur la concurrence des accès à un objet partagé est beaucoup plus problématique, et peut être rapproché des problèmes de contrôle de concurrence des transactions emboîtées vues dans le chapitre précédent (le service appelé par le terminal, peut être vu comme la transaction mère, et l'appel par ce service à des méthodes d'un autre service, comme l'exécution d'une sous transaction).

1.3.2 Les Applications «longues»

Les connexions des cartes à microprocesseur sur un terminal (comme un DAB[4], cabine téléphonique) sont intermittentes et éphémères. En effet, elles sont intermittentes car une carte se retrouve fréquemment dans la poche de son porteur. Et elles sont éphémères, car un utilisateur d'une carte n'acceptera pas de rester très longtemps devant son terminal.

Ces caractéristiques de la connexion d'une carte restent vraies pour une carte dans un terminal GSM. En effet, dans ce cas là, la carte est connectée en permanence dans son terminal, mais le terminal peut se retrouver déconnecté du réseau à tout moment.

A l'opposé de ces connexions cartes brèves, certaines applications peuvent durer dans le temps, et donc nécessiter plusieurs connexions de la carte. Dans cette partie, nous allons tout d'abord voir un exemple d'application longue, et ensuite étudier les problèmes que posent de telles applications.

1.3.2.1 Des exemples d'applications longues

Un exemple d'application longue mettant en jeu une carte à microprocesseur est la réservation de billets (spectacles, transport, etc...) au moyen d'un PME.

De manière plus détaillée, prenons l'exemple d'un voyage comprenant un billet d'avion, une location de voiture et une réservation d'hôtel.

4. DAB : Distributeur Automatique de Billets

Cette application va se faire en 4 connexions :

- Dans un premier temps, le porteur de la carte va contacter la compagnie aérienne pour réserver un billet d'avion. Nous allons partir du principe que le porteur dispose d'une carte fidélité et qu'il paye avec son PME.

- Après cela (peu importe le temps écoulé entre les deux transactions), le porteur doit réserver sa voiture de location (toujours avec son PME et sa carte de fidélité).

- Puis, le porteur va réserver sa chambre d'hôtel. Ici, nous allons considérer qu'il n'utilise pas son PME mais sa carte bancaire

- Enfin, pour terminer, l'utilisateur va réellement effectuer son trajet et donc les montants vont réellement être débités de ses différents comptes, et les points fidélité attribués.

Dans cet exemple, l'application globale (que l'on va appeler «Voyage») est persistante durant la déconnexion de la carte à microprocesseur. En effet, la carte doit garder des traces des achats effectués avec le PME, car ils ne seront validés que lors d'une connexion ultérieure.

1.3.2.2 Problèmes posés

L'exécution d'une même application sur plusieurs connexions de la carte implique un certain nombre de problèmes :

- **Persistance du contexte :** la carte à microprocesseur doit être en mesure de retrouver le contexte d'exécution de l'application pour que celle-ci se poursuive. Ce qui implique un contexte d'application persistant (donc sauvegardé en mémoire non volatile).

- **Blocage données des applications :** la carte ne peut pas être bloquée sur une application en cours d'exécution (surtout si celle-ci se poursuit sur plusieurs connexions). L'exécution de telles applications peut donc poser des problèmes d'accès concurrents à des données, car le porteur de la carte peut l'utiliser à d'autres fins que l'application longue en cours d'exécution (on peut prendre l'exemple d'un PME, qui doit rester disponible, même si un paiement est en cours sur plusieurs connexions, pour par exemple réserver un voyage).

Les problèmes posés par l'exécution d'une application sur plusieurs connexions de la carte peuvent être rapprochés de la problématique de transactions de longue durée (cf. le chapitre précédent). En effet, la carte doit être en mesure de participer à plusieurs transactions, dont certaines peuvent être en attente. Cela implique donc un mécanisme de reprise sur panne permettant la gestion de plusieurs contextes transactionnels, et un mécanisme de contrôle de concurrence sur les données pouvant être accédées par plusieurs transactions.

1.3.3 Les applications multi-cartes

En même temps que les applications liées aux cartes multi-services, sont en train d'apparaître les applications mettant en jeu plusieurs cartes.

Dans le cadre de ces nouvelles applications, nous pouvons différencier deux cas :

- 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-services que nous avons vu précédemment. 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 (qui sont connectées sur le même terminal).

- 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. Il peuvent disposer d'un certain laps de temps pour effectuer cette opération (par exemple, un vote électronique sur plusieurs jours). Les participants seront connectés sur des terminaux distincts. Il se pose alors notamment des problèmes d'établissement de la connexion.

Les problèmes posés par ce type d'application sont en fait à l'intersection des problèmes posés par les cartes multi-services et par les applications persistantes :

- **Le besoin de coopération entre applications** : Comment des services situés sur des cartes différentes peuvent coopérer ? Il faut pouvoir faire circuler les données de manière sécurisée, et gérer les droits d'accès aux différents services.

- **Maintien en cohérence de la carte** : Le second problème est celui de la mise à jour de la carte (et est donc lié à la problématique des applications persistantes). Le résultat de l'application dépend du résultat des différents services exécutés. Or, dans le cas où l'application doit contacter plusieurs cartes, le porteur d'une des cartes de l'application aura déconnecté sa carte avant la terminaison de l'application. Comment le porteur de la carte peut retrouver le résultat de l'application ? Dans quel état se trouvent les données manipulées par l'application non terminée?

1.4 Cartes et Applications Distribuées

Le développement du commerce électronique pose de nombreux problèmes, notamment la sécurisation des paiements. Dans cette section, nous commencerons donc par examiner les problèmes de sécurité dans les systèmes distribués (Internet, GSM, etc...) pour ensuite, étudier en quoi la carte peut aider à résoudre ces problèmes.

Après cela, à partir des contraintes technologiques cartes, nous verrons comment améliorer leur intégration dans les systèmes distribués, pour que la carte à microprocesseur soit en mesure de répondre au manque de sécurité dans les applications actuelles.

1.4.1 Sécurité et systèmes distribués

Il convient tout d'abord de définir ce que l'on entend par «sécurité». Généralement, le terme sécurité regroupe 4 principes de bases [**ECC94**] :

- L'identification : Il s'agit pour le système de s'assurer de l'identité d'une entité (par exemple, le nom utilisateur sous Unix).
- L'authentification : Le système doit s'assurer de l'authenticité d'une information (par exemple, sous Unix, le mot de passe authentifie le nom utilisateur).
- La confidentialité des données utilisées, mais aussi des traitements et des communications. Le système doit s'assurer que des entités non autorisées n'auront pas accès aux informations et traitements circulants sur le réseau.
- L'intégrité : Le système doit s'assurer que les informations n'ont pas été modifiées lors du traitement qu'elles ont subies (transport, stockage ou exécution).

A ces principes de bases s'ajoutent souvent le contrôle d'accès, la disponibilité des informations, la non-répudiation des actions (il est impossible de nier ce que l'on a reçu ou fait), et enfin, l'audit du système de sécurité.

Les fonctions de sécurité telles que l'authentification, la confidentialité ou l'intégrité sont basées sur des algorithmes de cryptographie à clé secrète (comme le DES[5] [**NBS77**]), ou à clé publique (comme le RSA[6][**RSA78**]). Tous les mécanismes de cryptographie sont décrits en détail dans [**Sch96**].

A partir de ces explications, il apparaît que les problèmes de sécurité dans les systèmes distribués sont très diversifiés :

- Dans les entreprises : Les entreprises sont de plus en plus dépendantes de leurs systèmes d'informations, car toutes leurs données s'y trouvent stockées. Parallèlement, beaucoup de sociétés ouvrent leur réseau vers l'extérieur, soit pour échanger des données entre des sites distants, soit tout simplement pour ouvrir un site Internet. Il se pose alors à la fois des problèmes de confidentialité, d'intégrité et de disponibilité des informations.
- Pour les personnes : De plus en plus d'informations confidentielles et personnelles sont distribuées sur le réseau. Il se pose alors inévitablement des problèmes de confidentialité, qui sont semblables à ceux des entreprises.
- Pour le commerce électronique : Le commerce électronique est appelé à se développer de plus en plus, notamment grâce à l'Internet. On doit alors être en mesure d'assurer à la fois l'intégrité des données manipulées, la non-répudiation des actions ainsi que l'authentification de tous les partenaires.

De plus, par définition, les systèmes distribués sont plus vulnérables que les systèmes centralisés, car le nombre de points succeptibles d'être attaqués est plus important. En plus de la faiblesse des machines (en terme de sécurité), on ajoute la faiblesse des réseaux rendus nécessaires par la répartition.

5. DES : Data Encrytion Standard
6. RSA : Rivest, Shamir, et Adelman, d'après le nom des inventeurs

On peut considérer que ce que définit le niveau de sécurité d'un domaine[7], est l'élément le moins sécurisé de ce domaine. Prenons l'exemple d'un achat sur Internet (cf. figure II.1.4).

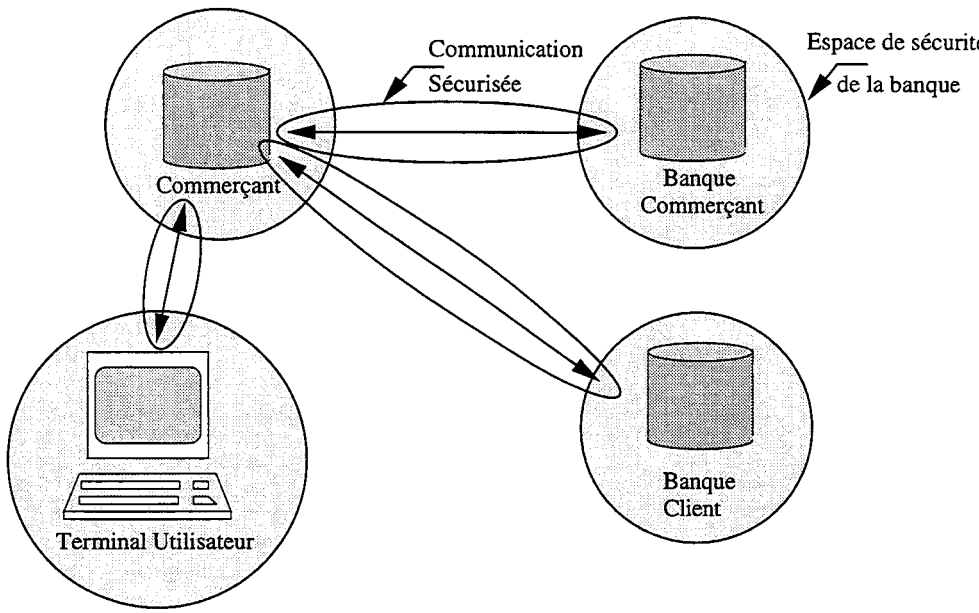


FIG. II.1.4 - *Espace à sécuriser*

Dans cet exemple, il apparaît que non seulement les communications entre les différents sites doivent être sécurisées, mais aussi que chaque site doit avoir un espace sécurisé propre.

Il est raisonnable de penser que les banques disposent d'un système d'information robuste aux attaques, et qui peut donc être de confiance. Il n'en n'est cependant pas de même pour le commerçant (qui peut soit avoir été attaqué, soit être malhonnête), ou pour le terminal utilisateur (qui peut avoir été attaqué par un utilisateur ou par un pirate).

En effet, dans cet exemple, nous partons du principe que l'utilisateur est connecté sur un terminal générique qu'il ne connaît pas (par exemple, dans un CyberCafé, ou d'une cabine téléphonique).

Des solutions commencent à voir le jour pour sécuriser les transferts sur Internet (on peut notamment citer SSL[8] de Netscape **[ID96]**. SSL permet uniquement de sécuriser le transfert. Même avec des mécanismes de certification basés sur X509 **[ISO94b]** (qui permettent l'authentification), ce système ne permet pas de s'assurer que le commerçant ou le terminal utilisateur n'a pas été piraté. En effet, on ne peut pas être assuré que l'espace de sécurité du commerçant est fiable (qu'il soit honnête ou non), et cela est encore plus vrai pour le terminal générique sur lequel se connecte l'utilisateur.

7. On appelle *domaine* d'une application l'ensemble des composants permettant l'exécution d'une application

8. SSL : Secure Sockets Layer

Pour les systèmes distribués utilisant un bus de communication entre objets CORBA, l'OMG a défini un service de sécurité [**OMG97d**]. Ce système permet lui aussi l'authentification des participants et la confidentialité des communications. Cependant, il ne permet pas d'assurer l'intégrité des services participants à l'application distribuée, et notamment du terminal sur lequel se connecte l'utilisateur.

1.4.2 Intégration des cartes dans les systèmes distribués

Comme nous l'avons vu dans la première partie de ce document, la carte à microprocesseur commence à être intégrée dans les systèmes distribués. Elle y est utilisée aussi bien comme serveur de données (données personnelles du porteur), ou de services (PME par exemple).

Plus récemment, de nouveaux projets visent à utiliser la carte comme une porteuse des informations nécessaires à la connexion d'un utilisateur sur le réseau (identification et authentification), et des certificats de l'utilisateur. On peut notamment noter le produit de la société ActivCard [**EN98**] qui propose un ensemble carte / lecteur pour sécuriser les accès à un site Internet.

Ces nouveaux projets partent de la constatation que les moyens de connexion utilisés actuellement sur les terminaux génériques (à savoir le nom d'utilisateur et le mot de passe), sont totalement insuffisants, car facilement fraudables.

Grâce à l'utilisation des cartes à microprocesseur comme outil de connexion pour les terminaux sur les architectures distribuées, les partenaires du client (c'est-à-dire les banques et le commerçant dans notre exemple), sont maintenant certains de l'identité et de l'authenticité de l'utilisateur.

Cependant, l'utilisation des cartes à microprocesseur dans les systèmes distribués, telle qu'elle est faite actuellement, est insuffisante. En effet, la carte est maintenant suffisamment évoluée pour tenter de pallier au manque de sécurité du terminal générique, et au manque de confiance dans le commerçant.

1.4.3 Sécurisation d'applications distribuées par une carte

Une application distribuée comme le paiement d'achats sur Internet peut se modéliser par une transaction distribuée. Prenons l'exemple de l'achat d'un produit sur Internet à l'aide d'un porte-monnaie électronique.

Dans le cas d'une telle transaction, il convient tout d'abord d'identifier les participants à la transaction :

- La banque du commerçant : il s'agit d'un site de confiance, bien protégé.

- Le commerçant : contrairement à sa banque, ce type de site (surtout sur Internet) ne peut pas être de confiance. De plus, la plupart d'entre eux sont la cible de pirates informatiques.

- Le terminal de l'utilisateur : il s'agit d'un terminal en accès libre, qui a d'autres rôles que celui de terminal de paiement sur Internet. Il n'est donc pas certain qu'il soit intègre.

- La carte de l'utilisateur : de part sa sécurité intrinsèque, la carte à microprocesseur est de confiance, car elle ne peut pas être piratée. Il suffit alors de prouver qu'elle est utilisée par son propriétaire légal.

Les failles de sécurité se situent donc au niveau du terminal générique, et dans une moindre mesure, au niveau du commerçant. Le but de notre propos dans ce mémoire, est d'utiliser au maximum les capacités des cartes à microprocesseur pour combler ces failles de sécurité.

Comme nous l'avons vu dans la partie 2.5.3, un des éléments clés des transactions distribuées, est le coordinateur de la transaction (qui est responsable de la validation de la transaction). Dans un contexte de sécurité, il est nécessaire que le coordinateur suive scrupuleusement l'algorithme du protocole de validation utilisé, signe (authentification, certification) les ordres qu'il enchaîne pour la réalisation du protocole et enfin, journalise les réponses authentifiées des partenaires (pour prévenir la répudiation d'un ordre).

En effet, le coordinateur pourrait forcer certaines parties de l'application à valider quand d'autres abandonneraient (attaque byzantine), par un non respect du protocole de validation distribuée. 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 utilisant une carte parmi les participants, le site de confiance peut être une machine identifiée du prestataire qui a chargé l'application résidente dans la carte (par exemple, une banque), ou bien la carte elle-même quand la connexion avec le serveur de confiance est coûteuse ou hasardeuse.

Dans l'exemple d'achat sur Internet que nous avons pris, la carte comme coordinatrice de la validation de la transaction est une bonne solution, car une carte monétaire est certifiée par les banques et acceptée par les commerçants, et l'utilisateur ne peut pas la corrompre.

Dans la pratique, la carte coordinatrice de transaction pose de nombreux problèmes tant à cause de la technologie carte, qu'à cause de son intégration dans les systèmes. En effet, comme nous l'avons vu précédemment, être coordinateur d'une transaction est très coûteux en terme de ressource, et en terme de communication. Hors, ce sont justement les deux points faibles des cartes à microprocesseur.

En ce qui concerne les communications, le coordinateur doit être disponible (enregistrement des participants, et différentes phases de validation), et doit émettre des requêtes vers les participants de la transaction (préparation à la validation, ou décision de validation).

1.4.4 D'une Carte Serveur à une Carte Cliente

Jusqu'à maintenant, la carte a toujours été considérée comme un serveur soit de données (pour des cartes comme la CQL, qui contient des tables), soit de code (pour des cartes qui contiennent, par exemple, des Porte Monnaie Electronique). Ce principe était bien adapté au protocole de communication de la carte T=0 : on envoyait une requête au serveur (c'est à dire la carte), elle effectuait un traitement, et on obtenait un résultat en réponse.

Aujourd'hui, les capacités de la carte, tant au niveau matériel, qu'au niveau logiciel, évoluent très rapidement. Nous proposons ici, un exemple d'application ou la carte peut être vue comme le client de l'application distribuée, en s'affranchissant des contraintes du protocole. Nous comparons cette nouvelle vision d'une application carte, à l'architecture «classique» des applications actuelles.

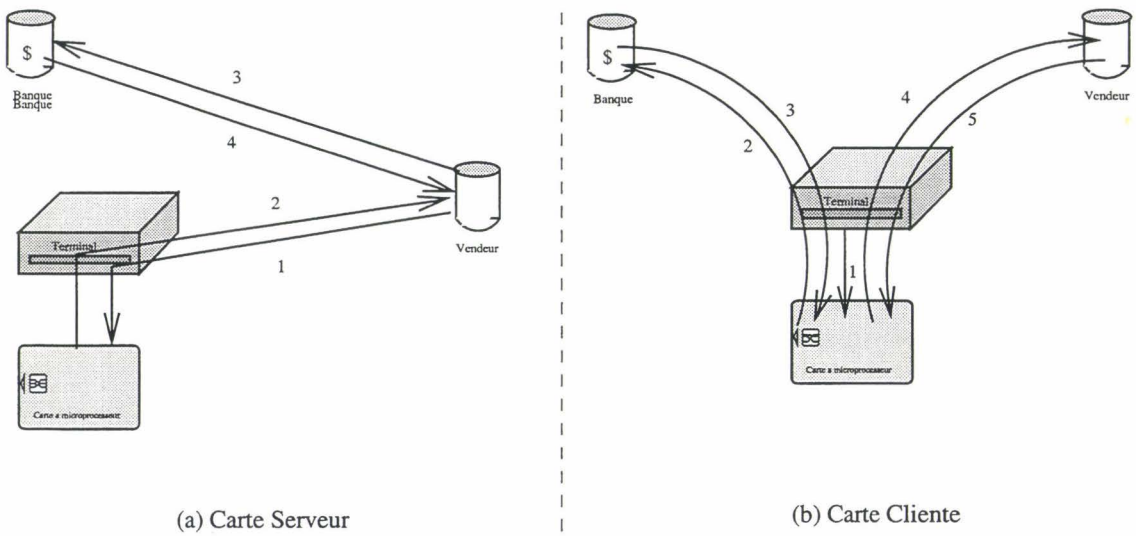


FIG. II.1.5 - *Exécution d'une Application*

Dans la figure II.1.5, nous présentons un exemple d'application, où :

- 1er cas : la carte est serveur (figure (a)). Dans ce cas, la carte est serveur de l'application de paiement du commerçant. Ce dernier commence par contacter la carte pour obtenir un certificat (1), en réponse à cette requête, la carte envoie son certificat (2). Après cela, le commerçant n'a plus qu'à contacter la banque pour demander le débit du compte, en fournissant le certificat de la carte. Dans cette solution, la carte se contente de donner son certificat, sans pouvoir agir contre l'utilisation de ce certificat par le commerçant. Le principal avantage de cette méthode est la simplicité du code embarqué dans la carte, et une réplication limitée au strict minimum du code. Cette organisation des applications est bien adaptée aux cartes actuelles.

- 2nd cas : la carte est cliente (figure (b)) d'une banque, et d'un vendeur. L'intérêt de ce cas de figure, est que toutes les requêtes transitent par la carte. C'est donc bien l'objet qui veut effectuer la transaction, qui dirige le tout : lorsque l'on veut acheter un bien avec sa carte, tout doit passer par la carte. Dans un premier temps, la carte est introduite dans le lecteur, et initialisée par ce dernier (étape 1). En réponse à cette initialisation, la carte envoie une requête vers la banque pour demander le débit du compte (2). En réponse, la banque renvoie un certificat à la carte (3). Après, la carte contacte le commerçant, en lui fournissant le certificat, et en demandant l'achat (4). Pour terminer l'ap-

plication, le commerçant valide la vente à la carte (5). Toutes les informations circulent donc par la carte, qui est l'élément sécurisé du réseau.

La carte ne doit donc plus être considérée uniquement comme un serveur d'une application distribuée (que ce soit de code ou de données). Elle est maintenant suffisamment puissante, en terme de processeur, en terme de mémoire, et en terme de fonctionnalité pour être, si cela est nécessaire, cliente d'une application. Le protocole de communication n'est pas une limite à cette évolution. Une application carte s'appuie sur ce protocole, qui est en fait une couche de transport des requêtes, et qui doit être transparent pour le programmeur (comme le protocole TCP/IP sur un réseau Internet).

1.4.5 Conclusion

La carte à microprocesseur est un apport indéniable de sécurité et de simplicité (notamment en terme de paiement électronique) dans les systèmes distribués. Nous ne parlons pas dans ce mémoire de sécurité au sens cryptographique du terme, car les mécanismes d'authentification et d'autorisation, utilisant les certificats X509, et fonctionnant avec des cartes, existent déjà.

Cependant, en terme de transactions distribuées mettant en jeu des cartes à microprocesseur, la coordination de la transaction ne pourra se faire que par un tiers de confiance (ce qui est proposé actuellement sur Internet), ou directement par l'intermédiaire de la carte (ce qui est préférable).

Il peut être donc nécessaire d'implanter dans une carte participant à des transactions distribuées les fonctions de coordination que proposent les moniteurs transactionnels, ou, à défaut, des mécanismes d'autorisation pour les coordinateurs de validation.

De toute façon, les cartes à microprocesseur participant à des applications distribuées doivent être en mesure de participer à la validation des actions de ces applications. Cette carte doit donc être, au minimum, en mesure de répondre aux ordres d'un coordinateur de validation de transaction distribuée (cf. Chapitre I.2).

1.5 Modèles de panne et carte

Plusieurs types de pannes peuvent survenir lors de l'exécution d'une application, dont au moins un des composants se trouve sur une carte à microprocesseur.

Mais, comme nous l'avons vu dans les chapitres précédents de ce document, la carte est un composant électronique spécifique, tant par ses caractéristiques matérielles que logicielles. Les pannes qui peuvent survenir sont donc caractéristiques de l'utilisation d'une carte à microprocesseur, tant au niveau de leurs conséquences sur l'application, qu'au niveau de leur fréquence.

Nous allons donc, dans cette section, faire une liste non exhaustive des différentes pannes spécifiques aux cartes à microprocesseur, et examiner leurs influences sur le cycle de vie de la carte et sur l'application en cours d'exécution.

1.5.1 L'arrachement de la carte

Le terme d'*arrachement* est utilisé pour représenter l'action qui consiste à retirer la carte de son terminal. Comme nous l'avons dit précédemment, lorsqu'elle n'est plus dans son terminal, la carte à microprocesseur se trouve privée d'alimentation électrique. Ainsi la carte ne peut plus exécuter la moindre action, et le contenu de sa mémoire volatile est perdu. Cet événement est très fortement probable (plus qu'une panne disque dans les SGBD[9])

Lorsque la carte est arrachée à la fin d'une exécution d'application, il n'y a pas de problème, car le système a pu effectuer les sauvegardes de données en mémoire non volatile, et il a renvoyé les mots d'état au terminal. Cependant, il se peut que l'utilisateur arrache sa carte du terminal avant la bonne fin de l'exécution (cf figure II.1.6), soit parce qu'il pense que la transaction est terminée, soit par tentative de fraude, ou bien simplement parce qu'il trouve l'exécution trop longue.

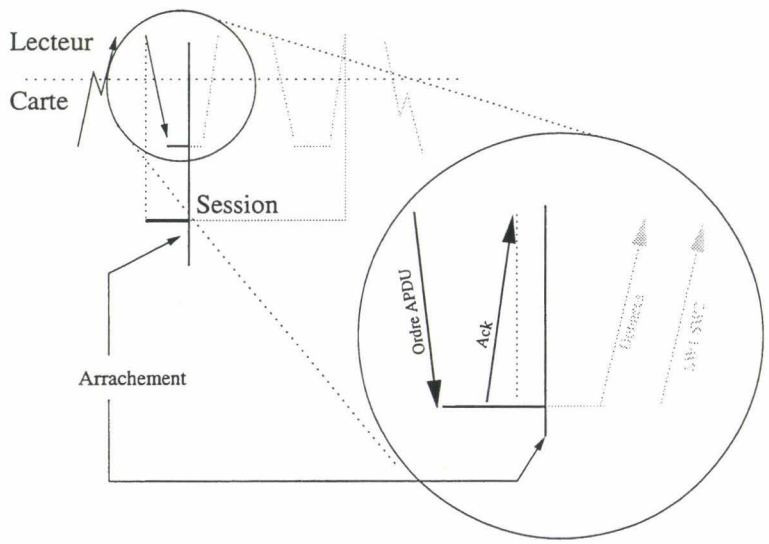


FIG. II.1.6 - *Conséquences d'un arrachement de la carte*

En cas d'arrachement de la carte, le terminal ne recevra pas les mots d'état (SW1 et SW2) de la carte. La carte sera donc considérée comme absente, et l'échange APDU non traité (une erreur est renvoyée à l'application).

Le problème est plus complexe au niveau de la carte à microprocesseur. En effet, celle-ci se trouve subitement privée de toute possibilité de traitement (alors qu'elle se trouvait en pleine exécution d'un ordre APDU). Il y a donc de fortes possibilités que le support carte soit dans un état incohérent. Il faut donc doter la carte à microprocesseur d'un mécanisme de reprise sur panne, pour qu'elle puisse se remettre dans un état cohérent à la prochaine connexion.

La notion de reprise sur panne pour la carte à microprocesseur peut avoir des implications sur les échanges APDU qui ont précédé l'arrachement de la carte. Par exemple, on peut avoir à défaire toutes les actions depuis le début de la session, car la

9. SGBD : Système de Gestion de Base de Données

mise en échec d'une partie de l'application peut entraîner un besoin d'annuler toutes les actions de cette application. Ou alors, on peut ne défaire que le dernier échange APDU (car c'est le seul dont le terminal n'a pas eu confirmation de l'exécution par les mots d'état).

1.5.2 Panne d'environnement d'exécution

Une carte seule ne peut rien faire : pour fonctionner, elle a besoin d'être connectée sur un terminal, sur lequel s'exécute une application spécifique, qui permet d'échanger des ordres avec la carte à microprocesseur. Nous appellerons ce terminal, l'*environnement d'exécution*, de la carte.

Dans le cas d'une panne logicielle, une panne de l'environnement d'exécution de la carte, est en quelque sorte le problème inverse à celui de l'arrachement (cf. figure II.1.7).

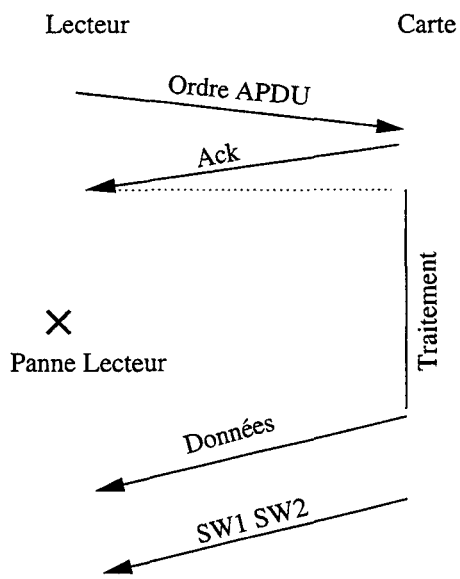


FIG. II.1.7 - *Conséquences d'une panne logicielle du terminal*

Comme on peut le voir, la carte effectue normalement son traitement, et n'a pas de moyen de savoir qu'il n'a pas été pris en compte par le terminal.

Dans le cas d'une application distribuée, ce type de panne est délicat à gérer, car le participant distant de l'application (le client), considère que la carte n'a pas exécuté le service demandé, alors que ce n'est pas le cas. Ce cas se terminera par l'arrachement de la carte par le porteur, quand celui-ci constatera la panne du terminal. Cependant (par rapport à un arrachement brusque), la carte ne sera pas en cours d'exécution d'une APDU.

Dans le cas d'une panne totale du terminal, nous cumulons les problèmes liés à la fois à l'arrachement soudain de la carte (car elle est alimentée en électricité par le terminal), et ceux liés à une panne logicielle du terminal.

Ce type de panne nécessite aussi un mécanisme de reprise sur panne pour que la carte et l'environnement extérieur soient remis en cohérence (il faut que la carte soit en mesure de défaire les actions exécutées alors que le terminal était en panne).

1.5.3 Perte, destruction

La perte ou la destruction d'une carte à microprocesseur marque la fin de son cycle de vie. Ainsi, toutes les informations contenues dans la carte peuvent être considérées comme perdues. Le problème du remplacement de la carte à microprocesseur se pose alors.

