Contrôle des ressources dans les cartes à microprocesseur

– papier d'intention –

Antoine Galland*,† Damien Deville‡ Gilles Grimaud* Bertil Folliot†

* Gemplus Software Research *prenom.nom*@gemplus.com
† Laboratoire Informatique de Paris VI (LIP6, UPMC/CNRS) *prenom.nom*@lip6.fr
‡ Laboratoire Informatique Fondamentale de Lille (LIFL, CNRS) *prenom.nom*@lifl.fr

Août 2001

Résumé

Les cartes à puces ont évolué vers le concept de cartes ouvertes. Ce sont des cartes où l'on peut charger des applications après qu'elles aient été émises (distribuées à leur porteur). De nombreux travaux ont consisté à sécuriser le code mobile qui pouvait être chargé dans la carte [18, 26] afin d'assurer l'intégrité et la confidentialité des données. Cependant, établir des garanties plus larges comme le contrôle des ressources ou le temps réel reste une question ouverte. Nous étudions ici les concepts de qualité de service (disponibilité, déni de service) liés aux systèmes extensibles dans le domaine de la carte à puces.

Mots-clé : contrôle des ressources, carte à puces, système d'exploitation, extensibilité, temps réel.

1 Problématique

On oppose souvent flexibilité des systèmes d'exploitation et sécurité, chacun de ces domaines ayant des propriétés orthogonales. Certains compromis ont cependant été trouvés permettant ainsi de charger du code mobile système ou applicatif tout en garantissant la stabilité de ces derniers. Néanmoins, l'orientation prise par certains systèmes flexibles ne permet pas aujourd'hui de garantir des propriétés de sécurités plus larges liées à la qualité de service ou au contrôle des ressources (mémoire, processeur, communication). Dans le contexte des cartes à puces, la sécurité est un facteur primordial. Lutter contre les attaques en déni de service ou en disponibilité sur le chargement de code mobile (extensions système, nouvelles applications) tout en ayant un système d'exploitation flexible et sûr reste une problématique non résolue.

Notre approche a donc pour but de concilier les domaines de recherche relatifs aux systèmes (noyau minimaliste, système extensible et adaptable, machine virtuelle), à la sécurité (confidentialité et intégrité des données) et à la qualité de service (disponibilité, déni de service). De nombreuses solutions ont été proposées dans chacun des domaines mais rares sont celles qui essayent d'allier ces différents mondes. Nous nous proposons de décrire quelques systèmes flexibles et d'analyser pourquoi ces derniers répondent partiellement à notre problème.

2 État de l'art

Notre état de l'art se décompose en une première étude sur les systèmes d'exploitation au sens général, puis une deuxième partie sur les systèmes d'exploitation pour cartes à microprocesseur.

2.1 Systèmes d'exploitation

Notre étude sur les systèmes d'exploitation regroupe trois domaines de recherche : sécurité, système flexible et qualité de service. Nous présentons quelques systèmes et techniques qui appartiennent à ces axes de recherche.

Sécurité : Dans le contexte de systèmes multi-tâches, le minimum requis pour construire un système qui réponde à notre problématique est d'assurer un mécanisme de protection mémoire. L'objectif de ce mécanisme est de garantir qu'un programme agit uniquement sur la zone mémoire qui lui a été allouée. Il s'agit d'isoler les programmes les uns par rapport aux autres. Ce confinement, aussi appelé *sandboxing*, est assuré via différentes techniques :

La première technique consiste à assurer un contrôle par adressage soit de manière :
- dynamique : grâce à un gestionnaire de mémoire matériel : *Memory Management Unit* (MMU) comme dans les systèmes classiques Unix, Linux ou Windows ;
- statique : procédé de transformation de code : *Software Fault Isolation* (SFI) [43].

La deuxième technique se fonde sur la théorie des types avec une approche orientée langage [23] : au niveau du bytecode (vérifieur Java [27]), d'assembleur typé (avec les travaux autour de TAL : *Typed Assembly Language* [33]), ou de code intermédiaire commun typé avec le langage FAÇADE [18]. Il faut noter que TAL et FAÇADE s'inspirent et réifient les travaux plus généraux de Necula et Lee sur le code comprenant sa preuve : *Proof-Carrying Code* (PCC) [34].

Si ces techniques servent de manière équivalente l'intégrité et la confidentialité des données, elles n'ont pas

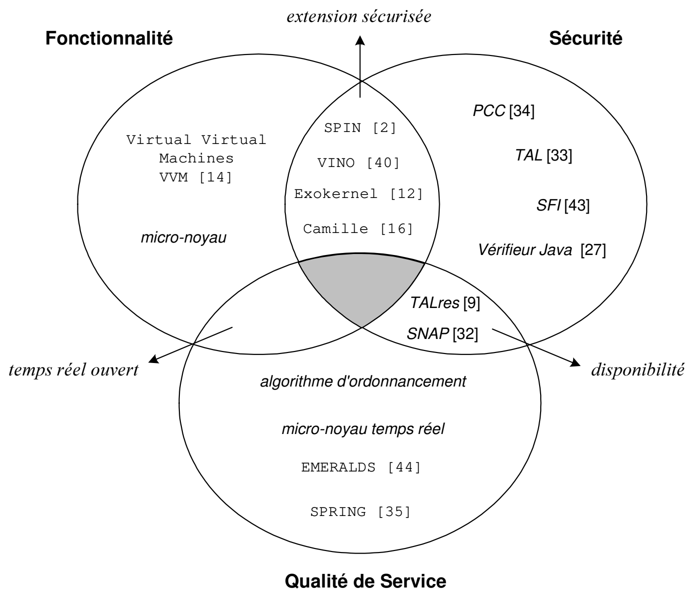

Figure 1 – *Classification possible des techniques et systèmes existants*

le même surcoût en terme de temps au niveau de leur utilisation à la compilation (approche statique), au chargement ou à l'exécution (approche dynamique). Une étude comparative de ces techniques revient à confronter l'amortissement du coût de départ [1] de chacunes d'elles pour analyser la plus pertinente liée à un contexte donné.

