Laboratoire d'Informatique Fondamentale de Lille

Thèse

présentée par

L'Université des sciences et Technologies de Lille

pour l'obtention du titre de

Docteur en Informatique

par

Jean-Jacques Vandewalle

Projet OSMOSE : modélisation et implémentation pour l' interopérabilité de services carte à microprocesseur par l'approche orientée objet

Soutenue le 27 mars 1997 devant le jury :

Président :	Jean-Marc Geib, prof. LIFL
Rapporteur :	Roland Balter, Prof. IMAG-INRIA
	André Gamache, Prof. Université Laval (Québec)
Examinateur :	Gérard Comyn, DG III
	Vincent Cordonnier, Prof. LIFL
	Christophe Gransart, MdC. LIFL
	Philippe Maes, Gemplus
	Pierre Paradinas, Gemplus

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.F. d'I.E.E.A. Bât M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 03.20.43.47.24 Fax. 03.20.43.65.66

« *Les ordinateurs ne seraient pas les puissants objets culturels qu'ils représentent désormais si leurs utilisateurs ne tombaient pas amoureux de leurs machines et des idées qu'elles produisent.* »

Sherry Turckle. Citée par Ingrid Carlander, *Le Media Lab aux avant-postes du cybermonde*, *in* Le Monde Diplomatique, n° 509, août 1996.

À Alfred et Cathy.

Préface

Les travaux présentés dans ce mémoire ont été effectués dans l'équipe de Recherche et Développement sur le Dossier Portable (RD2P) du Laboratoire d'Informatique de l'Université de Lille (LIFL). Cette équipe regroupe quatre partenaires : l'Université des Sciences et Technologies de Lille (Lille 1), l'Université de Droit et de Médecine (Lille 2), le Centre National de la Recherche Scientifique (CNRS) et l'industriel Gemplus, premier fabricant au monde de cartes à mémoire et de cartes à microprocesseur.

Ces travaux ont été réalisés dans le cadre du projet de recherche *OSMOSE* (Operating System and MObile SErvices) concernant les systèmes d'exploitation carte et l'intégration de la carte dans les systèmes d'information. Les objectifs de ce projet sont :

1. une nouvelle génération de carte à microprocesseur générique capable de charger de nouveaux services à tout moment de son utilisation,

2. le développement d'outils d'intégration et d'interopérabilité entre les cartes (supports de services personnels) et les systèmes ouverts,

3. des outils de gestion de la cohérence de transactions entre des cartes et des systèmes d'information, et

4. l'étude des services de mobilité intégrant des cartes à microprocesseur.

Ce projet est bien sûr le fait de toute une équipe. Ce mémoire présente les travaux menés sur les deux premiers points, à savoir : un nouveau système d'exploitation pour carte à microprocesseur appelé $C_OM\acute{b}^O$ et la définition d'une passerelle générique et dynamique entre cette carte et les systèmes d'informations orientés objet (le Card Object Adapter).

Une partie des réflexions ayant conduit à ces résultats a été enrichie par notre participation au projet européen ESPRIT EP 8670 intitulé CASCADE (Chip Architecture for Smart CArds and portable intelligent DEvices) de l'Open Microprocessor Initiative et par notre collaboration au programme interlaboratoires sur la communication avancée de la région Nord-Pas de Calais (Ganymède). À l'inverse, une partie des résultats présentés ici a été reprise par ces différents programmes.

Enfin, une partie des travaux présentés dans ce mémoire est actuellement transférée chez Gemplus dans une équipe de recherche travaillant sur une nouvelle génération de carte à microprocesseur générique.

Remerciements

Je tiens à remercier tout particulièrement Messieurs Vincent Cordonnier et Pierre Paradinas pour les conseils, encouragements et preuves d'amitié qu'ils m'ont prodigués durant les années passées et jusqu'à aujourd'hui. Sans leur encadrement et leur soutien, ce travail n'aurait pas pu voir le jour.

Je remercie Monsieur le Professeur Jean-Marc Geib qui me fait l'honneur de présider le jury de cette thèse, Messieurs les Professeurs Roland Balter et André Gamache pour avoir accepté de rapporter sur mon travail, ainsi que Monsieur Christophe Gransart pour ses remarques et sa relecture très attentive de ce mémoire.

Je ne saurais pas oublier les personnes avec qui j'ai été amené à travailler à diverses étapes de ce doctorat :

- Monsieur Gérard Comyn, pour une riche discussion sur le sujet de cette thèse avant même qu'elle ne débute,
- chez Gemplus, Messieurs Philippe Maes, Gilles Lisimaque et Éric Alzai, pour leur aide et leur support continus,
- à Québec, Monsieur André Gamache et toute l'équipe du projet « Carte Santé » autour de Monsieur Guy Lavoie, pour leur accueil et leur enthousiasme,
- à Lille, Messieurs Philippe Merle, Christophe Gransart, Sylvain Lecomte et Éric Dufresne, pour nos travaux en commun et pour le plaisir pris ensemble à les mener,
- aux étudiants, Messieurs Emmanuel Horckmans, Thomas Chamussy, Benoît Scieur et Marc Bianco, pour leur travail et surtout leur patience.

Enfin, il y a ceux avec qui je continue de travailler et pour lesquels j'espère manifester ma gratitude quotidiennement : Messieurs Patrick George et Patrick Biget.

Concluons, sans citer, en signalant ceux qui par leur amitié, leurs soutiens moral et matériel ou, tout simplement, leur existence, m'ont aidé à vivre et donc à réaliser ce travail.

Comment lire ce mémoire

Chacun des chapitres de ce mémoire est intitulé de façon à décrire relativement clairement ce qu'il contient. En principe, les chapitres doivent se lire consécutivement mais ils ont été conçus pour pouvoir être extraits plus ou moins indépendamment des autres. C'est pourquoi, se trouvent, en tête de chacun d'eux, un résumé, une table des matières, et que sont disposées, à la fin de chaque chapitre, les références bibliographiques.

Les chapitres sont regroupés selon quatre parties : « Problématiques » dans laquelle sont exposés les problèmes qui nous préoccuppent (chapitres 1, 2 et 3), « Réalisations » expose les travaux que nous avons réalisés (chapitres 4, 5, 6 et 7), « Conclusion » contient le dernier chapitre de conclusion et perspectives (chapitre 8) et « Annexes » fournit les

appendices. Brièvement, les chapitres contiennent :

Chapitre 1 fournit une introduction plaçant nos objectifs dans le contexte plus général des systèmes à grande échelle.

Chapitre 2 donne l'état de l'art des cartes à microprocesseur et leurs inaptitudes actuelles à satisfaire les besoins de cartes génériques.

Chapitre 3 expose les travaux réalisés sur l'interopérabilité des cartes et des systèmes informatiques jusqu'à aujourd'hui et leurs limites.

Chapitre 4 établit notre modèle général de système à cartes génériques.

Chapitre 5 propose un protocole d'authentification des utilisateurs de cartes génériques.

Chapitre 6 décrit la machine virtuelle et l'interpréteur, cœur du système d'exploitation de la carte générique C_oM^O.

Chapitre 7 présente les principes du Card Object Adapter servant de passerelle de communication entre CORBA et la carte C_oM^O.

Chapitre 8 conclut et dresse quelques perspectives comme suite à notre travail.

Enfin, un glossaire regroupe les définitions des termes importants employés dans ce mémoire, ainsi que la signification des acronymes. La première apparition d'un mot présent dans le glossaire est signalée par le symbole †.

Les références bibliographiques regroupées par chapitre sont aussi référencées de manière globale dans l'index : les titres des publications citées à l'entrée « Références bibliographiques » et leurs auteurs à l'entrée « Auteurs cités ».

Le symbole ◀▶ en marge d'un paragraphe signale que celui-ci décrit un travail proche du nôtre. Ces travaux sont regroupés dans l'index à l'entrée « Travaux voisins ».

Le symbole *** indique que le paragraphe fournit un tour d'horizon bibliographique sur un sujet particulier, l'entrée « Bibliographies » référence ces paragraphes dans l'index.

Table des matières

Liste des tableaux

Table des figures

Première partie

Problématiques

CHAPITRE 1

Introduction

« Dès lors les techniques se perfectionnent
La carte à puce remplace le Remington »

M.C. Solar. *Nouveau Western*, Paris, France, 1993.

Résumé

Dans ce chapitre nous définissons les applications de type systèmes à grande échelle dans lequel se situe notre travail sur les cartes à microprocesseur et leur intégration dans les systèmes informatiques. Nous donnons ensuite un modèle général de ces applications, ce modèle nous servant par la suite de référence pour définir la carte comme un serveur personnel dans ces systèmes. Enfin, nous aboutissons à notre double problématique : comment permettre le chargement de nouveaux services dans les cartes à microprocesseur ? et comment ces services peuvent-ils s'intégrer avec les éléments informatiques extérieurs ?

Sommaire

Tables

Figures

1.1 Les systèmes à grande échelle

Le cadre général du projet *OSMOSE* est celui des *systèmes à grande échelle*†. Ces systèmes informatiques sont en premier lieu caractérisés par le fait qu'ils sont répartis sur une grande zone géographique (une région, un continent, ou même le monde). Il s'agit d'applications qui, par nature, mettent en jeu des utilisateurs et des ressources distribués sur un vaste espace. Des exemples de tels systèmes sont le World Wide Web [Ude96, Hug94], le réseau européen de téléphone cellulaire GSM[1] [Sco95], ou encore le projet IRIDIUM[2].

Les systèmes informatiques à grande échelle connaissent aujourd'hui un essor de plus en plus important. L'interconnexion des réseaux et des télécommunications, mais aussi le développement de la téléphonie cellulaire et de la monnaie électronique favorisent l'émergence de telles applications. Ces applications reposent sur de vastes systèmes distribués, mais aussi sur des outils portables mis à la disposition des utilisateurs. La carte à microprocesseur est un outil priviligié pour personnaliser et sécuriser l'accès et l'utilisation des services que ces applications proposent. Il suffit pour s'en convaincre de suivre, par exemple, l'intérêt et les recherches que suscite le problème du paiement de services sur Internet : les solutions souvent abordées intègrent des cartes à microprocesseur pour stocker de la monnaie électronique et/ou sécuriser des transactions financières.

1. Global System for Mobile communication
2. Ce projet vise à couvrir numériquement la totalité de la planète avec 66 satellites en orbite autour de la Terre (voir le site Web `http://www.iridium.com`).

La carte à microprocesseur va donc en quelque sorte devoir sortir de ses « niches écologiques » (carte bleue CB française, ou carte SIM[3] d'abonné sur le réseau GSM) pour se confronter à un environnement beaucoup plus ouvert et imprévisible. Il lui faudra alors s'adapter aux besoins des utilisateurs et aux caractéristiques des systèmes ouverts à grande échelle.

Le travail que présente ce mémoire est une contribution à ce problème sur les deux points suivants :

1. Le chargement de nouvelles fonctionnalités et de nouveaux services tout au long du cycle de vie d'une carte à microprocesseur.
2. Les outils d'intégration et d'interopérabilité des cartes à microprocesseur dans les systèmes informatiques ouverts.

1.2 Caractéristiques des systèmes à grande échelle

Les systèmes à grande échelle peuvent être définis de différentes manières que nous allons dans un premier temps énumérer pour ensuite faire ressortir les problèmes qui nous intéressent vis-à-vis de notre travail sur les cartes à microprocesseur.

La principale caractéristique des systèmes à grande échelle est essentiellement d'ordre quantitative. En effet, ces systèmes intègrent généralement un grand nombre et une importante diversité à la fois :

- *des utilisateurs*, depuis le spécialiste qui réalise le système jusqu'au grand public,
- *des relais*, c'est-à-dire des équipements informatiques (matériels et logiciels) permettant aux utilisateurs d'accéder aux ressources du système, et
- *des ressources*, les informations et les services mis à la disposition des différents utilisateurs.

Cette diversité nous semble être impossible à surmonter « d'un seul coup » dans ce type d'application. En effet, il semble improbable de pouvoir mettre en œuvre une application à grande échelle en utilisant un seul outil informatique qui intégrerait les diverses couches de logiciels depuis l'interface homme-machine jusqu'à l'implémentation des différentes ressources. De plus, ces systèmes peuvent inclure plusieurs modes d'interaction entre les éléments impliqués, pour lesquels des supports d'information hétérogènes seront utilisés. Ce qui a pour conséquence de rendre encore plus complexe la tâche d'un tel logiciel qui prendrait tout à sa charge.

Nous caractériserons donc ces systèmes par la diversité de leurs composants (utilisateurs, relais, ressources), mais aussi par la diversité des modes d'interaction qui impliquent l'utilisation de multiples technologies. Cette diversité de technologies accroît encore la diversité des utilisateurs, des relais et des ressources. Une caractéristique importante des systèmes à grande échelle est donc aussi le besoin d'uniformité rencontré à tous les niveaux (*cf.* FIG. 1.1 page suivante).

3. Subscriber Identification Module

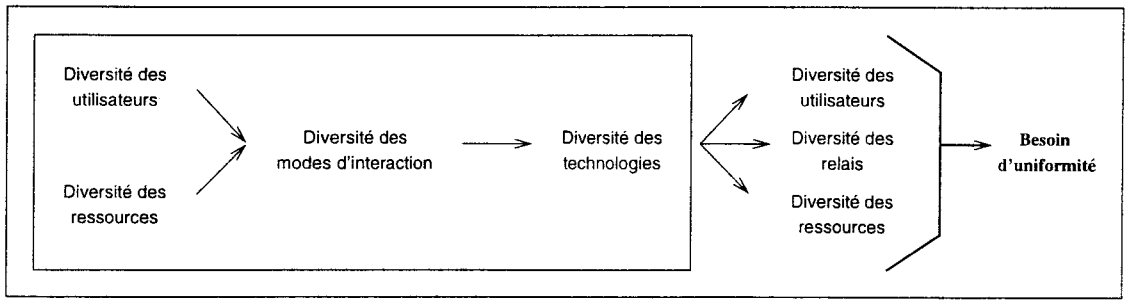

FIG. 1.1 - Caractéristiques des systèmes à grande échelle

Dans les deux paragraphes suivants nous détaillons les différents modes d'interaction. puis les besoins d'uniformité engendrés par la diversité des utilisateurs et des technologies des systèmes à grande échelle.

1.2.1 Divers modes d'interaction

Les systèmes à grande échelle fonctionnent le plus souvent selon différents modes d'interaction. Globalement, quatre grandes catégories d'interaction se dégagent. Chacune d'elle entraine l'utilisation de supports informatiques (matériel, réseau et logiciel) plus adaptés à la prise en charge des relations entre utilisateurs et ressources. La TABLE 1.1 présente ces quatre catégories d'interaction.

Catégorie d'interaction	Exemples	Principaux supports
Des individus accèdent à des individus (communication interpersonnelle)	Téléphone mobile, travail coopératif, e-mail	Réseaux de (télé)communication, interfaces homme-homme
Des individus accèdent à des ressources informatiques (autres que les ressources de leur propre machine)	Réservations Minitel, navigation World Wide Web	Serveurs de bases de données relationnelles ou objets, interfaces homme-machine
Des ressources informatiques accèdent à des ressources informatiques	Systèmes distribués	Supports de coopération entre composants logiciels (RPC, DCE, CORBA)
Des ressources informatiques accèdent à des individus (au travers de leurs données personnelles)	Applications dossier portable	Informatique nomade, carte à microprocesseur

TABLE 1.1 - Catégories d'interaction

◄► Les formes d'interaction individus-ressources et ressources-individus ont été détaillées par Van Hoecke (Marie-Pierre) [VH96] dans le cadre de la conception des systèmes d'information. Elle a défini la notion de *systèmes d'information communicationnels* décrivant les multiples formes de communication ($1 \rightarrow n$, $n \leftarrow 1$, et $n \leftrightarrow n$) entre 1 ou n individu(s) et 1 ou n système(s) d'information. Une implémentation a été réalisée dans laquelle un individu voit sa carte comme un moyen d'accès à la fédération des données qui le concernent dans des bases distantes [HC96].

Notre travail se différencie de celui-ci par le fait que nous abordons la problématique des cartes à microprocesseur et des systèmes d'information en terme d'intégration de composants technologiques hétérogènes. Un aspect qui nous intéressera dans ce mémoire sera donc principalement les passerelles de communication entre les cartes et les systèmes

informatiques implémentant un système d'information.

1.2.2 Des besoins d'uniformité

Ces diversités de modes d'interaction, d'utilisateurs, de relais et de ressources entrent en contradiction avec les objectifs qu'on pourrait attendre des systèmes à grande échelle, à savoir la délivrance de services et d'informations dont les utilisateurs ont besoin, quand, où et dans le format qu'ils souhaitent. Les besoins d'uniformité des multiples technologies impliquées dans les systèmes à grande échelle sont donc un problème crucial pour leur réussite. Nous déclinons ci-après ces besoins selon les différentes formes de diversité rencontrées.

1.2.2.1 Diversité des utilisateurs

Les utilisateurs ont des besoins et des compétences qui varient de l'usager « de base » effectuant toujours la même opération courante, jusqu'à « l'expert » capable de requérir des services complexes. Pour ces utilisateurs, il est nécessaire de mettre en place une interface conviviale et largement répandue afin de ne pas le dérouter. Néanmoins, cette interface doit aussi permettre aux utilisateurs de réaliser des transactions de plus en plus complexes avec la même ergonomie. Ceci, afin qu'un utilisateur de base puisse progressivement un jour devenir un expert sans avoir ni à changer d'outil, ni à lire nombre de manuels.

1.2.2.2 Diversité des relais

La diversité des relais utilisés implique des besoins d'interconnexion entre ceux-ci. Bien souvent pour accéder à une ressource (un service, une information) il va falloir passer successivement par une représentation de cette ressource (une API[4], un format), le système d'exploitation de la machine sur laquelle elle est stockée, des réseaux de communication et de transport pour la véhiculer, et enfin une interface homme-machine pour que l'utilisateur puisse manipuler cette ressource. Ceci implique donc que chaque couche concernée soit bien délimitée dans ses fonctions et ses interfaces avec ses couches adjacentes, mais aussi que des couches de même niveau puissent s'interconnecter pour permettre aux informations de passer de façon transparente d'un protocole ou d'une norme à un(e) autre.

1.2.2.3 Diversité des ressources

Enfin, la diversité des ressources qui peuvent être mises à la disposition des utilisateurs, demande aux concepteurs d'importants efforts de programmation si celles-ci sont stockées et programmées de différentes manières. Pour cela, il est nécessaire d'avoir recours à un modèle (ou méta-modèle) permettant de décrire uniformément les fonctionnalités de ces ressources. Le but est que les ressources deviennent accessibles indépendamment de leur implémentation machine, aussi bien pour la programmation du système que pour l'utilisation des ressources.

4. Application Programming Interface

Pour résoudre de tels problèmes il est classique et pratique de faire appel à un modèle général dans lequel se détachent plus facilement les méthodes et outils à mettre en œuvre.

1.3 Un modèle de système à grande échelle

Les besoins d'uniformité dans les systèmes à grande échelle ont été caractérisés par Brodie (Michael L.) [Bro93] sous la forme d'une architecture globale dans laquelle les qualités des *systèmes ouverts* doivent apparaître : interopérabilité, transparence, efficacité, intelligence, distribution et réutilisabilité de composants interchangeables. Le but est d'associer différemment des composants informatiques afin de générer rapidement de nouvelles applications.

Les motivations pour une telle architecture proviennent de la fusion actuelle entre les télécommunications, l'informatique, la video et l'électronique grand public en un vaste ensemble généralement identifié par le vocable de *technologies de l'information*† [Rud93]. Celles-ci sont de plus en plus appréhendées dans une architecture de « réseau intelligent » [5] permettant de mettre à disposition des utilisateurs une grande variété de services s'accordant à leurs besoins les plus personnels.

Cette architecture (*cf.* FIG. 1.2) est modélisée par un ensemble d'*agents* coopérants (individus ou ressources) qui s'interfacent avec *l'espace d'objets global* au travers une interface d'accès propre à leurs caractéristiques. Chaque agent peut agir comme utilisateur ou fournisseur (client ou serveur) de l'espace d'objets. En tant que serveur, un agent peut fournir tout ou une partie de ses informations et capacités. *La couche d'interface* doit permettre un accès transparent à l'espace d'objets en servant d'intermédiaire entre les langages et formats des agents et ceux de l'espace d'objets.

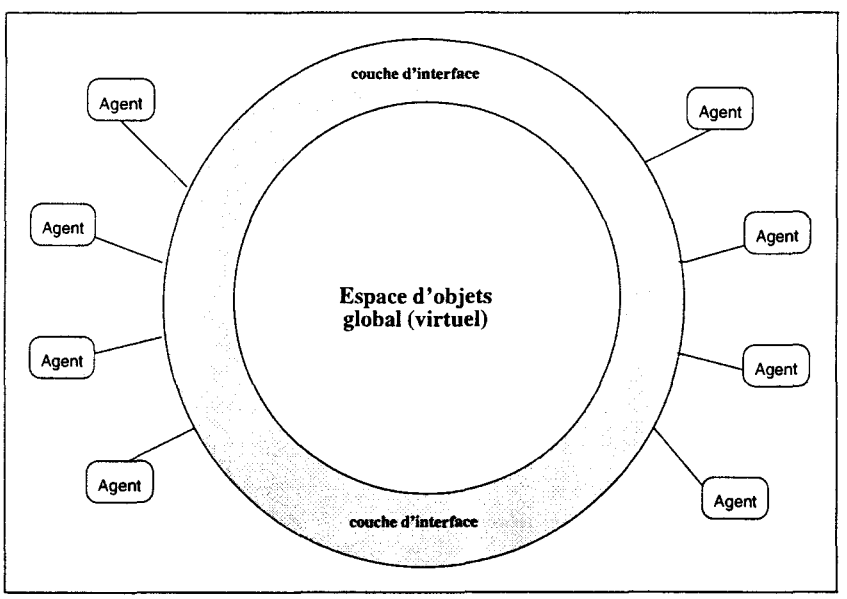

FIG. 1.2 - Modèle de système à grande échelle

5. Un des slogans de SUN MICROSYSTEMS proclame : « *The network is the computer* »...

1.3.1 L'espace d'objets global

L'espace d'objets global est virtuel car il représente l'ensemble des ressources informatiques distribuées dans les machines et potentiellement partageables entre les différents agents. Ces ressources doivent pouvoir être utilisées comme des composants indépendants pouvant s'intégrer ensemble et se recombiner de façon souple dans le but de mettre au point des formes variées d'applications. L'utilisation du paradigme objet permet d'*encapsuler* en une seule entité fonctionnelle – *l'objet* – la nature des ressources d'un système informatique (par exemple, données, documents, programmes, connaissances, services).

L'*approche orientée objet*† [MNC+91, Weg90] est un mécanisme d'abstraction dans lequel le monde est modélisé comme une collection d'objets indépendants qui communiquent les uns avec les autres par échange de messages. Un objet est caractérisé par une partie statique *privée*, un ensemble de données (ou attributs), et par une partie dynamique, un ensemble d'opérations (ou méthodes) qui agissent sur ces données. Les données et opérations sont *encapsulées* dans une même entité, appelée *objet*. Les données et l'implémentation des opérations sont cachées; les accès aux données ne sont possibles que *via* les opérations qui constituent *l'interface* de l'objet. La façon dont s'exécute une opération est de la responsabilité de l'objet. Appeler une opération d'un objet est en fait une requête, un *message* envoyé par un émetteur à un prestataire de services, demandant l'exécution d'une action.

Parmi les principes de l'approche orientée objet, ceux listés ci-dessous offrent de bonnes qualités pour la conception et la mise en œuvre des systèmes à grande échelle reposant sur la notion d'espace d'objets global [Bro93, Pan95] (*cf.* FIG. 1.3 page suivante) :

Abstraction L'implémentation et la représentation interne des objets sont masquées aux utilisateurs.

Encapsulation Chaque objet encapsule l'état qui le caractérise par des attributs et les opérations qui peuvent être appliquées sur l'objet. Il contrôle lui-même la manière d'exécuter une opération sur ses données.

Interface L'interface d'un objet est indépendante de son état et de son implémentation et peut donc être distribuée aux utilisateurs.

Modularité Les objets possédant des caractéristiques communes peuvent être regroupés dans des classes. Une classe d'objets peut ensuite être modifiée indépendamment des autres. Cette abstraction offre une modularité importante en décomposant le système en petits composants logiciels qui peuvent être réutilisés dans d'autres applications.

Identification Chaque objet a un identifiant unique qui le différencie de tous les autres objets. Cet identifiant est indépendant de l'état de l'objet.

Communication Les objets communiquent par échange de messages qui peuvent être transportés à travers différents espaces d'adressage.

De nombreuses branches de l'informatique ont été influencées par l'approche orientée objet[6] : les méthodes d'analyses, les bases de données, les langages de programmation, les

6. De plus, l'objet offre des fonctionnalités permettant de faciliter le développement d'applications : po-

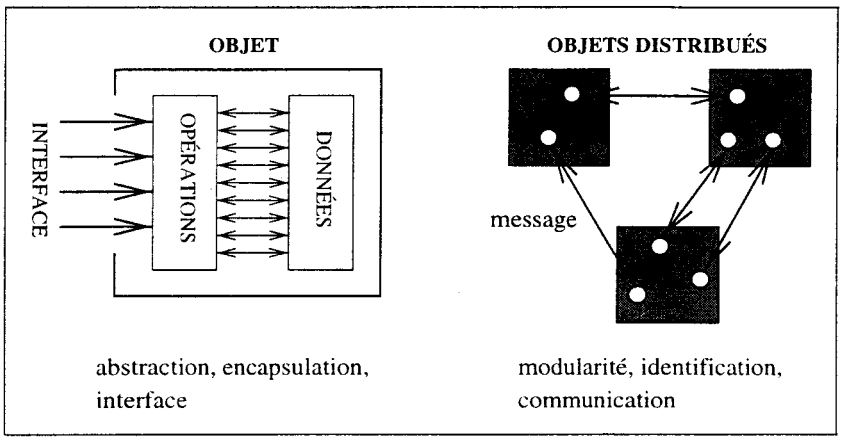

FIG. 1.3 - Principes de l'approche orientée objet

composants logiciels comme les interfaces graphiques et les systèmes distribués. Mais dans la plupart des cas, et malgré les bonnes propriétés révélées ci-dessus, les divers systèmes orientés objet ne peuvent pas coopérer ou s'interconnecter facilement. En effet, les services de base que chacun de ces systèmes utilise sont souvent beaucoup trop hétérogènes pour permettre une interopérabilité transparente des ressources.

1.3.2 Middleware d'interopérabilité

Afin de parvenir à l'*interopérabilité*† des ressources de l'espace d'objets global il est nécessaire de définir un modèle fédérateur de gestion de cette architecture. Ce modèle repose sur les idées fortes des architectures distribuées ouvertes [CDK94, chapitres 1 à 5] [Kha94b, Sol94], à savoir :

- des composants logiciels interagissant sous le mode *client-serveur*†,
- une description de ces composants dans un langage indépendant de leur implémentation (langage souvent appelé IDL[7]), et
- un ensemble de services communs distribués servant de serveurs partagés.

Le mode client-serveur est le principe général de communication. Les paramètres de communication entre clients et serveurs sont résolus par l'utilisation du langage de description des composants. Enfin, les services communs réalisent la gestion de l'espace d'objets global en prenant à leur charge le maximum de fonctions de bas niveau telles que l'environnement d'exécution, le transport des messages (requêtes, réponses), la gestion réseau, le nommage et la localisation des ressources, la sécurité, *etc.* Ces services sont fournis au dessus des plates-formes matérielles sous la forme d'une couche généralement appelée *middleware*† [Ber93] (*cf.* FIG. 1.4 page ci-contre).

lymorphisme (un même nom de traitement peut être associé à plusieurs objets, chaque objet pouvant implémenter différemment ce traitement), **réutilisabilité** (un même composant peut aussi servir par héritage à la construction de nouveaux composants), **composition** (les objets peuvent être assemblés par agrégation pour former des objets plus complexes) et **extensibilité** (la modularité, la réutilisabilité et la composition offrent une grande flexibilité pour étendre le système).

7. Interface Description Language

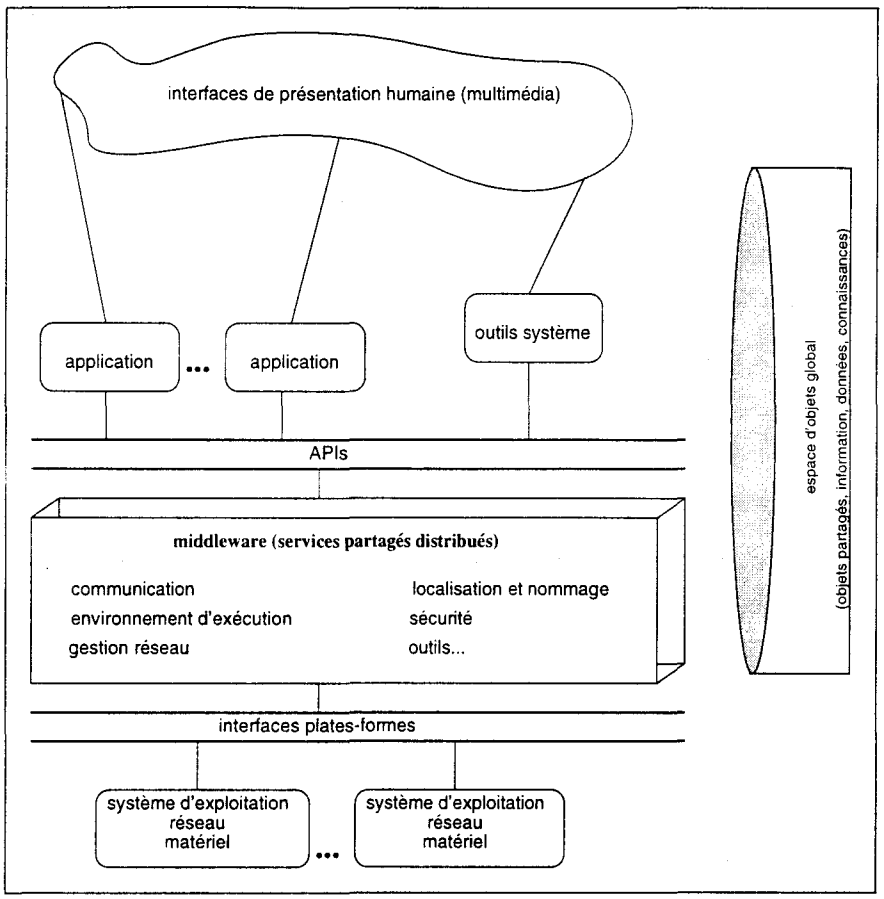

FIG. 1.4 - Couche middleware d'interopérabilité

Les services de middleware se doivent d'être *de jure* ou *de facto* standardisés pour être portables et offrir des garanties d'interopérabilité. Certains vendeurs promeuvent leurs propres interfaces middleware: par exemple, Digital avec NAS[8] et Microsoft avec WOSA[9]. Aussi, des consortiums se réunissent pour définir des ensembles limités de services middleware essentiels afin de ne pas déboucher sur des services qui pourraient se recouper dans leurs fonctions. Citons, par exemple, les travaux de l'X/Open ou de l'Open Software Foundation. Dans le domaine de l'objet, nous reviendrons longuement dans le Chapitre 7 page 123 sur les travaux de l'Object Management Group.

1.4 La carte à microprocesseur dans les systèmes à grande échelle

Le projet *OSMOSE* a pour but d'intégrer les cartes à microprocesseur dans les systèmes à grande échelle. Cet objectif est soutenu par le fait que les cartes ont un rôle à jouer dans des architectures du type « autoroutes de l'information » dans lesquels le marché grand public pourrait être le principal moteur [San95, p. 114]. Les ingrédients principaux de ce

8. Network Application Support
9. Windows Open Services Architecture

« succès annoncé » seront :

1. **la sécurité**, assurant les transactions entre les utilisateurs et les prestataires de services par des garanties contractuelles telles que l'authenticité (des parties réciproques, des informations), la validité dans le temps, le paiement, la livraison, *etc.*, et

2. **l'offre de services personnalisés**, permettant à l'utilisateur, par exemple, de contrôler la qualité de services qu'il désire, d'avoir une emprise sur les informations qui le concernent[10] ou de raffiner à volonté les prestations qu'il effectue.

1.4.1 Apports de la carte à microprocesseur

Aujourd'hui, beaucoup d'efforts ont porté sur la sécurité avec une utilisation de la carte comme unité cryptographique attachée à l'utilisateur. La carte est alors vue comme un module de sécurité portable contenant des informations secrètes (clés) ou des données infalsifiables (argent électronique). Elle est aussi capable de réaliser des traitements cryptographiques permettant à son détenteur de s'authentifier, par exemple sur une architecture distribuée du type DCE[11] [Mer95, MA94], ou de réaliser des transactions monétaires anonymes comme avec le système DigiCash de Chaum (David) [Fro96].

Pourtant la carte offre d'autres potentialités qu'il est intéressant d'encourager pour ramener les prestations de services plus proches de l'individu et lui permettre de mieux contrôler ses informations personnelles[12]. Les différentes catégories d'interaction de la TABLE 1.1 page 6 permettent de lister les différents types d'apport des cartes dans les applications à grande échelle (*cf.* TABLE 1.2).

Catégorie d'interaction	Apports de la carte
Des individus accèdent à des individus	Authentification mutuelle des interlocuteurs, fonction de filtre de communication, réponse automatique, *etc.*
Des individus accèdent à des ressources	Accès sécurisé et payant, accès personnalisé en fonction du profil et des préférences de l'utilisateur inscrits dans la carte, *etc.*
Des ressources accèdent à des ressources	La carte peut être elle-même porteuse de ressources et utilisée comme un serveur (porte-monnaie électronique), la carte peut requérir certains traitements complexes à un système hôte, *etc.*
Des ressources accèdent à des individus	Stockage d'information concernant le porteur de la carte, échanges d'information entre systèmes *via* la mise à jour de données sur la carte, *etc.*

TABLE 1.2 - Apports des cartes à microprocesseur

10. Souvent, l'utilisateur est simplement dépendant des applications qui manipulent *ses* données.

11. Distributed Computing Environment

12. En termes mercantiques on parlerait d'*autonomie* et de *responsabilité* de la personne, dans le discours philosophique, peut-être de *liberté*! La carte à microprocesseur comme toute technique peut-être nuisible à la liberté du citoyen si son usage sert à renforcer les outils de contrôle (lire le dossier « *Médias et contrôle des esprits* » du Monde Diplomatique, Manière de voir n° 27, août 95), mais elle peut aussi servir à préserver cette liberté par l'utilisation de la cryptographie pour rendre anonymes les transactions informatiques (lire Chaum (David), *Informatique et liberté*, *In* Pour la Science numéro 180, octobre 1992).

1.4.2 Trois défis carte

La fusion des technologies de l'information favorisent l'essor d'applications à grande échelle dans lesquelles les cartes à microprocesseur peuvent permettre de sécuriser, personnaliser et rendre plus flexibles les services offerts au grand public [Pey95]. Pour cela, la carte doit répondre à trois défis technologiques :

- la cryptographie à clé asymétrique,
- un système d'exploitation générique et ouvert, et
- l'intégration dans les systèmes informatiques.

1.4.2.1 La cryptographie à clé asymétrique

Les cartes doivent être capable de réaliser rapidement les calculs complexes que requièrent les systèmes cryptographiques asymétriques tels que le RSA[13]. Ces systèmes sont largement promus pour l'authentification d'entités communicantes et pour la signature de transactions électroniques dans les systèmes à grande échelle où le problème du grand nombre de clés à distribuer est crucial.

1.4.2.2 Un système d'exploitation générique et ouvert

La multiplication et la diversité des services que devront avoir à gérer les cartes nécessitent un système d'exploitation capable de suivre les besoins de l'utilisateur en permettant le chargement et le déchargement dynamique d'applications. Il s'agit de réaliser une carte générique (au même sens qu'un ordinateur personnel n'est dédié à aucune utilisation particulière) dont les avantages sont :

- pour les prestataires de services : livrer rapidement des services sans avoir à faire fabriquer une carte dédiée pour leur application,
- pour les utilisateurs : disposer d'une carte multi-fonctions[14] pour stocker, même temporairement, les services dont ils ont besoin, et
- pour le marché : se développer (quoi d'autre ?) par l'utilisation d'un nouvel outil ouvert.

1.4.2.3 L'intégration dans les systèmes informatiques

La prolifération des applications carte provoque une utilisation de la carte à microprocesseur dans une grande variété de systèmes informatiques. De plus en plus, les prestataires de services souhaitent ouvrir les services chargés dans les cartes de leurs clients à de nouveaux utilisateurs. Ce phénomène renforce encore l'hétérogéneité des machines avec lesquelles les cartes ont à s'interfacer.

13. Rivest Shamir Adelman
14. Il est déjà de coutume de se plaindre du trop grand nombre de cartes qu'il faut conserver dans son portefeuille !

1.4.3 Problématique de ce mémoire

Ce mémoire aborde donc les problèmes des systèmes d'exploitation et d'intégration. Les deux questions que nous traitons sont donc : quel système d'exploitation carte pour permettre le chargement dynamique de nouveaux services dans les cartes à microprocesseur ? Et, quelle couche middleware pour que cartes et services puissent s'intégrer dans la diversité des systèmes informatiques ? (*cf.* FIG. 1.5.)

Le problème de la cryptographie à clé asymétrique fait déjà l'objet de nombreuses recherches pour augmenter la puissance de calcul des microprocesseurs carte (par l'utilisation d'une architecture RISC [15] ou d'un coprocesseur cryptographique [16]) ou pour améliorer l'implémentation des algorithmes. Notre travail ne concerne pas ces problèmes mais en tient compte lors de la définition du protocole de sécurité d'authentification des utilisateurs de la carte *CoMbO* (*cf.* Chapitre 5 page 87).

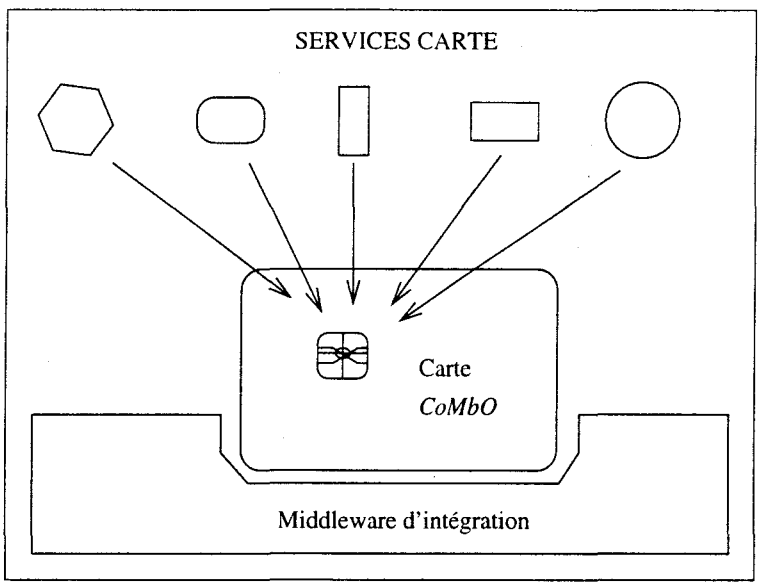

FIG. 1.5 - Problématiques de ce mémoire

Notre vision des cartes de demain est celle de serveurs portables, personnels et sécurisés qui peuvent charger des services adaptés aux besoins de leur propriétaire pour « parcourir» les autoroutes de l'information. Pour nous les cartes sont donc des éléments à part entière des systèmes à grande échelle, elles ont les mêmes capacités à faire partie de *l'espace d'objets global* (*cf.* § 1.3 page 8) en tant qu'agents d'accès mais aussi comme fournisseurs d'information et de traitement. Le problème d'hétérogénéité dans ses systèmes se pose de façon identique avec la carte, et notre « profession de foi » pour notre travail sur leur intégration sera cette définition de Bernstein (Philip A.) [Ber93] :

> « Integration *is the ability to work together in a consistent manner to perform tasks for users of an information system. Integration includes both interoperability and uniformity aspects.* Interoperability *is the extent to which a set of*

15. Reduced Instruction Set Computer

16. Processeur spécialisé dans l'exécution rapide d'algorithmes cryptographiques asymétriques grâce au câblage de fonctions arithmétiques d'exponentiation de grands nombres.

components invokes operations and exchange information effectively. Uniformity *is the extent to which a set of components is constant with respect to a set of attributes (e.g., equally distribuable, reliable or secure). Interoperability alone is not sufficient for integration, because it can be attained by gateways that are specific to each combination of components being integrated. Due to lack of uniformity, gateway-based integration often is hard to use and performs poorly.* »

1.5 Synoptique du mémoire

Ce paragraphe donne la démarche que nous allons suivre dans ce mémoire.

1.5.1 Première partie : Problématiques

Jusqu'à présent utilisée dans des applications relativement cloisonnées, chaque carte émise reste fortement liée au domaine d'application dans lequel elle est utilisée. Les cartes actuelles contiennent un système d'exploitation figé en fonction de l'application cible et répondent donc mal au besoin d'innovation et d'ouverture des systèmes ouverts. Le Chapitre 2 page 19 fait le tour de cette question et expose les besoins en matière de cartes génériques, c'est-à-dire de cartes capables de charger de nouveaux services.

À cause de la forte intrication « carte-application cible », l'interopérabilité des cartes avec les systèmes informatiques est le plus souvent limitée au(x) système(s) de base (logiciel, réseau) de l'application cible. Les cartes n'étant pas perçues comme des serveurs indépendants de l'application cible, le middleware de communication entre l'application et la carte est le plus souvent inapte à gérer toute modification. Le Chapitre 3 page 43 retrace les évolutions faites en ce domaine et énonce nos objectifs pour une intégration transparente des cartes dans les systèmes informatiques.

1.5.2 Deuxième partie : Réalisations

Notre proposition de carte générique repose sur la définition d'un nouveau cycle de vie permettant le chargement dynamique de services dans les cartes. Un modèle de système à cartes génériques est présenté dans le Chapitre 4 page 69 pour introduire les différentes réalisations qui en sont issues.

Le fonctionnement dynamique de cette carte (les services chargés et les utilisateurs ne peuvent en aucun cas être connus à l'avance) nécessite la mise en œuvre d'un protocole de sécurité lui aussi dynamique entre les utilisateurs du système à cartes génériques. Le Chapitre 5 page 87 présente ce protocole basé sur des certificats à clés asymétriques.

La sécurité entre les différents services chargés est le principal problème que doit gérer le système d'exploitation de la carte. Le Chapitre 6 page 105 présente le chargement de service dans la carte et les solutions que nous avons adoptées pour assurer la sécurité de l'exécution d'un service et l'étanchéité entre services. Ces solutions sont fondées sur le principe d'encapsulation de l'approche orientée objet et l'utilisation d'un interpréteur sécurisé pour l'exécution de code dans la carte.

L'intégration de cartes génériques dans les systèmes informatiques requiert une couche middleware offrant les qualités d'uniformité et d'interopérabilité. Le Chapitre 7 page 123 présente une passerelle générique et dynamique pour accéder à des services carte *via* une couche d'intégration dans les architectures de systèmes distribués orientés objet du type CORBA[17].

1.5.3 Troisième et quatrième partie : Conclusion et annexes

Pour conclure, nous donnons l'état du projet et les travaux qui continuent actuellement. Enfin, des perspectives sont dressées, notamment avec les autres thèmes du projet *OSMOSE*.

En annexe, nous détaillons principalement (Chapitre A page 145) les fonctionnalités de chacune des commandes disponibles à l'interface de la carte $C_OM\mathcal{B}^O$.

Bibliographie introductive

[Ber93] Bernstein (Philip A.). – *Middleware: An Architecture for Distributed System Services.* – Rapport de recherche n° 93/6, DEC CRL Digital Equipment Corporation Cambridge Research Lab, mars 1993.
http://www.research.digital.com/CRL/abstracts/93.6.html.

[Bro93] Brodie (Michael L.). – The promise of distributed computing and the challenges of legacy information systems, *In : Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems (DS-5), Lorne, Victoria, Australia. 16-20 November, 1992.* – pp. 1–31, 1993.

[Car95] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1995 conference proceedings, April 10 to 13, 1995, Washington, DC, U.S.A.*, avril 1995.

[CDK94] Coulouris (George), Dollimore (Jean) et Kindberg (Tim). – *Distributed Systems: Concepts and Design*, Assison-Wesley Publishing Company Inc., 1994, seconde.
http://www.dcs.qmw.ac.uk/research/distrib/book.html#guide.

Livre de cours sur les sytèmes distribués qui s'appuient sur de nombreuses études de cas. Les cinq premiers chapitres expliquent les notions de base des systèmes distribués (caractéristiques, implémentation, communications inter-processus et inter-composants au dessus d'un réseau). Les autres chapitres traitent des systèmes d'exploitation distribués, de la gestion des données partagées et des services de gestion de fichiers, de disponibilité, de sécurité et de nommage. Nombreux exercices en fin de chaque chapitre avec, pour les enseignants, des informations en ligne sur le serveur Web.

[Fro96] Froomkin (A. Michael). – *Flood Control on the Information Ocean: Living With Anonymity, Digital Cash, and Distributed Databases.* – Version 1.7 en cours de rédaction, à paraître dans le « University of Pittsburgh Journal of Law and Commerce», University of Miami School of Law, 1996.
http://www.law.miami.edu/~froomkin/articles/ocean.htm.

[HC96] Haye (Marie-Pierre) et Caron (Olivier). – ADEPTE project: an individual federated database on a smart card, *In : Proceedings of RIDE-NDS'96 Sixth International Workshop on Research Issues on Data Engineering: Interoperability of Nontraditional Database Systems, February 26-27, 1996, New Orleans, USA.* – IEEE Computer Society, février 1996.

17. Common Object Request Broker Architerture

[Hug94] Hughes (Kevin). – *Entering the World-Wide Web: a guide to Cyberspace.* – Version 6.1, Entreprise Integration Technologies, mai 1994.
http://www.eit.com/goodies/www.guide/.

[Kha94a] Khanna (Raman). – *Distributed Computing*, Prentice-Hall, 1994.

[Kha94b] Khanna (Raman). – Introduction, chap. 1, pp. 1–24. – In [Kha94a].

[MA94] Merckling (Roger) et Anderson (Anne). – *DCE Smart Card Integration.* – OSF/DCE SIG Request For Comments n° 57.1, Hewlett-Packard, mars 1994.
http://ice-www.larc.nasa.gov/ICE/SECURITY/bulletins/FIRST/caa/rfc571.txt.

[Mer95] Merckling (Roger). – ODISSEY: The Innovative Approach to Smart Card Integration into DCE, pp. 117–124. – In CardTech/SecurTech, Inc. [Car95].

[MNC+91] Masini (Gerald), Napoli (Amedeo), Colnet (Dominique), Léonard (Daniel) et Tombre (Karl). – *Les langages à objets*, InterEditions, 1991.

[Pan95] Pancake (Cherri M.). – The Promise and the Cost of Object Technology: A Five-Year Forecast. *Communications of the Association for Computing Machinery*, vol. 38, n° 10, octobre 1995, pp. 33–49.

[Pey95] Peyret (Patrice). – Which Smart Card technologies will you need to ride the Information Highway safely?, – *In: Proceedings os Smart Cards'95, London, England*, 1995.
http://www.dice.ucl.ac.be/~dhem/cascade/scard95.html.

[Rud93] Rudge (Alan W.). – I'll be seeing you. *IEEE Review*, novembre 1993, pp. 235–238.

[San95] Sandoval (Victor). – *Les autoroutes de l'information*, Hermès, 1995.

[Sco95] Scourias (John). – *Overview of the Global System for Mobile Communications.* – Rapport technique, University of Waterloo, mai 1995.
http://ccnga.uwaterloo.ca/~jscouria/GSM/gsmreport.html.

Article donnant une très bonne vue d'ensemble du système GSM : son histoire, les services fournis, l'architecture du réseau (station mobile, sous-système station de base et sous-système réseau), la transmission radio et les aspects de gestion réseau (canal radio, mobilité, sécurité et connexion). Bibliographie très complète.

[Sol94] Soley (Richard Mark). – Role of Object Technology in Distributed Systems, Appendix B, pp. 469–478. – In Khanna [Kha94a].

[Ude96] Udell (Jon). – Your Business Needs the Web. *BYTE*, vol. 21, n° 8, août 1996.
http://www.byte.com/art/9608/sec6/art1.htm.

[VH96] Van Hoecke (Marie-Pierre). – *Contribution à la Modélisation des Systèmes d'Information Communicationnels intégrant des Cartes à Micro-Processeur*, Thèse de Doctorat, USTL Université des Sciences et Technologies de Lille, janvier 1996.

[Weg90] Wegner (P.). – Concepts and Paradigms of Object-Oriented Programming. *ACM OOPS Messenger*, vol. 1, n° 1, août 1990.

CHAPITRE 2

Problématique des cartes à microprocesseur génériques

« *Ne pas confondre la carte avec le terrain.* »

Vieil adage populaire.

Résumé

Nous débutons par un état de l'art donnant un historique des cartes à microprocesseur, un tour d'horizon des normes qui les régissent et des standards du marché. Par la suite, nous détaillons les technologies impliquées à chaque étape du cycle de vie d'une carte et montrons comment une application carte est aujourd'hui développée. Dans cette partie, nous abordons les points suivants: microcontrôleur, développement de masques, outils de personnalisation, utilisation, sécurité et évolution des cartes. Enfin, nous exprimons le besoin de chargement de services à tout moment de l' utilisation d'une carte et soulignons l'impossiblité des cartes actuelles à répondre à ce problème. Nous terminons en pointant ce problème comme étant celui d'une nouvelle génération de système d'exploitation carte.

Sommaire

Tables

Figures

2.1 État de l'art des cartes à microprocesseur

*** Peu de références complètes sur les technologies des cartes à microprocesseur existent. Les livres datent [Bri88, GLR88, McC90, GS90] et/ou pêchent par une approche trop applicative [Che87, Svi87, Toe91]. La plus grande conférence carte [Car96] peut fournir des bonnes indications sur les derniers développements industriels et applicatifs, mais assez

peu dans le domaine de la recherche. Pour cela, se reporter de préférence aux actes des deux conférences CARDIS [CQ94, HPQ96]. Enfin, de bonnes sources pour une introduction aux cartes à microprocesseur sont le « Smart Card Tutorial » paru dans la revue *Smart Card News* [Eve94] et le numéro spécial « paiement électronique » de la revue *L'écho des recherches* du CNET[1] [CNE94]. Nous utilisons donc principalement ces dernières références pour réaliser ce paragraphe.

