



Université Joseph Fourier
U.F.R. Informatique &
Mathématiques Appliquées



Institut National Polytechnique
de Grenoble
ENSIMAG

I.M.A.G.

**ÉCOLE DOCTORALE
MATHÉMATIQUES ET INFORMATIQUE**

**DEA D'INFORMATIQUE :
SYSTÈMES ET COMMUNICATIONS**

Projet présenté par :

Christophe RIPPERT

**ANALYSE DU RÔLE DES CARTES À PUCE DANS UN
ENVIRONNEMENT RÉPARTI**

Effectué au laboratoire : Sirac

Date : 21 Juin 2000

Jury :

Ricardo CAFERRA
Andrzej DUDA
Daniel HAGIMONT
Brigitte PLATEAU

Table des matières

1	Introduction	5
1.1	Contexte du projet	5
1.2	Plan	6
I	État de l’art	7
2	Présentation des cartes à puce	9
2.1	Historique	9
2.2	Caractéristiques	9
2.3	Synthèse	11
3	Architecture logicielle	13
3.1	Le protocole de communication par APDUs	13
3.2	La station d’accueil	14
3.3	L’environnement d’exécution Java Card	14
3.4	Synthèse	17
4	Utilisations actuelles	19
4.1	Cartes d’identification	19
4.2	Cartes à valeur monétaire	22
4.3	Cartes applicatives	24
4.4	Recherche	24
4.5	Industriels	26
4.6	Synthèse	26
II	Analyse	27
5	Intérêts des cartes à puce	29
5.1	Disponibilité	29
5.2	Sécurité	30
5.3	Intérêts supplémentaires des cartes Java	31
5.4	Confrontation disponibilité/sécurité	32
5.5	Étude de deux applications existantes	33
5.6	Synthèse	34

6	Spécification d'une application typique	37
6.1	Architecture	37
6.2	Fonctionnement	38
6.3	Fonctionnalités supplémentaires	39
6.4	Avantages	39
6.5	Synthèse	40
7	Services systèmes	41
7.1	Logiciel de base	41
7.2	Communication station ↔ carte	42
7.3	Chargement de code	45
7.4	Gestion de la mémoire	45
7.5	Authentification	46
7.6	Intégrité et confidentialité des données	46
7.7	Compression	47
7.8	Synthèse	47
III	Prototypage	49
8	Prototypage	51
8.1	Environnement de développement	51
8.2	Protocole de développement	51
8.3	Restrictions	52
8.4	Performances	53
8.5	Exemple d'exécution	54
8.6	Perspectives	55
IV	Conclusion	59
9	Synthèse	61
10	Perspectives	63
11	Remerciements	65
V	Bibliographie	67
VI	Annexes	73
A	Exemple: <i>ping</i> pour Java Card	75
B	Quelques liens utiles	81

Chapitre 1

Introduction

Les cartes à puce font désormais partie intégrante de notre vie quotidienne. Cette technologie existant depuis plus de trente ans semble aujourd'hui parfaitement maîtrisée et pourtant de nouveaux problèmes apparaissent chaque jour. Les questions de confidentialité des informations personnelles, de sécurité des transactions financières sur Internet, de l'accès mobile à des services sont plus que jamais d'actualité dans un monde où les échanges se font désormais à l'échelle de la planète. Grâce aux progrès techniques importants réalisés ces dernières années, aussi bien au plan matériel que logiciel, les cartes à puce semblent désormais prêtes à répondre aux besoins de disponibilité et de sécurité des applications réparties modernes.

Le but de ce projet de DEA est d'étudier le rôle que les cartes à puce (principalement les cartes à microprocesseur) sont amenées à jouer dans un environnement réparti. Par environnement réparti, on entend une architecture matérielle distribuée sur laquelle s'exécutent des applications réparties. Le fait d'introduire des cartes à puce dans ce type de systèmes n'est pas anodin. Les caractéristiques intrinsèques de la carte (un support portable et hautement sécurisé) laissent entrevoir l'apport de ce média en terme de disponibilité et de sécurité des applications qu'il devra supporter. Ce travail aura donc pour finalité de mettre en évidence les points forts et les lacunes des cartes à puce comme support d'applications réparties, et de proposer le cas échéant de nouveaux services susceptibles de répondre aux attentes des développeurs d'applications encartées. Cette étude théorique devra être soutenue par le prototypage d'une application caractéristique sur la carte, qui permettra de se familiariser avec l'environnement de développement et d'exécution et d'évaluer concrètement les contraintes liées à l'utilisation de la carte.

1.1 Contexte du projet

Ce projet de DEA a été réalisé au sein du laboratoire Sirac (Systèmes informatiques répartis pour applications coopératives)¹. Les recherches menées dans ce laboratoire s'orientent autour de deux grandes thématiques. La première concerne la construction d'applications réparties adaptables. L'objectif est de fournir des outils et services pour utiliser efficacement l'Internet (ou un intranet) comme environnement d'exécution d'applications réparties. L'activité principale porte sur le développement d'un environnement de construction d'ap-

1. Sirac est une unité de recherche commune à l'Institut National de Recherche en Informatique et en Automatique (Unité de Recherche Rhône-Alpes), à l'Institut National Polytechnique de Grenoble, et à l'Université Joseph Fourier (Grenoble-1).

plications à base de composants, avec des capacités d'adaptation et de reconfiguration dynamiques pour répondre à l'évolution des besoins des applications et des conditions d'utilisation. Ce travail prend en compte l'existence d'une activité industrielle forte dans le domaine des composants Java (*Enterprise Java Beans*) et CORBA. Le deuxième axe de recherche consiste au développement d'un support système pour les serveurs en grappes. L'objectif est de fournir des services génériques et efficaces pour la construction de serveurs d'information extensibles, sur des grappes (*clusters*) de machines homogènes. L'activité principale porte sur les services système pour les réseaux de communication à capacité d'adressage (en utilisant la technologie d'interconnexion *Scalable Coherent Interface*), en vue de leur utilisation dans des grappes de processeurs de grande taille (plusieurs centaines de machines de type PC). Le projet de DEA présenté ici s'intègre dans la première thématique concernant les applications réparties adaptables.

Ce travail fait partie d'un projet du Réseau National de Recherche en Télécommunications, intitulé CESURE (Configuration et Exécution de Services pour les Usagers mobiles des Réseaux Étendus) [1]. Ce projet regroupe la société Gemplus, ainsi que le Laboratoire d'Informatique Fondamentale de Lille, l'Institut National des Télécommunications d'Evry et l'Unité de Recherche Rhône-Alpes de l'INRIA. Ce projet a pour objectif de fournir des outils permettant de concevoir des services proposés aux usagers du réseau ; de fournir l'infrastructure qui permet, via un terminal de nature quelconque, de configurer, déployer et superviser le système informatique distribué qui met en oeuvre l'accès à ces services ; et de permettre la configuration de ces services depuis le poste client en utilisant la carte à puce pour stocker la description de la configuration et l'état du service rendu. Pour cela, les membres du projets s'intéresseront plus particulièrement aux composants logiciels configurables et mobiles, aux langages de description d'architectures logicielles reconfigurables, aux infrastructures informatiques globales pour le support de services distribués, à la personnalisation et la sécurisation des services et aux nouveaux usages de la carte à puce pour la configuration de services. Ce projet de DEA s'intègre donc dans cette dernière partie concernant les nouveaux usages de la carte à puce, même si l'on ne s'intéressera pas ici aux problèmes de configuration et de reconfiguration.

1.2 Plan

Le but de ce projet de DEA est d'analyser le rôle des cartes à puce dans un environnement réparti. On commencera donc par effectuer une étude de l'état actuel du domaine, tant au niveau matériel que logiciel et applicatif. On passera ensuite à l'analyse à proprement parler, qui s'articulera autour de trois axes principaux. Tout d'abord, on cherchera à mettre en évidence les principaux intérêts de l'utilisation de cartes à puce comme support d'applications réparties. Puis on spécifiera à titre d'exemple une application typique, que l'on implémentera ensuite sur la carte. Enfin, on évaluera les services systèmes couramment proposés au développeur d'applications sur la carte et on formulera des propositions de nouveaux services pouvant faciliter le développement de ces applications, en s'inspirant notamment de l'expérience acquise lors du prototypage de notre application typique. La troisième partie de ce rapport détaillera justement le fonctionnement de notre prototype.

Première partie

État de l'art

Chapitre 2

Présentation des cartes à puce

On présente dans ce chapitre un bref historique des cartes à puce, avant de détailler les caractéristiques techniques des principaux modèles existant à ce jour.

2.1 Historique

La carte à puce a été inventée dans les années 1960–70. La paternité est généralement attribuée à deux ingénieurs allemands, Jurgen Dethloff et Helmut Grottrupp, qui ont construit le tout premier prototype en 1967. D'autres prototypes ont été développés de façon indépendante par le japonais Kunitaka Arimura (1970), l'américain Paul Casrucci (1971) et le français Roland Moréno (1974).

En France, la société Bull s'est rapidement intéressée à cette nouvelle technologie, et a commencé à commercialiser des cartes dès 1977. En 1980, France Télécom produit la première carte téléphonique (il s'agissait alors d'une carte magnétique). Cette carte s'exporte en Allemagne en 1983. En 1984, le Groupement des Cartes Bancaires met en place en France le premier réseau national de paiement par carte de crédit basé sur des cartes à puce. En 1987, France Télécom abandonne la technologie des cartes magnétiques et fabrique sa télécarte telle qu'on la connaît actuellement : basée sur un système de « fusibles », un par unité, qui sont brûlés au fur et à mesure que le client utilise sa carte. Ce système, beaucoup plus sûr que celui utilisant les cartes magnétiques, permet en outre d'éviter le rechargement abusif de la carte. En 1994, les sociétés Europay, Mastercard et Visa définissent la spécification internationale EMV afin de normaliser les paiements par cartes à puce dans le monde entier. Enfin, en 1996, la première expérimentation à grande échelle de porte-monnaie électronique est mise en oeuvre aux États-Unis durant les Jeux Olympiques d'Atlanta.

2.2 Caractéristiques

La norme ISO7816 définit de façon très précise les caractéristiques matérielles et logicielles des cartes à puce. Cette norme est découpée en 11 thèmes, certains n'étant encore qu'à l'état de brouillon [2]. On trouve de plus de nombreuses spécifications industrielles, typiquement orientées vers des utilisations spécifiques des cartes à puce. On peut citer notamment la spécification EMV [3], [4], [5], [6], développée par les principaux fournisseurs d'applications financières (Europay, Mastercard et VISA), et qui régit actuellement la majorité des cartes de paiement dans le monde. Une autre spécification bien connue est celle de la technologie Java Card, dont on parlera plus en détail au Chapitre 3.

On détaille ci-dessous les différents types de cartes à puce disponibles actuellement. On donne pour chacun de ces types leurs principales caractéristiques techniques. On notera que le terme « carte à puce » peut paraître inapproprié pour qualifier des cartes qui ne contiennent pas de puce électronique, comme les cartes magnétiques par exemple. C'est cependant la terminologie qui est couramment utilisée. Cette expression a d'ailleurs été reprise littéralement en anglais (*Integrated Circuit Card*), bien que l'expression plus familière *smart card* soit de nos jours la plus employée.

2.2.1 Les cartes magnétiques

Ces cartes disposent d'une bande magnétique collée sur le support plastique. La quantité d'information stockable sur ce type de média étant particulièrement faible, ces cartes sont de moins en moins utilisées de nos jours. Leur seul intérêt réside dans leur coût de fabrication très économique.

Caractéristiques techniques :

Capacité moyenne de la bande magnétique : 140 octets

2.2.2 Les cartes optiques

Ces cartes sont conçues comme des disques compacts : les données sont gravées une fois pour toute sur une couche d'aluminium réfléchissante protégée par une couche de plastique transparent. On utilise des lecteurs à faisceau laser pour lire les données qu'elles contiennent. Cette technologie permet de stocker une très grande quantité de données sur la carte, mais elles ne sont pas réinscriptibles. Les coûts de fabrication exorbitants des cartes et du lecteur font qu'elles sont en pratique très peu utilisées.

Caractéristiques techniques :

Capacité moyenne du disque optique : 5 méga-octets

2.2.3 Les cartes mémoires

Elles comportent une mémoire réinscriptible persistante, qui permet de stocker et de modifier des données. Les cartes classées dans cette catégorie sont en fait basées sur des technologies très différentes les unes des autres, et souvent dédiées à une application donnée. Leur intérêt a considérablement baissé depuis l'apparition des cartes à microprocesseur.

Caractéristiques techniques :

Capacité moyenne de la mémoire persistante réinscriptible : 8 kilo-octets

2.2.4 Les cartes à microprocesseur

Les cartes de ce type intègrent un microprocesseur afin de pouvoir exécuter des applications directement sur la carte. Elles disposent de plus de 3 sortes de mémoires différentes : une mémoire non réinscriptible persistante (de type ROM) qui sert en général à stocker des données ou du code de base, comme un système d'exploitation par exemple ; une mémoire réinscriptible persistante (de type EEPROM), qui permet de conserver des informations d'une session d'utilisation à l'autre ; et une mémoire réinscriptible volatile (de type RAM),

qui sert de zone de travail aux applications. Le coût de fabrication de ces cartes ayant considérablement baissé ces dernières années, le nombre de leurs applications est en constante augmentation.

Caractéristiques techniques :

Capacité moyenne de la mémoire persistante non réinscriptible : 32 kilo-octets

Capacité moyenne de la mémoire persistante réinscriptible : 32 kilo-octets

Capacité moyenne de la mémoire volatile : 2 kilo-octets

Fréquence moyenne du processeur : 5-20 MHz

2.2.5 Autres types de cartes

Il existe des cartes à puce spécifiques à une application, qui ne correspondent à aucune des catégories ci-dessus. On citera par exemple les cartes téléphoniques France Télécom, qui sont couramment employées en France depuis une vingtaine d'années.

2.3 Synthèse

Comme on l'a dit, les cartes à puce existent depuis plus de trente ans. Cependant, au début, ces cartes étaient le plus souvent dédiées à une application donnée (carte téléphonique, cartes bancaires, etc...). Ces cartes étaient bien sûr parfaitement incompatibles les unes avec les autres et nécessitaient des lecteurs spécifiques. Cela peut expliquer pourquoi les cartes à puce ont eu du mal à se développer les premières années, avec bien sûr le fait que la technologie était moins performante à l'époque que de nos jours. Depuis quelques années, des efforts importants de normalisation sont donc entrepris pour essayer de favoriser leur développement. Il est maintenant possible d'utiliser sa carte bancaire dans une cabine téléphonique pour régler sa communication par exemple. La montée en puissance des cartes à microprocesseur constitue l'étape suivante de cette volonté de normalisation : il ne s'agit non plus de construire des lecteurs acceptant plusieurs types de cartes différents, mais plutôt d'utiliser le même type de carte comme support d'applications différentes. La souplesse du microprocesseur permet de programmer n'importe quel type d'applications sur des cartes qui peuvent être génériques. Dans cette étude, on s'intéressera donc principalement aux cartes à microprocesseur qui nous semblent appelées à remplacer toutes les autres. Elles possèdent de plus un certain nombre d'intérêts spécifiques, comme on le verra en détail au Chapitre 5.

Chapitre 3

Architecture logicielle

Comme on l'a dit au Chapitre 2, on s'intéresse principalement dans cette étude aux cartes à microprocesseur qui permettent d'exécuter du code directement sur la carte. Toutes ces cartes fonctionnent selon un modèle client/serveur établi entre la station d'accueil (le client) et la carte (le serveur). Ce mode de fonctionnement implique que le code dans la carte soit passif (ie : c'est le code dans la station qui prend l'initiative d'appeler les méthodes dans la carte qui ne peuvent que répondre, et non l'inverse). On présente tout d'abord le mode de communication entre la station et la carte avant de détailler le fonctionnement des deux parties.

3.1 Le protocole de communication par APDUs

Pour assurer la communication entre la station d'accueil et l'environnement d'exécution sur la carte, la norme ISO7816-4 définit un protocole basé sur les unités de données applicatives (*Application Protocol Data Unit*, couramment notés APDUs) qui permet d'encapsuler les données dans un format commun aux deux parties. Il existe deux types d'APDUs : les commandes et les réponses. Les commandes sont envoyées par la station à la carte et sont basées sur le format suivant (les champs entre crochets sont facultatifs) : <CLA, INS, P1, P2, [Lc], [Données], [Le]> où

CLA permet d'identifier la classe de la commande. L'application peut ainsi ordonner ses différentes fonctionnalités en classes distinctes.

INS précise l'instruction choisit, dans la classe donnée.

P1 et P2 sont les paramètres de cette instruction.

Lc est la taille du champs de données qui suit.

Données est le champs de données qui peut contenir par exemple les arguments de l'instruction.

Le est la taille maximum de la réponse attendue.

Les réponses renvoyées par la carte sont quand à elles d'un format plus simple : <[Données], Statut> où

Données est le champs de données contenant le résultat de la fonction.

Statut est un entier indiquant si la fonction s'est exécutée normalement et le code d'erreur le cas échéant.

Ce protocole de très bas niveau n'est pas pratique à utiliser, notamment à cause de la limitation en taille des messages envoyés (au maximum 261 octets pour une commande). Il

est donc difficile d'envoyer des paramètres complexes (tableaux, objets, etc...) aux fonctions dans la carte, puisqu'il faut les découper en plusieurs APDUs que l'application encartée devra reconstituer.

De plus, le matériel utilisé actuellement ne permet pas d'obtenir des débits importants pour les transferts de données entre le lecteur et la carte (environ 200 octets/seconde). Ce problème force donc le programmeur à limiter au strict minimum les échanges entre la carte et la station. On peut cependant espérer que les progrès rapides des technologies utilisées permettront d'obtenir de meilleurs débits dans un proche avenir.

3.2 La station d'accueil

La plupart des cartes à puces doivent être insérées dans un lecteur pour être alimentées en électricité et pouvoir communiquer (bien qu'il existe des cartes capables de communiquer par ondes infrarouges ou radios). Ce lecteur fait partie d'une machine que l'on nommera « station d'accueil ». Ce terme désigne de façon générale la machine dans laquelle on insère la carte, et peut prendre des formes variées (ordinateur personnel, guichet automatique, ...). Le plus souvent la station d'accueil intègre des périphériques d'entrées/sorties afin de permettre à l'utilisateur d'interagir avec le système, ainsi qu'un logiciel de base.