L'autre point qui permet de les distinguer est la taille de leur base de confiance : *Trusted Computing Base* (TCB), car plus celle-ci est petite plus on peut borner et vérifier formellement l'exactitude de sa politique de sécurité [24]. Même si ce principe de base est connu depuis plus de vingt cinq ans [37], force est de constater qu'il est peu respecté [38].

Système flexible : Si les travaux sur la flexibilité des systèmes d'exploitation sont multiples et ont montré de bons résultats comme les μ-noyaux, les exo-noyaux, les systèmes extensibles, peu de systèmes ont essayé de mettre la sécurité comme point central. Parmi ceux qui se sont penchés sur le problème, on peut citer :

– *SPIN* [2] qui intègre un système de capacités [19] au-dessus d'un langage sûr : Modula-3 [5] ;
– *VINO* [40] qui compose un système transactionnel [39] au-dessus de techniques SFI intégrées dans un compilateur [41] ;
– Les *exokernels* [12] qui utilisent un système de capacités hiérarchiques [30] au plus près du matériel, même si d'autres techniques peuvent leur être appliquées [13] ;
– *Camille* [16] qui intègre un vérifieur de type au niveau le plus bas [18] (via une technique de PCC) et lui superpose un système à base de capacités au-dessus.

Tous ont été développés en pensant sécurité et flexibilité conjointement. De manière générale, ils permettent d'étendre leurs fonctionalités (extension du système) tout en garantissant l'intégrité et la stabilité lors de l'installation d'une extension.

En ce qui concerne la qualité de service, celle-ci reste un point négligé ou volontairement écarté. Le partage entre les ressources est fourni par l'ordonnanceur qui essaye d'assurer de manière dynamique une certaine équité entre les ressources mais ne permet pas de répondre à des contraintes plus fortes : disponibilité, déni de service ou temps réel.

Qualité de service : Il s'agit d'une problématique classique des systèmes temps réel et des applications réseaux (applications audio/vidéo, réseaux actifs). L'objectif est d'offrir des garanties sur la consommation des ressources (processeur, mémoire, communication) ou le comportement des tâches (prédiction temporelle). Différents moyens sont employés pour assurer ces garanties.

Dans le premier cas il s'agit de contrôler les ressources et de borner leurs utilisation, on peut mentionner :

– le système de type *TALres* [9], basé sur les types dépendants, qui a été incorporé dans TAL [33] ;
– le langage de bytecode SNAP [32] pour réseau actif qui contraint le langage afin d'en extraire des propriétés (consommation des ressources linéaire en fonction de la taille du code).

1. *startup cost amortization* introduit par Necula et Lee dans [34].

Type de Mémoire	*Point mémoire*	*Capacité*	*Délai en écriture*	*Grain*
ROM	référence	32 → 128 ko	lecture seulement	1 octet
FlashRAM	×2 − 3	16 → 64 ko	$2,5ms$	64 octets
EEPROM	×4	4 → 64 ko	$4ms$	1 → 64 octets
RAM	×20	128 → 4096 octets	$\leq 0,2\mu s$	1 octet

Tableau 1 – *Caractéristiques moyennes des mémoires carte*

Dans le second cas il s'agit d'ordonnancer les tâches aux mieux (analyse d'ordonnancement) et de pourvoir prédire le temps d'exécution d'un programme (temps d'exécution moyen ou au pire cas). On peut signaler :

- les multiples travaux sur les algorithmes d'ordonnancement fondées sur l'ordonnancement de tâches a priori (algorithme *Rate Monotonic* [28] ou *Earliest Deadline First* pour ne citer qu'eux) ;
- les recherches sur l'analyse de code source [8] ne comportant ni boucles non bornées, ni fonctions récursives, ni pointeurs sur fonctions, ni sémaphores afin d'avoir une estimation du temps d'exécution au pire cas;
- les micro-noyaux temps réel comme *Spring* [35] qui fournissent un support d'exécution prévisible pour construire une architecture temps réel.

Bien que certains systèmes temps réel aient été conçus pour de petites plate-formes embarquées comme *EMERALDS* [44], ils n'en sont pas pour autant des systèmes suffisamment flexibles. Allier qualité de service (temps réel) et système ouvert reste un domaine de recherche à explorer [42].

Conclusion : La figure 1 synthétise notre état de l'art suivant trois ensembles : Fonctionnalité, Sécurité et Qualité de Service. Elle n'a pas la prétention de définir une recherche exhaustive dans ces trois domaines mais plutôt d'identifier leurs interactions. Si flexibilité et sécurité des systèmes ont été étudiées et composées avec succès ces dernières années, la contrainte de qualité de service liée aux ressources n'a été que peu intégrée aux deux premiers. Un compromis acceptable reste encore à trouver pour ces trois domaines notamment dans les systèmes embarqués de l'informatique omniprésente [2].