La sécurité de la carte est principalement basée sur le fait que les données qu'elle contient ne peuvent pas être piratées. Or, on ne pourra pas regénérer une carte sans faire une sauvegarde des informations qu'elle contient.

Une telle sauvegarde pose donc plusieurs problèmes :

- Comment effectuer une sauvegarde sécurisée des informations de la carte sur un support tolérant aux fautes,
- Comment maintenir cohérentes les informations sur le support externe. En effet, il devra être mis à jour à chaque modification du contenu de la carte,
- Comment retrouver ce représentant de la carte sur le réseau à chaque connexion de la carte à microprocesseur sur un terminal connecté.

1.5.4 L'erreur d'exécution

Le dernier type de panne mettant en jeu des cartes à microprocesseur est une panne logicielle: Il se peut que l'application commence une action, mais ne souhaite pas la poursuivre. L'application décide alors d'abandonner son exécution. J. Gray indique que le nombre des transactions dans les SGBD, qui se terminent par une annulation n'est pas négligeable **[GMB81]**.

L'exemple le plus classique d'annulation pour une application utilisant une carte, est le crédit (dans le cas où le solde devient supérieur à la limite imposée), ou le débit d'un PME (dans le cas où le solde devient négatif). Il se peut aussi que l'erreur soit due à un autre participant que la carte (par exemple, l'achat effectué avec la carte est annulé).

Dans tous les cas, la carte à microprocesseur doit être en mesure de défaire les actions qui doivent être annulées.

1.6 Conclusion

Comme nous avons pu le voir dans ce chapitre, l'évolution des carte à microprocesseur va permettre de nouvelles applications mettant en jeu des cartes. Cependant, ces nouvelles applications, exécutées dans un milieu très sujet aux pannes, impliquent de nouveaux besoins au niveau des systèmes cartes.

Les besoins des nouvelles applications des cartes s'expriment de la façon suivante (figure II.1.8) :

- Les cartes multi-services imposent une gestion de plusieurs contextes d'exécution à l'intérieur de la carte. De plus, l'accès à plusieurs services d'une même carte à microprocesseur impose un mécanisme de contrôle de concurrence des accès aux informations,

- Les cartes exécutant des applications sur plusieurs connexions, doivent être en mesure de reprendre une exécution[10] là où elle s'était arrêtée. Cependant, la carte doit aussi être en mesure d'annuler les modifications effectuées depuis le tout début de l'exécution de l'application si finalement elle échoue. De plus, la carte peut être engagée dans de nouvelles applications avant que l'application longue soit terminée. Cela peut donc nécessiter un mécanisme de contrôle de concurrence aux données,

- Enfin, la participation des cartes à microprocesseur à des applications distribuées, nécessite qu'elles prennent part à la validation des actions de ces applications (soit en tant que maître, soit en tant qu'esclave).

	Chargement	Multi-Contexte	Reprise sur Panne	Validation 2PC	Ctrl Concurrence
Multi-Services	Opt	Oui	Oui	Non	Oui
Appli. Longues	Opt	Oui	Oui	Non	XX
Appli. Distribuées	Opt	Oui	Oui	Oui	XX

XX = Dépend de l'application

Opt = Optionnel

FIG. II.1.8 - *Rappel Besoins des nouvelles Applications*

Dans la figure II.1.9, nous avons examiné ce que proposaient les cartes à microprocesseur les plus couramment distribuées, ou les plus novatrices, en terme de chargement d'application, de gestion de plusieurs contextes, de reprise sur panne et de contrôle des accès concurrents aux données.

Les cartes les plus couramment répandues (c'est-à-dire les cartes répondant à la norme 7816-4) sont en fait les plus simples. Elles ne proposent aucun des mécanismes nécessaires aux applications décrites dans ce chapitre.

A l'inverse, la carte CQL propose un mécanisme de reprise sur panne, et un mécanisme de contrôle de concurrence. Ces mécanismes restent cependant très simples et limités, car cette carte ne peut pas gérer plusieurs contextes d'exécution.

10. Reprendre une exécution implique de recharger le contexte d'exécution du service, et de reprendre le contexte applicatif (c'est-à-dire les données du service)

	Chargement	Multi-Contexte	Reprise sur Panne	Ctrl Concurrence
Carte 7816-4	Non	Non	Non	Non
C.Q.L.	Non	Non	Oui	Oui
JavaCard	Oui	Non	Limitée	Non

FIG. II.1.9 - *Rappel caractéristiques des cartes existantes*

Enfin, la dernière génération de carte à microprocesseur, représentée dans cette figure par la JavaCard, marque un progrès dans le sens où elle permet le chargement de services, mais elle ne permet toujours pas la coopération de ces services. En effet, bien qu'elle permette une gestion de sauvegarde des contextes d'exécution, elle ne dispose toujours pas de contrôle de concurrence des accès aux données, et ses mécanismes de reprises sur panne restent très limités.

Il apparaît donc dans cette figure, que les cartes à microprocesseur actuelles sont très loin de répondre aux besoins des nouvelles applications. Une évolution des systèmes d'exploitation des cartes est donc encore nécessaire, pour qu'elles puissent gérer plusieurs contextes d'exécution, et assurer un contrôle de concurrence sur les données.

La gestion par les programmeurs des applications, des coopérations entre les applications et des fautes, se révèle être un véritable casse-tête pour les développeurs. Le modèle transactionnel présenté dans le chapitre précédent, définit un modèle d'exécutions simultanées qui garantit les propriétés recherchées par les applications explicitées ici. Ce modèle simplifie grandement la tâche du programmeur, mais il implique l'introduction de mécanismes de reprise sur panne et de contrôle de concurrence dans les systèmes d'exploitation des cartes à microprocesseur.

Chapitre II.2

Modèle d'Exécution Carte et Besoin Transactionnel

2.1 Introduction

Nous avons vu dans le chapitre précédent, que les nouvelles applications des cartes à microprocesseur posent de nombreux problèmes. A ces problèmes (propres aux applications), s'ajoutent ceux propres aux cartes. La solution envisagée pour pallier à cela est l'implantation dans la carte d'un gestionnaire de ressources, basé sur le modèle transactionnel.

La notion de transaction peut intervenir à différents niveaux dans le modèle d'exécution carte. Actuellement, il n'existe pas de procédé dans les cartes permettant d'assurer l'atomicité d'exécution de plusieurs traitements. Les discussions dans le JavaCard Forum[1], portent actuellement sur l'atomicité de plusieurs instructions à l'intérieur d'un même traitement APDU [**JC97**]. Dans ces conditions, les discussions sur l'exécution de transactions à l'intérieur des cartes à microprocesseur sont simplement ramenées à l'implantation d'un mécanisme de reprise sur panne «minimal»[2].

Or, la problématique des nouvelles applications carte nous pousse à adopter un niveau transactionnel plus élevé, qui peut aller des commandes APDU sur plusieurs

1. Le JavaCard Forum est un groupe de discussion formé de plusieurs industriels (on peut notamment citer Bull, Gemplus, IBM, Schlumberger, SUN Microsystem), destiné à promouvoir et faire évoluer la JavaCard

2. Ce mécanisme est basé généralement sur l'utilisation d'un buffer de taille limitée

sessions jusqu'à plusieurs connexions de la carte.

Dans ce chapitre, nous allons tout d'abord étudier les conséquences des pannes des cartes à microprocesseur sur les applications simples actuelles. Ensuite, nous poserons le problème des nouvelles applications étudiées dans le chapitre précédent, au niveau du modèle d'exécution de la carte. Pour cela nous allons reprendre chaque exemple d'application et étudier son fonctionnement au niveau de la communication entre la carte et son terminal (instruction carte, APDU, session et connexion) et analyser comment s'applique la notion de transaction, et quels problèmes cela implique pour les systèmes d'exploitation des cartes à microprocesseur.

Une fois bien cernée la problématique de l'implantation d'un modèle transactionnel dans les systèmes d'exploitation carte, nous étudierons les possibilités d'évolution à la fois interne à la carte, mais aussi externe (utilisation des cartes comme client de transaction, et utilisation d'une représentation externe permanente pour pallier aux limites et défauts des cartes à microprocesseur.

2.2 Remarques préliminaires sur propriétés ACID et Cartes

L'implantation des propriétés ACID dans un système se fait via deux mécanismes : un mécanisme de reprise sur panne et un mécanisme de contrôle des accès concurrents (cf. chapitre 2.2).

Cependant, dans le cadre des applications carte les plus simples, certaines propriétés sont respectées d'office du fait des hypothèses de fonctionnement (une seule application à la fois). Ainsi, actuellement, la plupart des applications concernent un seul service carte (par exemple le porte-monnaie électronique), et se terminent en une seule connexion (c'est ce que nous appellerons des *Applications simples*). Alors, le seul mécanisme à mettre en jeu pour ces cartes est un mécanisme de reprise sur panne. Le cas de ces applications simples sera traité dans la section 2.3.

A l'inverse, les applications plus complexes décrites dans le chapitre précédent vont nécessiter les deux mécanismes pour respecter le modèle transactionnel, car la propriété d'isolation peut ne plus être respectée. Les modèles d'exécution relatifs à ces applications seront traités dans la partie 2.4.

Enfin, dans le milieu de la carte à microprocesseur, la propriété de durabilité pose un problème. En effet, la durabilité «sûre» à 100 % n'existe pas, mais avec la carte, il est impossible d'effectuer des sauvegardes sur un serveur classique[3]. Une solution pour améliorer la durabilité des informations est d'utiliser un représentant de la carte sur le réseau. Ce point sera développé à la fin de ce chapitre.

3. Cela est techniquement possible, mais la sécurité de la carte repose sur la présence des données uniquement au sein de ce support sécurisé

2.3 Besoins d'exécution Atomique des instructions

Nous allons ici étudier les besoins liés uniquement aux problèmes de reprise sur panne pour les cartes dans les applications simples actuelles. Cette étude a pour principal but d'analyser plus en avant les techniques employées dans les cartes, mais aussi de montrer qu'en terme de reprise sur panne, les solutions proposées actuellement (pour les applications cartes existantes) ne sont pas suffisantes (il s'agit de la notion d'Atomicité intra-APDU).

Nous verrons donc dans cette partie, qu'une bonne gestion des pannes cartes nécessite une atomicité d'exécution au moins d'une commande APDU complète, mais aussi peut être de plusieurs APDU.

2.3.1 Atomicité intra-APDU

Comme nous l'avons vu précédemment, certaines cartes, comme les JavaCards, disposent des ordres «Begin_Transaction», «Abort_Transaction» et «Commit_Transaction» **[JC97][Gem93]**. Cependant, il ne s'agit là que de rendre atomique l'exécution d'un petit nombre d'instructions à l'intérieur d'un ordre APDU (cf. figure II.2.1).

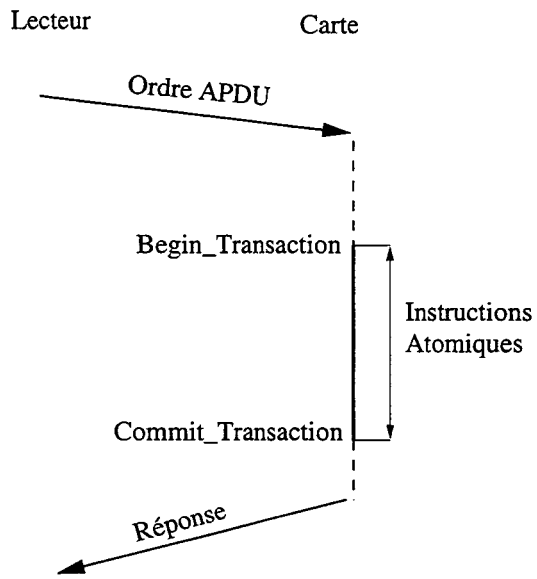


FIG. II.2.1 - *Instructions Atomiques Intra-APDU*

Ce degré d'atomicité, qui vise à rendre atomique l'exécution d'instructions dans la carte, répond aux besoins actuels des applications. En effet, dans les applications carte actuelles, il n'y a pas d'accès concurrents aux données (car il y a une seule application par connexion), et la reprise sur panne peut s'effectuer facilement (pas d'aspect distribué de l'application, et pas d'exécution sur plusieurs connexions).

Cependant, ce niveau de transaction est insuffisant pour rendre tolérant aux pannes l'exécution des services carte, même les plus simples. Nous allons voir que

actuellement, l'atomicité d'exécution des instructions doit être gérée par la carte à microprocesseur sur l'exécution complète de l'ordre APDU, voir même sur plusieurs ordres APDU.

2.3.2 Atomicité APDU

Ce degré d'atomicité, qui vise à rendre atomique l'exécution d'une APDU dans la carte (cf. figure II.2.2), a comme principal intérêt de rendre la carte tolérante à l'arrachement (ce qui est la panne la plus fréquente). En effet, lors d'un arrachement de la carte, le terminal ne reçoit pas ses mots d'état, et considère que la carte n'a pas effectué le traitement de l'APDU.

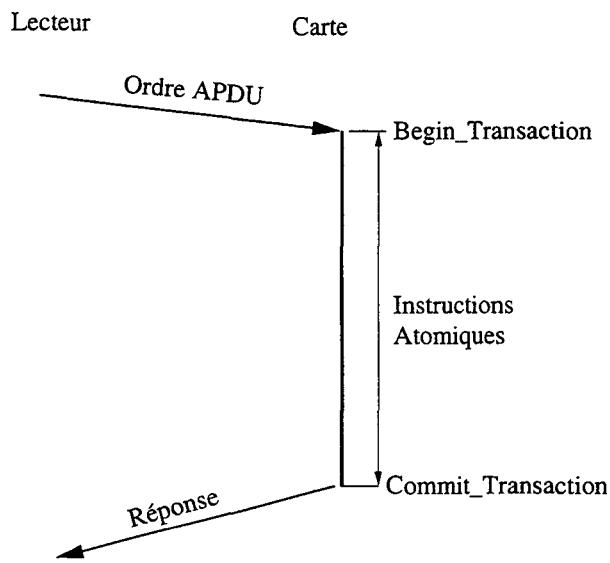


FIG. II.2.2 - *APDU exécutée atomiquement*

Le cas que nous venons de voir (atomicité intra-APDU) ne permet pas de garantir l'exécution totale, ou l'annulation totale en cas de panne, du traitement *complet* d'une APDU. En effet, cela oblige le programmeur du service à placer correctement les ordres de «Begin_Transaction» et de «Commit_Transaction». Dans l'état actuel des techniques utilisées dans les systèmes d'exploitation des cartes à microprocesseur, nous avons deux possibilités :

- Soit la carte a un système d'écriture en mémoire non volatile très simple (c'est à dire qu'elle n'utilise pas de cache). Dans ce cas là, les modifications sur les données sont rendues persistantes au fur et à mesure de l'exécution des traitements (sauf dans le cas de la figure II.2.1, où l'on stocke les modifications d'abord dans un buffer).

- Ou bien la carte est munie d'un cache pour accélérer les écritures, et dans ce cas là, les effets sur les données sont rendues persistantes au vidage du cache en mémoire non volatile.

En cas d'arrachement de la carte, il faut donc un mécanisme simple, implicite[4], et instantané qui permet de défaire les instructions exécutées depuis le début du traitement de l'APDU. Les mécanismes simples de reprise sur panne utilisés jusqu'à maintenant deviennent impossibles pour ce niveau d'atomicité, car ils sont généralement basés sur des buffers d'instructions, dont la taille est trop limitée[5] (cf. partie I.1.4.2 pour la carte CQL).

Ainsi, seul l'intégration dans les cartes, même les plus simples, d'un mécanisme de reprise sur panne performant, du type de ceux utilisés dans les modèles transactionnels, permettra de traiter de manière efficace les problèmes d'arrachement de carte en cours d'application.

2.3.3 Atomicité Intra-Session

Ce degré d'atomicité, vise à rendre atomique l'exécution de plusieurs APDU, faisant toutes partie de la même session, dans la carte. En effet, de manière à mieux gérer la reprise sur panne, le programmeur peut désirer rendre atomique l'exécution de plusieurs commandes APDU.

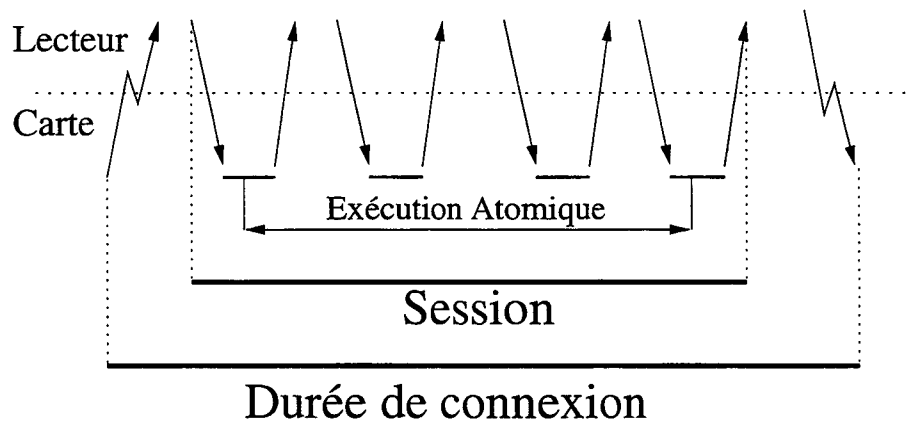


FIG. II.2.3 - *APDU Atomiques Intra-session*

Un exemple simple de traitement comprenant plusieurs APDU est le chargement d'un fichier ou d'un service dans la carte à microprocesseur. Si le chargement échoue, il faut défaire tous les traitements depuis la première APDU.

Un autre exemple est celui de l'invocation d'une méthode sur un objet dans une carte type JavaCard. Il faut invoquer la méthode, passer les paramètres et recevoir les résultats (dans le cas où il y a un retour). Cela se fait en plusieurs ordres APDU, et si la carte subit une panne, tout doit être défait (comme si la méthode n'avait pas été invoquée).

4. Qui ne nécessite pas de gestion de la part du programmeur, car l'arrachement peut survenir sur n'importe quelle APDU

5. On peut considérer qu'un buffer est un cache spécifique, qui est utilisé de manière différente

Contrairement au mécanisme qui rend l'exécution d'une APDU atomique, le mécanisme mis en jeu ici devra être explicite, pour plusieurs raisons :

- **Cohérence carte / terminal :** Le terminal devra être prévenu que les traitements effectués par la carte à microprocesseur sont susceptibles d'être défaits,
- **Exécution d'une transaction :** Il faut pouvoir délimiter les ordres que l'on veut atomiques. En effet, nous sommes dans le cas ou l'on exécute une transaction dans la carte. Le «Begin_Transaction» et «Commit_Transaction» définis pour certaines cartes prennent donc ici tout leur sens.

Ce type d'exécution atomique représente bien ce qui est actuellement désiré par l'industrie carte, pour rendre la carte totalement résistante à l'arrachement. Avec ce type de carte, on ne peut exécuter qu'une seule transaction à la fois. La seule propriété à implanter est donc un mécanisme de reprise sur panne gérant une seule transaction simultanée.

Dans ce cas, que l'on choisisse un mécanisme à base de journalisation, ou à base de pages ombres, les traitements relatifs à la validation ou bien à l'abandon d'une transaction ne seront pas très lourds à gérer, car :

- Une seule transaction implique un seul journal, ou une table d'indirection simple,
- Tous les problèmes d'abandon en cascade ou de compensation de transaction sont inexistants.

2.3.4 Conclusion

De manière à rendre les cartes à microprocesseur actuelles tolérantes aux pannes, le système d'exploitation doit assurer implicitement l'atomicité de l'exécution des instructions sur la durée d'une APDU, et le programmeur doit pouvoir demander explicitement l'atomicité d'exécution de plusieurs ordres APDU.

Cette conclusion impose l'intégration dans les cartes à microprocesseur d'un mécanisme de reprise sur panne. Ce mécanisme pourra demeurer relativement simple, car il ne s'agit pas ici de gérer plusieurs transactions, mais juste d'assurer l'atomicité de tout ou partie de l'exécution d'un service dans la carte.

Dans le cas de ces applications simples, il n'est bien entendu pas utile d'intégrer dans le système de la carte un mécanisme de contrôle de concurrence.

2.4 Nouvelles applications et modèle d'exécution carte

En terme de dialogue Carte / Terminal, les nouvelles applications décrites dans le chapitre précédent imposent des échanges plus élaborés. Ainsi, les services exécutés par la carte à microprocesseur peuvent s'effectuer sur plusieurs sessions, voir sur plusieurs connexions. De ce fait, nous allons voire qu'il ne s'agit plus seulement d'assurer la propriété d'atomicité, relative aux problèmes de reprise sur panne. Ces

nouvelles applications obligent d'autre part à un respect des propriétés ACID pour s'exécuter correctement.

Nous allons dans un premier temps étudier les problèmes relatifs à l'exécution d'une application sur une seule connexion de la carte (reprenant les problèmes liés aux applications faisant appel à plusieurs services d'une même carte), pour ensuite étudier la problématique des transactions sur plusieurs connexions de la carte (reprenant les problèmes liés aux applications «persistantes»).

2.4.1 Transaction Inter-Session et Intra-Connexion

La figure II.2.4 est la représentation au niveau de la communication carte / terminal d'une application mettant en jeu plusieurs services d'une même carte (du type de l'application PME / carte GSM décrite dans le chapitre précédent). Nous avons vu qu'une telle application, de part ses caractéristiques, et de part les caractéristiques de la carte à microprocesseur, nécessitait une exécution suivant le modèle transactionnel.

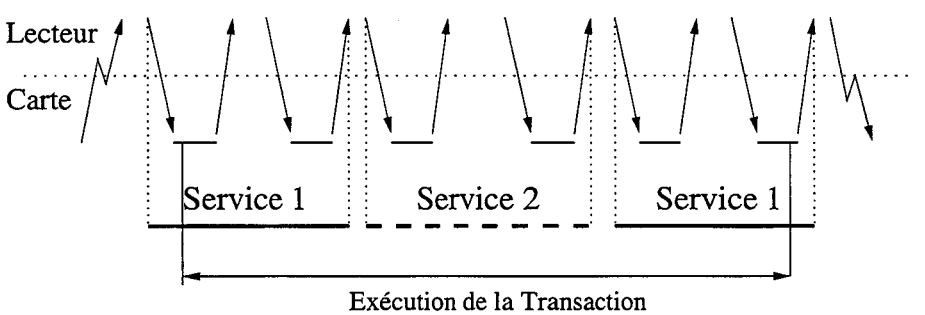


FIG. II.2.4 - *Transaction sur plusieurs sessions*

Dans le cas de cette application, l'exécution selon le modèle transactionnel se poursuit sur plusieurs sessions (étant donné qu'un changement de session correspond à un changement d'exécution de service carte). Il faut donc respecter les propriétés ACID, non seulement sur plusieurs APDU, mais aussi sur plusieurs sessions.

L'interruption du service 1 pour traiter le service 2 par la carte à microprocesseur implique un certain nombre d'actions à effectuer par la carte[6] :

- Tout d'abord, la carte doit sauvegarder sur le support non volatile les données relatives au service 1, ainsi que le contexte d'exécution de ce service,

- Ensuite, il faut s'assurer que le service 2 n'a pas déjà commencé une exécution au préalable (auquel cas, il faut recharger son contexte et reprendre l'exécution), et vérifier les droits d'accès à ce service.

6. Nous sommes ici dans le cas où le terminal fait appel au service 2. Le cas où le service 1 fait appel en interne à la carte au service 2 sera traité dans la partie 2.5

Concernant la propriété d'isolation des transactions dans le cadre de l'exécution de la transaction selon la figure II.2.4, nous avons deux possibilités :

- Soit l'exécution du second service se fait dans le cadre de la même application que le service 1. Dans ce cas, le service 2 a le droit d'accéder aux données, et de modifier les données déjà accédées par le service 1. Il n'y a donc pas de problèmes d'accès concurrents sur les données (l'identifiant de transaction est le même).

- Soit l'exécution du service 2 n'a pas de rapport avec l'exécution du service 1 (il ne fait donc pas partie de la transaction). Dans ce cas là, les accès aux données doivent suivre la règle de sérialisation des transactions.

Pour la propriété d'Atomicité, nous devons, là aussi, différencier les deux cas :

- Si l'exécution du service 2 se fait dans le cadre de la même transaction que le service 1, une seule transaction sera considérée comme étant en cours dans la carte à microprocesseur. En cas d'annulation de cette transaction, il faudra *défaire* les actions de la transaction : les effets des services 1 et 2 seront globalement annulés.

- Si l'exécution du service 2 se fait en dehors de la transaction, alors nous nous retrouvons dans le cas d'une exécution simultanée de deux transactions.

Si la carte doit fonctionner comme dans le premier cas, le mécanisme de reprise sur panne sera semblable à celui utilisé pour garantir l'atomicité sur plusieurs APDU. En effet, cela revient à considérer qu'il n'y a plus qu'une seule transaction en cours dans la carte à microprocesseur.

A l'inverse, si l'on désire que la carte puisse accepter des requêtes ne concernant pas la transaction qu'elle est en train de traiter, alors le mécanisme de reprise sur panne doit être en mesure de traiter efficacement plusieurs contextes transactionnels[7] simultanés.

Les cartes exécutants plusieurs services au cours d'une même connexion, doivent donc être munies des mêmes mécanismes que les serveurs «classiques». Cependant, ces mécanismes devront être adaptés aux caractéristiques des cartes à microprocesseur.

Concernant les mécanismes de contrôle de concurrence, il est à noter que la carte contient peu de données, et que donc, en cas d'exécution concurrente de services, le risque d'accès concurrent à ces données est élevé. Il semble donc préférable d'utiliser un mode de contrôle de concurrence pessimiste.

Pour le mécanisme de reprise sur panne, les mécanismes connus, comme les pages ombres, ou la journalisation, devront être adaptés aux cartes à microprocesseur, pour diminuer leur complexité, ainsi que la taille de mémoire utilisée.

7. On appelle contexte transactionnel l'ensemble des données permettant un bon déroulement de la transaction : identifiant de transaction, estampilles éventuelles, dépendances

2.4.2 Transaction Multi-Connexions

Ce dernier cas de communication Carte / Terminal (cf. figure II.2.5) est la représentation de l'exécution d'une application sur plusieurs connexions de la carte à microprocesseur.

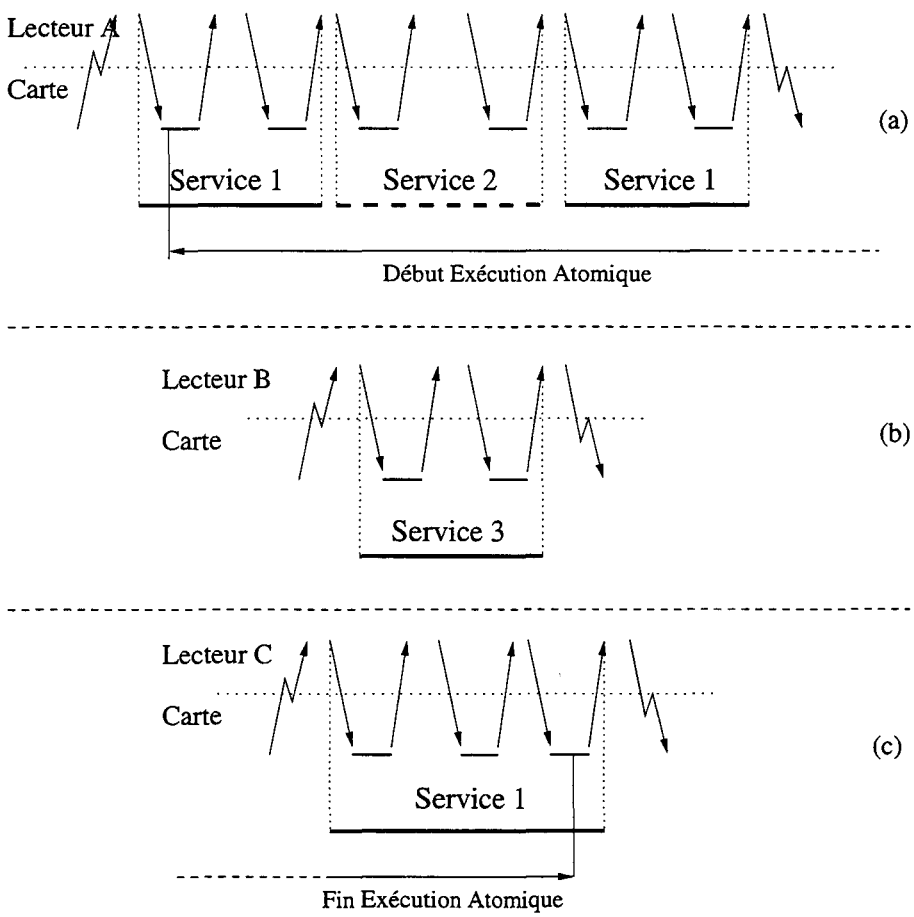


FIG. II.2.5 - *Transaction sur plusieurs connexions*

Ce type de dialogue Carte / terminal peut être divisé en trois phases :

- **La première phase** représente le début de l'application (c'est la phase (a) dans la figure II.2.5). L'application est initialisée au niveau de la carte à microprocesseur. Cela signifie qu'un contexte transactionnel est créé, et que tous les services s'exécutant dans le cadre de cette transaction devront faire référence à ce contexte. Cette phase se produit une seule fois dans le cadre de l'application.