2.1.1 Historique

2.1.1.1 Naissance de la carte à puce

La « *carte à puce*† » est issue des concepts inventés par Moreno (Roland) en 1974 à propos d'une mémoire protégée personelle permettant de réaliser des transactions de paiement. Ces idées ont, dans un premier temps, été concrétisées en France par des expérimentations menées sous le patronage du GIE[2] « carte à mémoire » regroupant des banques et la DGT[3] (future France Telecom). Ces expérimentations ont abouti, en 1983, au lancement de la téléphonie publique par carte prépayée baptisée « télécarte » et, en 1992, à la généralisation de la carte à microprocesseur pour l'ensemble des 16 millions de cartes bancaires françaises.

La technologie développée repose sur un support plastique au format d'une carte de crédit contenant un composant capable de stocker de l'information (mémoire) et de réaliser des traitements (microprocesseur) sécurisés (cryptographie). Ce composant encarté s'interface avec un périphérique spécifique (lecteur de carte) par l'intermédiaire d'une pastille de contacts (*cf.* FIG. 2.1 page suivante).

2.1.1.2 Différents types de cartes

Plusieurs sortes de cartes existent, depuis les simples cartes à mémoire jusqu'aux cartes comportant leur propre microprocesseur :

carte à mémoire† Ces cartes servent de mémoire « porte-jetons ». Elles contiennent un code applicatif simple permettant de lire et écrire en mémoire sans contrôle d'accès. Certaines zones peuvent être protégées par grillage d'un fusible après programmation pour stocker les informations de l'émetteur. La réécriture étant impossible elles sont, par exemple, utilisées comme cartes prépayées pour le téléphone [Bau95] ou les places de stationnement.

carte à logique câblée† Ces cartes utilisent de la mémoire EPROM[4] ou EEPROM[5]. Les accès à la mémoire sont protégés par des circuits câblés spécifiques aux besoins de l'application. Elles sont, par exemple, utilisées comme cartes de contrôle d'accès ou comme cartes prépayées rechargeables.

1. Centre National d'Études des Télécommunications
2. Groupement d'Intérêt Économique
3. Direction Générale des Télécommunications
4. Electrically Programmable Read-Only Memory
5. Electrically Erasable and Programmable Read-Only Memory

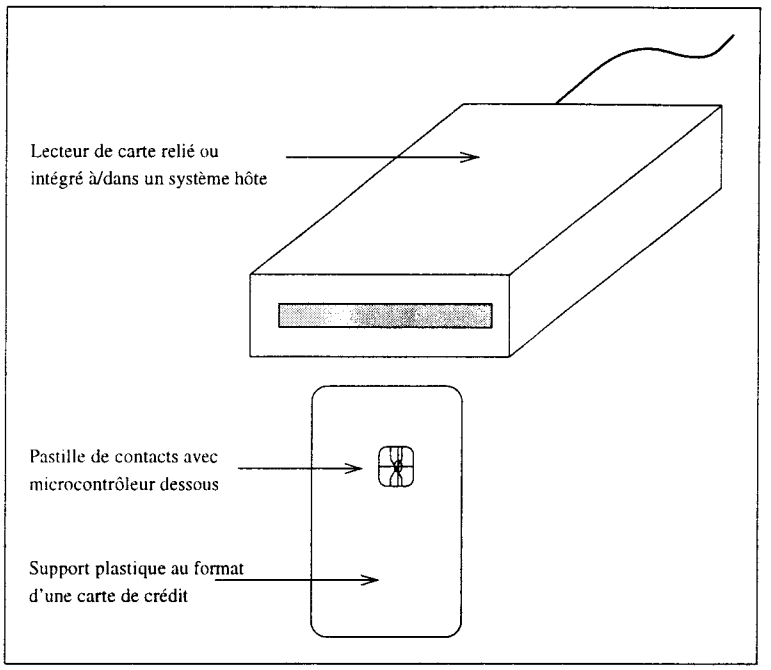

FIG. 2.1 - Technologie carte

carte à microprocesseur† Cartes contenant généralement un microprocesseur 8 bits avec son système d'exploitation gravé en mémoire ROM[6], une mémoire de travail RAM[7] et une mémoire non volatile. Le code du système d'exploitation gère la communication avec le monde extérieur, les accès en lecture, écriture et réécriture en mémoire et effectue des calculs algorithmiques permettant de contrôler en interne toutes les opérations. Un grand nombre d'applications est possible avec ces cartes.

carte sans contacts† Cartes communiquant avec un lecteur *via* un signal électromagnétique qui fournit l'alimentation de la carte et transmet commandes et données [TR94] [Eve94, partie 25, sept. 94]. Ces cartes disposent d'un circuit analogique en plus de leur circuit logique et de leurs mémoires. L'antenne est encapsulée dans le support plastique. Elles sont par exemple utilisées dans les systèmes de péage et de « billetique» (tickets pour les transports en commun) où les utilisateurs passent devant un lecteur sans avoir à connecter leur carte.

Ces différents types de cartes ont, depuis les premières expérimentations françaises, été utilisés dans une grande variété d'applications : systèmes de paiement, identification, contrôle d'accès logique et physique, dossier portable, *etc.* Celles qui nous intéressent dans ce mémoire sont les cartes à microprocesseur qui depuis 1983 sont normalisées au sein de l'ISO[8] et dont la popularité s'est développée par l'émergence de certains marchés et produits (*cf.* § 2.1.3 page 28).

6. Read-Only Memory
7. Random Access Memory
8. International Standards Organization

2.1.2 Normalisation de la carte à microprocesseur

Au niveau international, les cartes à microprocesseur sont normalisées par le sous-comité ISO/IEC JTC1 SC17 dont les normes ISO 7816-1 à 6 s'étendent des caractéristiques physiques des cartes jusqu'aux éléments de données partageables par différents secteurs industriels. Tous les chapitres de cette norme ne sont pas encore au stade final d'élaboration d'une norme (appelé IS[9]) mais devraient l'être rapidement [HP94].

2.1.2.1 Caractéristiques physiques des cartes

La norme ISO 7816 partie 1 [ISO87] définit la carte à microprocesseur comme « *une carte de type ID-1 (telle que spécifiée dans l'ISO 7810, l'ISO 7811 parties 1 à 5, l'ISO 7812 et l'ISO 7813) dans laquelle sont insérés un ou plusieurs circuits intégrés* ».

Les précédentes normes s'appliquent aux cartes d'identification (ID-1) et traitent de leurs caractéristiques physiques, des techniques d'enregistrement (position des caractères estampés, position des pistes magnétiques), du système de numérotation, de la procédure d'enregistrement pour les identificateurs d'émetteurs, et des cartes de transactions financières. La principale contrainte qu'elles imposent aux cartes à microprocesseur concerne leurs dimensions qui doivent être identiques à celles des cartes de transactions financières, à savoir : $85mm \times 54mm \times 0,76mm$.

De plus, la présente norme définit la dissipation thermique maximale du circuit intégré, la tenue aux rayonnements Ultra-Violets, X et électromagnétiques, ainsi qu'aux décharges électrostatiques, et que les données mécaniques suivantes : profil de surface des contacts, résistance au pliage et à la torsion.

2.1.2.2 Dimensions et positions des contacts

La norme ISO 7816 partie 2 [ISO88] définit la dimension des contacts ainsi que leur localisation en position médiane du côté embossé de la carte.

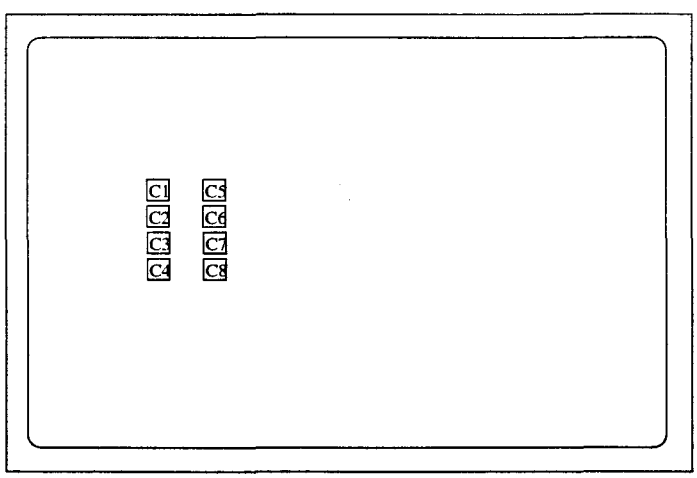

FIG. 2.2 - Contacts carte

9. International Standard

Le nombre des contacts est de huit avec les fonctions suivantes (*cf.* FIG. 2.2 page précédente) :

C1 Tension d'alimentation Vcc comprise entre 4,75V et 5,25V avec un courant de consommation maximal de 200mA. Cependant, de nouveaux composants acceptant une tension de 2,7V commencent à apparaître pour une utilisation des cartes dans des appareils portables.

C2 Signal de remise à zéro RST envoyé par le lecteur pour mettre la carte sous tension et débuter une session.

C3 Signal d'horloge CLK de 3,57 MHz ou 4,91MHz pour une transmission de 9 600 bits par seconde.

C5 Masse GND.

C6 Tension de programmation Vpp pour alimenter les composants disposant d'une mémoire EPROM qui nécessitent une tension de programmation de 12,5V, 15V ou 21V. Les mémoires de type EEPROM ne nécessitent pas d'utiliser ce contact car elles disposent d'une pompe de charge interne produisant la tension de programmation.

C7 Ligne d'entrée/sortie série unique fonctionnant en mode alterné ou mode semi-duplex, c'est-à-dire soit en émission soit en réception.

Les contacts C4 et C8 sont réservés pour une éventuelle définition ultérieure.

2.1.2.3 Protocoles de la carte

La norme ISO 7816 partie 3 [ISO89] [Eve94, parties 4–6, déc. 92-avril 93] spécifie l'interface entre une carte à microprocesseur et un lecteur de cartes. Les couches d'échange définissent un ensemble de caractéristiques permettant la communication entre cartes et lecteurs (*cf.* FIG. 2.3 page suivante).

La transmission d'un caractère La transmission se fait en série, en mode asynchrone à l'alternat. La transmission d'un caractère nécessite 11 bits (1 bit de départ, 8 bits de données, 1 bit de parité et 1 ou 2 bits d'arrêt). La détection des erreurs de transmission se fait au niveau du caractère grâce au bit de parité.

La réponse à la remise à zéro Lorsque la carte est mise sous tension par le lecteur, la carte doit renvoyer une réponse à la remise à zéro qui indique les conditions d'échange qui doivent suivre. La réponse est constituée de 2 caractères obligatoires, suivis d'au plus 31 caractères facultatifs. Les caractères d'interface précisent les paramètres de l'interface carte-lecteur à utiliser dans la suite de la communication. S'ils sont absents, des valeurs par défaut sont spécifiées. Mais la carte peut aussi proposer plusieurs protocoles et/ou vitesses de transmission, différents des valeurs par défaut. Des caractères dits «historiques» (16 au maximum) peuvent transiter dans la réponse pour permettre à l'émetteur de coder des informations applicatives.

Négociation des paramètres et du protocole Si une carte envoie dans sa réponse à la remise à zéro plusieurs protocoles et/ou paramètres de communication, le lecteur envoie à la carte une donnée définissant son choix. La carte renvoie alors en écho la même donnée si elle est d'accord.

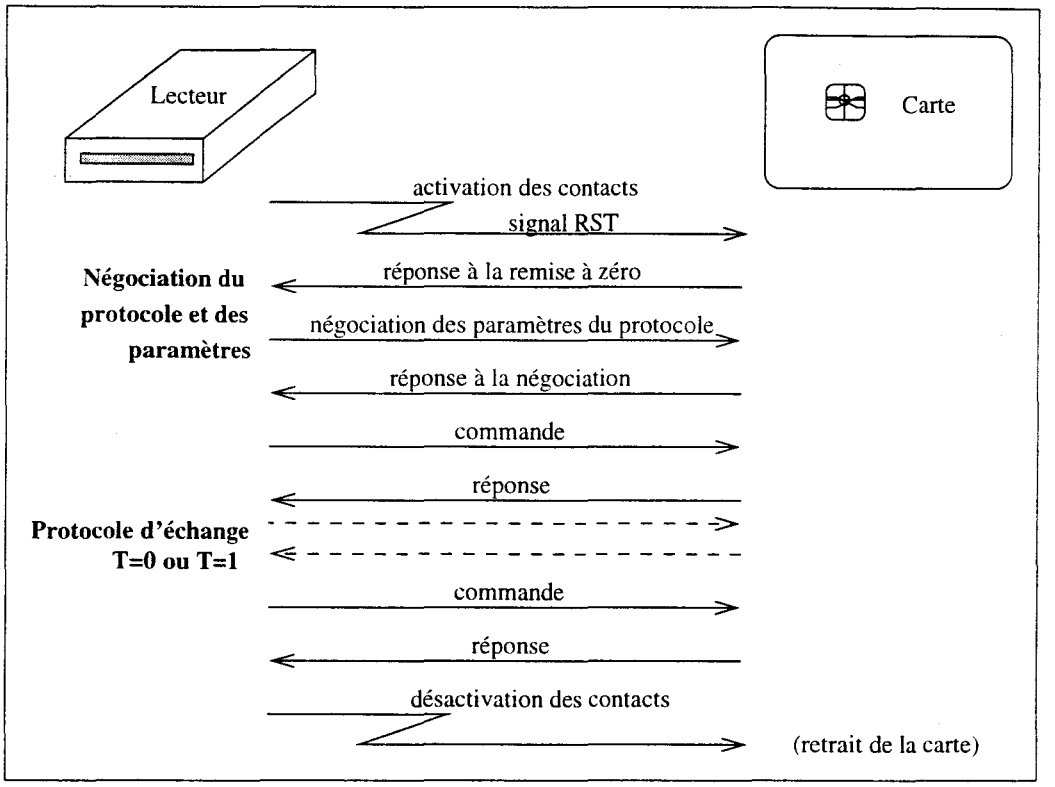

FIG. 2.3 - Interactions carte-lecteur

Protocole de transmission par caractère Ce protocole, appelé T=0, est majoritairement utilisé. Il définit, pour le lecteur, le moyen d'envoyer des données à la carte (données entrantes) ou de recevoir des données de la carte (données sortantes). La détection et la correction des erreurs se font au niveau du caractère.

Protocole de transmission par bloc Ce protocole, appelé T=1, est fondé sur l'échange de blocs entre carte et lecteur. Un CRC[10] à la fin des blocs de transmission permet de détecter les erreurs de transmission. Une numérotation des blocs permet de gérer leur séquencement. Ce protocole reste peu utilisé car il est plus difficile à implémenter dans une carte et plus lent que T=0 pour des données inférieures à 128 octets. Cependant il permet d'utiliser des circuits d'interface standard dans les lecteurs (type UART[11]) pour réémettre de façon « transparente» ce que leur envoie un ordinateur.

2.1.2.4 Jeu de commandes intersectorielles

La partie 4 de la norme ISO 7816 [ISO95] [Eve94, parties 7-8, mars-avr.93] traitant du jeu de commandes intersectorielles spécifie quatre caractéristiques internes aux systèmes d'exploitation des cartes : la structure des fichiers, la structure des messages, et les commandes de base.

10. Code de Redondance Cyclique
11. Universal Asynchronous Receiver/Transmitter

Structure des fichiers Il s'agit d'une structure hiérarchique de fichiers répertoires (appelés DF[12]). Un fichier répertoire peut comporter un ou plusieurs fichiers répertoires et/ou un ou plusieurs fichiers élémentaires (appelés EF[13]) comme descendants. La racine de l'arborescence est le fichier maître (appelé MF[14]).

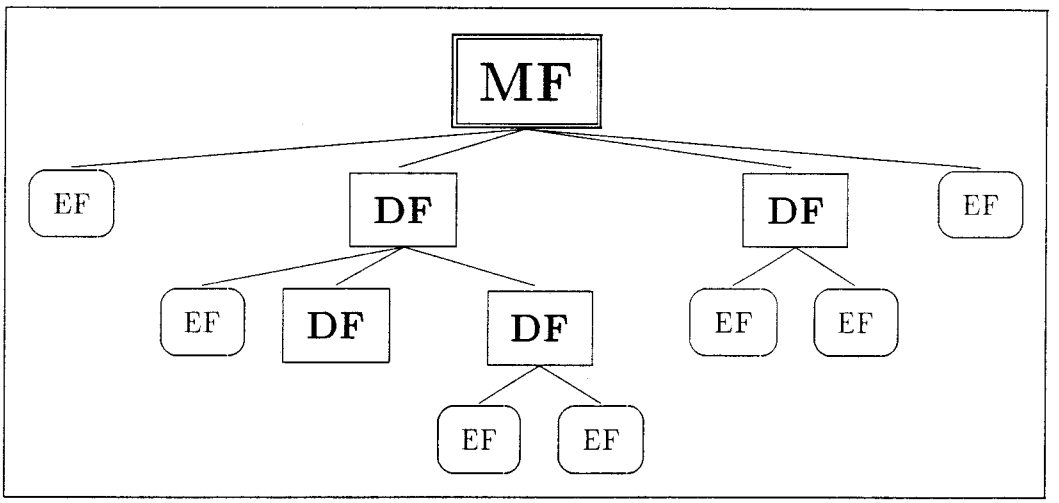

FIG. 2.4 - Organisation logique des fichiers ISO 7816-4

Un fichier élémentaire peut contenir des données organisées en enregistrements linéaires de longueur fixe, enregistrements linéaire de longueur variable, enregistrements cycliques de longueur fixe ou non organisées (structure transparente).

Structure des messages Cette partie de la norme introduit le concept d'unité de données pour le protocole applicatif (APDU[15]) permettant de décrire des commandes-réponses indépendamment du protocole de transport retenu (T=0 ou T=1). Une interaction élémentaire entre un système hôte et la carte est donc un envoi d'une APDU de commande à la carte, qui répond par une APDU de réponse. Il y a indépendance entre le protocole de transport (TPDU[16]) et le protocole d'application (APDU). La partie applicative transmet et reçoit des APDU au lecteur. Le lecteur transforme les APDU en TPDU suivant le protocole (T=0 ou T=1) utilisé par la carte.

La TABLE 2.1 page ci-contre donne la structure de l'APDU en fonction de la commande envoyée à la carte et de la réponse retournée par la carte. Les différents champs d'une APDU sont décrits dans la TABLE 2.2 page suivante.

Commandes de base Les commandes sont au nombre de 17 :

- 9 concernent la gestion des données et permettent de sélectionner un fichier, de lire les données contenues dans ce fichier et de les mettre à jour,

12. Dedicated File
13. Elementary File
14. Master File
15. Application Protocol Data Unit
16. Transport Protocol Data Unit

Cas	Commande	Réponse	APDU de commande	APDU de réponse
1	pas de données	pas de données	`CLA INS P1 P2`	`SW1 SW2`
2	données	pas de données	`CLA INS P1 P2 Lc Din`	`SW1 SW2`
3	pas de données	données	`CLA INS P1 P2 Le`	`Dout SW1 SW2`
4	données	données	`CLA INS P1 P2 Lc Din Le`	`Dout SW1 SW2`

TABLE 2.1 - Structure de l'APDU

Champ	Nom	Longueur (en octets)	Description
CLA	classe	1	classe d'instruction
INS	instruction	1	code d'instruction
P1	paramètre 1	1	paramètre 1 de la commande
P2	paramètre 2	1	paramètre 2 de la commande
Lc	longueur des données dans la commandes	variable < 4	nombre d'octets du champ de données
Din	données en entrée	variable = `Lc`	chaîne d'octets envoyés par la commande
Le	longueur des données en réponse	variable < 4	nombre maximum d'octets attendus en réponse
Dout	données en sortie	variable = `Le`	chaîne d'octets envoyés en réponse
SW1	mot d'état 1	1	classe d'erreur
SW2	mot d'état 2	1	numéro d'erreur

TABLE 2.2 - Champs de l'APDU

- 4 concernent la sécurité et permettent la vérification du code confidentiel, l'authentification d'une application, du monde extérieur et l'obtention d'un nombre aléatoire, et

- 3 commandes diverses assurent la gestion des canaux logiques dans la carte et la gestion des protocoles de transmission.

2.1.2.5 Identifiants d'application et procédures d'enregistrement

La partie 5 de la norme ISO 7816 [ISO94] [Eve94, parties 21–22, mai-juin 94] traite du système d'enregistrement d'identifiants d'applications carte. La norme définit le codage de ces identifiants ainsi que les mécanismes pouvant être employés pour sélectionner cette application dans la carte : sélection directe par l'identifiant d'application connu *a priori* par le système hôte, sélection indirecte par lecture du fichier d'index dans la carte contenant le moyen de sélectionner une application ou sélection implicite par la carte pour permettre une compatibilité avec les applications existantes.

2.1.2.6 Élements de données intersectorielles

La norme ISO 7816 partie 6 [ISO96] traitent des éléments de données intersectorielles. Le but est, pour des données d'utilisation fréquente (nom, identité bancaire, date d'expiration...), de disposer de codages et de moyens de localisation pour les rendre utilisables par toute application d'une carte multi-application.

2.1.3 Standards du marché

La carte à microprocesseur connaît un essor important avec le développement d'applications qui promeuvent son utilisation comme élément clé des systèmes [17]. Deux exemples pour illustrer notre propos : le système GSM [18] et les spécifications EMV [19].

2.1.3.1 GSM

Le système de radio téléphone GSM permet d'utiliser n'importe quel téléphone comme le sien quelque soit l'endroit en Europe. Ce système a pu se développer grâce à l'utilisation de cartes à microprocesseur. Chaque abonné reçoit une carte d'identification appelée SIM [20] qu'il peut introduire dans n'importe quel téléphone GSM. Après avoir tapé son code secret sur le poste téléphonique, l'abonné est identifié sur le réseau, reçoit ses appels sur ce poste et paye ses communications sur son compte personnel.

La carte permet une indépendance complète du réseau et des téléphones par rapport à l'abonnement souscrit par une personne. Elle offre les services de sécurité nécessaires à la facturation de l'abonné en vérifiant le code secret de l'abonné et en authentifiant son abonnement auprès du réseau qui peut alors acheminer les appels depuis et vers le poste utilisé en facturant le compte du porteur. De plus, les transmissions radio sont chiffrées à l'aide d'une clé de session calculée à la fois par la carte et par la station fixe de la cellule dans laquelle se trouve l'abonné. Elle permet aussi aux opérateurs d'ajouter des services personnalisés tels qu'une gestion de numéros abrégés ou des restrictions d'appels par liste de numéros, tranche horaire, *etc.*

Le standard GSM [ETS94] est aujourd'hui adopté par plus de 75 pays et ne cesse de se développer (aujourd'hui nous en sommes à la phase 2).

2.1.3.2 EMV

La carte bancaire française a depuis sa généralisation fait ses preuves. Trois principales institutions financières mondiales, Europay, Mastercard et Visa, ont depuis 1994 décidé de passer à cette technologie en créant des spécifications communes de systèmes de paiement basés sur l'utilisation de cartes à microprocessseur. Ces spécifications sont connues sous le nom d'EMV (initiales des trois promoteurs). Leur but est d'assurer une interopérabilité des cartes, des terminaux et autres équipements. Elles établissent aussi des normes de conception des matériels et des protocoles de communication et de sécurité pour les applications. Trois parties de ces spécifications (version 3.0) sont disponibles depuis juin 96 [EMV96a, EMV96b, EMV96c].

Ces spécifications définissent en fait un environnement ouvert de transactions financières par cartes. Elles créent un « programme commun » de carte que chaque organisme

17. La « télécarte» est la plus grande application de la carte à puce dans le monde (en 1993, les ventes de télécartes ont atteint les 100 millions sur un an). Cependant, il ne s'agit pas d'une carte à microprocesseur puisque sa technologie repose sur un composant dont la logique est câblée (*cf.* § 2.1.1.2 page 21 et [PD94]).
18. Global System for Mobile communication
19. Europay Mastercard Visa
20. Subscriber Identification Module

financier pourra compléter pour des utilisations de crédit, débit, porte-monnaie électronique (rechargeable ou non), application de fidélité, *etc.* Comme avec la norme GSM, possibilité est laissée aux opérateurs d'implémenter au dessus des recommendations des services personnalisés afin de retirer de leur exploitation de la valeur ajoutée.

Ces spécifications font l'objet de nombreuses attentions car elles sont produites par des organismes délivrant aujourd'hui près d'un milliard de cartes (à pistes magnétiques pour la plupart).

2.1.4 Carte base de données

Un dernier type de cartes à microprocesseur est à signaler car il propose une approche originale et innovante de la gestion des données au sein de la carte. Il s'agit d'une carte qui gère ses données en interne comme une base de données. La première carte basée sur ce schéma est la carte CQL[21]. Cette carte est le résultat de travaux de recherche et de prototypes développés à RD2P par Grimonprez (Georges) et Gordons (Édouard) [GG92, Gri92]. Depuis 1993, la carte CQL a été transférée et est devenue un produit de la société Gemplus [Gem93]. Depuis 1995, le groupe de travail 7 de l'ISO/IEC JTC1 SC17 (*cf.* § 2.1.2 page 23) travaille sur un projet de commandes avancées pour la carte qui intègre, comme pour CQL, des concepts de base de données.

La carte CQL est une carte à microprocesseur dont le système d'exploitation est un moteur de base de données et le jeu de commandes un sous-ensemble du langage standard SQL[22] [ISO92]. Le moteur situé dans la carte exécute en interne les requêtes CQL provenant de l'extérieur. La carte gère des utilisateurs qui, en fonction de leurs privilèges, peuvent exécuter des requêtes de manipulation de données (`SELECT`, `INSERT`, `UPDATE` ou `DELETE`) sur les tables, vues ou dictionnaires de la base de données carte.

Cette carte est typiquement destinée aux applications dites de « dossier portable » pour lesquelles la carte sert à maintenir le dossier personnel d'un individu (par exemple, un dossier étudiant, un dossier médical). Ce dossier est renseigné, mis à jour et édité par les différents systèmes auxquels est confronté son porteur. CQL apporte les avantages suivants à la gestion d'un tel dossier :

- structuration des données sous forme d'un schéma de base de données,
- simplification du partage des données et du contrôle d'accès par l'utilisation des mécanismes de vues et par l'établissement de privilèges[23], et
- souplesse de l'interrogation par le langage de requêtes sous-ensemble de SQL.

Ces concepts facilitent le développement d'application dossier portable mais restent encore marginaux dans le domaine des cartes à microprocesseur où domine l'approche gestion de fichiers.

21. Card Query Language
22. Structured Query Language
23. Une vue est un sous-ensemble logique et dynamique d'une table; attribuer un privilège de lecture sur une vue permet de restreindre l'accès à une table sans avoir, ni à la découper, ni à répliquer des données.

2.2 Technologie carte

On peut définir la carte à microprocesseur comme un micro-ordinateur[24] sécurisé embarqué sur sur un support plastique au format d'une carte de crédit. Le circuit intégré stocke de l'information et contrôle qui l'utilise et de quelle façon. Dans ce paragraphe nous détaillons les éléments techniques d'une telle définition.

2.2.1 Cycle de vie

Les caractéristiques techniques sont exposées en suivant les étapes dans lesquelles elles interviennent au cours du cycle de vie des cartes à microprocesseur (*cf.* FIG. 2.5) :

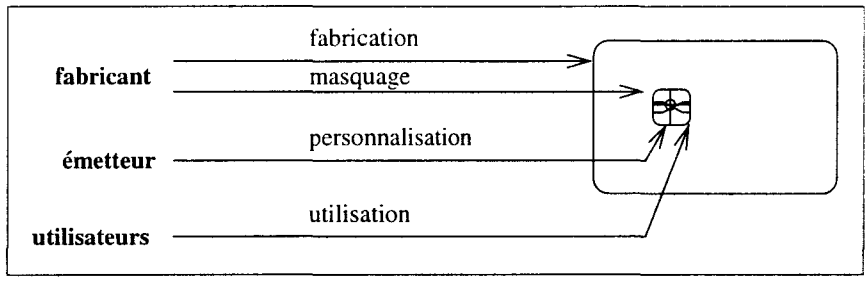

FIG. 2.5 - Cycle de vie d'une carte à microprocesseur

2.2.1.1 Fabrication

Le fabricant d'une carte à microprocesseur est responsable de l'« encartage » du microcontrôleur sur le support plastique en ABS[25] ou PVC[26]. Cette étape consiste à relier la pastille de contacts avec le microcontrôleur à l'aide de 8 fils d'or ou d'aluminium, puis à coller le module sur le corps imprimé de la carte.

Les microcontrôleurs utilisés sont du type MAM[27] défini par Ugon (Michel) chez BULL en 1978 [Ugo86]. Il s'agit d'un microcontrôleur ayant les caractéristiques suivantes (*cf.* FIG. 2.6 page suivante) :

microcalculateur il contient une capacité de traitement, un microprocesseur donc.

autoprogrammable il peut fonctionner de façon autonome, c'est-à-dire modifier directement sa mémoire de données (EEPROM ou EPROM).

monolithique il est contenu sur un substrat unique de silicium, ce qui impose des technologies compatibles pour l'ensemble de ses éléments.

encartable il doit pouvoir résister aux contraintes physiques définies dans la norme ISO 7816-1 , donc pouvoir fonctionner dans une large gamme de conditions et ne

24. Le terme « pico-ordinateur » serait peut-être plus juste...
25. Acrylonitrile Butadiene Styrene
26. Polyvinyl Chloride
27. Microcalculateur Autoprogrammable Monolithique

pas casser lors des torsions et flexions subies par la carte; ce qui limite sa surface à $25 - 30mm^2$ [28].

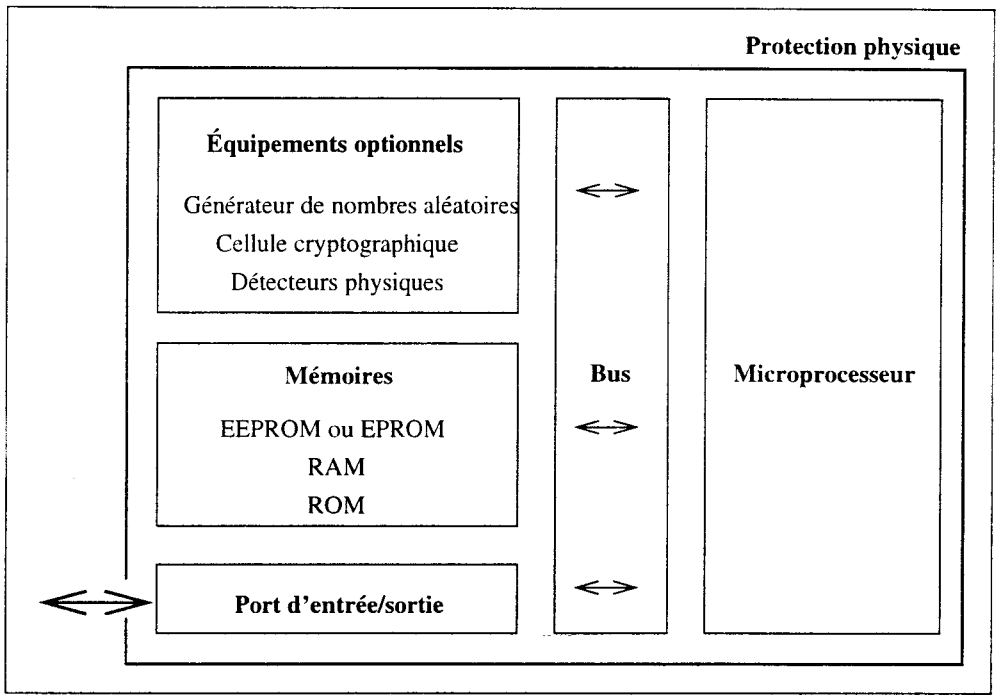

FIG. 2.6 - Microcalculateur Autoprogrammable Monolithique

La petite taille du microcontrôleur et son haut niveau de fiabilité imposent des contraintes fortes sur ces différents composants [29] :

Microprocesseur En général un processeur 8 bits du type Motorola 6805 (68HC05), Intel 8051 (80C51), SGS-Thomson 8048 ou Hitachi H8 (H8300). Ce microprocesseur est le point de passage obligé de toute opération dans le microcontrôleur, son fonctionnement est figé par un programme non modifiable gravé en ROM.

Mémoire ROM Mémoire indélébile contenant l'ensemble des fonctions du microprocesseur gravé en un masque lors de la fabrication (*cf.* § 2.2.1.2 page suivante). Sa capacité dans le microcontrôleur varie de 1,6 à 16 KO [30].

Mémoire RAM Mémoire volatile de type statique [31] servant de mémoire de travail au microprocesseur. Sa capacité varie de 36 à 512 octets.

Mémoire EPROM Premier type de mémoire non volatile utilisé dans les cartes, celui-ci tend à disparître car il nécessite une tension plus élevée. L'EPROM est effaçable par

28. L'épaisseur du microcontrôleur est de l'ordre de $0,28mm$.

29. Pour une liste non exhaustive des microcontrôleurs carte (et de leurs caractéristiques) usuellement utilisés aujourd'hui se reporter à [CN95].

30. Kilo-Octets

31. Les mémoires du type RAM dynamique ne sont pas utilisées dans les cartes car le gain en surface de silicium qu'elles apportent ne devient rentable qu'après 1 Mb. De plus, leur circuit de rafraichissement ne permettrait pas de mettre les cartes dans un mode d'attente pendant lequel la tension est baissée et l'horloge arrêtée.

rayons ultra-violet mais cette procédure est impossible avec la carte car le microcontrôleur est recouvert d'un plastique opaque, il s'agit donc d'une mémoire à écriture unique. Sa capacité varie de 1 à 8 KO.

Mémoire EEPROM Mémoire non volatile effaçable électriquement sans surcharge de courant. Elle sert à stocker les données applicatives mais peut aussi, dans certains cas, contenir des programmes (*cf.* § 2.2.1.3 page suivante). Sa capacité varie de 1 à 8 KO.

Port d'entrée/sortie Port de communication série asynchrone alterné pour recevoir des commandes du monde extérieur et retourner des réponses. Le taux d'échange est généralement de 9 600 BPS[32] mais peut parfois aller jusqu'à 19 200 ou 115 200 BPS.

Les autres éléments du microcontrôleur concernant la sécurité seront étudiés dans le § 2.2.2 page 35.

2.2.1.2 Masquage

La phase de masquage consiste au gravage en ROM par le fabricant de ce qu'il est courant d'appeler le « système d'exploitation » de la carte[33]. Il s'agit des routines de gestion de la couche matérielle, des fonctions implémentant le protocole de communication tel que défini par la partie 3 de la norme ISO 7816 , plus l'ensemble des requêtes disponibles pour l'extérieur. Les cartes fonctionnent comme des systèmes esclaves : le système d'exploitation est en fait un exécuteur de requêtes venant du monde extérieur. Les requêtes traditionnellement implémentées sont de quatre types [CN95] :

1. Organisation des données en mémoire et définition du schéma de sécurité (par exemple, commandes de création/destruction de zones logiques avec noms, tailles..., commandes de définition des protections des zones avec mots de passe, codes secrets...). Ces commandes ne sont généralement disponibles que pour l'émetteur pendant l'étape de personnalisation (*cf.* § suivant).
2. Vérification des droits d'un utilisateur ou d'un équipement en train d'accéder à des zones logiques (par exemple, commandes de présentation de code porteur ou d'authentification avec code secret...).
3. Manipulation de données (par exemple, commandes de lecture, écriture, mise à jour..., commandes de comptage de jetons..., commandes de recherche, comparaison...).
4. Autres commandes telles que des fonctions de génération de nombres aléatoires, de signature ou de déverrouillage de codes bloqués.

Selon la nature des requêtes implémentées quatre types de masques ressortent (*cf.* TABLE 2.3 page suivante).

32. Bits Par Seconde

33. Cette étape fait partie de la fabrication mais pour des raisons de clarté nous l'exposons après avoir décrit la couche matérielle.

Type de masque	Fonction
banalisé	implémentation stricte d'une norme (ISO 7816-4, GSM, EMV, *etc.*)
propriétaire	implémentation d'un système d'exploitation défini par le fabricant (CQL, carte TB100 de Philips, carte COS de Gemplus, *etc.*)
adapté	modification ou ajout de fonctions en ROM dans l'un des deux précédents types de masques pour une utilisation particulière
spécifique	masque sur mesure pour une application particulière

TABLE 2.3 - Types de masques cartes

Le développement d'un masque représente un investissement important (*cf.* FIG. 2.7) : les spécifications et le choix du microcontrôleur sont une étape difficile, les outils de génie logiciel restent assez rudimentaires, et la programmation se fait le plus souvent directement en assembleur sur un émulateur du microcontrôleur [Eve94, partie 11, juil. 93]. Néanmoins, certains fabricants proposent des « cross-compilers » du langage C pour leurs microcontrôleurs. Une approche innonvante a même été développée pour le masque de la carte CQL : un interpréteur réside en ROM, le système d'exploitation est écrit dans un langage proche du C (C-Card), compilé dans le langage interprété, ajouté en ROM puis exécuté par l'interpréteur dans la carte [GP91].

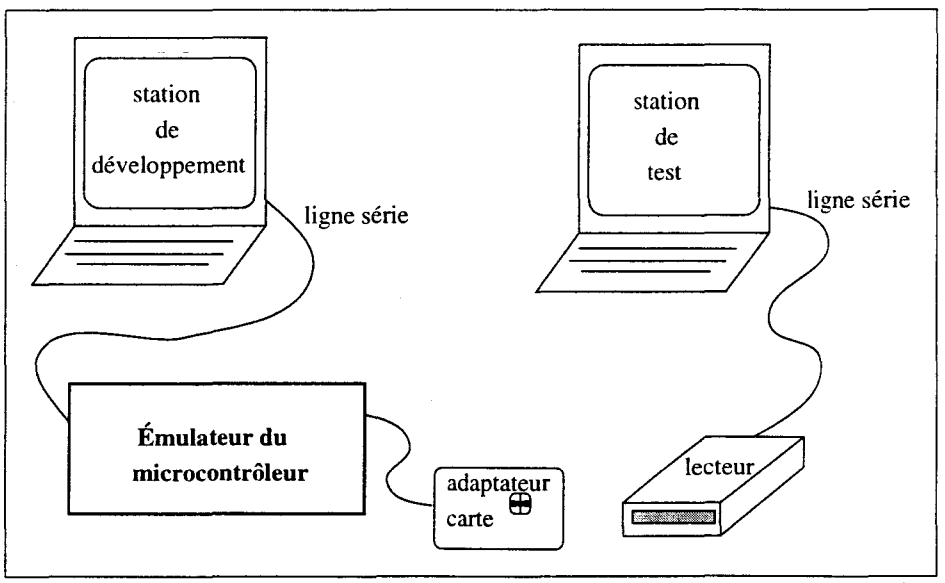

FIG. 2.7 - Banc de développement de masques carte

2.2.1.3 Personnalisation

Après le masquage du microcontrôleur et son encartage sur le support plastique, les cartes sont délivrées à un émetteur d'application. Celui-ci est le promoteur de la future utilisation de la carte. Il a pour tâches de :

- définir quels types de données seront manipulés par la carte, comment ils seront organisés dans la mémoire non volatile,
- définir les utilisateurs de la carte et les opérations qu'ils pourront réaliser sur les données,

- organiser un schéma de sécurité en accord avec les deux points précédents et le niveau de sécurité souhaité (mot de passe, simple ou double authentification, *etc.*).
- initialiser certaines données pour identifier l'application qu'il est en train de mettre en œuvre, et
- délivrer la carte en initialisant certaines données en fonction des caractéristiques du porteur.

Cette étape requiert classiquement l'utilisation du premier groupe de fonctions du système d'exploitation de la carte défini § 2.2.1.2 page 32. La personnalisation devient plus compliquée si la carte est prévue pour être utilisée dans plusieurs applications (plusieurs prestataires de services) qui se partagent des données. Dans ce cas, il sera nécessaire de faire appel à un émetteur principal qui aura autorité pour réaliser un schéma de données et de sécurité plus complexe en partageant des zones pour chacune des applications. Cependant, la conception logique de la carte reste toujours sous la responsabilité d'un unique émetteur.

À cette phase du cycle de vie, certaines cartes permettent à l'émetteur de rajouter du code exécutable chargé en EEPROM. La première carte à offrir cette fonctionnalité (appelée « filtre ») fut la carte COS[34] de Gemplus [GP91]. Cette approche est intéressante car elle offre un moyen de développer rapidement du code supplémentaire au dessus d'un système d'exploitation sans avoir à reprogrammer complétement un masque en ROM (*cf.* TABLE 2.3 page précédente, ligne 3). Par contre, cette fonctionnalité ne peut être mise en œuvre que pendant la phase de personnalisation : le filtre est en quelque sorte un « patch » d'adaptation ou de correction avant l'utilisation de la carte. De plus, les programmes résidants en EEPROM réduisent la place disponible pour les données.

2.2.1.4 Utilisation

Durant la phase d'utilisation la carte peut être utilisée par son porteur chez différents utilisateurs. Ces derniers sont équipés de terminaux (en-ligne ou déconnecté) qui sont programmés (ou reçoivent leurs instructions par réseau) pour mener des transactions avec la carte, c'est-à-dire qu'ils connaissent *a priori* les fonctions du système d'exploitation de la carte. S'il s'agit d'un jeu de commandes carte standardisé le développement de l'application hôte est relativement simple. Si le masque carte est du type adapté ou spécifique (*cf.* TABLE 2.3 page précédente), ou si des filtres ont été ajoutés il doit y avoir eu accord et transfert de connaissance entre l'émetteur et les utilisateurs. De plus, les utilisateurs doivent disposer d'informations secrètes (code, clé, *etc.*) leur permettant de s'authentifier comme des partenaires valides auprès de la carte. Ces données secrètes leur sont aussi fournies par l'émetteur.

Les transactions menées durant la phase d'utilisation utilisent les fonctions du masque carte des groupes 2 et 3, plus éventuellement celles du groupe 4 (*cf.* la liste § 2.2.1.2 page 32). Si la carte est du type multi-application la transaction débute par la sélection de la zone logique d'une application attribuée par l'émetteur principal lors de la personnalisation.

34. Card Operating System

La phase d'utilisation peut se terminer par une panne de la carte, le dépassement de la date d'expiration, l'arrêt de l'application, la perte ou le vol de la carte.

2.2.2 Sécurité

La carte à microprocesseur a surtout rencontré un succès important car elle est considérée comme un équipement sécurisé de haut niveau [GUQ91]. Dans ce paragraphe nous allons montrer les différents éléments qui concourrent à faire de la carte un matériel résistant aux attaques.

2.2.2.1 Sécurité physique

Les microcontrôleurs carte sont conçus pour résister à trois grandes classes d'attaques : la mise dans un environnement physique anormal du composant pour lui faire réaliser des opérations non prévues, l'espionnage des circuits pour en retirer de la connaissance et mieux pouvoir frauder, et l'essai de fonctions interdites [Tra95, chapitre 1].

Pour lutter contre la première classe d'attaques, les techniques employées sont :

- le recouvrement du composant par une couche de passivation diélectrique (isolant) qui empêche le passage de toute radiation (y compris celle d'un microscope électronique),
- l'utilisation de détecteurs physiques qui inhibent la fonction de sortie en cas de conditions anormales (détecteurs de lumière, de variations de température, tension et fréquence), et
- la présence de « cellules témoins » (composant Hitachi H8300) détectant l'effacement anormal de zones mémoire EEPROM.

La seconde classe d'attaques est contrée par la nature monolithique du microcontrôleur : le bus est enterré dans les couches de silicium empêchant ainsi toute possiblité de surveillance à faible coût ou d'espionnage par sonde entre les composants. De plus, sont utilisés des technique de courant aléatoire lors de la lecture de la ROM et le l'EEPROM, de brouillage des adresses ROM et EEPROM et, en général, de logique « sauvage » des circuits.

Enfin, des fusibles logiques et physiques sont grillés pour interdire l'utilisation des modes « test » lors de la remise des cartes aux émetteurs. Certains microcontrôleurs sont aussi équipés d'une « matrice de sécurité physique » supervisant les accès (lecture `L`, écriture `E` et exécution `X`) aux différentes mémoires selon la mémoire à partir de laquelle un programme s'exécute (*cf.* TABLE 2.4).

Accès à / Depuis	ROM	RAM	E(E)PROM
ROM	L,E,X	L,E,X	L,E,X
RAM	X	L,E,X	L,E,X
E(E)PROM	X	--	L,E,X

TABLE 2.4 - Matrice de sécurité physique des microcontrôleurs carte

2.2.2.2 Sécurité logique

Un première forme de sécurité logique est fournie par le verrouillage de chaque phase du cycle de vie de la carte [Eve94, partie 12, jan. 93]. En effet, chaque passage d'une étape à une autre du cycle de vie est irréversiblement bloqué par présentation successive de clés. La présentation d'une clé peut, par exemple, « lever un bit »[35] en mémoire EPROM (mémoire indélébile); la valeur de ce bit conditionnant l'utilisation de certaines fonctions du système d'exploitation. Ainsi des opérations possibles en phase de masquage deviennent inacessibles en phase de personnalisation après présentation de la clé du fabricant. De même lors du passage à la phase d'utilisation pour les fonctions utilisées par l'émetteur.

Lors de la personnalisation, l'émetteur est responsable de la mise en place de l'organisation des données dans la mémoire (découpage de la mémoire en zones, fichiers, tables, *etc.*). Sur ces structures de données, l'émetteur applique une *politique de sécurité*† dont le but principal est de définir la politique de contrôle d'accès des utilisateurs sur les structures de données. En ce sens, le système d'exploitation de la carte agit comme un *moniteur de référence*† [DOD85] qui assure l'authentification des utilisateurs, puis la gestion et la vérification de leurs droits d'accès[36]. Lorsqu'un utilisateur (authentifié comme tel) envoie une requête à la carte, le système d'exploitation vérifie si les droits d'accès existent, et, en conséquence, autorise ou interdit l'accès (*cf.* FIG. 2.8).

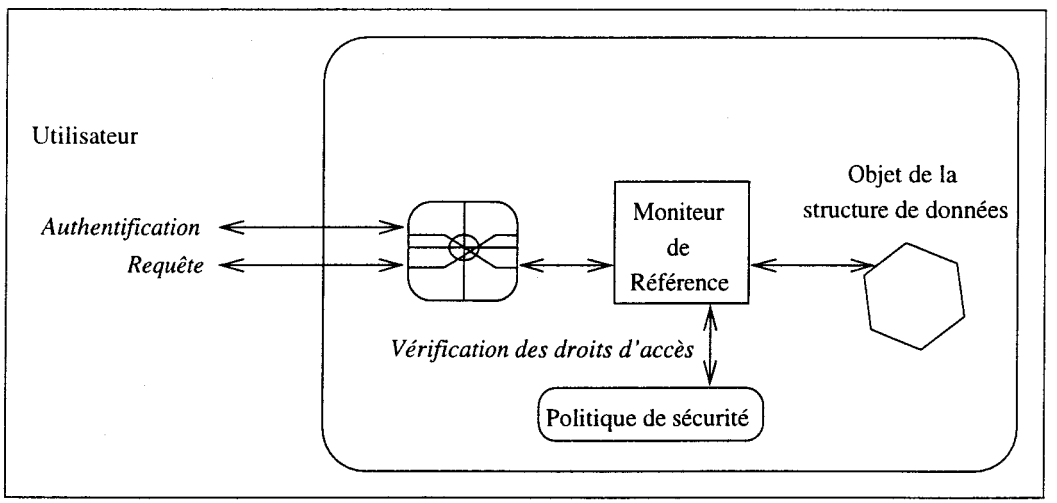

FIG. 2.8 - Moniteur de référence carte

Le moniteur de référence carte est au cœur du système de sécurité[37] en servant d'intermédiaire obligatoire entre tout utilisateur et tout objet. Il possède donc les attributs suivants :

Incontournable Le microprocesseur est le point de passage physique (*cf.* § précédent) obligé pour toute communication. Son système d'exploitation agissant comme un moniteur de référence est toujours invoqué lors de l'accès d'un utilisateur vers un objet de la structure de données.

35. Expression utilisée pour signaler qu'on fait passer un bit de l'état 0 à l'état 1.
36. Ces fonctions sont dites « obligatoires » ou « par mandats » car les droits sont régis uniquement par des règles établies une fois pour toutes (par l'émetteur).
37. Il est aussi appelé « noyau de sécurité ».

Inviolable Aucune attaque du moniteur de référence ne doit pouvoir modifier ses fonctionnalités.

Prouvé correct Le moniteur de référence doit obligatoirement être vérifié et prouvé correct vis-à-vis de ses spécifications.

2.2.2.3 Sécurité cryptographique

La mise en œuvre des procédures d'authentification dans les cartes est encore renforcée par l'utilisation de techniques cryptographiques. Ces techniques permettent, par exemple, de réaliser un protocole de double authentification entre l'utilisateur et la carte sans qu'aucune information secrète ne transite en clair sur la ligne et qu'ainsi le rejeu de transactions (par un espion qui aurait lu les messages échangés) soit impossible. Ces techniques utilisent dans la carte des fonctions de génération de nombres aléatoires, de hachage, de chiffrement symétrique ou asymétrique. Ces fonctions permettent de plus à la carte de réaliser des opérations de signature électronique, certification et chiffrement [Tra95, chapitre 1].

Les fonctions de chiffrement asymétriques sont considérées comme plus adaptées aux applications carte car elles facilitent le système de distribution de clés. Mais, aujourd'hui, la plupart des cartes du marché utilisent des fonctions de chiffrement symétriques (principalement le DES[38]) car elles sont plus faciles à implémenter et requièrent moins de ressources système.

2.2.3 Évolutions

Nous terminons ce paragraphe en donnant une liste non exhaustive des évolutions en cours des technologies utilisées dans les cartes à microprocesseur. La réduction de la géométrie des microcontrôleurs (en dessous du micron) [Mue95] et l'addition de nouvelles technologies devraient permettre aux composants carte d'offir :

Plus de mémoire De nouveaux types de mémoire non volatile permettent, soit d'augmenter la densité de cellules pour une même surface de silicium (mémoire Flash EEPROM [Pat90]), soit d'améliorer les temps de lecture/écriture (mémoire ferroélectrique RAM [Eve94, partie 26, oct. 94]).

Plus de traitement Des microprocesseurs basés sur une architecture RISC 32 bits sont à l'étude pour être intégrés dans les microcontrôleurs [Pey94].

Plus de sécurité Des co-processeurs cryptographiques spécialisés dans le traitement de fonctions de chiffrement asymétriques seront de plus en plus intégrés dans les microcontrôleurs. Des fonctions d'identification biométrique permettront aussi d'augmenter la sécurité des applications carte [Ale95].