2.2 Systèmes d'exploitation pour cartes à microprocesseur

Nous utilisons aujourd'hui tous des cartes à microprocesseur aussi communément appelées cartes à puce, parfois sans même le savoir. La carte est souvent utilisée comme représentante de son porteur. Elle assure le lien entre l'individu et les services informatiques devenus omniprésents. La carte SIM [3] dans les téléphones portables (GSM [4]), est non pas associée à un téléphone mais à un individu. Elle configure le téléphone portable, assure la sécurité entre le réseau sans fil et l'individu, mais surtout elle héberge l'abonnement et les services optionnels auxquels l'utilisateur a souscrit grâce aux applets SIM-ToolKit [1] chargées sur le téléphone.

2.2.1 Une architecture matérielle minimaliste

La carte à puce se distingue des autres matériels informatiques par son architecture minimaliste. Elle possède un processeur, de la mémoire de travail, un port de communication et un support de données persistantes qui lui sont propres. Le tout devant tenir sur une surface de silicium monolithique de $27mm^2$. Tous ces composants sont physiquement sécurisés pour résister aux attaques matérielles qui permettent d'extraire les données stockées dans la carte (on parle de matériel *tamper resistant* [22]).

Nous allons présenter plus en détails les contraintes et caractéristiques matérielles de la carte. Elles ont en effet un impact important sur les solutions techniques qui peuvent être appliquées à la carte.

Les mémoires : La carte dispose de différents types de mémoire :

- la RAM (Random Access Memory) ;
- la ROM (Read Only Memory) ;
- l'EEPROM (Electric Erasable Programmable Read Only Memory) ;
- la FlashRAM (aussi appelée Flash) est une mémoire EEPROM modifiée dont la granularité est plus importante mais avec un temps d'écriture plus rapide.

La RAM est dite mémoire volatile. La Flash et L'EEPROM sont dites mémoires non volatiles ou persistantes. Ces dernières sont souvent représentées comme le disque dur de la carte. Cela est partiellement vrai. Leur temps d'accès en lecture est aussi rapide que la RAM et elles servent de mémoire de travail en cas de nécessité. En revanche leur temps d'écriture pénalise grandement le programmeur carte (de 1000 à 10000 fois plus lent). Le tableau 1 résume les caractéristiques des différentes mémoires. L'utilisation de l'EEPROM soulève de plus un problème de fiabilité de fonctionnement. Ses cellules mémoires supportent un nombre limité d'écritures garanties (stress mémoire) : entre 10^4 et 10^6. Ceci limite

2. Ubiquitous or Pervasive Computing.
3. Subscriber Identity Module, module d'identification de l'abonné sur les réseaux GSM.
4. Global System for Mobile Communications.

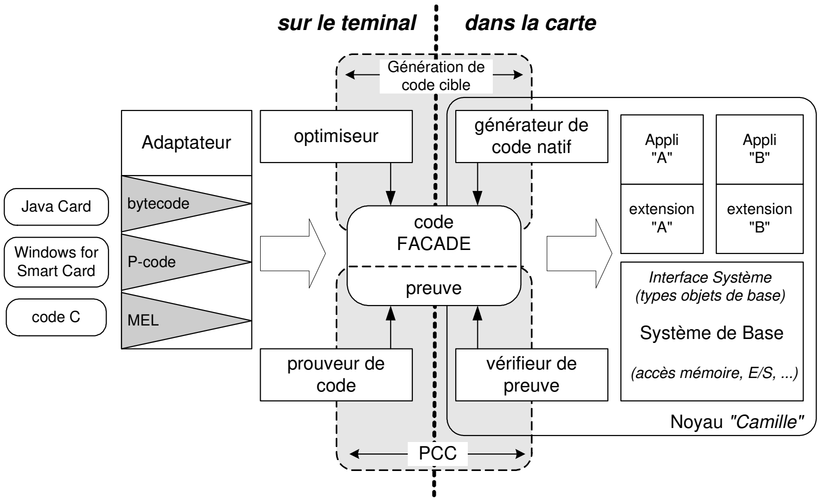

Figure 2 – *Architecture Camille*

l'utilisation de la mémoire persistante comme mémoire de travail.

Si la quantité d'EEPROM ou Flash est acceptable, la quantité de RAM implique une refonte de l'algorithmique de l'informatique traditionnelle. Cette dernière contrainte entraîne une programmation spécifique et contribue à distinguer la carte des autres systèmes embarqués.

Microprocesseurs dédiés : La plus part des microprocesseurs sont des architectures RISC 8 bits (parfois 16 ou 32) qui travaillent à des fréquences d'horloge de 4,77Mhz à 40Mhz pour les plus performants. L'écart entre les microprocesseurs encartés et ceux des stations de travail n'est donc pas comparable au gouffre qui sépare les mémoires carte des mémoires traditionnelles. Néanmoins, les progrès d'intégration et d'optimisation des circuits réalisés par les fondeurs sont plus souvent utilisés par les industriels pour diviser les coûts de production plutôt que d'augmenter la puissance des cartes à puce.

De manière générale, l'ensemble des contraintes carte (normes, sécurité, mémoires, puissance, coût de fabrication) ont une influence considérable sur les programmes encartés. Beaucoup d'applications supportent mal le passage à l'échelle lorsque l'on souhaite les encarter.

2.2.2 Les systèmes industriels

Il existe différents systèmes d'exploitation industriels pour cartes à microprocesseur, tels que *Java Card* [6], *Smart Card for Windows* [31], et *MULTOS* [29]. Le problème est que l'interopérabilité entre ces systèmes est inexistante. De ce constat est né l'architecture *Camille* [15, 16]. Nous allons résumer les solutions retenues dans *Camille* (à la fois au niveau système et sécurité) et montrer comment faire évoluer cette architecture afin d'y intégrer des propriétés de qualité de service.