- **La seconde phase** concerne la reconnexion de la carte à microprocesseur après le début de l'application (c'est la phase (b) dans la figure II.2.5). Cette phase peut bien entendu se produire plusieurs fois au cours de l'exécution de

l'application. A une reconnexion de la carte à microprocesseur, nous avons deux possibilités :

- Soit la carte est utilisée pour rendre un service sans aucun rapport avec l'application longue. Dans ce cas, il faut s'assurer que l'exécution de ce nouveau service n'entre pas en concurrence avec l'application longue. Cela peut passer par une exécution en *mode dégradé*[8] de la carte à microprocesseur. Cependant, un tel fonctionnement peut être problématique en cas de prolongement de l'application longue (cela peut, par exemple, bloquer l'utilisation du PME sur une longue période).
- Soit la carte se connecte pour poursuivre l'exécution de l'application longue. Dans ce cas là, le système de la carte à microprocesseur doit rétablir les données relatives à l'exécution de cette application, ainsi que les différents contextes.

- Enfin, **la troisième phase** concerne la dernière connexion de la carte relative à l'application longue. C'est dans cette phase que la décision de validation ou d'annulation de la transaction va être prise (c'est la phase (c) dans la figure II.2.5).

Il est important de bien comprendre que la seconde phase peut constituer le départ d'une nouvelle application (soit simple, soit multi-services, soit longue). Cette carte devra donc être en mesure de gérer plusieurs contextes transactionnels.

La problématique du fonctionnement de la carte à microprocesseur entre deux connexions de l'application longue, nécessite un contrôle de concurrence plus permissif que les contrôles de concurrence rigoureux, tels que le verrouillage à deux phases, ou l'estampillage (il faut assouplir l'isolation).

De plus, la carte étant déconnectée de son lecteur, le contenu de la mémoire volatile sera perdu. La fin d'une connexion de la carte à microprocesseur devra donc contenir une phase de sauvegarde du contexte de l'application, et une nouvelle connexion de la carte devra comporter une consultation de l'état de la carte (cette consultation pourrait être faite pendant l'ATR). Or, le protocole de connexion de la carte à microprocesseur ne comporte pas cette phase de chargement. La gestion de telles applications par la carte à microprocesseur impose donc, en plus d'une gestion d'un modèle transactionnel avancé, une modification du modèle d'exécution des applications cartes, permettant d'effectuer des traitements dans la carte à la reconnexion de celle-ci, avant même qu'elle ne reçoive d'ordre du terminal[9].

2.4.3 Conclusion

Le modèle d'exécution des services cartes s'avère quelque peu modifié par le besoin transactionnel des applications. Dans le cadre de la problématique des applications multi-services, chaque service doit vérifier qu'il n'a pas déjà démarré une

8. Par mode dégradé, on entend une exécution qui ne permet pas l'accès aux variables en cours d'utilisation

9. Cette modification qui concerne un traitement effectué automatiquement par la carte est à rapprocher du principe de la carte active décrit dans le chapitre I.1 **[NP97]**

exécution avant le nouvel appel. De plus, chaque accès à un objet doit être précédé d'une phase de contrôle de la possibilité d'accès (via un mécanisme de contrôle de concurrence). Enfin, bien que les exécutions simultanées de plusieurs services sur la carte ne soient pas possibles, le mécanisme de reprise sur panne devra être en mesure de gérer plusieurs transactions en cours. En effet, l'exécution d'un service carte peut être interrompue au profit d'un autre à tout moment.

La problématique des applications sur plusieurs connexions de la carte à microprocesseur impose elle aussi une modification du modèle d'exécution des services de la carte. En effet, les cartes permettant de telles applications devront «tester» leur état à chaque nouvelle connexion. Par tester, on entend rechercher si une application est en cours d'exécution ou non. De plus, ces applications nécessitent l'implantation d'un modèle transactionnel complexe, basé à la fois sur le modèle à flot de tâches, et sur la possibilité de coopération entre différentes transactions (pour, par exemple, permettre l'utilisation concurrente d'un PME).

2.5 Le cas des transactions emboîtées dans la carte

Dans le cas où un service de la carte à microprocesseur fait appel à un autre service de cette même carte, nous nous trouvons dans le cadre du modèle d'exécution présenté par la figure II.2.6.

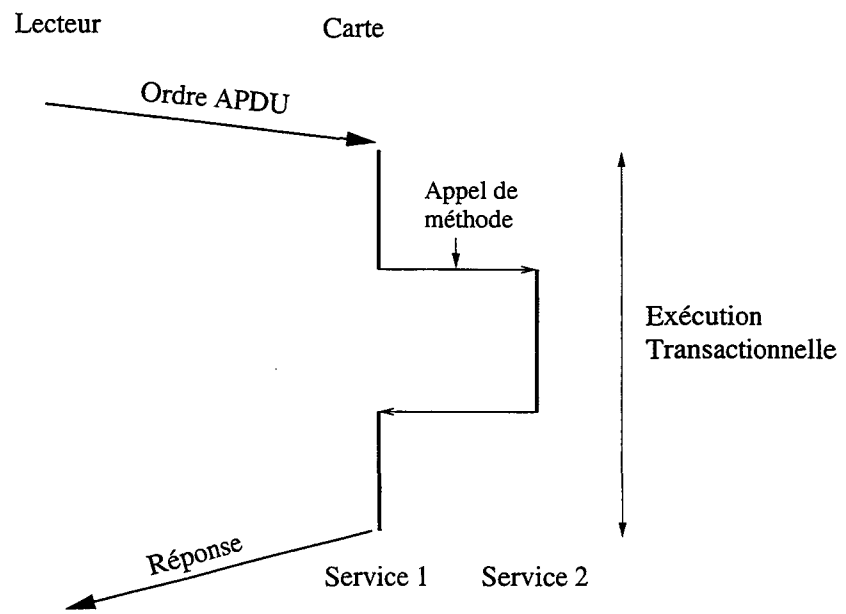


FIG. II.2.6 - *Appel de service Intra-APDU*

Les problèmes associés à une telle exécution sont avant tout des problèmes de sécurité, et de systèmes d'exploitation carte, liés à l'exécution de plusieurs services dans la carte.

Les problèmes de sécurité sont principalement liés aux schémas de sécurité à définir pour les transmissions de droits d'un service vers un autre [**T95**].

Le manque de mémoire volatile pose des problèmes en terme de système d'exploitation, en effet, il est impossible d'exécuter un second service dans la carte, sans rendre totalement disponible la RAM pour ce service. Il faut donc rendre persistante les informations du service 1 contenues dans la RAM, pour pouvoir ainsi la vider et la rendre disponible pour le service 2.

Dans les cas précédents (application multi-service, ou application longue) le changement de contexte s'effectuait entre deux ordres APDU. Dans ce cas, le contenu de la mémoire volatile est rendu persistant à la fin de chaque ordre APDU. Ici, le changement de contexte s'effectue pendant l'exécution de l'ordre APDU. Cela peut poser problème en cas d'arrachement de la carte à ce moment. Il faudra donc s'assurer de l'atomicité du changement de contexte (soit on se trouve en cours d'exécution du service 1, soit on se trouve en cours d'exécution du service 2).

2.6 La carte et les transactions distribuées

La carte doit être appelée à participer à des transactions distribuées comme serveur (pour par exemple fournir le service de paiement sécurisé). Nous allons voir dans cette section ce qu'implique cette intégration, tant au niveau des systèmes distribués classiques, qu'au niveau de la carte à microprocesseur.

Dans un premier temps, nous allons étudier les problèmes relatifs à la communication entre la carte et les autres partenaires de la transaction distribuée. En effet, il a été prouvé que l'intégration de la carte dans un système d'information ne devait pas modifier ce système [**H92**], car cela se révélerait beaucoup trop coûteux pour les fournisseurs et les utilisateurs de ces systèmes.

Ensuite, nous étudierons les problèmes posés par la mobilité de la carte à microprocesseur. En effet, la carte ne sera pas obligatoirement reconnectée sur le dernier terminal où elle se situait. Il se pose donc des problèmes de localisation de cette carte lors de sa future connexion.

Enfin nous terminerons en examinant les conséquences sur les systèmes distribués des contraintes de la carte, et notamment la possible utilisation d'un représentant permanent de la carte à microprocesseur sur le réseau.

2.6.1 Intégration dans les systèmes existants

L'introduction des cartes à microprocesseur dans les systèmes distribués débouche nécessairement sur une redistribution des services entre les cartes et les systèmes existants. Cependant, il n'est pas possible de modifier en profondeur les systèmes distribués classiques existants, car le coût en serait trop important (coût de développement, et coût des mises à jour pour les entreprises).

Un système distribué acceptant des cartes (figure II.2.7) est en fait un système classique muni de terminaux pour les cartes à microprocesseur. Le principal problème est de rendre l'accès aux cartes à microprocesseur semblable à l'accès à n'importe quel autre composant.

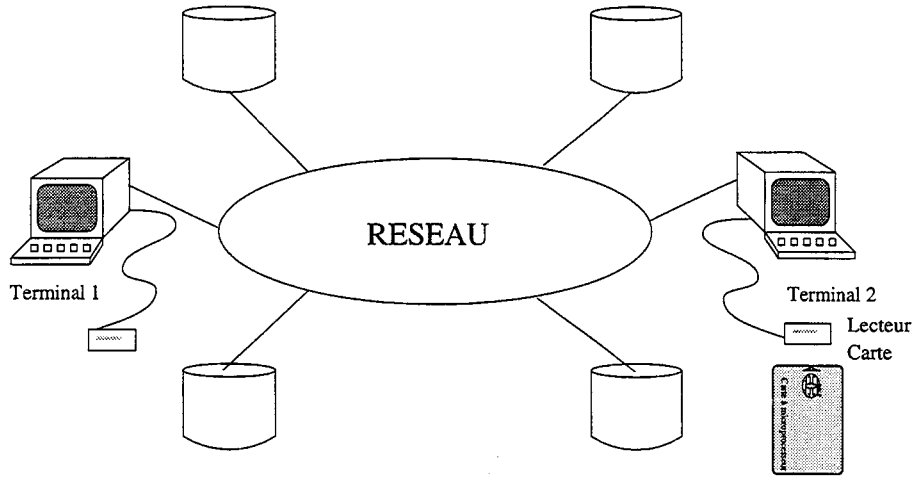


FIG. II.2.7 - *Système distribué acceptant des cartes*

En terme de communication, la carte utilise un protocole normalisé, qui n'est pas utilisé dans les systèmes d'information classiques. Ce problème a déjà été étudié, et la solution est basée sur l'utilisation d'un composant d'adaptation pour carte, qui présente les services de la carte sur le réseau, et qui assure la traduction des requêtes (projet OSMOSE) [**V97**].

Cependant, le modèle transactionnel et les nouvelles applications cartes posent de nouveaux problèmes :

- Il faut que la carte soit en mesure de participer au protocole de validation distribuée. Cela signifie qu'elle doit accepter les ordres émis par un coordinateur de validation (comme par exemple «Begin_Transaction», «Prepare», «Abort» et «Commit» pour le mécanisme de validation à deux phases), et qu'elle doit être en mesure de décider quelle transaction doit être validée ou abandonnée (Gestion de plusieurs contextes transactionnels par le gestionnaire de ressources de la carte).

- Il faut pouvoir localiser la carte lorsqu'elle se reconnecte : La problématique de représentation et de localisation de la carte à microprocesseur dans les réseaux est semblable à celle des ordinateurs mobiles, qui peuvent se reconnecter à n'importe quel endroit du réseau [**L94**]. Dans le monde de l'informatique mobile, des solutions basées sur l'utilisation d'une représentation permanente fixe de l'objet mobile voient le jour [**C98**]. Contacter un ordinateur mobile revient alors à contacter sa représentation permanente, pour obtenir la nouvelle adresse du mobile. Ainsi donc, à chaque connexion, l'ordinateur mobile doit contacter sa représentation pour lui signaler sa nouvelle adresse.

- Il faut être en mesure de prendre en compte les caractéristiques des cartes à microprocesseur dans la transaction distribuée (comme la déconnexion fréquente de la carte à microprocesseur). Il faut donc être en mesure de stocker

certaines informations sur des files d'attente, que la carte ira consulter à sa reconnection (selon le principe du message queuing [**Iso93**]).

2.7 La carte cliente de transaction

Nous avons vu dans le chapitre précédent, que d'un point de vue applicatif, il était intéressant de considérer la carte à microprocesseur non plus comme un serveur, mais aussi comme un client de transaction distribuée.

De plus, la carte à microprocesseur est un lieu d'exécution sécurisé, qui permettrait de garantir la bonne exécution d'un algorithme de validation distribuée (la carte pourrait être coordinateur de transaction). Cependant, le choix de la carte à microprocesseur comme coordinateur de validation de transaction, oblige à prendre en compte le mode déconnecté de la carte, qui se révèle bloquant en cas d'utilisation du protocole de validation à deux phases. De plus, actuellement, de par ses capacités de stockage, la carte peut difficilement remplir tous les rôles d'un coordinateur de validation (notamment au niveau de la journalisation des requêtes).

Cependant, la carte à microprocesseur pourra être envisagée comme coordinateur de validation de transaction distribuée lorsque certaines de ses limites seront correctement prises en compte dans les systèmes distribués (gestion du mode déconnecté de la carte, et possibilité d'archiver des informations carte, de manière sécurisée, en dehors de la carte).

2.8 Synthèse

Il apparaît que les problèmes et solutions liés à l'introduction dans les cartes à microprocesseur de mécanismes permettant d'assurer le bon déroulement de transactions sont multiples. Il ne s'agit donc plus ici de définir UN système d'exploitation orienté système transactionnel, mais bien différentes variantes dépendantes du type d'application à laquelle devra participer la carte à microprocesseur.

Ainsi, pour les prestataires de services qui désirent une carte mono-service tolérante aux pannes, un mécanisme de reprise sur panne simple et mono-transactionnel est suffisant (du type des pages ombres, ou un système de journalisation simplifié). A l'inverse, certains domaines d'applications (comme la téléphonie mobile) doivent toujours proposer plus de services. Ces prestataires de services préféreront donc des cartes multi-services, pouvant participer à tous types d'applications (applications multi-services carte, ou applications longues). Ces cartes nécessiteront un véritable gestionnaire de ressources transactionnelles, incluant un mécanisme de reprise sur panne évolué, ainsi qu'un mécanisme de contrôle des accès concurrents aux ressources de la carte à microprocesseur.

A coté de ces domaines d'applications classiques, de nouveaux horizons s'ouvrent à la carte à microprocesseur, principalement basés sur le développement de l'Internet commercial. La carte y trouve un nouveau rôle, plus de client que de serveur, voir de coordinateur de systèmes «classiques». Ces nouvelles cartes obligent à prendre en compte dans les systèmes distribués classiques le mode de fonctionnement déconnecté

de la carte à microprocesseur, de manière à ce que cette déconnexion ne soit pas bloquante pour les autres participants de la transaction.

Troisième partie

Solutions Proposées et Implantation

Chapitre III.1

Un Gestionnaire de Ressources Transactionnel Carte

1.1 Introduction

Nous avons vu dans la partie précédente, que les mécanismes à mettre en oeuvre pour gérer les ressources des cartes à microprocesseur dépendaient des applications pour lesquelles étaient conçues les cartes. Ainsi, pour être conformes au modèle transactionnel, les cartes dédiées à des applications les plus simples auront besoin uniquement d'un mécanisme de reprise sur panne peu évolué. A l'inverse, les cartes dédiées à des applications évoluées du type de celles étudiées dans le chapitre II.1 auront besoin d'un véritable gestionnaire de ressources transactionnel comprenant à la fois un mécanisme de reprise sur panne, mais aussi un mécanisme de contrôle de concurrence.

Dans ce chapitre, nous allons d'abord définir le système d'exploitation dont les applications décrites précédemment ont besoin. En nous basant sur l'état de l'art des systèmes d'exploitation pour carte, nous allons faire un certain nombre de choix, qui vont influer sur les mécanismes utilisés pour implanter un gestionnaire de transactions dans les systèmes cartes.

Ensuite, nous étudierons l'implantation dans les cartes d'un mécanisme de reprise sur panne. Nous commencerons par examiner comment nous avons adapté les mécanismes de reprise sur panne listés dans l'état de l'art au contexte des cartes à microprocesseur. Ensuite, pour établir des choix entre ces deux mécanismes, nous distinguerons les cartes actuelles, des cartes à utiliser pour les nouvelles applications.

Après cela, nous étudierons le mécanisme de contrôle de concurrence le mieux adapté aux cartes à microprocesseur, et nous tirerons des conclusions semblables à celles de la solution pour la reprise sur panne.

Enfin, nous terminerons ce chapitre en présentant comment nos solutions peuvent s'adapter à la JavaCard, et quelles sont les contraintes liées aux solutions envisagées pour le gestionnaire de ressources carte.

1.2 Choix du Système d'Exploitation de base

Les systèmes d'exploitation des cartes à microprocesseur sont encore relativement limités. Des travaux, basés sur les cartes génériques, sont en cours, notamment pour améliorer la gestion de la mémoire, mais aussi les relations entre les différents services carte [**GR97**].

Nous allons, dans cette section, définir les caractéristiques d'un système d'exploitation pour carte à microprocesseur, qui permettent d'intégrer efficacement un mécanisme de gestion transactionnelle des ressources carte, et de gérer (voir même d'installer) les nouvelles applications décrites précédemment.

1.2.1 Vers une mémoire virtuelle pour carte

Dans le cadre des cartes multiplicatives, une gestion avancée de la mémoire devient inévitable. En effet, les machines virtuelles utilisées dans le cadre de ces cartes génériques [**JC97b**] incluent la prise en compte de ramasse-miettes pour les allocations mémoire. Les gestionnaires de mémoire virtuelle (implanté de façon logicielle) ont donc plusieurs avantages :

- Ils masquent les contraintes liées à la granularité, en écriture, des banques de mémoire Flash (64 Octets),
- Ils permettent d'introduire des caches en RAM pour les écritures en Flash, sans pénaliser davantage les temps d'accès en lecture.

Le gestionnaire de mémoire virtuelle décrit par la figure III.1.1 fonctionne de la manière suivante :

- Lorsqu'un ordre de lecture est invoqué, le gestionnaire de mémoire virtuelle consulte la structure de données qui supporte l'indirection. Il obtient ainsi l'adresse physique, à partir de l'adresse virtuelle. Enfin, la lecture effective en mémoire peut être effectuée à partir de l'adresse obtenue. Nous ne décrirons pas ici le mécanisme d'indirection, dont un exemple est défini dans [**GR97**]. Il faut cependant noter que la complexité algorithmique d'un tel mécanisme peut se rapporter à une constante.
- Lorsqu'un ordre d'écriture est invoqué, le calcul de l'adresse effective (adresse physique) est obtenu comme précédemment. Si ce calcul indique une zone de mémoire RAM, l'opération d'écriture peut être faite sans différer (écriture dans le cache). Dans le cas contraire, il faut choisir un espace en RAM pour cacher l'écriture, et modifier la structure de données qui effectue l'indirection.

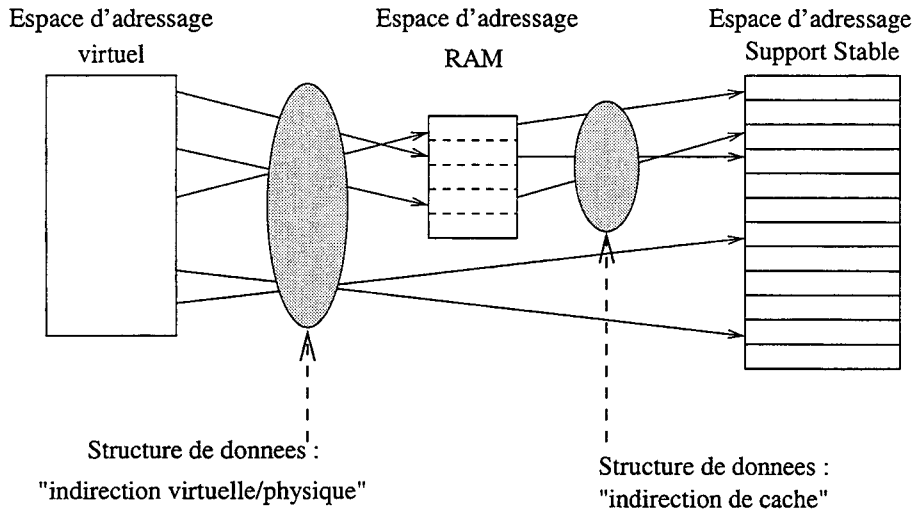


FIG. III.1.1 - *Correspondances de mémoire virtuelle cachée*

Une première mémoire virtuelle a été implantée dans le système d'exploitation CASCADE [**Esp95**]. Ce gestionnaire de mémoire, basé sur l'utilisation d'une table d'allocation permettant de s'affranchir de la fragmentation de la mémoire, prouve la faisabilité de la technique dans la carte.

A défaut de disposer d'une mémoire virtuelle, certains mécanismes que nous nous proposons d'utiliser, nécessiteront l'utilisation d'un cache en écriture. De manière à résoudre les problèmes posés par les nouvelles technologies de mémoire non volatile (granularité de la mémoire Flash, ou effacement lors de la lecture de la FeRAM), des techniques d'écriture en mémoire basées sur l'utilisation d'un cache en RAM voient le jour. Ainsi, lors de l'accès en écriture sur un banc de mémoire non volatile, le banc complet est copié en RAM, et les accès suivants s'effectuent dans la mémoire volatile. Ces procédés constituent un premier pas vers une gestion plus élaborée de la mémoire dans les cartes à microprocesseur.

1.2.2 Permettre le partage d'objets dans la carte

Nous avons vu dans le chapitre sur les nouvelles applications des cartes à microprocesseur, que ces applications allaient avoir à partager des données, voire des services.

La réalisation d'un tel partage pose des problèmes, tant au niveau transactionnel (nécessité d'un contrôle de concurrence), qu'au niveau de la sécurité.

Cependant, un tel partage est nécessaire si l'on veut permettre des applications telles que les applications multi-services, qui ont été définies dans les chapitres précédents.

Nous considérerons donc que le système d'exploitation de la carte à microprocesseur permet de partager des objets dans la carte, et nous nous pencherons uniquement sur les problèmes que cela pose au niveau de la gestion des ressources de la carte de manière transactionnelle.

1.3 Implantation des pages ombres dans une carte

Dans cette section, nous allons étudier plus en profondeur la technique de reprise sur panne basée sur l'utilisation de pages ombres [**BHG87**]. Après un rappel du principe de fonctionnement de ce mécanisme, nous étudierons la faisabilité d'une utilisation dans le cadre des cartes à microprocesseur (en terme de coût et de capacité à gérer des modèles transactionnels avancés).

1.3.1 Principes

Le mécanisme des pages ombres préserve l'état initial des données sur le support stable (rendant très facile la possibilité d'annuler une transaction). Toutes les mises à jour des données sont effectuées sur une copie de la page (appelée page ombre).

La méthode des pages ombres oblige donc :

- A dupliquer les données[1]. Les modifications sont réalisées sur une copie de la page mémoire.
- A maintenir à jour une table d'indirection supplémentaire. Il est nécessaire de définir une table qui permet soit d'accéder à la page de mémoire contenant les données non modifiées, soit d'accéder à la page ombre.

Le coût en place n'est donc pas négligeable, et il y a aussi un surcoût en temps : à la validation, la table d'indirection qui mène aux pages ombres doit être changée.

1.3.2 Implantation et type de mémoire

Comme nous l'avons décrit dans la partie I.1 de ce document, les cartes à microprocesseur peuvent disposer de différents types de mémoire non volatile (EEPROM ou Flash). Suivant ce type de mémoire, l'implantation du mécanisme de pages ombres est différente, et plus ou moins coûteuse en place et en temps [**DGL98**].

1.3.2.1 Implantation avec de la Flash

Une implantation en mémoire Flash du mécanisme des pages ombres nécessite une reconstruction en RAM de la table d'indirection active. En effet, de par la granularité des écritures en Flash, il est impossible de travailler directement sur le support non volatile.

Le mécanisme décrit par la figure III.1.2 comporte les éléments suivants :

- Deux tables d'indirection, permettant d'adresser soit la page contenant une donnée modifiée par une transaction, soit la page contenant la donnée non modifiée. Chaque table d'indirection a une taille de 256 octets[2] (elle est constituée de 4 lignes[3] de flash de 64 octets, chaque ligne permettant d'adresser un segment de 4 Ko de mémoire de données), ce qui permet d'adresser 16Ko de données en Flash.

1. C'est le cas de toutes les techniques de reprise sur panne
2. 1 octet pointe vers une page de 64 octets
3. au terme ligne, on peut préférer le terme Banque ou Page

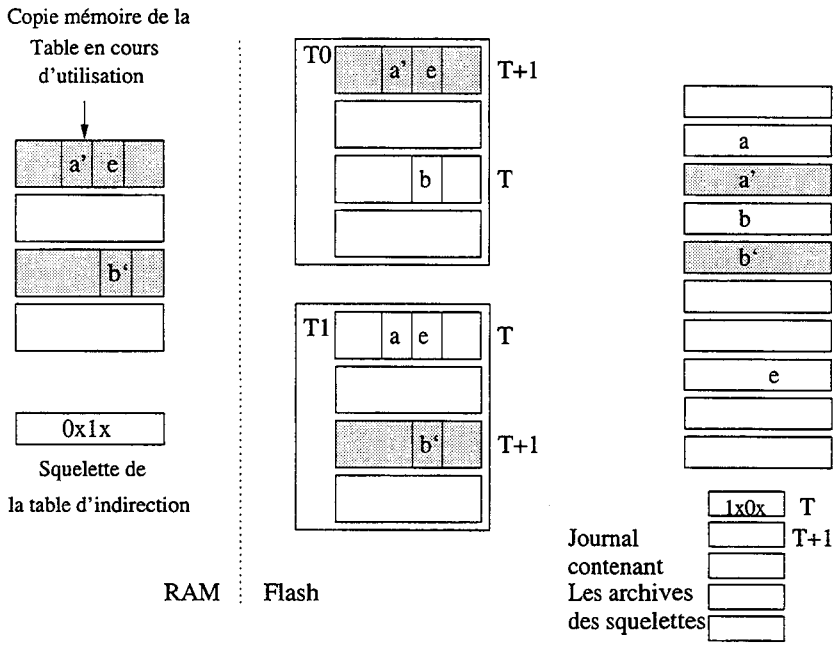


FIG. III.1.2 - *Pages Ombres et mémoire Flash*

- Une trace, en mémoire non volatile, indique les lignes de flash qui pointent vers des données non modifiées[4] (des données propres). Un 0 indique que la ligne de flash à utiliser est celle de la table 0, un 1 indique que la ligne à utiliser est celle de la table 1, et le x indique que le choix est indifférent : ce x vaut 0 ou 1, suivant la valeur précédente du squelette ; il n'y a pas eu de modification dans ce segment de mémoire par la transaction).

- Une troisième table d'indirection est reconstruite à partir des deux premières, et est située en RAM. Au début d'une transaction, cette table pointe vers les pages valides des données. Puis, au cours de la transaction, cette table va pointer vers les pages ombres contenant des données modifiées.

La figure III.1.2 décrit l'exécution d'une transaction T, modifiant deux données (a et b) sur le support non volatile. Avant l'exécution de la transaction, le squelette de la table d'indirection est formé par les pages marquées T (ce squelette vaut 1x0x, avec x=0 ou 1 suivant les précédentes transactions). Les modifications des données a et b sont sauvegardées dans les pages notées a' et b'. La table d'indirection active (c'est à dire située en RAM) ne pointe plus vers les pages contenant a et b, mais vers celles contenant a' et b'.

La validation de la transaction T se fait en deux étapes :

- La partie de la table d'indirection modifiée est recopiée en mémoire non volatile ligne par ligne. Si une ligne a été prise à partir de la table 0 en début de transaction, la nouvelle ligne est recopiée dans la table 1, et inversement (il s'agit ici des parties pointant vers a' et b').

4. Ces demi-octets forment en fait le *squelette* de la table d'indirection valide

- Une fois les tables d'indirection mises à jour, le squelette de la nouvelle table d'indirection est rendu persistant à son tour, en étant recopié de manière atomique sur le support non volatile[5]. Ainsi, la prochaine fois qu'une table devra être reconstruite en RAM, les dernières modifications seront prises en compte.

Avec une telle architecture, le surcoût en terme de temps, et d'occupation de la mémoire est relativement élevé (environ plus de 256 octets de RAM, et plus de 512 octets de Flash pour 16 Ko de mémoire de données). Une annulation de la transaction se fera en ne prenant pas en compte les modifications faites dans la table d'indirection en RAM (la table valide redevient celle du début de la transaction).

1.3.2.2 Implantation avec de l'EEPROM

Contrairement à l'implantation de ce mécanisme avec de la mémoire de type Flash, l'utilisation de mémoire de type EEPROM, permet de modifier directement les tables d'indirection sur le support non volatile (ce n'est plus la peine de reconstituer une table en mémoire volatile).

Le principale avantage de cette nouvelle implantation est alors d'occuper beaucoup moins de place en RAM que la solution proposée pour la mémoire Flash. Cependant, l'inconvénient de l'utilisation du support non volatile pour la table d'indirection, est une perte de performance en terme de rapidité d'exécution (l'EEPROM étant plus longue en temps de lecture que la RAM).