Au niveau applicatif, le code dans la station est basé sur l'architecture de l'OpenCard (*OpenCard Framework*) [7] [8]. Ce consortium regroupe de nombreuses entreprises impliqués dans le domaine des cartes à puce, et a pour vocation de proposer des standards implémentables sur le plus grand nombre de matériels possible. Cette architecture comprend par exemple un module spécifié par la société Visa et concerné principalement par la sécurité (*Visa Open Platform*). Cette architecture n'est qu'une spécification, et c'est généralement le constructeur du lecteur de carte qui fournit à ses clients son implémentation du modèle.

3.3 L'environnement d'exécution Java Card

Dans cette étude, on ne s'intéresse qu'aux cartes utilisant la technologie Java Card de Sun Microsystems. Cette technologie a été spécifiée en 1996, et a subi depuis de nombreuses modifications. La technologie Java Card est aujourd'hui utilisée par 90% des développeurs d'applications sur carte à puce. Il existe des alternatives à Java Card, principalement des solutions propriétaires, comme les cartes Visual Basic de Microsoft par exemple. Le succès de la technologie Java Card peut d'ailleurs s'expliquer principalement par le fait que sa spécification est libre et qu'il existe donc de nombreuses sociétés qui proposent leur implémentation de cette spécification. Toutes ces implémentations sont bien sûr conformes à la spécification de Sun Microsystems, et donc les applications développées pour l'une pourront facilement être portées sur une autre. Cette portabilité est la raison principale de notre choix de la technologie Java Card pour ce projet. Cette diversité permet en outre d'éviter de se trouver lié à un fabricant donné, comme c'est le cas avec les cartes Visual Basic de Microsoft. Le succès actuel du langage Java peut aussi expliquer l'engouement pour la technologie Java Card.

Sur la carte à puce, les applications sont exécutées dans l'environnement d'exécution Java Card (*Java Card Runtime Environment*) [9]. Cet environnement contient la machine virtuelle Java Card (*Java Card Virtual Machine*) [10], les classes de l'interface applicative Java Card (*Java Card API*) [11] [12] et les services associés (comme l'installateur d'applications par exemple).

3.3.1 Le langage et la machine virtuelle Java Card

La quantité de mémoire disponible sur les cartes à microprocesseur étant en général très limitée, ainsi d'ailleurs que la puissance de calcul des processeurs utilisés, le langage utilisé dans l'environnement d'exécution Java Card que l'on désignera ici sous le nom de « langage Java Card », est un sous ensemble du langage Java spécifié dans [13]. On n'y trouve pas de types réels par exemple, ni de gestion des chaînes de caractères (*class String*). De même, la machine virtuelle Java Card est très réduite par rapport à la machine virtuelle Java classique [14]. Elle ne supporte pas le chargement dynamique des classes (*dynamic class loading*) et n'inclut pas de support pour les processus légers Java (*Java threads*), ni de ramasse-miettes (*garbage collector*).

3.3.2 Durée de vie de la machine virtuelle

A la différence d'une machine virtuelle classique s'exécutant sur une station de travail, qui meurt lorsque son processus se termine, la machine virtuelle Java Card a une durée de vie égale à celle de la carte sur laquelle elle est implémentée. En effet, la mémoire persistante dont sont dotées les cartes Java permet de sauvegarder les données des applications et de la machine virtuelle lorsque la carte est retirée du lecteur de la station d'accueil. Ces données essentielles au fonctionnement du système, telle que le tas des objets par exemple, sont restaurées dès que la carte est de nouveau alimentée, et la machine virtuelle reprend son exécution au point où elle s'était arrêtée. On peut ainsi dire que la machine virtuelle Java Card ne s'arrête jamais.

3.3.3 Architecture d'une application Java Card

L'environnement d'exécution Java Card communique avec les applications grâce aux quatre méthodes détaillées ci-dessous, que doivent implémenter toutes les applications développées pour Java Card.

La méthode `install` sert à initialiser les objets composants l'application. Elle est appelée une et une seule fois par l'environnement d'exécution lorsque l'application est chargée sur la carte. En général, on utilise cette méthode pour créer les objets composants l'application, les initialiser avec des valeurs par défaut, affecter des variables internes et de façon général effectuer toutes les actions nécessaires à l'initialisation de l'application. Cette méthode doit appeler la méthode `register`, afin d'enregistrer l'application auprès de l'environnement d'exécution. Une fois enregistrée, l'application a une durée de vie égale à celle de la carte.

La méthode `select` permet à l'environnement d'exécution d'activer une application. En effet, l'environnement d'exécution Java Card permet d'installer plusieurs applications sur la même carte. Cependant, comme la machine virtuelle Java Card ne gère pas les processus légers Java, une seule application peut s'exécuter à la fois dans la machine virtuelle. Pour sélectionner cette application, l'utilisateur envoie à l'environnement d'exécution un APDU contenant l'identificateur de l'application. L'environnement se sert d'une table interne contenant la liste des applications installées ainsi que leur identificateur respectif pour trouver l'application demandée. Pour sélectionner une application, l'environnement d'exécution doit d'abord désélectionner l'application courante, en utilisant la méthode `deselect`. Puis il appelle la méthode `select` de l'application choisie. On peut définir une application qui sera sélectionnée par défaut, c'est à dire lors de la première utilisation de la carte et après une erreur lors d'une tentative de

sélection. Une telle erreur de sélection peut par exemple être provoquée volontairement si l'utilisateur répond mal à une demande d'authentification.

La méthode `process` contient le corps de l'application, puisque c'est cette méthode qui reçoit et traite les APDUs transmis par l'environnement d'exécution, et renvoie les éventuels résultats à la station d'accueil.

La méthode `deselect` permet à l'environnement d'exécution de désélectionner une application au profit d'une autre. Cette méthode permet à l'application d'effectuer toute les opérations de « nettoyage » nécessaires avant de passer la main.

On notera que ces méthodes ne sont pas appelées directement depuis la station. Le protocole de communication par APDUs impose que les méthodes dans la station envoient des APDUs à l'environnement d'exécution, qui se charge ensuite d'appeler les méthodes ci-dessus. Ainsi, si l'APDU commandé envoyé ne correspond ni à un ordre d'installation d'application, ni à la sélection d'une nouvelle application, l'environnement d'exécution se contentera d'appeler la méthode `process` en lui passant l'APDU en argument. Cette indirection permet à l'environnement d'exécution d'effectuer des vérifications sur le format des APDUs envoyés, ce qui évite au programmeur d'avoir à le faire dans son application.

3.3.4 Protection et communication interapplications

Le langage Java a été conçu de façon à aider le programmeur à éviter les erreurs, notamment grâce à son typage fort et aux qualificatifs d'accès (`private`, `protected`, ...) qui permettent de restreindre l'accès à certaines variables ou méthodes. L'environnement d'exécution Java Card ajoute à ces protections statiques une protection dynamique, sous la forme de pare-feux applicatifs (*applet firewall*) qui partitionnent l'espace d'allocation des objets en plusieurs sous espaces appelés contextes. Les pare-feux restreignent l'accès aux objets d'un contexte depuis un autre contexte. L'environnement d'exécution alloue en général un contexte par application. Pour permettre la communication entre deux applications, on peut définir une interface partageable (*shareable interface*) dans laquelle seront spécifiées les méthodes d'une application que l'autre aura le droit d'invoquer. L'appel à une méthode appartenant à un autre contexte nécessite un changement de contexte (*context switching*), afin de pouvoir exécuter la méthode dans son propre contexte. Les contextes sont ainsi empilés et dépilés selon les appels de méthodes. L'environnement d'exécution appartient à un contexte privilégié qui lui permet d'appeler n'importe quelle méthode dans n'importe quel contexte, ce qui est nécessaire notamment pour pouvoir changer l'application sélectionnée. Le contexte de l'environnement d'exécution dispose de points d'entrées, afin de permettre aux applications d'accéder aux services systèmes dont elles pourraient avoir besoin. C'est l'environnement d'exécution qui détermine quelle capacité est accordé à quelle application.

3.3.5 Transactions et atomicité

La machine virtuelle Java Card assure que toutes les mises à jour effectuées sur un objet persistant le seront de façon atomique. L'interface applicative Java Card propose de plus un ensemble de méthodes permettant de définir la notion de transaction atomique (*atomic transaction*) lorsque l'on désire mettre à jour plusieurs objets de façon atomique. Cela permet d'assurer la cohérence des données stockées en mémoire persistante, même en cas de panne (comme une déconnexion brutale de la carte du lecteur par exemple). Toutes les opérations effectuées après un appel à la méthode `beginTransaction` seront mises en attente jusqu'à l'appel à la méthode `commitTransaction`. Il est possible d'annuler une transaction en cours grâce à la méthode `abortTransaction`. L'interface applicative Java Card 2.1 ne permet pas d'effectuer des transactions imbriquées.

3.3.6 L'installateur d'applications

Le développement exact de l'installateur d'applications (*applet installer*) est laissé à la discrétion du concepteur de l'environnement d'exécution. Le plus souvent, cet installateur prend la forme d'une application pouvant être sélectionnée comme les autres. Quel que soit sa forme exacte, l'installateur effectue les mêmes opérations à chaque chargement d'application : identifier l'utilisateur, afin de s'assurer que l'installation est autorisée ; placer le code de l'application dans la mémoire de la carte ; appeler la méthode `install` pour initialiser l'application. Pour cela, l'installateur doit bénéficier de privilèges spécifiques qui lui permettent d'accéder directement à la mémoire de la carte et d'invoquer des méthodes n'appartenant pas à son contexte. Cet installateur n'est en général pas appelable par les applications, ce qui signifie qu'une application ne peut pas demander le chargement de code dynamiquement. C'est une restriction par rapport à l'environnement Java standard que l'on pourrait vouloir supprimer.

3.4 Synthèse

Comme on l'a dit au Chapitre 2, un effort important de normalisation est entrepris depuis quelques années. Les cartes à microprocesseur constituent une avancée importante, puisqu'elles permettent d'implémenter des applications diverses sur le même type de cartes. L'architecture logicielle détaillée ici va dans le sens de cet effort de normalisation. Le fait de disposer d'une architecture standard selon laquelle développer ses applications est un facteur important de portabilité du logiciel, puisque même si l'on est obligé de recoder l'application dans un autre langage, sa structure restera la même. De plus, si l'on se rallie à la quasi unanimité des développeurs d'applications sur cartes à microprocesseur, et qu'on utilise la technologie Java Card, on bénéficie alors de la portabilité du langage Java. Une application développée en utilisant la technologie Java Card sur une carte à microprocesseur donnée pourra être portée très facilement sur la grande majorité des cartes à puce et utilisée avec quasiment tous les lecteurs fabriqués récemment. On peut donc dire que cette tentative de normalisation a réussi, grâce aux cartes à microprocesseur équipée de la technologie Java Card. C'est pour cela qu'on s'intéressera principalement à ce type de cartes dans cette étude.

Chapitre 4

Utilisations actuelles

On présente dans ce chapitre quelques applications courantes des cartes à puce. On tentera de classer ces applications en catégories et on essayera de cerner les défauts ou limitations des systèmes actuels, ce qui nous sera utile lors de notre inventaire des services systèmes au Chapitre 7. Cette proposition de classification peut bien sûr être soumise à discussions, d'autant plus que certaines cartes pourraient être rangées dans plusieurs catégories. On présente ensuite quelques projets de recherches, universitaires ou industriels, caractéristiques de la volonté d'intégrer la carte à puce au coeur des techniques et technologies les plus récentes.

4.1 Cartes d'identification

Ces cartes sont utilisées pour identifier de façon indiscutable une personne, et/ou pour garantir la sécurité d'informations personnelles sensibles. Les cartes de ce type sont actuellement les plus répandues, comme en témoignent les nombreux exemples d'applications donnés ci-dessous. Le principal problème de ces applications est la sécurité, puisqu'elles ne disposent en général que de systèmes d'authentification assez simplistes (codes d'identification personnels (*Personal Identification Number*) courts, etc...), lorsqu'elles en disposent ce qui n'est même pas toujours le cas. Ce manque de confidentialité n'empêche pas ces cartes d'être utilisées quotidiennement, avec tous les problèmes que l'on connaît (détournement de numéros de cartes bancaires, falsification de cartes d'identité, etc...). L'apport des cartes à microprocesseur dans ce domaine est indéniable, puisque la sécurité est leur valeur ajoutée, comme on le verra au Chapitre 5. On peut espérer que la baisse du prix de ces cartes permettra leur plus grande diffusion, dans l'intérêt de l'utilisateur.

Carte d'identité Une telle carte d'identité a fait son apparition en Finlande en décembre 1999 (voir FIG. 4.1). Les informations habituellement trouvées sur une pièce d'identité classique sont écrites sur la carte ainsi que stockées dans la puce, ce qui permet au propriétaire de s'identifier sur Internet par exemple [15].

Carte de sécurité sociale Cette carte contient les informations concernant l'assuré social que l'on peut trouver sur une carte papier classique. Les cartes de sécurité sociale sont diffusées en France sous le nom de Carte Vitale (voir FIG. 4.2). La deuxième version de ces cartes devrait intégrer des informations sur la santé de l'utilisateur, comme les cartes médicales détaillées ci-dessous.

Carte médicale Elle contient les antécédents médicaux du patient. Le médecin peut donc connaître le passif de son client, même si c'est la première fois qu'il le voit. Cela



FIG. 4.1 – La carte d'identité Finlandaise



FIG. 4.2 – La carte Vitale (1^{re} version)

peut permettre d'accélérer l'établissement du diagnostic et d'éviter les erreurs de prescriptions (allergie à certains médicaments, etc...). L'information stockée sur cette carte est bien sûr confidentielle, et ne doit donc pouvoir être consultée ou modifiée que par un médecin. Les cartes de vaccination Vaccicarte sont un exemple typique de carte médicale (voir FIG. 4.3).



FIG. 4.3 – La carte de vaccination Vaccicarte

Permis de conduire Il contient les points restants ainsi que la liste des procès verbaux infligés au conducteur. Cela permet aux agents de la force publique de vérifier que ces contraventions ont bien été acquittées et d'en exiger le paiement immédiat le cas échéant. Ce type de permis est testé depuis 1995 dans la province de Mendoza, en Argentine. Ce système se révèle efficace puisque le taux de recouvrement des amendes, qui était seulement d'environ 30% auparavant, a doublé depuis l'instauration de ce permis électronique.

Carte d'étudiant Elle permet a un étudiant s'étant acquitté de ses droits d'inscription de passer des examens ou d'emprunter des ouvrages. Elle peut aussi permettre d'accéder aux salles machines après la fermeture des locaux par exemple. Cette carte est déjà utilisée dans de nombreuses universités, comme à l'université polytechnique de Hong-Kong, par exemple (voir FIG. 4.4).



FIG. 4.4 – La carte d'étudiant de l'université polytechnique de Hong-Kong

Carte d'identification informatique Elle permet à un utilisateur de s'identifier pour une session de travail (*login*). L'utilisateur insère sa carte dans le terminal et entre son mot de passe, qui pourra d'ailleurs être vérifié sur la carte ou sur le serveur de façon plus classique. La carte peut aussi stocker le profil de l'utilisateur (paramètres de configuration, données personnelles, ...), ce qui lui permet de travailler dans son environnement personnel depuis n'importe quel terminal. L'accès au serveur peut être sécurisé, comme avec le système Kerberos [16], pour lequel les clés de session pourront être stockées sur la carte elle même. Ce type d'application suscite actuellement un grand intérêt, comme le montre le nombre important d'articles sur le sujet, entre autres [17] [18] [19] [20], [21] et [22].

Carte d'accès Elle permet d'accéder à des locaux (parking, bureau, ...). Elle peut aussi servir à contrôler les heures d'arrivée et de départ du lieu de travail.

Carte bancaire Cette carte, utilisée partout dans le monde sous ses diverses formes (Carte Bleue Nationale, Visa, ...) n'est pas, comme on aurait pu le penser, une carte à valeur monétaire telle qu'on les définit ci-dessous. En effet, cette carte sert uniquement à identifier le client, avant que la station n'envoie à la banque l'ordre de débit. Les nouvelles générations de cartes Visa (voir FIG. 4.5) devraient intégrer des microprocesseurs afin de renforcer la sécurité des transactions et de permettre d'implémenter de nouvelles fonctionnalités, comme la gestion de certificats de transaction, qui permettront de conserver une trace des achats effectués, ce qui pourra être utile en cas de contestation par exemple. Le manque de sécurité de ce type de cartes est flagrant, puisque des informations confidentielles comme le numéro de la carte et sa date limite de validité sont inscrites lisiblement sur la carte elle même. Ces informations permettant d'effectuer des achats par correspondance dans le monde entier, il n'est pas étonnant de constater le nombre important d'utilisations frauduleuses de numéros de carte, surtout depuis

l'essor du commerce électronique. On reparlera de ces problèmes au Chapitre 5.



FIG. 4.5 – La nouvelle Carte Visa International

Carte SIM La carte SIM (*Subscriber Identity Module*) est utilisée dans les téléphones GSM (*Global System for Mobile communications*) pour vérifier l'identité de l'utilisateur (voir FIG. 4.6). Il est impossible d'utiliser le téléphone sans la carte et donc sans connaître le code secret de l'utilisateur. La carte peut aussi servir à stocker des données, comme le répertoire téléphonique de l'utilisateur par exemple. On détaillera le fonctionnement des cartes SIM au Chapitre 5.