38. Data Encryption Standard

2.3 Cartes génériques

Parmi les évolutions technologiques précédemment mentionnées il manque celles concernant le besoin de **plus de fonctionnalités** qui feront l'objet de ce paragraphe. De plus en plus de prestataires sont intéressés par l'utilisation des cartes à microprocesseur pour offrir des services personnalisés à leurs clients. Cependant, il sont freinés par la non-flexibilité des cartes actuelles.

2.3.1 Limites des cartes actuelles

Reprenons la description d'une application carte actuelle :

1. Le système d'exploitation de la carte est gravé en ROM par le fabricant. Il implémente les fonctions de communication ISO 7816-3 permettant l'interopérabilité des cartes et des lecteurs. Il implémente aussi un ensemble de commandes applicatives répondant soit à une norme (ISO 7816-4, GSM, EMV), soit à des spécifications issues d'un émetteur (par exemple, la carte santé allemande).

2. L'émetteur (qui peut agir au nom de plusieurs prestataires de services) initialise les structures de données en mémoire non volatile (définition de zones logiques, fichiers, tables, *etc.*) et définit la politique de sécurité associée à l'application. Il peut aussi, dans certains cas, ajouter des fonctions applicatives (filtres) en mémoire non-volatile. Enfin, il délivre la carte à un porteur avec un numéro d'identification personnel que ce dernier utilisera pour initialiser les transactions avec les utilisateurs.

3. Les utilisateurs réalisent des transactions avec la carte en utilisant les commandes du système d'exploitation (plus éventuels filtres) leur permettant uniquement de s'authentifier et d'agir sur le contenu des structures de données (lecture, écriture, comptage, *etc.*) en accord avec la politique de sécurité définie par l'émetteur. Si la carte est émise par plusieurs prestataires de services, l'utilisateur pourra disposer des fonctions définies par la norme ISO 7816-5 pour sélectionner une application carte.

Ce schéma montre que le fonctionnement de la carte ne peut en aucun cas sortir des limites définies par l'émetteur. Celui-ci (avec le fabricant pour le masquage de la ROM) définit entièrement et irrévocablement l'usage qu'il sera fait de la carte. Il est impossible, après émission de la carte, d'ajouter de nouvelles fonctionnalités – même pour l'application chargée dans la carte (les fonctions sont masquées, les structures de données figées et le schéma de sécurité statique) – et, *a fortiori*, de charger de nouvelles applications.

Une carte à microprocesseur mise à la disposition d'un porteur est un outil électronique dont les fonctionnalités sont prédéfinies et immuables[39]. Elle reste un outil fermé à cause de la forte intrication entre le matériel, les fonctions de système d'exploitation (au sens habituel du terme, à savoir couche de gestion des ressources matérielles et d'allocation des ressources pour les applications) et les fonctions applicatives. Cette situation aboutit

39. Cette situation est semblable à celle de l'informatique grand public avant la « revolution » du micro-ordinateur : les machines proposées était dédiée à un type particulier d'application (par exemple, traitement de texte, calculateur ou console de jeux) et ne pouvaient servir à rien d'autre.

au fait qu'il est aujourd'hui délicat de se lancer dans la technologie carte pour un prestataire voulant proposer de nouveaux services innovants s'accomondant mal des systèmes existants. Il lui faut, en effet, faire ou faire faire :

- une étude pour choisir le microcontrôleur à utiliser,
- des spécifications du masque de la carte, et
- l'implémentation de ce masque.

Il s'agit là d'un lourd investissement en argent et compétences que peu d'entreprises sont capables de fournir surtout si un risque existe envers les bénéfices attendus. Ce qui, pour une part, fait la richesse des « encarteurs » et explique sans doute pourquoi, jusquà aujourd'hui, l'usage de la carte a pu seulement être promu par des administrations, des groupes professionnels importants ou des grandes sociétés : carte bancaire, carte santé, norme GSM , EMV, *etc.*

2.3.2 Potentiel pour des cartes ouvertes

Les cartes sont pourtant de plus en plus intégrées dans les stratégies concernant les systèmes à grande échelle (*cf.* Chapitre 1 page 3), notamment dans ce qu'il est courant d'appeler aujourd'hui le « commerce électronique » [Gat96]. Les évolutions technologiques permettant des cartes plus puissantes, plus sûres, avec plus de mémoire font que celles-ci sont de plus en plus perçues comme des composantes personnelles et sécurisées des ordinateurs et des réseaux avec lesquelles il deviendrait plus simple de mettre en œuvre des services tels que : le paiement de marchandises, le stockage de points de fidélité chez les commerçants, la location de places de stationnement, le stockage des titres de transport, le stockage de données médicales ou encore le paiement sur Internet.

Pour cela, il est nécessaire de favoriser le développement de services carte par n'importe quelle entreprise désireuse d'investir ce marché. De système dédié, la carte est « à l'âge » – et elle est attendue sur ce terrain – de passer à système ouvert capable de supporter de multiples applications pour réaliser des services adaptés aux besoins du porteur.

Cette ouverture permettrait d'élargir le marché de la carte à de nombreuses entreprises qui pourraient rapidement développer et proposer des services carte à leurs clients. Une meilleure et plus rapide adaptation aux besoins du marché serait aussi possible en réduisant le temps de développement d'un masque carte à celui équivalent à l'écriture d'une application pour micro-ordinateurs. L'offre d'outils permettant de favoriser les transactions électroniques augmente souvent le nombre effectif de ces transactions et permet, par conséquent, de récupérer les sommes investies et de générer des profits[40].

Le porteur, d'un rôle passif, deviendrait le propriétaire de sa carte et pourrait la charger avec les services dont il a besoin, au moment et à l'endroit nécessaire et durant le temps qu'il désire. Il ne disposerait plus que d'une seule carte dans son portefeuille, plutôt que d'une multitude de cartes comportant des services indépendants les uns des autres. À long terme, il est aussi possible d'imaginer que les multiples services résidants dans une

40. L'expérience de la télécarte en France montre que depuis son introduction la longueur moyenne des communications téléphoniques dans les cabines publiques a augmenté de 50 % !

seule carte pourraient de quelques manières coopérer pour offrir des services encore plus intégrés.

2.3.3 Système d'exploitation carte générique

Pour les encarteurs, le défi consiste donc à fournir un système d'exploitation indépendant de la plate-forme matérielle et non prédéterminé par les applications qu'il aura à supporter [PV95]. Il s'agit de mettre à disposition des prestataires de services un système d'exploitation multi plates-formes offrant une API de programmation efficace et évolutive (possibilité pour l'encarteur de proposer de nouvelles versions de son système d'exploitation garantissant une compatibilité avec les applications existantes).

C'est à ce problème que nous appelons « carte générique » qu'une partie de ce mémoire apporte une contribution. Dans ce cadre, nous avons :

- addressé et mis en exergue la problématique des cartes génériques (présent chapitre),
- défini un modèle de système à cartes génériques basé sur un cycle de vie ouvert et sur l'utilisation d'un interpréteur sécurisé dans la carte (*cf.* Chapitre 4 page 69),
- étudié une architecture de sécurité permettant de mettre en œuvre des applications carte dans les systèmes ouverts (*cf.* Chapitre 5 page 87),
- proposé une architecture de machine virtuelle et développé un interpréteur pour code applicatif chargé dans la carte (*cf.* Chapitre 6 page 105), et
- implémenté un prototype de système d'exploitation pour carte générique (*cf.* Chapitre A page 145).

L'autre partie du mémoire concerne l'interopérabilité des cartes et des systèmes informatiques. Sa problématique est exposée dans le chapitre suivant.

Bibliographie sur les cartes à microprocesseur

[Ale95] Alexandre (Thomas). – *Manipulation de données multimédia dans la carte à microprocesseur : application à l'identification biométrique et comportementale*, Thèse de Doctorat, USTL Université des Sciences et Technologies de Lille, février 1995.

[Bau95] Bausson (Stéphane). – What you need to know about electronics telecards. – septembre 1995.
http://www.funet.fi/pub/doc/phonecards/chips/How_chips_work.

[Bri88] Bright (Roy). – *Smart Cards: Principles, Practice, Applications*, Halsted Press, 1988.

[Car91] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1991 conference proceedings, U.S.A.*, 1991.

[Car94] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1994 conference proceedings, April 10 to 13, 1994, Arlington, Virginia, U.S.A.*, avril 1994.

[Car95] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1995 conference proceedings, April 10 to 13, 1995, Washington, DC, U.S.A.*, avril 1995.

[Car96] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1996 conference proceedings, May 13 to 16, 1996, Atlanta, Georgia, U.S.A.*, mai 1996.

[Che87] Chemineau (Laurent). – *L'argent invisible. l'ère des flux électroniques*, Editions Autrement, 1987.

[CN95] Cordonnier (Vincent) et Naccache (David). – *Smart-Cards in Authentication Architectures: Present and Future Applications and Techniques.* – Rapport technique, RD2P Recherche et Développement Dossier Portable, 1995.

[CNE94] CNET Centre National d'Études des Télécommunications. – *L'écho des recherches num. 158, Numéro spécial « paiement électronique »*, France Telecom, 1994.

[CQ94] Cordonnier (Vincent) et Quisquater (Jean-Jacques). – *Proceedings of the first smart card research and advanced application conference, October 24–26, 1994, Lille, France*, RD2P Recherche et Développement Dossier Portable, octobre 1994.

[DOD85] DOD U.S. Department Of Defense. – *Trusted Computer Systems Evaluation Criteria* – novembre 1985.
http://www.disa.mil/MLS/info/orange/index.html.

[EMV96a] EMV Europay, Mastercard, Visa. – *EMV'96 Integrated Circuit Card Specification for Payment Systems* – juin 1996.
http://www.visa.com/sf/chip/CARDSPEC.PDF.

[EMV96b] EMV Europay, Mastercard, Visa. – *EMV'96 Integrated Circuit Card Terminal Specification for Payment Systems* – juin 1996.
http://www.visa.com/sf/chip/TERMSPEC.PDF.

[EMV96c] EMV Europay, Mastercard, Visa. – *EMV'96 Integrated Circuit Card Application Specification for Payment Systems* – juin 1996.
http://www.visa.com/sf/chip/APPLSPEC.PDF.

[ETS94] ETSI European Telecommunication Standard Institute. – *Digital cellular telecommunications system (phase 2); specification of the subscriber identity module – Mobile Equipment (SIM - ME) interface (GSM 11.11)* – novembre 1994, final draft prets 300 608.

[Eve94] Everett (David B.). – Smart card tutorial – Part 1 to Part 26. *Smart Card News*, Sept. 92 – October 1994.

[Gat96] Gatchell (Oliver). – *The Smart Card Dilemma.* – Rapport technique, Racom Systems Inc., 1996.
http://www.rmi.net/racom/page5.htm#The Smart Card Dilemma.

[Gem93] Gemplus. – *CQL Card Reference Manual* – avril 1993, version préliminaire.

[GG92] Gordons (Édouard) et Grimonprez (Georges). – A card as element of a distributed database. – In Paradinas et al. [PVSW92].

[GLR88] Guez (Fradji), Lauret (Anette) et Robert (Claude). – *Les cartes à microcircuit*, Masson, 1988.

[GP91] Grimonprez (Georges) et Paradinas (Pierre). – A new approach in code development: C-Card and Cossack. – In CardTech/SecurTech, Inc. [Car91].

[Gri92] Grimonprez (Georges). – *Etude et réalisation d'une carte à microprocesseur intégrée aux SGBDs*, Mémoire d'habilitation à diriger des recherches, USTL Université des Sciences et Technologies de Lille, 1992.

[GS90] Ganne (Roger) et Salomoni (Brigitte). – *La carte à mémoire*, Eyrolles, 1990.

[GUQ91] Guillou (Louis C.), Ugon (Michel) et Quisquater (Jean-Jacques). – The Smart Card: A Standardized Security Device Dedicated to Public Cryptography, *In : Contemporary Cryptology*, éd. par Simmons (G. J.), – pp. 561–614, IEEE Computer Society, 1991.

[HP94] Hiolle (Philippe) et Paillès (Jean-Claude). – Normalisation des cartes dans le secteur des télécommunications, pp. 49–58. – In *L'écho des recherches num. 158, Numéro spécial « paiement électronique »* [CNE94].

[HPQ96] Hartel (Pieter H.), Paradinas (Pierre) et Quisquater (Jean-Jacques). – *Proceedings of the 2nd International Conference CARDIS 1996 CWI, Amsterdam, The Netherlands, September 16-18, 1996*, Stichting Mathematisch Centrum, septembre 1996.

[ISO87] ISO International Standards Organization. – *ISO 7816-1:1987 Cartes d'identification - Cartes à circuit(s) intégré(s) à contacts - Partie 1: Caractéristiques physiques* - 1987.

[ISO88] ISO International Standards Organization. – *ISO 7816-2:1988 Cartes d'identification - Cartes à circuit(s) intégré(s) à contacts - Partie 2: Dimensions et emplacements des contacts* - 1988.

[ISO89] ISO International Standards Organization. – *ISO/IEC 7816-3:1989 Cartes d'identification - Cartes à circuit(s) intégré(s) à contacts - Partie 3: Signaux électroniques et protocoles de transmission* - 1989.

[ISO92] ISO International Standards Organization. – *International Standard ISO/IEC 9075 : Information Technology — Database — languages — SQL* - novembre 1992.

[ISO94] ISO International Standards Organization. – *ISO/IEC 7816-5:1994 Cartes d'identification - Cartes à circuit(s) intégré(s) à contacts - Partie 5: Système de numérotation et procédure d'enregistrement d'identificateurs d'applications* - 1994.

[ISO95] ISO International Standards Organization. – *ISO/IEC 7816-4:1995 Cartes d'identification - Cartes à circuit(s) intégré(s) à contacts - Partie 4: Commandes intersectorielles pour les échanges* - 1995.

[ISO96] ISO International Standards Organization. – *ISO/IEC DIS 7816-6: Cartes d'identification - Cartes à circuit(s) intégré(s) à contacts - Partie 6: Éléments de données intersectoriels* - 1996.
DIS distribué en version anglaise seulement..

[McC90] McCrindle (John). – *Smart cards*, IFS Publications/Springer-Verlag, 1990.

[Mue95] Muenchau (Daniel S.). – The Next Generation of Smart Card Microcontrollers – a Silicon Perspective, pp. 91–97. – In CardTech/SecurTech, Inc. [Car95].

[Pat90] Paterson (Mike). – "Memories are made of this..." ...a look at memory considerations for Smart Card applications. *Semiconductor engineering bulletin*, 1990.

[PD94] Paillès (Jean-Claude) et Depret (E.). – La télécarte de deuxième génération, pp. 59–61. – In *L'écho des recherches num. 158, Numéro spécial « paiement électronique »* [CNE94].

[Pey94] Peyret (Patrice). – RISC-based, next-generation smart card microcontroller chips, pp. 9–36. – In CardTech/SecurTech, Inc. [Car94].

[PV95] Paradinas (Pierre) et Vandewalle (Jean-Jacques). – New Directions for Integrated Circuit Card Operating System. *Operating System Review*, vol. 29, n° 1, janvier 1995, pp. 56–61.

[PVSW92] Paradinas (Pierre), Verrijn-Stuart (Alex) et White (George). – *IFIP WG8.4 Workshop and Tutorial, The portable office: Microprocessor cards as elements of distributed offices, 9-10 June, 1992, The University of Ottawa, Canada*, The University of Ottawa, juin 1992.

[Svi87] Svigals (Jerome). – *Smart cards, the new bank cards*, Macmillan publishing company, 1987.

[Toe91] Toernig (Jean-Pierre). – *Les systèmes électroniques de paiement*, Eyrolles, 1991.

[TR94] Thorigné (Yves) et Reitter (Renaud). – Nouvelle technologie de la carte à mémoire : la carte sans contact, pp. 43–48. – In *L'écho des recherches num. 158, Numéro spécial « paiement électronique »* [CNE94].

[Tra95] Trane (Patrick). – *Conception et réalisation d'un système de contrôle d'accès pour la carte à micro-processeur*, Thèse de Doctorat, USTL Université des Sciences et Technologies de Lille, septembre 1995.

[Ugo86] Ugon (Michel). – Un microcalculateur autoprogrammable au cœur de la carte CP 8. *Minis et Micros*, n° 153, 1986, pp. 33–36.

CHAPITRE 3

Problématique de l'interopérabilité des cartes et des systèmes informatiques

« L'échange et la communication sont des figures positives qui jouent à l'intérieur de systèmes complexes de restrictions; et ils ne sauraient sans doute fonctionner indépendamment de ceux-ci. »

Michel Foucault. *L'ordre du discours*, Leçon inaugurale au Collège de France prononcée le 2 décembre 1970, Gallimard, Paris. France, 1971.

Résumé

Ce chapitre traite la problématique de l'interopérabilité des cartes dans les systèmes informatiques. Nous débutons par un exposé des techniques qui favorisent une telle interopérabilité et montrons que les tentatives d'intégration des cartes menées par le passé sont encore insuffisantes. Deux exemples d'intégration de cartes – le premier dans le World Wide Web, le second dans les Systèmes de Gestion de Bases de Données – nous permettront d'analyser les avantages et inconvénients de chacun d'eux pour aboutir à une définition de couche d'interopérabilité basée sur une modélisation orientée objet des services cartes.

Sommaire

Tables

Figures

3.1 Mise en œuvre des cartes à microprocesseur dans les systèmes informatiques

3.1.1 Développement d'une application cliente d'une carte

Les normes des cartes à microprocesseur présentées dans le chapitre précédent définissent une approche en couches [1] pour intégrer les cartes dans les systèmes informatiques (*cf.* FIG. 3.1).

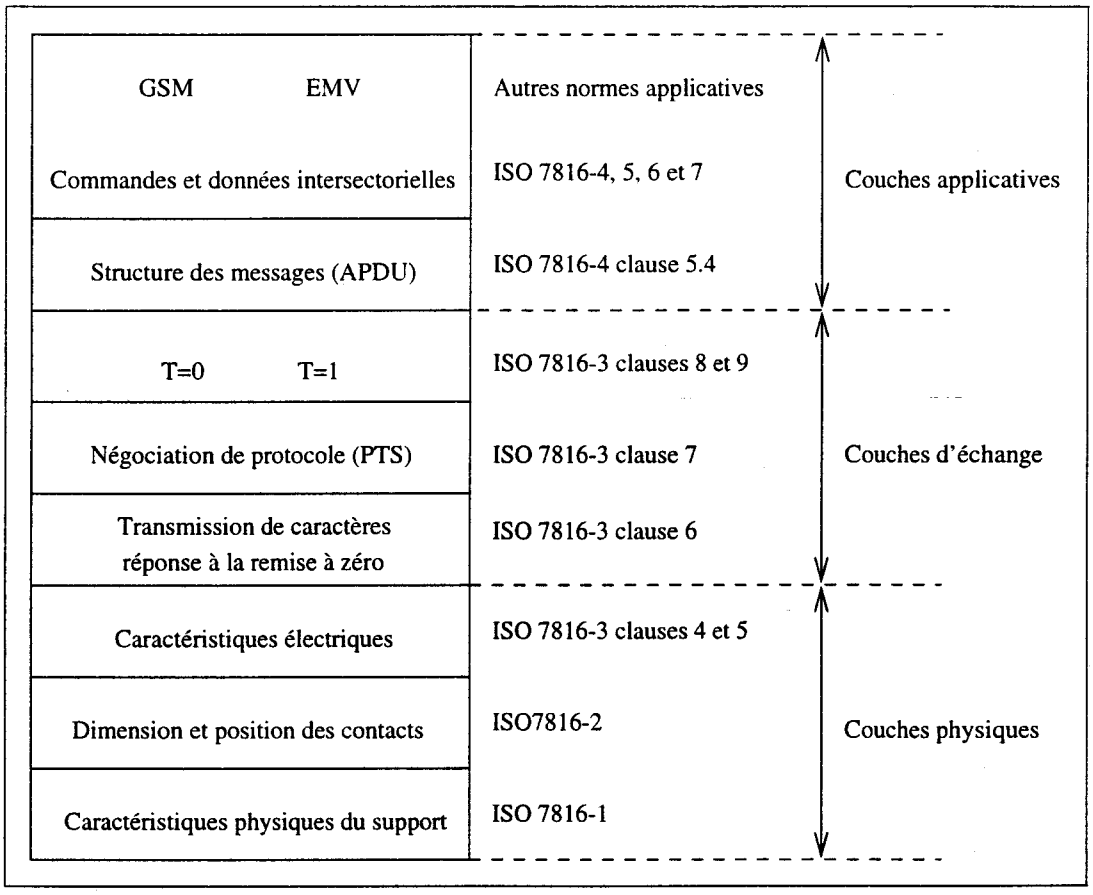

FIG. 3.1 - Couches d'interface des cartes dans les systèmes informatiques

Les couches physiques et d'échange aujourd'hui complètement normalisées au niveau international règlent les problèmes d'interopérabilité entre les cartes et les lecteurs : une carte peut être conçue pour interagir avec n'importe quel lecteur, un lecteur peut être conçu pour pouvoir interagir avec n'importe quelle carte. Elles permettent de :

1. connaître la dimension des cartes, la dimension et la position des contacts,

2. savoir quels courants électriques appliquer sur les contacts,

1. Cette approche en couche est difficilement comparable avec celle du modèle OSI pour l'interconnexion des systèmes ouverts qui elle s'intéresse à l'échange d'information entre paires de même niveau dans une communication réseau.

3. lire la réponse à la remise à zéro de la carte,
4. transmettre des caractères à la carte,
5. négocier un protocole, et
6. échanger des messages/réponses entre le lecteur et la carte avec le protocole de transport choisi lors de la phase de négociation.

Cependant ces interactions s'arrêtent au niveau du protocole de communication entre le lecteur et la carte[2]. Les couches applicatives définissant des jeux de commandes carte bien qu'abusivement appelés «systèmes d'exploitation» sont dépendantes des applications chargées dans la carte. Par exemple, les jeux de commandes externes d'une carte GSM et d'une carte bancaire française sont totalement différents. Pour développer une application utilisant des cartes il est donc nécessaire dès la conception de connaître la carte utilisée, c'est-à-dire son jeu de commandes et souvent aussi l'organisation interne de ses structures de données (dans quels fichiers seront stockées telles et telles informations).

L'intrication forte que nous avons constatée entre composants et masques carte (*cf.* § 2.3.1 page 38) se prolonge par une intrication forte entre les applications utilisatrices et les cartes.

Système hôte	1	Appel d'une commande carte à l'aide d'une API de haut niveau
	2	Transcription de la commande en APDU
	3	Transcription des APDU en TPDU
	4	Envoi des TPDU au lecteur par le protocole de transport TLP 224
Lecteur	5	Réception des TPDU sur la liaison depuis le système hôte par le protocole TLP 224
	6	Envoi des TPDU à la carte par le protocole de transport T=0 ou T=1
Carte	7	Réception des TPDU sur le contact d'entrée/sortie par le protocole T=0 ou T=1
	8	Décodage du TPDU et exécution de la commande par le masque de la carte

TABLE 3.1 - Envoi d'une commande carte depuis une application cliente

La TABLE 3.1 montre comment de façon classique on accède à une carte depuis une application cliente. L'application écrite dans un langage de haut niveau accède à la carte par des appels à l'API carte (ligne 1). Les fonctions de l'API sont implémentées dans une bibliothèque qui transforme les appels en APDU (ligne 2). Cette bibliothèque est fournie par l'émetteur qui a aussi défini le masque (jeu de commandes) de la carte (ligne 8). La transcription des APDU en TPDU et leur transport jusqu'à la carte sont des fonctions standards de lecteurs transparents (par exemple, le GCR200 de Gemplus [Gem92], lignes 5 et 6) et des bibliothèques de pilotage de ces lecteurs (par exemple, le « driver » GCR [Gem93], lignes 3 et 4).

Le développement d'une application cliente pour une carte nécessite donc l'écriture des appels à l'API de la carte dans un langage compatible avec la bibliothèque de fonctions fournie par l'émetteur (qui peut-être de très bas niveau) et pour le système d'exploitation du système hôte. Quand l'application carte est comme aujourd'hui complètement maîtrisée

2. Le protocole de communication entre un système hote et un lecteur n'est pas normalisé, mais est traditionnellement utilisé entre un PC et un lecteur externe relié au port série le protocole TLP 224 conçu par Bull pour ses premiers lecteurs.

par l'émetteur, le choix des plates-formes et le développement des applications hôte sont entièrement déterminés par celui-ci. Cette façon de procéder en circuit fermé permet de ne pas se préoccuper des problèmes d'interopérabilité entre la carte et les applications clientes.

Des facteurs nouveaux viennent pourtant troubler cette apparente absence de besoins en terme d'interopérabilité :

- le développement d'applications s'interfaçant avec plusieurs types de cartes empêche un unique émetteur d'imposer sa bibliothèque d'accès aux différentes cartes,
- le développement de cartes multi-applications, devant s'interfacer avec une plus grande variété de systèmes informatiques, freine l'imposition d'un type de plate-forme unique, et
- la volonté d'intégrer les cartes dans des applications déjà existantes oblige de pouvoir accéder à celles-ci à partir de plateformes non maîtrisées par l'émetteur.

Dans le paragraphe suivant nous montrons comment ces problèmes sont (ou ont été) abordés et pourquoi les solutions proposées restent insuffisantes. Dans le § 3.1.3 page 49 nous montrerons que ce problème d'interopérabilité entre des cartes et des systèmes informatiques devient encore plus crucial pour des cartes génériques.

3.1.2 Vers une interopérabilité cartes et systèmes informatiques

Les trois approches, qui ont jusqu'à aujourd'hui été utilisées, sont détaillées par la suite :

- une approche normative qui tente de promouvoir des commandes carte standards auxquelles toute carte aurait à répondre (§ 3.1.2.1),
- une approche par le bas qui vise à proposer une API fédérant plusieurs masques de cartes en un seul (§ 3.1.2.2 page suivante), et
- une approche par le haut qui se base sur une API applicative dont chaque fonction est « projetée » en un ensemble de commandes carte (§ 3.1.2.3 page 49).

3.1.2.1 Des API orientées normes

Bien que la norme ISO 7816-4 tente de promouvoir des commandes cartes suffisamment générales pour s'appliquer à différents domaines d'application, celle-ci n'est pas aujourd'hui considérée et utilisée comme une API unique d'accès aux cartes. En effet, les principaux obstacles rencontrés face à une telle démarche de définition d'une API carte « universelle » sont les suivants :

- présence de nombreuses extensions et de zones « floues » dans la norme qui ne favorisent pas l'interopérabilité de l'API en elle-même,
- incomplétude de la norme face à des fonctions applicatives particulières,

- difficulté voire impossibilité à faire migrer les applications existantes, sauf à changer toutes les cartes existantes, et
- définition *ad-hoc* d'une API répondant uniquement aux capacités techniques et aux besoins du marché du moment.

La suite des normes ISO 7816-5,6 et 7, qui s'appuient sur la partie 4, n'apportent pas de solutions à ces obstacles car elles n'en sont que des extensions cumulatives liées aux cartes multi-applications et à la gestion des données sous formes de bases de données. Elles augmentent quantitativement la norme ISO 7816-4, elles n'en réduisent pas l'incomplétude interne.

3.1.2.2 Des API carte orientées masques

Dès la fin des années 1980, les besoins d'intégration mais aussi d'interopérabilité apparaissent. Ces besoins ont, en premier lieu, été guidés par l'absence de normes (à l'époque), mais surtout par la volonté des émetteurs de multiplier leur source d'approvisionnement auprès des fabricants ou de se préparer à des migrations possibles entre types de cartes. Pour intégrer des cartes de types différents ou d'origines diverses des initiatives ont parfois abouti à des API d'interface issues du monde industriel :

- SEMCAM, proposition du CRTS[3] de Brest et la société Telesystem, permettait à une application de travailler avec des cartes de plusieurs fournisseurs. L'API était celle de la carte « la plus performante », les fonctions non réalisées par une carte étant prises en charge par le logiciel d'interface.
- CSE[4], proposition de Gemplus, offrait une interface unique pour accéder à l'ensemble des cartes de sa gamme.
- CAPI[5] [LL93], proposition plus récente de la société américaine Applied Systems Institute, offrait le même type de service pour l'ensemble des cartes disponibles au début des années 1990.

À notre connaissance, toutes ces tentatives ont aujourd'hui été abandonnées par leur promoteur à cause d'une définition de l'interopérabilité qui étaient trop orientée par le masque des cartes existantes à un instant t. Leurs fonctionnalités étaient la somme des fonctionnalités des systèmes d'exploitation des cartes pour lesquelles elles offraient une API unique qui pose les problèmes suivants :

- Les fonctionnalités offertes par l'API ne sont pas forcément adaptées aux traitements effectués par les applications qui l'utilisent.
- Si une fonction de l'API n'est pas supportée par une carte, soit l'API ne répond pas et l'application doit effectuer un traitement particulier, soit l'API simule cette opération en pouvant compromettre la sécurité des données dans la carte.
- Si une nouvelle carte apparaît il faut réécrire l'API.

3. Centre Régional de Transfusion Sanguine
4. Card Software Environment
5. Card Application Programming Interface

3.1.2.3 Des API carte orientées applications

Une approche orthogonale à celle présentée dans le paragraphe précédent consiste en la définition d'une API « universelle » d'accès aux cartes basée sur les fonctions applicatives auxquelles les cartes doivent répondre. Une telle API doit permettre d'accéder à différents types de cartes (avec des masques différents) du moment que celles-ci ont été définies pour remplir les fonctions applicatives de l'application cible, par exemple la gestion d'un dossier médical ou la maintenance d'informations de fidélité dans une chaîne de magasins. Nous prendrons comme exemple le cas de l'application « Projet Carte Santé » du Québec pour lequel les promoteurs ont développé un serveur « universel » de carte SCAM[6] [DAP+94]. Ce logiciel se compose de deux couches [Dur92] :

1. une couche logique (sous la forme d'une DLL[7] Windows) implémentant une interface de haut niveau pour manipuler les données d'un dossier patient indépendant de toute implémentation (zones logiques), et
2. une couche physique implémentant l'accès au dossier qui transforme l'accès aux zones physiques en une suite de commandes carte. Le lien entre zones logiques et zones physiques est paramétré par des matrices au chargement du logiciel.

Cette approche permet d'écrire des applications sans se préoccuper de l'interface avec la carte sous-jacente. Elle requiert néanmoins un consensus sur la définition de la couche logique (l'application), et demande à chaque émetteur d'écrire la couche physique qui implémente l'accès au dossier stocké dans une carte (*cf.* FIG. 3.2 page suivante).

3.1.3 Besoins pour une plus grande interopérabilité

Les différents types d'API illustrés précédemment présentent tous la principale caractéristique d'être « carto-centriques » car ils définissent une interface fonctionnelle qui n'est ni indépendante des systèmes d'exploitation et matériels des cartes (API orientées normes ou masques) ni indépendante de la nature de l'application carte (API orientées applications). Bien-sûr, cela semble irrémédiable puisque pour réaliser une application mettant en œuvre des cartes il faudra bien « qu'à un endroit ou un autre » une couche logicielle se situe entre l'application cliente et les cartes utilisées. Néanmoins, cette couche logicielle pour être véritablement une couche d'interopérabilité (*cf.* la définition de middleware d'interopérabilité § 1.3.2 page 10) devrait être suffisamment générale pour offrir un ensemble de services indépendants à la fois des API applicatives et des interfaces des plate-formes carte (*cf.* FIG. 1.4 page 11).

Nous reviendrons à la fin de ce chapitre sur ce plus grand besoin d'interopérabilité notamment dans le cadre de cartes génériques. Auparavant, et pour illustrer notre propos, nous présentons dans les paragraphes suivants deux exemples d'intégration d'une carte non générique (la carte CQL, *cf.* § 2.1.4 page 29) dans des systèmes informatiques qui sont le World Wide Web et les SGBD[8]. Nous verrons que dans le premier cas d'intégration – pour intéressant qu'il puisse être en terme d'application – la solution proposée cumule

6. Serveur de Carte À Mémoire
7. Dynamic Link Library
8. Système de Gestion de Base de Données

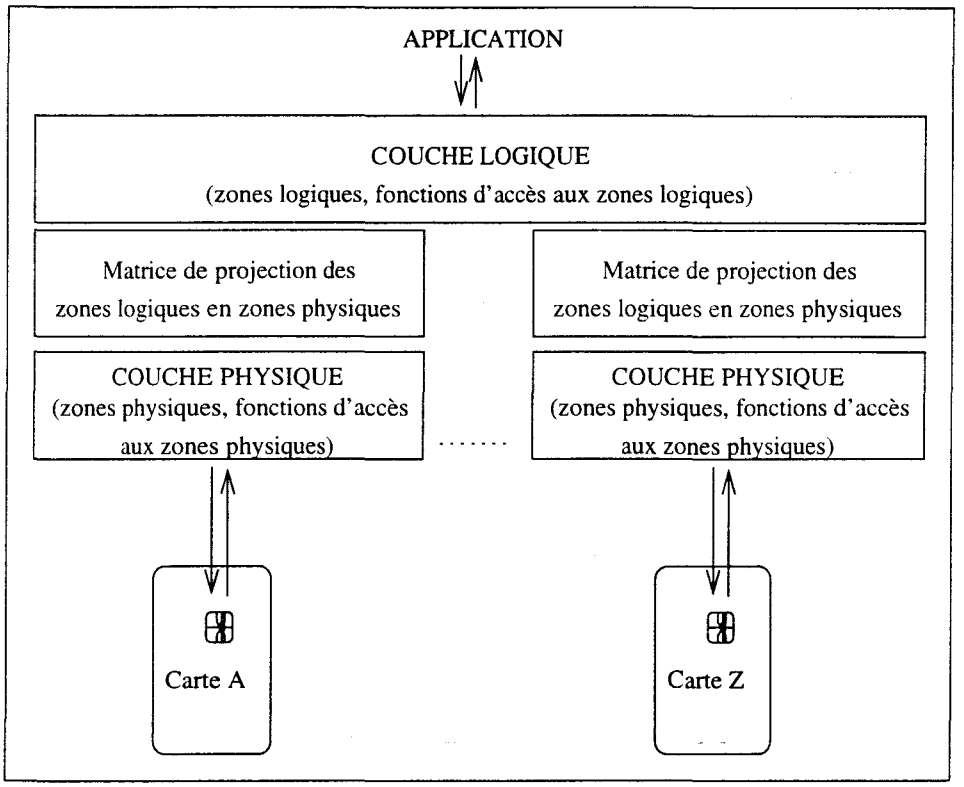

FIG. 3.2 - Couches d'accès à des cartes différentes implémentant un dossier patient

tous les inconvénients des « API carto-centriques », à savoir le développement d'une couche logicielle *ad-hoc* pour l'application et la carte. Le second exemple présente une approche plus intéressante car indépendante à la fois des applications et, dans une certaine mesure, de la carte. En dressant les limites de ces deux exemples particuliers et en nous replaçant dans le cadre des cartes génériques nous montrerons la ligne directrice de notre travail vers un serveur générique de cartes.

3.2 Intégration d'une carte dans le Web : le cas d'une carte patient de suivi médical

Une des applications les plus prometteuses de la carte à microprocesseur est celle du dossier portable regroupant sur un support unique et portable des informations sur une personne. Ces informations sont consultées et mises à jour par différents systèmes d'information faisant de la carte un « espace » de données et de services partagé par un ensemble d'applications. Les problèmes rencontrés par les concepteurs de dossiers portables sont : le partage de cet espace et la sécurité inter-applications, la place mémoire limitée du support (de l'ordre de 10 KO) et l'interfaçage des cartes dans les systèmes hétérogènes.

L'intégration de la carte dans le Web permet, par l'utilisation de navigateurs standards, de proposer une solution au problème de l'interfaçage, et par l'utilisation de liens URL[9] comme données carte, d'étendre virtuellement l'espace mémoire du dossier. Comme

9. Uniform Resource Locator

illustration de cette intégration, nous présentons dans cette partie l'utilisation de navigateurs Web pour la consultation d'informations contenues dans une carte dossier médical [MVD96]. Ces informations sont structurées selon un schéma entité-relation. La carte utilisée pour stocker ce schéma est une carte CQL. La principale caractéristique de ce dossier portable est de contenir des références URL sur des informations stockées en dehors de la carte.

3.2.1 Description des informations stockées dans la carte

3.2.1.1 Schéma entité-relation

La carte dossier portable utilisée dans le prototype présenté ici est basée sur une structure d'« OpenCard» en milieu hospitalier [Sab93]. Cette OpenCard est définie comme une base de données orientée objet pouvant contenir des « adresses » vers des données situées sur des sites distants [MMLM92]. Nous avons repris de ce travail précédent la structure de la base de données carte (qui chez nous n'est pas orientée objet mais purement relationnelle) et la notion d'adresse que nous avons implémentée sous forme de liens URL vers des documents ou services Internet. La carte contient les informations relatives à son porteur lors de ses différents séjours en hôpitaux pour le traitement de pathologies. Elle est consultée et mise à jour par les personnels soignants ou administratifs.

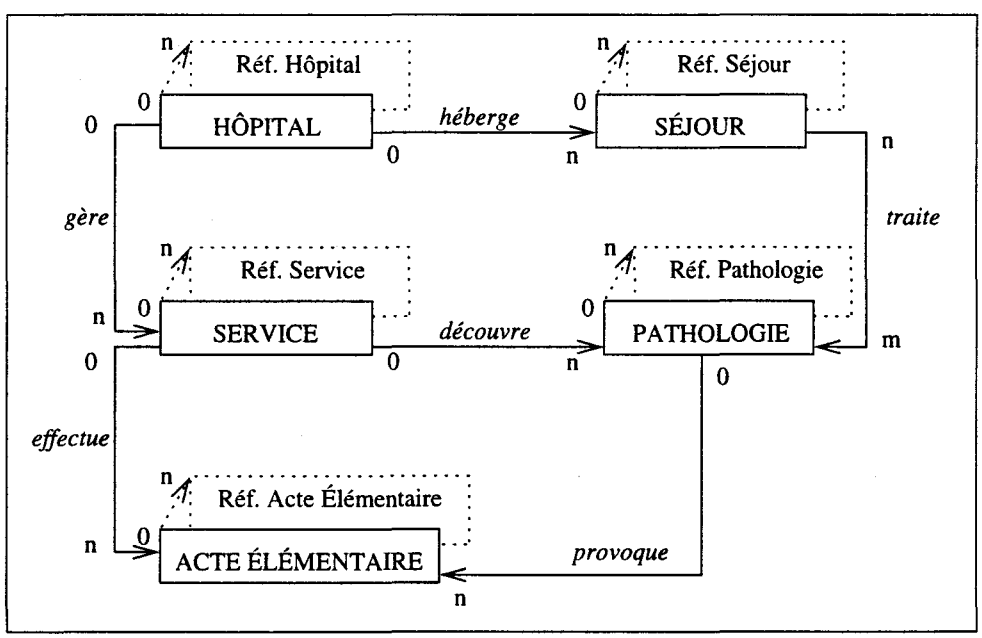

FIG. 3.3 - Schéma entité-relation de la carte patient

Le schéma entité-relation présenté FIG. 3.3 est un sous-ensemble du modèle conceptuel de données spécifié dans [Sab93]. L'entité HÔPITAL peut ***héberger*** plusieurs SÉJOURS, au cours desquels sont ***traitées*** des PATHOLOGIES, chacune pouvant ***provoquer*** des ACTES ÉLÉMENTAIRES. Aussi, l'HÔPITAL peut ***gérer*** plusieurs SERVICES, chacun ayant pu ***découvrir*** une PATHOLOGIE ou pouvant ***effectuer*** des ACTES ÉLÉMENTAIRES.

Aux tables du schéma FIG. 3.3 nous avons rajouté des tables stockant des informations administratives ou des données médicales communes : ADMIN contient les informations administratives du patient (nom, prénom, numéro de sécurité sociale, *etc.*), ALLG un descriptif

de ses allergies, FMIN sa fiche minimale (groupe sanguin, antécédents, commentaires), et VACC son carnet de vaccination.

3.2.1.2 Les tables de liens URL

Les tables de la FIG. 3.3 contiennent uniquement des informations de type texte, des clés d'indexation pour numéroter les enregistrements et des champs de relations entre tables. Par exemple, la table AE qui contient les enregistrements d'actes élémentaires est définie de la façon suivante :

NAE	INT	DATE	TYPE	CR	NSER	NP
N° d'index	Intitulé	Date de réalisation	Type d'acte	Compte-rendu	N° service effecteur	N° pathologie source

TABLE 3.2 - Définition de la table AE des actes élémentaires

Les références URL à des documents ou services Internet peuvent être associées à chacun des enregistrements des tables définies dans le schéma entité-relation. Ainsi, à chaque table est associée par une relation 0--n une table de références URL dont le nom est préfixé par R. La table RAE des URL correspondantes aux actes élémentaires est donc définie de la façon suivante :

TEXT	REF	NAE
Description textuelle de l'URL	Lien URL	N° de l'acte élémentaire

TABLE 3.3 - Table RAE des URL associées aux actes élémentaires

3.2.1.3 Implémentation dans une carte CQL

La carte utilisée pour stocker les informations définies § 3.2.1.1 page précédente est la carte CQL (*cf.* § 2.1.4 page 29). La structure entité-relation du dossier patient s'implémente naturellement dans la carte CQL. Les tables implémentent les entités, et les liens sont matérialisées par des champs de relations qui référencent les index des autres tables. Trois types d'utilisateurs – médecin, infirmière et hôtesse – sont définis dans la carte avec des droits d'accès différents sur les tables; la carte contrôlant elle-même les privilèges à chaque nouvelle session.

3.2.2 Intégration dans le Web

3.2.2.1 Le World Wide Web

Le World Wide Web est un système distribué, développé au CERN[10], pour connecter des bases d'information hétérogènes grâce à la métaphore de documents hypertextes. Le Web repose sur le modèle client-serveur dans lequel des clients (souvent appelés navigateurs) permettent aux utilisateurs de «naviguer» dans un graphe de documents multimédia – tels que des textes, des images, du son ou de la vidéo – stockés dans des serveurs. Les

10. Conseil Européen pour la Recherche Nucléaire

documents hypertextes sont décrits avec le langage HTML[11] [BLC95]. Le langage HTML décrit le contenu et les liens hypertextes (ou ancres) des documents.

Les clients et les serveurs communiquent par le protocole HTTP[12] [BLFFN96]. Un client émet une requête HTTP vers un serveur qui lui répond en lui renvoyant le document demandé. Les documents Web (ou ressources) sont référencés par des URL [BLMM] composées du nom du serveur (ou son adresse Internet) et du chemin d'accès local (c'est-à-dire dans le serveur) de la ressource. Les documents peuvent aussi être générés dynamiquement par des programmes externes appelés scripts CGI[13]. Le protocole CGI [McC95] est un standard pour interfacer des programmes externes avec les serveurs HTTP. Ce protocole définit le format des données échangées entre le serveur et les programmes externes *via* des variables d'environnement. Ces programmes peuvent être écrits dans divers langages de programmation (par exemple, Shell Unix, Perl, C). Un navigateur Web communique des informations à ces programmes à partir de formulaires (menus, zones de saisie) du langage HTML.

3.2.2.2 CGI d'accès à la carte

Pour le prototype développé, l'approche employée consiste à implémenter une couche logicielle intermédiaire pour réaliser l'intégration de la carte dans le Web, et fournir ainsi une interface transparente à la base de données carte depuis les navigateurs. L'intégration réalisée s'est focalisée sur l'utilisation de navigateurs Web pour la consultation des données dans la carte et la navigation sur les liens URL fournis par la carte[14].

De la même façon qu'en cliquant sur un lien dans une page HTML l'utilisateur passe d'un document à l'autre, il peut passer d'un ensemble d'information dans la carte, par exemple, les pathologies, vers un autre ensemble, par exemple, les actes élémentaires. Les liens URL associés à chaque information élémentaire (par exemple, une pathologie ou un acte élémentaire) permettent *de la même façon* d'accéder à des informations beaucoup plus riches du dossier du patient; ces dernières n'étant pas stockées dans la carte mais sur des serveurs du réseau Internet.

La couche d'accès à la carte est réalisée par le programme CGI W2C[15] qui fonctionne de la manière suivante (*cf.* FIG. 3.4 page suivante) :

1. Le navigateur Web présente un formulaire d'accès à des informations du dossier médical. En cliquant sur le lien d'accès à la carte le navigateur envoit une requête HTTP désignant le programme CGI W2C plus des paramètres pour ce programme (*cf.* § suivant l'énumération).

2. Le serveur HTTP (qui peut être localisé au niveau d'une organisation comme un hôpital, ou un centre DDASS[16]) reçoit la requête HTTP du navigateur, lance le

11. HyperText Markup Language
12. HyperText Transfert Protocol
13. Common Gateway Interface
14. Dans un deuxième temps nous nous sommes intéressés à la modification des informations carte *via* des formulaires HTML mais cette partie n'apportant pas de concepts techniques nouveaux nous ne l'aborderons pas ici.
15. Web To Card
16. Direction Départementale des Affaires Sanitaires et Sociales

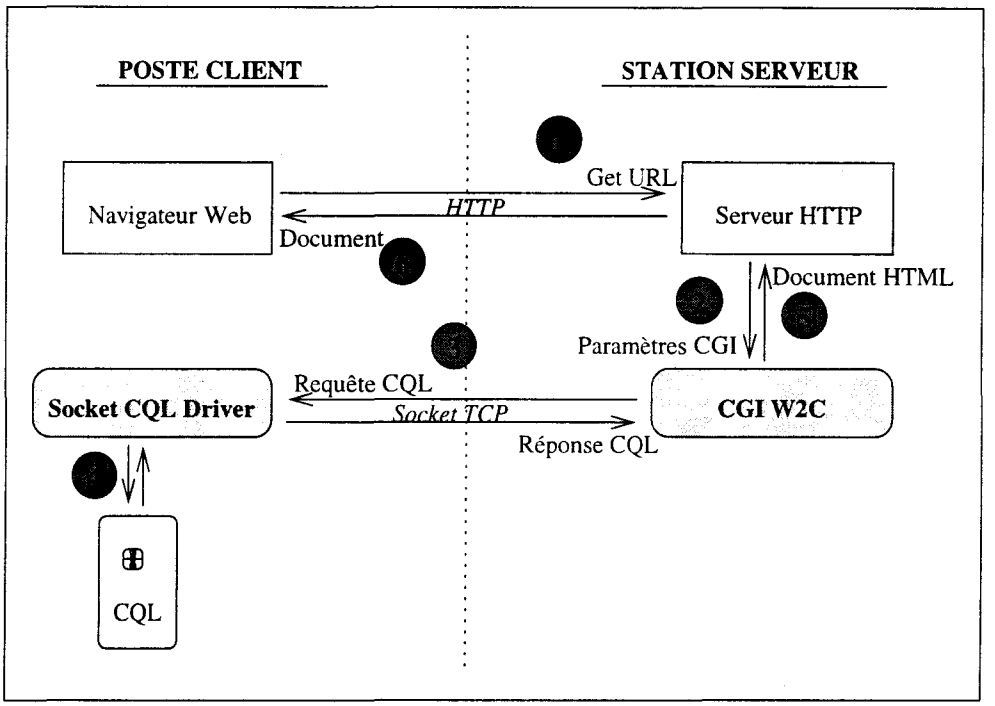

FIG. 3.4 - Architecture d'accès aux données de la carte patient

CGI W2C et lui transmet les paramètres reçus.

3. Ces paramètres indiquent les requêtes CQL que le programme CGI envoie au « Socket CQL Driver » [17] du poste client ayant émis la demande.

4. Le Socket CQL Driver transcrit ces requêtes au format standard APDU et les envoie à la carte en suivant les couches 3 à 8 de la TABLE 3.1 page 46.

5. En retour, les réponses CQL sont transcrites par le programme CGI W2C sous forme de pages HTML incluant les liens URL fournis par la carte.

6. Ces documents sont ensuite transmis au serveur Web qui les envoie au navigateur par le protocole HTTP.

L'accès au programme CGI W2C de l'application se fait en envoyant depuis le navigateur une requête HTTP du type `GET` avec une URL de la forme :

```
http://nom_serveur:80/cgi-bin/w2c?QUERY_STRING
```

La première partie de l'URL (jusqu'au point d'interrogation « ? ») désigne l'accès au programme W2C et permet donc au serveur HTTP d'activiter ce programme. La seconde partie, appelée `QUERY_STRING`, est une chaîne de caractères de la forme `c1c2c3c4c5c6` où :

- `c1` est un caractère désignant les informations à lire dans la carte (par exemple, « `p` » pour les pathologies),

17. Le « Socket CQL Driver » est une implémentation de l'API de haut niveau de la carte CQL qui reçoit les requêtes CQL sous forme de chaînes de caractères sur un socket TCP et renvoie sur le même canal les réponses CQL, elles aussi sous forme de chaînes de caractères.

- c2 est un chiffre donnant le type de l'utilisateur (par exemple, « 1 » pour le profil médecin),
- c3 est un nombre sur 3 chiffres désignant une option de l'action à effectuer (par exemple, « 001 » avec c1 = p pour lire les informations de la pathologie numèro 1),
- c4 est un booléen (0 ou 1) pour indiquer le fait que le navigateur travaille avec ou sans frames,
- c5 est un nombre sur 5 chiffres désignant le port TCP [18] du Socket CQL Driver du poste client (par exemple, « 10001 »), et
- c6 est une chaîne de caractères de longueur variable contenant l'adresse Internet de la station sur laquelle est installé le Socket CQL Driver (par exemple, « 176.16.50.35»).

Dans les pages HTML générées par W2C les URL permettant d'accéder aux informations de la carte sont donc formées avec la même syntaxe, ce qui permet lors de la prochaine activation du programme W2C de conserver des informations d'état avec le poste client [19]. Cela est nécessaire car HTTP est un protocole sans état [Shk96] ce qui nous empêche de conserver dans le programme CGI des informations telles que le profil utilisateur ou le port TCP du Socket CQL Driver installé sur le poste client [20].

3.2.3 Bilan de la première intégration

Les aspects intéressants que nous voulions montrer en développant ce prototype [21] sont :

- l'accès uniforme aux informations d'un dossier par l'utilisation de l'interface graphique et par le mode de navigation hypertexte des navigateurs Web (que les données soient dans la carte ou sur des serveurs distants),
- l'extension du contenu de la carte par l'ajout de liens URL qui référencent des documents non stockés dans la mémoire de la carte,
- l'extension de la richesse des informations de la carte car les liens URL peuvent donner accès à des documents multimédia ou à des services sophistiqués comme des requêtes sur des bases de données, et
- la non redondance des informations référencées par des liens URL puisqu'elles n'ont pas besoin d'être recopiées dans la carte.