2.2.3 Les apports de *Camille*

L'architecture *Camille* a été conçue pour répondre à la problématique des systèmes d'exploitation ouverts pour carte à microprocesseur. Il s'agit de cartes pouvant accepter du code après émission (au niveau du porteur de la carte pendant le cycle de vie de celle-ci). Cette architecture propose des éléments de réponse aux différents problèmes énumérés ci-dessous.

Hétérogénéité : Le support de l'hétérogénéité des machines virtuelles est une problématique récente [14]. La complexité de cette entreprise est telle qu'il est difficile de l'envisager dans la carte. C'est pourquoi le traitement a été distribué entre le terminal et la carte (cf. figure 2). La gestion de l'hétérogénéité des abstractions du matériel, et donc des machines virtuelles étant reportée en dehors de la carte, on parle alors « d'architecture exo-virtuelle ». Un code intermédiaire typé nommé FAÇADE [18] sert de jonction entre le terminal et la carte.

Portabilité : Le langage FAÇADE définit une abstraction très proche du matériel (page mémoire, valeur numérique, bloc de code exécutable ...) via un modèle de classe par héritage simple. Il n'y a pas de notion de tableau, ni de fichier. Cette approche assure une portabilité sur des supports matériels variés ainsi qu'une extensibilité accrue. Cette vision est dans ce sens proche des travaux sur les *exokernels* [11].

Sécurité : Les problèmes de sécurité, en terme de confidentialité et d'intégrité, sont résolus par l'intermédiaire d'un mécanisme d'inférence de type [18] inspiré des travaux de Lee et Necula sur le Proof-Carrying Code [34]. L'algorithme d'inférence dans la carte a été modélisé et prouvé en utilisant la méthode B [36] afin de garantir sa fiabilité.

Extensibilité : Les applications sont libres de redéfinir leurs abstractions suivant leurs besoins et ce dans un souci de performance. Cependant, l'efficacité est le premier verrou à l'extensibilité des systèmes d'exploitations. La génération de code natif est une réponse à ce problème même dans le contexte des cartes à microprocesseur. Le processus de compilation final [5] est distribué en deux parties afin que la génération de code natif dans la carte soit faite en une passe. La majeur partie des optimisations sur le code intermédiaire est faite sur le terminal.

Conclusion : Le code intermédiaire FAÇADE permet donc d'assurer la portabilité, l'extensibilité du système sans perte de performance, l'hétérogénéité des machines virtuelles ainsi que la sécurité (intégrité et confidentialité des données). Le tout contenu dans un micro-noyau de 17 ko. Si cette architecture offre une bonne solution aux cartes ouvertes, elle ne répond pas aux cartes multi-tâches (une même carte possédant plusieurs applications s'exécutant en même temps). Nous devons donc introduire les concepts multi-tâches dans cette architecture. Dans ce contexte, la gestion des ressources reste un point critique non résolu. Que ce soit sur le language FAÇADE ou bien sur le bytecode Java ou Java Card, l'inférence de type actuellement pratiquée permet seulement de vérifier des propriétés de sécurité (intégrité, condidentialité) ; établir des propriétés plus larges comme la consommation de ressources reste une question ouverte [6].

3 Perspective : *Camille* RC & RT [7]

Nous présentons dans cette section les évolutions que nous souhaitons apporter à *Camille* afin d'y intégrer les propriétés de contrôle mémoire et temps réel en tenant compte du multi-tâches.

3.1 *Camille* RC : Resource Control

La mémoire est une ressource rare dans les cartes à microprocesseur : environ 2/4 ko de mémoire volatile (RAM), 64/128 ko de ROM et 32/64 ko de mémoire non volatile (mémoire Flash ou EEPROM) pour les cartes les plus performantes du marché.

Aujourd'hui, le contrôle des ressources notamment mémoire est assuré lors de l'exécution : cette technique de surveillance est appelée *monitoring*. Dans certains systèmes, les applications ont une zone mémoire réservée dans le tas (*heap*) au démarrage. On contrôle à chaque allocation que l'on ne dépasse pas cette limite. Cette approche a pour avantage d'être facile à implanter et de ne pas imposer de contraintes sur le code ou la manière de programmer. Elle a pour inconvénient d'entraîner un surcoût lors de l'exécution ainsi que du code supplémentaire dans la carte nécessaire pour assurer le contrôle. Cette approche est *sous optimale* car il n'y a pas de renégociation de la mémoire réservée pour une application même s'il existe de la mémoire libre en dehors de la zone délimitée.

En ce qui concerne la gestion mémoire en cours d'exécution (allocation/libération), la qualité de service est difficile à garantir. Les applications cartes actuelles allouent la totalité de leur mémoire avant exécution ce qui offre un comportement prédictible : si l'allocation réussie à l'installation alors il n'y aura pas d'échec mémoire en cours d'exécution. Cette solution est fortement conseillée en Java Card 2.1 [6] (au niveau de la méthode `install()`).

À coté, les approches statiques liées au langage entraînent un surcoût à la compilation (ou au chargement) mais pas à l'exécution. Il n'y a pas de code lié au contrôle dans la carte. Elles imposent généralement des contraintes sur le langage et limitent par conséquent les applications.