FIG. 4.6 – La carte d'identification des téléphones GSM France Télécom

4.2 Cartes à valeur monétaire

Ces cartes permettent de stocker et d'utiliser des unités à valeur marchande. Ces unités sont généralement achetées et stockées dans la carte indépendamment du moment de leur utilisation. La notion d'unité est importante, puisqu'elle implique que l'accès au service est limité par le nombre d'unités achetées, du moins jusqu'à un éventuel rechargement de la carte. Ce type d'application n'intègre pas forcément de système de sécurité, notamment à cause de la faiblesse des valeurs généralement stockées dessus. Le principal problème de ces systèmes est qu'ils obligent le prestataire de service à s'équiper d'une station d'accueil qui peut être onéreuse, et qui est en général dédiée à un système conçu par un fabricant donné. En cas de multiplication des systèmes, par exemple des porte-monnaies électroniques

émis par des banques différentes, un manque de standardisation ne peut être que fatal à l'application puisqu'on voit mal un commerçant acheter une station d'accueil pour chaque marque de porte-monnaie existant. Ce problème ne se pose bien sûr pas pour des sociétés en position de monopole, comme France Télécom pour les cabines téléphoniques par exemple.

Porte-monnaie électronique Il permet de stocker une faible somme d'argent sous forme numérique. Il est en général équipé d'un système de protection contre le vol, le plus souvent un code d'identification personnel. De plus en plus de banques et d'organismes financiers proposent à leurs clients des services de ce type, qui semblent appelés à ce développer vu qu'ils évitent d'avoir à s'encombrer de pièces de monnaie pour régler ses achats courants, par exemple dans des distributeurs automatiques (voir FIG. 4.7).



FIG. 4.7 – *Le porte-monnaie électronique de Visa*

Carte de transport en commun Elle est utilisée quotidiennement par tous les utilisateurs de transports en commun. Le client achète un certain nombre de trajets pour un parcours donné, en général à un tarif préférentiel, et les utilise lorsqu'il emprunte les transports en commun. Elle se présente généralement sous la forme d'une carte magnétique à faible coût de fabrication, puisqu'elle n'est le plus souvent pas rechargeable et donc jetable. Elle n'intègre pas de système de protection contre le vol puisque son utilité très spécialisée (un nombre limité de trajet sur un parcours donné) et sa faible valeur rendent le vol peu rentable.

Carte téléphonique Elle est répandue dans le monde entier. Le client achète une carte contenant un certain nombre d'unités téléphoniques, qu'il dépense en utilisant les téléphones publics. Elle n'est en général pas rechargeable. On citera par exemple la Télécarte de France Télécom (voir FIG. 4.8) bien connue.



FIG. 4.8 – *La Télécarte de France Télécom*

4.3 Cartes applicatives

Ces cartes intègrent des applications qui nécessitent la présence d'un microprocesseur. On doit noter que vu la baisse du prix du matériel, de plus en plus de fabricants utilisent des cartes à microprocesseur afin de proposer de nouveaux services, ce qui implique que plusieurs cartes citées précédemment pourraient aussi être classées dans cette catégorie (carte bancaire, carte SIM, porte-monnaie électronique, etc...). Ces cartes nous semblent donc appelées à se développer considérablement au cours des prochaines années. On cite ci-dessous deux exemples d'applications dont on reparlera au cours de ce rapport.

Carte de fidélité Cette carte permet à un magasin de fidéliser ses clients en leur permettant d'accumuler des points à chaque achat. Ces points sont stockés sur la carte, qui est capable de calculer la réduction associée. L'intérêt d'une telle démarche est détaillé au Chapitre 5.

Carte de guidage routier Cette application a pour but d'assister le conducteur en affichant des informations de guidage vers sa destination. Cette application est spécifiée en détail au Chapitre 6.

4.4 Recherche

La France est un des pays où l'on s'intéresse le plus aux cartes à puce au niveau universitaire, ainsi d'ailleurs qu'au niveau industriel. On cite ci-dessous deux projets importants, caractéristiques de la diversité des thèmes de recherche liés aux cartes à puce.

4.4.1 Équipe Recherche et Développement Dossier Portable

Les membres de l'équipe RD2P du Laboratoire d'Informatique Fondamentale de Lille travaillent depuis plusieurs années avec la société Gemplus sur le thème des cartes à puces. Parmi leurs principaux travaux en cours, on peut citer :

L'étude de l'intégration de la carte dans une application distribuée par une approche orientée objet. Il s'agit du projet OSMOSE dont l'un des points clés est l'insertion de la carte sur un bus CORBA [23] [24] [25] [26] [27] [28] [29]

La définition et la spécification d'un agent de la carte. Ce concept permet à la carte qui n'est que rarement connectée au système d'information, de disposer d'un représentant permanent. Ce représentant est un agent de la carte qui est actif en permanence, en particulier par son implémentation dans un serveur du réseau [30].

L'étude et la définition des transactions entre la carte et son environnement. Il s'agit de répondre à la complexité croissante des applications que l'on désire encarter — notamment due aux cartes multiapplicatives — en respectant les contraintes de sécurité et de résistance aux défauts. Cela a conduit à la définition et à la conception de modèles généraux de transactions entre la carte et des serveurs extérieurs [31] [32] [33] [34].

L'étude du fonctionnement permanent de la carte. Jusqu'à présent, la carte n'est active que si elle est connectée à un lecteur. Les exigences de la mobilité et l'extension de ses prérogatives conduisent à envisager que la carte fonctionne de manière continue. Ceci peut

avoir des conséquences importantes sur le rôle qui lui est dévolu et sur ses caractéristiques internes.

Les membres du RD2P ont également beaucoup travaillé autour de l'intégration de la carte à puce dans des bases de données, pour aboutir au développement d'une carte intégrant un interpréteur du langage CQL, dérivé de SQL. Cette carte est aujourd'hui commercialisée par Gemplus sous le nom de GemDB. On pourra consulter à ce sujet [35], [36], [37], [38] et [39].

Tous ces travaux, qui s'orientent autour des nouvelles applications de la carte à puce, sont assez proches des nôtres et nous intéressent donc particulièrement.

4.4.2 Action de recherche coopérative INRIA

Cette action regroupe plusieurs projets dont notamment CRYSTAL de l'INRIA de Rocquencourt, OASIS de l'INRIA de Sophia-Antipolis et LANDE de l'IRISA de Rennes [40]. Les principaux thèmes étudiés concernent :

L'axiomatisation des sémantiques statique et dynamique de Java Card et la preuve de la sûreté du système de typage du langage à partir de cette axiomatisation. La propriété de sûreté est un préalable à toute spécification de politique de sécurité. Il est également envisagé de spécifier la machine virtuelle, sa sémantique, et le schéma de traduction entre Java Card et ByteCode de la machine virtuelle Java Card [41] [42].

La spécification de politiques de sécurité pour Java Card. La politique de sécurité définie pour Java Card diffère de celle spécifiée pour Java en permettant un partage plus ou moins restreint d'objets entre les applications chargées sur une carte. L'objectif est de formaliser cette politique de sécurité afin de pouvoir étudier son effet et les exigences qu'elle impose sur les composants d'une application Java Card. A partir de cette formalisation il sera possible d'étudier comment prouver la conformité d'un programme vis-à-vis de la politique de sécurité [43] [44] [45] [46].

L'optimisation d'applications Java Card. Il s'agit d'adapter et de formaliser des méthodes classiques d'optimisation de programmes Java Card ainsi que de développer de nouvelles techniques. Le but est de minimiser le temps d'exécution et la consommation de mémoire, tout en montrant que les optimisations ne mettent pas en péril la politique de sécurité.

La réalisation d'un environnement pour l'analyse sémantique de programmes Java Card. Cet environnement devrait servir d'outil de développement de programmes Java Card. Pendant la conception, on disposera d'informations sur le typage des variables (e.g. l'ensemble des types dynamiques que peut prendre une entité du programme) et sur les propriétés concernant la sécurité de l'application (e.g. l'ensemble des interactions avec l'environnement). Pendant l'exécution, on visualisera la topologie du graphe d'objets, leurs attributs, la constitution de sous systèmes et leur interaction générées par l'utilisation d'applications, les communications avec le monde extérieur, etc [47].

Cette action s'intéresse donc aux cartes à puce par le biais de méthodes formelles, ce qui est assez éloigné de notre approche. Elle permet cependant de montrer que les cartes à puce ne sont pas qu'un support et qu'elle peuvent susciter de véritables problèmes théoriques.

4.5 Industriels

Historiquement, la société Bull fut la première à s'intéresser au développement industriel des cartes à puce. Depuis, de nombreuses autres sociétés proposent leurs compétences, soit au niveau matériel, soit logiciel, voir même les deux. Parmi les plus connues, on pourra citer De La Rue, Gemplus, IBM, Motorola, Schlumberger, Sun Microsystems et bien sûr Visa, célèbre pour ses cartes bancaires.

La société Gemplus, avec laquelle le travail présenté dans ce rapport a été réalisé, est le numéro un mondial des solutions basées sur cartes à puce et cartes plastiques. Gemplus fabrique et commercialise des cartes à piste magnétique, cartes à mémoire et à microprocesseur, cartes à puce sans contact, étiquettes électroniques ainsi que des objets intelligents (ie : dotés d'un microprocesseur). La société conçoit et commercialise également des logiciels, outils de développement et des lecteurs de cartes. On lui doit notamment une implémentation de l'appel de méthode à distance (*Remote Method Invocation*) semblable à celle fournie avec l'interface applicative Java 2 (*Java 2 API*), qui permet à des méthodes situées sur la station d'accueil d'appeler des méthodes dans la carte plus facilement qu'avec le protocole de communication par APDUs décrit dans le Chapitre 3.

4.6 Synthèse

Comme on l'a vu dans ce chapitre, il existe à l'heure actuelle de nombreuses applications des cartes à puce. On doit cependant admettre que l'intérêt de la carte à puce n'est pas toujours flagrant pour toutes ces applications. On utilise aujourd'hui des cartes d'identité et de sécurité sociale sur papier sans que cela ne pose aucun problème, par exemple. La raison principale de l'utilisation des cartes à puce pour ce type d'applications est souvent économique (fidélisation du client avec les cartes de transport en commun, économie de papier pour les cartes bancaires qui coûtent moins chères aux banques que les chèques, etc...), et parfois simplement d'ordre pratique (un porte-monnaie électronique est moins encombrant que des pièces de monnaie par exemple). Heureusement, des équipes de recherches et certains industriels comme Gemplus travaillent actuellement sur des applications pour lesquelles la carte a une réelle valeur ajoutée. Le but de notre étude sera justement d'isoler les points forts des cartes à puce et d'étudier les services systèmes qui permettront d'exploiter au mieux ces caractéristiques. C'est ce que nous verrons dans la 2^e partie de ce rapport.

Deuxième partie

Analyse

Chapitre 5

Intérêts des cartes à puce

La première partie du travail réalisé dans cette analyse consiste à mettre en évidence les principaux intérêts des cartes à puce dans un environnement réparti. Il est intéressant de noter à quel point ces intérêts sont caractéristique de la technologie utilisée. En effet, la disponibilité est un avantage commun à tous les types de cartes à puce. La sécurité est quand à elle une valeur ajoutée de la présence d'un microprocesseur sur la carte. Les environnements d'exécution évolués permettent enfin d'obtenir des caractéristiques supplémentaires comme la multiapplicativité par exemple. Après avoir listé les principaux points forts des cartes à puce et mis en évidence la valeur ajoutée des cartes à microprocesseur, on s'intéressera à deux applications courantes, les cartes bancaires et les cartes SIM, afin d'évaluer l'utilisation qu'il est actuellement faite des technologies disponibles.

5.1 Disponibilité

Pour mettre en évidence l'intérêt des cartes à puce en termes de disponibilité, on se base sur un exemple d'application typique. Une chaîne de magasins souhaite proposer à ses clients les plus fidèles des réductions sur le montant de leurs achats. Les réductions sont calculées à partir de points accumulés par le client lors de ses achats.

Une première approche possible consiste à centraliser les points de tous les clients dans une base de données installée sur un serveur. On utilise un serveur de données plutôt qu'une base de données locale afin de permettre au client de bénéficier de ses réductions depuis n'importe quel magasin de la chaîne. Lorsqu'un client effectue un achat, le montant est envoyé au serveur qui calcule le nombre de points correspondant, incrémente le total du client et répond en renvoyant le montant réduit des achats. Si celui-ci choisit d'utiliser sa réduction, la station d'accueil envoie l'ordre au serveur de mettre à zéro le nombre de points du client et affiche le montant réduit. Cette solution qui concentre toute la charge de calcul sur le serveur permet de réduire le coût des stations locales (qui risquent d'être assez nombreuses) sans véritablement surcharger le serveur (les opérations effectuées pour calculer les points gagnés et le montant réduit sont négligeables par rapport à celles nécessaires à la gestion d'une base de données).

Cependant, la station doit communiquer avec le serveur par un réseau pour accéder au nombre de points du client. Ce réseau peut tomber en panne ou être congestionné. Même si on suppose que la station est capable de conserver le nombre de points gagnés lors de l'achat courant pour les enregistrer dans la base de données lorsque le réseau fonctionnera de nouveau, le client ne pourra bénéficier d'aucune réduction lors de son achat. Le service

est donc tributaire du bon fonctionnement du réseau.

Si l'on choisit de stocker les points du client sur une carte mémoire, on augmente considérablement la disponibilité des données. La station peut les lire et les modifier directement, sans passer par un réseau. Le système est donc plus fiable, et aussi beaucoup plus rapide puisque la liaison entre la station et le lecteur n'est utilisée que par un seul utilisateur à la fois.

On remarquera cependant que ces deux approches peuvent être complémentaires. Par exemple, on peut choisir de stocker les points des clients sur leur carte et d'utiliser la base de données comme sauvegarde. En effet, le client peut perdre sa carte ce qui reviendrait à lui faire perdre tous ses points accumulés. Si l'on sauvegarde le nombre de points dans la base de données, il sera très facile de donner une nouvelle carte au client, initialisée avec le nombre de points qu'il possédait. Pour ce type d'utilisation, une panne du réseau n'est pas vraiment un problème puisque la mise en cohérence de la sauvegarde peut parfaitement être différée. Le nouveau nombre de points de chaque client peut être stocké dans la station d'accueil et envoyé au serveur à la fin de chaque journée par exemple.

Il y a tout de même un problème avec l'approche à carte : si le client oublie sa carte, il ne pourra bénéficier du service. Cependant, dans ce cas, on peut considérer que c'est de la faute du client, alors que pour une panne du réseau, celui-ci serait en droit de se plaindre de ne pouvoir accéder au service.

5.2 Sécurité

De façon générale, les cartes à puce sont des systèmes particulièrement sûrs. Elles disposent en effet de protections matérielles qui rendent leur piratage difficile et coûteux. Sur la plupart des cartes, il est par exemple impossible d'extraire la puce sans la détruire irrémédiablement. Il est bien sûr toujours possible de casser n'importe quel système de sécurité, mais dans le cas des cartes à puce, le piratage matériel nécessite un équipement très coûteux et hors de portée de la plupart des particuliers et même des entreprises (microscope électronique, etc...).

D'un point de vue logiciel, le fait de disposer d'un microprocesseur sur la carte elle-même augmente considérablement la sécurité de l'application. En effet, tous les calculs qui devaient être effectués sur une station pourront l'être directement sur la carte. On n'aura donc plus besoin de copier les données sur la station pour effectuer des opérations dessus. Cela permet d'utiliser la carte avec des stations en libre accès, dans lesquelles on n'a pas forcément confiance comme par exemple dans le cadre d'une application de porte-monnaie électronique contenant des informations bancaires que l'on ne souhaite pas rendre accessibles à n'importe quel commerçant.

De plus, on peut mettre en place un système de filtrage des entrées qui garantit que les seuls accès possibles aux données sont ceux prévus lors du développement de l'application. En effet, les cartes à microprocesseur ne permettent pas à une application située sur la station d'accéder directement à leurs différentes mémoires (elles n'intègrent tout simplement pas le matériel nécessaire). Le seul moyen d'obtenir une donnée est donc d'envoyer une requête au logiciel sur la carte, qui aura tout loisir de vérifier si cette requête est autorisée ou non. Ainsi, pour le système de carte de fidélité ci-dessus, si on utilise une carte à microprocesseur et que l'on installe le logiciel de calcul des réductions dessus, il suffira de fournir à la station d'accueil une interface, où les seules opérations possibles seront `enregistrerNouvelAchat` et `utiliserPoints` par exemple. Cela protège les données contre une station défectueuse ou malveillante qui pourrait envoyer des ordres aléatoires à la carte pour pirater ou endommager les données.

Enfin, on peut désirer protéger non seulement les données, mais aussi le code d'une application. Un éditeur de logiciel pourra vendre son programme embarqué dans une carte à puce, en ne fournissant qu'une interface d'entrées/sorties pour dialoguer avec son environnement. Le code s'exécutera dans la carte, et ni lui ni les données n'auront besoin d'être chargés dans la station d'accueil. L'application entière sera donc protégée contre les duplications ou modifications interdites [48].

La sécurité des systèmes répartis étant de façon générale un domaine intéressant beaucoup de chercheurs, on trouve de nombreux projets concernant la sécurité des cartes à puce. On pourra consulter notamment [49] [50] [51] [52] [53] et [54] à ce sujet.

5.3 Intérêts supplémentaires des cartes Java

Avant l'introduction du langage Java Card, la plupart des applications destinées aux cartes à puce étaient écrites en assembleur, ce qui rendait le développement d'applications de taille conséquente très fastidieux et réservé à des programmeurs expérimentés dans les langages de bas niveau. De plus, chaque processeur ayant son propre langage d'assemblage, ces applications n'étaient pas du tout portables et devaient être entièrement ré-écrites chaque fois que l'on changeait de carte destination. Pour remédier à cela, certains fabricants vendaient avec leur matériel un compilateur C. Mais le problème de la portabilité se posait alors pour le compilateur lui-même, puisque la partie production de code devait être refaite pour chaque nouveau processeur.

L'utilisation du langage Java Card permet de se libérer de ces contraintes. La spécification très stricte de la machine virtuelle assure que le ByteCode produit par le compilateur Java sera interprété de la même façon quelque soit le matériel. Cela permet donc d'assurer la portabilité de l'application. De même, le compilateur Java étant lui-même entièrement écrit en Java, il est lui aussi portable et génère bien sûr exactement le même ByteCode quelle que soit la plateforme sur laquelle on l'exécute. De plus, le ByteCode étant un langage extrêmement compact (la plupart des instructions sont codées sur un ou deux octets) et d'assez haut niveau, les programmes générés seront en moyenne plus petits que leur équivalent en langage machine, ce qui peut être intéressant sur des systèmes comme les cartes à puces qui disposent d'une quantité assez réduite de mémoire. Enfin, le langage Java est un langage beaucoup plus simple à apprendre et à utiliser qu'un langage d'assemblage, ce qui évite d'avoir à confier le développement des applications à des programmeurs spécialistes et permet de réduire le temps et donc le coût de développement de ces applications.