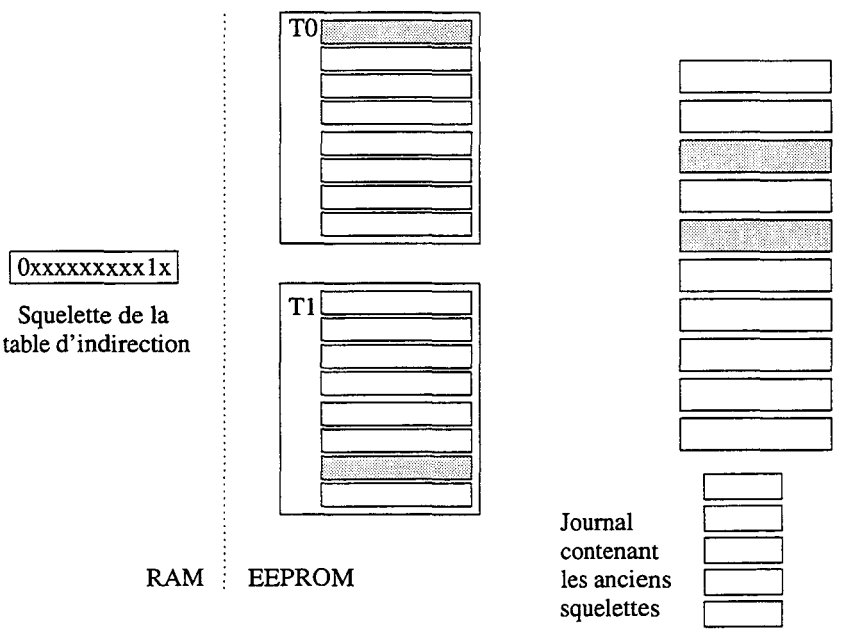


FIG. III.1.3 - *Pages Ombres et mémoire de type EEPROM*

L'architecture logicielle présentée dans la figure III.1.3 comporte donc à peu près

5. Avec une technologie Flash, on peut mettre à jour de manière atomique jusqu'à un octet

les mêmes éléments que dans le cas d'utilisation de mémoire Flash. On y trouve :

- Tout comme pour le mécanisme utilisant de la mémoire Flash, on a deux tables d'indirection de 256 octets sur le support de mémoire non volatile pour adresser une mémoire de données de 16 Ko.

- En RAM se trouve uniquement le squelette de la table d'indirection en cours d'utilisation. Lorsque l'on employait de la Flash, la table d'indirection était divisée en blocs de 64 octets. Dans le cas de l'EEPROM, on n'a plus de contrainte technique pour la taille des blocs[6]. Des recherches sont en cours dans le domaine des systèmes d'exploitation pour carte à microprocesseur, afin de déterminer la granularité idéale de division des tables d'indirection, dans le cas d'EEPROM : granule de 4, 8, 16 ou 32 octets[7] **[Esp95]** (la granularité d'écriture de la mémoire EEPROM est de 4 octets).

Le principe de fonctionnement diffère de celui décrit pour la mémoire Flash, dans le sens où il n'est pas nécessaire de reconstituer une table d'indirection en RAM.

Avec une telle architecture, le surcoût en terme de temps, et d'occupation de la mémoire est relativement faible (quelques octets de RAM, et un peu plus de 512 octets d'EEPROM). Une validation s'effectue en une seule étape : la mise à jour du journal qui est utilisé pour constituer une table d'indirection valide à partir des deux tables.

1.3.3 Synthèse sur les pages ombres

La faisabilité de l'utilisation du mécanisme des pages ombres dans les cartes à microprocesseur est ici démontrée.

Dans le cas de l'utilisation d'une carte basée sur de l'EEPROM, le mécanisme des pages ombres se révèle particulièrement léger à implanter (en occupation mémoire). Ce mécanisme peut se révéler comme étant une bonne solution pour des cartes simples, n'exécutant qu'une seule transaction simultanée, et disposant de peu de ressource (notamment en terme de mémoire volatile).

Pour les cartes à microprocesseur utilisant de la mémoire Flash comme support de données non volatile, ce mécanisme se révèle un peu plus coûteux en terme de place mémoire (notamment en RAM). Cependant, l'utilisation de Flash permet d'augmenter sensiblement la taille de la RAM embarquée. Cette technique peut donc s'avérer satisfaisante dans le cas des cartes à base de Flash exécutant une seule transaction simultanément.

Cependant, le mécanisme des pages ombres que nous avons présenté dans ce chapitre ne permet pas de gérer la reprise sur panne sur plusieurs transactions simultanées. En effet, ce mécanisme utilise la table d'indirection de la mémoire virtuelle de la carte à microprocesseur. Il ne peut donc y avoir qu'un seul squelette modifié en RAM à la fois. Dans le cas contraire, la validation d'une transaction (c'est à dire la recopie du squelette sur le support stable) risquerait d'annuler les modifications du squelette valide d'une transaction concurrente (qui aurait été recopié sur le support juste avant).

6. Une division de la table d'indirection en bloc de 32 octets forme un squelette de 8 bits
7. L'exemple de la figure III.1.3 est décrit avec des granules de 32 octets

1.4 Implantation d'une reprise sur panne à base de journaux

Dans cette section, nous allons reprendre la méthode basée sur la journalisation des données, pour adapter ce mécanisme à une implantation dans les systèmes d'exploitation des cartes à microprocesseur. Après un rappel des principes de base, nous étudierons diverses optimisations possibles, et nous conclurons sur la faisabilité d'une réalisation de ce mécanisme dans les cartes.

1.4.1 Rappels sur les Principes

Le journal est un «fichier» qui contient des informations permettant la reprise après un échec (panne, erreur d'exécution) lors du déroulement d'une transaction. Le journal peut contenir :

- L'identifiant de la transaction qui a fait la modification
- L'adresse de la page mémoire qui a été modifiée
- La nouvelle valeur de cette page, appelée image après
- Ou bien, l'ancienne valeur de la page appelée image avant.

De manière à pouvoir faire un Abort d'une transaction, il faut s'assurer que toutes les mises à jour, effectuées sur le support stable de données, aient leur image avant stockée dans le journal. Un Commit ne peut être fait que si toutes les images après des données sont sur le support stable **[HR83]**.

1.4.2 Coûts et optimisations

Un journal complet (avec tous les composants décrits précédemment), a un encombrement supérieur à deux pages de mémoire pour chaque enregistrement. Ceci est bien sûr très problématique dans une carte à microprocesseur, où la place mémoire est très limitée.

Une première optimisation, consiste à utiliser un journal des images avant. Avec un tel journal, on effectue directement le vidage du cache (c'est-à-dire la mise à jour des données) sur le support de travail persistant, et on journalise l'ancienne valeur de la page **[LC97]**. Avec un tel mécanisme, le surcoût dû au mécanisme de reprise sur panne est légèrement supérieur à une page par enregistrement.

On peut aussi utiliser un journal après, qui contient uniquement les images après des données modifiées, et qui laisse le support stable propre. Cependant, dans le cas de la carte à microprocesseur, cette méthode est peu adaptée, car elle oblige à forcer systématiquement le vidage du cache en mémoire persistante (pour rendre persistante la nouvelle valeur de la donnée), car le risque d'arrachement de la carte est très important. Nous ne détaillerons donc pas cette technique.

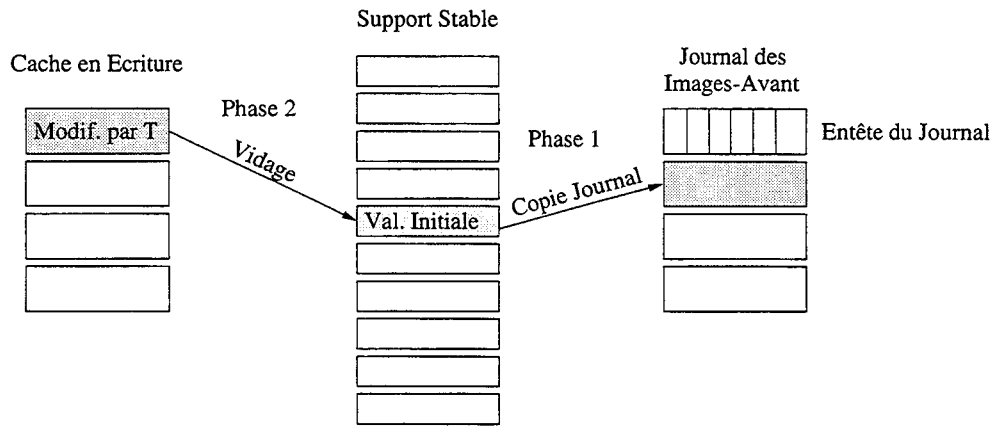


FIG. III.1.4 - *Journal des images-avant*

Dans la Figure III.1.4, lors du vidage d'une page modifiée du cache sur le support stable, les différentes étapes sont les suivantes :

- Tout d'abord, l'image avant de la page est copiée dans le journal de la transaction. Ce journal sera utilisé pour défaire la transaction en cas de besoin.
- Durant la transaction, en cas de besoin de place dans le cache, les pages modifiées sont directement réécrites sur le support stable de données (le journal n'est pas modifié).
- En cas de besoin d'accès à une nouvelle page mémoire, la page avant modifications, est copiée dans le journal, et est chargée dans le cache (on modifie l'entête du journal pour signaler la présence de la nouvelle page).
- Au moment du Commit, le cache est synchronisé avec le support stable, puis, la fin de la transaction est marquée par l'effacement du journal correspondant.
- En cas d'annulation de la transaction, il faut parcourir le journal en sens inverse, et recopier les valeurs initiales des données (qui ont été stockées dans le journal) à la place des valeurs modifiées. Le coût d'une annulation de transaction est donc relativement élevé.

D'autres optimisations visent à diminuer la taille de journal : plutôt que de journaliser la page entière, on journalise uniquement la zone de la page qui a été modifiée. Cette méthode est appelée «Page Diffing» et a été expérimentée dans les systèmes de gestion de base de données orientés objets [**D94**].

1.4.3 Synthèse sur la journalisation

Le principal défaut de ce mécanisme de reprise sur panne est le nombre d'écritures qu'il oblige à faire : à chaque écriture sur le support stable (vidage du cache en mémoire persistante) il faut, en réalité, effectuer deux écritures. Cela est bien sûr très pénalisant au niveau du temps d'exécution de la transaction.

Opposée à cela, la principale qualité de ce mécanisme est de pouvoir gérer plusieurs transactions simultanément (ce qui n'est pas le cas du mécanisme basé sur les pages ombres).

Ce mécanisme de journalisation convient donc mieux aux cartes «évoluées», aptes à gérer plusieurs transactions simultanément.

1.5 Choix du mécanisme de reprise sur panne

Dans cette section, nous allons définir les méthodes de reprise sur panne les mieux adaptées à chaque type de problème. La première nuance se fera autour de la carte en tant que serveur, ou en tant que client.

Dans le cas d'une carte rendant des services, nous étudierons dans un premier temps, le cas le plus simple : Celui de rendre les cartes actuelles tolérantes aux pannes. Ensuite, nous étudierons les mécanismes à mettre en jeu dans le cas des nouvelles applications cartes (ces mécanismes auront à gérer plusieurs transactions simultanément).

Dans le cas d'une carte cliente, nous étudierons les différentes complexités de transactions (sur une ou plusieurs sessions, et sur une ou plusieurs connexions).

1.5.1 Comparaison des deux mécanismes

Sans revenir dans les détails des deux mécanismes de reprise sur panne (que nous avons étudié précédemment), le tableau III.1.1 récapitule les principales caractéristiques du mécanisme de journalisation et des pages ombres.

Il apparaît dans ce tableau, que lorsque la carte à microprocesseur est équipée de mémoire Flash, le mécanisme des pages ombres se révèle très coûteux en terme de place en RAM (ceci est dû à la nécessité de reconstruire la table d'indirection en RAM) : pour une taille de mémoire de données de 16Ko, et une granularité de page de 64 octets, la table prend une place de 256 octets.

Concernant le surcoût en terme de nombre d'écritures sur le support non volatile (qui se révèle très pénalisant en terme de temps d'exécution), le mécanisme de journalisation est rapidement plus pénalisant que le mécanisme des pages ombres.

Le coût engendré par une validation de transaction est relativement léger dans les deux cas, puisqu'il consiste en la réécriture sur le support non volatile des parties modifiées de la table d'indirection pour les pages ombres, et qu'il se révèle sans coût pour le mécanisme de journalisation.

A l'inverse, le coût d'une annulation de transaction est important pour la méthode de journalisation, car il faut alors parcourir le journal en sens inverse, pour réécrire les anciennes valeurs des données modifiées par la transaction. Pour la méthode des pages ombres, le coût d'une annulation de transaction est quasi nul.

Enfin, la dernière différence entre ces deux mécanismes porte sur la possibilité de gérer ou non plusieurs transactions simultanées : cela est impossible avec le mécanisme des pages ombres.

	Pages Ombres	Journalisation
Coût RAM	Flash: >Taille Mem. / Taille Page EEPROM: 1 à 4 Octets	2 Octets
Nombre Ecriture	N+Cste	2N
Coût à la validation	Max: Table d'Indirection	0
Coût à l'annulation	ϵ	Nb Ecritures
Applications simultanées	NON	OUI

TAB. III.1.1 - *Etude comparée des mécanismes de reprise sur panne carte*

1.5.2 Cartes Actuelles

Nous avons vu dans le chapitre précédent, que pour rendre les cartes actuelles tolérantes à l'arrachement, et d'une manière plus générale, tolérantes aux pannes, celles-ci nécessitent un mécanisme de reprise sur panne.

Ces cartes sont mono-applicatives, et utilisent généralement des technologies simples et éprouvées (carte avec microprocesseur 8 bits, peu de mémoire RAM, et peu d'EEPROM).

Le mécanisme le mieux adapté à ce style de carte est sans conteste le mécanisme des pages ombres car:

- dans le cas d'une implantation sur une carte contenant de l'EEPROM, ce mécanisme utilise peu de RAM,
- il est adapté aux cartes capables de gérer une seule application à la fois,
- le surcoût en temps est faible, tant à la validation, qu'en cours d'exécution.

1.5.3 Cartes Multi-services

Dans le cas d'une carte pouvant être amenée à gérer plusieurs transactions simultanément (sur une ou plusieurs connexions), le mécanisme des pages ombres ne peut pas être appliqué (car il ne peut y avoir qu'une seule table d'indirection active à la fois).

Le mécanisme à utiliser est donc celui de la journalisation.

Les défauts de ce mécanisme (beaucoup de place utilisée sur le support non volatile, et coût en nombre d'écritures) sont compensés par le fait qu'il se destine à des cartes évoluées et «puissantes».

1.6 Mécanisme de contrôle de concurrence pour carte

De même que nous avons différencié plusieurs niveaux de complexité pour les mécanismes de reprise sur panne, nous allons étudier plusieurs niveaux pour les problèmes de contrôle de concurrence.

Cependant, comme nous l'avons dit précédemment, il n'y a pas de problème de concurrence dans les applications actuelles des cartes à microprocesseur. Nous allons donc étudier les mécanismes de contrôle de concurrence pour les applications les plus complexes des cartes à microprocesseur décrites dans le chapitre II.1 de ce document (applications multi-services, ou applications longues). Ce mécanisme concerne donc des cartes évoluées, capables de traitements et de stockage importants (ces cartes ne sont pas sur le marché actuellement).

1.6.1 Principes

Une carte contient relativement peu de données, et elle est souvent spécialisée (carte monétaire, carte santé, etc...). Le degré de contention sur les données de la carte à microprocesseur est donc potentiellement élevé.

Pour cela, nous avons décidé, dans un premier temps, d'intégrer dans les cartes un contrôle de concurrence continu lors de l'exécution d'une transaction (contrôle de concurrence pessimiste). Dans ces méthodes de contrôle de concurrence pessimiste, nous avons étudié deux techniques :

- L'estampillage, qui est relativement simple à implémenter (car il de pose pas de problème d'interblocage) mais qui risque de provoquer plus d'abandon en cascade, et un risque de privation pour une transaction (elle doit alors être sans arrêt relancée),

- Le verrouillage à deux phases donne un meilleur temps de réponse que la méthode d'estampillage, si la probabilité de conflit est forte, et si le nombre de requêtes est petit [**M82**], mais la table des verrous utilise beaucoup de place mémoire.

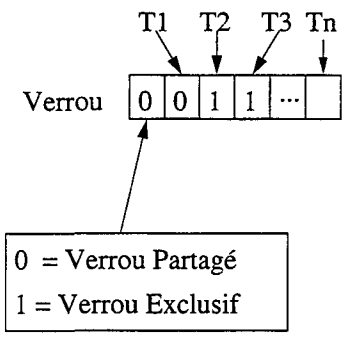
1.6.1.1 Le verrouillage à deux phases

Ce mécanisme de contrôle de concurrence pose deux problèmes majeurs dans le cas d'une carte à microprocesseur :

- Pour implanter le mécanisme de verrouillage à deux phases dans la carte, il faut tenir à jour une table des verrous (cf. Figure III.1.5). Le peu de place disponible, nous oblige à émettre des limites sur le nombre de transactions simultanées

dans la carte à microprocesseur. Cependant, ces limites sont temporaires, et seront levées au fur et à mesure de l'évolution de la technique carte.

- Ensuite, nous allons nous heurter aux traditionnels problèmes liés au verrouillage à deux phases : l'interblocage possible des transactions, ainsi qu'une libération des verrous après la fin d'une transaction, relativement lourde à gérer.



	Libre	0	0	0	0	...
WRITE	Ecriture(T1)	1	1	0	0	...
	Ecriture(T2)	1	0	1	0	...
	Ecriture(T3)	1	0	0	1	...
	⋮	⋮		⋮		
READ	Lecture(T1)	0	1	0	0	...
	Lecture(T2)	0	0	1	0	...
	Lecture(T3)	0	0	0	1	...
	Lecture(T1T2)	0	1	1	0	...
	Lecture(T1T3)	0	0	1	1	...
	Lecture(T2T3)	0	1	0	1	...
	Lecture(T1T2T3)	0	1	1	1	...
	⋮	⋮		⋮		

FIG. III.1.5 - *Exemple de table des verrous pour une carte gérant jusqu'à trois transactions*

La figure III.1.5 donne un exemple d'implantation de table des verrous dans la carte : les N premiers bits identifient la (ou les) transaction(s) qui verrouillent une donnée, alors que le dernier bit indique le type du verrou (Partagé ou Exclusif). Un verrou dont la valeur est nulle signale une variable libre.

A chaque lecture, ou écriture durant une transaction, il suffit de consulter le verrou correspondant, pour déterminer si l'on peut, ou non, effectuer l'opération.

A la fin d'une transaction, il faut parcourir toute la table des verrous pour lever les verrous posés par la transaction qui se termine. Une transaction en attente pour cause de blocage peut alors redémarrer (grâce au tableau II.1.2).

De manière à faciliter la libération des données après la fin d'une transaction, nous avons choisi d'utiliser une table globale des verrous, où sont stockés tous les verrous. Pour permettre l'exécution d'une transaction sur plusieurs connexions de la carte à microprocesseur, cette table doit être sur le support stable (ce qui se révèle extrêmement coûteux pour une carte munie de mémoire non volatile de type FlashRAM, ayant une granularité d'écriture de 64 octets).

Chaque granule devra donc contenir l'indice de positionnement de son verrou dans la table globale, de manière à y accéder facilement, soit pour le changer, soit

pour le tester.

La méthode de contrôle de concurrence basée sur le verrouillage à deux phases limite le nombre de reprises de transaction dans la carte, car elle permet la mise en attente d'une transaction. Cependant, elle est relativement lourde et complexe à implanter dans les cartes à microprocesseur.

Procédure de prise du verrou

Lorsqu'une transaction veut accéder à une donnée, elle doit commencer par tester si le verrou qu'elle veut poser sur cette donnée est compatible avec celui déjà existant.

Si c'est le cas, alors le verrou de la donnée est complété (dans le cas d'un accès en lecture) par une opération logique (un OU) entre le verrou posé par la transaction et celui existant, ou bien il est remplacé (dans le cas d'un accès en écriture) par le verrou en écriture de la transaction.

Sinon, la transaction doit être mise en attente. Pour cela, elle stocke dans sa table d'état, l'indice de la (ou des[8]) transaction(s) qui la bloque. L'exécution de la transaction est donc suspendue.

Indice	Identifiant	Etat
T1, T2 ou T3	T_Id	Soit l'indice de la transaction qui bloque Soit l'état de la transaction

TAB. III.1.2 - *table d'état des transactions en cours*

Procédure de libération des verrous

La validation d'une transaction (ou son annulation) entraîne la libération des variables qu'elle avait verrouillées. Pour cela, il faut parcourir toute la table, en faisant une opération logique entre chaque verrou et le masque de verrouillage de la transaction[9], pour supprimer le verrou de la transaction terminée. Pour cette opération, la table des verrous doit être chargée dans le cache. Ainsi, on est certain :

- Que la phase de libération des verrous sera plus rapide (car effectuée en RAM)
- Que la mise à jour de la table sera atomique, car la nouvelle table deviendra persistante au vidage du cache.

Pour chaque verrou, l'algorithme est le suivant :

```
Si (Verrou & masc != 0) Alors          \\ Si la transaction a le verrou
  Si (Verrou & masc_write != 0) Alors \\ Et que ce verrou est en Ecriture
      verrou = 0                       \\ Le verrou devient nul
  Sinon
      verrou = verrou XOR masc         \\ Sinon, il est en lecture
```

8. En effet, plusieurs transactions peuvent être en train d'accéder la donnée en lecture, alors que la nouvelle transaction veut l'accéder en écriture

9. Ce masque vaut par exemple 0100... pour T1, 0010... pour T2 et 0001... pour T3

```
  FSI
FSI
```

Après la libération des données détenues par la transaction terminée, il se peut qu'une transaction mise en attente puisse reprendre (si le temps qui lui était imparti n'est pas dépassé).

Pour retrouver facilement la transaction potentiellement bloquée, nous avons défini une table d'état des transactions en cours. Cette table est décrite dans le tableau III.1.2

Ainsi, après la fin d'une transaction, le système vient mettre à jour cette table (c'est-à-dire supprimer la transaction terminée), et consulte l'état des transactions restantes[10]. Si une transaction est bloquée par la transaction terminée, elle est relancée.

La colonne Etat est aussi un bon moyen de détection des interblocages entre les transactions, par détection de cycle. En effet, lorsqu'une transaction est bloquée, elle stocke dans la colonne Etat l'indice de la transaction qui la bloque. Il est donc possible de lancer à ce moment une détection de cycle, en allant lire le contenu de la colonne Etat, de la transaction bloquante, etc...

Remarques sur la taille de la table

La taille de la table des verrous dépend de deux critères :

- Le nombre de transactions simultanées envisagé. Par exemple, si on estime à trois le nombre de transactions maximal en cours simultanément, la taille d'un verrou sera de 4 bits.
- Le nombre de verrous. Ce nombre de verrous dépend du nombre d'applications installées dans la carte à microprocesseur (ainsi que de la granularité de verrouillage choisie).

Mis à part une occupation de place plus importante, la gestion d'un nombre important de transactions simultanées dans la carte à microprocesseur est tout à fait envisageable. Cependant, plus ce nombre sera important, plus la table sera grande, et plus la libération des verrous sera longue à la fin d'une transaction (car il faut alors parcourir toute la table).

1.6.1.2 La technique d'estampillage

La technique d'estampillage a comme principale qualité d'être simple à implémenter : pour tout objet A on définit deux estampilles, une en lecture (L(A)) et une en modification (M(A)). Chaque transaction T qui débute, se voit attribuer une valeur d'estampille V(T) (la distribution des estampilles suit une relation d'ordre).

Ainsi, pour l'accès en lecture à un objet A par une transaction T, l'algorithme est le suivant :

```
Si V(T) > M(A) alors        \\ Si l'objet n'est pas modifié par une
```

10. Nous verrons dans le chapitre suivant que le champ Etat peut aussi contenir la localisation du coordinateur de validation en cas de transaction distribuée

BU LILLE I

```
                          \\ transaction plus récente
   L(A) = max (V(T),L(A)) \\ T a les droits d'y accéder
  sinon
   Abandon de T           \\ sinon T doit être abandonnée
fin si
```

Et pour l'accès en modification :

```
si V(T) > max(M(A),L(A)) alors  \\Si A est n'est pas accédé par
                                \\une transaction plus récente
   M(A) = V(T)                  \\T a le droit de modification
  sinon
   Abandon de T                 \\ sinon T doit être abandonnée
fin si
```

Il peut paraître lourd de gérer systématiquement deux estampilles. Cependant, ces estampilles seront plus compactes que les verrous utilisés dans le cas du verrouillage à deux phases (une estampille de 4 bits permet de gérer jusqu'à 16 transactions simultanées).

Les principaux avantages de cette méthode par rapport au verrouillage à deux phases sont :

- **La facilité de mise en oeuvre :** il n'y a plus de problème à la validation de la transaction (libération des verrous dans le mécanisme de verrouillage à deux phases)
- **Pas d'interblocage possible :** Le problème d'une transaction bloquée indéfiniment par une autre ne se pose pas dans le cas de l'utilisation de l'estampillage.

Cependant, cette méthode implique des contre-parties pénalisantes :

- L'estampillage fixe inutilement des dépendances entre les transactions, ce qui entraîne donc des abandons non justifiés de transactions,
- Il ne faut pas oublier aussi la possibilité pour une transaction de redémarrer indéfiniment sans jamais être terminée.

1.6.2 Choix pour la carte à microprocesseur

La complexité de l'implémentation (notamment à la validation), ainsi que le coût d'utilisation (en terme de taille et de temps) semble être en faveur de l'estampillage dans le cas de la carte à microprocesseur.

Cependant, cette technique crée des dépendances plus fortes entre les diverses transactions s'exécutant simultanément. Ce qui nous amène à penser que, dans le cas de la carte à microprocesseur, où les données sont peu nombreuses, et à fort risque de concurrence, le nombre de transactions abandonnées sera plus important avec la méthode d'estampillage.

Au final, l'évaluation et la comparaison de ces deux mécanismes peuvent seulement se faire en terme de place mémoire et de surcoût en temps d'exécution, car nous ne disposons pas actuellement d'exemple d'implantation, et d'utilisation de telles applications dans les cartes.

1.7 Mise en Pratique : JavaCard et Transaction

La JavaCard est une carte à microprocesseur dédiée au développement rapide d'applications, en diminuant pour le programmeur les contraintes liées aux cartes classiques.

Un mécanisme capable d'assurer la prise en compte de tous les types de panne pour que le contenu de la carte demeure dans un état cohérent, serait donc le bienvenu.

Pour cela, nous avons étudié les possibilités d'intégration d'un gestionnaire de ressources assurant les propriétés du modèle transactionnel dans cette carte.

1.7.1 Architecture d'un Système Transactionnel pour JavaCard

L'exécution d'une application par une JavaCard peut échouer pour plusieurs raisons :

- Une mauvaise manipulation de l'utilisateur (par exemple, un arrachement de la carte à microprocesseur au cours de l'exécution de l'application)

- La définition de nouvelles variables peut provoquer un débordement de pile d'exécution.

- La création de nouveaux objets persistants (par exemple une trace d'exécution) peut provoquer une erreur d'allocation de mémoire persistante (par exemple, pas assez de place sur le support).

Le développeur de l'application doit donc prendre en compte toutes ces pannes possibles, pour que son application ne laisse pas la carte dans un état incohérent après son exécution[11].

Une solution est d'intégrer à cette carte à microprocesseur un mécanisme de reprise sur panne prenant en charge la gestion de ces erreurs d'exécution. Les ordres permettant l'implantation d'un tel mécanisme sont prévus dans l'API 2.0 de la JavaCard [**JC97**] (Begin_Transaction, Commit_Transaction, Abort_Transaction).

En ce qui concerne les besoins de contrôle d'accès concurrents aux données, les JavaCards actuelles ne sont pas encore suffisamment évoluées pour permettre l'exécution de plusieurs transactions. Il n'est donc pas encore nécessaire d'intégrer un tel mécanisme dans ces cartes.

11. Une mauvaise gestion de ces pannes crée des entrées pour des fraudes possibles

1.7.2 Applications

Les cartes à microprocesseur actuelles disposent de très peu de mémoire volatile (1 Ko maximum). Ainsi, pour une implantation d'un mécanisme de reprise sur panne sur les JavaCards commercialisées actuellement, nous sommes donc obligés de sélectionner les solutions les moins coûteuses en RAM.

Ainsi, nous obtenons deux possibilités :

- Si la carte est équipée d'un support non volatile de type EEPROM, la meilleure solution en terme de mécanisme de reprise sur panne sera le mécanisme des pages ombres, car il est le moins coûteux en terme de performance.

- A l'inverse, si la carte à microprocesseur est équipée de mémoires non volatiles de type flashRAM, la méthode de journalisation s'impose, car on ne peut pas réserver 256 octets de RAM pour la table d'indirection.

1.8 Limites du modèle transactionnel strict

Dans certains cas, l'utilisation du modèle transactionnel strict pose problème. En effet, le respect des propriétés ACID des transactions peut aboutir à un blocage des applications de la carte à microprocesseur.

Dans cette section, nous allons donc, dans un premier temps, étudier une application pour laquelle se posent des problèmes lors de l'utilisation d'un modèle transactionnel strict. Ensuite, nous proposerons une ébauche de solution à ce problème, basée sur la prise en compte de la sémantique des opérations.

1.8.1 Le cas de l'application GSM-PME-CB

Comme nous l'avons dit à plusieurs reprises, les cartes à microprocesseur contiennent peu de données, mais ces données sont très sensibles, et peuvent être fréquemment accédées de manière concurrente.

Dans certains cas, un simple mécanisme de contrôle de concurrence suffira pour assurer la sérialisation des transactions impliquées, mais il y a aussi des cas où ces mécanismes conduirons systématiquement à une annulation de la transaction. Nous allons décrire ici le cas d'une application qui nécessite un contrôle de concurrence plus élaboré qu'un simple contrôle strict pour aboutir.

Prenons le cas d'une application distribuée, basée sur un réseau de communication cellulaire, mettant en jeu les partenaires suivants :

- L'utilisateur du téléphone (c'est-à-dire de la carte SIM contenue dans ce téléphone), qui dispose plusieurs services chargés sur une carte à microprocesseur.

- L'opérateur de téléphonie mobile, qui fournit un accès payant au réseau. Dans le cas de l'application décrite ici, le montant de la communication sera directement prélevé sur le PME du client[12] présent dans la carte.

12. Dorénavant, les prestataires de services GSM fournissent tous des formules d'accès au réseau sans abonnement, basés sur le rechargement d'un compte type PME

- Un fournisseur de services, ou de biens, que l'utilisateur peut contacter pour effectuer des achats au moyen du PME chargé dans sa carte multi-services.