Le principal intérêt est donc l'utilisation du Web comme moyen d'accès aux données d'un dossier dont les éléments sont répartis sur des machines différentes, la carte étant

18. Transport Communication Protocol

19. Lors de la connexion à la carte depuis un poste client, un formulaire permet de renseigner certaines de ces informations pour initialiser le programme W2C.

20. Une autre solution aurait pu être d'utiliser les « cookies » maintenus par les navigateurs Netscape, mais nous l'avons écartée pour pouvoir faire fonctionner cette application sans être lié à des systèmes propriétaires.

21. Les programmes Socket CQL Driver et W2C et ont été développés en C pour station Sun Sparc sous Solaris 2.4.

le support individuel et portable d'organisation et de référence des différents éléments du dossier. Cette intégration est complètement transparente pour le Web puisqu'aucune modification n'a été effectuée sur son architecture, ni du coté client, ni du côté serveur[22].

En terme d'intégration de la carte, les enseignements que nous pouvons en tirer sont en fait assez décevants puisqu'il a fallu un programme CGI spécifique pour accéder à la carte. Le programme W2C est en fait doublement spécifique puisqu'il est fait « sur-mesure » (*had-hoc*) pour :

1. L'application dossier médical : toute la partie applicative se trouve dans le programme jusqu'à la mise en forme de l'interface graphique puisque c'est le programme W2C qui génère les pages HTML. Ainsi, pour toute autre application, ce programme devrait être réécrit.

2. La carte utilisée : les commandes envoyées à la carte sont des requêtes CQL et sont basées sur une organisation de type base de données des informations. Pour un autre type de carte il faudrait modifier la nature de ces requêtes et récrire la partie haute (correspondante aux couches 1 et 2 de la TABLE 3.1 page 46) du Socket CQL Driver.

3.3 Intégration d'une carte dans les SGBD : le driver ODBC

Ce second exemple d'intégration part de la constatation qu'un dossier portable stocké sur une carte à microprocesseur assure une fonction de lien de communication entre systèmes d'information qui ne peuvent ou ne veulent pas s'interconnecter pour des raisons techniques, légales, éthiques ou politiques. Par exemple, le projet INCA de la Commission Européenne a étudié la carte comme support d'informations administratives et légales pour le contrôle du transport de marchandises au sein de l'Union Européenne parce que les soucis de sécurité et de politique empêchent de simplement interconnecter les systèmes d'information des différents pays membres [ENS92]. La carte devient alors un élément qui sert de support de données échangées par des systèmes informatiques hétérogènes avec lesquels elles doit pouvoir s'interfacer de façon uniforme.

Se pose alors le problème d'interfacer la carte support de données avec de nombreuses applications différentes sans que les concepteurs ait chaque fois à développer une couche d'accès spécifique. L'utilisation de l'interface ODBC[23] pour accéder à des données permet de s'abstraire de leur implémentation physique. En considérant la carte CQL comme une source de données nous avons développé un driver ODBC permettant de banaliser l'accès à la carte depuis des applications hétérogènes [PV94b, PV94a].

3.3.1 Intégration de la carte depuis de multiples applications

Le problème que nous abordons est celui de l'accès à une carte depuis des applications hétérogènes sans que celles-ci aient chacune besoin d'une API spécifique correspondante

22. Les navigateurs utilisés sont indifféremment Netscape ou Mosaic, le serveur Web étant httpd du NCSA.

23. Open DataBase Connectivity

d'une part à la façon dont l'application manipule les données de la carte et d'autre part à la façon dont ces données sont physiquement implémentées dans une carte. Ce problème est illustré FIG. 3.5.

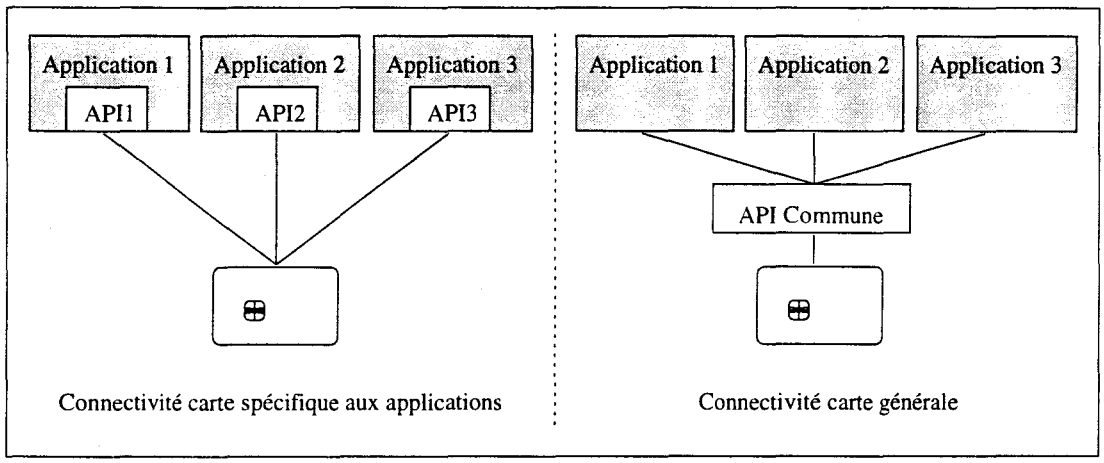

FIG. 3.5 - Connectivité de la carte avec des applications hétérogènes

Les principes que nous avons adoptés pour développer une interface commune d'accès à la carte sont basées sur les recommandations suivantes:

Approche par le haut Il s'agit d'essayer de s'abstraire de l'approche carto-centrique pour s'intéresser proprement dit à la façon dont les applications voient leurs sources de données; la carte n'étant qu'une de ces sources que nous nous proposons d'intégrer comme les autres.

Adoption des standards Pour réussir cette intégration dans le monde des applications informatiques il est important de s'appuyer sur un standard définissant une interface commune d'accès aux sources de données.

Couche de middleware Pour être le plus indépendant à la fois des applications et des sources de données ce standard doit proposer une couche middleware d'interopérabilité telle que celle décrite FIG. 1.4 page 11.

En considérant que les sources de données dans le monde applicatif sont de plus en plus organisées sous forme de SGBD nous avons choisi de travailler sur le standard « du marché » pour l'interopérabilité des bases de données appelé ODBC.

3.3.2 Open DataBase Connectivity

3.3.2.1 Architecture ODBC

ODBC est une implémentation réalisée par Microsoft des spécifications pour la connectivité des bases de données (appelées « SQL CLI[24] ») proposées par le « X/Open SQL Access Group » et aujourd'hui intégrées dans la norme SQL [ISO95]. ODBC définit une API commune d'accès à des bases de données quelconques. La FIG. 3.6 page suivante

24. Call Level Interface

montre l'architecture globale de programmes accédant à des bases de données différentes à travers l'interface commune d'ODBC.

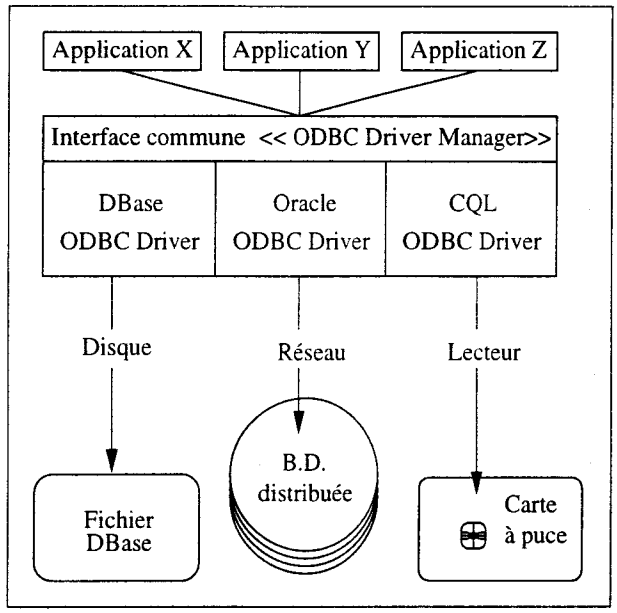

FIG. 3.6 - Architecture ODBC

L'interface ODBC est une API de fonctions qui permet à une application de se connecter à une base de données, d'exécuter des requêtes SQL et de récupérer leurs résultats. Chaque application utilise la même interface (l'API ODBC fournie par le « ODBC Driver Manager») pour accéder à des sources de données différentes grâce à des « drivers ODBC» spécifiques. Un même programme peut aussi accéder à des sources de données différentes (même au cours d'une même session) sans avoir à être recompilé et sans se préoccuper des couches et protocoles de communication sous-jacents. Le TABLE 3.4 montre comment sont traitées les fonctions ODBC depuis une application jusqu'à une source de données ou BD[25].

Application	Appelle des fonctions de l'API ODBC pour se connecter à une source de données, soumettre des requêtes SQL, récupérer les résultats
ODBC Driver Manager	Charge dynamiquement le driver ODBC de la source de données cible, transmet les requêtes à ce driver et retourne les résultats à l'application
Driver ODBC de la base de données	Traite les appels aux fonctions ODBC en extrayant les requêtes SQL pour les envoyer à la base de données cible et en retournant les résultats de la requête sous forme de tuples ou de codes erreurs ODBC
Couche de transport	Transporte les requêtes et les réponses entre la base de données et le driver (dépend de la nature de la BD)
Source de données (BD)	Exécute les requêtes provenant du driver et retourne les résultats au driver

TABLE 3.4 - Traitement des fonctions ODBC

Pour que l'accès à une source de données soit automatique il suffit que le fournisseur de la base de données fournisse le driver correspondant à son produit[26]. Le driver a pour

25. Base de données

26. Aujourd'hui de nombreux éditeurs proposent des drivers pour leur base de données (Oracle, Informix, Access, Paradox, SQL Server, *etc.*), mais aussi des outils de développement d'applications intégrant l'accès

fonction de traiter les appels à l'API ODBC, de transcrire les requêtes SQL en requêtes pour sa propre source de données, de transmettre les requêtes à la base en utilisant les couches basses de communication et de transcrire les réponses de sa source en réponses ODBC.

3.3.2.2 Implémentation d'un driver ODBC

Les fonctions de l'API ODBC sont divisées en trois groupes qui correspondent à des niveaux de conformité avec la norme (*cf.* TABLE 3.5) :

Noyau Le premier niveau correspond à une implémentation stricte du CLI défini par le SAG[27]; il s'agit du plus petit dénominateur commun d'interopérabilité.

Niveau 1 Le deuxième niveau ajoute des fonctions pour obtenir des informations à propos de la source de données et pour manipuler certaines capacités du driver.

Niveau 2 Ce niveau permet de traiter des fonctions plus évoluées de certaines bases de données (procédures stockées, information d'intégrité référentielle, *etc.*).

Fonctionnalités de l'API ODBC	Noyau	Niv. 1	Niv. 2
Se connecter à une source de données	X	X	X
Obtenir des informations sur le driver et la source de données		X	X
Manipuler des options du driver		X	
Préparer des requêtes SQL	X		X
Soumettre des requêtes SQL	X	X	X
Retrouver des résultats et des informations à propos des résultats	X	X	X
Obtenir des informations système sur la source de données		X	X
Terminer un traitement	X		
Terminer une connexion à une source de données	X		

TABLE 3.5 - Fonctionnalités prises en charge par les niveaux de l'API ODBC

Un driver doit au minimum implémenter les fonctions du noyau et peut ensuite n'exécuter que certaines fonctions des niveaux supérieurs. De plus, un driver doit permettre l'exécution de requêtes SQL qui sont elles aussi divisées en trois catégories :

SQL minimum Il s'agit ici aussi du plus petit dénominateur commun d'interopérabilité du driver qui doit au minimum permettre l'exécution de ces requêtes pour, par exemple, accéder à une base de données sous forme de fichier texte. Les requêtes SQL sont les simples `CREATE` et `DROP TABLE`, plus les formes simples des fonctions DML[28] de SQL (`SELECT`, `INSERT`, `UPDATE` et `DELETE`). Elles ne prennent en compte que des formes simples d'expressions logiques pour exprimer des conditions de sélection et ne manipulent que des données de types caractère (`CHAR`, `VARCHAR` et `LONG VARCHAR`).

à l'API ODBC (compilateurs Visual Basic ou C/C++, outils d'interrogations de bases de données comme Crystal Report ou Q by X) ou des passerelles permettant d'accéder à des « back-ends» ODBC (par exemple JDBC pour accéder à des sources de données *via* leur driver ODBC depuis une application Java).

27. X/Open SQL Access Group

28. Data Manipulation Language

SQL noyau Ces requêtes correspondent au niveau de conformité avec les spécifications SQL définies par l'X/Open. On y trouve plus de fonctions du DDL[29] de SQL (`ALTER TABLE`, `CREATE/DROP INDEX`, `CREATE/DROP VIEW`, `GRANT/REVOKE`), toutes les fonctionnalités de la requête `SELECT` avec des fonctions de calculs (`SUM`, `MAX/MIN`, `AVERAGE` et `COUNT`) et plus de types de données (`DECIMAL`, `NUMERIC`, `SMALLINT`, `INTEGER`, `REAL`, `FLOAT` et `DOUBLE PRECISION`).

SQL étendu Il s'agit d'extensions au SQL standard qui correspondent à des caractéristiques avancées de certaines bases de données. Par exemple, les types de données `DATE` et `TIME`, les jointures externes, les fonctions scalaires (fonctions qui opèrent sur des valeurs de données comme par exemple des fonctions de conversion) et les invocations de procédures stockées.

3.3.3 Driver ODBC pour carte CQL

La carte CQL, que nous avons une nouvelle fois utilisée ici (*cf.* § 2.1.4 page 29 et § 3.2.1.3 page 52), se prête idéalement à la fonction de source de données dans une architecture ODBC. En effet, l'organisation interne de ses informations sous forme de base de données nous a permis de réaliser un driver ODBC pour la carte CQL compatible avec le niveau 1 de l'API. La TABLE 3.6 résume les fonctionnalités de CQL qui ont été nécessaires au développement des fonctions du noyau et du niveau 1[30].

Fonctionnalités de l'API ODBC	Niveau	Fonctionnalités de la carte CQL
Se connecter à une source de données	1	Connexion à la carte avec présentation du nom d'utilisateur et du mot de passe
Obtenir des informations sur le driver	1	Interne au driver
Obtenir des informations sur la source de données	1	Lecture de la réponse à la remise à zéro et consultation des dictionnaires de la base de données maintenus dans la carte
Manipuler des options du driver	1	Interne au driver
Préparer des requêtes SQL	Noyau	Vérification syntaxique des requêtes SQL et traduction en requêtes CQL
Soumettre des requêtes SQL	1	Envoi de requêtes CQL à la carte
Retrouver des résultats	1	Manipulation du curseur de résultat maintenu dans la carte (ordre `FETCH`)
Retrouver des informations à propos des résultats	1	Traduction des mots de statut renvoyés par la carte dans les réponses APDU en messages d'erreurs ODBC
Obtenir des informations système sur la source de données	1	Lecture de la réponse à la remise à zéro et consultation des dictionnaires de la base de données maintenus dans la carte
Terminer un traitement	Noyau	Ordre `COMMIT` ou `ROLLBACK` de la carte pour valider ou non une transaction
Terminer une connexion à une source de données	Noyau	Mise hors-tension de la carte

TABLE 3.6 - Fonctionnalités CQL utilisées dans le driver ODBC CQL

29. Data Definition Language

30. Le niveau 2 de l'API n'a aucun besoin d'être implémenté puisque ses fonctions ne peuvent être exécutées par la carte CQL.

La conformité avec le langage SQL est assurée uniquement jusqu'au niveau des ordres simples de manipulation de données (DML) du niveau SQL minimum[31]. La carte CQL n'autorise que le type de données « chaîne de caractères de longueur variable[32] » et permet l'expression de conditions de sélection simples, ce qui correspond au niveau minimal de conformité.

Le driver ODBC CQL a été développé en C pour Windows 3.1. Il comprend deux DLL : une pour l'installation qui permet d'initialiser le « Driver Manager ODBC » et de configurer les paramètres de communication avec la carte (port série du lecteur, vitesse de transmission), l'autre pour traiter les appels aux fonctions de l'API ODBC et dont le principal rôle est de transcrire les ordres SQL en commandes CQL puis de les transmettre à la carte en utilisant le protocole APDU (*cf.* FIG. 3.7).

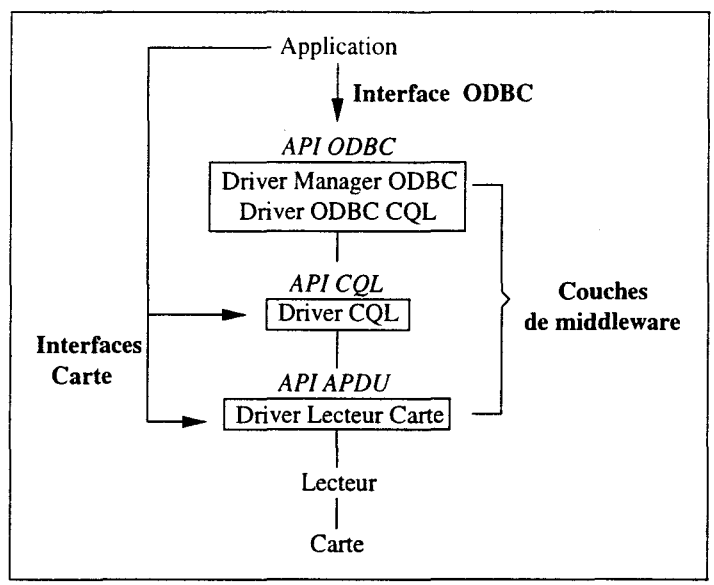

FIG. 3.7 - De l'application à la carte *via* ODBC

Le driver ODBC CQL utilise le driver CQL, qui propose une API de haut niveau (en langage C) pour transformer des requêtes CQL sous forme d'APDU (couches 1 et 2 de la TABLE 3.1 page 46), et un driver de lecteur carte, qui offre une API pour envoyer des commandes APDU au lecteur et à la carte (couches 3 et 4 de la même TABLE). En n'utilisant pas directement ces interfaces « carte» le concepteur d'une application se dégage des contraintes d'une application orientée carte pour se focaliser sur le travail de gestion des informations de sa source de données (quelle-qu'elle soit).

3.3.4 Bilan de la seconde intégration

Le driver ODBC CQL permet d'accéder aux cartes CQL depuis des applications standards du marché sans développer une seule ligne de code spécifique. Il devient ainsi possible

31. Les ordres **CREATE** et **DROP TABLE** pourtant présent dans CQL n'ont volontairement pas été implémentés car ils ne sont généralement utilisés que pendant la phase de personnalisation de la carte et n'auraient donc pas beaucoup d'intérêts à figurer dans un driver ODBC qui a pour but de servir en phase d'utilisation.

32. La longueur étant limitée par la taille du tampon d'entrée-sortie de la carte, soit environ 64 caractères.

de rapidement prototyper, tester, évaluer et valider une application carte. Cette intégration montre donc les points positifs suivants :

- l'accès uniforme aux informations d'une source de données par l'utilisation du langage d'expression de requêtes SQL; que les données soient dans une carte ou sur des bases de données distantes, et
- la possibilité d'intégrer les cartes dans des couches d'interopérabilité de haut niveau de manière transparente.

Par ce type d'intégration il devient possible de réaliser des applications carte utilisant toutes la même métaphore d'accès (la même API) aux données d'une carte. De plus, il faut noter que le driver ODBC réalisé pour la carte CQL aurait aussi pu être fait pour d'autres types de cartes (qui ont une gestion de données sous formes de fichiers) même si cela aurait été plus difficile. Ainsi, des applications ODBC écrites pour manipuler des données dans *une* carte pourrait continuer de fonctionner avec *d'autres* types de cartes.

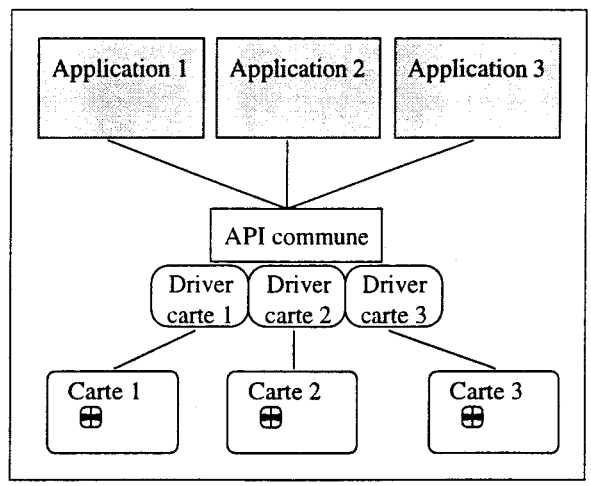

FIG. 3.8 - Connectivité carte généralisée

La connectivité carte généralisée présentée FIG. 3.8 est de la même forme que celle développée pour le projet « Carte Santé » au Québec (*cf.* § 3.1.2.3 page 49). Cependant, la solution ODBC propose véritablement une couche logicielle d'interopérabilité « standard » (car largement répandue aujourd'hui) et indépendante du domaine applicatif (et non limitée au domaine de la santé).

3.4 Vers une intégration de cartes génériques

3.4.1 Intérêts et limites des intégrations actuelles

Les intégrations présentées montrent que la carte peut être considérée comme un élément informatique à part entière – «non exotique» – pouvant s'interfacer avec des systèmes d'information. De part son mode de fonctionnement en système esclave (la carte ne fait que répondre aux requêtes qu'elle reçoit), son intégration dans une application est réalisée comme l'accès à un serveur à travers des couches de communication. Pour faciliter cette intégration, ces couches d'accès doivent être indépendantes des applications et des systèmes

d'exploitation des cartes afin d'être réutilisables et portables. Il s'agit de ne pas avoir à multiplier les développements *ad-hoc* entre applications et cartes mais plutôt de concevoir une couche middleware d'interopérabilité basée sur une interface commune d'accès aux cartes (*cf.* FIG. 3.9).

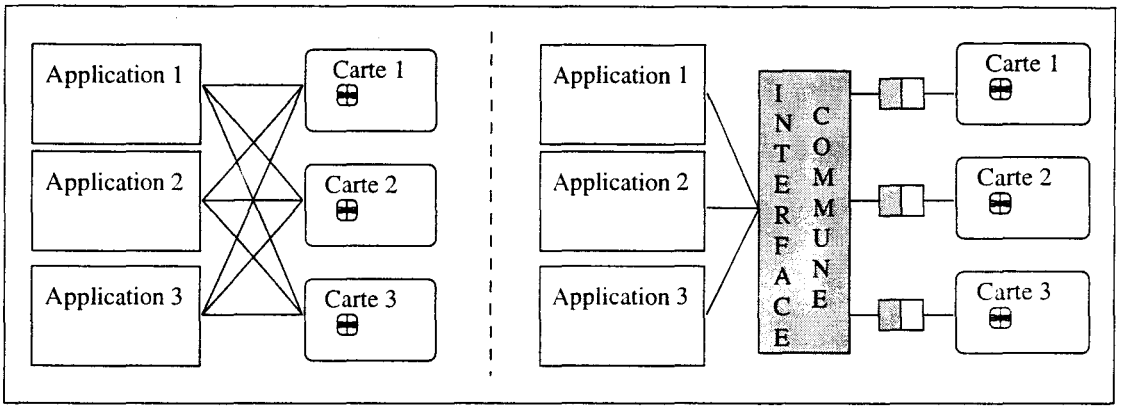

FIG. 3.9 - Intégration *ad-hoc versus* intégration middleware

L'approche middleware illustrée avec le driver ODBC offre le plus haut niveau d'abstraction permettant de développer des applications carte rapidement. Néanmoins, elle requiert le développement de drivers spécifiques pour chaque type de carte (qui sont écrits « une fois pour toutes ») mais ne permet, par l'utilisation du modèle SQL, de ne prendre en compte que des traitements orientés données. La carte est vue comme un serveur de données dont l'accès est rendu commun grâce à l'utilisation d'un langage standard de description et de manipulation de données. Ce procédé correspond bien aux systèmes d'exploitation des cartes actuelles qui sont généralement constitués de commandes de gestion de données.

3.4.2 Intégration de services carte plutôt qu'intégration de cartes

Les futures générations de cartes qui nous intéressent dans ce mémoire sont des cartes dans lesquelles il sera non seulement possible de décrire et manipuler des structures de données, mais surtout de charger et d'utiliser des nouveaux services combinant à la fois des données et des traitements associés. Ainsi, l'intégration de tels types de cartes devra permettre de lancer des requêtes à chaque fois différentes selon les services qui seront présents à un instant `t` dans une carte. L'interfaçage avec une carte ne pourra pas se limiter à une API de manipulation de données mais devra permettre de prendre connaissance dynamiquement des services présents dans une carte, d'offrir aux applications une représentation de ces services indépendantes de leur implémentation, et de transporter les requêtes et les réponses entre une application et un service carte.

C'est à ce second problème que ce mémoire apporte une contribution en :

- montrant l'insuffisance des travaux réalisés jusqu'à ce jour pour permettre d'intégrer des cartes génériques dans les systèmes d'informations (présent chapitre),
- définissant un modèle d'intégration de cartes basé sur une passerelle générique et dynamique (*cf.* Chapitre 4 page 69), et

- réalisant un prototype de cette passerelle sous la forme d'un serveur générique de cartes dans une architecture orientée objet distribuée (*cf.* Chapitre 7 page 123).

Bibliographie sur l'interopérabilité des cartes et des systèmes informatiques

[BLC95] Berners-Lee (Tim) et Connolly (Daniel W.). – *Hypertext Markup Language - 2.0.* – Internet Network Working Group n° RFC 1866. MIT/W3C, novembre 1995. *ftp://ds.internic.net/rfc/rfc1866.txt.*

[BLFFN96] Berners-Lee (Tim), Fielding (Roy T.) et Frystyk Nielsen (Henry). – *Hypertext Transfert Protocol - HTTP/1.0.* – Internet Network Working Group draft n° – Work in progress (expires August 19, 1996), MIT/LCS, UC Irvine, février 19, 1996. *ftp://ds.internic.net/internet-drafts/draft-ietf-http-v10-spec-05.txt.*

[BLMM] Berners-Lee (Tim), Masinter (Larry) et McCahill (Mark). – *Uniform Ressource Locators (URL).* – Rapport technique.

[Car93] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1993 conference proceedings, April 18 to 22, 1993, Arlington, Virginia, U.S.A.*, avril 1993.

[Car94] CardTech/SecurTech, Inc. – *CardTech/SecurTech 1994 conference proceedings, April 10 to 13, 1994, Arlington, Virginia, U.S.A.*, avril 1994.

[CQ94] Cordonnier (Vincent) et Quisquater (Jean-Jacques). – *Proceedings of the first smart card research and advanced application conference, October 24–26, 1994, Lille, France*, RD2P Recherche et Développement Dossier Portable, octobre 1994.

[DAP+94] Durant (Pierre), Ardouin (Pierre), Papillon (Marie-Jo), Gamache (André), Lavoie (Guy), Bérubé (Jocelyn) et Fortin (Jean-Paul). – A universal memory card server, pp. 133–139. – In Cordonnier et Quisquater [CQ94].

[Dur92] Durant (Pierre). – *Architecture et fonctionnalités d'un serveur de cartes à microprocesseur dans la mise en œuvre d'un dossier médical portable*, Mémoire de maitrise, Université Laval, Département des Sciences et de Génie, février 1992.

[ENS92] ENS 1A Projects Commision Européenne. – *INCA: Information Network and Card for the Adapted Management of European Road Transport and Traffic* – 1992. *http://www.sdn.dk/euweb/GENERAL/PROJ1A.html.*

[Gem92] Gemplus. – *GCI200 / GCR200 Manuel de référence (ROS V 7.3)* – 1992, version 1.3.

[Gem93] Gemplus. – *GCR Interface Library Programmer's Guide* – août 1993, version 3.0.

[ISO95] ISO International Standards Organization. – *IS 9075-3: International Standard for Database Language SQL - Part 3: Call Level Interface* – 1995.

[LL93] Lee (Philip S.) et Lee (Ji). – Innovate approaches to smart card integration, pp. 91–96. – In CardTech/SecurTech, Inc. [Car93].

[McC95] McCool (Robert). – *The Common Gateway Interface* – NCSA, 1995. *http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.*

[MMLM92] Möhr (J.R.), McDaniel (J.G.), Lezotte (D.) et Müller (H.A.). – Advanced card technology for an open health record system, *In : Proceedings of MEDINFO 92.* International Medical Informatics Association, – pp. 710–715, 1992.

[MVD96] Merle (Philippe), Vandewalle (Jean-Jacques) et Dufresne (Eric). – Intégration d'environnements hétérogènes : World Wide Web, Carte à microprocesseur et Corba, *In : Actes du XIVieme congrès INFORSID, 4–7 juin 1996, Bordeaux, France.* – pp. 235–248, juin 1996.

[PV94a] Paradinas (Pierre) et Vandewalle (Jean-Jacques). – A Personal and Portable Database Server: the CQL Card, *In : Applications of Databases, First International Conference, ADB-94, Vadstena, Sweden, June 1994, Proceedings*, éd. par Litwin (Witold) et Risch (Tore). – pp. 444–457, juin 1994.

[PV94b] Paradinas (Pierre) et Vandewalle (Jean-Jacques). – How to integrate Smart Cards in Standard Software without writing specific code ?, pp. 69–86. – In CardTech/SecurTech, Inc. [Car94].

[Sab93] Sabre (Sébastien). – *Etude de la mise en place d'une OpenCard en milieu hospitalier*, Rapport de stage effectué au CERIM, École d'ingénieur EUDIL, juin 1993.

[Shk96] Shkarl (Leon). – Web Access to Legacy Data, – *In : Tutorial Notes of the Fifth International World Wide Web Conference, Paris, France, may 6–10, 1996*, mai 1996.

Deuxième partie

Réalisations

CHAPITRE 4

Modèle de système à cartes génériques

« L'automation ne supprime pas la possibilité d'erreur humaine, elle transfère cette possibilité d'erreur du stade de l'action au stade de la conception et du développement. »

Andrew Stratton, cité par Paul Virilio dans *Exposer l'accident*, *in* Les réthoriques de la technologie, Traverses, numéro 26. Centre Georges Pompidou, Paris, France, octobre 1982.

Résumé

Dans ce chapitre nous présentons un modèle général de systèmes à cartes génériques basé sur une définition sous forme d'objet des services cartes. Dans un premier temps, nous faisons un tour d'horizon des recherches qui ont été ou sont menées dans ce domaine et en tirons une définition des cartes génériques qui est, en fait, tout entière sous-tendue par un nouveau cycle de vie appliqué à la carte à microprocesseur. Ensuite, nous détaillons les trois aspects principaux de notre modèle : authentifier et contrôler les droits d'accès des créateurs et utilisateurs des services cartes; créer, charger et exécuter de façon sécurisée ces services dans des cartes génériques; interfacer ces services *via* une couche middleware d'accès générique depuis des applications clientes.

Sommaire

Tables

Figures

4.1 Cadre général

Nous présentons dans ce chapitre la notion de carte générique telle qu'elle s'est dégagée des travaux universitaires précèdemment menés et des recherches industrielles en cours actuellement. Ensuite, nous définissons ce que nous entendons par « carte générique » pour préciser les enjeux de recherche qui sont le thème de notre travail.

4.1.1 Travaux vers des cartes génériques

4.1.1.1 La carte blanche

◄► Des spécifications pour permettre un cycle de vie plus souple des cartes à microprocesseur ont été décrites par [Van91, Car93, Pel95]. Elles définissent un nouveau cycle de vie des cartes leur permettant à tout moment de charger de nouveaux services, et de les proposer aux utilisateurs des cartes. Le modèle dit de la « carte blanche» définit une carte qui lors de son émission est vierge de tout code applicatif. Le porteur d'une telle carte peut alors à tout moment faire charger sur sa carte des services. Ces services sont fournis par un prestataire de services. Un prestataire de services charge dans la mémoire non volatile de la carte les données rémanentes du service, les fonctions qui permettent de manipuler ces données, puis les droits d'accès et fonctions de sécurité relatives à l'utilisation du service. La carte peut ensuite être utilisée dans les guichets qui interagissent avec ce service dans la carte. L'intérêt est que la carte n'est plus dédiée dès l'origine à une seule application (ou à un ensemble figé d'applications) comme actuellement. Pour cela il est nécessaire de développer un système d'exploitation permettant le chargement (et le déchargement) de services tout au long du cycle de vie de la carte, assurant des contrôles d'accès sur ces

services et offrant les mécanismes de base de communication entre la carte et le monde extérieur.

4.1.1.2 Systèmes autour de la carte blanche

Certains auteurs se sont attachés à définir un système d'exploitation dans le modèle de la carte blanche mais pour des machines allant des cartes à microprocesseur jusqu'aux objets nomades en général [Van91, Pel95]. Ceci donne une latitude beaucoup plus large quant aux ressources physiques de la machine cible. notamment en ce qui concerne la taille des mémoires. Le stockage de données et aussi de codes exécutables dans la mémoire de la machine est dès lors envisageable. Ainsi, Peltier (Thierry) [Pel95] a pu définir un système d'exploitation capable de mettre en oeuvre le modèle de la carte blanche en s'intéressant aux questions de code s'exécutant sur place, de code réentrant, mais aussi de partage de données et de fonctions entre différents services et aux problèmes d'exclusion mutuelle.

Carlier (David) [Car93] s'est intéressé aux problèmes de l'évolution des services dans la carte blanche, et à ceux de l'évolution des applications dans les systèmes hôtes qui accèdent aux services de cette carte. S'appuyant sur le paradigme orienté objet, il a défini les services dans la carte comme des objets et a supposé que l'évolution de ces services se fassent au moyen de l'héritage. Les applications dans les systèmes hôtes, quant à elles, accèdent aux services de la carte par l'invocation de méthodes. Il a ensuite étendu la notion de conformité et utilisé la notion de point de vue afin de permettre à une application d'accéder à l'objet dans la carte pour lequel elle a été définie (même si l'objet a été redéfini). De même, une application, prévue pour utiliser des services redéfinis dans la carte, peut continuer de fonctionner avec des cartes qui disposent d'anciennes versions de ces services.

4.1.1.3 Recherches industrielles

Depuis 1996 l'intérêt des industriels pour des cartes génériques s'est développé de façon très importante. De nombreux projets ont, semble-t-il, été lancés sans que nous ayons beaucoup de renseignements à leur propos, à part des informations commerciales (annonces) ou de courtes présentations très générales. Parmi ceux annoncés deux types se dégagent :

1. Des projets qui ont pour objectif d'aboutir à la réalisation d'une carte complètement générique en la dotant d'un interpréteur permettant d'exécuter du code chargé dynamiquement dans la carte :

   - la carte de la société américaine Integrity Arts qui propose un langage de définition d'applications carte sécurisées permettant de charger dans la carte du code interprété et dans le terminal le logiciel d'interface permettant de dialoguer avec l'application chargée, et
   - la carte Cyberflex de la société française Schlumberger qui propose un interpréteur du langage Java de Sun Microsystems pour exécuter du code dans la carte.

2. Des projets visant à coupler un interpréteur avec un système d'exploitation carte «classique» (basé soit sur la norme **ETSI!** [1] TE9 ou ISO 7816-4) pour lui permettre d'adapter son comportement par des scripts téléchargés :
   - la carte HOST de la société française SYSECA, et
   - la carte IOS [Far96] de la société italienne Incard.

4.1.2 Définition des cartes génériques

Nous appelons *carte générique*† une carte à microprocesseur capable au cours de son cycle de vie de charger et décharger de nouvelles fonctionnalités devenant disponibles pour les utilisateurs sous la forme de commandes carte. Ces fonctionnalités sont regroupées sous la forme de services comprenant des données et des traitements s'effectuant sur ces données. Il s'agit donc de nouveaux programmes chargés et exécutés dans la carte ainsi que de nouvelles données chargées dans la mémoire rémanente de la carte. Ce principe de fonctionnement va à l'encontre de l'usage classique des cartes où habituellement l'ensemble des fonctionnalités de la carte est figé après l'étape de personnalisation. La FIG. 4.1 montre de manière qualitative comment une carte «classique» et une carte générique voient leurs fonctionnalités évoluer au cours de leur cycle de vie.

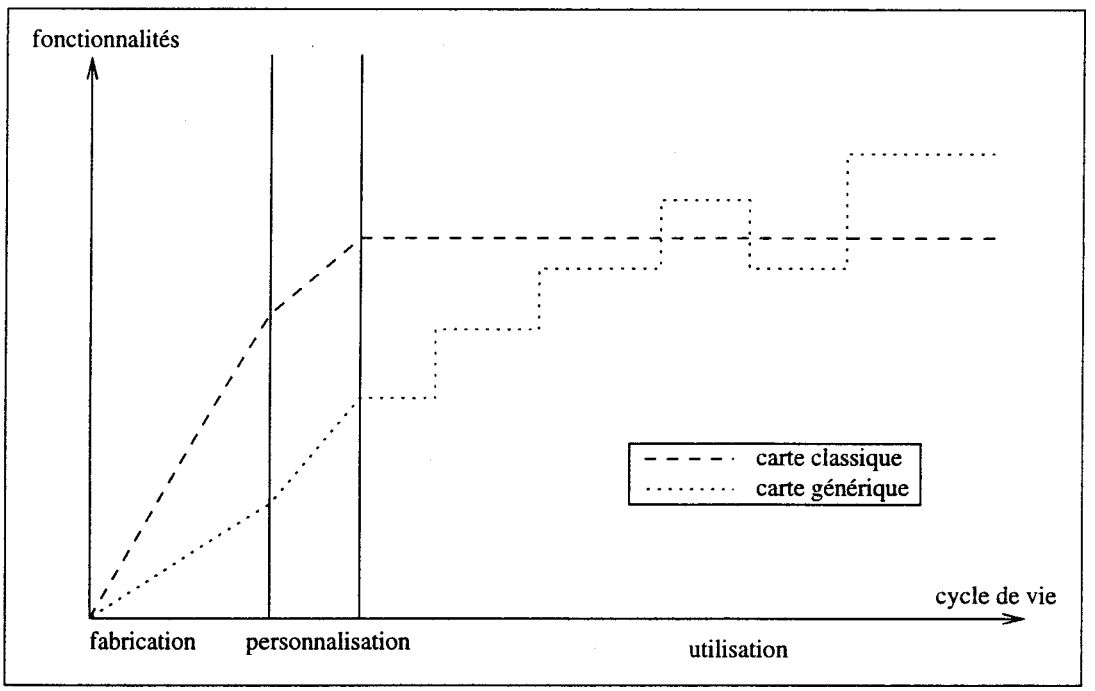

FIG. 4.1 - Évolution du niveau de fonctionnalité des cartes

Dans le cas d'une carte classique les fonctionnalités et le système d'exploitation sont définis pendant l'étape de fabrication et sont «masqués» en mémoire ROM une fois pour toutes. Pendant la personnalisation de la carte par l'émetteur certaines fonctionnalités peuvent être ajoutées sous la forme de programmes filtres. Ensuite, et durant toute son utilisation, la carte ne fait que répondre aux commandes qui lui ont été chargées au

1. **ETSI!**

préalable; sans jamais pouvoir héberger de nouveaux programmes qui lui permettraientt d'offrir de nouvelles fonctionnalités – cette possibilité lui étant interdite par le système d'exploitation qu'elle contient et qui est inaltérable et par la confusion délibérée entre fonctions système d'exploitation et fonctions applicatives.

Dans le cas d'une carte générique les fonctionnalités chargées pendant l'étape de fabrication correspondent au noyau minimum (ou système d'exploitation) de la carte qui vont lui permettre par la suite d'héberger de nouvelles fonctionnalités. Pendant la phase de personnalisation l'émetteur peut, dans ce cas-là aussi, ajouter des fonctionnalités soit en étendant le noyau de base du système d'exploitation, soit en utilisant la fonction de chargement de services. Enfin, pendant la phase d'utilisation, des prestataires de services vont pouvoir charger dans la carte de nouvelles fonctionnalités (croissance du graphe) qui deviendront alors automatiquement disponibles pour les utilisateurs. De même un prestataire de services pourra aussi retirer un service après un temps donné d'utilisation (décroissance du graphe)[2]. On peut même imaginer que des services temporaires soient automatiquement effacés par le système d'exploitation au delà de leur date limite de validité ou au delà d'un certain nombre d'utilisations.

4.1.3 Cycle de vie de la carte générique

Ce mode de fonctionnement des cartes génériques s'appuie sur une nouvelle définition du cycle de vie de la carte identique dans ses grands traits au modèle de la carte blanche [Pel95]. Le cycle de vie de la carte générique ressort principalement des différents partenaires de la carte et de leurs fonctions propres. Des fonctions exécutées par chacun des partenaires se dégagent des besoins spécifiques à partir desquels nous définissons un modèle de système à cartes génériques permettant de prendre en charge ces besoins.

4.1.3.1 Partenaires et rôles

Interviennent au cours de ce cycle cinq catégories de partenaires (la FIG. 4.2 page suivante schématise les relations entre les trois principaux partenaires) :

Le fabricant Comme aujourd'hui il intègre le microcontrôleur sur le support plastique et charge le système d'exploitation de la carte. Ce système d'exploitation est défini classiquement [Tan89] comme le gestionnaire des ressources matérielles qui fournit les fonctions de chargement et d'exécution de programmes. Il doit aussi fournir des fonctions de sécurité permettant notamment de contrôler les utilisateurs de la carte. Aucune fonction applicative n'est alors présente dans la carte.

L'émetteur Il est l'autorité responsable de la délivrance de la carte à un porteur. Pour cela, il réalise l'opération de personnalisation de la carte qui consiste à identifier la carte à son porteur (avec, par exemple, l'enregistrement de son NIP[3]). Il est aussi l'autorité qui distribue des autorisations de charger des services à des prestataires de services; ces autorisations étant par la suite vérifiées par la carte. L'émetteur peut lui aussi charger des services.

2. Pour l'instant nous demeurons volontairement flous sur les termes de *service, donnée, programmme, utilisateur, etc.* parce que nous préciserons leur sens au cours de ce chapitre.

3. Numéro d'Identification Personnel

Les prestataires de services Ils sont des fournisseurs d'applications carte. Pour cela, ils définissent des services qu'ils peuvent, sous réserve de posséder une autorisation de l'émetteur, charger dans la carte à la demande du porteur. Ils distribuent aussi aux clients de leurs services des autorisations pour utiliser ces services. Pour que les clients puissent développer des applications utilisant ces services, les prestataires de services distribuent aussi une interface d'accès à leurs services.

Les clients Ils sont les utilisateurs finaux des services chargés dans la carte. Ils développent des applications utilisant les services d'une carte à l'aide des interfaces reçues des prestataires. L'utilisation d'un service se fait, après vérification par la carte de l'autorisation reçue d'un prestataire, en demandant l'exécution par la carte de fonctions du service.

Le porteur Il est le propriétaire de la carte. En ce sens il est à l'origine de toutes les opérations effectuées par la carte (chargement, déchargement et exécution de services) en autorisant ou non celles-ci et en validant ou non par la présentation de son NIP certaines transactions.

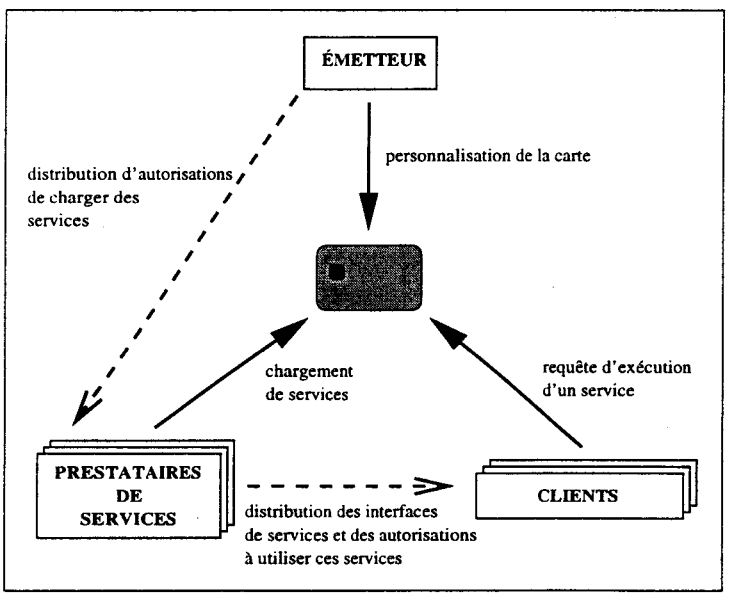

FIG. 4.2 - Cycle de vie des cartes génériques

4.1.3.2 Impacts du cycle de vie

Les impacts du cycle de vie présenté ci-dessus concernent principalement la gestion des services qui doit s'effectuer dans un environnement sécurisé. En effet, il est nécessaire de veiller à ce que les différents partenaires de la carte prouvent leurs droits pour pouvoir effectuer une tâche. La principale difficulté provient du fait que la carte ne connaît pas à l'avance ni les prestataires de services ni les clients. Chacun de ces partenaires est pourtant placé sous la responsabilité d'une autorité. Ainsi, les prestataires de services sont accrédités par l'émetteur de la carte et chaque prestataire de services accrédite ses clients. Il est donc nécessaire de mettre en place un mécanisme de sécurité qui permette à la carte de contrôler ces accréditations.

Le second impact concerne la nature des services qui sont constitués de données et de programmes stockés dans la carte. Les programmes s'exécutent dans la carte sur les données du service. Il est important de veiller à ce que ces programmes ne puissent pas, d'une part utiliser (et surtout modifier) les données d'un autre service, d'autre part modifier le fonctionnement général de la carte en prenant la main sur le système d'exploitation. Il est donc nécessaire de définir à l'intérieur de la carte un environnement d'exécution sécurisé de programmes.

Enfin, les clients doivent pouvoir développer des applications utilisant les services d'une carte de façon rapide puisqu'ils doivent en permanence s'adapter aux nouveaux services apparaissant sur le marché. Par conséquent, il est intéressant de proposer une méthode de développement d'applications carte permettant de « facilement » configurer un outil générique d'accès à la carte à partir d'une description des services qu'elle contient.

4.2 Modélisation

L'approche que nous avons retenue pour adresser les trois besoins induits par le cycle de vie de la carte générique consiste en un modèle général de système à carte générique dans lequel :

- Nous définissons une architecture générale de sécurité permettant à tout moment d'authentifier les partenaires de la carte et de vérifier leurs droits.
- Nous définissons les services chargés dans la carte comme des objets[4] dont la nature permet de contrôler finement leur exécution.
- Nous définissons les descriptions de services comme des interfaces d'objets qui sont utilisés pour paramétrer un serveur générique de cartes dans les applications clientes.

4.2.1 Architecture de sécurité

La carte générique a besoin au cours de son cycle de vie de contrôler les utilisateurs qui accèdent aux services qu'elle contient. Pour cela, il lui est nécessaire d'authentifier deux catégories d'utilisateurs : les prestataires de services et les clients[5]. Cette authentification sera la base du contrôle par le système d'exploitation des droits d'un utilisateur à exécuter une opération sur un service. Les opérations que peuvent réaliser les utilisateurs sont pour un prestataire de services, le chargement/déchargement de services, pour un client, les opérations qu'il est autorisé à invoquer sur des services. Les autorisations d'un prestataire de services sont attribuées par l'émetteur de la carte, celle d'un client sur un service par le prestataire de services qui a chargé le service dans la carte (*cf.* TABLE 4.1 page suivante).

4. Au sens de l'orienté objet...

5. Le fabricant et l'émetteur n'interviennent qu'une fois, en début de cycle de vie de la carte. Le rôle du porteur est uniquement d'autoriser ou de refuser une transaction (entre sa carte et les autres utilisateurs : prestataires de services ou clients) par une présentation de code secret. Leur authentification n'est donc pas un problème qui demande la mise en place d'une architecture de sécurité complexe; elle n'est donc pas traitée ici.

Utilisateurs	Droits	Accordé par
Prestataire de services	Chargement/Déchargement de services	Émetteur
Client	Exécution d'une opération d'un service	Prestataire ayant chargé le service

TABLE 4.1 - Droits d'accès

Deux particularités se dégagent du cycle de vie de la carte générique :

- La communauté des utilisateurs est non close, c'est-à-dire que l'ensemble des prestataires de services et des clients potentiels de la carte n'est pas connu. À tout moment un nouveau prestataire de services peut être autorisé par l'émetteur à charger des services dans une carte générique déjà en circulation (c'est-à-dire émise et appartenant à un porteur). À tout moment un nouveau client peut être autorisé par un prestataire de services à utiliser un service qui aurait déjà été chargé dans de nombreuses cartes en circulation.
- La communauté des services est non close, c'est-à-dire que l'ensemble des services potentiels de la carte n'est pas connu. À tout moment un nouveau service peut être ajouté dans une carte par un prestataire de services. L'accès aux opérations de ce nouveau service devant immédiatement être contrôlé par la carte pour tous les futurs clients potentiels.

Il est donc impossible de fonder le contrôle d'accès sur une matrice d'accès (concept défini par Lampson (B.W.) en 1974 [Lam74] illustré TABLE 4.2) prédéfinie et chargée dans la carte qui permettrait d'associer à chaque opération une liste exhaustive des utilisateurs autorisés. En effet, il faudrait pour cela pouvoir mettre à jour *en temps réel* ces matrices dans toutes les cartes déjà en circulation, ce qui est inenvisageable dans notre cas.

Services / **Utilisateurs**	S_1	S_2	S_3
U_1	Op_1, Op_2, Op_3	Op_1	Op_1, Op_3
U_2	Op_1, Op_3	Op_1, Op_2	
U_3		Op_1	Op_1, Op_2, Op_3

TABLE 4.2 - Matrice d'accès

Classiquement, une matrice d'accès peut être factorisée de deux manières :

Par colonne : les listes de contrôle d'accès. À chaque service est associée la liste des opérations exécutables par les utilisateurs. Cette liste est stockée avec le service [Pfl89]. Par exemple, la liste de contrôle d'accès du service S_3 est $\{U_1(Op_1, Op_3), U_3(Op_1, Op_2, Op_3)\}$. Avant chaque exécution d'une opération sur le service, la liste est parcourue pour vérifier si l'utilisateur à le droit de lancer cette exécution.

Par ligne : les listes de capacités. À chaque utilisateur est attribué un ensemble de capacités qui représentent les opérations que cet utilisateur peut faire exécuter [dVdVG95]. Par exemple, la liste de capacités de l'utilisateur U_3 est $\{S_2(Op_1), S_3(Op_1, Op_2, Op_3)\}$.