Dans les deux cas il n'y a pas de solution globale. Une composition entre ces aspects est nécessaire. La solution que nous proposons se compose de deux parties :

- d'une part, une *validation statique* qui en analysant le code extrait des informations concernant la consommation mémoire (durée de vie des objets, taille consommée dans les différentes mémoires) ;
- d'autre part, un *pilotage dynamique* de l'ordonnanceur grâce aux informations de consommation extraites de l'analyse statique.

Dans les systèmes temps réel, les contraintes (temporelles, les priorités des tâches, leurs relations de précédence, leurs périodes) servent à piloter l'ordonnanceur. La prise en compte des contraintes de ressources par les algorithmes d'ordonnancement constitue l'un des problèmes les plus difficiles à résoudre. Il y a généralement réservation de toutes les ressources nécessaires avant exécution de la tâche, ce qui conduit à une sous utilisation des ressources systèmes. En effet, une tâche peut bloquer plusieurs ressources, éventuellement demandées par d'autres tâches en attente, alors qu'elle ne va pas les utiliser nécessairement simultanément pendant son exécution.

Nous pensons cependant que la prise en compte de la consommation mémoire (prédiction à court ou moyen terme) peut améliorer la politique d'ordonnancement et accroître les performances. Si on sait qu'une tâche A va libérer de la mémoire à un instant t, on peut retarder

5. *génération de code cible* sur le schéma de la figure 2.
6. Parmi les conclusions des travaux de Gilles Grimaud [16] et Xavier Leroy [25].
7. Resource Control & Real Time.

(suspendre) une tâche B jusqu'à la libération de cette mémoire afin que B puisse correctement s'exécuter.

Une meilleure connaissance des temps d'exécution et consommation mémoire permet donc un meilleur entrelacement des tâches. On limite ainsi la sous utilisation des ressources systèmes.

Dans cette optique, l'utilisation d'analyses statiques de flots de données comme *l'analyse d'échappement* [8] [3, 7] offrent des clefs (durée de vie des variables, allocation en pile plutôt qu'en tas) à la gestion mémoire et à notre *validation statique.*

Notre deuxième réponse consiste à enrichir le système de type du langage intermédiaire FAÇADE avec un sous ensemble des types dépendants comme utilisé dans les travaux sur *TALres* [9]. Les types dépendants sont des types qui dépendent de la valeur des expressions. Nous projetons de les utiliser pour compléter notre analyse statique dans les cas de variables dépendantes qui seront résolues à l'exécution. Dans les deux cas, les informations issues de la validation statique vont servir à piloter l'ordonnanceur.

Conclusion Nous souhaitons développer le contrôle et la gestion mémoire afin de lutter contre les attaques en déni de service. Nous proposons une solution de *validation statique* afin de mieux *piloter l'ordonnanceur.* Ce développement s'inscrit dans la philosophie *Camille* : chargement d'extensions système sécurisées. Cette analyse de code statique sera également l'occasion d'obtenir des prédictions sur la consommation CPU (nombre de cycle) donnant ainsi la possibilité de faire un système temps réel avec ces informations. Nous expliquons cette perspective dans notre seconde partie.

3.2 *Camille* RT : Real Time

L'autre objectif que nous nous proposons d'atteindre est la maîtrise du temps d'exécution des programmes encartés. En effet, dans l'architecture *Camille*, comme dans la culture carte en général, l'activité du microprocesseur n'est pas partagée entre plusieurs applications. Le modèle d'exécution tel qu'il a été normalisé dans ISO 7816-3 & 4 [21, 20] ne prévoit l'activation d'une application qu'après réception d'une commande provenant de l'extérieur, le microprocesseur étant obligatoirement libéré lorsque celle-ci a pu fournir sa réponse. De plus, le protocole de communication se présentant sous la forme d'un échange question/réponse entre le terminal et la carte, aussi cette dernière ne peut être amenée à recevoir et à traiter plusieurs requêtes simultanément. Ce modèle d'exécution/communication intègre, dans le cadre des systèmes d'exploitation pour téléphonie mobile, une prise en charge de la consommation énergétique. En effet, après l'émission du dernier bit de réponse, la carte est tenue d'activer un mode économie d'énergie (*idle*) où le microprocesseur est désactivé et ne sera réactivé qu'après une remise sous tension ou la réception d'un bit entrant émis par le terminal.

Cependant, ce modèle atteint ses limites, en particulier dans le contexte de la téléphonie mobile où, d'une part la carte peut être amenée à initier des communications avec son terminal, d'autre part, elle peut être amenée à recevoir une seconde requête alors que la première n'a pu encore être traitée. Pour apporter une réponse satisfaisante à ces deux exigences du GSM, l'industrie de la carte a su proposer un modèle nommé *proactif* construit à partir des normes ISO. C'est le modèle implanté dans les cartes SIM et dans les applets SIM-ToolKit [1].

Ce modèle soulève de nouveaux problèmes dans le contexte des cartes ouvertes où des applications étant mutuellement méfiantes peuvent être chargées sur une même plate-forme carte. Dans ce cas, des attaques en déni de service sont possibles, en particulier une application peut monopoliser le microprocesseur de telle sorte qu'il soit impossible à une autre application de fournir la réponse à une question en temps et en heure. L'application gérant le paiement de la communication téléphonique doit être par exemple capable de fournir périodiquement des clefs de session, ainsi qu'un décompte des unités pour le terminal et le réseau sans fils. Il ne doit pas être possible d'attaquer cette application en déni de service, faute de quoi l'utilisateur se verra privé de l'usage de son téléphone.