Comme on l'a vu précédemment, la technologie Java Card permet d'implémenter plusieurs applications sur la même carte et rend possible la coopération entre ces applications, tout en garantissant la protection des données privées. Ainsi, dans le cas de notre exemple de carte de fidélité, une deuxième chaîne de magasin pourra s'associer à la première pour proposer un service équivalent. Chaque firme installera son application sur la carte, applications qui ne partageront que le nombre total de points. Les méthodes de calcul des points pourront être différentes et aucune application n'aura accès aux données ou au code de l'autre [55].

De plus, la technologie Java Card permet l'installation de nouvelles applications même après l'émission de la carte. Cela rend possible le développement d'une carte générique, sur laquelle le client pourra stocker les applications correspondant aux services auxquels il s'abonne. Par exemple, dans le cadre de notre exemple de carte de fidélité, la chaîne de magasins proposant ce service n'aura pas à émettre de nouvelle carte pour ses clients, mais pourra charger l'application directement sur leur carte générique. Ainsi chacun n'aura qu'une seule carte, ce qui est plus pratique que d'avoir une carte par application. On pourra mettre

en place un système de sauvegarde comme précédemment afin de simplifier la génération d'une nouvelle carte en cas de perte.

5.4 Confrontation disponibilité/sécurité

A l'issue de cette analyse des principaux intérêts des cartes à puce, il peut être intéressant de souligner l'opposition traditionnelle entre les principes de sécurité et de disponibilité (voir FIG. 5.1). Comme on l'a mis en évidence ci-dessus dans le cadre de notre carte de fidélité, le principal défaut de l'approche serveur de données est le manque de disponibilité engendré par la dépendance de l'application vis à vis du fonctionnement du réseau. Réciproquement, un point que l'on aurait aussi pu souligner est l'absence totale de sécurité si l'on choisit d'utiliser une carte mémoire pour stocker les points du client. Même si l'on crypte ces informations avant de les stocker dans la carte, rien n'empêche un utilisateur un peu expérimenté de récupérer ces données directement dans la mémoire de la carte et d'essayer de les décrypter. Et il est bien connu qu'aucun système de cryptage n'est inviolable. De plus, puisque la carte n'intègre pas de microprocesseur, les données devront être copiées sur la station pour être décryptées et utilisées. Comme on l'a dit précédemment, on n'a pas forcément confiance dans la station d'accueil sur laquelle pourrait s'exécuter un programme espion qui pourrait récupérer les données décryptées directement en mémoire. Il est vrai que pour une carte de fidélité, cela n'a pas vraiment d'importance, mais pour une carte bancaire par exemple, ce manque de sécurité peut avoir des conséquences désastreuses pour le client. C'est là qu'on comprend mieux la véritable valeur ajoutée des cartes à microprocesseur. En effet, ces cartes se comportent comme de véritables « serveurs mobiles », qui allient disponibilité et sécurité. Jusqu'à présent, il est vrai que leur coût important interdisait leur utilisation à grande échelle. De plus, les limitations matérielles étaient telles que l'on ne pouvait objectivement pas envisager de les utiliser comme support d'applications un temps soit peu évoluées. Des progrès importants ayant été accomplis ces dernières années dans ces deux domaines, il paraît certain que les cartes à microprocesseur vont se répandre et être utilisées pour des applications plus conséquentes, ce qui ne peut être que bénéfique pour l'utilisateur.

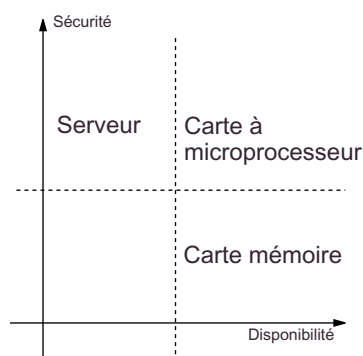


FIG. 5.1 – Comparaison des domaines de sécurité et de disponibilité

5.5 Étude de deux applications existantes

On détaille ci-dessous le fonctionnement des deux applications sur cartes à puce les plus couramment utilisées de nos jours : les cartes bancaires et les cartes SIM que l'on trouve dans les téléphones mobiles. Cette étude critique nous permet de mettre en évidence l'intérêt des nouvelles technologies dans le cadre précis de ces deux applications.

Ces deux applications nous permettent au passage de souligner le fait qu'en France, la plupart des gens sont maintenant habitués aux cartes à puce, puisque cela fait des années qu'ils s'en servent quotidiennement. Cette confiance de la population vis-à-vis des cartes à puce est très intéressante puisqu'elle favorise l'adoption des nouvelles applications. Cela n'est pas le cas dans d'autres pays, comme aux États-Unis par exemple. Un excès de confiance peut cependant être risqué, comme on le verra ci-dessous avec les cartes bancaires.

5.5.1 Les cartes bancaires

Sur le plan de la disponibilité, les cartes bancaires n'apportent pas de réel bénéfice. En effet, le service est toujours tributaire du bon fonctionnement du réseau puisque l'ordre de débit devra de toute façon être communiqué à la banque. Cela peut être gênant pour le client s'il ne peut pas régler ses achats, mais on ne voit pas comment éviter la communication entre le magasin et la banque. On pourrait concevoir un système dans lequel les transactions seraient mémorisées dans la station et effectuées quand le réseau serait opérationnel, mais cela empêcherait d'effectuer des vérifications, par exemple si la carte est en opposition. Ce problème de disponibilité se pose également lorsque l'utilisateur veut retirer de l'argent à un guichet automatique, sauf bien sûr si ce guichet est attaché à l'agence gérant le compte du client.

Les cartes bancaires sont l'exemple type d'applications pour lesquelles la sécurité est un problème critique. Malheureusement, à l'heure actuelle cette sécurité n'est absolument pas garantie par les systèmes utilisés. Bien qu'il soit extrêmement difficile d'obtenir des informations précises sur le fonctionnement exact des cartes bancaires (les fournisseurs étant bien naturellement peu enclins à dévoiler leurs systèmes de sécurité), on peut décrire de manière simplifiée leur principe. Les cartes actuelles sont en fait des cartes mémoire sur lesquelles sont stockées les informations nécessaires à la transaction (l'adresse du serveur de la banque, le numéro de compte du client, etc...). Ces informations sont bien sûr cryptées, à l'aide d'algorithmes et de clés complexes. Cependant, l'actualité récente nous montre si besoin était encore qu'aucun système de cryptage n'est infaillible, puisque qu'un ingénieur a récemment réussi à identifier l'un des algorithmes de cryptage utilisés avec les cartes bleues françaises. Bien qu'il semble qu'il n'y ait eu aucune volonté de nuire dans sa démarche, sa découverte compromet gravement la sécurité de toutes les cartes bancaires françaises, puisqu'elles utilisent toutes le même procédé de cryptage. Le fait de pouvoir accéder librement à l'information, même chiffrée, est une faille de sécurité importante. Et il reste bien évidemment le problème ahurissant des informations confidentielles imprimées sur la surface de la carte.

L'intérêt des cartes à microprocesseur semble donc évident pour améliorer la sécurité des cartes bancaires. Les organismes émetteurs semblent d'ailleurs avoir enfin pris conscience du manque flagrant de sécurité des systèmes actuels et certains imposent même dans leur nouvelle norme l'utilisation de cartes à microprocesseur (c'est notamment le cas du Groupement des Cartes Bancaires en France). Ces mesures semblent indispensables, surtout si l'on considère la formidable progression du commerce électronique. À l'heure actuelle, un client désirant acheter un produit sur Internet doit envoyer au fournisseur son numéro de carte de crédit et la date de validité. Cela pose bien sûr d'énormes problèmes de sécurité,

puisque non seulement la communication peut être interceptée, mais de plus rien ne garantit au client que le commerçant auquel il s'adresse est bien honnête. Au constate en effet de plus en plus de fraudes commises par des faussaires ayant construit un faux site de commerce électronique dans le seul but de récupérer des numéros de cartes bancaires. Certains systèmes plus sûrs existent à l'heure actuelle, comme le système de paiement NetBill par exemple [56], mais ils nécessitent la mise en place d'une infrastructure qui peut être complexe. La carte à microprocesseur peut avoir un rôle intéressant à jouer dans ce domaine. On pourrait imaginer un système de transactions commerciales inspiré du système NetBill utilisant la carte à microprocesseur : le client se connecte sur un site de commerce électronique et passe sa commande ; le serveur du fournisseur et la carte bancaire du client s'identifient mutuellement ; le fournisseur demande à la carte d'effectuer le transfert de fonds ; la carte envoie l'ordre de virement au serveur de la banque du client ; la banque effectue le transfert et informe le fournisseur qu'il a été payé. L'intérêt de ce système est qu'à aucun moment le serveur du fournisseur ne possède d'informations confidentielles au sujet du client (aucune information bancaire notamment). Le fournisseur n'intervient pas dans la transaction bancaire qui s'effectue entre la carte du client et le serveur de la banque. Ce système nécessite bien sûr la présence d'un lecteur de carte à puce en standard dans les PCs. Vu le prix réduit de ce matériel, cela sera vraisemblablement le cas dans les années à venir, si les applications sur la carte continuent à se développer.

5.5.2 Les cartes SIM

Les cartes SIM sont présentes dans la majorité des téléphones mobiles. Cette carte est une carte à microprocesseur qui intègre un logiciel d'identification du type défi/réponse, basé sur une clé secrète dépendante du code de l'utilisateur. Lorsque l'utilisateur essaie de se servir de son téléphone, celui-ci envoie son numéro d'identification unique au serveur de l'opérateur. Le serveur génère un nombre aléatoire x qu'il envoie au téléphone portable. La carte SIM crypte ce nombre à l'aide de la clé secrète k et renvoie le résultat $\{x\}_k$ au serveur de l'opérateur. De son côté, celui-ci crypte x à l'aide de la même clé secrète k , et obtient un résultat $\{x\}'_k$. Il lui suffit alors de comparer $\{x\}_k$ et $\{x\}'_k$ pour savoir s'il doit autoriser ou refuser l'accès au réseau à l'utilisateur.

Du point de vue de la sécurité, la carte ne semble pas vraiment indispensable. En effet, on pourrait obtenir une sécurité analogue en intégrant un circuit capable d'effectuer le cryptage directement dans le téléphone. Ce circuit pourrait être protégé grâce aux mêmes techniques que les processeurs des cartes à puce (verniss spéciaux, dispositifs d'autodestruction en cas de tentative d'extraction, etc...), bien qu'ici, on n'est pas vraiment besoin de ce type de protection. L'algorithme de cryptage est public et le code de l'utilisateur n'est pas stocké dans la mémoire persistante de la puce.

Cependant, la carte contient aussi des données personnelles à l'utilisateur (carnet d'adresse, préférences, etc...). Or il est possible d'utiliser sa carte SIM dans la plupart des téléphones mobiles, quelque soit le fabricant. Cela permet à l'utilisateur de conserver son environnement personnel lorsqu'il achète un nouveau téléphone ou s'il emprunte celui d'un collègue lorsque le sien tombe en panne par exemple. La carte SIM permet donc d'obtenir un bon niveau de disponibilité des données personnelles de l'utilisateur.

5.6 Synthèse

Comme on l'a mis en évidence dans ce chapitre, les principaux intérêts des cartes à microprocesseur sont la disponibilité des données et de l'application, conséquence logique de

leur présence sur la carte qui peut être considérée comme un périphérique local à la station, et la sécurité assurée aussi bien au niveau logiciel que matériel. Il convient maintenant de montrer en quoi ces points forts peuvent être exploités dans le cadre d'une application concrète, comme on le verra au Chapitre 6.

Chapitre 6

Spécification d'une application typique

Pour mettre en évidence l'apport des cartes à puce dans le cadre d'une application répartie précise, on a choisi de spécifier et de prototyper une application de guidage routier. Cette spécification nous a notamment permis d'isoler certains services systèmes pouvant être utiles au développement d'applications sur carte à puce. Ces services seront détaillés dans le Chapitre 7. Le prototypage fera quand à lui l'objet de la 3^e partie de ce rapport.

6.1 Architecture

Cette application est basée sur 4 composants matériels (voir FIG. 6.1) :

La carte à puce dans laquelle va s'exécuter le code de l'application.

La station d'accueil embarquée dans la voiture et composée d'un écran, d'un clavier et du lecteur de carte à puce.

Le GPS qui va permettre à l'application de connaître à chaque instant la position de la voiture.

Le réseau sans fil (GSM) qui relie l'application au serveur du fabricant. Ce composant est en fait facultatif comme on le verra ci-dessous.

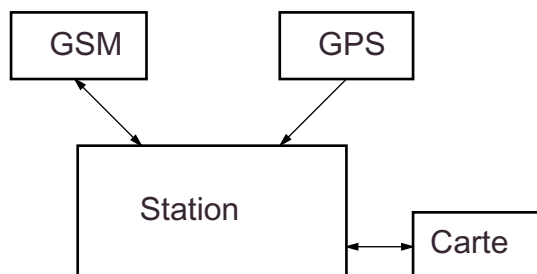


FIG. 6.1 – Architecture de l'application de guidage routier

6.2 Fonctionnement

Le client insère sa carte dans le lecteur et choisit sa destination dans la liste proposée. Cette liste peut être soit locale (elle peut par exemple être téléchargée par Internet et installée sur la station par l'utilisateur lui-même, qui pourra aller voir de temps à autre sur le site du fabricant si celui-ci propose de nouvelles cartes routières), soit téléchargée directement par l'application à chaque nouvelle utilisation, grâce au réseau sans fil. Cette solution est bien sûr plus pratique pour l'utilisateur, mais nécessite une communication par le GSM à chaque utilisation, ce qui peut finir par être coûteux. Au cas où il y aurait plusieurs fabricants de cartes routières, l'application devra scruter tous les sites les uns après les autres et faire l'intersection des destinations proposées, ou bien c'est l'utilisateur qui choisira lui-même son fournisseur préféré.

Une fois la destination choisie, deux cas sont possibles : soit cette destination est située dans une autre ville, soit dans la ville courante. Dans le premier cas, l'application contacte le serveur du fabricant par le biais du réseau sans fil et demande la liste des cartes routières nécessaires (ie : la liste des villes à traverser pour arriver à destination), en tenant compte de la position courante donnée par le GPS. Dans le deuxième cas, cette étape n'est pas nécessaire. On notera encore une fois que la connexion au réseau peut être évitée, puisque le client peut préparer son voyage en téléchargeant la liste des villes à traverser par Internet et la stocker dans la station d'accueil. Cette liste n'aura pas besoin d'être protégée puisqu'il ne s'agit que d'une suite d'identificateurs simples, comme des entiers par exemple. Elle pourra donc circuler librement sur le réseau et être stockée dans la station d'accueil sans être cryptée.

Une fois la liste des villes connue, l'application demande le chargement de la première carte (ie : celle de la ville courante). La carte est téléchargée depuis le serveur du fabricant vers la station d'accueil et ensuite chargée sur la carte à puce. Cette carte routière est cryptée, afin de protéger ses données et son format. Elle ne sera décryptée que dans la carte à puce, puisqu'on ne peut pas considérer la station comme sûre (un utilisateur pourrait y mettre un programme espion qui scruterait la mémoire pour récupérer les cartes routières). Encore une fois, la connexion au réseau n'est pas obligatoire, puisque l'utilisateur pourrait télécharger par Internet toutes les cartes routières dont il aura besoin avant de commencer son voyage, et les charger ensuite dans la station d'accueil. On voit bien l'intérêt du cryptage si on laisse l'utilisateur manipuler ainsi les cartes routières.

Une fois la carte routière chargée dans la carte à puce, elle est décryptée et l'application calcule le chemin optimal jusqu'à la sortie de la ville adéquate (ou la destination finale si on est dans la dernière ville). On peut supposer que la sortie de la ville la plus avantageuse est communiquée par le serveur du fabricant en même temps que la carte routière, compte tenu de la destination finale.

L'application peut maintenant commencer le guidage. Le GPS envoie périodiquement la position courante à l'application. Chaque changement significatif de position constitue un pas d'exécution du programme, et provoque une mise à jour de l'affichage sur l'écran de la station d'accueil. Cet affichage sera principalement composé d'une carte de la ville (ou d'une portion de la ville) courante sur laquelle seront superposées les informations de guidage (par exemple une flèche indiquant la direction à suivre).

Une fois la sortie de la ville atteinte, l'application pourra télécharger la carte de la ville suivante et recommencer le même processus jusqu'à la destination finale. La gestion du trajet entre les villes ne semble pas poser de problème particulier (par exemple, sur une autoroute, il suffit d'indiquer la sortie à prendre).

6.3 Fonctionnalités supplémentaires

En plus du fonctionnement de base détaillé ci-dessus, on aimerait pouvoir doter l'application d'un certain nombre de fonctionnalités supplémentaires intéressantes dont on donne quelques exemples ci-dessous :

Gestion des accidents En cas d'accident sur le trajet précalculé, le serveur du fabricant doit pouvoir prévenir l'application afin qu'elle puisse recalculer un nouveau chemin optimal. On pourra se servir du réseau sans fil pour envoyer l'information à la station qui la transmettra à la carte. On peut aussi donner à l'utilisateur la possibilité de forcer un recalcul du chemin, par exemple s'il se trouve confronté à un accident non détecté par le serveur du fabricant. Cet accident pourrait alors être signalé au serveur directement par l'utilisateur qui s'y trouve confronté. Cette deuxième solution permet d'éviter une connexion permanente avec le serveur du fabricant, qui serait vraisemblablement très onéreuse.