Pour un système à cartes génériques le mécanisme des listes de contrôle d'accès est lui aussi impraticable. Il faudrait que le prestataire de services puisse, au moment où il charge

un service, donner la liste totale des clients actuels et futurs qui pourront exécuter des opérations sur son service. À l'inverse, le mécanisme des listes de capacités s'applique bien à notre problème puisqu'il permet à un prestataire de services de distribuer ces capacités aux clients même après avoir chargé le service dans de nombreuses cartes[6]. La carte devant recevoir au moment du chargement d'un service des informations qui lui permettront par la suite de vérifier la validité des certificats des clients.

Cependant, si nous nous en tenons *stricto sensu* à sa définition une capacité ne contient pas l'identité de l'utilisateur et peut donc être propagée sans aucun contrôle par l'émetteur de la capacité. Il est donc nécessaire d'ajouter à une capacité des informations permettant à la fois de prouver qu'elle a bien été émise par le propriétaire du service et de prouver que l'utilisateur qui la détient est bien celui à qui elle était destinée. Nous appelons de telles « capacités nominatives » des *certificats* qui en combinant capacités et techniques cryptographiques permettent :

- à un prestataire de services de s'authentifier comme prestataire de services en prouvant qu'il a bien reçu de l'émetteur de la carte sa capacité à charger des services, et
- à un client de s'authentifier comme client d'un service en prouvant qu'il a bien reçu du prestataire de services sa capacité à exécuter des opérations du service.

Ce mécanisme de certificats a d'ailleurs été étendu à des applications à grande échelle intégrant le Web, CORBA et des cartes [Hor96] dans le cadre d'une étude menée sur des travaux communs à Merle (Philippe) dans le cadre de sa thèse [Mer97] et à l'application de carte santé intégrée au Web et présentée § 3.2 page 50.

4.2.2 Modélisation des services cartes comme des objets

De façon un peu grossière, une carte à microprocesseur classique peut être vue comme un objet, elle contient des données qui ne sont accessibles que par son interface (le jeu de commande de la carte) : si un utilisateur souhaite accéder à ses données il doit « poliment » en faire la demande auprès de l'interface de la carte. De plus, la représentation interne des données, l'implémentation de ses commandes d'interface et l'état interne d'une carte sont encapsulés par la boîte noire qu'est son système d'exploitation masqué en ROM[7]. Néanmoins, cet « objet système d'exploitation » offre une interface unique et inaltérable pour toutes les données que la carte contient, même si ces données sont utilisées pour des services différents. Cette caractéristique fait du système d'exploitation carte un objet dont la multiplicité des méthodes mêle fonctions système et opérations applicatives hétérogènes (cas des cartes multi-applicatives). Elle est contraire aux bonnes pratiques de la programmation orientée objet où, en fait, chaque service carte devrait apparaître comme un objet offrant l'interface la mieux adaptée pour manipuler les données propres au service encapsulées au sein de l'objet.

6. Cette même technique s'applique aussi pour les autorisations à charger/décharger des services distribuées par l'émetteur aux prestataires de services.

7. Un premier parallèle entre les cartes et les objets avait déjà été tenté par Jean-Marc Geib en 1991 [Gei91].

Les cartes génériques permettant le chargement/déchargement dynamique de services pourraient tirer bénéfices des technologies orientées objet en terme de développement logiciel (méthodologies orientées objet), partage de code (mécanisme de classes), réutilisabilité (héritage), *etc.* Cependant, ces techniques seraient encore bien trop lourdes à utiliser et à implémenter pour la réalisation de relativement petits services pour carte. Par contre, en ce qui concerne le problème de l'environnement d'exécution sécurisé des services au sein de la carte, l'approche orientée objet en obligeant l'écriture de programme sous forme de petits composants autonomes apportent deux caractéristiques intéressantes [BGV96. PV96]:

Modularité Des services carte définis sous forme d'objets sont des petits modules qui exécutent chacun des tâches indépendantes les unes des autres. Ceci permet de les tester plus facilement puisque chaque module a des entrées/sorties clairement identifiés et que chacune des opérations d'un module est sans effet de bord sur les autres modules. Cette caractéristique apporte une qualité d'*auditabilité* des services carte.

Encapsulation L'encapsulation signifie une indépendance complète entre les objets, leurs interactions étant restreintes à des interfaces bien définies. Il s'agit d'une forme d'isolation qui permet d'assurer que chaque opération d'un service n'opère que sur les données propres au service et n'invoque que les autres opérations internes du service. Les échanges d'information entre services sont limités à des messages entre objets qui restent sous le contrôle de l'environnement d'exécution (le système d'exploitation). Ces messages correspondent à des invocations d'opérations disponibles à l'interface d'un objet (*cf.* FIG. 4.3).

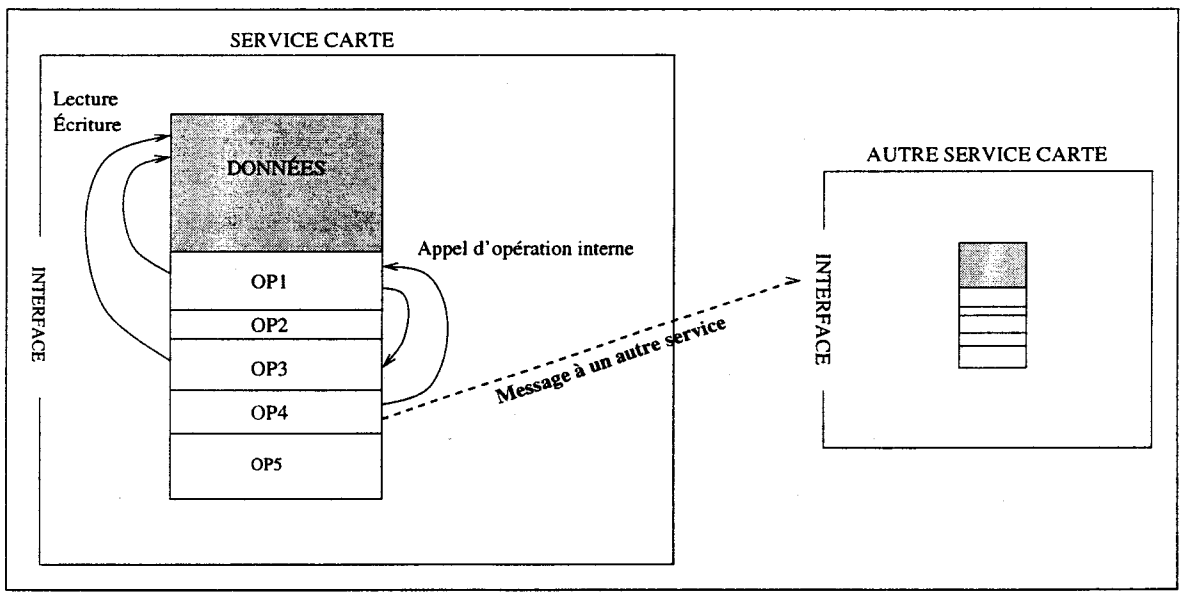

FIG. 4.3 - Encapsulation des services cartes

Le principe d'encapsulation des objets permet d'isoler l'exécution d'une opération d'un service à l'intérieur de l'espace d'adressage de l'objet : les ordres de lecture/écriture ne peuvent accéder qu'aux données du service, et les appels aux sous-routines qu'aux autres opérations internes du service. Néanmoins, pour réaliser cette fonctionnalité, il est nécessaire que l'environnement d'exécution fourni par le système d'exploitation de la carte assure que l'exécution du code ne puisse pas déroger à ces règles. Il s'avère aujourd'hui que

si le code exécuté est du code natif il sera impossible d'assurer que le principe d'encapsulation soit respecté par une opération d'un service qui agirait frauduleusement[8]. En effet, les contraintes matérielles des microcontrôleurs carte ne permettent pas de restreindre les accès d'un code natif applicatif par rapport à un code natif système. Ces contraintes sont :

- l'absence d'un mode utilisateur qui permettrait de restreindre les possibilités pour un code applicatif d'exécuter certaines instructions ou d'accéder à certains registres; ce mode utilisateur étant placé sous le contrôle d'un mode privilégié ou mode superviseur dans lequel s'exécute le système d'exploitation, et
- l'absence d'un gestionnaire de mémoire (ou MMU[9] en anglais) qui permettrait de définir de façon matérielle les zones mémoires EEPROM auxquels un code applicatif pourrait avoir accès (les matrices de sécurité vues § 2.2.2.1 page 35 restent trop imprécises puisqu'elles ne permettent que de limiter globalement les accès à un type de mémoire depuis un autre type).

La solution à ce problème consiste donc à créer un mode utilisateur « artificiel » lors de l'exécution d'un code applicatif (le code d'une opération d'un service dans notre cas) en faisant interpréter ce code sur une machine virtuelle plutôt qu'en l'exécutant directement sur la machine physique (en langage machine). Cette idée d'un interpréteur dit « sécurisé » pour cartes a pour la première fois été proposée par Coulier (Charles), Grimonprez (Georges) et Gordons (Édouard) [CGG93] et était énoncée de la façon suivante :

> « *Un interpréteur est installé en ROM. Il a la charge de l'exécution des programmes ainsi que du contrôle des accès aux mémoires. Il constitue une interface logicielle incontournable entre les programmes applicatifs et le matériel.* »

Nous reprenons entièrement cet énoncé mais le traduisons dans notre contexte objet de la façon suivante (*cf.* FIG. 4.4 page suivante) :

1. Lorsqu'un client demande l'exécution d'une opération d'un service, le système d'exploitation de la carte générique recherche ce service dans ses tables systèmes afin de pouvoir initialiser les trois registres de sécurité `rd`, `ri` et `rf` qui limitent les espaces de données et de code.
2. Le système d'exploitation lance ensuite l'interpréteur (passage en mode utilisateur) ayant comme première instruction à exécuter le point d'entrée de l'opération invoquée.
3. L'interpréteur exécute séquentiellement chacune des instructions de l'opération en contrôlant à chaque fois que :
   - toute opération de lecture/écriture a lieu entre `rd` et `ri`, et que
   - tout déplacement du pointeur d'instruction s'effectue entre `ri` et `rf`.

8. Intentionnellement comme dans le cas de l'attaque dite du « cheval de Troie », ou malencontreusement par une erreur du programme lui-même.
9. Memory Management Unit

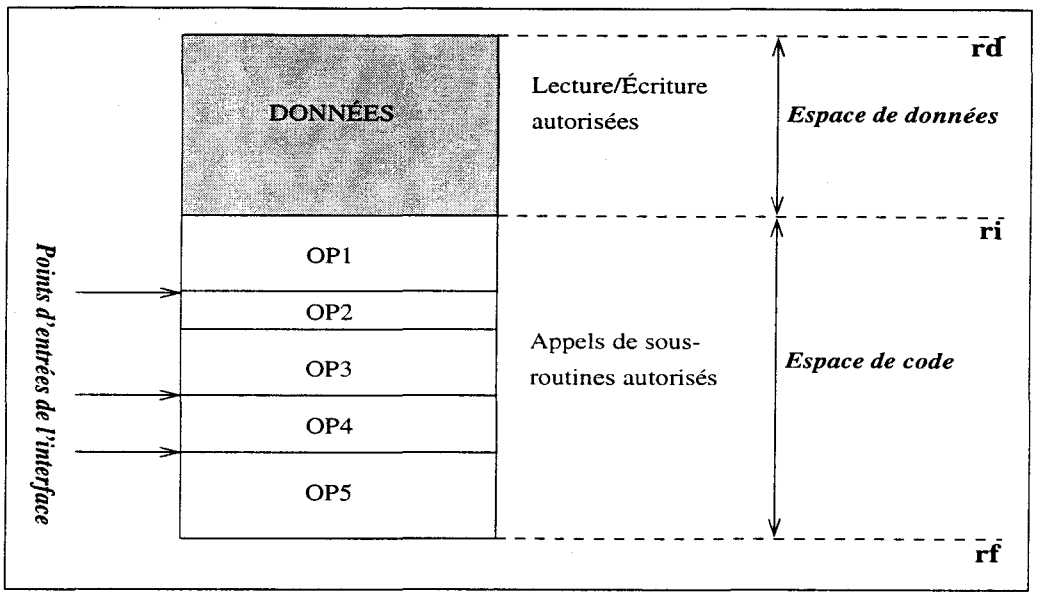

FIG. 4.4 - Espace mémoire d'un service carte

À la fin de l'exécution ou en cas de problème, l'interpréteur génère une erreur et rend la main au système d'exploitation.

La surêté de fonctionnement de l'interpréteur est assuré par le fait qu'il est inscrit en mémoire ROM inaltérable et ne peut donc pas fonctionner autrement que de la façon dont il a été codé – ce qui peut être contrôlé et certifié par des audits extérieurs.

4.2.3 Interfaçage des objets carte dans les applications clientes

Avec une carte classique, l'interface de la carte est définie par le jeu de commandes proposé par son système d'exploitation. Avec une carte générique, l'interface de la carte est la somme des interfaces de tous les services (ou objets carte) qu'elle contient. De plus, pour une même carte, cet ensemble d'interfaces évolue continuellement puisqu'il est possible à tout moment d'ajouter ou de retirer des services. Par conséquent, il devient irréaliste de pouvoir définir à l'avance une API unique (*cf.* § 3.1.2 page 47) d'accès à tous les services carte possible puisque nous ne pouvons connaître l'ensemble des futurs services – donc des futures fonctionnalités qu'il faudrait intégrer à l'API – qui seront un jour disponibles dans les cartes.

Nous traitons ce problème par l'utilisation de descriptions des services carte servant à configurer dynamiquement un serveur générique[10] de cartes. L'approche orientée objet permet de fournir des interfaces bien définies des services cartes et par conséquent apporte une bonne modélisation de l'interface de la carte elle-même (somme de toutes les interfaces des services qu'elle contient). De plus, ces interfaces sont indépendantes de la façon dont sont implémentés les services eux-mêmes et peuvent donc être facilement décrites dans un langage souvent appelé IDL. Deux usages de ces descriptions d'interfaces IDL sont

10. Au sens où ce serveur permet d'accéder à des cartes sans connaître à l'avance les services qu'elles contiennent.

possibles :

- Soit elles sont utilisées avec des couches standards de logiciels d'accès aux cartes (*cf.* TABLE 3.1 page 46) pour produire localement des librairies d'accès aux services qu'elles décrivent. Pour cela, il est nécessaire d'écrire le code qui transcrit les requêtes objet d'appels aux opérations du service carte en requêtes APDU qui pourront être transmises aux couches standards d'accès à la carte.
- Soit elles sont utilisées pour construire dynamiquement des requêtes aux services cartes qui sont prises en charge par un serveur générique réalisant le travail de transcription des requêtes objet en commandes APDU.

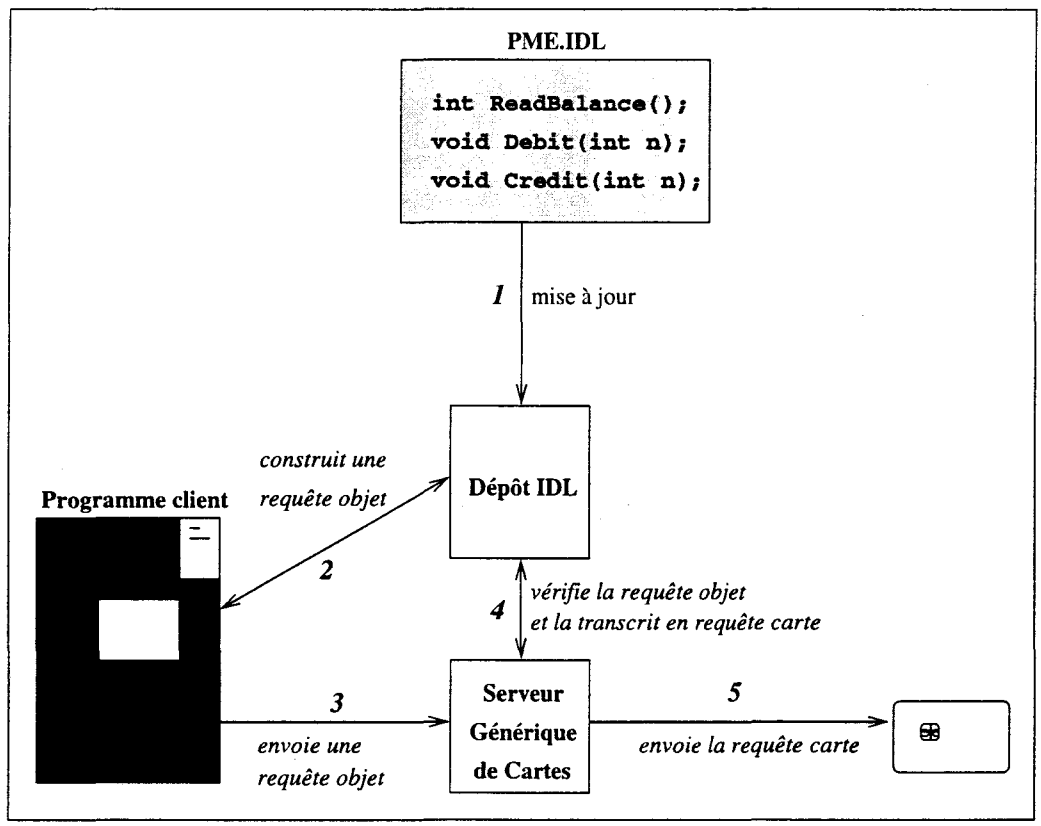

FIG. 4.5 - Utilisation de description IDL de services carte

La seconde solution est plus flexible car elle permet de se passer d'une phase d'écriture de code spécifique aux services. Une application cliente peut ainsi s'autoconfigurer aux services d'une carte par la consultation des descriptions IDL qu'elle reçoit des prestataires de services et stocke dans un dépot d'interfaces. La FIG. 4.5 montre comment une application cliente peut s'abstraire des contraintes d'accès aux cartes en utilisant un serveur générique de carte qui gère les requêtes aux services cartes en consultant le dépot de descriptions IDL (les chiffres qui suivent correspondent à ceux de la figure) :

1 Un prestataire de services de porte-monnaie électronique (par exemple, une banque) charge des objets PME[11] dans des cartes génériques. Il distribue à tous les utilisateurs

11. Porte-Monnaie Électronique

clients potentiels du PME (par exemple, des commerçants) la description IDL de son service. Cette description est stockée dans un dépôt d'interfaces IDL dans le système d'information du client.

2 Le client conçoit une application qui envoie des requêtes objet correspondant aux opérations du service carte décrite en IDL sans se soucier de la façon dont elles seront transportées jusqu'à la carte.

3 Ces requêtes sont toutes adressées à un serveur générique de cartes.

4 Le serveur générique de cartes utilise le dépôt de descriptions IDL pour vérifier l'exactitude de la requête et la transcrit avec ses paramètres en une commande carte APDU demandant l'exécution par l'objet PME de l'opération invoquée.

5 Cette commande est ensuite envoyée à la carte du porteur en utilisant les couches standards d'accès aux cartes.

Cette approche d'intégration de la carte générique dans les systèmes informatiques est basée sur une couche middleware d'interopérabilité (le serveur générique de cartes) offrant une interface commune d'accès aux services carte. Cette interface a pour fonction de prendre en charge les invocations d'opérations sur des services carte en traduisant des requêtes objet en commandes cartes. Cette façon de procéder est une extension de celle utilisée dans le driver ODBC où des requêtes SQL sont traduites en commandes carte (*cf.* § 3.3.3 page 60). On peut cependant imaginer que la traduction d'une requête objet en commandes carte ne puisse pas toujours se faire de manière automatique (requête objet complexe nécessitant l'envoi de plusieurs commandes cartes, différent type de carte génériques, prise en compte de cartes existantes[12], *etc.*) si les cartes sous-jacentes n'offrent pas la même interface que les cartes génériques telles que nous les définissons. Il est alors intéressant de pouvoir paramétrer le serveur générique de cartes par des petits scripts simples qui expriment la manière de traduire une requête objet (correspondant à l'invocation d'une opération d'un service carte décrit dans le dépôt de descriptions IDL) en une suite de commandes carte et ceci selon le type de carte sous-jacente utilisée (*cf.* FIG. 4.6 page ci-contre).

Ces scripts ont pour rôle :

- de faire correspondre les types de données utilisés en IDL en type de données de la carte cible afin de pouvoir réaliser le passage des paramètres et le retour des résultats,
- de donner la transcription des requêtes objet en la séquence de commandes carte (APDU avec les octets `CLA`, `INS`, *etc.*) correponsdante, et
- de pouvoir réaliser d'autres traitements spécifiques à certains types de cartes « exotiques » (protocole non standard, connexion sans contacts, *etc.*).

Ils sont en quelque sorte l'équivalent des différents drivers ODBC qu'il serait nécessaire de réaliser avec des types de cartes différentes. Seulement ici, il ne s'agit plus à chaque fois de récrire complétement une partie de la couche de middleware mais juste de la

12. Cartes qu'on pourraient intégrer dans les « *legacy systems*† ».

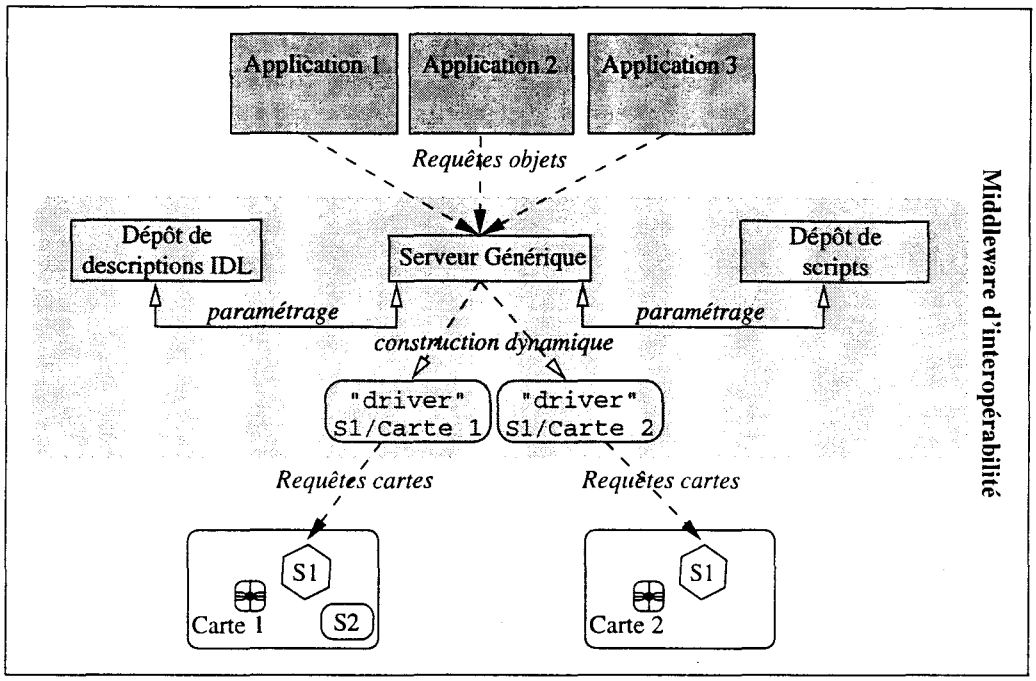

FIG. 4.6 - Paramétrage du serveur générique de cartes par des scripts

paramétrer par des petits programmes simples qui permettront au serveur générique de créer dynamiquement le driver qui convient pour le service auquel l'application accède et pour le type de carte sous-jacente.

4.3 Conclusion : un modèle général

Le modèle général que nous proposons est basé sur une gestion des services carte sous forme d'« objets distribués »[13] dans des serveurs qui sont des cartes à microprocesseur génériques [GPV94]. Ces objets distribués sont intégrés dans un schéma global qui permet :

- d'authentifier et de contrôler les droits d'accès des créateurs et utilisateurs des objets,
- de créer, charger et exécuter de façon sécurisée ces objets dans des cartes génériques, et
- de fournir une couche middleware d'accès générique à ces objets depuis des applications clientes.

Les travaux qui ont plus particulièrement été menés sur ces différentes fonctions seront détaillés au cours des trois prochains chapitres. Ils couvrent un protocole de sécurité (Chapitre 5 page 87), un choix de machine virtuelle pour l'exécution sécurisée des services (Chapitre 6 page 105) et un prototype de serveur générique de cartes dans un environnement orienté objet distribué CORBA (Chapitre 7 page 123).

13. Il s'agit bien sûr ici d'un abus de langage.

Leurs résultats – pour les deux premiers travaux – prennent principalement forme dans une implémentation de système d'exploitation de carte générique que nous appelons CoMbO[14]. Ce système d'exploitation est composé de trois couches logiques (*cf.* FIG. 4.7) :

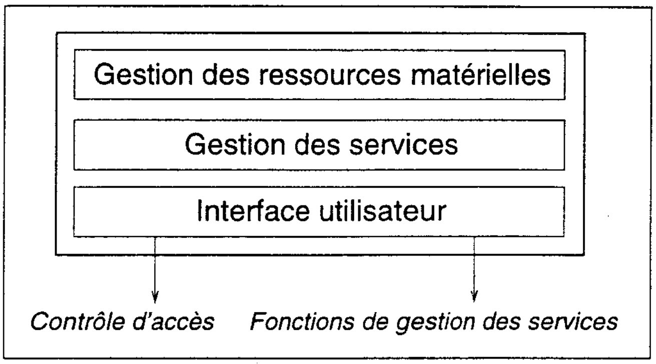

FIG. 4.7 - Architecture du système d'exploitation CoMbO

Gestion des ressources matérielles Cette couche gère les ressources matérielles de la carte, à savoir principalement les entrées/sorties et la gestion dynamique de la mémoire rémanente de la carte réalisé par Biget (Patrick) dans le cadre du projet CASCADE[15]. Cette couche ne sera pas décrite dans le cadre de ce mémoire.

Gestion des services Il s'agit du cœur du système d'exploitation qui fournit la gestion des services, à savoir leur chargement et leur exécution dans l'environnement sécurisé d'une machine virtuelle.

Interface utilisateur Enfin, les fonctions accessibles depuis l'extérieur de la carte qui réalisent principalement deux sortes de tâches : le contrôle d'accès des utilisateurs et l'accès aux fonctions de gestion des services.

Des détails de cette implémentation sont fournis en annexe Chapitre A page 145.

Bibliographie du modèle de système à cartes génériques

[BGV96] Biget (Patrick), George (Patrick) et Vandewalle (Jean-Jacques). – How Smart Cards Can Benefit from Object-Oriented Technologies, pp. 175–194. – In Hartel et al. [HPQ96].

[Car93] Carlier (David). – *Utilisation de concepts orientés objet dans la carte à microprocesseur*, Mémoire de DEA, USTL Université des Sciences et Technologies de Lille, juillet 1993.

[CEP96] CEP Exposium / Cartes. – *Cartes 96: 11th international forum for plastic card technologies and applications, 29-30-31 October 1996, Paris, France*, octobre 1996.

[CGG93] Coulier (Charles), Gordons (Édouard) et Grimonprez (Georges). – *Carte à mémoire et procédé de fonctionnement.* – Brevet d'invention n° 93 14668, Institut National de la Propriété Industrielle, décembre 1993.

14. Ce nom ne veut en soi rien dire si ce n'est « petite formation de musiciens de jazz » en anglais ou «combinaison» d'aliments ou de vêtements en américain. Pour nous, pourquoi pas, combinaison de services dans une carte générique...

15. Chip Architecture for Smart CArds and portable intelligent DEvices

[CQ94] Cordonnier (Vincent) et Quisquater (Jean-Jacques). – *Proceedings of the first smart card research and advanced application conference, October 24-26, 1994, Lille, France*, RD2P Recherche et Développement Dossier Portable, octobre 1994.

[dVdVG95] de Vivo (Marco), de Vivo (Gabriela O.) et Gonzalez (Luis). – A Brief Essay on Capabilities. *ACM SIGPLAN Notices*, vol. 30, n° 7, juillet 1995, pp. 29–36.

[Far96] Farrugia (Augustin J.). – When your card starts to look like a computer. – In CEP Exposium / Cartes [CEP96].

[Gei91] Geib (Jean-Marc). – Note sur les techniques de l'Orienté-Objet appliquées à la carte à microprocesseur. – juin 1991.
Mémo interne au Laboratoire d'Informatique Fondamentale de l'université de Lille I (3 pages).

[GPV94] Gamache (André), Paradinas (Pierre) et Vandewalle (Jean-Jacques). – Worldwide Smart Card Services, pp. 141–148. – In Cordonnier et Quisquater [CQ94].

[Hor96] Horckmans (Emmanuel). – *Un modèle de contrôle d'accès pour les espaces d'information orientés objets et hétérogènes du type Web-Corba-Carte*, Mémoire de DEA, USTL Université des Sciences et Technologies de Lille, juillet 1996.

[HPQ96] Hartel (Pieter H.), Paradinas (Pierre) et Quisquater (Jean-Jacques). – *Proceedings of the 2nd International Conference CARDIS 1996 CWI, Amsterdam, The Netherlands, September 16-18, 1996*, Stichting Mathematisch Centrum, septembre 1996.

[Lam74] Lampson (B.W.). – Protection. *ACM Operating System Review*, vol. 8, n° 1, 1974.

[Mer97] Merle (Philippe). – *CorbaScript - CorbaWeb : propositions pour l'accès à des objets et services distribués*, Thèse de Doctorat, USTL Université des Sciences et Technologies de Lille, janvier 1997.

[Pel95] Peltier (Thierry). – *La Carte Blanche : Un nouveau système d'exploitation pour objets nomades*, Thèse de Doctorat, USTL Université des Sciences et Technologies de Lille, décembre 1995.

[Pfl89] Pfleeger (Charles P.). – *Security in Computing*, Prentice-Hall, 1989.

[PV96] Paradinas (Pierre) et Vandewalle (Jean-Jacques). – Object-oriented approach for smart card operating system and integration into information systems, *In : Proceedings of the IFIP TC11 WG11.2 Workshop on Small System Security, Samos, Greece*, éd. par Eloff (Jan H.P.). IFIP, – pp. 140–147, mai 20, 1996.

[Tan89] Tanenbaum (Andrew). – *Les systèmes d'exploitation, Conception et mise en œuvre*, InterEditions, 1989.

[Van91] Vandewalle (Jean-Jacques). – *Une approche système d'exploitation pour équipement portable*, Mémoire de DEA, USTL Université des Sciences et Technologies de Lille, juin 1991.

CHAPITRE 5

Protocole de sécurité

« *La plus grande pulsion n'est pas la libido mais le besoin de sécurité.* »

Jean Delumeau.

Résumé

Nous étudions dans ce chapitre la problématique de l'authentification et du contrôle d'accès des différents partenaires d'un système à cartes génériques. Après quelques rappels concernant la sécurité et les techniques cryptographiques nous présentons les grandes classes de protocoles d'authentification existants et en retenons deux pour notre système : les protocoles à cryptographie asymétrique avec ou sans serveur de certification. Enfin, nous montrons comment ces deux protocoles pourraient être utilisés pour, dans le premier cas, authentifier les prestataires de services et, dans le second cas, authentifier les clients. L'implémentation réalisée dans la carte $C_0M_0^O$ aboutit à un protocole simplifié du second cas.

Sommaire

Tables

Figures

5.1 Introduction et rappels

5.1.1 Certificats du système à carte générique

Dans ce chapitre nous présentons des protocoles de sécurité qui pourraient être mis en place pour faire du contrôle d'accès dans une carte générique telle que nous l'avons définie dans notre modèle § 4.2.1 page 75. Le problème principal que nous avons soulevé est que nous avons à traiter des certificats qui sont délivrés *off-line*, c'est-à-dire pour nous en dehors de toute transaction avec la carte, et qui doivent être vérifiés *on-line*, c'est-à-dire par la carte au moment où un utilisateur souhaite réaliser une transaction [Van95] (*cf.* FIG. 5.1 page suivante).

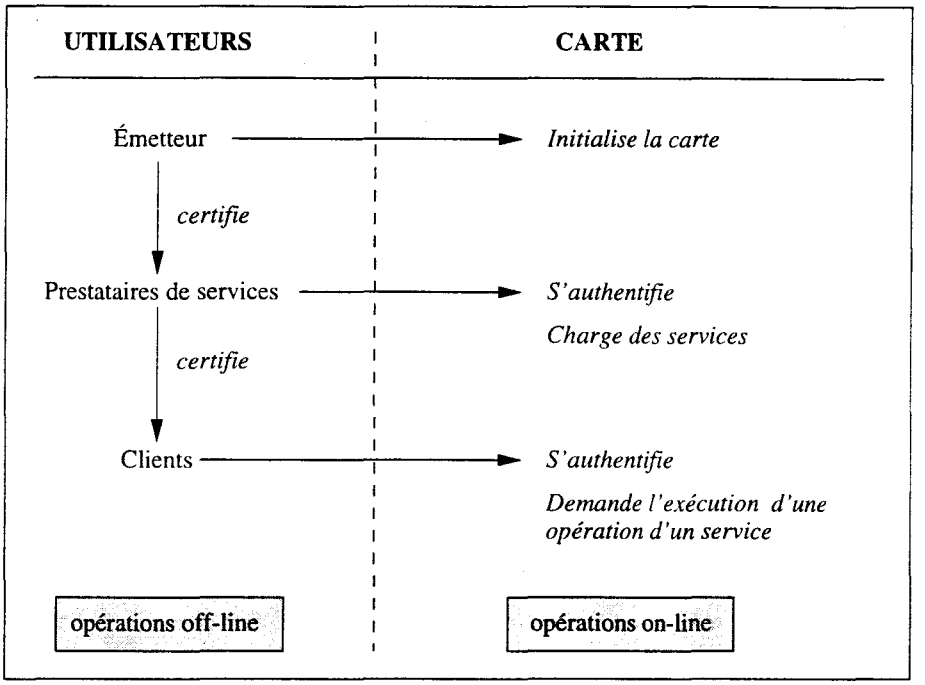

FIG. 5.1 - Schéma de certification

Il faut distinguer dans ce schéma deux types de certificats :

- Celui délivré par l'émetteur à un prestataire de services qui l'accrédite en tant que prestataire de services autorisé à charger des services dans la carte que l'émetteur a émise. Dans ce cas, le certificat contient juste une preuve de la qualité de prestataire de services. Il s'agit donc d'une preuve qui va permettre *d'authentifier* l'utilisateur. Le moyen de vérifier cette preuve est donc associé à une information secrète relative à l'émetteur. Cette information secrète est présente dans la carte au moment de l'authentification du prestataire de services.

- Celui délivré par un prestataire de services à un client qui l'accrédite en tant que client autorisé à exécuter des opérations sur le service qu'il a chargé dans la carte. Dans ce second cas, le certificat contient une preuve de la qualité de client sur un ensemble d'opérations d'un service. Il s'agit donc d'une preuve qui va permettre *d'authentifier* l'utilisateur et de *vérifier* les droits de cet utilisateur à exécuter une opération sur un service identifié. Le moyen de vérifier cette preuve est donc associé à une information secrète relative au prestataire de services et au service qu'il a chargé dans la carte. Cette information secrète est présente dans la carte au moment de l'authentification du client.

Il apparaît clairement après cette brève distinction que la façon dont ces certificats seront traités pour authentifier les utilisateurs sera relativement identique dans les deux cas puisqu'il s'agit d'avoir la certitude à la fin de la période d'authentification que le certificat est valide en utilisant une information secrète stockée préalablement dans la carte par l'autorité qui a émis le certificat. Un certificat aura donc pour caractéristiques de contenir une preuve de l'identité de l'utilisateur qui pourra être vérifiée par la carte et d'être infalsifiable, c'est-à-dire de ne pouvoir ni être généré ni être modifié par tout autre personne que l'autorité émettrice. Différentes techniques d'authentification existent et nous

allons les présenter dans la section suivante avant de discuter celles qui nous semblent les plus appropriées pour notre système à cartes génériques. Auparavant, nous allons dresser quelques rapides rappels sur la sécurité et la cryptographie.

5.1.2 Rappels sur la sécurité et la cryptographie

5.1.2.1 Sécurité et cryptographie

Les opérations de base à réaliser pour assurer la sécurité d'un système sont [RN93]:

- Authentifier l'utilisateur désirant réaliser une opération. Cette opération est, pour nous, réalisée par l'utilisation de protocoles cryptographiques permettant de vérifier la validité d'un certificat.
- Vérifier les droits de cet utilisateur à réaliser cette opération. Cette opération est, pour nous, réalisée par l'utilisation de listes de capacités incluses dans le certificat de l'utilisateur.
- Assurer la confidentialité et l'intégrité des messages échangés entre l'utilisateur et le système (pour nous, la carte). Cette opération a, dans notre cas, été écartée de notre étude.
- Garder des traces des transactions réalisées pour permettre la tenue d'audits. Cette opération a, aussi, été écartée de notre étude.

5.1.2.2 Systèmes cryptographiques

Les mécanismes de base utilisés pour réaliser des certificats sont des systèmes cryptographiques utilisés soit dans une relation de chiffrement, soit dans une relation de signature. Un système cryptographique comprend :

- Une fonction de chiffrement utilisant une clé k notée $m' = \{m\}_k$ où m est le message en clair, k la clé de chiffrement et m' le cryptogramme obtenu en chiffrant le message m avec la clé k.
- Une fonction de déchiffrement utilisant une clé k^{-1} notée $m = \{m'\}_{k^{-1}}$ où m' est le cryptogramme, k^{-1} la clé de déchiffrement et m le message en clair obtenu en déchiffrant le cryptogramme m' avec la clé k^{-1}.

Il existe deux grandes classes de systèmes cryptographiques :

Système symétrique Ce système est aussi appelé système à clé secrète ou système à clé partagée. Dans ce système, les fonctions de chiffrement et de déchiffrement utilisent la même clé, soit $k = k^{-1}$. Cette clé ne doit donc être connue que de l'entité qui chiffre et de l'entité qui déchiffre au risque de compromettre toute la sécurité du système. L'algorithme de ce type le plus connu est le DES [NBS77].

Système asymétrique Ce système est aussi appelé système à clé publique. Dans ce système, la clé de chiffrement k est publique et peut être connue de tout le monde afin que chacun puisse chiffrer un message. Tandis que la clé k^{-1} est gardée secrète

par le récepteur d'un cryptogramme afin d'être le seul à pouvoir le déchiffrer. Ce système est basé sur le fait qu'à partir de k il est « très difficile » (c'est-à-dire impossible sans des moyens énormes) de trouver k^{-1}. L'algorithme de ce type le plus connu est le RSA [RSA78].

Les systèmes cryptographiques sont principalement utilisés soit pour faire du chiffrement, et dans ce cas ils doivent toujours permettre de retrouver le message en clair après un chiffrement suivi d'un déchiffrement, c'est-à-dire vérifier la propriété suivante :

$$\forall m \qquad \{\{m\}_k\}_{k^{-1}} = m$$

Ce qui est le cas du DES et du RSA.

Soit pour faire de la signature et, dans ce cas, ils doivent toujours permettre de prouver qu'un message a bien été envoyé par l'émetteur et lui seul, c'est-à-dire vérifier la relation suivante dans le cas du RSA [RSA78] car il a la propriété d'être commutatif :

$$\forall m \qquad \{\{m\}_{k^{-1}}\}_k = m$$

En effet, seul celui qui connaît la clé secrète k^{-1} a pu « chiffrer » m [1].

5.1.2.3 Protocoles de sécurité

Les systèmes cryptographiques (dans leurs fonctions de chiffrement ou de signature) sont généralement mis en œuvre dans des protocoles de sécurité qui permettent d'assurer au sein d'un système des fonctions d'authentification, de confidentialité, d'intégrité, *etc.* Un protocole est défini de façon générale comme une suite d'étapes, impliquant au moins deux entités, dans le but d'accomplir une tâche. Chaque étape étant constituée soit de calculs effectués par au moins une des entités soit de messages envoyées d'une entité à l'autre. Un protocole se termine soit par un succès pour toutes les entités engagées, dans ce cas la fonction de sécurité attendue (authentification ou confidentialité ou...) a été assurée, soit par un échec pour au moins une des entités engagés, dans ce cas la fonction n'a pas été remplie.

Les caractéristiques d'un protocole sont les suivantes [Sch94] :

- Chaque entité participant à un protocole doit connaître à l'avance toutes les étapes du protocole à suivre.
- Chaque entité participant à un protocole s'engage à le suivre (sauf à vouloir frauder).
- Un protocole ne doit pas être ambigu : chaque étape doit être clairement définie sans aucune possibilité de mauvaise compréhension.
- Un protocole doit être complet : pour toutes les situations possibles une action à tenir doit être spécifiée.

1. Pour éviter les confusions, lorsque nous chiffrons avec k, nous parlons de chiffrement et lorsque nous « chiffrons » avec k^{-1} nous parlons de signature.

L'intérêt d'un protocole est qu'il permet de s'abstraire des processus et mécanismes mis en jeu dans l'accomplissement d'une tâche. La robustesse d'un protocole de sécurité doit pouvoir être évaluée indépendamment de son implémentation. C'est pourquoi nous nous bornerons à ne travailler que sur cet aspect là car il est en dehors de notre propos de nous intéresser aux algorithmes à mettre en œuvre ou aux temps de calculs nécessaires qu'ils impliquent.

5.1.2.4 Exemple et notation

L'exemple suivant nous servira à montrer un premier protocole d'authentification rudimentaire et à introduire les notations que nous utiliserons dans la suite de ce chapitre. Il s'agit de l'exemple d'authentification lors de la procédure de « login » sur une station Unix. L'utilisateur est noté U et le système hôte H. La fonction de chiffrement utilisée pour masquer les mots de passe dans le fichier `/etc/passwd` est notée f, il s'agit d'une fonction à sens unique dont la propriété bien connue est d'être « très difficilement » réversible (étant donné y, le mot de passe masqué, il est « difficile » de trouver p, le mot de passe en clair, tel que $f(p) = y$). Une étape de communication où U envoie un message p à H est notée $U \longrightarrow H$: p, tandis qu'une étape où H réalise des calculs est noté H : ... avec ... décrivant les calculs effectués.

$U \longrightarrow H$	: U
$H \longrightarrow U$	: "passwd: "
$U \longrightarrow H$	: p
H	: calcule $y = f(p)$
	: dans `/etc/passwd` retrouve $(U, f(passwd_U))$
	: si $y = f(passwd_U)$ alors acceptation sinon rejet

FIG. 5.2 - Protocole de login Unix

5.2 Étude des protocoles d'authentification

Nous présentons dans ce paragraphe les grandes classes de protocoles d'authentification basées sur des techniques cryptographiques [WL92]. Il est important de noter que ces exemples servent à illustrer les principes généraux qui forment la base de protocoles réalistes beaucoup plus complexes et raffinés.

5.2.1 Authentification cryptographique

L'authentification consiste à vérifier l'identité proclamée d'une entité. Dans un protocole d'authentification, deux entités, au minimun, sont toujours impliquées :

Le prouveur appelé P qui commence toujours par déclamer son identité (« Je suis P » ou, plus simplement, P) et qui, ensuite, prouve qu'il est bien P.

Le vérifieur appelé V qui est l'entité auprès de laquelle le prouveur doit prouver qu'il est bien celui qu'il prétend être.

La vérification repose sur une caractéristique propre et non falsifiable du prouveur qui peut être :

- une propriété physique (par exemple, une empreinte digitale),
- une possession (par exemple, une carte d'identification), ou
- une connaissance (par exemple, un mot de passe).

Lorsqu'il s'agit d'entités informatiques la preuve repose toujours sur la connaissance d'une information secrète. Il existe deux moyens de prouver la connaissance de cette information secrète :

- en révélant le secret au vérifieur (cas du login Unix vu § 5.1.2.4 page ci-contre) mais, dans ce cas, il faut que le prouveur ait une confiance complète dans le vérifieur et qu'il soit certain que leur communication ne puisse être écoutée, ou
- en montrant au vérifieur qu'on détient le secret sans le révéler grâce à l'exécution d'une opération que le prouveur ne peut effectuer que s'il connaît le secret (par exemple, le calcul d'une signature).

Dans le cas d'entités informatiques qui ne peuvent *a priori* pas se faire confiance, la vérification de la connaissance d'une information secrète repose toujours sur un (ou plusieurs) calcul(s) que seul le prouveur a pu effectuer.

5.2.2 Authentification avec systèmes cryptographiques symétriques

Ces protocoles sont fondés sur l'assertion suivante :

> « *Si le prouveur peut correctement chiffrer un message en utilisant une clé que le vérifieur croît n'être connue que du prouveur avec l'identité proclamée alors cet acte constitue une preuve suffisante de l'identité du prouveur*[2]. »

La puissance de ce principe repose sur la résistance aux attaques de l'algorithme utilisé (impossibilité de chiffrer un message sans connaître la clé et impossiblilité de retrouver la clé en espionnant des messages) et sur la teneur secrète de la clé partagée par les seuls prouveur et vérifieur.

5.2.2.1 Protocoles de base

De cette assertion découle directement le protocole de base présenté FIG. 5.3 page suivante. Sa principale faiblesse est sa vulnérabilité face au rejeu. En effet, un espion

2. « ***If a principal can correctly encrypt a message using a key that the verifier believes is know only to a principal with the claimed identity (outside the verifier), this act constitues sufficient prouf of identity.*** » ***[WL92]***

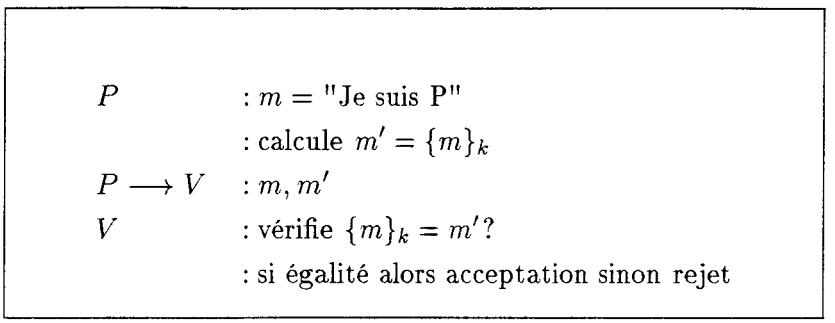

FIG. 5.3 - Protocole d'authentification de base avec cryptographie symétrique

pourrait enregistrer m et m' et plus tard se faire passer pour P auprès de V en lui fournissant ces mêmes messages. Pour pallier à ce genre d'attaque, à chaque nouvelle tentative d'authentification, le vérifieur peut choisir un message différent qu'il met au défi d'être correctement chiffré par le prouveur. Pour que le message soit différent à chaque tentative le vérifieur utilise généralement un nombre aléatoire noté n (FIG. 5.4). Pour augmenter la sécurité du système il peut, à chaque session, soumettre au prouveur plusieurs aléas.

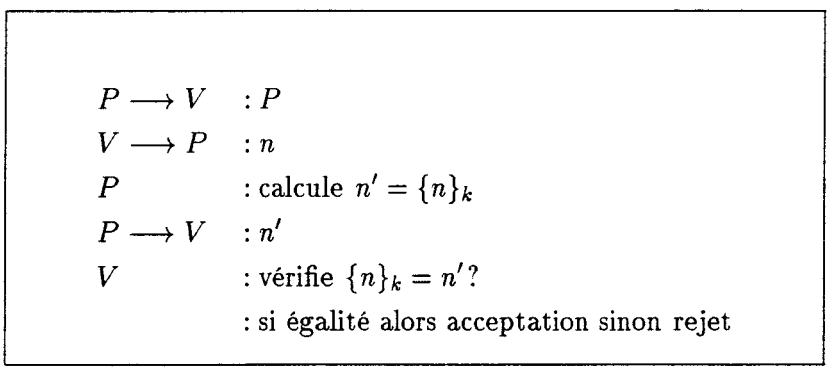

FIG. 5.4 - Protocole d'authentification avec cryptographie symétrique et nombre aléatoire

Néanmoins, ce système reste peu pratique car il nécessite que chaque vérifieur possède une information secrète partagée avec chaque prouveur. Ainsi, si dans le système il y a N entités qui peuvent s'authentifier mutuellement, chaque entité devra conserver de façon sûre un nombre de $N-1$ clés, ce qui fait un nombre total de clés dans le système égal à $N(N-1)/2$. Ce grand nombre de clés pose un problème de distribution à l'initialisation du système et oblige chaque entité à protéger le stockage de ses secrets (une seule entité compromise met en danger l'ensemble du système).

5.2.2.2 Avec un serveur d'authentification

Pour remédier au problème du grand nombre de clés distribuées, un serveur d'authentification centralisé peut être mis en place [NS78]. Ce serveur, que nous appelons A, partage une clé secrète k_x avec chaque entité X du système. Un protocole d'authentification entre P et V se fait donc en utilisant une communication en ligne avec A qui est chiffrée par les clés k_p et k_v (*cf.* FIG. 5.5 page suivante). Ce protocole permet à chaque entité de ne plus conserver qu'une seule clé qu'elle partage avec le serveur. Comme toutes les clés (N au total) résident chez lui, le serveur doit ***absolument*** être de confiance, fortement protégé

et toujours en ligne au moment d'une authentification.

$P \longrightarrow V$: P
$V \longrightarrow P$: n
P : calcule $n' = \{(V, n)\}_{k_p}$
$P \longrightarrow V$: n'
V : calcule $n'' = \{(P, n')\}_{k_v}$
$P \longrightarrow A$: n''
A : retrouve $(P. n') = \{n''\}_{k_v}$
: calcule $m = \{\{n'\}_{k_p}\}_{k_v}$
$A \longrightarrow V$: m
V : vérifie $\{(V, n)\}_{k_v} = m$?
: si égalité alors acceptation sinon rejet

FIG. 5.5 - Protocole d'authentification avec cryptographie symétrique et serveur d'authentification

5.2.3 Authentification avec systèmes cryptographiques asymétriques

Dans les systèmes asymétriques chaque entité P publie sa clé publique k_p et garde secrète sa clé privée ${k_p}^{-1}$. Ainsi, seul P peut signer des messages m avec sa clé privée : $\{m\}_{{k_p}^{-1}}$, et toutes les entités connaissant la clé publique de P peuvent vérifier la validité de cette signature (système commutatif). Les protocoles d'authentification avec systèmes cryptographiques asymétriques sont donc fondés sur l'assertion suivante :

> « *Si le prouveur peur correctement signer un message en utilisant la clé secrète de l'entité dont il proclame l'identité, alors cet acte constitue une preuve suffisante de l'identité du prouveur*[3]. »

5.2.3.1 Protocole de base

Le protocole de base est présenté FIG. 5.6 page suivante. Il dépend du fait que $\{n\}_{{k_p}^{-1}}$ ne peut être produit sans la connaissance de ${k_p}^{-1}$ et que chaque vérifieur V détient la clé k_p publiée par P. Chaque entité dans le système doit donc posséder sa propre paire de clés publique et privée ainsi que de toutes les clés publiques des autres entités.