Le déroulement de l'application qui pose problème avec le modèle transactionnel est le suivant :

- Dans un premier temps, l'utilisateur ouvre une connexion sur le réseau. Les coûts relatifs à cette connexion seront débités directement sur le PME présent dans la carte à microprocesseur.
- Une fois connecté au réseau, l'utilisateur désire acheter un objet (ou un service), en le payant avec son PME.
- S'il ne reste plus assez d'argent sur le PME, l'utilisateur doit être en mesure de contacter sa banque, pour recharger le PME avec son service «Carte Bancaire».
- Une fois le rechargement effectué, l'utilisateur peut poursuivre son achat, et terminer sa connexion.

Il est impossible d'exécuter une telle application avec l'architecture que nous venons de décrire dans ce chapitre, car il y a de multiples accès simultanés sur le PME de la carte à microprocesseur.

Pourtant, ce type d'application va devenir courant dans les années à venir, et devra être géré facilement par le programmeur. Il convient donc de fournir un mécanisme (de type transactionnel), en offrant une possibilité de sérialisation des transactions plus élevée sur certains types d'applications.

1.8.2 Problématique

Les méthodes de contrôle de concurrence les plus répandues ne permettent pas une forte sérialisation des transactions. Dans le cas d'un PME sur une carte à microprocesseur cela se révèle problématique, car il est souvent le centre des ressources de la carte.

En cas de verrouillage du PME par une transaction, une partie non négligeable des services de la carte peut se retrouver bloquée, alors que les ressources nécessaires sont encore présentes dans la carte[13].

1.8.3 Prise en compte de la commutativité des opérations

Une solution pour autoriser une plus grande sérialisation des transactions est de prendre en compte la sémantique des objets accédés [**CFR89**], de manière à établir des propriétés de commutativité sur les actions typées[14].

Il existe de règle de commutativité [**CFG94**]:

- **La commutativité en arrière :** dans le cas de modification directe du support au cours de l'exécution de la transaction, deux opérations *op1* et *op2*,

13. Il est difficilement concevable de bloquer un PME contenant 1000 FF pour une transaction portant sur 50 FF

14. Deux opérations sont en conflit si elles ne commutent pas

exécutées par T1 et T2 (dans l'ordre T1:op1;T2:op2) commutent en arrière si, quelque soit l'état initial de la base, l'exécution dans l'ordre inverse (c'est à dire T2:op2;T1:op1) a le même effet sur la base.

- **La commutativité en avant :** dans le cas où la base n'est pas modifiée tans que les transaction ne sont pas validées, deux opérations *op1* et *op2* exécutées par T1 et T2, vont utiliser en entrée le même état de la base. On peut dire que ces deux opérations commutent en avant, si les deux exécutions (T1:op1;T2:op2 et T2:op2;T1:op1) ont le même effet sur la base

Le modèle que nous avons choisi pour les cartes à microprocesseur, dans le cas des cartes multi-services, consiste à utiliser un mécanisme de reprise sur panne basé sur la journalisation des images avant des données. La mise à jour du support stable se fait donc au fur et à mesure de l'exécution de la transaction.

Dans le cadre des PME, on peut reprendre les tables de commutativité pour les compteurs et pour les objets de type compte (avec les opérations de débit et crédit), qui ont déjà été étudiées **[B95]** (cf. Tableau III.1.3 : les lignes représentent l'opération de T1, et les colonnes l'opération de T2).

	Crédit(x,V2)	**Débit(x, v2, ok)**	**Débit(x, v2, non)**
Crédit(x, v1)			
Commute en avant	OUI	NON	OUI
Commute en arrière	OUI	OUI	NON
Débit(x,v1,ok)			
Commute en avant	OUI	OUI	NON
Commute en arrière	OUI	NON	OUI
Débit(x,v1,non)			
Commute en avant	NON	OUI	OUI
Commute en arrière	NON	OUI	OUI

TAB. III.1.3 - *commutativité des actions sur un compte* ***[B95]***

Avec l'utilisation d'un protocole de concurrence pessimiste, comme le protocole de verrouillage à deux phases, dans certains cas, on ne peut pas connaître le résultat de l'opération de la seconde transaction, avant de l'avoir exécutée. Dans le cas où ces résultats sont indispensables (ici, dans le cas ou la deuxième transaction effectue un débit) l'opération doit être effectuée dans un espace de travail privé, pour pouvoir tester la commutativité.

Si on se contente de considérer la commutativité en arrière, il y a encore beaucoup de scénario où les actions ne commutent pas. Une nouvelle de commutativité, appelée en Avant-Arrière permet d'améliorer cela **[GFP95]**.

Une telle prise en compte de la sémantique de l'objet accédé rend possible l'application décrite précédemment, car les conflits se trouvent réduits.

1.9 Débordement des données à l'extérieur de la carte

Les architectures décrites dans ce chapitre ont été adaptées à la carte à microprocesseur. Cependant, il est possible qu'elles conduisent à un remplissage total plus rapide de la mémoire de la carte (notamment suite à l'utilisation de journaux). Nous proposons donc de reprendre le principe, utilisé dans l'informatique mobile [C98] d'un représentant externe, comblant les lacunes de la carte à microprocesseur [LT97].

La solution proposée pour éviter les problèmes de remplissage de la carte à microprocesseur consiste à externaliser une partie des données de la carte à microprocesseur vers un serveur de données (la carte contient alors uniquement le pointeur vers ces données) [V97].

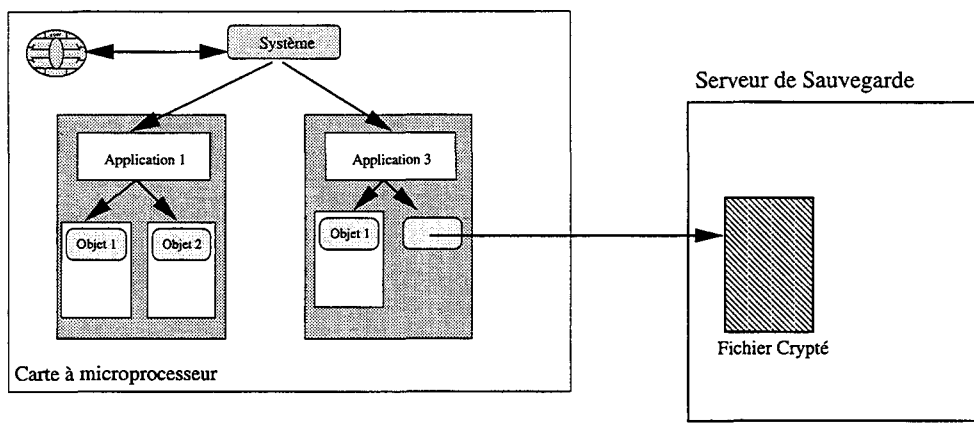


FIG. III.1.6 - *Externalisation des informations carte*

[B98b] propose un protocole sécurisé pour stocker de manière sûre des données cartes sur des serveurs externes (cf. Figure III.1.6), en cryptant les informations. Cette technique peut être utilisée de manière à sauvegarder en dehors de la carte à microprocesseur les informations les moins sensibles (comme les traces d'exécution de certaines applications). Ainsi, la place sur le support volatile, qui est diminuée lors de l'utilisation de mécanismes de reprise sur panne, est virtuellement étendue.

1.10 Conclusion

L'intégration, dans les cartes à microprocesseur, de mécanismes de reprise sur panne et de contrôle de concurrence ne peut pas se faire en définissant des solutions «définitives» : Chaque type de carte, et chaque application aura une solution optimale différente.

Ainsi, si l'on prend le cas des mécanismes de reprise sur panne, pour une carte mono-applicative, équipée d'un support non volatile de type EEPROM, la solution optimale sera un mécanisme basé sur le principe des pages ombres (faible coût en

RAM, et faible coût en nombre d'écritures). A l'inverse, si la carte est munie d'un support de type FlashRAM, ce mécanisme des pages ombres s'avère trop coûteux en occupation de mémoire volatile.

En ce qui concerne les cartes à microprocesseur plus évoluées, qui devraient voir le jour d'ici quelques années[15], la solution optimale consiste en un mécanisme de reprise sur panne basé sur un journal des images avant des données, alors que le mécanisme de contrôle de concurrence pourra être basé sur le verrouillage à deux phases, tout en tenant compte de la sémantique de certains objets (par exemple un PME).

Une évaluation plus précise des coûts de chaque mécanisme en fonction de l'application utilisée et du type de mémoire est encore en cours actuellement [**DGL98**]. Pour parfaire cette évaluation, il faut notamment estimer le nombre de pages de mémoire accédées par les applications, ainsi que leur localisation. Ce travail s'effectue maintenant dans le cadre d'une réflexion plus générale sur la modularité des systèmes d'exploitation pour carte à microprocesseur, et sur l'interdépendance des divers composants de ces systèmes d'exploitation[16].

15. On parle ici des cartes multi-services ayant de grosses capacités de stockage et de calcul
16. Ces travaux sont effectués par G. Grimaud au sein du laboratoire RD2P (projet CAMILLE)

Chapitre III.2

La carte transactionnelle dans un contexte distribué

2.1 Introduction

Dans le chapitre précédent, nous avons défini ce que devait être une carte à microprocesseur capable de résister aux pannes, et d'exécuter localement des transactions. Nous allons dans ce chapitre, décrire comment cette carte peut participer à des transactions distribuées sur le réseau.

Après avoir revu les principes d'intégration des cartes à microprocesseur dans les systèmes distribués, nous étudierons les solutions permettant de mieux intégrer les cartes à microprocesseur en tant que serveurs dans les systèmes transactionnels existants. Puis, à partir des contraintes technologiques carte, nous proposerons un ensemble de composants logiciels (basés sur l'utilisation d'une représentation permanente sur le réseau de la carte à microprocesseur). Ces composants logiciels devront permettre à la carte de rechercher les informations lui manquant après une panne afin de décider correctement de la terminaison de la transaction en cours d'exécution.

Enfin, nous conclurons ce chapitre en étudiant la possibilité de réaliser une carte cliente, utilisant des services distribués sur le réseau (de manière à mener à bien une application) ou même des cartes prenant en charge le protocole de validation des transactions distribuées.

2.2 La carte à microprocesseur serveur transactionnel

Le but de l'architecture proposée dans cette section, est de rendre disponibles dans le système distribué les services détenus par la carte à microprocesseur, pour qu'ils puissent prendre part à une transaction distribuée. Dans cette section, nous allons donc uniquement considérer la carte à microprocesseur comme un serveur transactionnel, recevant des requêtes d'un client distant.

Nous considérons dans cette section, que la carte offrant ses services sur le réseau dispose d'un gestionnaire de ressources transactionnel, tel que ceux étudiés dans le chapitre précédent, de manière à assurer un mécanisme de reprise sur panne, et un contrôle des accès concurrents adéquats.

2.2.1 Choix de l'environnement

Les nouveaux systèmes d'exploitation pour carte à microprocesseur sont développés avec une méthodologie orientée objet. Pour faire communiquer des objets distants entre eux, trois environnements de développement d'applications distribuées sont actuellement en plein développement :

- Microsoft fourni DCOM, un mécanisme de communication entre objets, pour plate-forme de type PC, munie de systèmes d'exploitation Microsoft.

- SUN, avec les RMI[1] [**JAV97**], propose un mécanisme d'invocation d'objets Java distants.

- CORBA, l'environnement de l'OMG, a comme principal avantage d'assurer l'interopérabilité entre différentes plates-formes, et de proposer de nombreux services (Transactions, Sécurité, Nommage) [**OMG97c**].

Nous avons choisi de proposer une architecture basée sur l'environnement CORBA, car il est de plus en plus utilisé dans les systèmes Client / Serveur (Sun propose un ORB dans sa dernière version du Kit de Développement Java [**JAV98**]), et de nombreux outils et interfaces ont déjà été spécifiés.

Cependant, les concepts définis ici seront transposables dans une architecture DCOM munie du système transactionnel de Microsoft (MTS).

2.2.2 Définition de l'architecture

Notre architecture d'intégration des cartes dans un environnement distribué, pour participer en tant que serveur à des transactions, est donc basée sur l'environnement CORBA, muni de son service transactionnel OTS.

Pour plusieurs raisons, la solution idéale serait l'intégration dans la carte à microprocesseur d'un ORB muni d'un service transactionnel (cf. figure III.2.1). En effet,

1. RMI : Remote Method Invocations

une telle architecture aurait plusieurs avantages :

- **Communication entre services carte :** une carte à microprocesseur contenant un bus logiciel de communication permettrait aux services qu'elle contient de communiquer facilement entre eux, par envoi de requêtes.
- **Un coordinateur de confiance :** l'architecture logicielle définie par la figure III.2.1 permet d'intégrer à l'intérieur de la carte un gestionnaire de transactions. En effet, nous avons vu précédemment que, pour sécuriser la transaction, la validation devait être confiée à un coordinateur de confiance. L'intégration du mécanisme de validation dans la carte, qui est un support ultra-sécurisé, permet d'obtenir cette confiance.
- **Une carte indépendante du monde extérieur :** Les communications entre la carte et le monde extérieur sont limitées au strict minimum, puisque toutes les procédures d'enregistrement auprès de l'OTS se font en interne à la carte.
- **Interopérabilité avec d'autres composants :** En cas de besoin de coopération avec un autre OTS, la carte peut utiliser la technique d'interposition, pour placer son OTS en maître du second OTS.

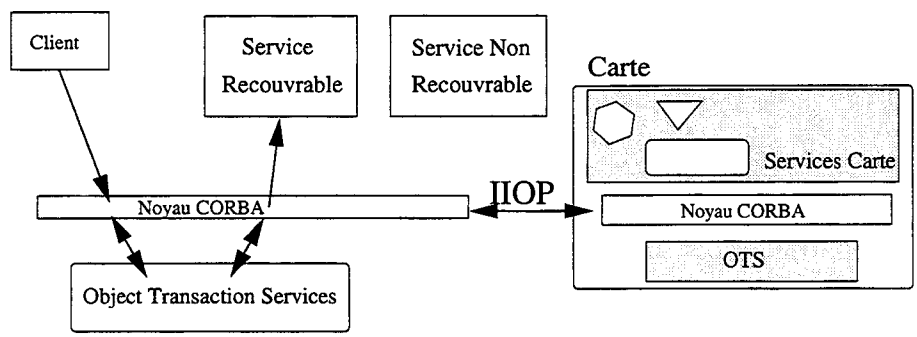


FIG. III.2.1 - *Cartes et OTS : Architecture Idéale*

Cependant, une telle architecture se heurte à de nombreux problèmes, qui la rendent difficilement réalisable dans les années à venir :

- Les interactions entre plusieurs ORB sont basées sur un ensemble de protocoles réseaux GIOP[2], dont une implémentation (IIOP[3]) repose sur le protocole TCP/IP, et permet à un objet situé sur Internet, sur n'importe quel ORB, de dialoguer avec d'autres objets distants. Le protocole de communication limité de la carte à microprocesseur pose donc problème pour ces communications.
- Les caractéristiques matérielles des cartes à microprocesseur (taille des mémoires, puissance des processeurs) permettent difficilement d'envisager l'intégration d'un ORB au sein d'un système d'exploitation carte.

2. General Inter-ORB Protocol
3. Internet Inter-ORB Protocol

De cette architecture idéale, nous avons donc extrapolé une architecture plausible, permettant aux cartes à microprocesseur de participer, en tant que fournisseurs de services, à une transaction distribuée (cf. figure III.2.2). Cette architecture passe par l'utilisation de plusieurs composants, nécessaires pour rendre disponibles les services de la carte sur le réseau, mais aussi pour permettre à la carte de participer à une transaction distribuée.

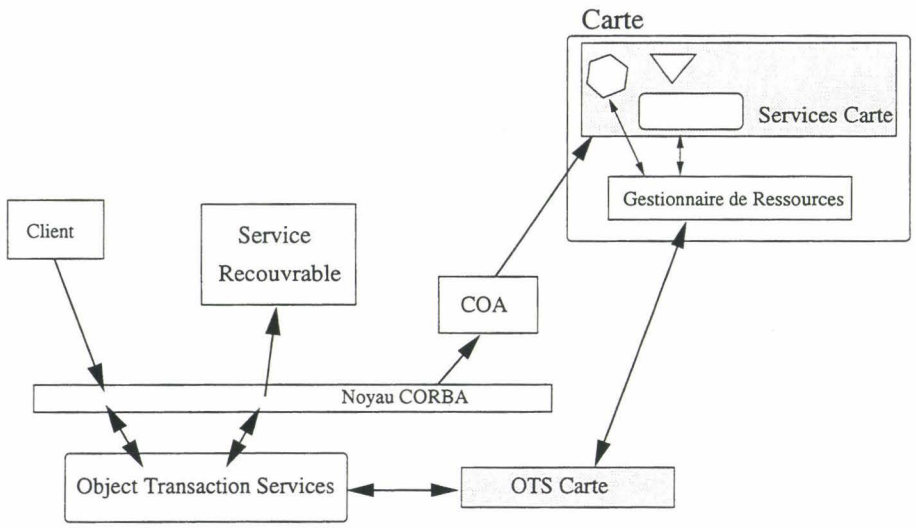


FIG. III.2.2 - *Cartes et OTS : Architecture Proposée*

- Le COA a ici le même rôle que dans [**V97**]. Il permet l'accès aux services de la carte, et la transcription des requêtes CORBA en requêtes Carte, et inversement. Le travail effectué par ce composant d'adaptation doit rester minimal (principalement pour des problèmes de sécurité).
- L'OTS carte, est un OTS classique, qui contient en plus une interface de communication vers les serveurs cartes. Ce service ne peut communiquer qu'avec des cartes ou un autre OTS.

Le fonctionnement de ces deux modules sera explicité dans la section suivante.

2.2.3 Choix de l'algorithme de validation distribuée

Le but de ce mémoire est de permettre à des services cartes de participer à des transactions distribuées dans les systèmes existants. La carte à microprocesseur doit donc s'interfacer avec le mécanisme de validation distribuée le plus couramment répandu (à savoir le protocole de validation à deux phases).

Ainsi, la carte (ou tout du moins son composant d'adaptation) doit être en mesure de répondre aux ordres du protocole de validation («Prépare» et «Commit» ou «Abort»). La carte doit aussi être en mesure de s'enregistrer auprès du gestionnaire de transaction distribuée.

Il est cependant important de noter que ce protocole n'est pas le mieux adapté au monde carte, car il est bloquant : si la carte est retirée après avoir voté (fin de la phase «Prépare»), mais avant avoir reçu la décision finale du coordinateur, ou si le coordinateur subit une panne, la carte se retrouve dans un état indécis, qui l'obligera à recontacter le coordinateur à la prochaine connexion.

Dans l'avenir, il serait intéressant de se pencher sur les nouveaux protocoles de validation à une phase [**AP97**], décrits dans le chapitre de présentation du modèle transactionnel.

2.3 Caractéristiques de l'OTS Carte et du COA

Dans cette section, nous allons revenir plus en détail sur les deux composants nécessaires à la participation des cartes à une transaction distribuée dans l'environnement CORBA (à savoir le composant d'adaptation pour la carte, le COA, et l'OTS).

Après avoir décrit les évolutions du COA, nous présenterons les schémas de communications, lors d'un accès, pendant une transaction, à un service carte, ainsi que lors de la validation de cette transaction.

Enfin, nous présenterons diverses architectures logicielles d'intégration des cartes à microprocesseur dans une transaction distribuée : l'utilisation d'un OTS générique, et celle d'un OTS spécialisé pour carte à microprocesseur.

2.3.1 Une évolution du COA...

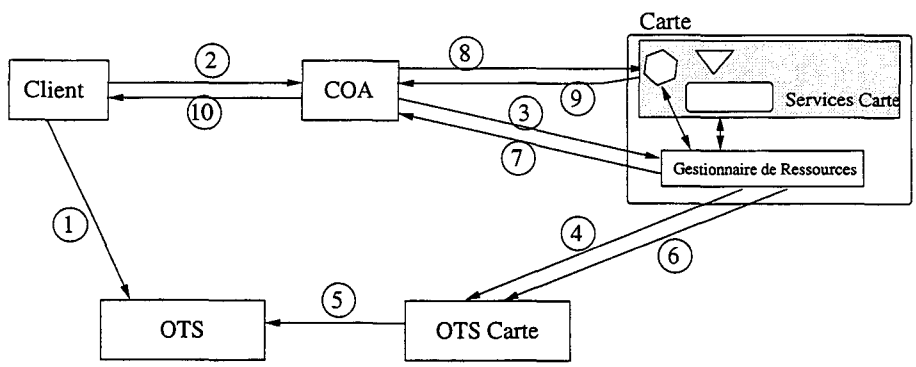


FIG. III.2.3 - *Schéma de communication avec technique d'interposition*

Le COA devra subir une légère modification(cf. Figure III.2.3), car il doit être en mesure de gérer les contextes transactionnels. En effet lorsqu'une requête contenant un identifiant de transaction est envoyée vers un service carte (2), le COA doit séparer une requête CORBA en plusieurs requêtes carte :

- le COA doit contacter le gestionnaire de ressources de la carte pour effectuer le «Begin_Transaction» dans la carte (3). Si celui-ci peut gérer une nouvelle transaction, il crée alors une nouvelle transaction auprès de l'OTS carte (4). A

ce moment là, l'OTS Carte va s'enregistrer auprès de l'OTS du client (5), et le serveur s'enregistrer auprès de son OTS (6). Le gestionnaire de ressources peut alors envoyer son accord pour la création de la transaction au COA (7).

- Après cela, le COA peut envoyer à la carte la requête vers le service demandé (8). Cette requête sera traitée par la carte en mode transactionnel, et la réponse (9) sera traduite par le COA pour le Client (10).

2.3.2 Protocoles de Communication

Le protocole de communication au début et à la fin d'une transaction distribuée est donc celui donné par la figure III.2.4.

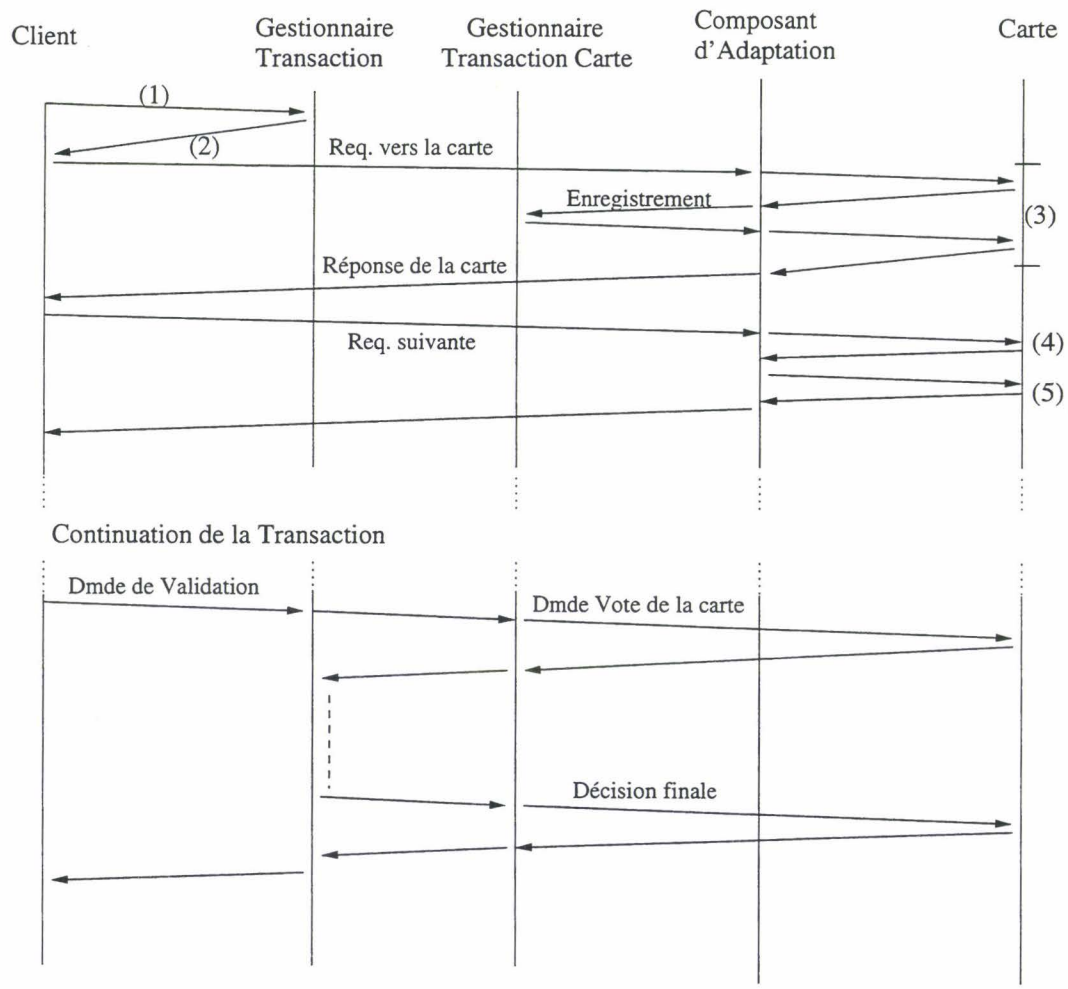


FIG. III.2.4 - *Protocole de Communication Carte / Gestionnaire de Transactions*

Un récapitulatif des ordres reçus par la carte, et des réponses correspondantes est donné dans le tableau III.2.1.

Tout d'abord, le client de la transaction distribuée commence par contacter son OTS pour démarrer la transaction (il effectue le «Begin_Transaction») (1). En ré-

Ordre	Provenance	Données	Réponse	Conséquence
Begin_Transaction	COA	T_Id	Enregistrement auprès de l'OTS Carte	La carte passe en mode transactionnel
Is_Registred[a]	COA	T_Id	OK ou NOK	Aucune
Prepare	OTS Carte	T_Id et l'IOR du Recovery Coordinator	Résultat du Vote	La carte se prépare à valider
Commit	OTS Carte	T_Id	OK	Valide les actions de la transaction
Abort	OTS Carte	T_Id	OK	Annule les actions de la transaction
GetRecovery Coordinator	COA	Aucun	Retourne l'IOR du Recovery Coordinator	Cf. Explication

TAB. III.2.1 - *Ordres reçus par la carte durant une transaction*

[a] Le COA peut tenir une table des transactions à jour, qui évite de consulter la carte, mais il sort alors de son rôle initial, qui est celui de traducteur de requêtes

ponse à cet ordre, l'OTS démarre une nouvelle transaction (il créé un nouveau contexte transactionnel), et renvoie l'identifiant[4] de la transaction au client (2).

Ensuite, le client effectue ses requêtes sur les divers serveurs, qui s'enregistrent auprès de leur OTS. Dans le cas du serveur carte, l'invocation du service carte, et l'enregistrement de ce service auprès de son OTS ont été explicités dans la partie 2.3.1 (3).

A chaque invocation d'un service carte, le COA remplit son rôle de traducteur de requêtes. Cependant, à chaque réception d'une requête dans le cadre d'une transaction, le COA va recevoir un contexte transactionnel. Il doit alors vérifier auprès de la carte que cette requête est déjà enregistrée auprès d'un OTS pour cette transaction (4), avant d'envoyer celle-ci au service carte (5).

A la demande de validation de la transaction par le client, la requête de lancement de l'algorithme du protocole de validation à deux phases est transmise à l'OTS carte par l'OTS du client. L'OTS carte effectue alors la phase de «Prépare» auprès du gestionnaire de ressources de la carte, qui décide alors de voter la validation, ou l'annulation de la transaction[5]. La décision de l'OTS Carte est alors transmise à

4. l'identifiant de la transaction est en fait une référence d'objet CORBA de 128 octets, ce qui représente 1/4 de la RAM disponible dans une carte à microprocesseur.

5. si les données de la carte n'ont pas subi de modification, le gestionnaire de ressources peut aussi voter ReadOnly.

l'OTS du client, qui lance alors la seconde phase du protocole de validation.

L'ordre GetRecoveryCoordinator sert à retrouver, sur le réseau, la décision prise par le coordinateur d'une transaction, pendant laquelle la carte a subi une panne et s'est retrouvée bloquée. Cet ordre renvoie l'IOR du Recovery Coordinator de la précédente transaction. Le stockage de la référence de cet objet est particulièrement utile en cas d'arrachement de la carte entre la phase de préparation et la phase de décision du protocole de validation à deux phases.

2.3.3 L'intérêt d'un OTS dédié aux cartes

Une autre architecture pourrait consister en l'utilisation d'un OTS générique pour la carte à microprocesseur, et à confier la transcription des requêtes de l'OTS au COA (cf. Figure III.2.5).

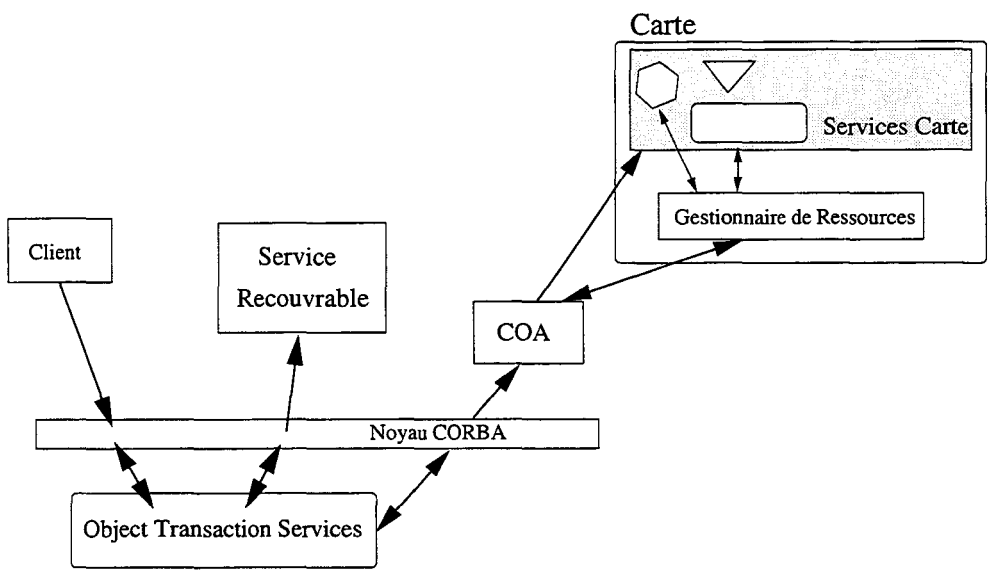


FIG. III.2.5 - *Utilisation d'un OTS Générique*

Cette architecture ne change pas fondamentalement les échanges décrits précédemment. Le principal avantage de cette dernière architecture, est que la carte dispose d'un interlocuteur unique sur le réseau : le COA.