Paiement à l'utilisation On peut imaginer deux modes de paiement des cartes routières : soit à l'unité (ie : le client achète sa carte une fois pour toute et s'en sert autant de fois qu'il le désire), soit à l'utilisation (ie : le client doit payer à chaque fois qu'il se sert de la carte routière). Dans le premier cas, les cartes routières pourront être stockées dans la station d'accueil (ou même dans la carte à puce bien que cela risque de poser des problèmes compte tenu de la quantité très faible de mémoire disponible). Dans le deuxième cas, on devra mettre en place un système pour garantir que l'utilisateur ne pourra pas sauvegarder la carte pour s'en resservir plus tard. Un moyen possible consiste à attribuer une durée de vie à chaque carte routière.

Multiplicité des sources Il paraît raisonnable d'envisager que cette application soit proposée par plusieurs sociétés concurrentes. Dans ce cas, il serait intéressant de permettre au client d'acheter des cartes routières provenant de ces différentes sociétés, en fonction de leur prix. Il est cependant peu probable que le format de ces cartes soit standardisé, puisque le format même des données d'une application peut avoir une valeur commerciale. Dans ce cas, les sociétés vendant des cartes routières devront fournir avec ces cartes le module de code permettant de les lire et l'application devra être capable d'installer dynamiquement ce module de code pour se mettre à jour.

6.4 Avantages

La plupart des constructeurs de voitures proposent maintenant ce type d'applications d'aide à la conduite en option ou même en série dans leurs modèles haut de gamme. Cependant, les systèmes de ce type restent très onéreux pour le client (environ 10000F) et comportent deux principaux inconvénients qui sont le manque de disponibilité (ie : le système est physiquement intégré à la voiture et ne peut pas être déplacé dans une autre) et le manque de portabilité (le système fabriqué par un constructeur n'est pas forcément adapté aux véhicules d'un autre constructeur). Notre application présente donc plusieurs avantages par rapport aux systèmes existants :

Sécurité La carte à puce est le seul environnement que l'on peut considérer comme sûr et il est donc important que toutes les données sensibles (principalement les cartes routières) soient cryptées lorsqu'elles sont stockées sur la station ou quand elles transitent par le réseau. Le fait de pouvoir exécuter le code applicatif sur la carte est donc un grand avantage du point de vue de la sécurité puisqu'on n'aura pas besoin de décrypter ces données dans la station pour travailler dessus. Cette protection des cartes n'est pas compromise par l'affichage à l'écran car les données qui lui sont envoyées par la carte

à puce peuvent se limiter à de simples tableaux de pixels, difficilement exploitables par eux mêmes. Le programme lui même sera à l'abri dans la carte des duplications ou modifications illicites.

Disponibilité des données Comme on l'a vu, le réseau sans fil n'est pas indispensable au bon fonctionnement de l'application. Il est vrai que l'utilisateur doit de toute façon se connecter à Internet pour télécharger les cartes routières en autres, mais il le fait avant de commencer à utiliser le service et ne risque donc pas de se retrouver sans guidage à l'entrée d'une ville inconnue, comme cela pourrait se produire si le réseau sans fil tombait en panne brusquement. Les cartes routières stockées dans la station sont donc disponibles localement.

Disponibilité de l'application Le client ayant acheté l'application peut s'en servir dans n'importe quel véhicule équipé d'une station d'accueil. Par exemple, un voyageur arrivant dans une ville inconnue pourra louer une voiture et utiliser son application pour trouver son chemin. L'application stockée dans la carte est donc toujours disponible au client qui la transporte avec lui.

Portabilité En séparant le code applicatif de l'interface homme/machine, dépendante du matériel, on facilite le portage de l'application. Un fabricant de véhicule désirant supporter cette application n'aura qu'à implémenter une interface homme/machine adaptée à son matériel, sans toucher au code applicatif. Le format des échanges entre la station et la carte peut facilement être standardisé.

6.5 Synthèse

L'application de guidage routier présentée dans ce chapitre est un exemple typique d'application répartie pour laquelle les points forts de la carte à puce mis en évidence au Chapitre 5 permettent d'améliorer le service fourni. Au niveau de la sécurité, le fait de disposer d'un microprocesseur dans la carte permet de manipuler les cartes routières sous forme cryptée à l'extérieur et de les décrypter pour travailler dessus à l'intérieur de la carte. On n'a donc pas besoin de faire confiance à la station d'accueil. Du point de vue de la disponibilité, l'application encartée n'est pas liée à un véhicule donné et peut donc être utilisée dans n'importe quelle voiture équipée de l'architecture matérielle définie plus haut. De même, les cartes routières téléchargées au préalable par l'utilisateur pourront être considérées comme des données locales puisqu'elles se trouveront dans la station (on peut même envisager que ces cartes routières soient stockées dans la carte à puce, ce qui permettrait à l'utilisateur de les transporter avec lui s'il change de véhicule).

Cette application nous permet en outre de mettre en évidence un certain nombre de services systèmes qui nous paraissent utiles au développement d'application sur la carte. Dans le cadre précis de cette application, on aura au moins besoin d'un service de cryptage. Un service de compression des cartes routières pourrait également être utile, notamment si l'on compte stocker celles-ci dans la carte à puce. Cela permettrait aussi d'accélérer les transferts de données via le réseau. Enfin, on aura besoin d'un service de chargement de code dynamique si l'on souhaite permettre au client d'utiliser des cartes routières provenant de fournisseurs différents, comme on l'a expliqué plus haut. Ces services, en autres, seront détaillés au Chapitre 7.

Chapitre 7

Services systèmes

Un des buts de cette étude est de proposer des services systèmes pouvant faciliter le développement d'applications sur les cartes à puce. En évaluant les services couramment proposés par les concepteurs d'environnements d'exécution, on a pu mettre en évidence des lacunes gênantes pour le programmeur d'applications. De même, en se basant sur l'exemple de guidage routier présenté au Chapitre 6, on a isolé un certain nombre de services qui auraient pu nous aider lors du prototypage de l'application. Certains de ces services seront vraisemblablement implémentés dans un futur proche.

7.1 Logiciel de base

Par « logiciel de base », on entend non seulement les systèmes d'exploitations, mais aussi les environnements d'exécution de plus haut niveau fournissant aux applications des services de base telle que la gestion de la mémoire par exemple. Les machines virtuelles tombent donc dans cette catégorie. Comme on peut le voir ci-dessous, il existe plusieurs types de systèmes de base, chacun répondant à des besoins particuliers. Cette diversité nous paraît très profitable au développeur d'applications et il ne nous semble donc pas nécessaire de chercher de nouveaux systèmes de base.

7.1.1 Système intégré

L'application peut gérer directement les ressources matérielles dont elle a besoin. Par exemple, l'application de carte de fidélité vue au Chapitre 5 pourrait s'occuper elle-même de la gestion de la mémoire persistante dont elle a besoin pour stocker les points, ou bien de la réception et de l'émission des APDUs. On voit tout de suite les inconvénients d'un tel choix, puisque cela augmente le travail du programmeur d'applications qui doit lui-même prendre en charge des aspects de bas niveau souvent complexes. De plus, cela rend l'application fortement dépendante du matériel utilisé dans la carte, et donc très peu portable. Enfin, si l'on suppose que l'on dispose d'un mécanisme supplémentaire permettant d'implémenter et d'utiliser plusieurs applications sur la même carte, on devra implémenter la gestion des ressources dans chaque application, ce qui explique pourquoi ces systèmes ne sont en général utilisés que sur des cartes monoapplicatives. Dans ce cas, cela permet de n'implémenter que les services dont on a véritablement besoin, et donc d'augmenter la compacité et la rapidité de l'application, ce qui est intéressant sur des systèmes disposant d'une quantité très faible de mémoire et d'une puissance de calcul limitée. Ces systèmes peuvent donc être intéressants

pour des applications simples implémentées sur des cartes bas de gamme.

7.1.2 Système dédié

Le système d'exploitation intégré sur la carte peut être dédié à un type d'applications particulier. On peut par exemple installer un système de gestion de base de données sur la carte et ensuite développer des applications autour de ce système, ou bien fournir un système de fichiers UNIX afin de pouvoir monter la carte sur une station de travail [57]. Ce type de gestion est intéressant car il permet de cibler les services proposés tout en restant plus général que le précédent, notamment car il évite d'avoir à recoder les services pour chaque applications. Il est néanmoins limité à un type d'applications particulier.

7.1.3 Système générique

On peut enfin essayer de fournir un système d'exploitation générique, qui permettra d'implémenter et d'utiliser facilement n'importe quel type d'application. Ce peut être un système UNIX ou bien l'environnement d'exécution Java Card (qui est lui même basé sur un système d'exploitation minimal fournissant des primitives d'allocations de la mémoire et de communication très restreintes) par exemple. L'avantage de ces systèmes est bien évidemment qu'ils fournissent un ensemble de services généraux (gestion évoluée de la mémoire, communication interapplications, etc...) qui permettent au développeur de coder son application presque comme sur une station de travail. Comme on l'a vu avec la technologie Java Card par exemple, les contraintes matérielles, principalement la taille réduite de la mémoire, ne permettent pas d'implémenter un système équivalent à ceux qu'on trouve sur les stations de travail. Mais les systèmes réduits fournis permettent tout de même de coder facilement un grand nombre d'applications différentes. L'inconvénient de ces systèmes est la place mémoire et la puissance de calcul nécessaires à leur gestion, qui en restreignent l'utilisation à des cartes haut de gamme. On peut citer, en plus de l'environnement Java Card, le système MULTOS qui fournit un support d'exécution pour des programmes écrit en C et compilés vers un langage intermédiaire qui sera interprété sur la carte. C'est une démarche assez proche de celle de Sun avec Java Card et les fabricants des deux environnements travaillent d'ailleurs de concert afin de garantir à terme la compatibilité des deux produits. MULTOS est développé par un consortium réunissant Gemplus, Hitachi, Mastercard International, Mondex International, Motorola et Siemens [58]. A l'opposé de cette démarche d'ouverture, on peut citer le système propriétaire Windows CE pour carte à puce de Microsoft, qui dispose de la facilité d'utilisation caractéristique des systèmes Windows, du moins si l'on se restreint à utiliser les outils de développement de Microsoft (principalement Visual Basic et Visual C++), pour des applications destinées à s'exécuter dans un environnement composés exclusivement de PC fonctionnant sous Windows [59].

7.2 Communication station ↔ carte

7.2.1 Invocation de méthodes distantes

La norme ISO7816-4 définit un protocole de communication de très bas niveau assez peu pratique à utiliser. La technologie Java Card fournit au dessus de ce protocole un mécanisme d'invocation de méthodes à distance qui permet aux méthodes situées sur la station d'accueil d'appeler les méthodes stockées dans la carte par le biais de la méthode `process`. Ce système facilite le travail du programmeur, bien qu'il soit encore obligé de manipuler des APDUs pour sélectionner la méthode qu'il désire invoquer et lui passer des paramètres. Un

service d'invocation de méthodes à distance de plus haut niveau (comparable par exemple à celui fourni par l'interface applicative Java 2 disponible sur les stations de travail, par exemple (*Java RMI*)) est un plus indéniable, mais n'est malheureusement pas spécifié dans l'architecture Java Card 2.1. Ces services sont donc en général fournis par l'implémenteur de l'environnement d'exécution et ne sont donc par définition pas standardisés. Il est cependant assez probable qu'une version de RMI inspirée d'une de ces implémentations propriétaires soit spécifiée dans une version future de Java Card, peut-être même la version 2.2.

7.2.2 Invocation bidirectionnelle de méthodes distantes

La définition même du protocole de communication par APDUs implique que les méthodes stockées sur la carte soient passives, c'est à dire qu'elles peuvent être invoquées par le programme s'exécutant sur la station d'accueil et fournir une réponse, mais qu'elles ne peuvent pas prendre l'initiative d'appeler une méthode de la station. La carte à puce peut donc être assimilée à un serveur de calcul qui répond aux requêtes du client qui s'exécute sur la station. Ce modèle peut être inadapté, puisque le programme principal de l'application pourrait parfaitement s'exécuter dans la carte, et c'est la station d'accueil qui jouerait le rôle de serveur, par exemple pour l'affichage d'une interface utilisateur. Un service d'invocation de méthodes à distance dans le sens carte \rightarrow station pourrait donc être utile. On peut même considérer que la carte et la station jouent toutes deux le rôle de serveur, chacune à son tour. Il pourrait donc être intéressant de fournir un système d'invocation de méthodes à distance bidirectionnel, pour fournir une certaine souplesse de programmation au développeur d'applications.

7.2.3 Invocations imbriquées de méthodes distantes

Une fois que l'on dispose de l'invocation bidirectionnelle, il peut être intéressant de supporter l'invocation imbriquée de méthodes distantes (voir FIG. 7.1). Soit par exemple une application de cryptage encartée. La station envoie à la carte une commande de cryptage d'une donnée x (**Commande A**). Pour effectuer le cryptage, la carte doit contacter le serveur du fabricant pour obtenir une clé k par exemple. Elle envoie donc une commande à la station pour que celle-ci contacte le serveur du fabricant (**Commande B**). Une fois que celui-ci a répondu, la station renvoie la clé k à la carte (**Réponse B**) qui effectue le cryptage et renvoie la valeur cryptée $\{x\}_k$ (**Réponse A**). On notera que dans ce sens (ie : c'est la station qui émet la première commande), ce schéma ne pose pas de problème particulier de réalisation. En effet, le code sur la station d'accueil peut parfaitement comporter plusieurs flots d'exécution (*multithreading*) puisqu'il est écrit en Java standard. Par contre, le schéma inverse (ie : c'est la carte qui émet la première commande) semble beaucoup plus difficile à réaliser puisque la machine virtuelle Java Card ne supporte pas les processus légers Java. Le même genre de problèmes se poserait d'ailleurs si l'on souhaitait implémenter un mécanisme d'appel de méthodes distantes asynchrone de la carte vers la station, alors que cela ne poserait pas de problème particulier de la station vers la carte.

7.2.4 Alternatives à l'invocation de méthodes à distance

L'invocation de méthodes à distance étant le schéma de communication le plus simple pour les applications réparties, il n'est pas étonnant que ce soit celui qui a été choisi pour les cartes à puce. Il existe cependant d'autres formes de communication entre les applications sur la carte et celles sur la station d'accueil qu'il paraît intéressant d'étudier.

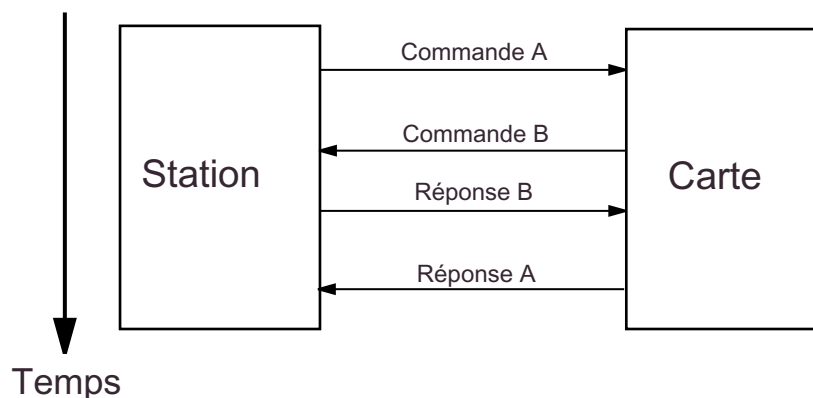


FIG. 7.1 – *Invocations imbriquées de méthodes distantes*

Un système de communication par messages pourrait être mis en place. Un des intérêt principaux d'un bus à messages est de permettre la diffusion d'une requête ou d'une information à plusieurs destinataires, alors que l'appel de procédures à distance impose une communication point à point. Ainsi, l'application active sur la carte pourra envoyer des messages à tous ou certains des composants de l'application situés sur la station d'accueil. Ce peut être intéressant, par exemple si l'application dans la carte souhaite communiquer avec plusieurs machines reliées à la station d'accueil qui sert alors de routeur. Par contre, cela ne semble pas véritablement intéressant dans le sens station \rightarrow carte, dans la mesure où il ne peut y avoir qu'une seule application active en même temps sur la carte, et donc qu'un seul destinataire potentiel pour un message provenant de la station d'accueil.

D'une façon assez similaire, il pourrait être intéressant de disposer d'un système de gestion des événements. Ainsi, dans notre application de guidage routier, le GPS pourrait envoyer périodiquement un événement pour signifier à l'application que la voiture a changé de position et lui donner la nouvelle. De même, l'application pourrait réagir à cette notification en envoyant à la station d'accueil un événement pour lui demander de mettre à jour l'affichage en tenant compte de la nouvelle position. Un tel système de gestion des messages pourrait d'ailleurs parfaitement être implanté sur un bus à messages comme vu précédemment.

Dans le cas d'une application nécessitant beaucoup d'échanges de données, on pourrait concevoir un mécanisme de transmission par flot de données. Chaque échange serait effectué selon le schéma « **connexion ; envoie des données ; déconnexion** ». Le fait de fournir à l'application une primitive **envoie de données** permet à celle-ci de s'affranchir de la nécessité de découper elle même les données en APDUs pour les recomposer ensuite sur la carte, et permet de travailler sur des flots de données qui représentent une abstraction pouvant simplifier l'architecture de l'application (ie : elle pourra simplement être structurée selon le modèle de machine séquentielle bien connu pour lequel on dispose de beaucoup d'algorithmes dans la littérature).

Enfin, un modèle à agents mobiles serait aussi envisageable. On pourrait par exemple envisager une application de configuration, comportant un agent stocké sur la carte, et qui se déploierait et effectuerait son travail de configuration sur les stations auxquelles la carte serait connectée. Cela pourrait également permettre de faciliter les opérations de mise à

jour de logiciels, l'administrateur n'ayant plus qu'à insérer sa carte dans la station et laisser l'agent faire son travail.