5.2.3.2 Avec un serveur de certification

Dans le protocole précédent la gestion des clés publiques peut aussi amener des problèmes de distribution et de stockage car elles doivent toutes être distribuées à l'ensemble

3. « *If a principal can correctly sign a message using the private key of the claimed identity, this act constitues sufficient prouf of identity.* » *[WL92]*

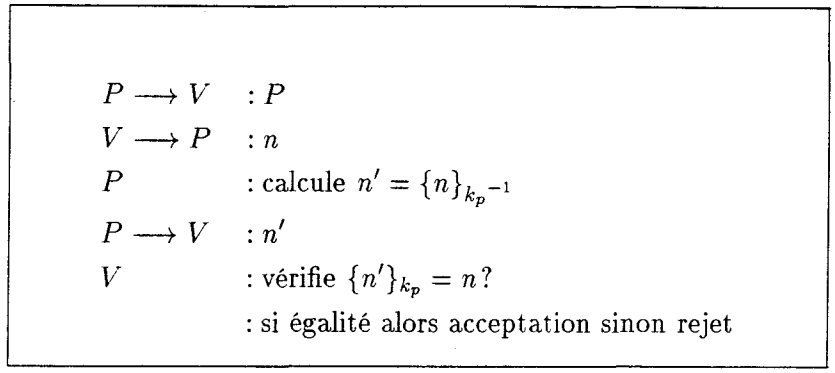

FIG. 5.6 - Protocole d'authentification de base avec cryptographie asymétrique

des entités. Une solution peut être là aussi d'employer un serveur pour stocker dans une BD les clés publiques de toutes les entités du système (*cf.* FIG. 5.7).

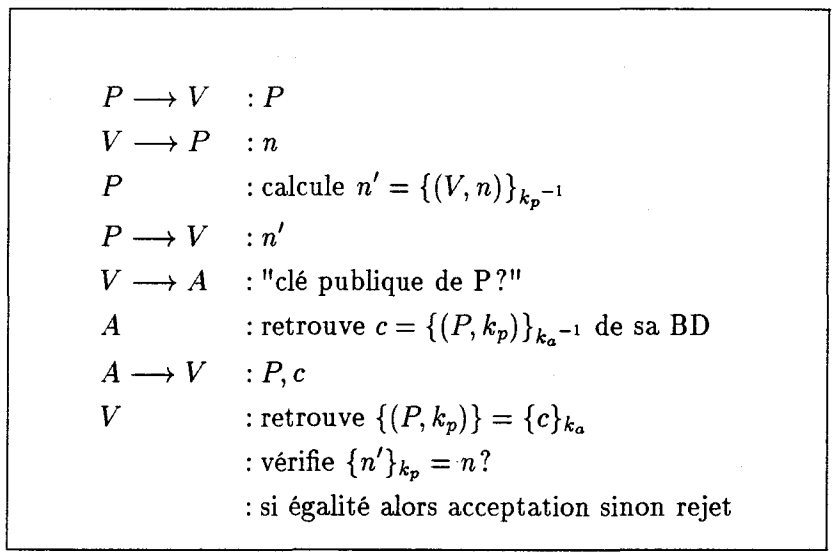

FIG. 5.7 - Protocole d'authentification avec cryptographie asymétrique et serveur de certification

Ainsi chaque entité n'a besoin que de posséder sa propre paire de clés asymétriques et de connaître la clé publique du serveur. Ce serveur est appelé serveur de certification car il permet de certifier les clés publiques de toutes les entités du système sans pour autant pouvoir réaliser une authentification à leur place (il n'a pas la connaissance des clés privées des entités). Ce serveur est beaucoup plus intéressant que le serveur d'authentification car il n'a pas besoin d'être aussi sécurisé (il ne contient que des clés publiques) et ses certificats peuvent être délivrés à l'avance aux entités, ce qui évite qu'il soit forcément en ligne au moment d'une authentification.

5.3 Protocoles d'authentification pour systèmes à cartes génériques

5.3.1 Choix du protocole

Malgré la simplicité des protocoles décrits dans le paragraphe précédent[4], ils représentent les grandes classes de techniques utilisées pour faire de l'authentification dans les systèmes d'information. Dans notre système à cartes génériques deux types d'authentification sont à réaliser : les prestataires de services par l'émetteur et les clients par des prestataires de services. Ces deux types d'authentification sont réalisés non pas directement entre les entités mais par la carte générique qui ne dispose pas *à l'avance* de toutes les clés qui permettraient d'authentifier toutes les entités possibles (communauté des utilisateurs et des services non close). Notre protocole d'authentification sera donc à base de certificats cryptographiques à clés publiques qui permettent de réduire le nombre de clés secrètes distribuées dans le système et de dissocier la délivrance des certificats de leur utilisation au moment de l'authentification. Les deux protocoles présentés ci-après assurent les fonctions suivantes :

- description des phases de certification des prestataires de services et des clients,
- réalisation de doubles authentifications (carte-prestataire de services et carte-client), et
- génération d'une clé unique à la session en cours (clé de session notée k et supposée ici clé partagée) et connue seulement de la carte et de l'entité authentifiée (prestataire de services ou client).

Entités	Identité	Clés	Certificats	Aléas
Carte	C	k_c, k_c^{-1}	C_c	n_c
Émetteur	E	k_e, k_e^{-1}	-	-
Prestataire de services	PS	k_{ps}, k_{ps}^{-1}	C_{ps}	n_{ps}
Clients	CL	k_{cl}, k_{cl}^{-1}	$C_{cl/ps}$	n_{cl}

TABLE 5.1 - Notations utilisées

5.3.2 Authentification des prestataires de services avec certification en ligne

L'authentification des prestataires de services doit être extrêmement contrôlée car elle autorise ces derniers à pouvoir charger des services dans la carte. Pour cela nous utilisons un serveur de certification (l'émetteur de la carte E) qui maintient la BD des cartes et des prestataires de services certifiés. Cette base de données contient l'identité de chaque carte C émise par E ainsi que sa clé publique k_c qui reste *valide* tant que l'émetteur considère la carte comme correcte (*cf.* FIG. 5.8 page suivante).

4. Des véritables protocoles sont généralement beaucoup plus complexes car ils intègrent un schéma de distribution de clés, mêlent des techniques cryptographiques symétriques et asymétriques et peuvent réaliser des fonctions d'authentification, de confidentialité, d'intégrité, *etc.*

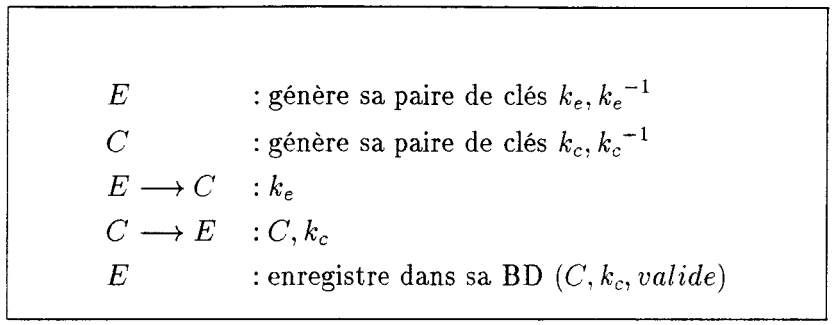

FIG. 5.8 - Personnalisation de la carte

La base de données contient aussi les identités et les clés publiques de tous les prestataires de services PS que l'émetteur certifie comme valide (*cf.* FIG. 5.9). L'intérêt de cette base de données est de pouvoir être consultée en ligne au moment de l'authentification pour vérifier qu'une carte ou qu'un certificat de prestataire de services est toujours valide : elle permet donc de mettre en œuvre un mécanisme de *répudiation* des certificats mais oblige à chaque authentification à avoir une liaison en ligne avec le serveur.

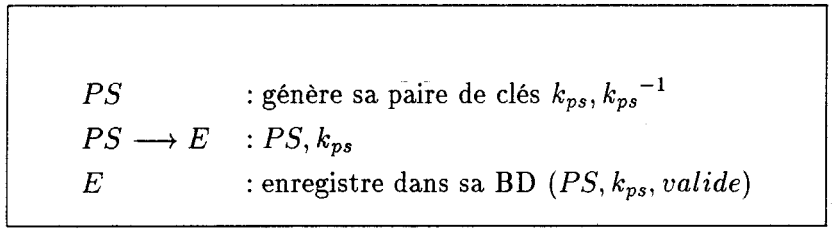

FIG. 5.9 - Certification d'un prestataire de services

L'authentification (*cf.* FIG. 5.10 page ci-contre) se fonde sur la fourniture par l'émetteur des certificats carte et prestataire de services qui sont ensuite utilisés pour effectuer :

- la vérification par la carte que le prestataire de services est certifié par l'émetteur en déchiffrant C_{ps} avec k_e pour retrouver n_c qui assure la « fraicheur » du certificat (que le prestataire de services n'a pu retrouver que s'il possède k_{ps}^{-1}), et
- la vérification par le prestataire de services que la carte est certifiée par l'émetteur en lui demandant de chiffrer n_{ps} avec la clé de session k (que la carte n'a pu retrouver que si elle possède k_c^{-1}).

Cette authentification est lourde à mettre en place car elle nécessite une connection en ligne avec le serveur de certification (l'émetteur) mais elle permet d'assurer, au moment de l'authentification, la validité des certificats.

5.3.3 Authentification des clients avec certification déconnectée et contrôle d'accès

L'authentification des clients est de la responsabilité de chaque prestataire de services. Chaque prestataire de services délivre un certificat à ses clients qui sera utilisé pour authentifier les clients au moment où ceux-ci demandent l'exécution d'un service carte chargé par le prestataire. Ici, nous proposons un mode d'authentification où les certificats des clients

$PS \longrightarrow C$	: "reset"
$C \longrightarrow PS$	: C, n_c
$PS \longrightarrow E$	: PS, C, n_c
$E \longrightarrow PS$	: n_e
PS	: calcule $n'_e = \{n_e\}_{k_{ps}{}^{-1}}$
$PS \longrightarrow E$	: n'_e
E	: authentifie PS en vérifiant $\{n'_e\}_{k_{ps}} = n_e$?
	: génère la clé de session k
	: vérifie la validité de C et PS en consultant sa BD
	: calcule $C_c = \{C, k_c, k\}_{k_e{}^{-1}}$ et $C_{ps} = \{n_c, k, PS\}_{k_e{}^{-1}}$
	: calcule $m_e = \{C_c, C_{ps}\}_{k_{ps}}$
$E \longrightarrow PS$	: m_e
PS	: retrouve $(C_c, C_{ps}) = \{m_e\}_{k_{ps}{}^{-1}}$
	: retrouve $(C, k_c, k) = \{C_c\}_{k_e}$
	: calcule $m_{ps} = \{C_{ps}, n_{ps}\}_{k_c}$
$PS \longrightarrow C$	: m_{ps}
C	: retrouve $(C_{ps}, n_{ps}) = \{m_{ps}\}_{k_c{}^{-1}}$
	: retrouve $(n_c, k, PS) = \{C_{ps}\}_{k_e}$
	: authentifie PS en retrouvant n_c ?
	: calcule $n'_{ps} = \{n_{ps}\}_k$
$C \longrightarrow PS$	: n'_{ps}
PS	: authentifie C en vérifiant $n_{ps} = \{n'_{ps}\}_k$?
$PS \longrightarrow C$	: charge un (des) service(s)...$\{n_{ps}\}_k = n'_{ps}$?

FIG. 5.10 - Authentification du prestataire de services

sont vérifiés par la carte en utilisant la clé publique k_{ps} du prestataire de services qui a été chargée dans la carte en même temps que le service S_{ps} (*cf.* FIG. 5.11 page suivante).

Ici, le schéma d'authentification n'utilise pas de serveur d'authentification en ligne, c'est-à-dire que le certificat d'utilisation du service S_{ps} est délivré *une fois pour toutes* au client CL par le prestataire de services PS (*cf.* FIG. 5.12 page suivante). Ce certificat est composé des informations suivantes :

- identification du client CL,
- clé publique du client k_{cl},
- identification du prestataire de services k_{ps},
- identification du service S, et
- masque d'utilisation du service S_{ps} contenant la liste des opérations du service (points

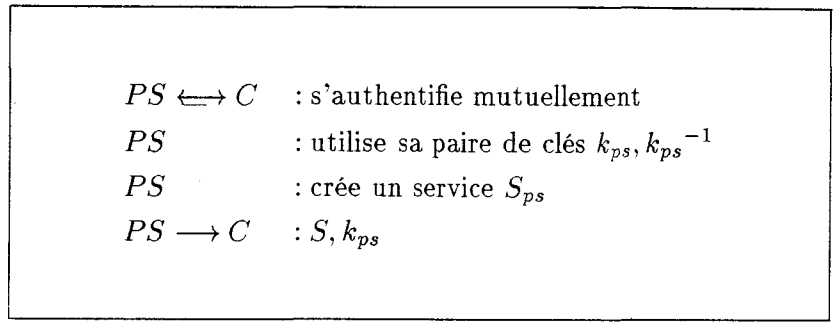

FIG. 5.11 - Chargement d'un service dans la carte

d'entrée) que le client peut utiliser (dont il est autorisé à demander l'exécution par la carte) : $Mask_{S_{cl/ps}}$.

Le certificat étant chiffré avec la clé secrète ${k_{ps}}^{-1}$ du prestataire de services, seul ce dernier peut le générer. Si le prestataire de services a chargé plusieurs services $S_1, S_2, \ldots, S_n$ dans la carte, le certificat qu'il délivre à un client contient l'ensemble des « masques » pour les services $S_i, S_j, \ldots, S_m \quad (m < n)$ que le client est autorisé à utiliser, soit :

$$(Mask_{S_{i\,cl/ps}}, Mask_{S_{j\,cl/ps}}, \ldots, Mask_{S_{m\,cl/ps}})$$

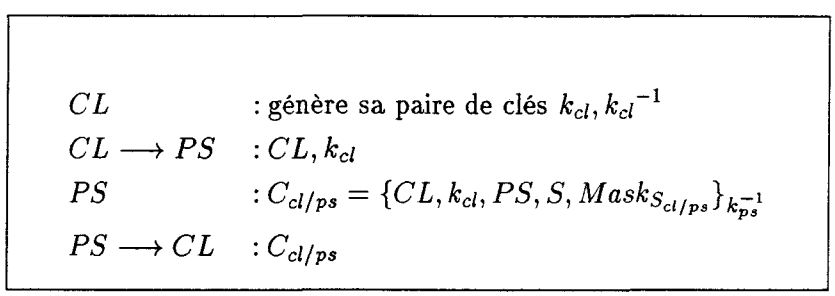

FIG. 5.12 - Certification d'un client

Lorsque le client CL demande à la carte l'exécution d'une opération *op* d'un service S, la carte utilise la clé publique k_{ps} du prestataire de services PS qui a chargé le service dans la carte pour déchiffrer le certificat $C_{cl/ps}$. En retrouvant la clé publique k_{cl} du client la carte l'authentifie en lui demandant de chiffrer un aléa n avec sa clé secrète ${k_{cl}}^{-1}$ qu'il est le seul à posséder. Ensuite, elle contrôle les droits d'accès du client sur le service S en vérifiant que l'opération demandée *op* est incluse dans le masque $Mask_{S_{cl/ps}}$ qu'elle a aussi retrouvé dans le certificat. L'authentification de la carte se fait au moyen d'un certificat C_c chargé dans la carte par l'émetteur au moment de la personnalisation. Le client peut déchiffrer ce certificat au moyen de la clé publique de l'émetteur k_e dont il a la connaissance (*cf.* FIG. 5.13 page suivante).

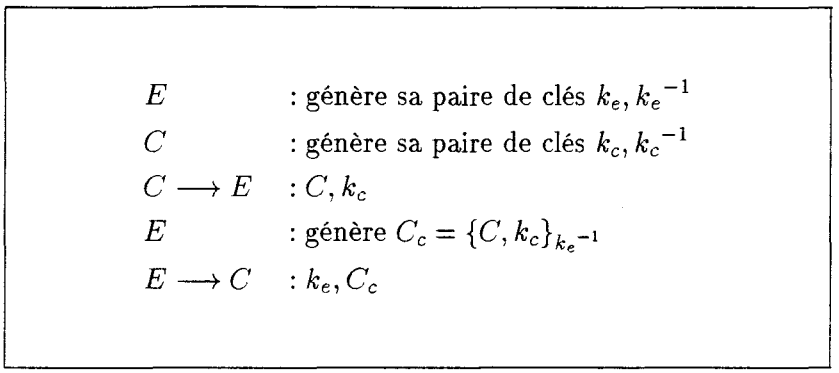

FIG. 5.13 - Personnalisation et certification de la carte

Le protocole présenté FIG. 5.14 réalise donc une double authentification carte $\longleftrightarrow$ client, effectue un contrôle des droits d'accès du client sur les services d'un prestataire de services (au moyen d'une capacité listant les points d'entrées de chaque service que le client peut utiliser) et aboutit à la génération d'une clé de session k générée aléatoirement par la carte.

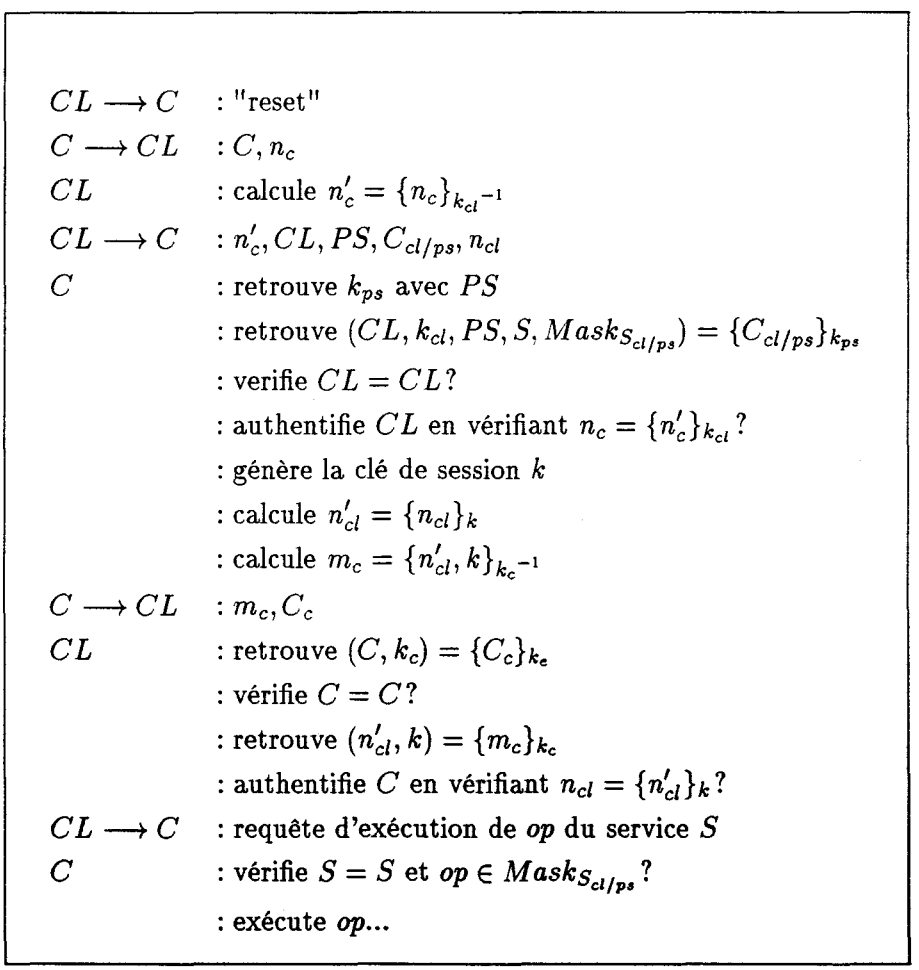

FIG. 5.14 - Authentification et contrôle d'accès du client

5.4 Conclusion

Les schémas de certification, d'authentification et de contrôle d'accès présentés ci-dessus reposent sur les assertions suivantes :

- Chaque partenaire possède une identification unique ainsi qu'une paire de clés publique et privée.
- Il existe une autorité de certification des cartes et des prestataires de services que nous avons confondu par commodité avec l'émetteur de la carte. Cependant, cette autorité pourrait être une hiérarchie d'autorités se certifiant en cascade les unes les autres comme dans les schémas de certification X509 [ISO94].
- Les clients de services sont certifiés par les prestataires de services eux-mêmes.
- Le contrôle d'accès des clients s'effectue par des capacités contenant la liste des opérations du service que le client peut exécuter. Chaque service proposé par un prestataire a une identification unique.

Deux protocoles d'authentification ont été proposés (*cf.* FIG. 5.10 page 99 et *cf.* FIG. 5.14 page précédente) : le premier par l'utilisation d'un serveur en ligne permet de vérifier «en direct» la validité des certificats, le second n'utilisant pas de serveur ne permet pas la répudiation des certificats[5]. Dans notre prototype de carte C_{OMO}, nous avons retenu les assertions listées ci-dessus et avons choisi d'implémenter, dans un premier temps, des schémas d'authentification n'utilisant pas de serveur en ligne[6], et n'aboutissant pas à la génération de clés de session. Ceci nous permet d'aboutir à la réalisation de protocoles d'authentification des prestataires de services et des clients en trois échanges seulement qui sont implémentés dans la carte par deux commandes décrites Chapitre A page 145.

Bibliographie du protocole de sécurité

[ISO94] ISO International Standards Organization. – *Information Technologie - Open Systems Interconnection - The Directory: Authentication Framework* – juin 1994.

[NBS77] NBS National Bureau of Standards. – *Data Encryption Standard, DES* – 1977. *Federal Information Processing Standards FIPS-46.*

[NS78] Needham (Roger M.) et Schroeder (Michael D.). – Using encryption for authentication in large networks of computers. *Communications of the Association for Computing Machinery*, vol. 21, n° 12, décembre 1978, pp. 993–999.
Existe aussi sous la forme d'un rapport de recherche Xerox Research Report, CSL-78-4, PARC.

5. L'utilisation d' « estampilles » (ou timestamps) qui permet de donner une validité temporelle à un certificat (par exemple, jusqu'au 31 décembre 1999) ne peut être utilisée dans notre cas puisque la carte qui vérifie ces certificats ne dispose pas d'horloge interne. Il est donc impossible à une carte de vérifier si une date limite incluse dans un certificat est toujours valide... Par contre, le stockage de ces certificats dans d'autres cartes à microprocesseur comme les cartes des professionnels de santé en France permettrait d'augmenter leur sécurité.

6. la répudiation d'un certificat est donc impossible, mais nous pensons dans ce cas qu'il s'agit plus d'une question de choix d'architecture globale que de fonctions à implémenter dans la carte.

[RN93] Rousseau (L.) et Natkin (S.). – *Etude de l'architecture d'un systeme sécurisé réparti à objets.* – Rapport de recherche n° CEDRIC-96-07, Centre d'Études et De Recherche en Informatique du CNAM, 1993.
ftp://ftp.cnam.fr/pub/CNAM/cedric/tech_reports/RRC-96-07.ps.gz.

[RSA78] Rivest (R.L.), Shamir (A.) et Adleman (L.). – A Method for Obtaining Digital Signatures and Public-Key Cryptosystem. *Communications of the Association for Computing Machinery*, vol. 21, n° 2, février 1978, pp. 120–126.

[Sch94] Schneier (Bruce). – *Applied Cryptography – Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, 1994.

[Van95] Vandewalle (Jean-Jacques). – Loading several services into multi-purpose integrated circuit cards, *In: Proceedings of European Research Seminar on Advances in Distributed Systems, ERSADS'95, L'Alpe d'Huez, France, April 3–7, 1995.* IMAG & INRIA, – pp. 317–322, avril 1995.

[WL92] Woo (Thomas Y.C.) et Lam (Simon S.). – Authentication for Distributed Systems. *IEEE Computer Review*, janvier 1992, pp. 39–52.

CHAPITRE 6

Machine virtuelle C_oMb^O

« Je meurs d'impatience de voir naître la technologie : lorsque je posséderai la connaissance des ordinateurs, je serai l'être suprême. »

Citation tirée du film *Bandits* de Terry Gilliam.

Résumé

Ce chapitre présente la machine virtuelle C_oMb^O qui permet l'exécution sécurisée des services chargés dans notre prototype de carte générique. Elle est basée sur une architecture de machine à pile « à la Forth», dispose d'un jeu d'instructions définissant le langage C_oMb^O, et d'un interpréteur exécutant des « codes filés» (ou threaded codes) selon la technique décrite dans la littérature comme étant du type « token-token ».

Sommaire

Tables

Figures

6.1 Choix de la machine virtuelle

6.1.1 Pourquoi une machine virtuelle?

La machine virtuelle C_oMb^O constitue le noyau du système d'exploitation de notre carte générique. Elle permet de sécuriser l'exécution du code des opérations des services chargés dans la carte (*cf.* § 4.2.2 page 77) par l'utilisation d'un interpréteur. L'interprétation est un processus qui charge, vérifie et exécute les instructions d'un programme. Les avantages des interpréteurs sont les suivants :

Sécurité L'interpréteur de la machine virtuelle permet de contrôler *à la volée* les adresses mémoire accédées par les instructions et donc de garantir que l'exécution d'une opération d'un service ne déborde pas de la zone mémoire réservée aux données du service.

Portabilité L'interpréteur de la machine virtuelle permet d'offrir un langage identique pour l'écriture des codes des services carte quels que soient les microprocesseurs de cartes sur lesquels ils s'exécutent. Un changement de microprocesseur cible requiert uniquement la réécriture de l'interpréteur.

Facilité Il est plus facile d'écrire un code en langage interprété – même de faible niveau – que d'écrire un code en langage machine.

Compacité Chaque instruction de l'interpréteur se décompose en un ensemble d'instructions machine; ainsi les programmes chargés dans la carte, écrits dans le langage de l'interpréteur et exécutés sur la machine virtuelle sont plus compacts que les mêmes programmes écrits en assembleur.

Cependant, l'utilisation d'un interpréteur dans la carte présente deux inconvénients majeurs [Noy88]. D'une part, l'interprétation d'un programme prend en général plus de temps que l'exécution du même programme en langage machine. D'autre part, il faut tenir compte du coût en place mémoire (ROM pour le code l'interpréteur et RAM pour ses données internes) que prend l'interpréteur.

On peut répondre à ces problèmes en argant que le coût supplémentaire en temps d'exécution est la contre-partie inévitable au gain en compacité du code qui est stocké en mémoire EEPROM, mais surtout permet d'offrir un support d'exécution sécurisé des programmes chargés dans la carte grâce au contrôle d'adressage réalisé par l'interpréteur [CGG93]. Ces deux arguments ont un poids considérable pour des cartes génériques pour lesquelles il est très important de gérer au plus juste la mémoire EEPROM servant à stocker les services, et pour lesquelles la sécurité du code des services qu'elles exécutent est, comme nous l'avons vu, une condition d'utilisation *sine qua non*. De plus, la taille de la mémoire ROM des cartes étant assez importante (jusqu'à 20 KO), la place occupée par l'interpréteur dans le masque n'est pas *a priori* une contrainte très forte. Il faut cependant que l'interpréteur reste assez petit pour ne pas consommer trop de mémoire vive qui elle reste encore limitée (512 octets maximum). Sur ce dernier point, le choix de la machine cible et de l'interpréteur, en fonction de leur complexité, du langage, et des stratégies d'implémentation, peut considérablement faire baisser la taille de mémoire RAM nécessaire à l'interpréteur et, de plus, faire augmenter la rapidité d'exécution des programmes. Ces critères sont discutés dans la section suivante.

6.1.2 Choix de l'interpréteur

Le choix de l'interpréteur et le choix de la machine virtuelle sur laquelle il s'exécute sont inextricablement liés. Les critères retenus pour augmenter la rapidité de l'interpréteur et diminuer sa taille doivent conduire à une architecture de machine virtuelle définie pour supporter au mieux le mode d'exécution du langage. Traditionnellement, la rapidité d'un interpréteur peut significativement être augmentée en le faisant fonctionner sur un langage *codé* de façon à diminuer le temps passé par l'interpréteur à, soit faire de l'analyse syntaxique ou lexicale, soit faire des calculs internes. Pour cela, les techniques suivantes sont employées [Noy88] :

1. simplifier le traitement des nombres en les traitant sous forme binaire et sans tenir compte de conventions multiples ou de formats multiples,
2. mettre les expressions sous forme polonaise post-fixée qui simplifie leur évaluation et a de plus l'avantage d'être très peu encombrante,
3. remplacer les identificateurs – noms des variables, des champs, des procédures, et même des mots-clés du langage – par un numéro pouvant servir d'index dans un

tableau permettant de retrouver les caractéristiques courantes de chaque identificateur,

4. supprimer le travail d'analyse syntaxique de l'interpréteur en la faisant faire dans un phase préalable à celle de l'exécution. et

5. supprimer le travail de calcul d'adresse (branchements, instructions d'itération, appels de sous-routines) de l'interpréteur en la faisant faire dans une phase préalable à celle de l'exécution.

De façon évidente, pour diminuer la taille de l'interpréteur il est intéressant de définir une machine virtuelle peu complexe et de réduire les primitives de base du langage. Il y a ici un compromis à trouver entre la complexité de la machine virtuelle, la puissance d'expression du langage et les contraintes en place mémoire et en temps d'exécution dans la carte. En effet, plus la machine virtuelle est complexe et plus le langage est puissant, plus l'interpréteur est difficile à écrire, occupe plus de place et est lent. Par contre, les programmes sont plus faciles à écrire et plus compacts en mémoire. A contrario, moins la machine virtuelle est complexe et moins le langage est puissant, plus l'interpréteur est facile à écrire, occupe moins de place et est rapide. Cependant, les programmes sont plus difficiles à écrire et occupent plus de place en mémoire.

6.1.3 Les machines à pile et Forth

L'étude de différents types de machines virtuelles nous a amené à nous intéresser aux machines à pile et au langage Forth qui est souvent utilisé pour programmer ces machines. En effet, cette architecture permet – sous certaines implémentations – de répondre au mieux aux cinq critères définis en 6.1.2.

6.1.3.1 Les machines à pile

Les machines à pile [Koo89] sont particulièrement intéressantes par rapport aux machines conventionnelles (généralement des machines à base de registres) parce qu'elles ont l'avantage d'être simples (donc facile à implémenter), génériques et rapides.

La généricité est fournie par le fait que les piles permettent d'implémenter facilement les opérations suivantes :

- évaluation des expressions sous forme polonaise post-fixée,
- stockage des adresses de retour lors d'appels de sous-routines,
- mémoire de stockage temporaire, et
- passage de paramètres entre sous-routines.

La compacité et la rapidité d'exécution des programmes nous sont fournies par une implémentation efficace de l'interpréteur qui s'exécute sur cette machine virtuelle et que nous avons dérivée de l'étude des implémentations des interpréteurs Forth.

6.1.3.2 Forth

Une importante littérature est disponible sur Forth. Sur l'historique du langage et comme introduction à Forth on lira avec intérêt [RCM93] et [Jr.93]. Comme première approche à Forth je ne saurais trop conseiller [Bro87]. Sur la philosophie et la méthodologie de développement de programmes Forth on pourra se reporter à [Bro84] et [Woe92]. Pour se donner des idées sur une implémentation de Forth pour microprocesseurs embarqués voir [Pay90]. Pour des articles sur la recherche autour de Forth et ses applications consulter les publications du Special Interest Group on Forth de l'ACM [ACM92] et [Koo91]. À l'instar du langage C, il existe une norme ANSI pour Forth [ANS93]. Enfin, est donnée en annexe, Chapitre B page 159, une présentation de Forth tirée de [Jr.93].

Bien qu'à l'origine conçu comme un environnement de développement complet pour des petits composants embarqués Forth est essentiellement un interpréteur fondé sur une machine abstraite à pile. Ce langage est dû à Moore (Charles) et date de 1970. Forth a été implémenté sur un grand nombre de microprocesseurs parce qu'il a la propriété d'être rapide et très petit. Il occupe généralement moins de 8 KO et peut même être codé sous moins de 2 KO. De plus, il est très efficace puisque l'exécution d'un programme Forth peut être, au mieux, jusqu'à 1,1 fois plus lente que celle d'un programme écrit en langage machine et que les codes Forth occupent couramment moins d'espace mémoire que des codes en langage machine.

L'interpréteur Forth est chargé de l'exécution des programmes Forth. Son rôle est donc de faire pointer le compteur d'instructions sur la suite de mots à exécuter. Ces mots sont rangés dans un dictionnaire qui contient, soit des primitives (mots de bases de l'interpréteur), soit des secondaires (mots constitués de mots de bases et/ou d'autres secondaires). L'implémentation que nous avons choisie de l'interpréteur nous apporte compacité du code et rapidité d'exécution en prenant une structure de dictionnaire où tous les adressages sont directs [Hon92].

6.2 La machine virtuelle C_0Mb^0

6.2.1 Architecture de la machine virtuelle C_0Mb^0

La machine virtuelle C_0Mb^0 est le support d'exécution du code des opérations des services chargés dans la carte. Le code des opérations est décrit dans le langage C_0Mb^0 comme une suite de *tokens* (*cf.* § 6.3 page 112). Le moteur C_0Mb^0 est une machine abstraite à pile qui est construite autour de cinq composants logiques (*cf.* FIG. 6.1 page suivante) :

Pointeur d'instructions Il pointe sur le token courant à exécuter. Il est noté IP pour *Instruction Pointer.*

La pile de données Elle sert à l'évaluation des expressions, au stockage des variables locales et au passage des arguments entre fonctions. Elle est notée DS pour *Data Stack.* Le sommet de pile est noté TOS pour *Top Of Stack.*

La pile de retour Elle sert à stocker temporairement les adresses de retour lors des appels de sous-routines. Elle sert aussi à stocker les valeurs des compteurs de boucles. Elle est notée RS pour *Return Stack.*

L'espace de données Il sert à stocker les données rémanentes (dans la mémoire non volatile de la carte) des services carte. Ces données sont directement manipulées par les opérations de lecture/écriture du langage $C_{o}^{\flat}\mathcal{O}$. Il est noté DS pour *Data Space*.

L'espace de code Il sert à stocker (dans la mémoire non volatile de la carte) les codes des opérations des services carte. Il est la zone d'adressage du pointeur d'instructions. Il est noté CS pour *Code Space*.

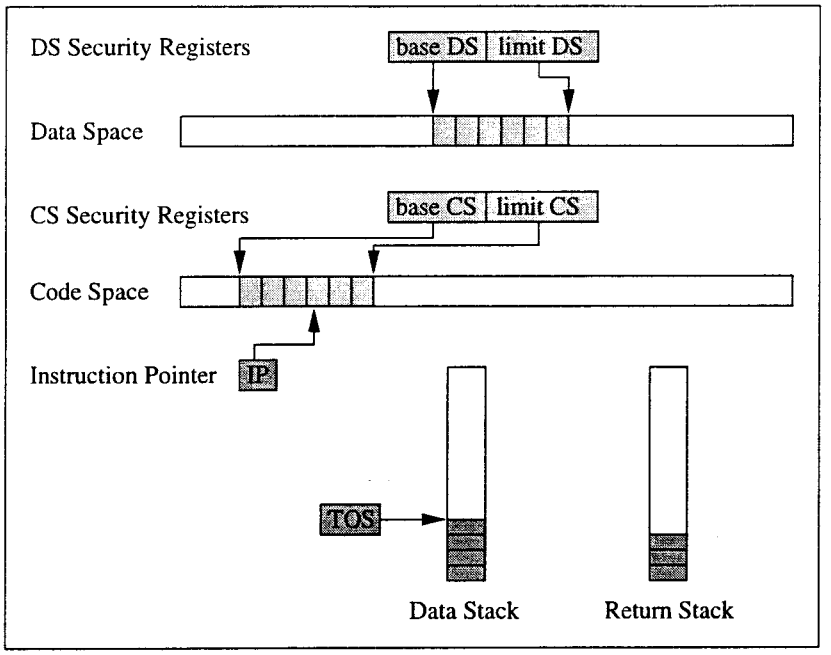

FIG. 6.1 - Éléments de la machine virtuelle $C_{o}^{\flat}\mathcal{O}$

La machine virtuelle $C_{o}^{\flat}\mathcal{O}$ implémente aussi quatre registres de sécurité :

- Deux registres de sécurité pour l'espace de données (*Base DS* et *Limit DS*) qui limitent l'espace de données accessible lors de l'exécution d'un mot du langage $C_{o}^{\flat}\mathcal{O}$.
- Deux registres de sécurité pour l'espace de code (*Base CS* et *Limit CS*) qui limitent l'espace de code accessible lors de l'exécution d'un mot du langage $C_{o}^{\flat}\mathcal{O}$.

La FIG. 6.2 page ci-contre est une représentation sous forme de *diagramme bloc* de la machine virtuelle $C_{o}^{\flat}\mathcal{O}$.

6.2.2 Implémentation de la machine virtuelle $C_{o}\mathcal{M}\mathcal{O}$

6.2.2.1 Types et adressage

Les deux piles ont une taille de 32 *cellules*. Chaque cellule est un élément de 16 bits. La cellule est le seul type concret de la machine virtuelle $C_{o}^{\flat}\mathcal{O}$, elle peut contenir :

- une adresse mémoire sur l'espace de données ou sur l'espace de code (adressage maximum de 65535 mots mémoire),
- une instruction du langage $C_{o}^{\flat}\mathcal{O}$,

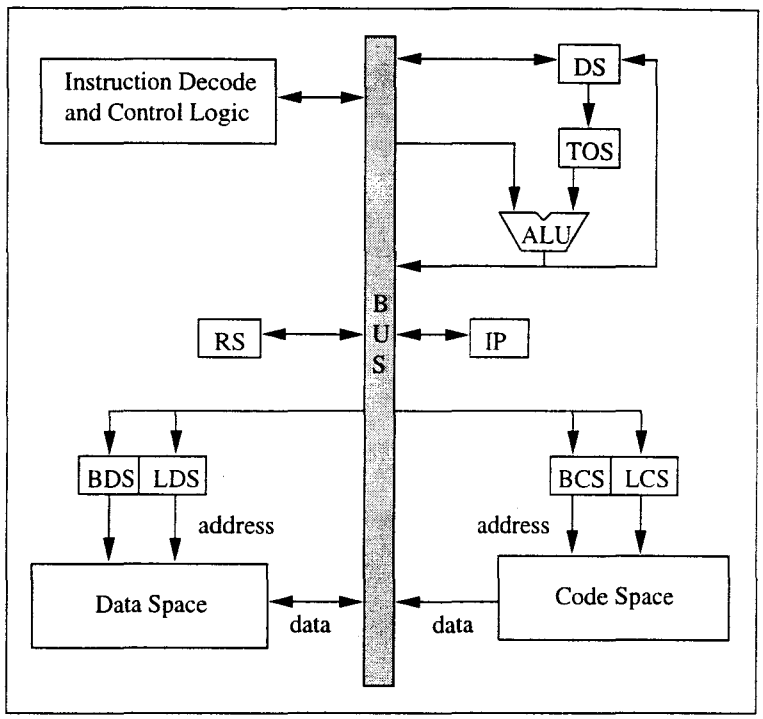

FIG. 6.2 - Diagramme bloc de la machine virtuelle C_0M_0O

- un nombre entier sur deux octets, ou
- un caractère (tenant sur 8 bits et chargé dans l'octet de poids faible de la cellule, l'octet de poids fort étant mis à 0).

Il n'y a pas de notion de type explicitement défini dans la machine virtuelle, toutes les opérations travaillent sur des valeurs de 16 bits. La façon dont sont traitées ces valeurs dépend de l'instruction qui les manipule. Les instructions sont sans opérande, elles travaillent implicitement sur les éléments des piles de la machine virtuelle. Il s'agit d'un mode d'adressage appelé *pure stack addressing* ou *0-operand addressing* sans contrôle de type.

6.2.2.2 Sécurité

Les registres de sécurité sont initialisés par le système d'exploitation avant chaque exécution d'une opération d'un service. Leurs valeurs sont positionnées par le système d'exploitation afin de limiter l'espace de données aux données du service et l'espace de code aux opérations du service. Aucun contrôle n'est effectué *a priori* sur les instructions du langage. Une erreur survient en cours d'exécution d'une opération d'un service et l'interprétation cesse chaque fois :

- qu'une instruction de lecture/écriture en mémoire de données déborde de la zone autorisée de l'espace de données,
- qu'une instruction de saut ou d'appel de sous-routines déborde de la zone autorisée de l'espace de code, et

- qu'un débordement de pile (tentative de retrait d'un élément d'une pile alors qu'elle est vide ou tentative d'ajout d'un élément sur une pile alors qu'elle est pleine) survient.

L'avantage de cette architecture est de fournir un moyen d'assurer l'encapsulation des services carte puisque les instructions sont contrôlées *à la volée* pour ne pas pouvoir déborder de la zone de données et de la zone d'adressage du service. Ces zones sont allouées statiquement en adressage continu par le système d'exploitation lors du chargement des services. L'exécution d'une opération d'un service en mode interprété assure qu'un programme utilisateur ne pourra pas aller modifier de lui-même les contrôles *câblés* dans l'interpréteur et gérer par le système d'exploitation.

6.3 L'interpréteur C_OMbO

La machine virtuelle C_OMbO exécute des instructions dites de *code filé*† ou TIL [1] [Hon92]. Chaque mot ou instruction de ce langage est appelé ici *token* car il est sans opérande et est donc exécuté directement par l'interpréteur comme un *jeton* dans un distributeur automatique. Il y a deux types de *tokens*:

Les primitives Instructions de base qui sont directement implémentées dans l'interpréteur. Ces mots définissent le langage de base de l'interpréteur et sont écrits dans le langage machine du processeur cible de la carte.

Les secondaires Ils sont définis comme des séquences de primitives. Un secondaire peut aussi contenir d'autres secondaires. L'exécution d'un secondaire correspondant donc à l'exécution de chacune de ces primitives et, en cascade, des primitives des secondaires qu'il contient. Ainsi l'exécution d'un secondaire ressemble au déroulage d'une bobine de fil (d'où l'expression *langage de code filé*) afin de faire exécuter toutes les primitives contenues dans son code et dans les secondaires qu'il contient.

Les opérations d'un service sont décrits comme des secondaires, c'est-à-dire comme une liste de tokens. L'exécution d'une opération consiste donc en l'exécution par l'interpréteur de chacun des tokens du secondaire (*cf.* FIG. 6.3 page suivante).

6.3.1 Les primitives

6.3.1.1 Notation

Ci-après est donnée la liste de toutes les primitives implémentées dans l'interpréteur C_OMbO. Ces primitives sont regroupées en six catégories : opérations internes, opérations de gestion de piles, opérations arithmétiques, opérations de comparaison, opérations logiques, structures de contrôle.

Chaque description contient une notation de l'opération avec son effet sur les piles écrites de la façon suivante :

```
(pile: contenu avant l'opération -- contenu après l'opération)
```

1. Threaded Interpretive Languages

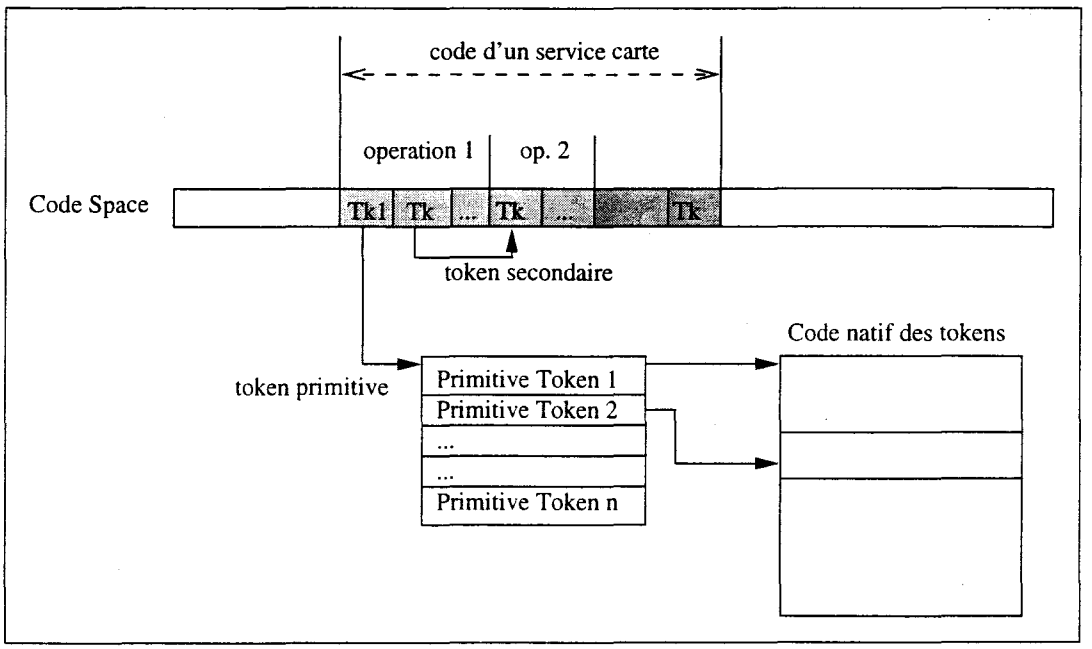

FIG. 6.3 - Structure d'une opération d'un service carte

La lecture du contenu de la pile se fait du sous-sommet vers le sommet de droite à gauche. La pile est notée `RS:` quand il s'agit de la pile de retour et `DS:` quand il s'agit de la pile de données.

Les élements présents sur les piles sont typés par l'opération couramment exécutée. Leur type implicite est noté de la façon suivante :

`addr` une adresse sur 16 bits,

`x` une valeur (entier) sur 16 bits,

`c` un caractère sur 8 bits, et

`flag` un booléen sur 16 bits (`0` pour `FALSE` sinon `TRUE`).

6.3.1.2 Opérations internes

EXIT
Operation Exit subroutine
Token `0`
Stack `(RS: addr -- )`
Description Retire l'adresse de retour `addr` de `RS` et positionne `IP` sur cette adresse

LIT
Operation Literal
Token `3`
Stack `(DS: -- x)`
Description Met le token suivant sur `DS`

!
Operation Store
Token `4`
Stack `(DS: x addr -- )`
Description Écrit la valeur `x` à l'adresse `addr`
Pseudocode `*addr=x`

C!
Operation Store character
Token `5`
Stack `(DS: c addr -- )`
Description Écrit le caractère `c` à l'adresse `addr`
Pseudocode `*addr=c & 0x00FF`

@
Operation Fetch
Token `6`
Stack `(DS: addr -- x)`
Description Charge la valeur `x` stockée à l'adresse `addr`
Pseudocode `x=*addr`

C@
Operation Fetch character
Token `7`
Stack `(DS: addr -- c)`
Description Charge le caractère `c` stocké à l'adresse `addr`
Pseudocode `c=(*addr) & 0x00FF`

6.3.1.3 Opérations de gestion de piles

DROP
Operation Drop
Token `9`
Stack `(DS: x -- )`
Description Efface le sommet de pile

DUP
Operation Duplicate
Token `10`
Stack `(DS: x -- x x)`
Description Recopie le sommet de pile

OVER
Operation Over
Token `11`
Stack `(DS: x1 x2 -- x1 x2 x1)`
Description Recopie le sous-sommet de pile

ROT
Operation Rotate
Token `12`
Stack `(DS: x1 x2 x3 -- x2 x3 x1)`
Description Déplace le sous-sous-sommet de pile en sommet de pile

SWAP
Operation Swap
Token `13`
Stack `(DS: x1 x2 -- x2 x1)`
Description Échange le sous-sommet de pile et le sommet de pile

>R

Operation	To return stack
Token	14
Stack	(DS: x --) (RS: -- x)
Description	Déplace le sommet de DS sur RS

R>

Operation	From return stack
Token	15
Stack	(RS: x --) (DS: -- x)
Description	Déplace le sommet de RS sur DS

6.3.1.4 Opérations arithmétiques

+

Operation	Plus
Token	16
Stack	(DS: x1 x2 -- x3)
Description	Additionne le sommet au sous-sommet de pile
Pseudocode	x3=x1+x2

-

Operation	Minus
Token	17
Stack	(DS: x1 x2 -- x3)
Description	Soustrait le sommet au sous-sommet de pile
Pseudocode	x3=x1-x2

*****

Operation	Multiply
Token	18
Stack	(DS: x1 x2 -- x3)
Description	Multiplie le sommet au sous-sommet de pile
Pseudocode	x3=x1*x2

/MOD

Operation	Divide
Token	19
Stack	(DS: x1 x2 -- x3 x4)
Description	Division entière du sous-sommet par le sommet
Pseudocode	x3=x1/x2 et x4=x1 mod x2

6.3.1.5 Opérations de comparaison

=

Operation	Equal
Token	20
Stack	(DS: x1 x2 -- flag)
Description	Met 1 sur la pile si et seulement si le sous-sommet est égal au sommet de pile, 0 dans le cas contraire
Pseudocode	flag=(x1==x2)

<>

Operation	Not equal
Token	21
Stack	(DS: x1 x2 -- flag)
Description	Met 1 sur la pile si et seulement si le sous-sommet est différent du sommet de pile, 0 dans le cas contraire
Pseudocode	flag=(x1!=x2)

<

Operation	Less than
Token	22
Stack	`(DS: x1 x2 -- flag)`
Description	Met 1 sur la pile si et seulement si le sous-sommet est inférieur au sommet de pile, 0 dans le cas contraire
Pseudocode	`flag=(x1<x2)`

<=

Operation	Less equal
Token	23
Stack	`(DS: x1 x2 -- flag)`
Description	Met 1 sur la pile si et seulement si le sous-sommet est inférieur ou égal au sommet de pile, 0 dans le cas contraire
Pseudocode	`flag=(x1<=x2)`

>

Operation	Greater than
Token	24
Stack	`(DS: x1 x2 -- flag)`
Description	Met 1 sur la pile si et seulement si le sous-sommet est supérieur au sommet de pile, 0 dans le cas contraire
Pseudocode	`flag=(x1>x2)`

>=

Operation	Greater equal
Token	25
Stack	`(DS: x1 x2 -- flag)`
Description	Met 1 sur la pile si et seulement si le sous-sommet est supérieur ou égal au sommet de pile, 0 dans le cas contraire
Pseudocode	`flag=(x1>=x2)`

6.3.1.6 Opérations logiques

AND

Operation	And
Token	26
Stack	`(DS: x1 x2 -- x3)`
Description	Et logique du sous-sommet et du sommet de pile
Pseudocode	`x3= x1 && x2`

OR

Operation	Or
Token	27
Stack	`(DS: x1 x2 -- x3)`
Description	Ou logique du sous-sommet et du sommet de pile
Pseudocode	`x3= x1 || x2`

XOR

Operation	Xor
Token	28
Stack	`(DS: x1 x2 -- x3)`
Description	Ou-exclusif logique du sous-sommet et du sommet de pile
Pseudocode	`x3=x1 xor x2`

INVERT

Operation	Invert
Token	29
Stack	`(DS: x1 -- x2)`
Description	Inverse tous les bits un à un du sommet de pile
Pseudocode	`x2= x1`

6.3.1.7 Structures de contrôle

BRANCH
Operation Unconditionnal branch
Token 30
Stack (DS: --)
Description Positionne IP à l'adresse constituée par la cellule suivante dans l'espace de données

BRANCH0
Operation Conditionnal branch
Token 31
Stack (DS: addr flag --)
Description Positionne IP à l'adresse constituée par la cellule suivante dans l'espace de données si et seulement si 0 en sommet de pile

DO
Operation Initiate loop
Token 32
Stack (DS: x1 x2 --) (RS: -- x1 x2)
Description Déplace les paramètres de contrôle d'une boucle (x1, la limite, et x2 l'index) sur RS

LOOP
Operation Execute loop
Token 33
Stack (DS: -- flag) (RS: x1 x2 -- x1 x3)
Description Incrémente x1 donnant x3, si x1=x3 alors met 1 sur DS sinon met 0

I
Operation Inner loop index
Token 34
Stack (DS: -- x) (RS: x -- x)
Description Copie le compteur de la boucle la plus interne sur DS

J
Operation Outer loop index
Token 35
Stack (DS: -- x) (RS: x -- x)
Description Copie le compteur de la boucle la plus externe sur DS

LEAVE
Operation Leave a loop
Token 36
Stack (RS: x1 x2 --)
Description Efface les paramètres de contrôle d'une boucle et termine la boucle à la prochaine occurence du token LOOP

UNLOOP
Operation Discard loop control parameters
Token 37
Stack (RS: x1 x2 --)
Description Efface les paramètres de contrôle d'une boucle

6.3.2 Les secondaires

Les secondaires sont implémentés comme des tokens dont le numéro n'est pas un numéro de primitive. En fait, un token secondaire est l'adresse dans l'espace de code de la liste de tokens qu'il contient. Son adresse doit être supérieure au code du dernier token primitive ce qui est pris en charge par la machine virtuelle en faisant commencer l'espace

de code à une adresse supérieure à celle du dernier token primitive. Ainsi, chaque fois qu'un token est exécuté par l'interpréteur, si son numéro est un numéro de token primitive son code machine correspondant est exécuté, sinon le pointeur d'instructions est modifié pour aller pointer dans l'espace de code sur le premier token du secondaire (*cf.* FIG. 6.4). Lorsque le token EXIT est rencontré on revient au token suivant l'appel du token secondaire qui vient d'être exécuté en récupérant son adresse sur la pile de retour. Si la pile de retour est vide, l'exécution est terminée.