Le rôle du COA dans cette dernière architecture est donc très important, car il doit non seulement effectuer la traduction des requêtes entre l'ORB et la carte, mais aussi assurer l'interface entre le gestionnaire de ressources de la carte, et le coordinateur de la transaction disponible sur le réseau.

Dans le cadre d'un déploiement à grande échelle, la solution mettant en jeu un OTS spécifique à la carte semble une meilleure solution, car :

- Cela permet à l'émetteur de cartes d'être assuré du bon fonctionnement de l'OTS auquel sera connectée la carte. En effet, un tel mécanisme de validation devra être présent sur les machines de connexion carte, et sera certifié par les émetteurs de carte.

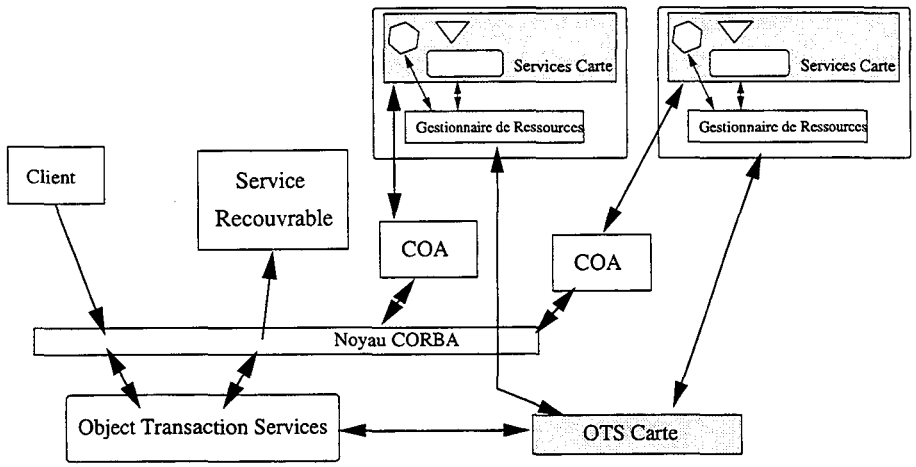


FIG. III.2.6 - *Architecture pour une application Multi-cartes*

- Définir un OTS spécifique à la carte permet de prendre plus facilement en compte les spécificités des cartes à microprocesseur dans le déroulement de l'algorithme de validation (identifiant de transaction plus léger, prise en compte des possibles pannes de la carte).
- Enfin, l'utilisation d'un OTS spécifique à la carte permet de séparer au niveau de la communication les ordres applicatifs des ordres de validation des transactions.

De plus, dans le cadre d'une application mettant en jeu plusieurs cartes (cf. Figure III.2.6), un seul OTS carte est nécessaire, alors que plusieurs COA sont utilisés (car ils sont propres aux services chargés dans la carte).

2.4 Tolérance aux pannes de cette Architecture

L'architecture d'intégration proposée nécessite un fonctionnement en mode connecté de tous les participants (et notamment de la carte à microprocesseur). Cependant il peut arriver que certains participants tombent en panne et disparaissent alors du réseau (par exemple, une carte arrachée).

Cette section à comme principal but, l'étude des conséquences, sur l'application distribuée, d'une panne carte, ainsi que des solutions proposées pour remédier à ces pannes.

2.4.1 Transaction distribuée et panne carte

Une panne carte diffère d'une panne d'un serveur classique, car il est impossible de prévoir sur quel terminal du réseau la carte va se reconnecter, et de connaître la durée de la déconnexion.

Les stratégies de gestion des pannes cartes vont donc se rapprocher des mécanismes de traitement des déconnexions de l'informatique mobile. Celle-ci utilise des

représentants permanents sur le réseau [**CLT96**], qui sont utilisés à plusieurs fins [**C98**] :

- **La prise en compte des faibles possibilités du mobile :** ces représentants sont utilisés pour effectuer des traitements trop complexes, ou pour adapter les données aux capacités du mobile.

- **La gestion du mode déconnecté :** Les représentants permanents des ordinateurs mobiles sont, comme leur nom l'indique, toujours présents sur le réseau. Ils peuvent donc recevoir les requêtes destinées au mobile pendant la durée où celui-ci est absent du réseau.

On peut considérer que le représentant de la carte sur le réseau à fait son apparition en tant qu'objet d'adaptation qui adapte les données pour la carte à microprocesseur (le COA). Nous allons maintenant faire évoluer cette représentation pour qu'elle soit en mesure de gérer les déconnexions des cartes.

2.4.2 D'un objet d'adaptation à une véritable représentation externe

Du point de vue du modèle d'exécution, le principe d'utilisation d'une représentation externe pour la carte à microprocesseur est relativement simple (cf. figure III.2.7) [**LT97**] :

- Lorsque la carte à microprocesseur n'est pas connectée au réseau, la représentation stocke les informations destinées à la carte (comme par exemple les décisions du coordinateur du protocole de validation distribué).

- Lorsque la carte est connectée au réseau, la représentation externe agit comme un objet d'adaptation des requêtes du réseau au protocole de communication de la carte (et inversement).

La gestion de cette représentation externe de la carte à microprocesseur implique un certain nombre de contraintes au niveau du modèle d'exécution de la carte :

- Lors de sa reconnexion, la carte doit aller contacter sa représentation externe sur l'ancien site où elle était connectée, pour lui indiquer sa nouvelle adresse, et transférer les données en attente. Une fois les files d'attentes vidées, la représentation est alors détruite et recréée sur le nouveau terminal de connexion de la carte à microprocesseur.

- Pendant l'exécution de l'application distribuée, la carte communique uniquement avec sa représentation. Cette dernière traduit et redirige les requêtes vers leur destinataire.

Le lieu idéal de stockage d'une telle représentation est le dernier terminal où la carte à microprocesseur s'est connectée, car il est connu de la carte, de tous ses interlocuteurs durant sa connexion et il est le premier prévenu de la déconnexion de la carte [**CLT97**]. Cependant, on doit s'assurer que ce terminal soit en permanence

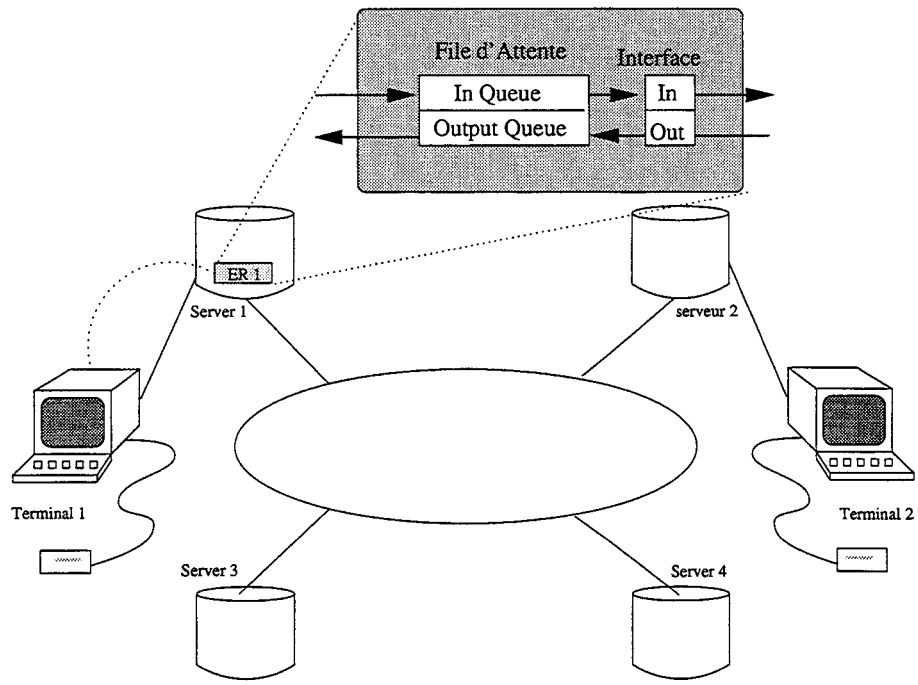


FIG. III.2.7 - *Utilisation d'une représentation externe pour la carte*

connecté. Si cela n'est pas le cas, la représentation devra être stockée en un autre lieu : Prenons l'exemple d'un téléphone cellulaire. La représentation de la carte ne doit pas être sur le terminal (c'est-à-dire le téléphone), car il est aussi souvent déconnecté que la carte qu'il contient. La représentation devra donc être stockée sur un serveur du prestataire de l'application de téléphonie mobile (qui gère le réseau mobile), et les requêtes destinées à la carte devront être routées vers ce serveur (selon les mêmes principes que les boites vocales). Ce mécanisme est déjà utilisé en téléphonie mobile pour stocker les SMS[6] qui sont envoyés au terminal téléphonique.

Ce type de représentation pose cependant un problème : une telle architecture impose de se connecter sur un terminal présent sur le réseau. Il peut donc arriver qu'une carte se connecte sur un terminal qui n'est pas en ligne. Il faut alors définir un modèle d'exécution dégradé de la carte à microprocesseur. Ce fonctionnement est directement fourni par le gestionnaire de ressources carte :

- Si la carte est en attente d'information concernant une transaction en cours dans la carte, le mécanisme de contrôle de concurrence va préserver les données de la carte tant que l'ordre de fin de la transaction ne lui est pas parvenu.

- Il se peut aussi que le lancement d'une nouvelle transaction soit dans la file d'attente de la représentation externe. Dans ce cas là, cette nouvelle transaction sera prise en compte lors d'une connexion ultérieure de la carte (si elle est encore valable).

6. SMS : Short Message System

2.4.3 Tolérance aux pannes du site accepteur de représentation externe

En cas d'arrachement de la carte à microprocesseur lors d'une transaction, il y a deux possiblités :

- Soit la carte est arrachée avant le début du protocole de validation distribuée. La carte subit alors un abort implicite de la transaction en cours, et la transaction est considérée comme terminée pour elle.

- Soit la carte est arrachée après le début du protocole de validation distribuée (c'est à dire après qu'elle ait voté la validation. Dans ce cas, la transaction est bloquée, et la carte devra alors conserver verrouillées les objets accédés par la transaction en cours (elle ne peut pas connaître l'issue de la terminaison de cette transaction).

De plus, le cas d'une panne du terminal lors de la précédente connexion carte doit être envisagé [**CLT97**]. Dans ce cas, la carte ne peut aller chercher les données stockées pour elle par sa représentation permanente, et dans le cas où l'exécution de la transaction est bloquée, elle ne pourra pas obtenir le résultat de la validation de cette transaction.

La carte devra stocker à la fois l'adresse du terminal de la nouvelle connexion, mais aussi celle du terminal en panne (de manière à refaire une tentative ultérieurement).

En cas de panne prolongée provoquant un blocage permanent de certains objets de la carte, une intervention extérieure (du prestataire qui a émis la carte) peut être requise.

2.4.4 Architecture globale des machines de connexion carte

L'architecture des serveurs, où vont se connecter les cartes à microprocesseur, que nous proposons est celle décrite par la figure III.2.8.

Dans cette architecture nous retrouvons tous les composants décrits dans ce chapitre :

- L'OTS Carte (un par machine de connexion), qui exécute localement l'algorithme de validation distribuée. En cas de présence de la carte (le cas le plus courant lorsqu'une transaction se déroule), il communique directement avec les gestionnaires de ressources carte. Sinon, ses messages sont stockés dans la file d'attente de la carte (il s'agit principalement des décisions prises après la phase «Prepare» du protocole de validation à deux phases, si la carte est retirée entre les deux phases de la validation). D'un point de vue réseau, il accède à d'autres OTS, pour s'enregistrer sur demande de la carte. Seuls ces OTS (qui participent à la transaction) peuvent envoyer un ordre à l'OTS Carte.

- Un COA représentant la Carte (un par carte), qui présente les services inclus dans la carte sur le réseau, et qui traduit les requêtes. Ce composant est créé au moment de la connexion de la carte sur le terminal.

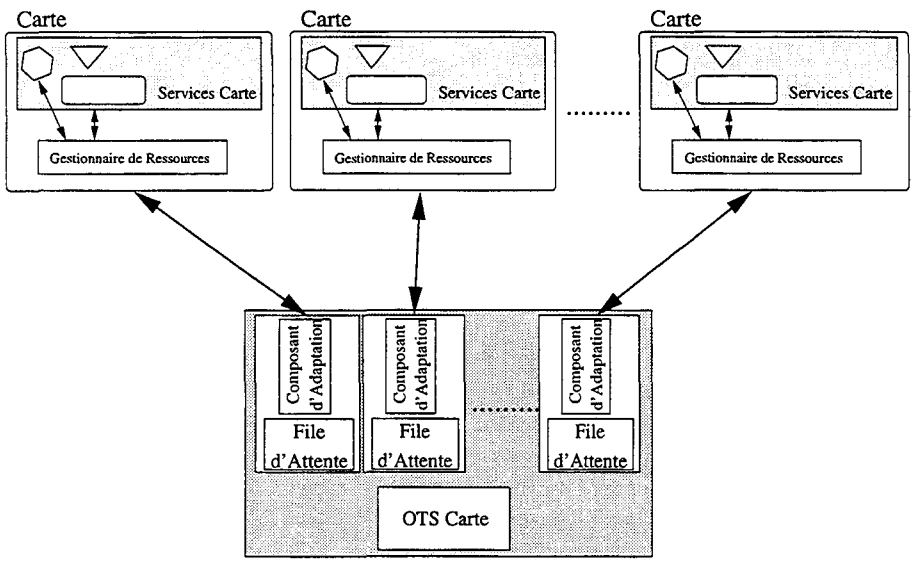


FIG. III.2.8 - *Composants logiciels d'une machine de connexion carte*

- La file d'attente de messages (une par carte), qui stocke les messages en attentes de connexion de la carte (même ceux de l'OTS Carte). Les informations contenues dans ce composant sont récupérées sur le précédent site de connexion de la carte à microprocesseur (et une file d'attente est recréé sur le nouveau site).

Une telle architecture devra être mise en place sur les terminaux destinés à utiliser la carte à microprocesseur comme serveur dans des transactions distribuées (notamment les terminaux de paiement sur Internet).

Il est bien entendu que tous les échanges entre la carte et sa représentation externe devront être sécurisés (intégrité et authenticité des messages garanties).

2.5 Un rôle plus important pour la carte ...

Nous avons étudié dans la partie précédente, l'intégration d'une carte dans un système distribué en tant que serveur. Nous allons voir dans cette section s'il existe une possibilité d'utiliser la carte soit en tant que coordinateur de validation, soit en tant que client de transactions distribuées (c'est-à-dire demandeur de services).

Pour ces deux études, nous allons procéder en deux étapes : tout d'abord nous décrirons l'architecture qu'il est possible d'utiliser, puis, nous discuterons de la pertinence de cette architecture (en terme de réalisation et en terme d'utilisation).

2.5.1 Le Coordinateur dans la carte

Nous avons vu dans le chapitre sur les nouvelles applications des cartes à microprocesseur, que pour des raisons de sécurisation de la validation des transactions, il était intéressant de confier le déroulement du protocole à la carte.

Si l'on se place en dehors de toute contrainte technique du côté de la carte, l'architecture qui en résulte serait celle décrite par la figure III.2.9.

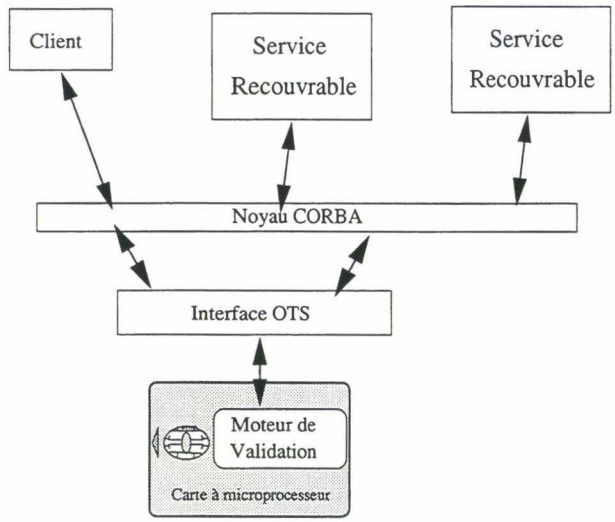


FIG. III.2.9 - *Une carte coordinatrice de validation de transaction*

Cependant, une telle architecture n'est envisageable, que si la carte prend part effectivement à la transaction (par exemple, en tant que fournisseur de un ou plusieurs services). L'architecture que l'on retrouve alors est celle que nous avons qualifiée d'«idéale» au début de ce chapitre (cf. Figure III.2.1), et qui, on le sait, n'est pas envisageable immédiatement.

2.5.2 La carte Cliente de Transactions

Une carte initiant une application distribuée au sens Client (dans le modèle Client / Serveur), serait un gage de sécurité pour beaucoup de participants d'applications distribuées (cf. Chapitre II.1). En se basant sur les composants que nous avons définis dans ce chapitre, l'architecture distribuée d'une application pilotée par une carte serait celle de la figure III.2.10.

Nous avons vu dans le chapitre II.1, que l'utilisation de la carte comme cliente d'une application distribuée pose de nombreux problèmes en terme d'aménagement du protocole de communication.

De même qu'un composant d'adaptation est proposé pour transcrire les requêtes dans le cas d'une carte serveur, un composant d'adaptation peut assurer l'interface avec une carte cliente, en commençant par demander à la carte à microprocesseur«Que faut-il faire?». La réponse à cette requête est alors la requête à exécuter sur le réseau.

Cependant, comme la carte à microprocesseur ne dispose pas d'entrée / sortie telles qu'un clavier ou un écran, un tel dispositif doit être utilisé sur un lecteur de carte sécurisé, muni de son propre clavier, et de son propre écran de visualisation des données renvoyées par la carte.

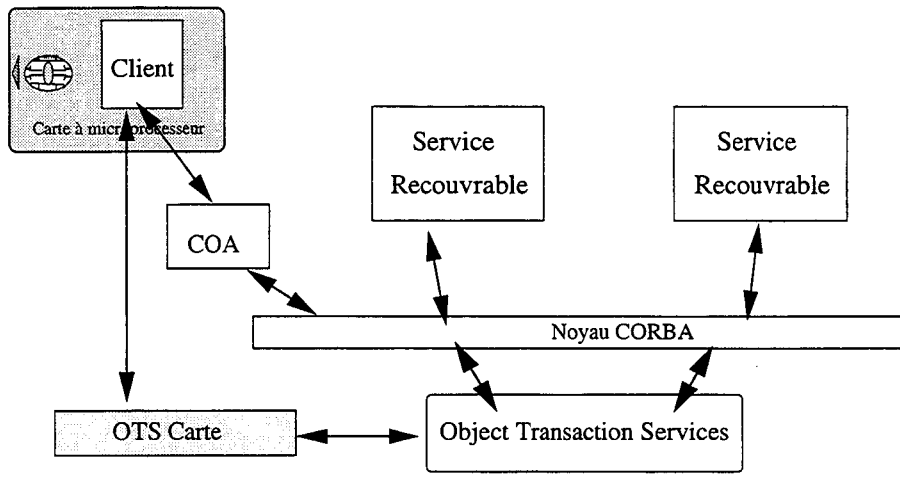


FIG. III.2.10 - *Carte client d'une transaction distribuée*

2.6 Conclusion

L'aboutissement de ces travaux nous a mené à définir une architecture permettant à une carte à microprocesseur de participer à une transaction distribuée en tant que serveur transactionnel. Cette intégration passe principalement par l'utilisation d'un composant d'adaptation, traduisant les requêtes provenant de l'ORB en requêtes carte (et inversement). Nous avons aussi défini un service transactionnel basé sur les spécifications OTS de l'OMG, intégrant une interface de communication vers les cartes à microprocesseur.

Les cartes à microprocesseur pourraient s'interfacer avec un OTS classique (i.e. tel que spécifié par l'OMG), mais cela conduirait à donner un rôle important à l'objet d'adaptation carte, qui devrait prendre en charge à lui seul les spécificités des cartes à microprocesseur. Cela est envisageable dans le cas d'applications mettant en jeu une seule carte, mais devient difficile à gérer pour les applications accédant à plusieurs cartes.

La solution idéale, représentée par un service transactionnel interne à la carte communiquant avec les services de la carte via un bus façon CORBA, a plusieurs avantages par rapport à l'architecture présentée ici :

- **Gestion interne à la carte :** lors du déroulement d'une transaction mettant en jeu plusieurs services internes à la carte à microprocesseur, cet OTS pourrait assurer la validation «distribuée»[7]

- **Sécurisation des validations :** L'OTS contenu dans la carte à microprocesseur serait de totale confiance.

- **Une meilleure communication entre les services :** Les problèmes de coopération entre différents services carte seraient plus facile à gérer, les services

7. La notion de distribution étant alors réduite à différents composants logiciels internes à la carte à microprocesseur.

communiquant entre eux via des envois de requête.

De plus, l'étude de l'intégration des cartes à microprocesseur dans les mécanismes de transactions distribuées ne doit pas s'arrêter à une carte serveur. En effet, la carte a un rôle sécuritaire à jouer en tant que Client de l'application. Du point de vue de l'intégration d'une telle carte dans les systèmes distribués, l'architecture décrite ici est tout à fait concevable.

Cependant, les limites actuelles des cartes en terme de communication, de capacité à émettre des requêtes, ou de capacité de stockage d'informations interdisent une telle utilisation des cartes. Il faut pour cela, que le système d'exploitation des cartes à microprocesseur, et les terminaux de connexion soient complètement repensés (Gestion des entrées / sorties, et génération de requêtes).

Chapitre III.3

Une Implantation de nos solutions : GemXpresso et Transaction

3.1 Introduction

Le but de la maquette que nous allons présenter dans ce chapitre est de prouver la faisabilité des propositions énoncées dans les deux chapitres précédents. Etant donné l'intérêt croissant pour l'utilisation des cartes à microprocesseur dans le domaine du commerce électronique sur Internet [**P98**], nous avons décidé de simuler une application de paiement sur un réseau distribué, à l'aide d'une carte à microprocesseur contenant un porte-monnaie électronique.

La carte choisie pour contenir l'application de PME et le gestionnaire de ressources transactionnel est la GemXpresso, qui est une implémentation des spécifications de la JavaCard 2.0 par Gemplus. Nous commencerons ce chapitre par une description de cette carte, et des raisons qui nous ont poussés à la choisir pour notre maquette.

Après cela, nous expliciterons le schéma retenu pour l'intégration du mécanisme de reprise sur panne dans cette carte, en décrivant le procédé retenu, ainsi que le modèle d'intégration que nous avons été obligés de suivre de par les contraintes actuelles de la carte.

Puis, nous décrirons l'intégration de notre maquette carte, dans un système distribué, basé sur un bus CORBA. Cette description portera sur le contenu du com-

posant d'adaptation de la carte à microprocesseur, ainsi que sur les interfaces de communication entre le moniteur transactionnel et la carte.

Enfin, après avoir décrit plus en détail le fonctionnement de l'application, nous étudierons les possibilités de portabilité de la maquette vers d'autres architectures distribuées, et nous conclurons sur les résultats et apports suggérés par cette maquette.

3.2 Point de Départ : La GemXpresso

La GemXpresso est l'implantation de la spécification JavaCard de SUN [**JC97**] par la société Gemplus. Cette carte permet le chargement, et le déchargement d'applications JavaCard.

Dans la version que nous avons utilisée pour notre maquette, la GemXpresso permet l'utilisation de deux mécanismes, qui nous ont conduit à choisir cette carte :

- Un mécanisme de publication d'interface, appelé DMI, pour Direct Method Invocation [**VV98**], proche de celui utilisé dans le cadre du projet OSMOSE, qui nous a facilité le développement de l'objet d'adaptation carte.

- Dans sa version préliminaire, cette carte permet la communication entre applications dans la carte, ce qui nous permet d'utiliser une application pour simuler le mécanisme de reprise sur panne.

Enfin, cette carte est basée sur un processeur RISC 32 bits relativement puissant, et dispose de suffisamment de mémoire pour supporter nos techniques de reprise sur panne.

3.3 Problèmes spécifiques aux composants utilisés

La carte utilisée pour notre maquette est en réalité une version préliminaire (version beta0.21) d'un futur produit. Cette carte étant un produit destiné à la commercialisation, nous ne pouvions pas avoir accès aux sources du système d'exploitation. Il était donc impossible d'y inclure un mécanisme de reprise sur panne directement dans le système d'exploitation.

Concernant l'ORB, dans un premier temps, nous avons utilisé le bus logiciel OmniBroker (récemment renommé ORBacus) dans sa version 2.0 Beta4 [**ORB98**]. Les principaux problèmes relatifs à cet ORB sont les suivants :

- Le bus de communication ne prévoit pas le transport d'un contexte transactionnel en mode implicite. Nous serons donc obligés d'utiliser le mode de propagation explicite de l'OTS.

- Aucun service transactionnel n'a été développé pour cet OTS. Cela nous a donc obligé à développer un OTS «minimal», en partie conforme aux spécifications de l'OMG.

Puis, dès qu'il a été rendu public, nous avons utilisé le bus logiciel de Java livré avec la première version béta du JDK 1.2 [**JAV98**], et nous avons mis en conformité l'OTS réalisé avec l'API JTS[1] de SUN [**JAV98b**].

3.4 Réalisation d'un mécanisme de Reprise sur Panne

Nous avons choisi, pour notre maquette, de travailler sur un exemple d'application relativement simple : il s'agit d'effectuer un transfert d'un PME sur notre carte, vers une banque connectée à un réseau (par exemple Internet). Le choix du type d'application (qui est en fait une application carte simple, dans un contexte distribué) implique uniquement l'intégration d'un mécanisme de reprise sur panne. Il n'est pas nécessaire d'intégrer un mécanisme de contrôle de concurrence.

Compte tenu des contraintes liées à l'utilisation d'une carte du commerce, non prévue pour participer à une transaction distribuée, nous décrirons en détail la solution choisie pour implanter dans cette carte un mécanisme de reprise sur panne, puis nous détaillerons les limites d'une telle architecture applicative.

3.4.1 Mécanisme de reprise après panne implanté

Afin de simuler la présence d'un mécanisme de reprise sur panne dans le système d'exploitation de la carte à microprocesseur, nous avons utilisé deux applications communicant entre elles (cf. figure III.3.1) :

- **L'application PME :** cette application est en fait un compte monétaire classique, qui fournit les méthodes de gestion d'un compte (Débit, Crédit et Solde).

- **L'application Transaction :** cette application permet à la fois à la carte de répondre aux ordres du coordinateur de la transaction distribuée, et de gérer la reprise sur panne pour la carte à microprocesseur.

Pour répondre aux ordres du coordinateur de la transaction, l'application Transaction fournit les méthodes suivantes :

- **Begin_Transaction** : qui permet à la carte d'accepter, ou de refuser de participer à une nouvelle transaction. La cause de refus de participer à une nouvelle transaction sera qu'elle est déjà partie prenante dans une autre.

- **Prepare** : cette méthode est appelée par le coordinateur pour demander à la carte son vote à la première phase de la validation.

- **Commit** : Lorsque cette méthode est appelée, la carte valide les changements effectués depuis le Begin_Transaction.

- **Rollback** : Lorsque cette méthode est appelée, la carte annule tous les changements effectués depuis le Begin_Transaction.

1. JTS : Java Transaction Service

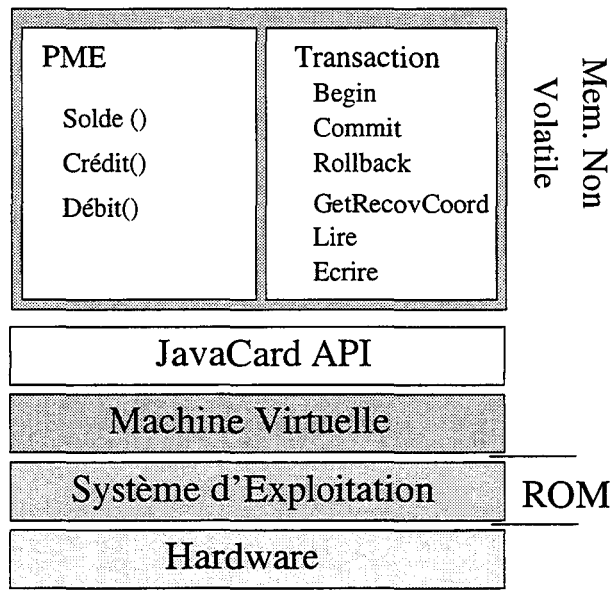


FIG. III.3.1 - *Architecture utilisée dans la GemXpresso*

- **GetRecoveryCoordinator** : cette méthode est utilisée pour retrouver le résultat de la terminaison d'une transaction. Ce cas se produit lorsque la carte a été arrachée entre sa réponse à la première phase, et la seconde phase (décision du coordinateur) du protocole de validation.