Cet état des lieux montre la pertinence d'une approche système temps réel dans la carte à microprocesseur et plus particulièrement dans les cartes à microprocesseur pourvue d'un système ouvert capable de charger des applications d'origine incertaine (non fiable) tout au long de son cycle de vie. Les problèmes temps réel sont bien connus dans le contexte de l'informatique conventionnel [28, 4]. Dans le contexte de *Camille* RT deux problèmes sont à résoudre. D'une part les cartes à microprocesseur n'ont, jusqu'à présent, pas été équipées de systèmes temps réel, et les contraintes propres au développement de logiciels encartés semble compromettre leur mise en œuvre. D'autre part, les propriétés temps réel sont peu compatibles avec les concepts et propriétés des systèmes ouverts. En effet, les systèmes temps réel requièrent un minimum d'informations sur les tâches qu'ils supportent. Une fois celles-ci identifiées et analysées, des algorithmes de partage du temps appropriés peuvent être mis en œuvre [4]. Dans un système ouvert où de nouvelles tâches peuvent être ajoutées dynamiquement dans le système, il faut être capable de s'assurer d'une part que les contraintes des tâches déjà supportées ne sont pas affectées, et d'autre part d'assurer à la nouvelle tâche chargée qu'elle aura toujours à disposition le temps nécessaire à son exécution. Notons enfin que le processus de chargement et de création d'une nouvelle tâche doit être capable lui-même de s'intégrer dans une gestion temps réel de l'activité de la carte à puce.

8. Escape Analysis.

Les points critiques que nous avons à prendre en compte sont le chargeur de code intermédiaire et plus particulièrement l'analyse du code chargé (rappelons qu'elle assure déjà la confidentialité et l'intégrité des données manipulées par les applications), elle doit maintenant être capable de détecter et de borner en temps d'exécution les sections critiques, de recevoir et de valider les exigences propres à une tâche (temps de réponse, durée de vie, ...) et d'intégrer l'activité propre à cette tâche dans l'ensemble des applications déjà chargées.

De plus la réalisation d'un système temps réel peut se baser sur deux approches : dans un premier cas, *Camille* RT respecte la philosophie préconisée par l'ISO ; dans une autre approche, l'introduction d'un ordonnanceur de tâche nécessite une refonte des modèles de communication et d'exécution. Nous privilégions la seconde approche en nous orientant vers un système multi-tâches. Il ne sera cependant pas possible de faire l'impasse sur l'intégration des problématiques d'économie d'énergie, et de gestion des entrées/sorties qui doivent être intégrées dans un ordonnanceur temps réel pour cartes à microprocesseur.

Conclusion Des architectures extensibles telles que celle de *Camille* ont montré leur aptitude à charger dynamiquement de nouvelles fonctionnalités systèmes [16, 17] (système de fichiers, générateur de nombres aléatoires, ...). L'intégration de propriétés non fonctionnelles telles que les modèles de mémoires recouvrables ont pu être mise en œuvre au-dessus du micro-noyau *Camille* [10]. Néanmoins d'autres propriétés non fonctionnelles telles que le temps réel, qu'il soit statique ou dynamique, et le contrôle de ressources nécessitent un enrichissement du système de type impliquant une retouche du langage intermédiaire FAÇADE.

3.3 Plan de travail

Le micro-noyau *Camille* a été entièrement développé en assembleur sur un microprocesseur AVR 8/16 bits de la société ATMEL (5ko RAM, 32ko EEPROM de données, 32ko EEPROM de code). Si ce choix a été fait dans une optique de code minimaliste ayant des performances d'exécution optimales afin de prouver les concepts de l'architecture *Camille*, il a pour défaut de ne pouvoir être exploité tel quel sur d'autres architectures matérielles. Une réécriture du noyau en langage C devra être étudiée.

Une réflexion au niveau de l'architecture *Camille* devra être faite en vue d'intégrer les évolutions des nouveaux processeurs cartes (MMU, cache processeur). Le compromis entre sécurité et abstraction matérielle de bas niveau sera conservé afin de charger des extensions système. Notre objectif est de garder les fondements de *Camille* en terme de sécurité et système flexible en lui rajoutant des propriétés de qualité de service liées à la gestion des ressources mémoires et temporelles (CPU). Notre dessein est d'avoir une base commune pour le noyau encarté, puis deux orientations, l'une nommée *Camille RC* plus axée sur la gestion de la mémoire l'autre *Camille RT* spécialisée dans la gestion du temps CPU (au sens système temps réel). Le dernier travail consiste à réunir dans un même noyau les travaux de contrôle des ressources et de temps réel. Il s'agira de réunir ces deux aspects dans un unique ordonnanceur. La difficulté résidant dans le fait de définir une politique d'ordonnancement compatible avec ses deux contraintes.

4 Conclusion

La gestion et le contrôle des ressources (mémoire, processeur et communications) dans les systèmes à la fois sûrs et flexibles restent aujourd'hui un facteur non maîtrisé. Si l'on souhaite faire cohabiter dans les cartes à puce les concepts de cartes ouvertes et cartes multi-tâches, il est primordial d'aller plus loin que la vérification d'intégrité et de confidentialité des données. Pallier les attaques en disponibilité (mémoire ou temps) est donc une étape nécessaire à cette réalisation.

Références

[1] 3RD GENERATION PARTNERSHIP PROJECT (3GPP). Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM–ME) interface. Reference Number : GSM 11.14, 1999. `http://www.3gpp.org`.