7.3 Chargement de code

Comme on l'a vu, l'installateur d'application fourni par le constructeur de la carte ne permet en général pas à une application de commander le chargement de code. C'est une restriction par rapport à Java, qui permet notamment d'utiliser des classes distantes (*remote classes*) comme si elles étaient locales. Le téléchargement est alors implicitement effectué par la machine virtuelle. Ce mécanisme est rendu possible notamment par le fait que l'édition de lien en Java est dynamique et qu'une application peut utiliser du code qui n'était pas présent à la compilation. En Java Card, l'édition de lien est statique et la machine virtuelle n'est pas capable de télécharger du code distant. On aimerait donc pouvoir supprimer cette restriction et implémenter un service de chargement dynamique de code. Cela pourrait être utile par exemple pour mettre à jour l'application sur la carte comme dans notre application de guidage routier, pour permettre l'installation de modules de code permettant de lire des cartes routières de formats différents. On peut donc imaginer que l'application soit capable de se rendre compte qu'elle ne possède pas le module nécessaire et qu'elle soit capable de le télécharger et de l'installer de façon transparente pour l'utilisateur. Ce mécanisme de chargement de code sera vraisemblablement basé sur le schéma d'appel de méthode à distance dans le sens carte → station présenté plus haut, puisque c'est de la carte que partira l'ordre de chargement.

7.4 Gestion de la mémoire

La gestion de la mémoire dans Java Card est un des aspects les plus contraignants du système. Comme on l'a dit au Chapitre 3, la machine virtuelle ne dispose pas de ramasse-miettes. Cela signifie que les objets alloués en mémoire persistante (grâce à l'instruction **new** du langage) ou même en mémoire volatile (grâce à des fonctions spécifiques) ne seront jamais désalloués et persisteront jusqu'à ce qu'on réinitialise la carte (dans le cas d'objets alloués en mémoire volatile, le contenu de l'objet sera perdu quand on retirera la carte du lecteur, mais la zone sera toujours marquée comme occupée). Il n'est donc pas possible de programmer comme en Java standard, en allouant des objets quand on en a besoin tout au long du programme. On risquerait alors de saturer rapidement la mémoire puisque ces objets seraient réalloués à chaque exécution du programme et jamais désalloués. C'est pour cela que les programmeurs d'applications Java Card initialisent leurs objets dans la méthode **install** qui ne sera appelée qu'une fois lorsque l'application sera chargée sur la carte. Cela contraint donc le programmeur à évaluer statiquement la quantité de mémoire maximale utilisée par son application. Ce calcul peut être très difficile voir impossible à effectuer, par exemple si la taille d'un objet à allouer dépend d'un choix de l'utilisateur. Ce problème ne paraît pas soluble, à moins de réussir à implémenter un ramasse-miettes sur la carte à puce, ce qui risque d'être extrêmement difficile vu la puissance de calcul et la mémoire disponible (une solution plus simple pourrait consister à fournir une instruction de libération explicite de la mémoire, comme la fonction **free** en C par exemple, mais cela violerait la spécification Java qui interdit que l'utilisateur puisse désallouer lui-même des objets, pour éviter les problèmes de pointeurs nuls que connaissent justement les programmeurs C). En dehors de ce problème de désallocation de mémoire, la contrainte principale pour le programmeur reste tout de même la quantité extrêmement faible de mémoire disponible.

Pour palier à ce manque de mémoire, on pourrait utiliser la mémoire installée sur la station d'accueil. En effet, ajouter quelques méga-octets de RAM dans la station est très bon marché et ne pose pas de problème technique, alors que la mémoire embarquée dans la carte est beaucoup plus coûteuse et limitée par des contraintes matérielles dues à la miniaturisation nécessaire. Même si on peut penser que les progrès techniques permettront d'augmenter la quantité de mémoire embarquées, il est vraisemblable que la demande en espace mémoire des applications augmentera elle aussi et on sera toujours confronté au même problème. Il pourrait donc être intéressant d'implémenter un service de déchargement de la mémoire de la carte dans la mémoire de la station, tout comme UNIX permet de décharger des pages de mémoire vive sur le disque dur par exemple. Un tel mécanisme serait vraisemblablement basé sur les techniques de cryptage et de compression de données détaillées ci-dessous, afin d'assurer la sécurité et l'efficacité du service [60]. De plus, si on veut pouvoir décharger des modules de code inutilisés et les recharger ensuite, on aura besoin du service de chargement dynamique présenté ci-dessus, à moins que l'on ne permette le déchargement de pages de code (c'est à dire de pages de mémoire contenant du code) et non pas de modules complets, ce qui reviendrait en fait à traiter le code comme des données.

7.5 Authentification

La carte doit disposer d'un service d'authentification de l'utilisateur. Certaines cartes, comme les cartes bancaires par exemple, permettent une identification grâce à un code personnel. Ce type de code est souvent assez court (4 chiffres pour une carte bancaire) et donc particulièrement vulnérable aux attaques, même à une simple attaque exhaustive. De façon plus générale, on sait qu'aucun code n'est inviolable, et que le niveau de sécurité d'un système de protection est souvent mesuré en fonction du temps nécessaire pour le percer. Certaines cartes proposent donc des codes personnels de plus grande taille, mais cela peut se révéler encore plus dangereux que des codes normaux, puisque les utilisateurs risquent d'avoir du mal à mémoriser beaucoup de chiffres et pourraient noter leur code par écrit. Des solutions plus avancées sont actuellement à l'essai, comme l'identification par numérisation des empreintes digitales ou de l'iris de l'oeil de l'utilisateur. L'image numérique utilisée peut être assimilée à un très grand code, ce qui assure un bon niveau de sécurité, code qui n'aura bien sûr pas à être mémorisé par l'utilisateur. Ces systèmes semblent appelés à se développer, bien que leur coût élevé reste encore un handicap important [61].

7.6 Intégrité et confidentialité des données

Le cryptage des données peut être une façon efficace de garantir leur non falsification, tout en assurant une certaine confidentialité. Ces techniques sont nécessaires si l'on veut pouvoir échanger des données avec une station d'accueil dans laquelle on n'a pas forcément confiance. La plupart des fabricants de cartes proposent des services de cryptage implantés matériellement dans le processeur. Cela permet d'obtenir une vitesse d'exécution bien supérieure à celle qu'on aurait pu obtenir grâce à un algorithme logiciel. Une autre technique fréquemment utilisée consiste à calculer une somme de contrôle (*checksum*) ou un code à redondance cyclique (*Cyclic Redundancy Code*) qui permettent de détecter la corruption des données et des programmes. On assure ainsi la protection contre la falsification, mais sans confidentialité. Ces techniques sont aujourd'hui maîtrisées et semblent parfaitement adaptées aux besoins actuels.

7.7 Compression

Pour pallier au manque de mémoire sur les cartes à puce, on peut mettre en place un service de compression des données. Cela permet de réduire la taille des données stockées en mémoire persistante qui ne sont pas actuellement utilisées. De même, puisqu'on sait que la vitesse de transfert entre la carte et la station est assez faible, on pourrait avoir intérêt à compresser les données échangées, comme c'est souvent le cas avec les fichiers téléchargés sur Internet. Le problème avec ces algorithmes de compression est qu'ils sont en général assez coûteux en puissance de calcul, et cela risque donc de ralentir l'exécution de l'application. L'idéal serait de disposer d'algorithmes implantés matériellement dans le processeur, comme c'est le cas avec les algorithmes de cryptage.

7.8 Synthèse

Comme on l'a vu dans ce chapitre, il reste encore beaucoup de services systèmes à implémenter sur la carte à puce. Des services comme l'invocation bidirectionnelle de méthodes distantes, le chargeur dynamique de code ou la gestion d'une mémoire de déchargement sur la station sont nécessaires au développement d'applications évoluées sur la carte. De façon plus générale, on a vu dans cette partie que les cartes à microprocesseur constituaient un support privilégié pour des applications réparties nécessitant à la fois une forte disponibilité et un haut niveau de sécurité pour les données et le code applicatif. Il convient maintenant de permettre d'exploiter au mieux ces caractéristiques intrinsèques en fournissant au programmeur les services dont il a besoin pour développer ses applications. Il reste cependant à s'assurer que les caractéristiques matérielles (principalement la faible taille mémoire et la puissance de calcul limitée) des cartes actuelles permettront d'implémenter de manière raisonnablement efficace ces services. Des éléments de réponse à cette question seront donnés dans la 3^e partie de ce rapport.

Troisième partie

Prototypage

Chapitre 8

Prototypage

A l'issue du travail d'étude présenté dans la 2^e partie de ce rapport, on a choisi d'implémenter un prototype de l'application de guidage routier présentée au Chapitre 6. Cette étape nous a permis de nous rendre compte des limitations réelles de la carte à puce, principalement au niveau matériel ce qui est particulièrement important notamment quand aux choix des services systèmes qu'il paraît réaliste de vouloir implémenter.

8.1 Environnement de développement

On utilise le kit de développement sur carte à puce gracieusement fourni par la société Gemplus (*GemXpresso RAD 211*). Les cartes fournies avec ce kit intègrent un processeur 8 bits fonctionnant à 5 MHz, 32 Ko de ROM, 32 Ko d'EEPROM et 2 Ko de RAM. La machine virtuelle stockée en ROM respecte la spécification Java Card 2.1 de Sun Microsystem. Cette machine virtuelle est basée sur un système d'exploitation propriétaire compatible avec les normes ISO7816-3 et ISO7816-4. Il propose entre autres des fonctions d'entrées/sorties, de gestion de la mémoire et de cryptographie. La carte intègre enfin la partie serveur de l'architecture OpenCard dont on a parlé plus haut et qui permet notamment l'authentification sécurisée des utilisateurs et des applications. Au final, cela laisse environ 20 Ko d'EEPROM et 1.5 Ko de RAM disponibles pour les applications.

8.2 Protocole de développement

Le développement d'applications Java Card suit un protocole standard :

Développement de la partie serveur C'est le code qui se trouvera sur la carte. Ce code est un ensemble de fonctions de traitement, chacune identifiée par un tuple $\langle \text{CLA}, \text{INS}, \text{P1}, \text{P2} \rangle$ comme vu au Chapitre 3. Le langage et la machine virtuelle Java Card étant très limités par rapport à Java, c'est généralement cette partie du développement qui pose le plus de problèmes techniques. De plus, comme on l'a dit précédemment, l'absence de ramasse-miettes oblige le programmeur à allouer toutes les variables de taille conséquente (ie : les tableaux et les objets) dans le constructeur de l'application, qui sera appelé une seule fois lors de l'installation (et non pas de la sélection) de l'application. Cela peut être gênant si les tailles des variables à allouer ne sont connues qu'à l'exécution et oblige le programmeur à évaluer à priori ces tailles. Vu la taille de la mémoire disponible sur la carte, il est crucial d'optimiser l'occupation mémoire et donc de l'évaluer correctement. On regrette à ce sujet l'absence d'un outil capable de calculer

la place mémoire maximale utilisée à l'exécution (Gemplus fournit un outil capable de calculer la taille mémoire occupée par l'application statiquement, mais ne tient pas compte des allocations dynamiques (ie : à l'aide de l'opérateur `new`)). Pour plus de détail sur la programmation d'applications Java Card, on pourra se référer à [62], [63], [64], [65] et [66]. Il existe aussi un groupe de discussions `alt.technology.smartcards`, sur lequel circulent de très nombreuses informations techniques intéressantes.

Développement de la partie client Cette partie de l'application, qui va se trouver sur la station, peut être programmée en Java et est donc beaucoup plus facile à développer que la précédente. L'interface applicative de l'OpenCard fournit une méthode, `ResponseAPDU send(CommandAPDU)`, qui permet d'envoyer facilement un APDU commande à la carte et de récupérer l'APDU réponse en résultat. On note bien l'aspect synchrone de cet appel d'une fonction de la carte, caractéristique du mode de communication par appel de procédure à distance. La seule difficulté réside dans le découpage des paramètres passés aux fonctions de la carte pour qu'ils tiennent dans des APDU commande de taille réduite.

Conversion de la partie serveur La machine virtuelle Java Card n'est pas capable de lire les fichiers `.class` produits par le compilateur Java. C'est une des restrictions dues au manque de place mémoire sur la carte à puce (pour information, une machine virtuelle Java version 1.2 exécutant sous UNIX une boucle infinie vide occupe environ 6 Mo de mémoire). On utilise donc un logiciel capable de convertir un fichier `.class` standard vers un format simplifié que la machine virtuelle Java Card pourra interpréter facilement. De plus, comme on utilise le compilateur Java standard pour compiler les applications, on doit vérifier que la partie serveur respecte bien les limitations du langage Java Card (ex : pas de réels, pas de chaînes de caractères, etc...). C'est également un des buts de cette phase de conversion.

Chargement de la partie serveur sur la carte Une fois l'application convertie, elle doit être chargée sur la carte. Gemplus nous fournit un ensemble de méthodes Java permettant d'effectuer cette installation. Ces méthodes peuvent être utilisées soit dans un programme indépendant (ie : un chargeur d'application), soit directement dans le code de la partie client de l'application. Cela signifie que l'on peut réaliser des applications Java Card capables de s'installer automatiquement sur la carte. Malheureusement, ces méthodes ne peuvent être utilisées dans la partie serveur et ne permettent en outre que de charger des applications complètes et non pas des modules à ajouter à une application déjà chargée. La levée de ces restrictions permettrait d'obtenir le service de chargement de code dynamique décrit précédemment. Après l'avoir chargée, l'installateur appelle la méthode `install` de l'application qui est maintenant prête à être utilisée.

8.3 Restrictions

Tout au long du développement de notre prototype, on a cherché à rester le plus proche possible de la spécification présentée au Chapitre 6. Par manque de temps, nous n'avons malheureusement pas pu implémenter toutes les fonctionnalités détaillées. Le prototype réalisé ne gère pas les accidents, ne simule pas la facturation des cartes et ne reconnaît qu'un format de carte routière. Les cartes routières sont stockées sous formes de fichiers locaux à la station.

8.4 Performances

On a été constamment confronté au problème de la taille mémoire disponible. Il a fallu notamment restreindre au strict minimum les variables utilisées par l'algorithme permettant de trouver le chemin optimal dans la ville, qui est de loin la partie la plus coûteuse en taille mémoire et en puissance de calcul de l'application. On a également été obligé de se limiter à des cartes routières de très petite taille (12x12 cases dans l'exemple d'exécution détaillé ci-dessous), à cause de la place mémoire qu'elles occupent sur la carte bien sûr, mais aussi et surtout à cause du temps nécessaire pour trouver le chemin optimal dans la ville.

Les cartes routières étant assimilées à des labyrinthes, l'algorithme de calcul du chemin optimal utilisé est un algorithme par inondation classique des problèmes de ce type [60]. Son principe est simple : en partant de la position initiale, on avance d'une case dans toutes les directions possibles à chaque itération. Chaque case visitée est étiquetée avec le numéro de la case à partir de laquelle on y a accédé. Une fois la destination atteinte, il suffit de refaire le chemin en sens inverse pour retrouver le chemin optimal depuis le départ. On est sûr que le chemin trouvé est le meilleur, puisque cet algorithme revient en fait à un simple parcours en largeur d'abord de l'arbre (virtuel) de tous les chemins possibles (voir FIG. 8.1).

```
procedure trouver_chemin_optimal
  marquer(position_initiale, visitee)
  ajouter_en_fin(position_initiale, liste)
  tant_que non_vider(liste)
    case_courante = extraire_premier_element(liste)
// c'est une amélioration de l'algorithme afin d'éviter de finir de parcourir
// tout l'arbre même quand on a déjà trouvé le chemin optimal
    si case_courante = destination alors
      fin_procedure
    fin_si
// un algorithme plus raffiné pourrait prendre en compte plus de directions
// pour_chaque direction parmi (NORD, EST, SUD, OUEST)
// visitable renvoie vrai ssi la case n'est pas un obstacle et n'a pas déjà
// été visitée
    si visitable(case_suivante(case_courante, direction)) alors
      marquer(case_suivante(case_courante, direction), visitee)
// on étiquette une case avec la case d'où on venait
      etiquetter(case_suivante(case_courante, direction), case_courante)
      ajouter_en_fin(case_suivante(case_courante, direction), liste)
    fin_si
  fin_pour_chaque
fin_tant_que
fin_procedure
```

FIG. 8.1 – *Algorithme de recherche du chemin optimal*

On donne ci-dessus une courbe (voir FIG. 8.2) illustrant les temps de calcul en fonction du nombre de cases des cartes routières (les cartes sont carrées). Il faut noter que la vitesse de l'algorithme étant dépendante de la complexité du labyrinthe (ie : principalement le nombre de cases entre le point de départ et la sortie), ces mesures ne sont qu'approximatives bien qu'on ait essayé de construire des labyrinthes les plus semblables possible. Les mesures ont été effectuées pour des cartes 8x8, 12x12, 16x16 et 20x20 (la taille doit être un multiple de quatre pour simplifier l'affichage). Au delà de 20x20, les cartes routières sont trop volumineuses pour être stockables dans la mémoire de la carte (non pas à cause de la carte routière elle-même mais à cause de la taille des variables (tableaux) intermédiaires nécessaires à l'algorithme de recherche du chemin optimal). La forme parabolique de la courbe nous laisse

supposer que les temps de calcul deviendront de toute façon rapidement trop importants pour que l'application puisse raisonnablement être utilisée avec des cartes routières de tailles supérieures.

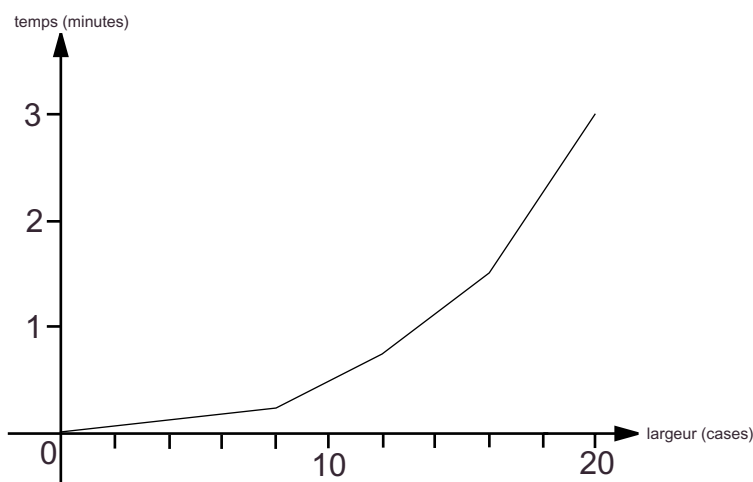


FIG. 8.2 – Performances de l'algorithme de calcul du chemin optimal

8.5 Exemple d'exécution

On détaille ici un scénario d'utilisation de notre prototype.