En plus de cela, l'application Transaction dispose de méthodes partagées avec d'autres applications (c'est-à-dire que d'autres applications carte peuvent appeler). Ces méthodes permettent de «détourner» les ordres d'écriture et de lecture sur le support non volatile, de manière à utiliser une solution similaire à celle des pages ombres, sans modifier le système d'exploitation. Pour stocker les données, nous utilisons deux tableaux sauvegardés sur le support non volatile. Le premier tableau, que nous avons appelé «Page Courante», contient les données valides, et le second tableau, appelé «Page Ombre» contient les données en cours de modification par une transaction (il est utilisé si un Begin_Transaction a été exécuté).

Les méthodes pour accéder à ces données sont :

- **Lire** : qui permet à une application (ici, l'application PME), de lire la valeur de ses données. Si on est dans une transaction, la lecture se fait dans le tableau «Page Ombre», sinon, elle s'effectue dans le tableau «Page Courante».

- **Ecrire** : qui permet à une application d'écrire la nouvelle valeur d'une donnée. Si on est dans une transaction, l'écriture se fait dans le tableau «Page Ombre», sinon, elle se fait dans le tableau «Page Courante».

Au début de la transaction, le contenu de «Page Courante» est recopié dans le tableau «Page Ombre».

A la validation, le sens des deux tableaux est inversé : le tableau «Page Ombre» devient le tableau «Page courante», et inversement.

3.4.2 Limites de cette solution

Cette solution pour la reprise sur panne possède bien sûr les limites du mécanisme des pages ombres (une seule transaction à la fois). Mais elle est aussi des limitée par le fait que cette implantation n'est pas réalisée directement dans le système d'exploitation.

En effet, nous sommes assez éloignés de l'implantation proposée dans le chapitre III.1. Ainsi, les principaux défauts de notre solution sont :

- **Un programme spécifique** : L'écriture de l'application PME est modifiée pour fonctionner avec l'application Transac. De plus, l'utilisation d'une application non sécurisée (puisque devant être accédée par toutes les autres applications pour effectuer les lectures et les écritures), n'est pas une bonne solution, car elle fragilise la sécurité de la carte à microprocesseur

- **Une communication complexe** : notre solution utilise des mécanismes de communication entre les services d'une même carte à microprocesseur, qui ne sont pas encore disponibles (pour des raisons de sécurité) sur les cartes commercialisées.

3.5 Intégration dans une Application Distribuée

Comme nous l'avons déjà dit précédemment, nous ne pouvons pas modifier en profondeur les systèmes traditionnels pour prendre en compte les cartes à microprocesseur (cf. Chapitre III.2).

Pour intégrer la carte à microprocesseur dans les systèmes distribués traditionnels, nous avons donc utilisé un composant d'adaptation, inspiré de celui décrit dans le projet OSMOSE [**V97**]. Le principe de base de ce composant d'adaptation, qui sert à publier les interfaces des services présents dans la carte à microprocesseur pour permettre au monde extérieur d'y accéder rapidement, est maintenant repris par la société Gemplus pour sa carte GemXpresso [**VV98**].

3.5.1 Description du composant d'adaptation

Le composant d'adaptation de la carte (appelé COA) est en fait constitué de trois parties (cf. figure III.3.2):

- **Connexion Carte** : cette partie contient les méthodes utiles à la gestion de la connexion et de la déconnexion de la carte. Lorsqu'une requête est envoyée vers la carte à microprocesseur, l'objet d'adaptation teste si la carte est sous tension. Si ce n'est pas le cas, il exécute la méthode PowerOn, pour connecter la carte. A la fin de la session, la carte est déconnectée (c'est-à-dire mise hors tension) via la méthode PowerOff.

- **Application** : Cette partie de l'objet d'adaptation sert à publier les interfaces des services proposés par la carte à microprocesseur (ici, nous présentons l'interface du service PME). C'est en fait la partie qui traduit les requêtes externes en requêtes carte.

- **Transaction** : Cette partie est dédiée aux ordres transactionnels en provenance de l'OTS qui coordonne la transaction à laquelle participe la carte à microprocesseur. Cette partie de l'objet d'adaptation effectue le lien entre l'OTS et le gestionnaire de ressources de la carte.

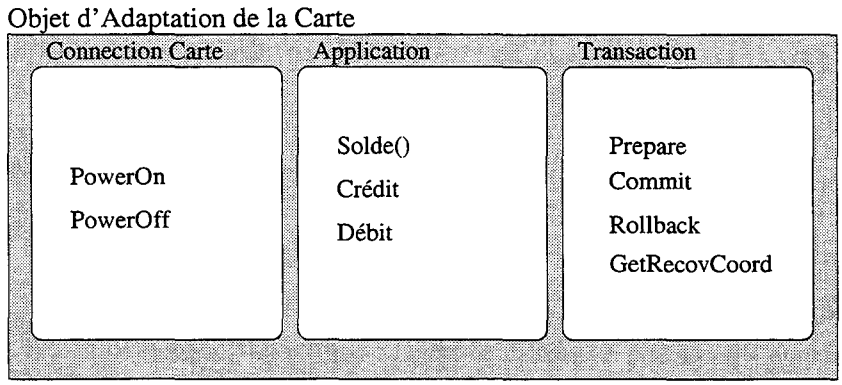


FIG. III.3.2 - *Architecture du composant d'adaptation carte*

En résumé, l'objet d'adaptation remplit plusieurs rôles dans la participation de la carte à microprocesseur à une transaction distribuée :

- Il présente les services de la carte, et assure la traduction des requêtes vers ces services.
- Il enregistre la carte auprès du coordinateur de la transaction distribuée.
- Il gère et convertit les références des transactions auxquelles participe la carte.

Nous allons donc maintenant décrire plus en détail le rôle de l'objet d'adaptation, en différenciant les appels d'un client, et les appels du coordinateur de la transaction.

3.5.2 Fonctionnement du COA

Le COA reçoit des ordres externes en provenance de deux sources : le client de la transaction (qui est l'initiateur), et le coordinateur de la transaction (le service transactionnel, ici OTS).

Lorsque le COA reçoit une requête du client de la transaction, cette requête comporte deux informations : la méthode invoquée sur la carte à microprocesseur(avec ses arguments), et le contexte transactionnel qui identifie la transaction pour laquelle est demandée cette méthode.

A partir de ce moment, il y a deux possibilités : soit la transaction est déjà connue du COA (c'est-à-dire que la carte est déjà enregistrée auprès de l'OTS), soit c'est la première fois qu'il reçoit une requête pour cette transaction, et il doit donc effectuer le Begin_Transaction auprès de la carte, et s'enregistrer auprès de l'OTS.

Si la carte est déjà enregistrée auprès de l'OTS, le COA se contente de transcrire la requête CORBA en requête carte.

En ce qui concerne les ordres de l'OTS («Prepare» et «Commit» ou «Rollback»), ces ordres sont directement traduits par le COA, et envoyés à l'application «Transaction» de la carte à microprocesseur.

3.6 Le service transactionnel

De manière à assurer la validation distribuée de notre transaction, nous sommes obligés d'utiliser un coordinateur de transaction (ici, le service transactionnel OTS).

Après avoir expliqué les problèmes rencontrés, qui nous ont obligé à écrire notre propre OTS, nous décrirons les parties de l'OTS réalisé.

3.6.1 Problèmes rencontrés

Etant donné qu'aucun service transactionnel OTS n'a été développé pour OmniBroker (l'ORB que nous avons choisi d'utiliser), nous avons développé un OTS «minimal», en partie conforme aux spécifications de l'OMG.

De manière à rendre notre travail portable sous d'autres environnements, nous nous sommes attachés à suivre les spécifications de l'OMG [**OMG97c**] et aux interfaces du JTS de SUN [**JAV98b**], de manière à ce que notre carte, et notre composant d'adaptation puissent s'adapter à d'autres services transactionnels.

L'écriture de cet OTS minimal nous a permis de bien assimiler le fonctionnement de ce service transactionnel.

3.6.2 Description de l'OTS réalisé

Nous allons ici décrire les différents composants de l'OTS que nous avons décidé d'implémenter (tout ou partie). De façon générale, nous avons uniquement implémenté les interfaces correspondants au mode direct (le client et les services accèdent directement aux objets de l'OTS) / explicite (le contexte transactionnel est propagé explicitement, c'est un paramètre de chaque méthode).

- La première interface de notre OTS est l'interface *TransactionFactory*, qui permet de créer une nouvelle transaction. La méthode utilisée pour cela est la méthode `create()`, qui renvoie un identifiant unique de transaction qui est représenté par un objet *control*.

- Concernant l'interface de l'objet *control*, nous avons implémenté deux méthodes :

  - La méthode `getTerminator () throws unavailable`, qui retourne le *Terminator* de la transaction,
  - et la méthode `getCoordinator () throws unavailable`, qui retourne le *Coordinator* de la transaction.

  Pour ces deux méthodes, une exception est levée si les objets n'existent pas.

- L'interface de l'objet *Terminator* contient deux méthodes :
  - La méthode `commit()`, qui permet de lancer le protocole de validation d'une transaction distribuée,
  - et la méthode `rollback()`, qui permet d'annuler une transaction.

  Généralement, ces méthodes sont appelées par le client à la fin de la transaction.
- L'interface de l'objet *Coordinator* contient lui quatre méthodes :
  - La méthode `GetStatus()`, qui retourne l'état de la transaction en cours (`Status`). Les différents états possibles gérés par notre OTS «minimal» sont au nombre de 6 :
    - `StatusUnknow`, qui est le status de la transaction à sa création,
    - `StatusActive`, qui est le status de la transaction lorsque le coordinateur commence l'exécution du protocole de validation à deux phases,
    - `StatusPrepared`, qui signifie que toutes les ressources ont répondu à l'ordre `prepare`,
    - `StatusCommited`, qui signifie que la transaction a été validée,
    - `StatusRollback`, qui signifie que la transaction a été annulée,
    - et enfin, `StatusMarkedRollback`, qui signifie que la transaction ne peut être qu'annulée, car un serveur a demandé un `RollbackOnly` suite à un problème.
  - La méthode `RegisterRessource(Ressource ressource) throws Inactive` permet au serveur transactionnel de s'enregistrer en tant que *Ressource* auprès du *Coordinateur*. Un *RecoveryCoordinateur* est renvoyé au serveur transactionnel.
  - La méthode `RollbackOnly() throws Inactive`, peut être appelée par un serveur, pour modifier l'état d'une transaction. La transaction est alors marquée `StatusMarkedRollback`, et seule l'annulation de la transaction est alors possible.
  - La dernière méthode de l'objet *Coordinator*, est la méthode `getTransactionname()`, qui renvoie le nom de la transaction en cours. Ce nom est représenté par une chaîne de caractères (`String`).
- Enfin, l'interface de l'objet *Ressource* contient trois méthodes, qui sont utilisées par le *Coordinator* pour propager les ordres du protocole de validation à deux phases auprès des serveurs :
  - La méthode `Prepare()` est appelée pour demander à la ressource son `Vote`, pour la validation de la transaction distribuée. La ressource peut voter `voteRollback` si elle a rencontré un problème au cours de la transaction, `voteCommit` si elle est prête à valider, ou encore `votereadonly`, si elle n'a pas modifié ses données (elle est alors indifférente au résultat de la transaction).

- La méthode `valide()`, sert à valider les changements sur les données de la ressource,
- et enfin, la méthode `rollback()`, sert à annuler les changements et à retourner à l'état précédant la transaction.

Tout ces éléments ont été écrits en langage Java, et utilisent le bus logiciel CORBA ORBacus 2.0.4.

3.7 L'application : Transfert d'un compte Bancaire sur un PME

L'application réalisée consiste en un transfert d'un compte bancaire sur un PME. Le but est de simuler un rechargement de PME (ou un paiement) sur un réseau distribué (Internet par exemple).

Les participants à cette transaction bancaire sont :

- La personne qui réalise le transfert (ici appelée le Client)
- Sa banque, pour accéder au compte bancaire (appelée Serveur Bancaire)
- Et la carte à microprocesseur contenant le PME, sur lequel sera transféré l'argent du compte bancaire.

Dans cette partie, nous allons successivement décrire l'architecture globale de l'application, puis, nous reviendrons en détail sur chacun des composants.

3.7.1 Architecture Globale

L'application met en jeu un client et deux serveurs, utilisant divers services CORBA (le service OTS et le service de nommage).

Nous avons utilisé le service de nommage proposé par CORBA car il fournit un système de désignation pour retrouver les objets à partir de noms symboliques. Ce système est organisé en graphes de répertoires contenant les références sur les objets **[OMG97b]**.

L'architecture globale est décrite par la figure III.3.3. On y trouve le client, le serveur banque, et le serveur carte, constitué de la carte à microprocesseur et de son objet d'adaptation.

Nous allons maintenant décrire, pour plus de clarté, le déroulement de l'application composant par composant. Le code source de chacun de ces clients est disponible en annexe.

3.7.2 Le Client

De manière résumée, le code source du client est le suivant :

```
\\ début de la transaction
  contextId = TransactionFactory.create();
```

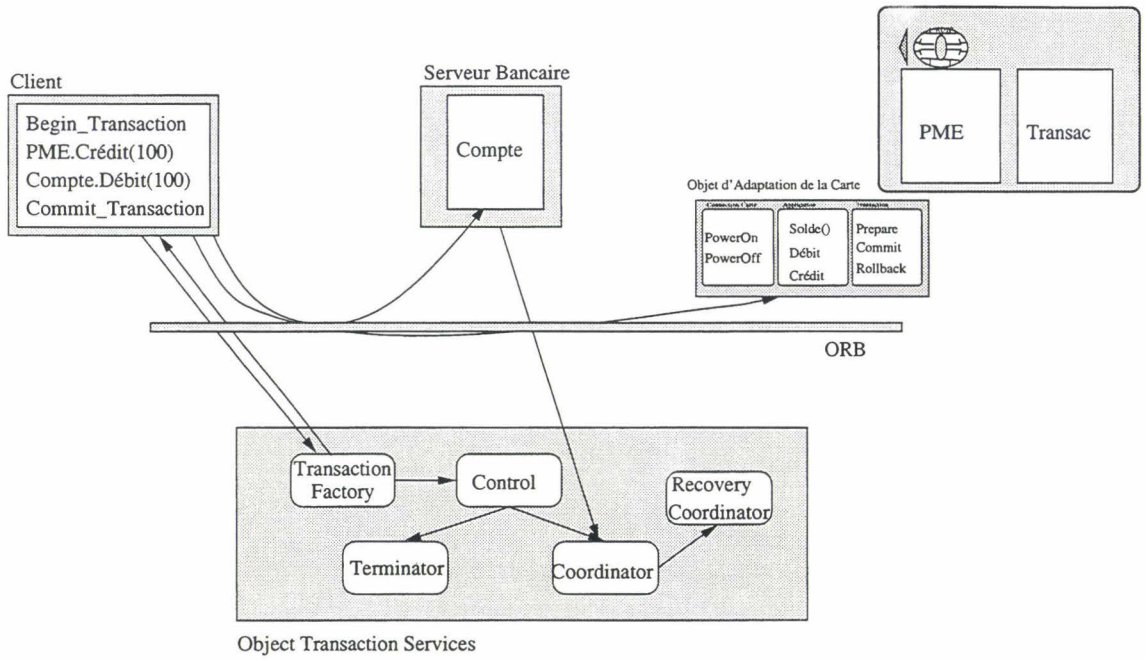


FIG. III.3.3 - *Application de rechargement d'un PME dans un système distribué*

```
\\ virement d'argent
  compte.debit(100, contextId);
  PME.credit(100,contextId);

\\validation de la transaction
  contextId.getTerminator().commit();
```

Le client démarre la transaction en effectuant un Begin_Transaction, qui consiste à créer un contexte transactionnel auprès de son OTS, en invoquant la méthode `create()` de l'objet *TransactionFactory*. Cette méthode retourne au client de la transaction un objet *Control*, qui représente un identifiant unique de cette transaction.

Ensuite, le client effectue le transfert d'argent, en contactant les serveurs concernés. Lors de l'envoi de la requête auprès de ces serveurs, le client joint le contexte de la transaction.

Enfin, le client décide de valider, ou d'abandonner, la transaction, en invoquant auprès de l'OTS, soit la méthode `commit()`, soit la méthode `Abort()` de l'objet *Terminator*.

3.7.3 Le Serveur Banque

Le serveur bancaire, est un serveur transactionnel de gestion d'un compte bancaire. Le terme serveur transactionnel, signifie qu'il est apte à répondre à une requête, en exécutant une suite d'actions vérifiant les propriétés du modèle transactionnel, et qu'il peut aussi demander l'annulation d'une transaction distribuée.

A la réception de la requête du client (requête qui contient un contexte transactionnel), le serveur bancaire va s'enregistrer auprès de son OTS (dans notre maquette, il s'agit du même OTS que celui du client). Pour cela, il fait appel à la méthode `registerRessource(Ressource)` de l'objet *Coordinator*. L'objet *Coordinator* est obtenu grâce à la méthode `getCoordinator()` de l'objet *Control*, qui est lui-même obtenu par le contexte transactionnel :

```
// enregistrement de la ressource bancaire
// auprès de l'OTS

 RecoveryCoordinator rc;
 rc =ContextId.getCoordinator().RegisterRessource(new ressource());

// traitement de la requête (par exemple un débit)
...
```

L'objet *RecoveryCoordinator* est utilisé pour connaître la terminaison de la transaction en cas de problème sur le serveur.

3.7.4 Le serveur Carte

Le serveur carte est beaucoup plus lourd, car ce terme englobe la carte munie de son composant d'adaptation. Une partie du code source de l'objet d'adaptation est le suivant :

```
// Connexion avec la carte
cardPME=...
cardTransaction=...

//Verif. du status de la carte
try {

// On recherche le contexte d'un transaction
  etat = cardTransaction.getRecoveryCoordinator();
//On va recherche l'etat de la transaction
  status = etat.get_status();

  if (status = StatusCommitted)
      cardTransaction.commit();

  if (status = StatusRolledback)
      cardTransaction.rollback();
}
catch (Unavailable ex)  {
  //La carte n'a pas de transaction en cours}

// enregistrement de la ressource bancaire
```

```
// auprès de l'OTS

 RecoveryCoordinator rc;
 rc =ContextId.getCoordinator().RegisterRessource(new ressource());

// Envoie de la requête (par exemple un débit)
...
```

Le composant d'adaptation va recevoir le même style de requêtes que le serveur bancaire, mais au lieu de les traiter directement, il va les envoyer à la carte (en les traduisant). C'est alors la carte qui va prendre la décision de participer ou non à la transaction.

Cependant, avant le traitement de la première requête du client de la transaction, l'objet d'adaptation de la carte doit vérifier que celle-ci se trouve dans un état apte à traiter une nouvelle transaction. Pour cela, il doit vérifier si la carte contient une transaction en cours (dans ce cas là, elle a stocké la référence de son *Coordinator*). Si c'est le cas, l'objet d'adaptation va consulter le résultat de la transaction auprès du *Coordinator*, ce qui permet de terminer la transaction de la carte à microprocesseur.

3.7.5 Conclusion et Résultats

Les résultats obtenus ici, sont avant toute chose des résultats de faisabilité sur la capacité d'une carte à microprocesseur à participer à une transaction distribuée.

Il est donc possible, via un réseau type Internet, d'accéder à un service transactionnel, chargé dans une carte à microprocesseur, pour exécuter une transaction, distribuée sur d'autres sites.

La première application de la maquette, telle que nous l'avons réalisée, est le paiement sécurisé sur Internet, via l'utilisation d'un PME sur carte à microprocesseur.

Cependant, cette maquette contient un certain nombre de points faibles :

- La solution que nous avons envisagée pour le gestionnaire de ressources carte, n'est pas envisageable telle quelle, pour une carte actuellement commercialisée. Les possibilités de fraudes, et de mauvais fonctionnements sont trop nombreuses. Il est donc indispensable de développer un système d'exploitation conformes aux spécifications du chapitre III.1.

- L'écriture du composant d'adaptation s'effectue actuellement manuellement : pour chaque service transactionnel dans la carte, il faut développer la partie du composant correspondante. Il est cependant possible de modifier le chargeur d'application dans la carte à microprocesseur, de manière à ce que les ajouts que nous avons faits (l'équivalent du Begin_Transaction), soient insérés automatiquement.

3.8 Portabilité de la maquette

Notre maquette a été réalisée sur la base d'un bus de communication CORBA, muni de son service transactionnel OTS. Cette architecture est de plus en plus répandue, sur tout type de machine.

Cependant, sur les architectures de type PC / Windows (Personnal Computer), l'architecture de l'OMG est en concurrence avec l'architecture COM de Microsoft (utilisant le service transactionnel MTS) (cf. Chapitre I.2).

Nous avons donc commencé l'étude de la portabilité de notre maquette sous cet environnement. Ce travail est dorénavant pris en charge dans l'équipe de recherche et développement de la société Gemplus, qui travaille sur l'intégration de la carte à microprocesseur dans l'environnement de Microsoft.

Quatrième partie

Conclusion et Perspectives

Chapitre IV.1

Conclusion

1.1 Une utilisation plus évoluée des cartes à microprocesseur

L'utilisation des cartes à microprocesseur est en plein développement, grâce aux technologies telles que l'Internet ou la téléphonie mobile. Ce développement nous pousse à mieux utiliser ce composant, à la fois synonyme de très haute sécurité, mais aussi de mobilité du code et des données qu'il contient.

En parallèle, la technologie mise en jeu dans les cartes à microprocesseur se rapproche de plus en plus des technologies utilisées dans les systèmes informatiques traditionnels (tant au niveau du matériel, que du logiciel embarqué). Ces avancées permettent des implantations d'applications de plus en plus élaborées au sein des cartes (cf. Chapitre I.1).

Cependant, ces nouvelles applications se heurtent à certaines caractéristiques qui sont problématiques pour les cartes à microprocesseur. Ainsi, la durée d'inutilisation des cartes (correspondant au temps de déconnexion), ainsi que les risques de panne propres aux cartes, rendent difficiles le développement et la gestion de nouvelles applications mettant en jeu plusieurs partenaires dans un contexte distribué (cf. Chapitre II.1).

Dans ce mémoire, nous utilisons le modèle transactionnel pour faciliter la tâche du programmeur dans la gestion de la distribution (utilisation d'un mécanisme de validation distribuée), des accès concurrents aux données des cartes et des pannes (cf. Chapitre I.2).

L'implantation d'un tel modèle dans la carte à microprocesseur passe par l'intégration, au coeur du système d'exploitation de cette carte, d'un mécanisme de reprise sur panne et d'un mécanisme de contrôle de concurrence. La carte ainsi obtenue est capable d'assurer les propriétés ACID des transactions lors de la modification des données qu'elle contient (nous avons appelé ce type de carte la COST (Carte Orientée Services Transactionnels)).

De plus, si pour que la carte COST participe à des transactions distribuées, il faut assurer deux choses :

- Tout d'abord, la carte doit être en mesure de répondre aux ordres d'un coordinateur de validation de transaction (par exemple, les ordres Begin, Prepare et Commit ou Abort du protocole de validation à deux phases).

- Ensuite, cette carte doit être en mesure de communiquer avec les systèmes informatiques traditionnels, sans que ceux-ci nécessitent de modification. Ce qui nous a conduit à définir un composant d'adaptation entre la carte et le service transactionnel de validation.

1.2 COST : un gestionnaire de ressources transactionnel pour la carte

La définition d'un gestionnaire de ressources transactionnel pour carte à microprocesseur se révèle beaucoup plus complexe que la simple définition d'un mécanisme de reprise sur panne et de contrôle de concurrence.

En effet, le domaine d'utilisation des cartes à microprocesseur est de plus en plus large. Ce qui va nous conduit à utiliser des cartes, aussi bien dans le cadre d'applications très simples, nécessitant peu de ressources, que dans des applications complexes, se déroulant sur plusieurs connexions de la carte à microprocesseur (cf. Chapitre II.2).

Les mécanismes mis en jeu dans le cadre des utilisations simples des cartes à microprocesseur ne peuvent pas être les mêmes que ceux utilisés pour les applications cartes complexes. En effet, le coût de fabrication d'une carte pour application simple doit rester très bas. Ainsi, la solution utilisée doit nécessiter le moins possible de ressource.

Cela nous oblige à adapter la solution envisagée en fonction du type d'application et du coût de la carte.

1.2.1 La reprise sur panne

Les stratégies de reprise sur panne peuvent s'avérer différentes en fonction de la complexité du type d'application à laquelle participe la carte à microprocesseur.

En terme de mécanisme de reprise sur panne, nous avons plus particulièrement étudié le mécanisme des pages ombres, et le mécanisme basé sur l'utilisation de journaux des images avant.

Il s'avère que le mécanisme des pages ombres est bien adapté aux «petites cartes», munies d'un support de mémoire non volatile de type EEPROM, et mono-applicatives (peu de place occupée en mémoire volatile, et surcoût faible en temps d'exécution).

A l'inverse, même s'il se révèle très coûteux en terme de nombre d'écritures (ce qui conduit à un temps d'exécution de l'application beaucoup plus important), le mécanisme de journalisation des images avant est bien adapté pour les cartes multi-applicatives. Il peut se révéler aussi comme étant un bon compromis pour les cartes mono-applicatives munies de mémoire FlashRAM. En effet, dans ce cas, le mécanisme des pages ombres se révèle extrêmement coûteux en mémoire RAM[1].

Des études plus précises des coûts des divers mécanismes sont en cours, sur des cartes et des exemples d'applications existantes. Il est en effet difficile d'évaluer le nombre de pages différentes accédées au cours de l'exécution d'un service, car il dépend de plusieurs critères (politique du cache en écriture, et politique de chargement des applications dans la carte).

1.2.2 Le contrôle de concurrence

Les problèmes d'accès concurrents aux données d'une carte se posent uniquement pour des cartes déjà complexes, pouvant accepter plusieurs transactions simultanées (cela ne concerne pas les cartes «actuelles»).

Le premier mécanisme de contrôle de concurrence que nous avons étudié en détail est le mécanisme de verrouillage à deux phases. Il est bien adapté à la problématique carte, car les cartes à microprocesseur contiennent peu de données, mais il y a un risque important d'accès concurrent sur ces données.

L'implantation de ce mécanisme dans les cartes à microprocesseur reste relativement coûteux, car il nécessite l'utilisation d'une table contenant les verrous, et chaque accès à une donnée est précédé de la consultation du verrou et / ou de sa mise à jour. De plus, la validation, ou l'abandon, d'une transaction, nous oblige à parcourir toute la table des verrous, pour libérer les données bloquées par la transaction terminée.

Concernant les problèmes d'interblocage, des solutions basées soit sur des temps limites de fin d'exécution, ou sur des détections de cycle dans les graphes d'exécution peuvent être envisagées.

Un autre mécanisme, plus simple à implanter, et moins lourd à utiliser, est la technique d'estampillage. Cette technique, basée sur l'utilisation de deux estampilles (une pour les lectures, et une pour les modifications), ne comporte pas de phase de libération à la fin des transactions. Cependant, elle induit de nouvelles dépendances entre des transactions qui auraient pu s'exécuter correctement avec un autre contrôle de concurrence.

Le principal défaut de ces méthodes de contrôle de concurrence est donc qu'elles ne permettent pas un grand niveau de concurrence des transactions, et qu'elles sont coûteuses en terme de taille mémoire. Un mécanisme utilisant la sémantique des objets serait donc mieux adapté dans certains cas pour la carte à microprocesseur.

1. Ce cas sera limité, car l'emploi de FlashRAM est particulièrement destiné à augmenter les capacités des cartes à microprocesseur, car cette mémoire utilise moins de place sur le support

Tout comme pour les mécanismes de contrôle de concurrence, des études complémentaires restent à mener, sur des exemples réels de systèmes d'exploitation, et d'applications.

1.3 La carte dans les Transactions Distribuées

Les discussions sur l'intégration des cartes à microprocesseur dans les systèmes distribués traditionnels ont été entamées depuis plusieurs années **[V95][V97]**. Il ne s'agissait pas dans ce mémoire de recommencer ce travail.

Ici, nous nous sommes attachés à appliquer les notions définies précédemment au modèle transactionnel, tout d'abord en considérant la carte à microprocesseur comme un serveur, puis en étudiant les possibilités d'évolution du rôle de cette carte.

1.3.1 Résultats

L'utilisation d'une carte COST comme serveur transactionnel dans une transaction distribuée se fait via un composant d'adaptation, qui permet de transcrire les requêtes du client et du coordinateur, en requêtes pour les cartes (cf. Chapitre III.2). La faisabilité de cette architecture a été démontrée par la maquette développée avec une JavaCard, dans un environnement CORBA, utilisant les services transactionnels OTS (cf. Chapitre III.3).

La portabilité de cette maquette dans d'autres environnements (notamment DCOM de Microsoft) et utilisant d'autres services transactionnels, est en cours d'étude.

Si l'on veut qu'une carte à microprocesseur participe à une transaction distribuée, il est impératif qu'elle soit en mesure de répondre aux ordres du coordinateur de la transaction. Dans le cas de l'utilisation d'un protocole de validation à deux phases, ces ordres sont le Begin_Transaction, le Prepare, le Commit et le Rollback. Ces ordres sont actuellement absents des normes spécifiant les cartes à microprocesseur (normes ISO/IEC 7816 - 4 à 7, et API de la JavaCard).

D'autre part, à cause du développement des possibilités des cartes à microprocesseur (puissance de calcul, taille des mémoire, complexité des programmes embarqués), il est tentant de vouloir confier un rôle de plus en plus important aux cartes dans les systèmes distribués. Ainsi, la carte cliente d'une application distribuée permettrait de faire progresser le niveau de sécurité des terminaux non dédiés (terminaux génériques), dans le sens où le code client est sûr (cf. chapitre II.1). Les modifications du protocole de communication rendues nécessaires pour la réalisation d'une carte cliente, peuvent être en partie assumées par le composant d'adaptation de la carte à microprocesseur.

1.3.2 Liens avec les travaux sur l'informatique mobile

Les cartes à microprocesseur ont un certain nombre de contraintes communes avec les ordinateurs mobiles :

- Les cartes à microprocesseur sont très souvent déconnectées, tout comme les ordinateurs mobiles, dont la connexion sans fil peut être interrompue à tout moment.