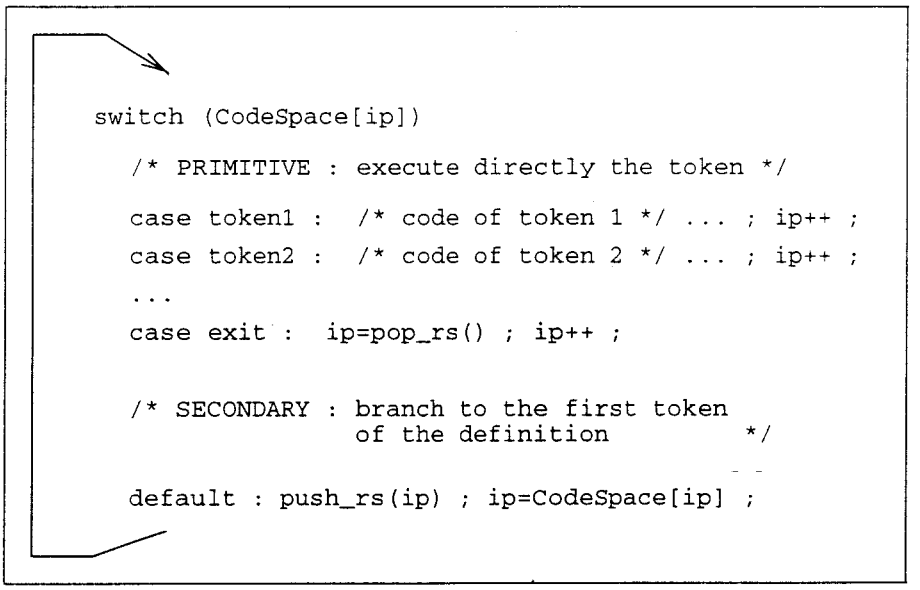

FIG. 6.4 - Boucle d'exécution de l'interpréteur C_OMb^O

Cette façon d'implémenter permet de facilement écrire l'interpréteur en langage C et de pouvoir le porter rapidement sur un grand nombre d'architectures. Elle est décrite dans [Hon92] comme *Token-Token Type* et dans [Ert] comme *Direct token threaded code.* Ses avantages sont les suivants :

- compacité du code puisque chaque secondaire est directement exécuté après un seul saut dans l'espace de code (pas de table d'indirection pour retrouver leur définition), et
- rapidité de l'exécution lorsque l'interpréteur est écrit en C et compilé sur différentes architectures cibles.

D'autres implémentations décrites dans [Hon92] et [Ert93] peuvent être plus rapides écrites en code natif pour certains processeurs mais elles ont l'inconvénient de ne pas être portables et d'utiliser plus d'espace mémoire pour le stockage des secondaires. Donc, pour des raisons de portabilité et de compacité de code nous avons retenu l'implémentation décrite ici.

6.4 La chaîne de développement C_OMb^O

Les services carte sont écrits dans un langage de haut niveau qui permet de décrire un service comme un objet : données et opérations agissant sur ces données. Les données

d'un service chargé dans la carte sont en fait une zone réservée de l'espace de données. Les opérations d'un service chargé dans la carte sont des listes de tokens chargés dans l'espace de code. L'opération qui consiste à passer de cette description de haut niveau en son implémentation dans une carte $C_o^{\mu}{}_o^O$ est réalisée par un compilateur qui transforme cette description en une description intermédiaire. Cette dernière est chargée dans la carte par un module du système d'exploitation appelé *Loader*.

Le compilateur réalise les opérations classiques d'analyse lexicale et syntaxique des services. Par contre, le compilateur ne peut prendre en charge l'adressage des données du service puisqu'il ne peut connaître à l'avance à quelles adresses dans l'espace de données d'une carte seront stockées les données du service. Il en va de même pour les calculs de saut et les appels à d'autres tokens secondaires puisque le compilateur ne peut connaître à l'avance l'endroit dans l'espace de code où seront stockés les opérations du service. Ce travail ne peut être réalisé que par le *Loader* qui alloue l'espace de données et l'espace de code pour stocker le service qu'il est en train de charger. Ainsi le *Loader* peut transformer toutes les adresses relatives de la description intermédiaire en adresses absolues en analysant le code intermédiaire au moment du chargement. Les adresses relatives sont repérées dans la description intermédiaire par des tokens particuliers qui sont supprimés au moment du chargement pour être remplacés par des adresses absolues.

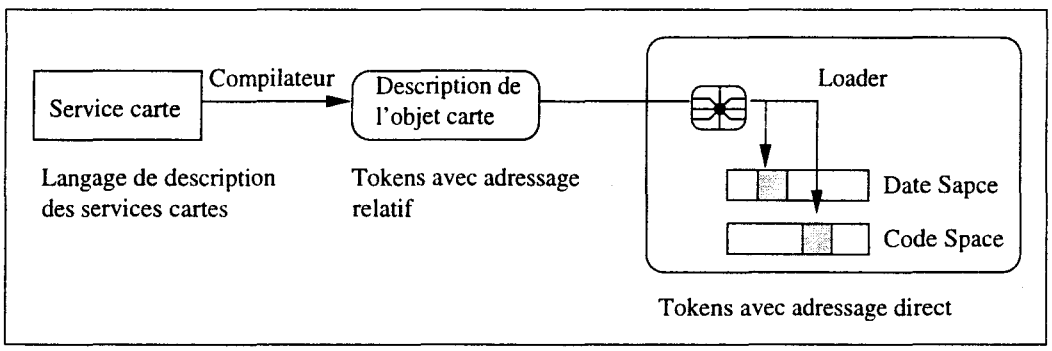

FIG. 6.5 - Chaîne de développement d'un service carte $C_o^{\mu}{}_o^O$

Suit dans cette section :

- la description en langage de haut niveau d'un service carte, et
- la description du travail du *Loader* sur les adresses relatives au moment du chargement dans la carte.

6.4.1 Langage source d'un service carte

Un service carte est décrit dans un langage source qui définit :

- le nom du service,
- les données du service et leur type (les types implémentés sont les types caractères, entier et tableau contraint de caractères ou d'entiers), et
- les opérations du service en langage Forth avec nommage explicite des données.

```
<service>       ::= <class_decl> <data_decl>+ <code_def>+
<class_decl>    ::= "class" <identifier>
<identifier>    ::= {"a"|"b"|...|"z"|"A"|"B"|...|"Z"}+
<data_decl>     ::= <type> <identifier>
                    ["["<numeric>"]"] [<init_values>] ";"
<type>          ::= "char" | "int"
<numeric>       ::= {"0"|"1"|...|"9"}+
<init_values>   ::= "=" "{" <numeric> {"," <numeric>}* "}"
<code_def>      ::= <category> <identifier> ":" <forth_code> ";"
<category>      ::= "public" | "anonymous"
```

FIG. 6.6 - Description source d'un service carte $C_{o}^{\flat}$

Les valeurs d'initialisation sont toutes décrites sous la forme de numériques. Pour un caractère, l'initialisation se fera donc avec son code ASCII[2]. Les paramètres des opérations ne sont pas donnés car ils sont implicitement retrouvés sur la pile de données par le programme. La catégorie « anonymous » définit une opération interne au service non directement exécutable depuis l'interface de la carte.

Le choix de Forth comme langage de haut niveau vient du fait qu'il se transcrit très facilement dans le langage de token $C_{o}^{\flat}$. Le sous-ensemble de Forth utilisé est donné dans la TABLE 6.1.

+	-	*	/MOD	AND
INVERT	OR	XOR	=	<>
<	>	<=	>	>=
DROP	DUP	OVER	ROT	SWAP
R>	R<	DO	LOOP	I
J	LEAVE	UNLOOP	IF	ELSE
THEN	!	@	C!	c@

TABLE 6.1 - Sous-ensemble de Forth utilisé pour écrire les services carte

6.4.2 Tokens utilisés pour l'adressage relatif

Toutes les adresses relatives de données dans le code intermédiaire ont pour base la valeur 0. Elles sont transformés en une adresse correspondante à l'adresse de la donnée dans l'espace de données dont l'emplacement est alloué dynamiquement par le *Loader*.

Les adresses d'appels à d'autres opérations du service sont transformées en l'adresse de l'opération dans l'espace de code dont l'emplacement est alloué dynamiquement par le *Loader*.

Les adresses de saut à l'intérieur d'une opération sont transformées en adresse de saut depuis la base de l'espace de code alloué pour cette opération.

2. American Standard Code for Information Interchange

Token	Traitement effectué par le *Loader*
`RELATIVE_DATA RelativeAddress`	`LIT RelativeAdress+BaseDataSpace`
`RELATIVE_CODE MethodNumber`	`MethodAddress`
`BRANCH RelativeAddress`	`BRANCH RelativeAddress+BaseCodeSpaceMethod`
`BRANCHO RelativeAddress`	`BRANCHO RelativeAddress+BaseCodeSpaceMethod`

TABLE 6.2 - Traitement par le *Loader* des adresses relatives

6.5 Conclusion

La machine virtuelle C_0M_b0 est le cœur du système d'exploitation C_0M_b0 décrit Chapitre A page 145. Dans un premier temps, elle a été implémentée sur Station Sparc de Sun sous Solaris 2.4 puis, dans un prototype de carte C_0M_b0 pour le projet CASCADE sur ARMulator pour Windows NT, un programme qui émule le jeu d'instructions pour le processeur ARM 7 [ARM95]. Sur cette dernière plate-forme, le système d'exploitation au complet (gestion mémoire, gestion des entrées/sorties, machine virtuelle, interface utilisateur) prend moins de 10 KO avec un développement en langage C sans aucune optimisation effectuée. Il est raisonnable de penser qu'une implémentation optimisée pourrait tenir en moins de 6 KO.

Les performances de l'exécution n'ont pas été évaluées car nous ne disposons pas à notre connaissance d'autres mesures issues de travaux sur des cartes génériques. Cependant, il nous semble que le choix fait concernant la machine virtuelle et l'implémentation de l'interpréteur en code filé soit validé par les nombreuses études menées pour implémenter des petits systèmes efficaces dans les systèmes embarqués (voir la bibliographie sur Forth § 6.1.3.2 page 109). La surcharge de travail engendrée par les contrôles à la volée des zones de données et de code est la contre-partie inévitable au besoin de sécurité des services carte.

Bibliographie de la machine virtuelle C_0M_b0

[ACM92] ACM's Special Interest Group on Forth. – *ACM SIGFORTH Newsletter*, ACM Press, 1991–1992.
Ensemble des numéros de SIGFORTH Newsletter encore disponibles.

[ANS93] ANSI. – *Draft proposed American National Standard for Information Systems – Programming Languages – Forth (X3J14dpANS-6)* – juin 1993, ansi/ieee x3.215-199x, 220p..
http://www.taygeta.com/forth/dpans.html.

[ARM95] ARM Advanced RISC Machines Ltd. – *ARM Software Development Toolkit Reference Manual and Programming Techniques* – juin 1995, version 2.0.

[Bro84] Brodie (Leo). – *Thinkink Forth : a language and philosophy for solving problems*, Prentice Hall, 1984.

[Bro87] Brodie (Leo). – *Starting Forth*, Prentice Hall, 1987, second.
Première édition en 1980..

[CGG93] Coulier (Charles), Gordons (Édouard) et Grimonprez (Georges). – *Carte à mémoire et procédé de fonctionnement.* – Brevet d'invention n° 93 14668, Institut National de la Propriété Industrielle, décembre 1993.

[Ert] Ertl (Anton). – Threaded Code. –
http://www.complang.tuwien.ac.at/forth/threaded-code.html.
Page Web sur les langages de code filé donnant une définition de ces langages, discutant les différentes techniques d'implémentation et fournissant une bonne bibliographie sur le sujet

[Ert93] Ertl (Anton). – A Portable Forth Engine, – *In : EuroFORTH '93 conference proceedings*, 1993.
http://www.complang.tuwien.ac.at/papers/ertl93.ps.Z.

[Hon92] Hong (P. Joseph). – Threaded Code Designs for Forth Interpreters. *ACM SIGFORTH Newsletter*, vol. 4, n° 2, décembre 1992, pp. 11–16.

[Jr.93] Jr." (Philip J. "Koopman). – A brief introduction to Forth, *In : The second ACM SIGPLAN History of programming languages conference, Cambridge, Massachusetts, USA, April 20-23, 1993.* – pp. 357–358, mars 1993.

[Koo89] Koopman Jr. (Philip J.). – *Stack Computers: the new wave*, Ellis Horwood, 1989.
http://www.cs.cmu.edu/~koopman/stack_computers/.

Livre de référence sur les machines à pile. Permet de se faire une très bonne compréhension de ce que sont les machines à piles. Traite de nombreux aspects concernant ces machines : leurs architectures, leurs logiciels de programmation (principalement Forth), leurs applications, leurs limites et leurs importances. Enfin, on trouvera en annexe une liste de processeurs conçus autour d'une architecture à pile. Dernier avantage : livre introuvable mais présent sur le Web.

[Koo91] Koopman Jr. (Philip J.). – *The proceedings of the Second and Third Annual Workshop for the ACM Special Interest Group on Forth - SIGForth'90 & SIGForth'91*, ACM Press, 1991.

[Noy88] Noyelle (Yves). – *Traitement des langages évolués - compilation, interprétation, support d'exécution*, Masson, 1988, *Manuels Informatiques Masson.*

[Pay90] Payne (William H.). – *Embedded Controller Forth for the 8051 family*, Academic Press, 1990.

[RCM93] Rather (Elizabeth D.), Colburn (Donald D.) et Moore (Charles H.). – The evolution of Forth, *In : The second ACM SIGPLAN History of programming languages conference, Cambridge, Massachusetts, USA, April 20-23, 1993.* – pp. 177–199, mars 1993.

[Woe92] Woehr (Jack). – *Forth: the new model, a programmer's handbook*, M&T Books, 1992.

Présentation claire du langage (conforme ANSI) avec exercices et solutions. Contient un source Forth portable et une très intéressante extension d'un Forth orienté objet avec héritage multiple.

CHAPITRE 7

Serveur générique de cartes

« *Un bruissement léger semblait s'élever de cette carte, peupler la chambre close et son silence d'embuscade.* »

Julien Gracq. *Le rivages des Syrtes.* José Corti, Paris, France, 1951.

Résumé

Dans ce chapitre, nous présentons les technologies proposées par l'OMG pour définir une architecture uniforme de système d'information orienté objet distribué. Nous proposons de travailler sur l'intégration de la carte générique dans une telle architecture qui représente pour nous un support d'information privilégié pour développer des applications à grande échelle. Ensuite, nous montrons quels sont les besoins en terme d'intégration de cartes génériques et comment nous les avons abordés dans une telle architecture. Nous détaillons le prototype de serveur générique de cartes que nous avons réalisé avec un ORB du marché et discutons certains aspects de cette implémentation.

Sommaire

Tables

Figures

7.1 Cadre de travail

7.1.1 Intégration de la carte générique

La carte générique C_{O}MO prend tout son sens si elle peut facilement s'interfacer avec les systèmes d'information des clients qui utilisent les services que la carte contient. Pour cela, il est nécessaire que les clients puissent rapidement développer des programmes qui vont exécuter des services carte et qui s'intégrent dans leurs systèmes d'information. Prenant comme modèle les systèmes à grande échelle nous proposons un serveur générique de cartes pour les applications orientées objet dans une architecture prenant en compte les problèmes d'interopérabilité.

Dans ce domaine, un consortium de vendeurs, développeurs et utilisateurs de systèmes orientés objet – l'OMG [1] [Sol94] – a été créé pour promouvoir l'utilisation des technologies objet pour le développement de systèmes informatiques distribués interopérables. Cette organisation a proposé un modèle de référence appelé OMA [2] [OMG O92] dans lequel chaque composant logiciel est défini comme un objet communicant avec les autres objets au travers d'un ORB [3]. L'ORB est le substrat de communication fournissant les mécanismes grâce auxquels les objets peuvent, de façon transparente, invoquer des opérations sur d'autres objets et récupérer les réponses.

Cette architecture est un bon modèle pour développer des applications interopérables dans lesquels les cartes génériques pourraient proposer leurs services afin de développer des systèmes à grande échelle intégrant la notion de système d'information individuel. Intégrer la carte générique dans un tel cadre peut donc fournir une base de développement pour

1. Object Management Group
2. Object Management Architecture
3. Object Request Broker

toute une série d'applications proches de ce modèle en reprenant une partie des travaux menés ici.

7.1.2 L'architecture CORBA

CORBA[4] est le nom donné au standard définissant l'ORB [OMG 093]. Le service de base fourni par CORBA est le transport de messages (ou requêtes) depuis un client jusqu'à une implémentation objet et, inversement, le transport de réponses (valeurs de retour ou erreurs) depuis l'objet invoqué jusqu'à l'appelant. Le *client* est l'entité qui désire exécuter une opération sur un objet et *l'implémentation objet* désigne le code et les données qui implémentent cet objet.

Le substrat de communication générique fourni par CORBA est rendu possible par la définition d'une interface standard avec l'ORB et par l'utilisation d'un langage de spécification unique appelé IDL [OMG 094] qui est utilisé pour spécifier les interfaces des objets indépendamment des langages qui les implémentent. Ce langage de description d'interfaces peut être projeté dans n'importe quel langage de programmation en utilisant un compilateur IDL qui va ajouter au langage de projection les « bouts de code » nécessaires à l'utilisation du noyau ORB pour réaliser l'appel d'opérations, le passage de paramètres et la réception des résultats. Le source d'un objet en IDL est compilé pour produire du code pour le client de l'objet (les *stubs*) et du code pour l'implémentation de cet objet (les *squelettes*). Ces codes prennent en charge l'empaquetage et le dépaquetage des requêtes et des réponses transportées ainsi de façon transparente par l'ORB (*cf.* FIG. 7.1 page suivante).

Quand le client dispose des stubs correspondants aux objets auxquels il accède, il s'agit d'un schéma d'appel *statique* dans lequel le programme appelant est compilé pour exécuter des requêtes sur ces objets distants. CORBA offre aussi un schéma d'appel *dynamique* dans lequel le client peut découvrir à l'exécution les interfaces d'objets disponibles, construire une requête sur une opération d'un de ces objets, envoyer sa requête et récupérer la réponse de l'objet. Tout cela est possible sans connaissance *a priori* des interfaces des objets au moment de la compilation, c'est-à-dire sans aucun stub pour l'objet qui est accédé. Ce mécanisme, appelé *Invocation Dynamique*, utilise la découverte dynamique des interfaces disponibles qui sont stockées dans un service standard de l'ORB appelé *dépôt d'interfaces* ou IR[5]. Cet IR est renseigné par les constructeurs d'implémentations objet chaque fois qu'un nouvel objet est rendu disponible. L'IR maintient une représentation dynamique des interfaces de tous les objets disponibles dans le système distribué CORBA.

Une importante extension du noyau de communication ORB est la possibilité de définir des *Adaptateurs d'Objets* qui sont des mécanismes d'abstraction permettant de cacher les détails de l'implémentation objet à l'ORB. L'adaptateur d'objets permet d'adapter la variété des implémentations d'objets au modèle générique défini par l'ORB. Par exemple, des objets peuvent être implémentés à l'intérieur de processus d'un système d'exploitation ou peuvent être implémentés comme des objets dans une base de données orientée objet.

En résumé, CORBA, par son modèle fédérateur et par la variété des possiblités de développement qu'il offre, peut être vu comme la spécification de l'interface de program-

4. Common Object Request Broker Architerture
5. Interface Repository

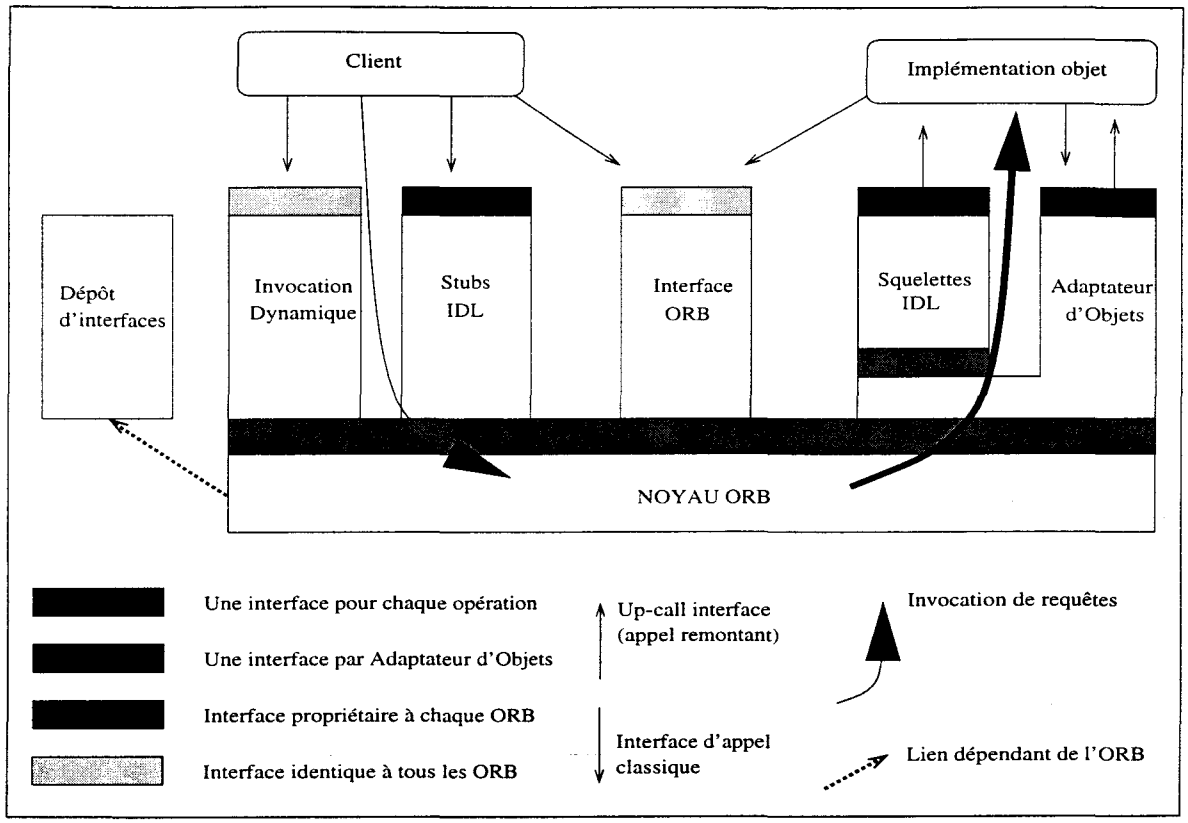

FIG. 7.1 - Architecture CORBA

mation des prochains systèmes d'exploitation orienté objet. Les applications construites sur ces systèmes sont aussi plus facilement portables et interopérables puisque CORBA est bâti comme une couche de middleware telle que celle présentée FIG. 1.4 page 11. C'est pourquoi nous avons choisi de travailler sur l'intégration de la carte générique dans une telle architecture.

7.2 Intégration de la carte générique dans CORBA

7.2.1 Besoins spécifiques à la carte générique

Le problème de l'interfaçage des services carte dans l'architecture CORBA est que l'implémentation objet du service est à l'intérieur de la carte qui, elle, utilise des protocoles de communication spécifiques. Par conséquent, il est impossible de directement intégrer un service carte sur la couche de communication fourni par un ORB. Il est donc nécessaire de pouvoir transcrire des messages objet d'un ORB en commandes et réponses carte décrites par des APDU. Il s'agit donc de définir un adaptateur d'objets pour cartes, que nous appelons COA[6], qui fournit une interface d'interopérabilité entre des applications CORBA et des services carte.

Le second problème provient de la nature même des cartes génériques qui à un instant

6. Card Object Adapter

donné de leur cycle de vie peuvent contenir des services dont les interfaces ne sont pas connues à l'avance ni par les programmes client ni par le COA. En utilisant une description d'interfaces indépendante de l'implémentation telle que celle fournie par le langage IDL, les interfaces de services carte peuvent être distribuées par les prestataires de services à tous leurs clients en même temps que les certificats d'autorisation. Ainsi, ces descriptions de services carte peuvent être stockées dans les dépôts d'interfaces des clients pour pouvoir être utilisées pour construire les programmes client et pour paramétrer dynamiquement leur COA.

7.2.2 Implémentation du COA

Le COA est une passerelle générique entre un ORB et des cartes génériques. Il fournit une interface objet des services carte en créant des « représentants » (ou proxys) qui sont des implémentations objets conformes à l'interface de l'ORB et qui servent d'objets intermédiaires entre un programme client et des services carte. Un proxy est créé dynamiquement par le COA lorsqu'un client accède pour la première fois à un service carte. Pour cela, le COA utilise le dépôt d'interfaces standard de l'ORB (l'IR) pour retrouver la description IDL du service carte. Les invocations d'opérations sur ce service sont ensuite prises en charge par son proxy qui :

- contrôle la validité de la requête reçue grâce à la signature de l'objet trouvée dans l'IR,
- transcrit la requête et ses arguments en une (ou plusieurs) commande carte grâce à un dépôt de codes carte (ou CCR[7]) qui associe à chaque interface d'opération la/les commande(s) APDU correspondante(s) pour la carte,
- envoie la/les commande(s) APDU à la carte, et
- retourne la/les réponse(s) carte au client en transcrivant les valeurs de retour en valeurs conformes aux types attendus par le client.

Le CCR contient une description de la translation à effectuer depuis une opération d'un objet CORBA jusqu'à une commande carte. Cette translation peut être exprimée dans un langage de scripts qui associe à chaque opération une commande APDU ainsi que la projection des arguments en données de la commande (champ `Din` de la commande APDU) et des valeurs de retour (champs `SW1`, `SW2` et `Dout` de la réponse APDU) dans les types de données attendus en retour pour cette opération. Par initialisation du CCR pour différents types de cartes, le COA peut s'interfacer avec des cartes différentes qui contiennent le même service, et offrir ainsi aux programmes clients la même interface d'accès au service. Selon la carte utilisée, le COA transcrit les requêtes objet en des commandes APDU conformes avec le système d'exploitation de la carte sous-jacente.

7.2.3 Scénario d'utilisation du COA

Dans ce paragraphe, nous illustrons un exemple de fonctionnement d'une application carte utilisant le COA (*cf.* FIG. 7.2 page suivante).

7. Card Code Repository

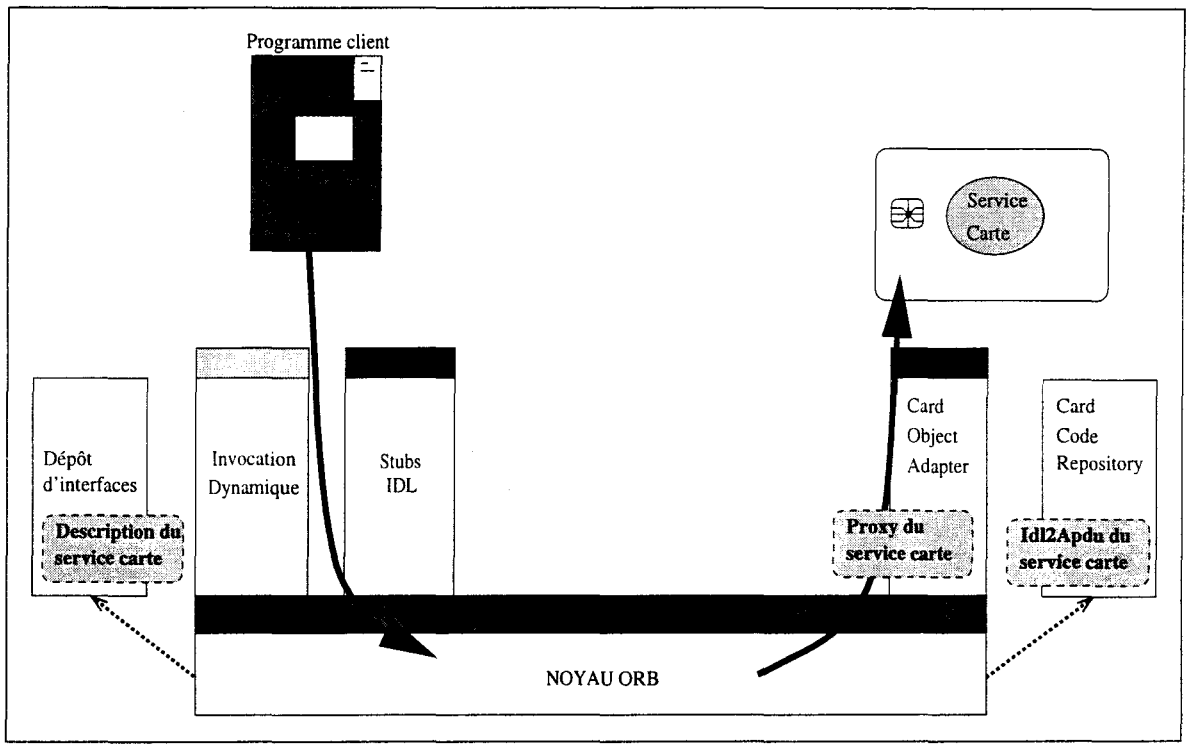

FIG. 7.2 - Application carte dans une architecture CORBA

1. Nous supposons que chaque client carte utilise un ORB comme couche de communication pour les composants logiciels de son système d'information. Il dispose aussi d'un lecteur de carte connecté à son système informatique et du COA qui implémente la couche standard de transport de requêtes APDU vers le lecteur connecté. Enfin, un IR et un CCR sont présents sur son ORB.

2. Les prestataires de services carte implémentent leurs services dans des cartes génériques et distribuent à leurs clients les certificats d'autorisation leur permettant d'accéder à ces services. Ils distribuent aussi l'interface IDL ainsi que les scripts `Idl2Apdu` de ces services leur permettant de mettre à jour leur IR et leur CCR.

3. Les clients peuvent compiler les descriptions IDL des services carte pour produire les stubs qu'ils intégrent dans leurs programmes applicatifs afin de pouvoir exécuter *via* le COA des opérations de services carte. Ils peuvent aussi utiliser le mécanisme d'invocation dynamique pour construire au moment de l'exécution des requêtes vers les services carte.

4. Quand une carte est connectée, le programme client présente son certificat d'autorisation pour s'authentifier auprès d'un service carte. Il peut ensuite exécuter des requêtes sur ce service en passant par le COA qui dynamiquement crée le proxy correspondant au service. Le proxy réalise ensuite la transcription des requêtes et réponses entre l'ORB et l'implémentation du service dans la carte.

7.3 Prototype du COA réalisé sous Orbix

7.3.1 Problématique du COA

Un prototype du COA basé sur les principes présentés ci-dessus a été réalisé par Chamussy (Thomas) dans le cadre d'un mémoire de DEA[8] [Cha95] encadré par l'auteur et Merle (Philippe) qui a passé une thèse sur l'utilisation de l'interface Web pour accéder à des objets distribués sur des architectures CORBA [MGG96a, MGG96b, Mer97]. Ce prototype a été implémenté pour station SunSparc sous Solaris 2.4 en mettant en œuvre la plate-forme CORBA Orbix 1.3 [ION95], un ORB interfacé avec le langage C++.

Le prototype de COA implémente un serveur qui va servir d'intermédiaire entre des clients et des services carte. Le rôle de ce serveur consiste principalement à créer des objets représentants des services carte en accord avec leur description IDL retrouvée dans le dépôt d'interfaces de l'ORB. Le principal problème à résoudre est que le COA ne connaît pas à l'avance le type des objets et de leurs méthodes pour lesquels il sert de serveur. En effet, il ne dispose pas à l'avance – c'est-à-dire au moment où il est compilé – des squelettes des services carte pour lesquels il sert de passerelle puisqu'il doit offrir une interface uniforme d'accès aux services carte sans avoir à être recompilé chaque fois qu'une nouvelle description IDL provient d'un prestataire de services carte.

Le COA implémente donc un mécanisme de création dynamique de squelettes pour permettre au client de se lier avec les services cartes au fur et à mesure qu'ils sont créés. Ces squelettes créés dynamiquement doivent offrir les mêmes services que ceux issus de la compilation en classes C++ d'une description IDL. Cette implémentation a donc nécessité une étude des couches basses d'Orbix non documentées, notamment en ce qui concerne le code généré par la compilation en C++ d'une description IDL d'un objet pour produire les stubs et squelettes qui résolvent les problèmes de liaison et d'empaquetage/dépaquetage des requêtes/réponses transitant sur l'ORB. Ensuite, nous avons recréé dans le COA ces mêmes classes mais cette fois-ci de façon dynamique et en y ajoutant la partie transcription de requêtes objet en commandes APDU.

7.3.2 Étude d'une application Orbix

Afin d'étudier l'organisation et le rôle des différentes classes générées par le compilateur IDL nous avons écrit un serveur de porte-monnaie électronique qui implémente un objet **pme** très simple (*cf.* FIG. 7.3 page suivante).

La compilation de cette description IDL génère principalement quatre classes (*cf.* FIG. 7.4 page suivante) qui serviront à la réalisation des applications clientes de ce service et à l'implémentation de ce service :

La classe pme Cette classe hérite de la classe `CORBA::Object`. Elle définit les opérations qu'une application cliente peut utiliser sur un objet de type **pme**. Elle fournit aussi les opérations nécessaires à la liaison d'un client sur le serveur fournissant l'implémentation d'un objet **pme**. Cette classe n'est utilisée que dans les stubs pour permettre

8. Diplôme d'Études Appronfondies

```
// IDL -- pme.idl
interface pme {
  readonly attribute int solde;
  void credit(in int value);
  void debit(in int value);
};
```

FIG. 7.3 - Description IDL de la classe pme

au client de programmer son application utilisant un objet de type pme comme un objet local.

La classe `pmeBOAImpl` Cette classe hérite de la classe pme et fournit le code nécessaire à la création du dispatcher de requêtes `pme_dispatch`.

La classe `pme_i` Cette classe contient l'entête des opérations de l'objet pme dont le programmeur doit écrire le code afin d'implémenter effectivement les objets du type pme. Cette classe n'est utilisée que pour écrire le code effectif d'une implémentation de l'objet pme.

La classe `pme_dispatch` Cette classe hérite de la classe `CORBA::PPTR`. Elle prend en charge l'empaquetage et le dépaquetage des requêtes et des réponses vers l'implémentation de l'objet pme. Lorsqu'une requête arrive au serveur fournissant un objet pme, c'est cette classe qui prend en charge la requête. Après avoir déterminé le type d'opération invoquée, elle vérifie la syntaxe de la requête ainsi que les paramètres fournis à l'aide de filtres de correspondance entre les valeurs reçues de l'ORB (nom de l'opération invoquée, paramètres, *etc.*) et celles attendues par l'objet pme. Une fois ces vérifications terminées, elle invoque effectivement l'opération sur l'implémentation de l'objet. Elle récupère ensuite la réponse de l'exécution pour la coder et la transmettre à l'ORB.

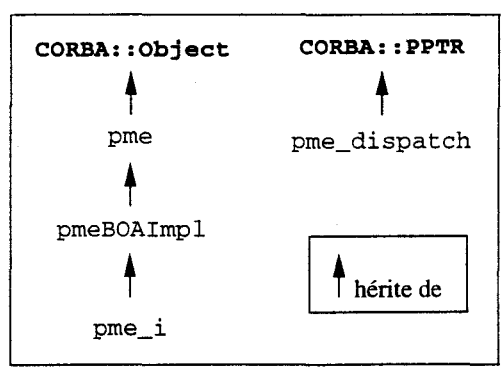

FIG. 7.4 - Classes Orbix pour l'objet pme

Une autre classe importante d'Orbix est défini par `CORBA::LoaderClass` qui sert au chargement en mémoire des objets dont le serveur a besoin pour fournir le dispatcher correspondant à l'objet auquel le client souhaite accéder.

Il ressort de cette étude que le travail principal à fournir pour réaliser un serveur générique concerne la classe implémentant le dispatcher. En effet, dans le cas du pme, les squelettes contiennent un dispatcher dont les filtres sont entièrement dédiés au traitement

des requêtes pour les opérations de cet objet. Dans notre cas, le dispatcher doit utiliser des filtres créés dynamiquement dans le serveur pour pouvoir traiter les requêtes dont il ne connaît au moment de la compilation ni la syntaxe ni les types des paramètres et des valeurs de retour.

7.3.3 Le COA Orbix

La façon de créer dynamiquement les filtres pour chaque type d'objet carte accédé par une application cliente sera d'utiliser le dépôt d'interfaces pour renseigner « à la volée » le dispatcher sur la syntaxe des requêtes à vérifier. Ensuite, le dispatcher ainsi renseigné avec les filtres pour le service carte accédé se charge du codage des requêtes objet en commandes carte qu'il envoie de manière standard au format APDU au lecteur de carte pour lequel le COA sert de passerelle.

7.3.3.1 Classes du COA

Le COA réalisé sous Orbix est construit autour de quatre classes principales (*cf.* FIG. 7.5 page suivante) :

La classe `CoaLoader` Cette classe hérite de `CORBA::Loader`. Elle sert à établir la liaison avec l'application cliente (par simple héritage de la classe `CORBA::Loader`) et à instancier les autres objets du COA.

La classe `CoaObject` Cette classe hérite de `CORBA::Object`. Elle sert à créer l'objet correspondant au service carte invoqué par le client. Cet objet est en fait le proxy auquel le client se lie dans son application. Cet objet n'a ensuite pour rôle que de créer l'objet `CoaDispatch` pour le service carte invoqué.

La classe `CoaDispatch` Cette classe hérite de `CORBA::PPTR`. Elle réalise le décodage et la vérification des requêtes provenant de l'ORB pour les transmettre à la carte; et inversement avec les réponses issues de la carte. Les informations nécessaires à la vérification des requêtes (filtres) et à leur transcription en commandes APDU sont conservées dans un objet de type `CoaInterface`.

La classe `CoaInterface` Cette classe maintient les informations nécessaires au `CoaDispatch`. Elle contient l'ensemble des filtres nécessaires à la vérification d'une requête (`signature`, `TCassert` et `ParamSequence`) et au codage de la réponse (`ConvertToReply`) plus, un pointeur vers les scripts du CCR à exécuter pour convertir une requête en des commandes APDU.

7.3.3.2 Fonctionnement des classes du COA

Le fonctionnement de ces classes est illustré sur un exemple de porte-monnaie électronique, cette fois-ci implémenté comme un service chargé dans une carte C_{OSO}.

1. Au départ, le COA est actif et référencé comme serveur sur le bus ORB. Seul l'objet `CoaLoader` est chargé en mémoire.

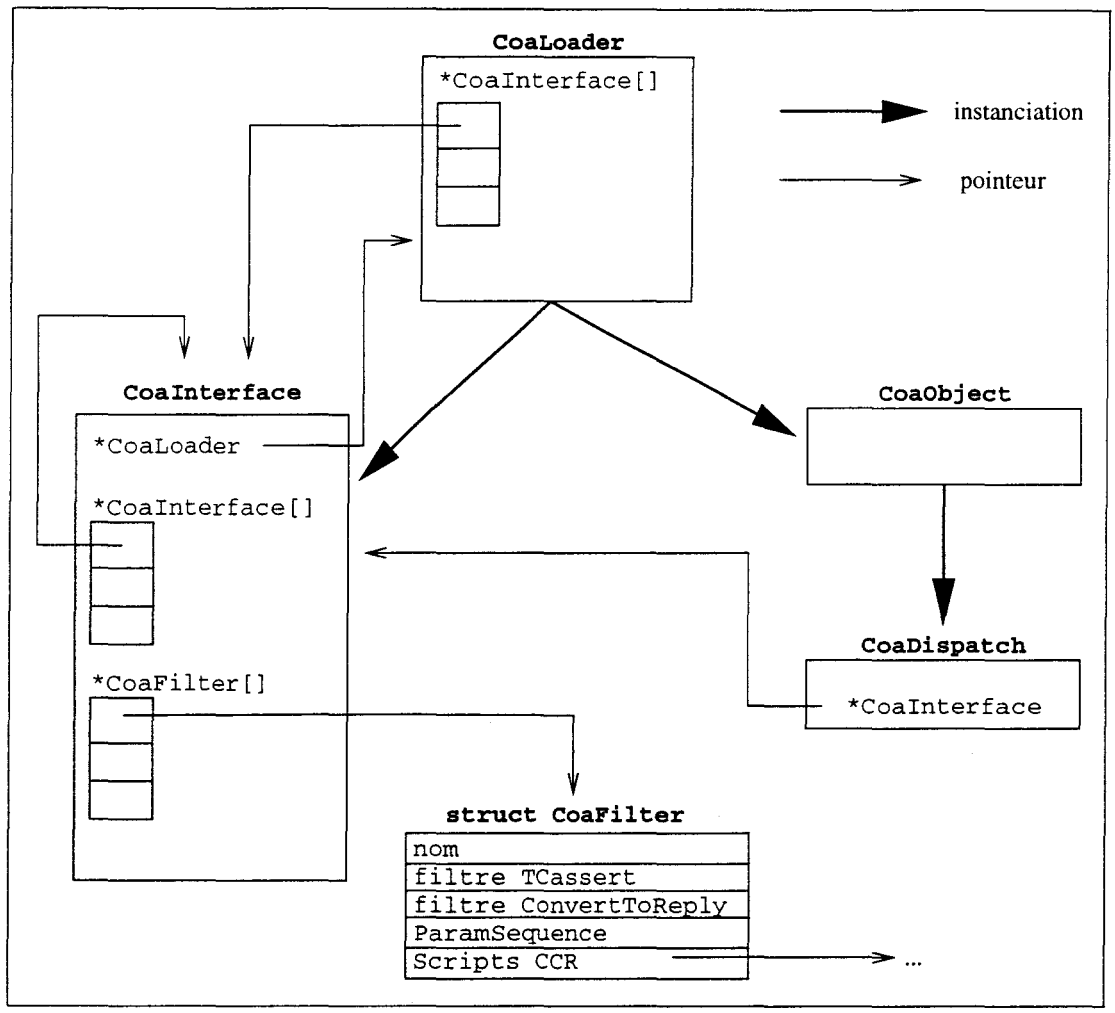

FIG. 7.5 - Classes du COA

2. Une application cliente se connecte au serveur générique de cartes et demande l'accès à un objet de type pme. Le CoaLoader reçoit cette requête. Il commence par vérifier s'il n'existe pas déjà un objet CoaInterface instancié pour le type pme grâce au tableau *CoaInterface[] qu'il maintient. S'il n'en n'existe pas il instancie un nouveau CoaInterface et un nouveau CoaObject pour le type pme.

3. L'objet CoaInterface se connecte dès sa création au dépôt d'interfaces (IR) d'Orbix pour récupérer la description de l'interface du service pme. Il construit alors les filtres correspondants à toutes les opérations de l'interface décrite dans la description IDL. Il se connecte ensuite au dépôt de codes cartes (CCR) pour récupérer les scripts de transcriptions des requêtes objets de l'interface pme en commandes APDU pour la carte $C_{o}^{\mu O}$. Si l'interface pme hérite d'autres interfaces il consulte le tableau des *CoaInterface[] maintenu par le CoaLoader pour connaître les objets CoaInterface déjà créés pour ces interfaces. S'ils n'existent pas il les instancie de façon récursive en remontant la chaîne d'héritage qui est conservée dans le tableau *CoaInterface[] maintenu par chaque objet CoaInterface.

4. L'objet CoaObject est créé pour le type pme. Cet objet est le proxy auquel l'application cliente est lié. Il ne réalise aucun traitement si ce n'est de créer l'objet CoaDispatch pour le type pme qui recevra par la suite toutes les requêtes adressées

à l'objet `pme` auquel est liée l'application cliente. Lors de son instanciation l'objet `CoaDispatch` reçoit de l'objet `CoaObject` un pointeur sur l'objet `CoaInterface` correspondant au type `pme`. Ce pointeur permet ensuite au `CoaDispatch` de retrouver toutes les informations nécessaires au traitement des opérations invoquées sur l'objet `pme`.

5. Lorsque le client lance une requête sur l'objet `pme`, celle-ci est directement prise en charge par l'objet `CoaDispatch` qui, grâce à la signature de l'opération invoquée, retrouve la structure `CoaFilter` qui va lui permettre de :

   - vérifier la syntaxe de la requête grâce au filtre `TCassert`,
   - vérifier la conformité des paramètres effectifs avec la description des paramètres formels contenue dans `ParamSequence`,
   - transcrire la requête objet en la commande APDU C_{OBO} grâce au script contenu dans le CCR,
   - transcrire la réponse carte en réponse objet grâce aussi au script contenu dans le CCR, et
   - vérifier la conformité du type de la réponse avec le filtre `ConvertToReply`.

Les requêtes aux services carte prises en charge par le COA peuvent être, du côté client, invoquées dynamiquement au travers du DII[9] ou compilées dans un programme à l'aide des stubs générés par le compilateur IDL. Le COA ne fait aucune différence entre ces deux types de requêtes et les traite exactement de la même façon.

7.3.3.3 Remarques sur l'implémentation réalisée

Dynamic Skeleton Interface L'implémentation réalisée a demandé un important travail sur la façon dont Orbix effectue l'empaquetage et le dépaquetage des requêtes transistant sur l'ORB. Ces tâches sont en fait codées « en dur » aux travers de filtres inclus dans les stubs et squelettes générés par le compilateur IDL. La difficulté consistait à comprendre comment ces filtres sont codés et utilisés dans les squelettes pour pouvoir dynamiquement les regénérer dans le COA. Ce travail a été mené en compilant l'ensemble des types IDL et en étudiant les filtres générés. Ensuite, nous avons reproduit le travail du compilateur IDL dans l'implémentation du COA qui réalise ce même travail mais, cette fois-ci, dynamiquement en retrouvant dans l'IR les informations sur la description IDL.

Par conséquent, nous avons dû faire du « *reverse engineering* » sur le produit Orbix afin de pouvoir apporter de la généricité à la prise en charge des requêtes par le serveur. Cela peut sembler un peu décevant mais nous n'avions pas à l'époque d'autres moyens d'y arriver. Depuis lors, l'OMG a spécifié un nouveau service appelé DSI[10] qui permet à un serveur de délivrer des requêtes à une implémentation objet dont il ne connait pas le type au moment de la compilation [OMG 095, YD96]. L'utilisation du DSI nous permettrait de développer plus aisement le COA tout en tirant bénéfice d'un nouveau service standard conçu spécifiquement pour améliorer l'interopérabilité et la dynamicité des applications.

9. Dynamic Invocation Interface
10. Dynamic Skeleton Interface

Card Code Repository Actuellement, le CCR développé est très rudimentaire puisqu'il ne permet que la transcription simple d'une requête à un service carte en la requête `ExecuteMethod` pour la carte *CoMbO* (*cf.* § A.3.3.2 page 157). Il associe donc à une description IDL d'un service carte les informations données TABLE 7.1.

Information IDL	Information *CoMbO*
Nom de l'interface	ObjectRef, IssuerRef
Nom de la méthode	MethodNumber
Paramètres	-

TABLE 7.1 - Contenu du Card Code Repository

Les paramètres n'étant que de types simples (`int`, `character`, `int[]` et `string`) leur transcription en chaînes d'octets dans la commande APDU envoyés à la carte est automatiquement effectuée par le COA. Ensuite, la requête APDU est envoyée à la carte par l'intermédiaire d'un driver standard de lecteur carte.

Pour permettre un paramétrage plus fin de cette opération de transcription, nous envisageons l'utilisation d'un langage de script tel que *CorbaScript* décrit dans la thèse de Merle (Philippe) [Mer97] de la façon suivante :

> « *CorbaScript est un langage de script interprété, à typage dynamique, modulaire et orienté objet. Il permet d'invoquer n'importe quelle opération, de consulter ou de modifier n'importe quel attribut de tout objet Corba. Pour cela, ce langage met en oeuvre les mécanismes dynamiques de Corba : le référentiel des interfaces (IR) pour découvrir les métadonnées de typage, l'interface d'invocation dynamique (DII) pour construire les requêtes et l'interface de squelettes dynamiques (DSI) pour recevoir des requêtes. CorbaScript fournit ainsi une "colle logicielle" pour exploiter tout service Corba de manière simple et flexible.* »

Ainsi, nous pourrions très bien concevoir un COA capable de dynamiquement construire des requêtes APDU pour n'importe quel type de cartes, y compris des cartes non génériques pour lesquelles la transcription d'une opération de débit sur un porte-monnaie électronique aboutirait à l'envoi de plusieurs commandes APDU et même à du traitement à l'intérieur du serveur.