Lorsque l'utilisateur lance l'application, une fenêtre apparaît pour demander l'insertion de la carte à puce (voir FIG. 8.3).

Une fois la carte insérée, l'utilisateur est invité à choisir sa destination parmi toutes celles connues par l'application (voir FIG. 8.4). Dans notre prototype, la liste des destinations possibles et la liste des cartes routières nécessaires sont connues statiquement par l'application.

Une fois la liste des villes connue de la carte à puce, celle-ci demande le chargement de la première carte routière (voir FIG. 8.5). Les cartes sont stockées localement, sous formes de fichiers ASCII.

Une fois la carte routière chargée dans la carte à puce, l'application calcule le chemin optimal pour traverser la ville (voir FIG. 8.6). La position initiale est contenue dans la carte, ainsi que la sortie de la ville à rejoindre.

Une fois le trajet calculé, l'application affiche les informations de guidage. Pour simuler le GPS, on fournit une fenêtre de contrôle permettant d'effectuer manuellement les pas d'exécution du programme (voir FIG. 8.7).

La fenêtre d'affichage des informations de guidage comporte trois zones distinctes (voir FIG. 8.8). La zone de gauche montre un fragment de la carte de la ville dans laquelle on se trouve. Cette carte n'est pas affichée entièrement pour améliorer la lisibilité, mais aussi pour protéger les données. En effet, puisque la station n'est pas considérée comme un environnement sûr, il est possible qu'un pirate mette en place un logiciel permettant par exemple de capturer les informations affichées à l'écran. Si on découpe la carte en

fractions suffisamment petites, il devient beaucoup moins facile de récupérer une carte car cela nécessiterait de parcourir toute la ville pour réassembler les morceaux de carte ensuite. Dans cette zone, c'est le point bleu représentant la voiture qui se déplace afin de fournir une vision globale de la position de la voiture. La ligne bleu clair indique le chemin parcouru et restant à parcourir. La zone du milieu présente un agrandissement des rues de la ville autour du véhicule. Cela permet de donner plus de détails pour faciliter le repérage, comme les noms des bâtiments par exemple, ainsi bien sûr que les informations de direction, qui apparaissent ici sous la forme d'une flèche verte indiquant la direction à suivre. Dans cette zone, le point reste immobile et c'est la carte qui évolue à chaque pas afin de permettre d'obtenir les informations de façon continue. La zone de droite présente quand à elle un quadrillage représentant le découpage de la carte de la ville en zones, et la zone courante peinte en rouge.

On suppose pour simplifier que les villes sont reliées par des autoroutes, sur lesquelles il n'est pas nécessaire de donner des informations de guidage. Ce passage sur l'autoroute est ici simulé par une simple fenêtre d'information (voir FIG. 8.9).

Une fois que l'on atteint la ville suivante, le même schéma est appliqué, à partir du chargement de la carte routière, jusqu'à ce qu'on arrive à la destination finale (voir FIG. 8.10).

8.6 Perspectives

On pense faire évoluer l'application de façon significative. En effet, les temps de calcul du chemin optimal nous paraissent trop importants pour pouvoir raisonnablement envisager de commercialiser l'application telle qu'elle est actuellement (même si notre but n'est pas de développer une application pour la vendre ensuite, on aimerait cependant montrer qu'il est possible de réaliser des applications commercialisables avec la carte à puce). De la même façon, la place mémoire disponible est une grande restriction sur la taille des cartes routières chargeables dans la carte. Une première amélioration possible serait de changer la représentation interne des cartes routières. En effet, en passant d'une structure de labyrinthe à celle d'un graphe (chaque noeud représentant une case visitable), on réduirait déjà la taille mémoire occupée puisque les cases non accessibles ne seraient pas représentées. De même, on envisage de déporter le calcul du chemin optimal de la carte vers la station. Cela peut paraître très radical puisqu'on a mis en évidence qu'un des intérêts principaux de la carte à puce était justement la sécurité apportée par le fait de pouvoir exécuter le code applicatif à l'intérieur de la carte. Exécuter l'algorithme de calcul du chemin optimal sur la station implique que l'on copie les cartes routières décryptées sur celle-ci et donc qu'on les rendent vulnérables au piratage. Cependant, il convient de réfléchir à ce qu'il est véritablement intéressant de protéger dans cette application. L'algorithme de calcul du chemin optimal est disponible sur Internet et n'a donc en soi aucune valeur commerciale. De même, les cartes routières nues (ie : juste le tracé des routes) peuvent facilement être obtenues par numérisation de cartes sur papier bon marché. La véritable valeur de l'application réside dans l'assemblage de tous ses composants c'est à dire dans le travail de programmation qui a du être effectué pour l'obtenir. On doit donc avant tout protéger l'application contre les duplications illicites. Pour cela, il suffit de placer le programme principal dans la carte à puce. Le reste du code (notamment toutes les méthodes qui nécessitent une grande puissance de calcul) peut parfaitement être exécuté sur la station puisqu'il serait assez difficile de reconstituer l'application même si on arrivait à récupérer les méthodes dans la station. Ces fonctions peuvent être stockées initialement dans la carte et déployées lorsque l'utilisateur insère sa carte dans la station. On voit donc l'intérêt de cette application de guidage routier qui nous aura non seulement servi à illustrer les points forts des cartes à microprocesseur

et à isoler les services systèmes nécessaires au développement d'applications réparties sur ce média, mais également à mieux appréhender le rôle exact des cartes à puce comme support d'applications réparties. Il paraît maintenant évident qu'il n'est pas réaliste de vouloir intégrer tout le code d'une application dans la carte. Les limitations matérielles sont et resteront vraisemblablement telles qu'on devra toujours avoir recours à un serveur pour exécuter des algorithmes coûteux en taille mémoire et en puissance de calcul. Les cartes à microprocesseur semblent donc être un support idéal pour l'exécution de programmes simples nécessitant un haut niveau de sécurité et une bonne disponibilité des données.



FIG. 8.3 – Insertion de la carte à puce

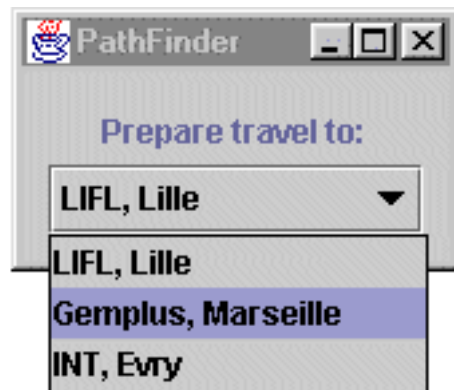


FIG. 8.4 – Choix de la destination

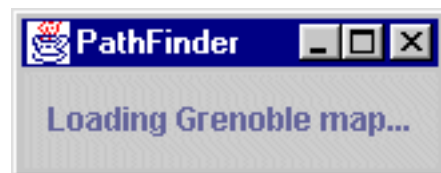


FIG. 8.5 – Chargement de la carte routière dans la carte à puce

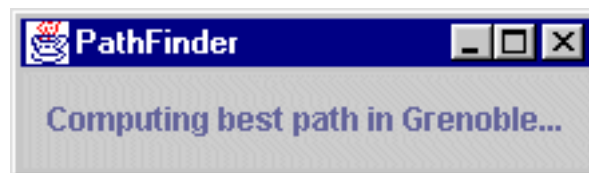


FIG. 8.6 – Calcul du chemin optimal dans la ville



FIG. 8.7 – Fenêtre de controle

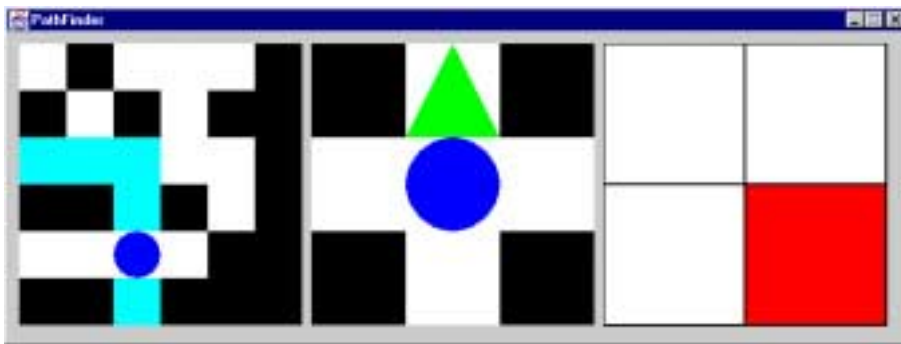


FIG. 8.8 – Affichage des informations de guidage

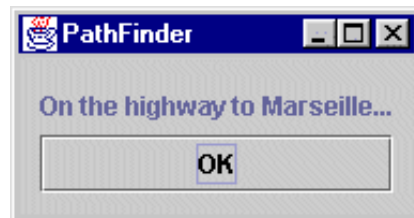


FIG. 8.9 – Information sur l'autoroute

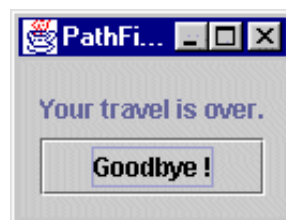


FIG. 8.10 – Fin du voyage

Quatrième partie

Conclusion

Chapitre 9

Synthèse

Les cartes à puce existent depuis plus de trente ans. Cependant, ce n'est que depuis quelques années que la technologie matérielle et logicielle est devenue suffisamment performante pour permettre l'utilisation de cartes à puce comme support d'applications évoluées. Et cela fait très peu de temps que l'on commence à réaliser leur véritable valeur ajoutée.

En effet, si l'on regarde avec un esprit critique les principales utilisations actuelles des cartes à puce, on se rend compte qu'elles n'y sont pas forcément nécessaires. Il n'est pas très contraignant d'avoir un peu de monnaie sur soi pour téléphoner ou prendre le bus, et il est toujours possible de payer ses achats par chèques. Les principales raisons de l'utilisation des cartes à puce sont actuellement d'ordre purement économique, puisqu'une carte de bus peut être par exemple un moyen de fidéliser le client en lui proposant d'acheter à l'avance ses trajets, éventuellement à un tarif préférentiel. On trouve cependant des applications où l'usage de la carte apporte un plus non négligeable, comme dans le cas des cartes de fidélité où la disponibilité des données est une propriété importante.

La principale avancée réalisée ces dernières années est l'arrivée de cartes à microprocesseur performantes et bon marché. Couplées à un environnement logiciel évolué comme Java Card par exemple, elles se révèlent être un support privilégié pour le développement d'applications nécessitant à la fois la disponibilité des données et un haut niveau de sécurité. Les services apportés par l'environnement Java Card facilitent la tâche du programmeur, bien qu'on puisse regretter qu'ils ne soient pas plus développés.

On pourrait souhaiter par exemple qu'un mécanisme d'invocation de méthodes distantes de la carte vers la station soit fourni, ce qui permettrait d'encarter le programme principal de l'application. De plus, certaines applications peuvent nécessiter des modes de communications autres que l'invocation de méthodes distantes, comme des schémas de communication par événements ou flots de données par exemple. De façon plus générale, on regrette fortement l'absence d'un chargeur de code dynamique dans la machine virtuelle Java Card. Ce mécanisme serait particulièrement utile pour permettre la mise à jour du code installé dans la carte. Enfin, un mécanisme de déchargement de la mémoire de la carte vers la mémoire de la station nous semble particulièrement intéressant compte tenu de la taille réduite de la mémoire disponible sur la carte.

Néanmoins, la principale leçon à tirer du prototypage de notre application de guidage routier est que malgré tous les progrès accomplis ces dernières années, les cartes à puce restent très limitées au niveau de la taille mémoire et de la puissance de calcul disponible. Les délais très importants nécessaires à l'algorithme de recherche du chemin optimal pour arriver au résultat nous font craindre que la technologie ne soit pas encore suffisamment avancée pour permettre le développement d'applications commerciales complexes exécutées

intégralement sur la carte.

Après l'enthousiasme suscité par l'arrivée de cartes à microprocesseur performantes et bon marché, accompagnée d'un environnement évolué tel que Java Card, il convient maintenant de réfléchir aux applications et services qu'il paraît réaliste d'implémenter sur la carte. Le rôle des cartes à puce n'est pas, et ne sera vraisemblablement jamais, de remplacer les serveurs performants utilisés actuellement dans les environnements répartis. La carte à microprocesseur se pose plutôt comme un serveur mobile de faible puissance mais hautement sécurisé, utile dans tous les cas où la disponibilité des données est un facteur déterminant de la réussite et de la qualité du service proposé.

Chapitre 10

Perspectives

A court terme, on espère pouvoir mettre en place le service de chargement dynamique de code dans la machine virtuelle Java Card. Cela nécessitera une modification de cette machine, dont il n'existe malheureusement aucune implémentation publique. Il pourrait donc être intéressant de développer un simulateur de machine virtuelle Java Card, afin de pouvoir tester les services implémentés dans un environnement le plus proche possible de la réalité (le développement d'une véritable machine virtuelle encartable serait beaucoup plus problématique à cause de la législation française qui régit très strictement la vente de cartes à puce vierges). L'implémentation du système de déchargement de la mémoire de la carte dans la mémoire de la station nous paraît également intéressant à implémenter. Nos collaborateurs de Gemplus travaillent actuellement à l'implémentation du service d'appel de méthodes distantes de la carte vers la station dont on a parlé précédemment et on espère pouvoir en disposer dans un proche avenir.

A plus long terme, il pourrait être intéressant de réfléchir aux parallèles que l'on peut trouver entre la carte à puce et d'autres supports d'exécution mobiles comme les téléphones portables ou les assistants digitaux personnels (*Personal Digital Assistants*). Ces supports proposent en effet des quantités de mémoire et des puissances de calcul bien supérieures à celles des cartes à puce, et sont de plus véritablement autonomes puisqu'ils intègrent un écran et un clavier, ainsi qu'une batterie. Ils ne disposent pas cependant de la même sécurité que les cartes à puce, et se révèlent donc inadaptés à des applications critiques comme les applications bancaires par exemple. Comme on l'a noté auparavant pour les téléphones mobiles, il est tout à fait possible d'appliquer les mêmes techniques de protections matérielles aux circuits intégrés dans ce type de support qu'au microprocesseur des cartes à puce. Le jour où les fabricants auront compris l'intérêt d'augmenter la sécurité de ces supports mobiles, il est à craindre que les cartes à puce ne disparaissent. Elles auront tout de même permis de comprendre que la sécurité et la disponibilité des données sont des éléments clés de la réussite et de la qualité des services mobiles.

Chapitre 11

Remerciements

L'auteur tient à remercier particulièrement Daniel Hagimont pour ses conseils tout au long de cette année de DEA.

Merci à Ricardo Caferra, à Andrzej Duda et à Brigitte Plateau de s'être intéressés à ce travail.

Merci à Roland Balter de m'avoir accueilli dans son laboratoire, ainsi qu'aux autres membres des projets Sirac et Dyade : Luc Bellissard, Céline Charles, Béatrice Claudio, Fabienne Déchamboux, André Freyssinet, Sacha Krakowiak, Gilles Kuntz, Serge Lacourte, Jacques Mossière, Michel Riveill et Xavier Rousset de Pina.

Tous mes remerciements également aux membres du projet CESURE pour leurs conseils et explications concernant leurs propres travaux : nos collaborateurs de Gemplus, ainsi que nos collègues du LIFL et de l'INT d'Evry.

Merci enfin à mes collègues étudiants et ingénieurs experts pour leur soutien permanent : Sara Bouchenak, Nicolas Bragato, Eric Bruneton, Olivier Charra, Sébastien Chassande-Barioz, Laurent Chauvirey, Emmanuel Cecchet, Noël De Palma, Fabrice Dutron, Olivier Fambon, David Féliot, Leila Ismail, Philippe Laumay, Vania Marangozova, Stéphane Martin, Cyril Perrin, Jean-Charles San-Sévérino, Aline Senart et Nicolas Tachker.

Cinquième partie

Bibliographie

Bibliographie

- [1] M.C. Pellegrini, O. Potonniée, R. Marvie, S. Jean, and M. Riveill. *CESURE : une plateforme d'applications adaptables et sécurisées pour usagers mobiles*, deuxième semestre 2000. <http://www.lifl.fr/RD2P>.
- [2] Information Systems Organization. *ISO7816 (part 1-3) asynchronous smartcards information*. <http://www.iso.org>.
- [3] Europay, Mastercard, and Visa. *EMV2000, Integrated Circuit Card Specification for Payment Systems, book 1*, 2000. <http://www.emvco.com/docs/book1.PDF>.
- [4] Europay, Mastercard, and Visa. *EMV2000, Integrated Circuit Card Specification for Payment Systems, book 2*, 2000. <http://www.emvco.com/docs/book2.PDF>.
- [5] Europay, Mastercard, and Visa. *EMV2000, Integrated Circuit Card Specification for Payment Systems, book 3*, 2000. <http://www.emvco.com/docs/book3.PDF>.
- [6] Europay, Mastercard, and Visa. *EMV2000, Integrated Circuit Card Specification for Payment Systems, book 4*, 2000. <http://www.emvco.com/docs/book4.PDF>.
- [7] IBM. *OpenCard Framework: General information web document*, october 1998. <http://www.opencard.org/docs/gim/ocfgim.pdf>.
- [8] IBM. *OpenCard Framework 1.2: Programmer's guide*, december 1999. <http://www.opencard.org/docs/pguide/PGuide.pdf>.
- [9] Sun Microsystems, Inc. *Java Card 2.1 Runtime Environment Specification*, february 1999.
- [10] Sun Microsystems, Inc. *Java Card 2.1 Virtual Machine Specification*, march 1999.
- [11] Sun Microsystems, Inc. *Java Card 2.1 Application Programming Interface*, march 1999.
- [12] Sun Microsystems, Inc. *Java Card 2.1 Reference Implementation*, may 1999. <http://java.sun.com/products/javacard/index.html>.
- [13] James Gosling, Bill Joy, and Guy Steele. *The Java Langage Specification*. Addison-Wesley, 1996.
- [14] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, second edition, 1999.
- [15] *Electronic ID card*, january 2000. <http://www.vaestorekisterikeskus.fi/fineid.htm>.
- [16] *Kerberos: The Network Authentication Protocol*. <http://web.mit.edu/kerberos/www/>.
- [17] Bastiaan Bakker. *Mutual Authentication with Smart Cards*. USENIX Workshop on Smartcard Technology, may 1999. <http://www.usenix.org>.
- [18] Distributed Systems Technology Centre. *JavaCards. the OpenCard Framework and Single Sign On*. <http://www.dstc.com>.
- [19] Refik Molva and Gene Tsudik. Authentication method with impersonal token cards. In *IEEE Symposium on Research in Security and Privacy*, 1993.
- [20] Victor Shoup and Avi Rubin. *Session key distribution using smart cards*.