[2] BERSHAD, B. N., SAVAGE, S., PARDYAK, P., SIRER, E. G., FIUCZYNSKI, M. E., BECKER, D., CHAMBERS, C., AND EGGERS, S. Extensibility, safety and performance in the SPIN operating system. In *the 15th ACM Symposium on Operating Systems Principles (SOSP'95)* (Dec. 1995). `http://www.cs.washington.edu/research/projects/spin/www/`.

[3] BLANCHET, B. Escape Analysis for Object Oriented Languages. Application to Java™. In *ACM conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99)* (Denver USA, november 1999). `http://pauillac.inria.fr/~bblanche/escape.html`.

[4] CARDEIRA, C., AND MAMMERI, Z. Ordonnancement de tâches dans les systèmes temps réel et répartis. *Revue RAIRO, Automatique Productique Informatique Industrielle (APII) 28* (1994), 353–384.

[5] CARDELLI, L., DONAHUE, J., GLASSMAN, L., JORDAN, M., KALSOW, B., AND NELSON, G. Modula-3 language definition. In *ACM SIGPLAN Notices* (Aug. 1992), pp. 15–42. 27(8).

[6] CHEN, Z. *Java Card™ Technology for Smart Cards : Architecture and Programmer's Guide*. The Java™ Series. Addison Wesley, June 2000. `http://java.sun.com/products/javacard/`.

[7] CHOI, J.-D., GUPTA, M., SERRANO, M., SREEDHAR, V. C., AND MIDKIFF, S. Escape Analysis for Java™. In *ACM conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99)* (Denver USA, november 1999), pp. 2–19.

[8] Colin, A., and Puaut, I. Worst-Case Execution Time Analysis of the RTEMS Real-Time Operating System. In *the 13th Euromicro conference on realtime systems* (Delft, The Netherlands, June 2001). http://www.irisa.fr/solidor/doc/ps01/ecrts2hades.ps.gz.

[9] Crary, K., and Weirich, S. Resource Bound Certification. In *the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Boston, MA, USA, January 2000), pp. 184–198. http://www.cs.cornell.edu/talc/papers/resource_bound/res-abstract.html.

[10] Donsez, D., Grimaud, G., and Lecomte, S. Recoverable Persistant Memory of SmartCard. In *the 3rd Smart Card Research and Advanced Application Conference (CARDIS'98)* (Louvain-la-Neuve, Belgium, September 1998), no. 1820 in LNCS, Springer-Verlag, pp. 13–26. http://www.lifl.fr/RD2P/pub/DGL98/.

[11] Engler, D. R., and Kaashoek, M. F. Exterminate All Operating System Abstractions. In *the 5th Workshop on Hot Topics in Operating Systems (HotOS V)* (Orcas Island, Washington, May 1995), IEEE Computer Society, pp. 78–83. http://www.pdos.lcs.mit.edu/papers/hotos-jeremiad.ps.

[12] Engler, D. R., Kaashoek, M. F., and Jr, J. O. Exokernel: An operating system architecture for application-level resource management. In *the 15th Symposium on Operating Systems Principles (SOSP'95)* (Copper Mountain, Colorado, 3-6 December 1995), pp. 251–266. http://www.pdos.lcs.mit.edu/~engler/sosp-95.ps.

[13] Engler, D. R., Kaashoek, M. F., and O'Toole Jr., J. W. The operating system kernel as a secure programmable machine. In *the 6th ACM SIGOPS European workshop: Matching operating systems to application needs* (Dagstuhl Castle, Wadern, Germany, September 1994), pp. 62–67. http://www.pdos.lcs.mit.edu/papers/xsigops.ps.

[14] Folliot, B., Piumarta, I., and Riccardi, F. Virtual Virtual Machines. In *the 4th Cabernet Radical Workshop* (Rethimnon, Crete, 17-20 September 1997). http://www-sor.inria.fr/publi/VVM_radical97.html.

[15] Grimaud, G. Introduction à une Architecture Logicielle Nouvelle pour les Cartes à Microprocesseur Ouvertes. In *French Chapter of ACM-SIGOPS: CFSE'1* (Rennes, France, Juin 1999), V. Issarny, Ed., pp. 13–24. http://www.lifl.fr/RD2P/pub/G99/.

[16] Grimaud, G. *CAMILLE: un système d'exploitation ouvert pour carte à microprocesseur.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, décembre 2000. http://www.lifl.fr/~grimaud/memoire/Camille.ps.gz.

[17] Grimaud, G., and Deville, D. Evaluation d'un micro-noyau dédié aux cartes à microprocesseur. In *French Chapter of ACM-SIGOPS: CFSE'2* (Paris, France, Avril 2001), B. Folliot and P. Sens, Eds., pp. 117–127. http://www-src.lip6.fr/cfse2/articles/Grimaud_deville.ps.

[18] Grimaud, G., Lanet, J.-L., and Vandewalle, J.-I. FACADE: A Typed Intermediate Language Dedicated to Smart Cards. In *Software Engineering–ESEC/FSE'99* (Toulouse, France, October 1999), no. 1687 in LNCS, Springer, pp. 476–493. http://www.gemplus.com/smart/r_d/publications/art29.htm.

[19] Grimm, R., and Bershad, B. N. Security for extensible systems. In *the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)* (Cape Cod, Massachusetts, May 1997), pp. 62–66. http://www.cs.washington.edu/homes/rgrimm/papers/hotos97.pdf.

[20] International Standard Organizaqion for Standardization (ISO). Information technology – Identification cards – Integrated circuit(s) cards with contacts – *Part IV*: interindustry commands for interchange, 1995. Reference Number: ISO/IEC 7816-4.