- Les caractéristiques des cartes à microprocesseur, même si elles sont en progression, sont encore nettement en retrait des stations fixes. Il en est de même (à une moindre échelle) pour les ordinateurs mobiles.

Des solutions sur la base d'une représentation externe permanente des ordinateurs mobiles sont actuellement proposées pour palier à ces contraintes [**C98**].

En application de ces résultats aux cartes à microprocesseur, nous avons défini, dans ce mémoire, un représentant permanent de la carte sur le réseau, qui stocke les requêtes destinées à la carte lorsqu'elle est déconnectée. Cette représentation remplit aussi le rôle de traducteur de requêtes, tel qu'il a été défini pour le composant d'adaptation de la carte.

L'architecture globale qui découle de l'utilisation d'une représentation externe pour carte est bien entendu plus simple que ce qui est spécifié pour l'informatique mobile : seuls les partenaires ayant communiqué avec la carte durant sa connexion connaissent l'emplacement de sa représentation externe, et celle-ci disparaît lors de la connexion suivante de la carte à microprocesseur, pour se recréer sur le nouveau terminal de connexion.

Chapitre IV.2

Perspectives

2.1 Evolution des systèmes d'exploitation carte

En dehors de certains problèmes connus, auxquels devront répondre les futurs systèmes d'exploitation pour carte à microprocesseur (comme le partage d'objet), ce mémoire démontre que l'on ne pourra plus parler de *système d'exploitation générique* pour les cartes à microprocesseur.

En effet, suivant l'architecture matérielle des cartes à microprocesseur, ainsi que le type de services qu'elles vont embarquer (carte mono-service, etc...), les solutions systèmes à envisager pour le traitement des pannes, ou des accès concurrents aux données seront différentes.

Des recherches basées sur la division des systèmes d'exploitation en modules, que l'on peut agencer les uns avec les autres pour former le système final, viennent de commencer au sein du laboratoire RD2P. Les solutions présentées dans ce mémoire sont en adéquation avec ces recherches, car comme nous l'avons vu, suivant les cartes, les mécanismes intégrés dans les systèmes d'exploitation pour gérer les transactions ne sont pas les mêmes.

La génération de systèmes d'exploitation contenant des mécanismes transactionnels permettra de mieux évaluer le coup de chaque mécanisme, en offrant des simulations globales de système d'exploitation. En effet, dans ce mémoire, nous avons tenté d'évaluer le coup de chaque technique en terme de place mémoire, ou de nombre d'écritures sur le support non volatile. Mais l'évaluation globale des performances est actuellement extrêmement compliquée, car elle dépend de beaucoup de facteurs (présence ou non d'un cache en écriture, politique d'éviction de ce cache, stratégie

d'implantation des objets sur le support non volatile, nombre de pages différentes accédées pendant l'exécution d'un service carte, etc...).

2.2 Un monde carte en pleine évolution

L'écart qui sépare les systèmes cartes des systèmes informatiques traditionnels, en terme de performance technique, est en train de considérablement se réduire. La progression de plus en plus rapide des performance des cartes à microprocesseur nous a permis d'appliquer des mécanismes, bien connus des systèmes traditionnels, au monde carte (gestionnaire de ressources transactionnel, ou protocole de validation distribuée).

Cependant, jusqu'à maintenant, nous avons principalement étudié la carte comme participant en tant que serveur à des transactions distribuées. Or, nous avons vu que d'un point de vue pratique, une carte cliente et / ou coordinatrice de validation de la transaction, serait un plus indéniable en terme de sécurité sur les applications distribuées.

Il est donc maintenant important d'étudier les nouvelles formes d'intégration des cartes à microprocesseur dans les systèmes distribués, et d'en tirer les conséquences en terme de définition de nouvelles fonctionnalités, et de protocoles de communication carte, qui datent d'un temps où la carte à microprocesseur ne pouvait que contenir des données, et non pas des services.

La définition d'une telle carte permettrait notamment de déporter une partie des services et des données actuellement dans la carte, et ne nécessitant pas obligatoirement la sécurité de la carte, vers des serveurs spécialisés, vers lesquels la carte pourrait émettre des requêtes.

2.3 L'arrivée de très petits composants dans le monde distribué

L'intégration des cartes à microprocesseur dans les systèmes distribués est un premier pas pour la prise en compte, par ces systèmes, de nouveaux composants informatiques.

Avec l'arrivée de systèmes tels que les étiquettes électroniques, qui peuvent être lues et écrites à distance (c'est à dire sans contact physique avec un terminal), il va devenir possible d'exécuter des transactions largement distribuées : si on prend l'exemple d'achat de stock (un peu comme cela se fait aux halles de Rungis), chaque marchandise pourra disposer de sa propre étiquette électronique, et sera donc capable de participer directement à la transaction de vente (d'où une plus grande sûreté de fonctionnement).

Le monde des systèmes transactionnels «classiques» doit donc tenir compte de l'arrivée de ces nouveaux composants, qui disposent de très peu de ressources.

Cinquième partie

Annexes

Chapitre V.1

Interfaces OTS de l'OMG

Cette annexe présente la totalité des interfaces du module CosTransactions de l'OTS spécifié par l'OMG.

Seule une partie de ce module a été implémentée pour notre maquette.

```
#include <Corba.idl>

module CosTransactions {
// DATATYPES
enum Status {
   StatusActive,
   StatusMarkedRollback,
   StatusPrepared,
   StatusCommitted,
   StatusRolledBack,
   StatusUnknown,
   StatusNoTransaction,
   StatusPreparing,
   StatusCommitting,
   StatusRollingBack
};

enum Vote {
```

```
   VoteCommit,
   VoteRollback,
   VoteReadOnly
} ;

// Structure definitions
struct otid_t {

   long formatID; /*format identifier. 0 is OSI TP
   long bqual_length;
   sequence <octet> tid;
} ;

struct TransIdentity {
   Coordinator coord;
   Terminator term;
   otid_t otid;
} ;

struct PropagationContext {
   unsigned long timeout;
   TransIdentity current;
   sequence <TransIdentity> parents;
   any implementation_specific_data;
} ;

// Forward references for interfaces defined later in module
interface Current;
interface TransactionFactory;
interface Control;
interface Terminator;
interface Coordinator;
interface RecoveryCoordinator;
interface Resource;
interface Synchronization;
interface SubtransactionAwareResource;
interface TransactionalObject;

// Heuristic exceptions

exception HeuristicRollback {};
exception HeuristicCommit {};
exception HeuristicMixed {};
exception HeuristicHazard {};

// other transaction-specific exceptions
```

```
exception Subtransactionsunavailable {};
exception NotSubtransaction {};
exception Inactive {};
exception NotPrepared {};
exception NoTransaction {};
exception InvalidControl {};
exception unavailable {};
exception SynchronizationUnavailable {};

// Current transaction
interface Current : CORBA::Current {
   void begin()
      raises(SubtransactionsUnavailable);
   void commit(in boolean report-heuristics)
      raises(
         NoTransaction,
         HeuristicMixed,
         HeuristicHazard
      );
   void rollback()
 raises(NoTransaction);
void rollback-only()
 raises(NoTransaction);

   Status get_status();

   string get_transaction_name();
   void set_timeout(in unsigned long seconds);
   Control get_control();
   Control suspends;
   void resume(in Control which)
      raises(InvalidControl);
};

interface TransactionFactory {
   Control create(in unsigned long time_out);
   Control recreate(in PropagationContext ctx);
};

interface Control {
   Terminator get-terminator()
      raises(Unavailable);
   Coordinator get-coordinator()
      raises(Unavailable);
};
```

```
interface Terminator {
   void commit(in boolean report-heuristics)
      raises(
        HeuristicMixed,
        HeuristicHazard
      );
   void rollback();
};

interface Coordinator {
   Status get_status();
   Status get_parent_status();
   Status get_top_level_status();

   boolean is_same_transaction(in Coordinator tc);
   boolean is_related_transaction(in Coordinator tc);
   boolean is_ancestor_transaction(in Coordinator tc);
   boolean is_descendant_transaction(in Coordinator tc);
   boolean is_top_level_transaction();

   unsigned long hash_transaction();
   unsigned long hash_top_level_tran();

   RecoveryCoordinator register_resource(in Resource r)
      raises(Inactive);

   void register_synchronization (in Synchronization sync)
      raises(Inactive, SynchronizationUnavailable);

   void register_subtran_aware(in SubtransactionAwareResource r)
      raises(Inactive, NotSubtransaction);

   void rollback-only()
      raises(Inactive);

   String get_ransaction_name();
   Control create_subtransaction()
      raises(subtransactionsUnavailable, Inactive);
   PropagationContext get_txcontext ()
      raises(Unavailable);
};

interface RecoveryCoordinator {
   Status replay_completion(in Resource r)
      raises(NotPrepared);
```

```
};

interface Resource {
   Vote prepare()
      raises(
        HeuristicMixed,
        HeuristicHazard
      );
   void rollback()
      raises(
        HeuristicCommit,
        HeuristicMixed,
        HeuristicHazard
      );
   void commit()
      raises(
        NotPrepared,
        HeuristicRollback,
        HeuristicMixed,
        HeuristicHazard
      );
   void commit-one-phase()
      raises(
        HeuristicHazard
      );
   void forget();
};

interface TransactionalObject {
};

interface Synchronization : TransactionalObject {

   void before_completion();
   void after_completion(in Status status);
};

interface SubtransactionAwareResource : Resource {
   void commit_subtransaction(in Coordinator parent);
   void rollback_subtransactiono;
};

}; // Fin du Module
```

Chapitre V.2

Acronymes utilisés

- A -

ACID : Atomicité, Cohérence, Isolation, Durabilité
AP : Application Program (X/OPEN)
APDU : Application Protocol Data Unit (Carte à microprocesseur)
API : Application Programming Interface
ASP : Active Server Page (Microsoft)
ATR : Anwers To Reset : Réponse à l'initialisation de la carte (Carte à microprocesseur)

- B -

BPS : Bits Par Seconde

- C -

CB : Carte Bancaire
COA : Carte Object Adaptor
COM : Component Object Model (Microsoft)
CORBA : Common Object Request Broker Architecture
COST : Carte Orientée Services Transactionnels
CPU : Central Processing Unit
CQL : Card Query Language (Carte à microprocesseur)

CRM : Communication Ressource Manager (X/OPEN)

- *D* -

DCOM : Distributed Component Object Model (Microsoft)
DES : Data Encryption Standard
DMI : Direct Method Invocation
DTP : Distributed Transaction Processing

- *E* -

EEPROM : Electrically Erasable Programmable Read Only Memory
ETSI : European Telecommunications Standards Institute

- *G* -

GIE : Groupement d'Intéret Economique
GIOP : General Inter-ORB Protocol
GSM : Global System for Mobile communications

- *H* -

HTML : HyperText Markup Language
HTTP : HyperText Transfer Protocol

- *I* -

IEEE : Institute of Electrical and Electronics Engineers (Organisation)
IIOP : Internet Inter-ORB Protocol
IIS : Internet Information Server (Microsoft)
IP : Internet Protocol (Réseaux)
ISO : International Standardization Organisation (Organisation)

- *J* -

JDK : Java Development Kit (Java)
JTS : Java Transaction Services (Java)

- *L* -

LIFL : Laboratoire d'Informatique Fondamentale de Lille (Organisation)

- *M* -

MTS : Microsoft Transaction Server

- *N* -

NIP : Numéro d'Identification Personnel (Carte à microprocesseur)

- *O* -

OMG : Object Management Group (Organisation)
OSI : Open System Interconnection (Organisation)
OTS : Object Transaction Services
OSMOSE : Operating System for Mobile Object SErvices

- *P* -

PC : Personnal Computer
PME : Porte-Monnaie Electronique (Carte à microprocesseur)

- *R* -

RAM : Random Access Memory
RD2P : Recherche Développement Dossier Portable (Organisation)
RISC : Reduced Instruction Set Computer
RM : Ressource Manager (X/OPEN)
RMI : Remote Method Invocation (Java)
ROM : Read Only Memory
RSA : Rivest, Shamir, et Adelman

- *S* -

SIM : Subscriber Identity Module (Carte à microprocesseur)
SGBD : Système de Gestion de Bases de Données
SQL : Structured Query Language
SSL : Secure Sockets Layer
STIC : Systèmes Transactionnels Intégrants des Cartes à microprocesseur
SW : Status Word (Carte à microprocesseur)

- *T* -

TCP/IP : Transmission Control Protocol / Internet Protocol
TM : Transaction Manager (X/OPEN)

Chapitre V.3

Bibliographie

[A95] T. Alexandre,
Manipulation de données multimédia dans la carte à microprocesseur: application à l'identification biométrique et comportementale,
Thèse à l'université de Lille 1, février 1995.

[AP97] M. Abdallah, P. Pucheral,
A single phase Non Blocking Commitment Protocol,
proceedings de BDA'97, Grenoble, Septembre 1997.

[B95] D. Billard,
La Reprise dans les Systèmes Transactionnels Exploitant la Sémantique des Opérations Typées,
Thèse de Doctorat, Université de Montpellier II, Mai 1995.

[B98] D. Brienne,
TAGs et Système d'Information,
Mémoire de DEA, université de Lille 1, Juillet 98.

[B98b] P. Biget,
The Vault, an Architecture for Smartcards to Gain Infinite Memory,
CARDIS'98, Third Smartcard Research and Advanced Application Conference, proceedings Springer-Verlag, Belgique, Septembre 1998.

[BCF97] J. Besancenot, M. Cart, J. Ferrié, R. Guerraoui, P. Pucheral, B. Traverson,

Les systèmes Transactionnels : concepts, normes et produits,
Edition Hermes,ISBN 2-86601-645-9, 1997.

[BGM95] P.G. Bosco, E. Grasso, C. Moiso,
An Extended Transaction Model for an Object Distributed Architecture,
First International Workshop on High Speed Networks and Open Distributed Platforms, Berlin, Mai 1995.

[BGV96] P. Biget, P. George, J.J. Vandewalle,
How Smart Cards Can Take Benefits from Object-Oriented Technologies,
CARDIS'96, Second Smartcard Research and Advanced Application Conference, Hartel, Paradinas, Quisquater eds, Septembre 1996.

[BHG87] P.A. Bernstein, V. Hadzilacos, and N. Goodman,
Concurrency Control and Recovery in Database Systems,
Addison-Wesley, 1987

[BN97] P.A. Bernstein, E. Newcomer,
Principles of Transaction Processing for the systems professional,
M. Kaufmann Publishers, 1997.

[BSW88] C. Beeri, H.J. Scheck, G. Kaiser,
Multi-Level Transaction Management, Theorical Art or Practical Need?,
In proceedings International Conference on Extending Database Technology, pp134-155, Venise, Mars 1988.

[C92] V. Cordonnier,
Assessing The Future of Smart Cards,
In proceedings CardTech, Septembre 1992.

[C94] C. Cormier,
PicoRisc, Architecture d'un microprocesseur spécifique aux cartes à accès protégés,
Mémoire de DEA, Université de Lille 1, Juin 1994

[C96] V. Cordonnier,
The future of SmartCards : Technology and Application,
In proceedings IFIP World Conference on Mobile Communication, Camberra, Septembre 1996.

[C98] D. Carlier,
Représentation permanente, coordonnée par une carte à microprocesseur, d'un utilisateur mobile,
Thèse à l'université de Lille 1, Janvier 1998.

[CFG94] M. Cart, J. Ferrié, M. Guerni, J.F. Pons,
The Impact of typed Objetcs on the update in place and deferred update transaction models,
In proceedings International Symposium on Computer ans Information Sciences, Turquie, Novembre 1994.

[CFR89] M. Cart, J. Ferrié, H. Richy,
Contrôle de l'exécution des transactions concurrentes,
Technique et Sciences Informatiques, Vol. 8, pp225-240, 1989.

[CLT96] D. Carlier, S. Lecomte, P. Trane,
SmartCard use to manage user's mobility,
CARDIS'96, Second Smartcard Research and Advanced Application Conference, Hartel, Paradinas, Quisquater eds, Septembre 1996.

[CLT97] D. Carlier, S. Lecomte, P. Trane,
The use of a Representation as a Solution to Wireless Drawbacks: Application in the context of Smart Card,
IEEE-ICPWC'97, Bombay, Décembre 1997.

[CR91] P. Chrysanthis, K. Ramamritham,
Synthesis of Extended Transaction Models using ACTA,
In proceedings Very Large Data Bases 1991.

[CT96] D. Carlier, P. Trane,
On the importance of Detecting Sites Failure in Moblie Computing Systems,
Institute of Electronics, Information and Communication Engineers, spring conference , Mars 1995.

[D94] D. Donsez,
WEA, un Gérant d'Objets Persistants pour des Environnements Distribués,
Thèse à l'université de Paris VI, Septembre 1994.

[DGL98] D. Donsez, G. Grimaud, S. Lecomte,
Recoverable Persistent Memory for SmartCard,
CARDIS'98, Third Smartcard Research and Advanced Application Conference, proceedings Springer-Verlag, Belgique, Septembre 1998.

[E92] A. K. Elmagarmid,
Database Transaction Models for Advanced Applications,
M. Kaufmann Publishers, 1992.

[ECC94] European Communities Commision,
ITSEC: Information Technology Security Evaluation Criteria,
Office de publications officielles de la communauté Européenne, ISBN 92-826-7024-4, 1994.

[EN98] F. Engel, J. Nyholm,
Secure Banking Applications: The Nordbanken Case,
EUROSEC'98, France, Mars 1998.

[ETSI95] European Telecommunication Staandard Institue,
European digital cellular telecommunication system (phase 2); Specification of the Subscriber Identity Module - Mobile Equipement interface

(GSM 11.11),
ETSI, ETS 300 608, 1995.

[Esp93] Esprit program EP8670,
CASCADE: Chip Architecture sor Smart CArd and portable intelligent DEvices,
projet européen ESP8670, 1993

[Esp95] Esprit program EP8670,
CASCADE: Operating System,
projet européen ESP8670 , Avancement du développement du Système d'Exploitation, Avril 1995.

[FM96] J. S. Fritzinger, M. Mueller,
Java Security,
Technical Report, Sun Microsystems, 1996.

[G81] J. Gray,
The Transaction Concept: Virtues and Limitations,
Very Large DataBases 1981, pp 144-154, Septembre 1981.

[Gem93] Gemplus, *CQL-Card Reference Manual*, 1993.

[Gem94] P. Paradinas, J.J. Vandewalle,
Procédé de conduite d'une transaction entre une carte à puce et un système d'information,
Brevet Français, numéro de publication 2-720-848, Gemplus, France, 1994.

[Gem96] P. Biget, P. George, S. Lecomte, P. Paradinas, J.J. Vandewalle,
Procédé de chargement d'un programme d'utilisation dans un support à puce,
Brevet international, numéro d'enregistrement 96-162-12, Gemplus, France, 1996.

[Gem98] Gemplus,
Tutorial GemXpresso Version 1.0,
Documentation du Kit de développement GemXpresso 1.0, 1998.

[GFP95] M. Guerni, J. Ferrié, J.F. Pons,
Concurrency and Recovery for Typed Objects using a new Communativity Relation,
In proceedings Deductive and Object Oriented Databases, Singapour, Décembre 1995.

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

[GGM97] J.M. Geib, C. Gransart, P. Merle,
CORBA : Des concepts à la pratique,
édition Masson, ISBN 2-225-83046-0, Aout 1997.

[GLJ97] J. Gray, R. A. Lievano, R. Jennings,
Microsoft Transaction Server 2.0,
Reger Jennings' Database Workshop, ISBN 0-672-31130-5, Indianapolis, 1997.

[GLS96] R. Guerraoui, M. Larrea, A. Shiper,
Reducing the Cost for Non-Blocking in Atomic Commitment,
16th IEEE ICDCS, 1996.

[GS87] H. Garcia-Molina, K. Salem,
SAGAS,
Preceedings ACM International Conference on Management of Data, pp 249-259, San-Francisco, mai 1987.

[GUT97] S. B. Guthery
Java Card: Internet Computing on a Smart Card,
IEEE Internet Computing Vol 1 No 1, Jan-Fev 1997

[GMB81] J. Gray, P. McJones, M. Blasgen, B. Lindsay, R. Lorie, T. Price, F. Putzolu, I. L. Traiger,
The Recovery Manager of the System R database Manager,
ACM Computer Survey, pp223-242, Juin 1981.

[GR93] J. Gray, A. Reuter,
Transaction processing Concepts and Techniques,
Morgan Kaufman, 1993.

[GR97] G. Grimaud,
Mecan'OS, Evaluation des techniques de génie logiciels utilisables dans le domaine des dossiers portables,
Mémoire de DEA, Université de Lille 1, juillet 97.

[GV89] G. Gardarin, P. Valduriez,
Relationnal Databases and Knowledge Bases,
Addison-Wesley Publishing Company, 1989.

[H92] M. P. Haye,
Designing an Information System Using Smart Cards: a network and distributed data base approach,
In proceedings, Congrès IFIP, Ottawa 1992.

[HR83] T. Haerder, A. Reuter,
Principles of Transaction-Oriented Database Recovery,
ACM Computing Surveys, Vol. 15, 1983

[ID96] A. Freier, P. Karlton, P. Kocher,
The SSL Protocol Version 3.0,
Work In Progress, Internet Draft, Transport Layer Security Working Group, IETF, Novembre 1996.

[Iso87] International Standardization Organisation (ISO),
Cartes d'identification - Cartes à circuit intégré à contacts - Partie 1 : Caractéristique physiques,
Norme ISO 7816-1, 1987

[Iso88] International Standard Organisation (ISO),
Cartes d'identification - Cartes à circuit intégré à contacts - Partie 2 : Dimensions et emplacements des contacts,
Norme ISO 7816-2, 1988

[Iso92] International Standard Organisation (ISO),
Information Technology - Database Languages- SQL,
Norme ISO/IEC 9075, , 1992.

[Iso93] International Standard Organisation (ISO),
Information Transfert, Retrieval and Management for OSI,
Norme ISO/IEC JTC 1/SC 21, Mai 1993

[Iso94] International Standard Organisation (ISO),
Cartes d'identification - Cartes à circuit intégré à contacts - Partie 5 : Système de numérotation et procédure d'enregistrement d'identificateurs d'applications,
Norme ISO 7816-5, 1994

[ISO94b] International Standard Organisation (ISO),
Infomation Technologie, Open Systems Interconnection,
Juin 1994.

[ISO94c] ISO International Standards Organization (ISO),
Information Technology - Open System Interconnection - The Directory: Authentication Framework,
Juin 1994.

[Iso95] International Standard Organisation (ISO),
Cartes d'identification - Cartes à circuit intégré à contacts - Partie 4 : Commandes intersectorielles pour les échanges,
Norme ISO 7816-4, 1995

[Iso96] International Standard Organisation (ISO),
Cartes d'identification - Cartes à circuit intégré à contacts - Partie 6 : Eléments de données intersectoriels,
Norme ISO 7816-6, 1996

[ISO96b] International Standard Organisation (ISO),
Distributed Transaction Processing OSI-TP,
Norme ISO/IEC 10026, 1996.

[JAV97] Sun Microsystems,
Remote Method Invocation Specification,
Rev 1.4, JDK 1.1, Fevrier 1997.

[JC97] Sun Microsystems,
JavaCard 2.0 Application Programming Interfaces,
Rev 1.0 final, Octobre 1997.

[JC97b] Sun Microsystems
JavaCard 2.0 Language Subset and Virtual Machine Specification,
Rev 1.0 final, Octobre 1997.

[JAV98] Sun Microsystems,
Java Platform 1.2 Beta 4 Core API,
Juillet 1998.

[JAV98b] Sun Microsystems,
JTS : Java Transaction Service API, version 0.7,
Juillet 1998.

[KR81] H. T. Kund et J. T. Robinson,
On Optimistic Method for Concurrency Control,
ACM Transaction Database System, Juin 1981.

[L94] S. Lecomte,
KIOTO : Objets Nomades et Travail Coopératif,
Mémoire de DEA, Université de Lille 1, Juillet 1994.

[L97] J. Liang,
The Comcepts and Specifications os Open Distributed Transaction Processing Platforms,
These de l'Ecole Nationale Supérieure des Télécommunications, 1997.

[LC97] D.E. Lowell, P.M. Chen,
Free Transactions with Rio Vista,
In proceedings of 16th Symposium on Operating Systems Principles, Octobre 1997.

[LD97] S. Lecomte, D. Donsez,
Intégration d'un Gestionnaire de Transaction dans les Cartes à Microprocesseur,
Colloque International NOTERE'97, Pau, France, Novembre 97

[LS76] B. Lampson, H. Sturgis,
]em Crash Recovery in a distributed data storage system,
Rapport Technique, Computer Science Laboratory, Xerox, 1976.

[LT97] S. Lecomte, P. Trane,
Failure Recovery Using Action Log for Smartcard Transactions based Systems,
IEEE On Line Testing Workshop 1997, Crète, Juillet 1997.

[LS98] J. Liang, S. Sédillot,
MAAO OTS: An OMG Object Transaction Service based on CORBA,
White Paper INRIA, Révision 1.1, Mars 1998.

[M82] J. Madelaine,
Evaluation d'Algorithmes de Contrôle de Concurrence,
Rapport de recherche de l'Institut National de Recherche en informatique et Automatique N164, Octobre 1982.

[M85] J. E. B. Moss,
Nested Transaction : an Approach to Reliable Distributed Computing,
MIT Press series in Information systems, 1985.

[Mic97] Microsoft Corporation,
Transaction Server - Transactionnal Component Services,
White Paper (disponible sur le site internet Microsoft), Decembre 1997.

[MVD96] P. Merle, J.J. Vandewalle, E. Dufresne,
Intégration d'environnements hétérogènes : Wold Wide Web, Cartes à Microprocesseur et Corba,
Inforsid 1996 Bordeaux, pp235-248, Juin 1996.

[NBS77] Nationnal Bureau of Standards,
Data Encrytion Stardard, DES,
Federal Information Processind Standards, FIPS-46, 1977.

[NP97] B. Noclercq, J.M. Place,
A Advanced Card Operating System on the Cascade Platform,
EMMSEC'97 Conference, Novembre 1997.

[OMG97] Object Management Group,
The Common Object Request Broker : Architecture and specification version 2.1,
Septembre 1997.

[OMG97b] Object Management Group,
Naming Services,
CORBA Services Specification, Chapitre 10, décembre 1997

[OMG97c] Object Management Group,
Object Transaction Services,
CORBA Services Specification, Chapitre 14, décembre 1997

[OMG97d] Object Management Group,
Security Services,
CORBA Services Specification, chapitre 15, décembre 1997.

[ORB98] Object Oriented Concepts Inc,
ORBacus 2.0.1 Reference Manual,
Documentation Technique, Disponible sur le site Web de OOC, http://www.ooc.com. Mise à jour permanente.

[OV91] M. Tamer Ozsu, P. Valduriez,
Principles of Distributed Databases System,
Prentice Hall Intl., ISBN 0-13-715681-2, 1991.

[P88] P. Paradinas,
La Biocarte : Intégration d'une carte à microprocesseur dans un réseau professionnel santé,
Thèse à l'université de Lille 1, Novembre 1988.

[P95] T. Peltier,
La Carte Blanche : Un nouveau Système d'exploitation pour objets nomades,
Thèse à l'université de Lille 1, Décembre 1995.

[P98] J. Posegga,
Java Smart Cards as a Platform for Electronic Commerce,
In proceedings of International IFIP Working Conference on Trends in Electronic Commerce, TREC'98, Hambourg, Juin 1998.

[R92] M. Ruffin,
KITLOG Un service de journalisation générique,
thèse à l'université de Paris VI, Septembre 1992.

[RSA78] R. L. Rivest, A. Shamir, L. Adleman,
A method for Obtaining Digital Signature and Public Key Cryptosystem,
Communication of the ACM, vol.21, février 1978.

[RLS78] D. Rosenkrantk, R. Stearns, P. Lewis,
System of Level Concurrency Control for Distributed Database Systems,
ACM Transactions on Database Systems, Juin 1978.

[S81] D. Skeen,
Non Blocking Commit Protocols,
ACM SIGMOD International Conference on Management of Data, ACM Press. 1981.

[S96] W.R. Stevens,
TCP/IP illustré, Les protocoles,
Volume 1, Editions International Thomson Publishing France, 1996

[Sch96] B. Schneier,
Applied Cryptography, 2nd edition,
John Wiley & Sons, 1996.

[SES96] G.I.E. Sesam Vitale,
Cahier des charges SESAM-Vitale 1996,
Version 1.0, Décembre 1996.

[SET96] MaterCard et Visa,
Secure Electronic Transaction, Book 3: Technical Specifications,
Juin 1996.

[T89] A. Tanenbaun,
Les Systèmes d'Exploitation : Conception et mise en oeuvre,
InterEditions, ISBN 2-7296-0259-2, Janvier 1989.

[T91] B. Traverson,
Stratégies d'optimisation et évaluation de performance du protocole de validation à deux phases,
Thèse à l'université paris VI, Septembre 1991.

[T95] P. Trane,
Conception et Réalisation d'un système de contrôle d'accès pour la carte à microprocesseur,
Thèse de Donctorat à l'Université de Lille 1, Septembre 1995.

[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, pp317-322, Grenoble, Avril 1995.

[V97] 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.

[VaH96] 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.

[VV98] J.J. Vandewalle, E. Vetillard,
Developing SmartCard based Applications Using JavaCard
, CARDIS'98, Third Smartcard Research and Advanced Application Conference, proceedings Springer-Verlag, Belgique, Septembre 1998.

[XOpen96] X/OPEN Guide,
Distributed Transaction Processing Reference Model Version 3,
1996.

UNIVERSITÉ DES SC ET TECHN
BIBLIOTHÈQUE UNIVERSITAIRE
LILLE