- [21] Alain Macaire and David Carlier. *A Personal Naming and Directory Service for Mobile Internet Users*. USENIX Workshop on Smartcard Technology, may 1999. <http://www.usenix.org>.
- [22] Tage Stabell-Kulø, Ronny Arild, and Per Myrvang Harald. *Providing Authentication to Messages Signed with a Smart Card in Hostile Environments*. USENIX Workshop on Smartcard Technology, may 1999. <http://www.usenix.org>.
- [23] G. Grimaud and S. Jean. *Code mobile et carte à puce*. Ecole d'informatique des systèmes parallèles et répartis (ISYPAR'2000), Toulouse, février 2000. <http://www.lifl.fr/RD2P>.
- [24] C. Gransart and D. Simplot. *Communicating Mobile Objects*. Electronic Proc. Gemplus Developer Conference (GDC'2000), Montpellier, june 2000. <http://www.lifl.fr/RD2P>.
- [25] S. Jean, D. Donsez, and S. Lecomte. *Smart card intergration in distributed information systems: the interactive execution model*. IEEE International Symposium on Advanced Distributed Systems (ISADS'2000), Guadalajara Jalisco, Mexique, march 2000. <http://www.lifl.fr/RD2P>.
- [26] Didier Donsez, Gilles Grimaud, and Sylvain Lecomte. *Recoverable Persistent Memory for Smartcard*. Laboratoire d'Informatique Fondamentale de Lille, équipe Recherche et Développement Dossier Portable, septembre 1998. <http://www.lifl.fr/RD2P>.
- [27] Gilles Grimaud. *Introduction à une architecture Logicielle Nouvelle pour les Cartes à Microprocesseurs Ouvertes*. Laboratoire d'Informatique Fondamentale de Lille, équipe de Recherche et Développement Dossier Portable, juin 1999. <http://www.lifl.fr/RD2P>.
- [28] G. Grimaud and J.-J. Vanderwalle. *Procédé de Factorisation des Codes pour Cartes à Puces permettant des chemins de migration depuis plusieurs langages sources vers plusieurs plate-formes matérielles ou logicielles cibles*. Brevet N° 99 07239 déposé par la société Gemplus, juin 1999. <http://www.lifl.fr/RD2P>.
- [29] D. Donsez, S. Jean, and S. Lecomte. *Evolution du rôle de la carte à microprocesseur dans les systèmes d'information distribués*. Rapport technique 99-11, LIFL, Univ. Lille 1, juin 1999. <http://www.lifl.fr/RD2P>.
- [30] D. Carlier. *Représentation permanente, coordonnée par une carte à microprocesseur, d'un utilisateur mobile*. Thèse de Doctorat, Univ. Lille 1, janvier 1998. <http://www.lifl.fr/RD2P>.
- [31] Sébastien Jean and Gilles Grimaud. *GUM-E² : Une approche orientée carte à microprocesseur pour la gestion sécurisée de la mobilité de l'utilisateur dans les échanges électroniques*. Laboratoire d'Informatique Fondamentale de Lille, équipe de Recherche et Développement Dossier Portable, juin 1999. <http://www.lifl.fr/RD2P>.
- [32] V. Cordonnier, A. Watson, and Nemchenko. *Time as an aid to improving security in smart cards*. Proc. 7th Annual Working Conference on Information Security Management and Small Systems Security, Amsterdam, Pays-Bas, september 1999. <http://www.lifl.fr/RD2P>.
- [33] G. Grimaud and S Jean. *Interoperability of services in multiapplications smart cards, new approaches for security and flexibility*. Proceedings of the 1st Eurosmart security conference, Marseille, june 2000. <http://www.lifl.fr/RD2P>.
- [34] G. Grimaud, J.-L. Lanet, and J.-J. Vanderwalle. *FACADE : a typed intermediate language dedicated to smart cards*. Proc. Joint 7th European Software Engineering Conference (ESEC) and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), Berlin, Allemagne, september 1999. <http://www.lifl.fr/RD2P>.
- [35] Sylvain Lecomte and Didier Donsez. *Intégration d'un Gestionnaire de Transaction dans les cartes à microprocesseurs*. Laboratoire d'Informatique Fondamen-

- tale de Lille, équipe Recherche et Développement Dossier Portable, novembre 1998. <http://www.lifl.fr/RD2P>.
- [36] S. Lecomte. *COST-STIC: des cartes orientées services transactionnels et des systèmes transactionnels intégrant des cartes*. Thèse de Doctorat, Univ. Lille 1, novembre 1998. <http://www.lifl.fr/RD2P>.
- [37] S. Lecomte, G. Grimaud, and D. Donsez. *Implementation of transactional mechanisms for open smartcard*. Electronic Proc. Gemplus Developer Conference (GDC'99), Paris, june 1999. <http://www.lifl.fr/RD2P>.
- [38] D. Donsez and S. Jean. *Extension des capacités de traitement des cartes à puce bases de données*. Rapport technique 99-12, LIFL, Univ. Lille 1, décembre 1999. <http://www.lifl.fr/RD2P>.
- [39] D. Donsez, S. Jean, and S. Lecomte. *Utilisation des bases de données pour la flexibilité de services coopérants dans la carte à microprocesseur*. Actes du 18e congrès Inforsid (Inforsid 2000), Lyon, mai 2000. <http://www.lifl.fr/RD2P>.
- [40] Thomas Jensen. *Java Card, langage de programmation pour les cartes à puces*. Institut National de Recherche en Informatique et en Automatique, projet Lande, janvier 1999. <http://www.inria.fr/INedit/INedit-fra.html>.
- [41] Isabelle Attali. *A Formal Semantics and an Interactive Environment for Java*. INRIA, Sophia-Antipolis, OASIS project, january 1999. http://www.ercim.org/publication/Ercim_News/enw36/attali.html.
- [42] I. Attali and al. *A formal executable semantics for Java*. Proceedings of Formal Underpinnings of Java - OOPSLA'98 Workshop, Vancouver, october 1998. <http://www-sop.inria.fr/oasis/personnel/Isabelle.Attali/>.
- [43] T. Jensen, D. Le Métayer, and T. Thorn. *Verification of control flow based security policies*. Proceedings of 20th IEEE Security and Privacy Symposium, Oakland, Etats-Unis, 1999. <http://www.irisa.fr/lande/jensen/javacard.html>.
- [44] T. Jensen, D. Le Métayer, and T. Thorn. *Security and Dynamic Class Loading in Java: A Formalisation*. Proceedings of the 1998 IEEE International Conference on Computer Languages, may 1998. <http://www.irisa.fr/lande/jensen/javacard.html>.
- [45] A. Fau. *Analyse et éléments de formalisation de la sécurité Java*. Mémoire de DEA, juin 1999. <http://www.irisa.fr/lande/jensen/javacard.html>.
- [46] X. Leroy and F. Rouaix. *Security properties of typed applets*. 25th ACM conference on Principles of Programming Languages, january 1998. <http://www.irisa.fr/lande/jensen/javacard.html>.
- [47] Carine Courbis. *Simulation d'Applications Java Card*. Rapport de DEA, Unité de Formation et de Recherche en Informatique de l'Université Claude Bernard—Lyon 1, juillet 1998. <http://www-sop.inria.fr/oasis/personnel/Carine.Courbis/pubs/index.html>.
- [48] Tuomas Aura and Dieter Gollmann. *Software Licence Management with Smart Cards*. USENIX Workshop on Smartcard Technology, may 1999. <http://www.usenix.org>.
- [49] Ross Anderson and Markus Kuhn. *Low cost attacks on tamper resistant devices*. Cambridge Computer Laboratory and COAST Laboratory Purdue University. <http://www.cl.cam.ac.uk/ftp/users/rja14/tamper2.ps.gz>.
- [50] Ross J. Anderson. *Tamperproofing of Chip Card*. Cambridge Computer Laboratory, september 1997. http://www.infowar.com/class_2/class2_091197a.html-ssi.
- [51] Salvatore J. Stolfo, David W. Fan, Wenke Lee, and Andreas L. Prodromidis. *Credit card fraud detection using meta-learning: issues and initial results*.
- [52] Common Criteria for Information Technology Security Evaluation. *Protection Profile: Smartcard Integrated Circuit, version 2.0*, september 1998. <http://www.crsc.nist.gov/cc>.

- [53] Howard Gobioff, Sean Smith, J. D. Tygar, and Bennet Yee. *Smart cards in hostile environments*.
- [54] Armando Fox and Steven Gribble. *Security on the move : indirect authentication using Kerberos*.
- [55] Michael Montgomery and Ksheerabdh Krishna. *Secure Object Sharing in Java Card*. USENIX Workshop on Smartcard Techonology, may 1999. <http://www.usenix.org>.
- [56] Marvin Sirbu and J.D. Tygar. *NetBill: an Internet commerce system optimized for network delivered services*. Carnegie Mellon University, Pittsburgh, Pennsylvania, Etats-Unis, 1995. <http://www.ini.cmu.edu/netbill/pubs/CompCon.ps.Z>.
- [57] Naomaru Itoi, Peter Honeyman, and Jim Rees. *SCFS: A UNIX Filesystem for Smartcards*. USENIX Workshop on Smartcard Technology, may 1999. <http://www.usenix.org>.
- [58] Multos Consortium. *A guide to the Multos scheme*, 2000. <http://www.multos.com>.
- [59] Microsoft Corporation. *The Windows CE 3.0 smart card subsystem*, 2000. <http://www.microsoft.com>.
- [60] Matt Blaze. *High-bandwidth encryption with low-bandwidth smartcards*, december 1995.
- [61] OKI. *Oki Debuts Revolutionary New Personal Identification System at Meeting of Forensic Scientists*, august 1996. <http://www.oki.co.jp/OKI/Home/English/New/OKI-News/1996/z9630.html>.
- [62] Zhiqun Chen and Rinaldo Di Giorgio. *Understanding Java Card 2.0*. Java World, march 1998. <http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev.html>.
- [63] Zhiqun Chen. *How to write a Java Card applet: a developer's guide*. Java World, july 1999. <http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard.html>.
- [64] Sun Microsystems, Inc. *Java Card Technology Frequently Asked Questions*, july 1998. <http://java.sun.com/products/javacard/faq.html>.
- [65] Sun Microsystems, Inc. *Smart Card Overview*, september 1998. <http://java.sun.com/products/javacard/smartcards.html>.
- [66] Jean-Jacques Vanderwalle. *Construction d'applications avec la carte à puce*. Gemplus Research Lab, août 1999. <http://sirac.imag.fr/ecole/99/cours/index.html>.

Sixième partie

Annexes

Annexe A

Exemple : *ping* pour Java Card

On présente ici à titre d'exemple le code d'une version pour Java Card du célèbre programme *ping*. La partie cliente du programme envoie des paquets de données aléatoires qui sont renvoyés tel quel par la partie serveur sur la carte à puce. Ce petit programme nous a servi pour avoir une idée de la vitesse de transmission des données entre le terminal et la carte.

La partie serveur

L'interface ServerItf

```
/**
 * Interface du serveur Ping
 *
 * @author Christophe RIPPERT
 */

package server;

public interface ServerItf {

    // La classe de l'instruction supportée
    final public static byte CLA = (byte) 0xB0;

    // L'instruction supportée
    final public static byte INS = (byte) 0x00;

    // La taille des données transmises
    final public static byte PACKET_DATA_SIZE = (byte) 32;

}
```

La classe Server

```
/**
 * Serveur Ping
 *
 * @author Christophe RIPPERT
 */

package server;
```

```

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;

final public class Server
    extends Applet {

    // Constructeur du serveur
    private Server() {
        // Enregistrement de l'application auprès du JCRE
        this.register();
    }

    // Fonction d'installation de l'application
    final public static void install(byte[] params,
                                    short offset,
                                    byte length) {
        new Server();
    }

    // Fonction de traitement
    final public void process(APDU apdu)
        throws ISOException {
        // Réception de l'en-tête de l'APDU
        byte[] buffer = apdu.getBuffer();
        if ((buffer[ISO7816.OFFSET_CLA] == ISO7816.CLA_ISO7816) &&
            (buffer[ISO7816.OFFSET_INS] == ISO7816.INS_SELECT)) {
            return;
        } else if (buffer[ISO7816.OFFSET_CLA] != ServerItf.CLA) {
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        } else if (buffer[ISO7816.OFFSET_INS] == ServerItf.INS) {
            // Réception des données
            apdu.setIncomingAndReceive();
            // Construction de l'APDU réponse
            Util.arrayCopy(buffer, ISO7816.OFFSET_CDATA, buffer, (short) 0,
                ServerItf.PACKET_DATA_SIZE);
            // Emission de l'APDU réponse
            apdu.setOutgoingAndSend((short) 0, ServerItf.PACKET_DATA_SIZE);
        } else {
            ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }
}

```

La partie cliente

La classe Main

```

/**
 * Client Ping
 *
 * @author Christophe RIPPERT
 */

package client;

```



```

        (byte) 0x00,
        (byte) 0x00,
        (byte) 0x00,
        (byte) 0x02));
    }

// Fonction réalisant l'échange de données
final private static void ping()
    throws Exception {
    // L'APDU commande envoyé
    byte[] apdu = new byte[Main.APDU_COMM_HEAD_SIZE +
        ServerItf.PACKET_DATA_SIZE +
        Main.APDU_COMM_TAIL_SIZE];

    for (;;) {
        // Les données sont aléatoires
        Arrays.fill(apdu, (byte) (new Random()).nextInt(256));
        // Construction de l'en-tête :
        // la classe d'instruction
        apdu[ISO7816.OFFSET_CLA] = ServerItf.CLA;
        // l'instruction
        apdu[ISO7816.OFFSET_INS] = ServerItf.INS;
        // les paramètres
        apdu[ISO7816.OFFSET_P1] = 0;
        apdu[ISO7816.OFFSET_P2] = 0;
        // la taille de la zone de données
        apdu[ISO7816.OFFSET_LC] = ServerItf.PACKET_DATA_SIZE;
        // la taille de la zone de données attendue pour l'APDU réponse
        apdu[apdu.length - 1] = ServerItf.PACKET_DATA_SIZE;
        CommandAPDU comm = new CommandAPDU(apdu);
        System.out.print("Sent " +
            (comm.getLength() -
                Main.APDU_COMM_HEAD_SIZE -
                Main.APDU_COMM_TAIL_SIZE) +
            " data bytes...");
        long timeSend = System.currentTimeMillis();
        // Envoie de l'APDU commande et réception de l'APDU réponse
        ResponseAPDU resp = serv.sendAPDU(comm);
        long timeRecv = System.currentTimeMillis();
        System.out.print("received " +
            (resp.getLength() - APDU_RESP_TAIL_SIZE) +
            " data bytes in " +
            (timeRecv - timeSend) +
            " ms, checking...");
        // Vérification de l'intégrité des données
        if (Main.equals(apdu, Main.APDU_COMM_HEAD_SIZE,
            ServerItf.PACKET_DATA_SIZE,
            resp.getBuffer(), 0,
            resp.getLength() - Main.APDU_RESP_TAIL_SIZE)) {
            System.out.println("ok");
        } else {
            System.out.println("data is corrupted !");
        }
    }
}

// Fonction d'arrêt du lecteur et de la carte
final private static void end()
    throws Exception {
    SmartCard.shutdown();
}

// Fonction de comparaison de deux tableaux d'octets

```

```

final private static boolean equals(byte[] a1,
                                   int o1,
                                   int s1,
                                   byte[] a2,
                                   int o2,
                                   int s2) {

    if (s1 != s2) {
        return false;
    }
    int i, j;
    for (i = o1, j = o2; (i < o1 + s1) && (a1[i] == a2[j]); i ++, j ++);
    return (i == o1 + s1);
}

// Programme principal
final public static void main(String[] args)
    throws Exception {
    Main.init();
    Main.ping();
    Main.end();
}
}

```

Exécution

```

Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 171 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 201 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 180 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 191 ms, checking...ok
Sent 32 data bytes...received 32 data bytes in 170 ms, checking...ok
...

```


Annexe B

Quelques liens utiles

Les membres du projet CESURE

Sirac

- Site Internet : <http://sirac.inrialpes.fr>
- Contact : Roland Balter
- Email: Roland.Balter@imag.fr

Gemplus

- Site Internet : <http://www.gemplus.fr>
- Contact : Pierre Paradinas
- Email: Pierre.Paradinas@gemplus.com

Institut National des Télécommunications d'Evry

- Site Internet : <http://www.int-evry.fr>
- Contact : Guy Bernard
- Email: Guy.Bernard@int-evry.fr

Laboratoire d'Informatique Fondamentale de Lille

- Site Internet : <http://www.lifl.fr>
- Contact : Jean-Marc Geib
- Email: geib@lifl.fr

D'autres projets de recherches

Lande

- Site Internet : <http://www.irisa.fr/lande/>

Oasis

- Site Internet : <http://www-sop.inria.fr/oasis/>

Quelques industriels

Bull

- Site Internet : <http://www.cp8.bull.com>

De La Rue

- Site Internet : <http://www.delarue.com/card>

Groupement des cartes bancaires

- Site Internet : <http://www.gie-cartes-bancaires.fr/>

IBM

- Site Internet : www.chipcard.ibm.com

Microsoft

- Site Internet : www.microsoft.com

Motorola Corporation

- Site Internet : <http://www.mot.com>

Multos Consortium

- Site Internet : <http://www.multos.com>

Schlumberger

- Site Internet : <http://www.schlumberger.com/>

Sun Microsystems

- Site Internet : <http://java.sun.com/products/javacard>

Visa

- Site Internet : <http://www.visa.com/openplatform>