7.4 Conclusion

L'interface d'une carte générique dans un système d'information correspond à l'ensemble des interfaces des services qu'elle contient. Son intégration requiert donc de pouvoir décrire ces interfaces de façon « transportable» pour que les applications clientes utilisant ces services puissent facilement mener des transactions avec la carte. La description IDL d'un service carte permet de mettre à jour le système d'information d'un client. Pour cela, le prestataire de services fournit à ses clients la description IDL de son service et le script de transcription des requêtes en commande APDU pour la carte (voir même pour plusieurs types de cartes). Les clients ajoutent ces informations respectivement dans l'IR et dans le CCR de leur ORB. Les clients peuvent ensuite développer des applications accédant à ces services sans se préoccuper de leur implémentation puisque le COA assure de façon dynamique leur interfaçage avec la carte.

Le COA est donc une passerelle générique et dynamique à des services carte puisqu'il se met à jour dynamiquement par la consultation de l'IR et qu'il permet l'interfaçage générique avec n'importe quel service carte. Il implémente donc une couche middleware d'interopérabilité avec les cartes à microprocesseur qui pourrait prendre place dans les services de base des architectures de systèmes à grande échelle.

Bibliographie du serveur générique de cartes

[Cha95] Chamussy (Thomas). – *Intégration de la carte multi-applicative dans une architecture CORBA*, Mémoire de DEA, USTL Université des Sciences et Technologies de Lille, septembre 1995.

[ION95] IONA Technologies Ltd. – *Orbix: Programmer's Guide and Advanced Programmer's Guide* – avril 1995, 1.3.

[Kha94] Khanna (Raman). – *Distributed Computing*, Prentice-Hall, 1994.

[Mer97] Merle (Philippe). – *CorbaScript - CorbaWeb : propositions pour l'accès à des objets et services distribués*, Thèse de Doctorat, USTL Université des Sciences et Technologies de Lille, janvier 1997.

[MGG96a] Merle (Philippe), Gransart (Christophe) et Geib (Jean-Marc). – CorbaScript and CorbaWeb: A Generic Object-Oriented Dynamic Environment upon CORBA, – *In: Proceedings of TOOLS Europe'96, Palais des Congrès, Paris, France, February 26–29, 1996*, février 1996.

[MGG96b] Merle (Philippe), Gransart (Christophe) et Geib (Jean-Marc). – CorbaWeb: A Generic Object Navigator, – *In: Proceedings of the Fifth International World Wide Web Conference, Paris, France, may 6–10, 1996*, mai 1996.

[OMG O92] OMG Object Management Group. – *Object Management Architecture Guide 2.0* – septembre 1992.

[OMG O93] OMG Object Management Group. – *The Common Object Request Broker: Architecture and Specification Version 1.2* – décembre 1993.

[OMG O94] OMG Object Management Group. – *IDL C++ language mapping specification* – septembre 1994.

[OMG O95] OMG Object Management Group. – *The Common Object Request Broker Architecture And Specification Version 2.0* – juillet 1995.
http://www.omg.org/docs/ptc/96-03-04.htm.

[Sol94] Soley (Richard Mark). – Role of Object Technology in Distributed Systems, Appendix B, pp. 469–478. – In Khanna [Kha94].

[YD96] Yang (Zhonghua) et Duddy (Keith). – CORBA: A Platform for Distributed Object Computing. *Operating System Review*, vol. 30, n° 2, avril 1996, pp. 4–31.
Un état de l'art sur CORBA.

Troisième partie

Conclusion

CHAPITRE 8

Conclusion et perspectives

« A PhD to show you're smart
With textbook formulas
But you're used up
Just like a factory hand »

Dead Kennedys. *Well paid scientist*, San Fransisco, États-Unis, 1982.

8.1 Originalité de ce mémoire

Le plus souvent, les cartes à microprocesseur sont perçues comme des outils inviolables et donc uniquement intéressants pour stocker des secrets et réaliser des calculs cryptographiques. L'aspect « serveur personnel » qu'elles apportent est généralement oublié à cause de leur trop faible capacité de stockage et/ou de traitement. Pourtant, commencent à apparaître sur le marché des composants pour cartes offrant des mémoires importantes et des processeurs puissants. Dès lors, des concepteurs envisagent d'utiliser ces cartes pour offrir une multitude de services tels que le paiement électronique, la billétique, la personnalisation de services, le dossier portable, *etc.* Malheureusement, leurs ardeurs sont vite éteintes à cause de la lourdeur et de la rigidité de ces machines qui ne rendent un service que si elles sont programmées de A à Z pour ce service ! L'idée de réaliser une carte générique, non dédiée à une application, une carte, en quelque sorte, identique à un PC dans laquelle sont chargés les programmes et les données nécessaires à l'utilisateur et s'appuyant sur un système d'exploitation ouvert, cette idée est donc dans l'air. Le travail présenté dans ce mémoire est une première proposition dans ce domaine. Il bénéficie ainsi des avantages liés à la nouveauté mais doit aussi en assumer les inconvénients dont le principal est le manque de points de repère et de comparaison.

En plaçant la carte générique dans les architectures de systèmes ouverts, nous nous sommes efforcés de montrer que les services qu'elle héberge doivent résoudre des problèmes similaires à ceux rencontrés dans les systèmes à grande échelle, à savoir :

- authentification et contrôle d'accès des utilisateurs,
- exécution sécurisée des programmes, et

- intégration dans les systèmes d'information.

L'originalité de ce travail réside donc essentiellement dans le fait que nous avons abordé le problème des cartes génériques d'une manière globale en tenant compte non seulement des spécificités carte mais aussi des caractéristiques des systèmes environnants. C'est ce que nous avons appelé le *modèle de système à cartes génériques* (*cf.* Chapitre 4 page 69) qui définit un nouveau cycle de vie évolutif pour les cartes génériques ayant pour impacts les problèmes énoncés ci-dessus.

8.2 Apports de ce mémoire

L'approche orientée objet nous a servi de modèle pour définir un service chargé dans une carte générique comme un « objet distribué » dont :

1. l'utilisation par des clients est contrôlée de manière off-line, c'est-à-dire sans pouvoir utiliser un service centralisé qui maintiendrait des listes statiques de contrôle d'accès,
2. l'exécution ne doit pas compromettre, en cas d'erreur ou en cas de fraude volontaire, la confidentialité et l'intégrité des autres services, et
3. l'utilisation par des applications clientes doit être facilitée par une interface d'intégration uniforme dans les architectures de système ouvert.

Les apports de ce mémoire concernent ces trois problèmes pour lesquels nous avons proposé les solutions suivantes :

Certificat d'accès aux objets Nous utilisons des certificats cryptographiques à clé asymétrique contenant des capacités pour authentifier et contrôler l'accès aux services carte. Les certificats sont délivrés off-line sans notification aux services déjà distribués dans des cartes. Ces certificats sont vérifiés par la carte lors de chaque accès à un service : leur signature par une autorité supérieure à l'appelant les authentifie comme valides et intègres, la carte authentifie l'appelant grâce à sa clé publique retrouvée dans le certificat et lui autorise à exécuter les opérations du service en fonction de la capacité incluse dans le certificat. Le protocole de sécurité que nous avons proposé est à base de cryptographie asymétrique pour permettre de faire de la signature et pour réduire le nombre de clés dans le système. Les capacités permettent de ne pas maintenir dans la carte des listes de contrôle d'accès pour chaque service.

Encapsulation de l'exécution des objets Nous utilisons une machine virtuelle sécurisée comme support d'exécution des services. Cette machine virtuelle définit quatre registres de sécurité qui limitent physiquement l'accès à la mémoire, au code et aux données propres à l'objet qui s'exécute. Il y a ainsi étanchéité totale entre les différents services chargés dans la carte. Le système d'exploitation charge les services de la carte et maintient dans une table les valeurs de ces quatre registres correspondant au code et aux données de cet objet. Avant une exécution, il positionne ces registres sur l'espace réservé au service dans la machine virtuelle et lance l'interpréteur qui contrôle s'il y a,à chaque accès mémoire, violation des zones réservées au service. L'interpréteur est implémenté pour exécuter un langage de codé filé selon la méthode *direct token* qui a les avantages d'être efficace, écrit en C (et donc

plus facilement portable) et de produire du code compact. Un prototype du système d'exploitation, appelé *CoBO*, avec la machine virtuelle sécurisée, a été implémenté.

Accès dynamique aux objets Nous utilisons un langage de description des objets indépendant de leur implémentation pour mettre à jour un serveur générique de cartes appelé *Carte Object Adapter*. Le serveur utilise cette description pour créer un représentant du service carte dans le système d'information de l'application cliente. Le client voit ce représentant comme un objet « classique » de son système et lui adresse ses requêtes. Le représentant s'occupe de la transcription des requêtes objet en commandes carte en utilisant la description de l'interface pour vérifier sa syntaxe et, éventuellement, un langage de scripts pour paramétrer cette transcription. Un prototype de COA dans l'architecture CORBA Orbix a été implémenté. Il est générique et dynamique et offre ainsi une couche middleware d'intégration de tout service carte dans les architectures CORBA.

8.3 Perspectives

Les perspectives de recherche qui peuvent dès lors être tracées sont au moins de deux ordres.

D'une part, il serait intéressant de continuer le travail sur le système d'exploitation *CoBO* afin d'augmenter la qualité des services que la carte générique peut contenir. On songe, par exemple, à :

- l'ajout de nouveaux types de bases,
- l'agrégation et la composition d'objets pour décrire de façon plus riche les services,
- la création d'objets (ce qui signifie allocation de mémoire) par des services,
- l'accès à des ressources système (par exemple, des fichiers), et
- l'utilisation de bibliothèques d'objets de haut niveau (par exemple, pour des calculs cryptographiques).

Le principal problème à résoudre avec de tels ajouts sera bien sûr de conserver sans trop de pénalités le même niveau de sécurité qu'aujourd'hui où les zones mémoire de chaque objet sont statiques et où les objets sont hermétiquement cloisonnés les uns par rapport aux autres. Ce problème devient encore plus difficile si l'on veut permettre, à des services qui ne se connaissent pas *a priori*, de coopérer en se partageant des données.

D'autre part, le travail sur le serveur générique de cartes doit pouvoir être intégré dans un cadre beaucoup plus global dans lequel seraient abordés les problèmes de sécurité entre une application cliente et un service carte en intégrant au COA des services permettant :

- de prendre en charge le protocole de sécurité que nous avons défini de la manière la plus transparente possible pour le client, et
- d'assurer la cohérence des transactions menées par un client et un service carte (ou plusieurs services carte) avec d'autres services.

Quatrième partie

Annexes

ANNEXE A

Le système d'exploitation C_0Mb^O

« Vous m'avouerez que vos norvégiens, tous médaillés qu'ils sont, ils n'ont inventé ni le TGV, ni le Concorde, ni la carte à puce. Loin s'en faut! »

Marionnette de Thierry Roland in Les Guignols de l'Info, Canal+, 1994.

Résumé

Cette annexe décrit le rôle et le fonctionnement des commandes du système d'exploitation disponible à l'interface de la carte C_0Mb^O. Ces commandes définissent le système d'exploitation de la carte. Elles permettent d'authentifier les utilisateurs de la carte et de gérer les services que la carte contient.

Sommaire

Tables

Figures

A.1 Architecture générale

Ce chapitre décrit l'implémentation actuelle du système d'exploitation C_{OS} tel qu'il a été implémenté sous forme d'un simulateur tournant sur Sparcstation Solaris 2.4. Il a aussi été implémenté pour le projet CASCADE sur ARMulator pour Windows NT, un programme qui émule le jeu d'instructions pour le processeur ARM 7.

A.1.1 Interface utilisateur

L'interface utilisateur (*cf.* FIG. A.1) reçoit les commandes, les analyse et les fait exécuter par les modules du système d'exploitation de la carte.

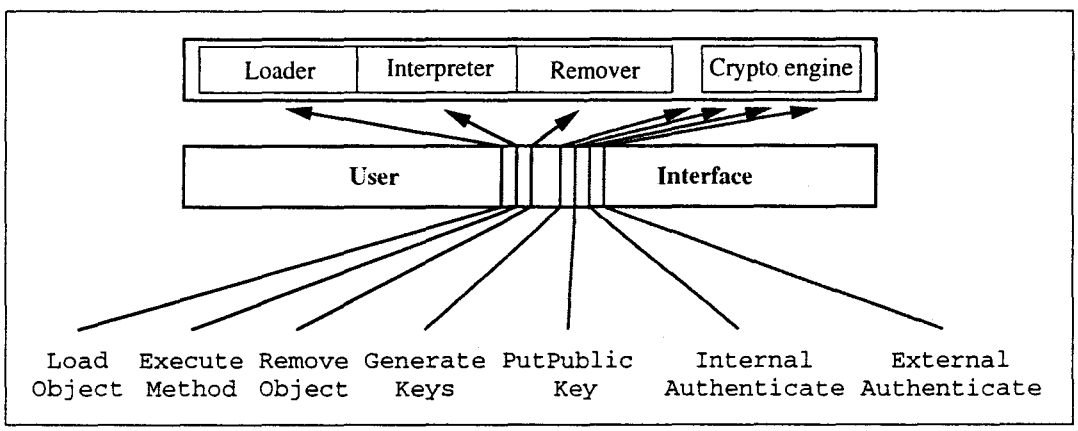

FIG. A.1 - Interface utilisateur du système d'exploitation C_{OS}

Trois commandes sont dédiées à la gestion des services carte :

- `LoadObject`
- `RemoveObject`
- `ExecuteMethod`

Quatre commandes sont dédiées à la gestion de la sécurité :

- `GenerateKeys`
- `PutPublicKey`
- `InternalAuthenticate`
- `ExternalAuthenticate`

Ces commandes seront décrites sous leur format APDU (*cf.* § 2.1.2.4 page 25).

A.1.2 Tables système

Le système d'exploitation utilise essentiellement deux tables système pour maintenir les informations concernant les prestataires qui ont chargé des services (*cf.* TABLE A.1), et les services qui ont été chargés dans la carte (*cf.* TABLE A.2).

Issuer Reference	Public Key
Identification unique du prestataire de services	Clé publique du prestataire de services

TABLE A.1 - Table système des prestataires de services

Object Reference	Issuer Reference	Base DS	Limit DS	Base CS	Limit CS	MethodList (0..15)
Identification unique du service	Prestataire ayant chargé le service	Début de la zone de données de l'objet	Fin de la zone de données de l'objet	Début de la zone de code de l'objet	Fin de la zone de code de l'objet	Adresses des 16 points d'entrées du service (opérations de l'objet)

TABLE A.2 - Table système des services

A.2 Commandes pour la gestion de la sécurité

A.2.1 Lors de la personnalisation

Parmi les quatres commandes dédiées à la gestion de la sécurité, deux sont utilisées lors de la personnalisation de la carte (*cf.* FIG. A.2) :

- `GenerateKeys`
- `PutPublicKey`

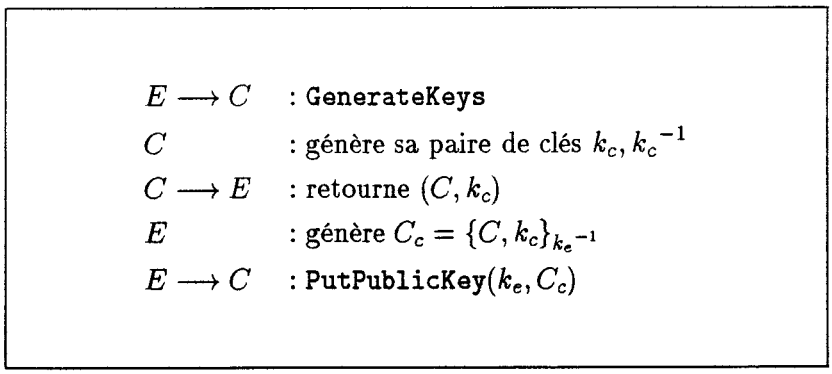

FIG. A.2 - Commandes de sécurité lors de la personnalisation

À la fin de l'étape de personnalisation, la carte stocke de façon inaltérable sa paire de clés publique et privée k_c et k_c^{-1}, la clé publique de l'émetteur k_e et son certificat C_c.

A.2.1.1 Commande GenerateKeys

CLA	INS	P1	P2	Lc	Din	Le
00	F6	00	00	00	–	Taille de (C, k_c)

TABLE A.3 - Format APDU de la commande GenerateKeys

Dout	SW1	SW2
C, k_c	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: unable to perform operation (card problem)	67	50

TABLE A.4 - Format APDU de la réponse à la commande GenerateKeys

A.2.1.2 Commande PutPublicKey

CLA	INS	P1	P2	Lc	Din	Le
00	F8	00	00	Taille de (k_e, C_c)	(k_e, C_c)	00

TABLE A.5 - Format APDU de la commande PutPublicKey

Dout	SW1	SW2
–	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: unable to perform operation (card problem)	67	60

TABLE A.6 - Format APDU de la réponse à la commande PutPublicKey

A.2.2 Lors de l'authentification

Les deux autres commandes sont utilisées lors de l'authentification d'un prestataire de services ou d'un client :

- **InternalAuthenticate**
- **ExternalAuthenticate**

A.2.2.1 Authentification d'un prestataire de services

L'authentification d'un prestataire de services est décrite par FIG. A.3 page suivante.

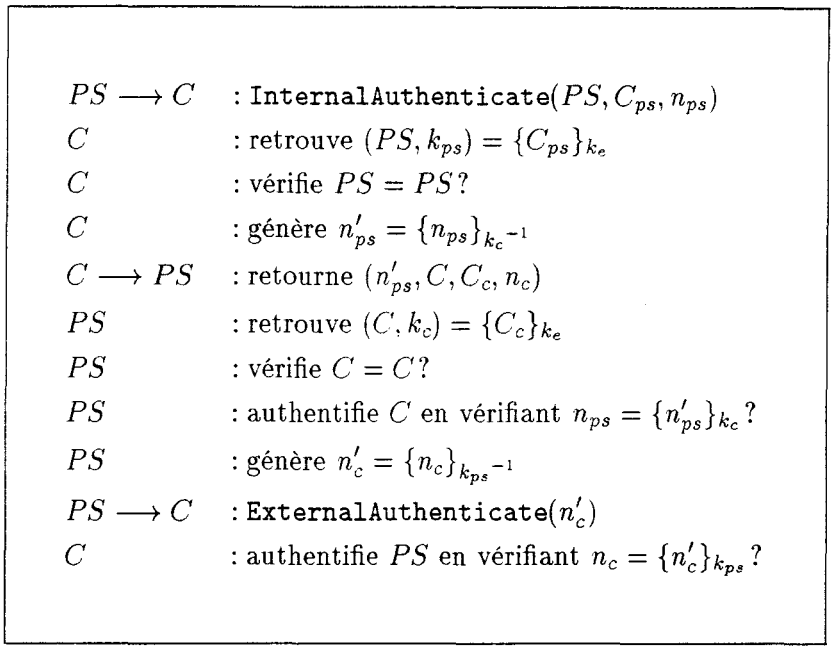

FIG. A.3 - Commandes de sécurité lors de l'authentification d'un prestataire de services

Cette opération permet aussi au prestataire de services d'authentifier la carte comme une carte émise par l'émetteur. À la fin de cette authentification le système d'exploitation ajoute PS et k_{ps} dans sa table des prestataires de services. Si l'authentification échoue ou si le prestataire de services existait déjà la table n'est pas modifiée.

Commande `InternalAuthenticate` pour un prestataire de services Ci-après le format APDU de la commande `InternalAuthenticate` (et de sa réponse) quand elle est utilisée lors de la phase d'authentification d'un prestataire de services.

CLA	INS	P1	P2	Lc	Din	Le
00	FA	01	00	Taille de (PS, C_{ps}, n_{ps})	(PS, C_{ps}, n_{ps})	Taille de (n'_{ps}, C, C_c, n_c)

TABLE A.7 - Format APDU de la commande `InternalAuthenticate` pour un prestataire de services

Dout	SW1	SW2
(n'_{ps}, C, C_c, n_c)	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: wrong expected length (Le)	67	17
Error: unable to perform operation (card problem)	67	70
Error: authentication failed (wrong certificate)	67	71

TABLE A.8 - Format APDU de la réponse à la commande `InternalAuthenticate` pour un prestataire de services

Commande `ExternalAuthenticate` pour un prestataire de services Ci-après le format APDU de la commande `ExternalAuthenticate` (et de sa réponse) quand elle est utilisée lors de la phase d'authentification d'un prestataire de services. Cette commande doit toujours suivre la commande `InternalAuthenticate`.

CLA	INS	P1	P2	Lc	Din	Le
00	FC	01	00	Taille de (n'_c)	(n'_c)	00

TABLE A.9 - Format APDU de la commande `ExternalAuthenticate` pour un prestataire de services

Dout	SW1	SW2
-	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: unable to perform operation (card problem)	67	80
Error: authentication failed (wrong cryptogram)	67	81

TABLE A.10 - Format APDU de la réponse à la commande `ExternalAuthenticate` pour un prestataire de services

A.2.2.2 Authentification d'un client

L'authentification d'un client est décrite par FIG. A.4.

$CL \longrightarrow C$: `InternalAuthenticate`$(CL, PS, C_{cl/ps}, n_{cl})$
C : vérifie $PS \in$ table des prestataires de services?
C : retrouve k_{ps}
: retrouve $(CL, k_{cl}, PS, ObjRef, Mask_{ObjRef_{cl/ps}}) = \{C_{cl/ps}\}_{k_{ps}}$
C : vérifie $CL = CL$?
C : vérifie $(PS, ObjRef) \in$ table des services?
C : génère $n'_{cl} = \{n_{cl}\}_{k_c{}^{-1}}$
$C \longrightarrow CL$: retourne (n'_{cl}, C, C_c, n_c)
CL : retrouve $(C, k_c) = \{C_c\}_{k_e}$
CL : vérifie $C = C$?
CL : authentifie C en vérifiant $n_{cl} = \{n'_{cl}\}_{k_c}$?
CL : génère $n'_c = \{n_c\}_{k_{cl}{}^{-1}}$
$CL \longrightarrow C$: `ExternalAuthenticate`(n'_c)
C : authentifie CL en vérifiant $n_c = \{n'_c\}_{k_{cl}}$?

FIG. A.4 - Commandes de sécurité lors de l'authentification d'un client

Cette opération permet aussi au client d'authentifier la carte comme une carte émise par l'émetteur. À la fin de cette authentification le client est autorisé à exécuter les opérations listées dans son masque d'autorisation $Mask_{ObjRef_{cl/ps}}$ pour l'objet $ObjRef$. Les points d'entrée de chacune des opérations sont retrouvés dans la carte dans sa table des services.

Commande `InternalAuthenticate` pour un client Ci-après le format APDU de la commande `InternalAuthenticate` (et de sa réponse) quand elle est utilisée lors de la phase d'authentification d'un client.

CLA	INS	P1	P2	Lc	Din	Le
00	FA	02	00	Taille de $(CL, PS, C_{cl/ps}, n_{cl})$	$(CL, PS, C_{cl/ps}, n_{cl})$	Taille de (n'_{cl}, C, C_c, n_c)

TABLE A.11 - Format APDU de la commande `InternalAuthenticate` pour un client

Dout	SW1	SW2
(n'_{cl}, C, C_c, n_c)	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: wrong expected length (Le)	67	17
Error: unable to perform operation (card problem)	67	70
Error: authentication failed (wrong certificate)	67	71

TABLE A.12 - Format APDU de la réponse à la commande `InternalAuthenticate` pour un client

Commande `ExternalAuthenticate` pour un client Ci-après le format APDU de la commande `ExternalAuthenticate` (et de sa réponse) quand elle est utilisée lors de la phase d'authentification d'un client. Cette commande doit toujours suivre la commande `InternalAuthenticate`.

CLA	INS	P1	P2	Lc	Din	Le
00	FC	02	00	Taille de (n'_c)	(n'_c)	00

TABLE A.13 - Format APDU de la commande `ExternalAuthenticate` pour un prestataire de services

Dout	SW1	SW2
-	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: unable to perform operation (card problem)	67	80
Error: authentication failed (wrong cryptogram)	67	81

TABLE A.14 - Format APDU de la réponse à la commande `ExternalAuthenticate` pour un client

A.3 Commandes pour la gestion des services carte

Les opérations disponibles à l'interface de la carte pour gérer les services réalisent trois fonctions :

`LoadObject` Permet à un prestataire de services authentifié de charger un service dans la carte. Cette commande est prise en charge par un module du système d'exploitation appelé *Loader*.

`RemoveObject` Permet à un prestataire de services de retirer un service de la carte. Le prestataire de services doit être authentifié comme étant le propriétaire du service qui doit être retiré de la carte. Cette commande est prise en charge par un module du système d'exploitation appelé *Remover*.

`ExecuteMethod` Permet à un client d'exécuter une opération d'un service. Le client doit être authentifié et doit avoir les droits d'accès pour exécuter l'opération demandée. Cette commande est prise en charge par un module du système d'exploitation appelé *Executer*.

Le chargement et le retrait de services dans la carte sont basés sur une gestion dynamique de la mémoire non volatile qui permet au système d'exploitation d'allouer et de libérer de la mémoire en fonction de ses besoins. Ce gestionnaire de mémoire a été réalisé par Biget (Patrick) dans le cadre du projet CASCADE.

A.3.1 Chargement de services

A.3.1.1 Principe

Pour charger un service dans la carte, il faut d'abord être authentifié comme prestataire de services (*cf.* FIG. A.5).

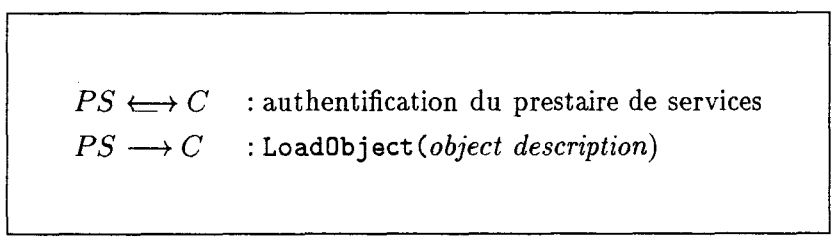

FIG. A.5 - Chargement de services

Les informations transmises à la carte lors de cette opération constitue la description du service (*object description*) qui contient :

- l'*ObjectRef*, identifiant unique du service appartenant au prestataire de services préalablement authentifié,
- les valeurs initiales des données du service, et
- le code des opérations du service sous forme de mots de l'interpréteur (*tokens*).

Le *Loader* ajoute les données du service dans le *DataSpace*, ajoute le code des opérations du service dans le *CodeSpace* et met à jour la table système des services.

A.3.1.2 Object Description

La description d'un service (*object description*) se présente dans un format de structure TLV[1] qui nomme chacune des informations du service par un identifiant unique (*Tag*), spécifie sa longueur (*Length*) et enfin fournit sa valeur (*Value*).

```
Header          = TagHeader + LengthHeader + ValueHeader
TagHeader       = 01
LengthHeader    = 4
ValueHeader     = ObjectRef
DataSpace       = TagDataSpace + LengthDataSpace + ValueDataSpace
TagDataSpace    = 02
LengthDataSpace = service data length
ValueDataSpace  = service data
CodeSpace       = TagCodeSpace + LengthCodeSpace + ValueCodeSpace
TagCodeSpace    = 03
LengthCodeSpace = number of operations of the service
ValueCodeSpace  = TagMethod + LengthMethod + ValueMethod
TagMethod       = 'A' (anonymous) ou 'P' (public)
LengthMethod    = number of tokens of the method code
ValueMethod     = token list of the method code
```

FIG. A.6 - Description TLV d'un service chargé dans la carte

Les opérations du service déclarées anonymes ne sont pas ajoutées dans la liste des points d'entrée du service et ne sont accessibles qu'à l'intérieur du service même. Elles servent à factoriser du code pour les opérations du service et sont inaccessibles directement aux clients du service.

A.3.1.3 Commande `LoadObject`

CLA	INS	P1	P2	Lc	Din	Le
00	F0	00	00	object length	object description (TLV)	00

TABLE A.15 - Format APDU de la commande `LoadObject`

1. Tag Length Value

Dout	SW1	SW2
–	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: wrong object header	67	20
Error: wrong dataspace	67	21
Error: wrong codespace	67	22
Error: invalid TLV structure	67	23
Error: call to unexisting external method	67	25
Error: call to unauthorised external method	67	26
Error: call to external object not present in card	67	27
Error: object already present in card (with the same ObjectRef)	67	28
Error: maximum allowed number of object reached	67	29

TABLE A.16 - Format APDU de la réponse à la commande `LoadObject`

A.3.2 Retrait de services

A.3.2.1 Principe

Pour retirer un service dans la carte il faut d'abord être authentifié comme prestataire de services (*cf.* FIG. A.7).

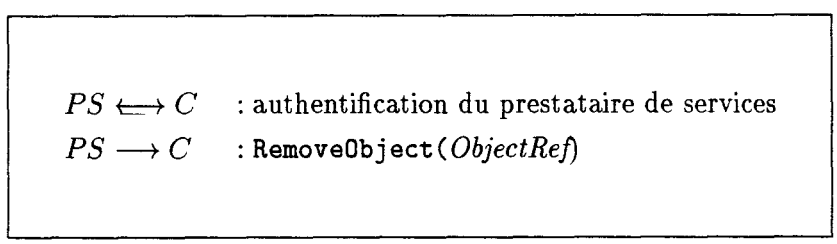

FIG. A.7 - Retrait de services

La seule information nécessaire au retrait d'un service est l'identifiant du service (*ObjectRef*). Le *Remover* vérifie que le service existe et appartient bien au prestataire de services qui s'est préalablement authentifié. Il libère alors la zone du *DataSpace* occupée par les données du service et la zone du *CodeSpace* occupée par les opérations du service. Enfin, il retire l'entrée correspondante au service de la table système des services.

A.3.2.2 Commande `RemoveObject`

CLA	INS	P1	P2	Lc	Din	Le
00	F2	00	00	04	ObjectRef	00

TABLE A.17 - Format APDU de la commande `RemoveObject`

Dout	SW1	SW2
–	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: wrong expected length (Le)	67	17
Error: unknown object reference (object not present in card)	67	30

TABLE A.18 - Format APDU de la réponse à la commande `RemoveObject`

A.3.3 Exécution de services

A.3.3.1 Principe

Pour exécuter une opération d'un service dans la carte il faut d'abord être authentifié comme client de ce service (*cf.* FIG. A.8).

$PS \longleftrightarrow C$: authentification du client

$PS \longrightarrow C$: `ExecuteMethod`
(*ObjectRef, IssuerRef, MethodNumber, MethodInputParameters*)

FIG. A.8 - Exécution de services

Les informations nécessaires à l'exécution d'une opération d'un service sont :

- l'identifiant du service fourni par l'*ObjectRef* et l'*IssuerRef* qui permettent de retrouver le service dans la table système des services,
- le numéro de l'opération à exécuter (*MethodNumber*) qui permet de retrouver le point d'entrée (adresse) de l'opération dans le *CodeSpace*, et
- les paramètres en entrée de l'opération.

L'*Executer* vérifie que le service existe et que le client préalablement authentifié possède bien les droits d'accès (*Mask*) sur l'opération qu'il souhaite exécuter. Il récupére ensuite les paramètres en entrée pour les stocker sur la pile de données (*DataStack*) de l'interpréteur. Il positionne alors le pointeur d'instructions de l'interpréteur sur le point d'entrée de l'opération. Il positionne aussi les registres de sécurité `BDS`, `LDS`, `BCS` et `LCS` sur les zones de données et de codes accessibles au cours de l'exécution de l'opération puis il lance l'interpréteur $C_{o}^{\not M}\not O$ qui exécutera le code de cette opération sur la machine virtuelle définie Chapitre 6 page 105. Enfin, le résultat de l'interprétation de l'opération (erreur ou succès) est récupéré par l'*Executer* qui est renvoyé au client avec les valeurs de retour positionnées par l'interpréteur sur la *DataStack*.

A.3.3.2 Commande `ExecuteMethod`

CLA	INS	P1	P2	Lc	Din	Le
00	F2	00	00	09 + Length of MethodInputParameters	*ObjectRef*, *IssuerRef*, *MethodNumber*, *MethodInputParameters*	00

TABLE A.19 - Format APDU de la commande `ExecuteMethod`

Dout	SW1	SW2
–	90	00
Error: wrong class byte (CLA)	67	10
Error: wrong instruction code (INS)	67	11
Error: wrong parameters (P1-P2)	67	15
Error: wrong length (Lc)	67	16
Error: unkwon command (wrong method number)	67	41
Error: Data Stack underflow	67	42
Error: Data Stack overflow	67	43
Error: Return Stack underflow	67	44
Error: Return Stack overflow	67	45
Error: security violation (access attempts to unauthorised code or data)	67	46
Error: unknown object reference (object not present in card)	67	47

TABLE A.20 - Format APDU de la réponse à la commande `ExecuteMethod`

ANNEXE B

Présentation du langage Forth

« *Apprendre un langage, c'est apprendre en même temps que ce langage sera payant dans telle ou telle situation.* »

Pierre Bourdieu. *Ce que parler veut dire, in* Questions de sociologie, Les Éditions de Minuit, Paris, France, 1984.

Résumé

Cette présentation du langage Forth est presque intégralement traduite de la brève introduction à Forth de Philip J. Koopman Jr.[1]

Sommaire

B.1 Une machine abstraite à deux piles

Forth se présente comme un langage interprété pour une machine abstraite à pile. La machine abstraite possède au minimum un compteur d'instructions, une mémoire, une

1. Philip J. Koopman Jr., United Technologies Research Center, East Hartford, CT.

unité arithmétique et logique, une pile d'évaluation de données, et une pile d'adresses de retour de sous-routines.

Les traitements sur les données sont réalisés sur la pile de données en utilisant la notation polonaise inverse. Par exemple, la séquence suivante tapée au clavier :

```
3 4 + 5 * .     35 ok
```

met de façon interactive la valeur 3 sur la pile, puis la valeur 4 au dessus de 3, ajoute en les retirant 4 et 3 pour obtenir 7 qui est multiplié par 5. L'opération . affiche le résultat qui se trouve alors en sommet de pile, soit 35. « ok » est traditionnellement le prompt de la machine Forth. Des opérations telles que SWAP (échange) et DUP (réplication) réordonnent et dupliquent les éléments en sommet de la pile de données.

B.2 Sous-routines

À un niveau plus profond, les programmes Forth utilisent la notation polonaise inverse comme un moyen d'obtenir une syntaxe simple et des composants modulaires. Pour réaliser des opérations complexes, on écrit des petits programmes simples qui réutilisent des séquences de code communes.

Par exemple, la fonction suivante fait la somme des carrés de deux entiers en sommet de pile de données et laisse le résultat sur le sommet de pile :

```
:SUM-OF-SQUARES  ( a b -- c )   DUP *  SWAP  DUP *  + ;
```

À l'exécution du mot SUM-OF-SQUARES la pile de données doit contenir deux entiers a et b. À la fin de l'exécution elle n'en contient plus qu'un qui est c. Le caractère : signifie qu'on définit une fonction de nom SUM-OF-SQUARES. Le caractère ; termine la définition. Les commentaires sont inclus entre les parenthèses. Cet exemple suit la convention de codage des commentaires en montrant les effets sur la pile de l'exécution de la fonction. Le second élément a de la pile et le sommet b sont consommés comme les paramètres en entrée sur la pile, et c est le résultat produit en sommet de pile.

Ce premier programme peut être réécrit en définissant une nouvelle fonction SQUARED qui permet de partager les opérations communes de duplication et de multiplication d'un nombre en sommet de pile. La séparation entre la pile de données et la pile de retour dans la machine abstraite permet de passer proprement les valeurs sur la pile de données à travers plusieurs niveaux de sous-routines sans overhead à l'exécution. Dans cette nouvelle version, les éléments de la piles de données sont implicitement passés comme paramètres depuis SUM-OF-SQUARES vers SQUARED :

```
: SQUARED  ( n -- n**2 )   DUP * ;
: SUM-OF-SQUARES  ( a b -- c )  SQUARED SWAP SQUARED + ;
```

Un bon programmeur Forth doit s'efforcer d'écrire des programmes contenant des très petits segments de codes réutilisables et bien documentés. Généralement, un programme Forth fait plus d'appels de sous-routines que d'opérations sur la pile. Pour augmenter la vitesse d'exécution il est aussi fréquent d'écrire certaines fonctions, souvent utilisées, directement en assembleur.

Écrire un programme Forth est équivalent à étendre le langage pour inclure toutes les fonctions nécessaires à l'implémentation d'une application. De fait, programmer en Forth peut être vu comme créer des extensions au langage spécifiques à l'application. Ce

paradigme, associé à un cycle très rapide d'écriture/compilation/test permet d'augmenter de façon signicative la productivité. Chaque fois qu'une fonction est écrite, elle peut être directement testée au clavier. Par exemple avec les fonctions ci-dessus :

```
3 SQUARED .     9 ok
3 4 SUM-OF-SQUARES .     25 ok
```

B.3 Interprétation, compilation et exécution

Les systèmes Forth utilisent deux niveaux d'interprétation : un interpréteur de texte et un interpréteur d'adressage. Quand on entre un texte au clavier ou qu'on donne un fichier à lire, l'interpréteur de texte extrait les chaînes de caractères séparées par des espaces. En mode d'interprétation il tente d'exécuter le *mot* correspondant à la chaîne de caractères (les entrées numériques font l'objet d'un traitement spécifique afin d'être stockées sur la pile de données). : est un mot qui crée une nouvelle entrée dans le *dictionnaire* avec le nom de la fonction et qui place l'interpréteur de texte en mode compilation. Pendant le mode compilation, la plupart des mots extraits du flux d'entrée ne sont pas interprétés mais sont compilés dans le dictionnaire sous la forme de pointeurs vers des définitions de mots du dictionnaire.

Un programme Forth compilé est un ensemble de mots, chacun d'entre eux contient une liste de pointeurs vers d'autres mots. En bout de course, les pointeurs aboutissent sur les primitives de base du langage écrites en assembleur. L'interpréteur d'adressage Forth est utilisé pour exécuter les mots compilés en utilisant de façon classique les techniques de « codes filés » (threaded code). L'interpréteur de texte Forth, bien que non utilisé à l'exécution de programmes compilés, est souvent inclu dans les applications pour servir de base à une interface utilisateur en mode ligne.

Les systèmes Forth utilisent une compilation en une passe. Il n'y a pas à proprement parler de parser Forth et, *a fortiori*, de grammaire formelle. Les mots de contrôle de flux ont un attribut spécial *immediate*, et sont exécutés immédiatement même si l'interpréteur de texte est en mode compilation. Les mots immédiats, lorsqu'ils sont exécutés, provoquent généralement la compilation de structures spéciales. Par exemple, `IF` compile un branchement conditionnel sur la valeur du sommet de la pile de retour à l'exécution, et `THEN` (le mot pour « endif » !) met à jour l'adresse de saut. Les utilisateurs peuvent aisément créer leur propres mots immédiats, et ainsi étendre le compilateur en ajoutant des nouvelles instructions de contrôles ou d'autres caractéristiques au langage.

Les structures de données sont créées par une autre classe spéciale de mots : *les mots de définition*. Les mots de définition ont deux parties : la clause `CREATE` crée une entrée dans le dictionnaire pour l'instance de la structure de données, tandis que la clause `DOES` est une définition partagée par toutes les structures de données créées par ce mot de définition. Par exemple, un mot de définition d'un tableau crée un tableau avec un nom et réserve un espace mémoire pour le tableau avec sa clause `CREATE`, et calcule l'adresse d'un élément à partir de son indice dans sa clause `DOES`. Les mots de définition sont couramment utilisés pour cacher les implémentations de structures de données et pour créer des familles de mots similaires.

ANNEXE C

Glossaire

Les définitions ci-dessous précisent l'emploi des termes importants de ce mémoire. Elles correspondent le plus possible aux définitions couramment rencontrées dans la littérature. La table des acronymes est fournie afin de lever les ambiguïtés potentielles que ceux-ci peuvent parfois provoquer.

C.1 Définitions

Approche Orientée Objet Mécanisme d'abstraction dans lequel le monde est modélisé comme une collection d'objets indépendants qui communiquent les uns avec les autres par échange de messages.

Carte générique Carte à microprocesseur capable au cours de son cycle de vie de charger et décharger de nouvelles fonctionnalités. Ces fonctionnalités sont regroupées sous la forme de services comprenant des données et des traitements s'effectuant sur ces données. Ces traitements apparaissent comme des nouvelles commandes à l'interface de la carte.

Carte sans contacts Carte communiquant avec un lecteur *via* un signal électromagnétique qui fournit l'alimentation de la carte et transmet commandes et données. Ces cartes disposent d'un circuit analogique en plus de leur circuit logique et de leurs mémoires. L'antenne est encapsulée dans le support plastique. Elles sont par exemple utilisées dans les systèmes de péage et de « billetique » (tickets pour les transports en commun) où les utilisateurs passent devant un lecteur sans avoir à connecter leur carte.

Carte à logique câblée Ces cartes utilisent de la mémoire EPROM ou EEPROM. Les accès à la mémoire sont protégés par des circuits câblés spécifiques aux besoins de l'application. Elles sont par exemple utilisées comme cartes de contrôle d'accès ou comme cartes prépayées rechargeables.

Carte à mémoire Ces cartes servent de mémoire « porte-jetons ». Elles contiennent un code applicatif simple permettant de lire et écrire en mémoire sans contrôle d'accès. Certaines zones peuvent être protégées par

grillage d'un fusible après programmation pour stocker les informations de l'émetteur. La réécriture étant impossible elles sont par exemple utilisées comme cartes prépayées pour le téléphone ou les places de stationnement.

Carte à microprocesseur Cartes contenant généralement un microprocesseur 8 bits avec son système d'exploitation gravé en mémoire ROM (d'une capacité allant jusqu'à 16 KO) et une mémoire de travail RAM entre 96 et 512 octets. La mémoire non volatile est du type EEPROM avec une capacité comprise entre 1 et 16 KO. Le code du système d'exploitation gère la communication avec le monde extérieur, les accès en lecture, écriture et réécriture en mémoire et effectue des calculs algorithmiques permettant de contrôler en interne toutes les opérations. Un grand nombre d'applications est possible avec ces cartes.

Carte à puce Terme générique pouvant désigner aussi bien une carte à mémoire, qu'une carte à logique câblée ou encore une carte à microprocesseur.

Client-Serveur Modèle d'architecture pour les systèmes distribués dans lequel les ressources partagées sont gérées par des *serveurs*. Les *clients* émettent des requêtes à ces serveurs chaque fois qu'ils ont besoin d'accéder à des ressources. Si la requête est correcte, le serveur exécute l'action demandée et retourne la réponse au client. Un serveur peut aussi accéder à des ressources gérées par d'autres serveurs ; dans ce cas, il est à la fois serveur et client.

Code filé Code interprété écrit dans un jeu d'instructions où chaque instruction est l'adresse du code à exécuter pour réaliser l'opération. Cette adresse peut pointer sur du code natif ou sur d'autres codes filés. Ces langages sont appelés en anglais *Threaded Interpretive Languages* (TIL).

Interopérabilité Des systèmes sont dits interopérables si un programme sur un système peut accéder aux programmes et données des autres systèmes. En terme de réseau, l'interopérabilité peut se définir comme la capacité à véhiculer une information d'un point à un autre, avec des matériels et des débits différents, sans qu'il y ait dégradation de cette information.

Legacy Systems Le terme anglais « legacy systems » (difficilement traduisible en français; peut-être par « patrimoine applicatif») désigne de façon générique les systèmes informatiques du passé devenus difficiles à gérer et à maintenir, soit par leur support technologique (cartes perforées, machines dédiées et inaltérables, gros programmes COBOL), soit par leur envergure, soit par leur incapacité à évoluer vers les nouvelles technologies ou à s'intégrer dans les architectures actuelles.

Middleware Ensemble de services distribués possédant des interfaces de programmation et protocoles standards. Ces services sont appelés

middleware parce qu'ils se situent « au milieu » (*"in the middle"*), au dessus du système d'exploitation et du réseau et sous les applications.

Moniteur de référence Noyau de sécurité d'un système informatique qui contrôle que chaque accès à un objet du système est conforme avec la politique de sécurité définie.

Politique de sécurité « *Une politique de sécurité est un ensemble de règles qui spécifient les procédures et mécanismes requis pour maintenir la sécurité d'un système, ainsi que les objets de sécurité et les sujets de sécurité sur lesquels porte la politique.* » (*Security in Open Systems: a Security Framework.* – ECMA (European Computer Manufacturers Association), TR/46, juillet 1988).

Système à Grande Échelle Système informatique implémentant une application de grande envergure au niveau du nombre d'utilisateurs, de ressources mises en œuvre et de surface géographique couverte. Un tel système est principalement caractérisé par la grande diversité des ressources informatiques (matériel, réseau, logiciel) à faire coopérer et interagir pour le bon fonctionnement de l'application.

Technologies de l'Information Ensemble des technologies capables de produire, transporter ou utiliser de l'information. Autrefois assez indépendantes les unes des autres, aujourd'hui de plus en plus intégrées les unes aux autres grâce au langage commun du numérique, le développement des communication haut-débit et l'utilisation de normes de codage.

C.2 Acronymes

ABS Acrylonitrile Butadiene Styrene. Plastique utilisé pour la fabrication du corps des cartes.

APDU Application Protocol Data Unit (ISO 7816-4).

API Application Programming Interface.

ASCII American Standard Code for Information Interchange.

BD Base de données.

BOA Basic Object Adapter (OMG).

BPS Bits Par Seconde (Aussi appeé « bauds »).

CAPI Card Application Programming Interface.

CASCADE Chip Architecture for Smart CArds and portable intelligent DEvices (projet ESPRIT de la Comission Européenne faisant partie du programme de l'Open Microprocessor Initiative).

CCR Card Code Repository (*Infra*).

CERN Conseil Européen pour la Recherche Nucléaire. En fait, désigne aujourd'hui le laboratoire européen de physique des particules.

CGI Common Gateway Interface (Internet).

CLI Call Level Interface.

CNET Centre National d'Études des Télécommunications.

COA Card Object Adapter (*Infra*).

CORBA Common Object Request Broker Architerture (OMG).

COS Card Operating System (Gemplus).

CQL Card Query Language (Gemplus).

CRC Code de Redondance Cyclique ou Cyclic Redundancy Check en anglais.

CRTS Centre Régional de Transfusion Sanguine.

CS Code Space (Machine virtuelle C_oMb^O).

CSE Card Software Environment (Gemplus).

DDASS Direction Départementale des Affaires Sanitaires et Sociales.

DCE Distributed Computing Environment (OSF).

DDL Data Definition Language (SQL).

DEA Diplôme d'Études Appronfondies.

DES Data Encryption Standard.

DF Dedicated File (ISO 7816-4).

DGT Direction Générale des Télécommunications. Ancien nom de France Telecom.

DII Dynamic Invocation Interface (OMG).

DIS Draft International Standard (ISO).

DLL Dynamic Link Library (Microsoft).

DML Data Manipulation Language (SQL).

DS Data Stack (Machine virtuelle C_oMb^O).

DSI Dynamic Skeleton Interface (OMG).

EF Elementary File (ISO 7816-4).

EMV Europay Mastercard Visa. Initiales des trois promoteurs de ce système de transactions financières avec cartes à microprocesseur.

EPROM Electrically Programmable Read-Only Memory.

EEPROM Electrically Erasable and Programmable Read-Only Memory.

FRAM Ferroelectric Random Access Memory.

GCR Gemplus Card Reader (Gemplus).

GIE Groupement d'Intérêt Économique.

GSM Global System for Mobile communication.

HTML HyperText Markup Language (Internet).

HTTP HyperText Transfert Protocol (Internet).

IDL Interface Description Language (OMG).

IP Instruction Pointer (Machine virtuelle C_oMb^O).

IR Interface Repository (OMG).

IS International Standard (ISO).

ISO International Standards Organization.

JDBC Java DataBase Connectivity (Sun).

KO Kilo-Octets.

MAM Microcalculateur Autoprogrammable Monolithique. Appelé SPOM en anglais : Self-Programming One-chip Microprocessor.

Mb Megabit.

MF Master File (ISO 7816-4).

MIME Multi-purpose Internet Mail Extension (Internet).

MMU Memory Management Unit.

NCSA National Center for Supercomputing Applications.

NAS Network Application Support (Digital).

NIP Numéro d'Identification Personnel.

ODBC Open DataBase Connectivity (Microsoft).

OMA Object Management Architecture (OMG).

OMG Object Management Group.

ORB Object Request Broker (Object Request Broker).

OSF Open Software Foundation.

OSI Open Systems Interconnection. (ISO)

OSMOSE Operating System and MObile SErvices (*Infra*).

PME Porte-Monnaie Électronique.

PVC Polyvinyl Chloride. Plastique utilisé pour la fabrication du corps des cartes.

PTS Protocole Type Selection (ISO 7816-3).

RAM Random Access Memory.

RFC Request For Comments.

ROM Read-Only Memory.

RISC Reduced Instruction Set Computer.

RS Return Stack (Machine virtuelle C_oMb^O).

RSA Rivest Shamir Adelman. Initiales des trois auteurs de ce système de cryptographie asymétrique.

SAG X/Open SQL Access Group.

SCAM Serveur de Carte À Mémoire (Projet Carte Santé du Québec).

SGBD Système de Gestion de Base de Données.

SIM Subscriber Identification Module (GSM).

SQL Structured Query Language.

TCP Transport Communication Protocol (Internet).

TIL Threaded Interpretive Languages.

TLV Tag Length Value.

TOS Top Of Stack (Machine virtuelle C_oMb^O).

TPDU Transport Protocol Data Unit (ISO 7816-4).

UART Universal Asynchronous Receiver/Transmitter.

URL Uniform Resource Locator (Internet).

W2C Web To Card (*Infra*).

WOSA Windows Open Services Architecture (Microsoft).

WWW World Wide Web ou W3 (Internet).

ANNEXE D

Index

Les numéros de pages en gras référencent des pages contenant des informations importantes à propos de l'entrée, par exemple, une définition ou un commentaire. Les intervalles de pages en italique correspondent à des paragraphes traitant essentiellement de l'entrée. Les autres numéros de pages sont des références textuelles.

– A –

– B –

– C –

– F –

– I –

– L –

– M –

– O –

– P –

– R –

– S –

– T –

– W –