[21] International Standard Organization for Standardization (ISO). Information technology – Identification cards – Integrated circuit(s) cards with contacts – *Part III*: Electronic signals and transmission protocols, 1997. Reference Number: ISO/IEC 7816-9.

[22] Kömmerling, O., and Kuhn, M. G. Design Principles for Tamper-Resistant Smartcard Processors. In *USENIX Workshop on Smartcard Technology (Smartcard'99)* (Chicago, Illinois, USA, May 10–11 1999), pp. 9–20. http://www.cl.cam.ac.uk/~mgk25/sc99-tamper.pdf.

[23] Kozen, D. Language-Based Security. In *Conf. Mathematical Foundations of Computer Science (MFCS'99)* (September 1999), M. Kutylowski, L. Pacholski, and T. Wierzbicki, Eds., vol. 1676 of *LNCS*, Springer-Verlag, pp. 284–298. http://www.cs.cornell.edu/kozen/papers/lbs.ps.

[24] Lanet, J.-L. Are Smart Cards the Ideal Domain for Applying Formal Methods? In *International Conference of Z and B Users (ZB'2000)* (York, UK, September 2000), vol. 1878 of *LNCS*, Springer-Verlag, pp. 363–374. http://research.gemplus.com/smart/r_d/publications/index.html.

[25] Leroy, X. Java bytecode verification: an overview. In *the 13th Conference on Computer Aided Ferification (CAV'2001)* (Paris, France, July 2001), vol. 2102 of *LNCS*. http://pnuillac.inria.fr/~xleroy/.

[26] Leroy, X. On-card bytecode verification for java card. In *International Conference on Research in Smart Cards, E-smart 2001* (Cannes, France, September 19–21 2001), I. Attali and T. Jensen, Eds., vol. 2140 of *LNCS*, Springer-Verlag. http://pauillac.inria.fr/~xleroy/.

[27] Lindholm, T., and Yellin, F. *The Java™ Virtual Machine Specification.* The Java™ Series. Addison-Wesley, September 1996. http://java.sun.com/docs/books/vmspec/.

[28] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *ACM 20*, 1 (January 1973), 46–61.

[29] Maosco Consortium Ltd. Multos™: The multi-application operating system for smart cards. http://www.multos.com.

[30] Mazières, D., and Kaashoek, M. F. Secure applications need flexible operating systems. In *the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)* (May 1997). http://www.pdos.lcs.mit.edu/papers/mazieres:hotos6.ps.gz.

[31] Microsoft Corp. Windows for Smart Cards. http://www.microsoft.com/smartcard/.

[32] Moore, J. T., Hicks, M., and Nettles, S. Practical Programmable Packets. In *the 20th IEEE Computer and Communication Society INFOCOM Conference* (April 2001), IEEE. `http://www.cis.upenn.edu/~switchware/papers/snap.pdf`.

[33] Morrisett, G., Walker, D., Crary, K., and Glew, N. From system F to Typed Assembly Language. In *the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, CA, USA, January 1998), pp. 85–97. `http://www.cs.cornell.edu/talc/papers/tal-popl.ps.gz`.

[34] Necula, G. C., and Lee, P. Safe kernel extensions without run-time checking. In *the 2nd USENIX of Symposium on Operating System Design and Implementation (OSDI'96)* (Seattle, Washington, 28-31 October 1996), USENIX Assoc., pp. 229–243. `http://www.cs.cmu.edu/~necula/osdi96.ps.gz`.

[35] Ramamritham, K., and Stankovic, J. Real-Time Systems: The Spring Project. `http://none.cs.umass.edu/rts/spring.html`.

[36] Requet, A., Casset, L., and Grimaud, G. Application of the B formal method to the proof of a type verification algorithm. In *the 5th IEEE High Assurance Systems Engineering Symposium (HASE 2000)* (Albuquerque, New Mexico, November 2000).

[37] Salter, J., and Schroeder, M. The protection of information in computer systems. In *proc. of IEEE* (September 1975), vol. 63, No. 9. `http://web.mit.edu/Saltzer/www/publications/protection/index.html`.

[38] Schneider, F. B., Morrisett, G., and Harper, R. A Language-Based Approach to Security. In *Informatics: 10 Years Back, 10 Years Ahead* (2001), R. Wilhelm, Ed., vol. 2000 of *LNCS*, Springer-Verlag, pp. 86–101. `http://www.cs.cmu.edu/~rwh/papers/langsec/dagstuhl.pdf`.

[39] Seltzer, M., Endo, Y., Small, C., and Smith, K. Dealing with Disaster: Surviving Misbehaved Kernel Extensions. In *the 2nd Symposium on Operating System Design and Implementation (OSDI'96)* (1996). `http://www.eecs.harvard.edu/~vino/vino/osdi-96/paper.ps`.

[40] Seltzer, M., Endo, Y., Small, C., and Smith, K. A. An Introduction to the Architecture of the VINO Kernel. Tech. Rep. 34–94, VINO: The 1994 Fall Harvest, Harvard Computer Center for Research in Computing Technology, Dec. 1994. `ftp://das-ftp.harvard.edu/techreports/tr-34-94.ps.gz`.

[41] Small, C. A Tool for Constructing Safe Extensible C++ Systems. In *the 3rd USENIX Conference on Object-Oriented Technologies (COOTS'97)* (June 16-20 1997). `http://www.eecs.harvard.edu/~vino/vino/papers/misfit.ps`.

[42] Stankovic, J. A., and Al. Strategic Directions in Real-Time and Embedded Systems. In *ACM Computing Surveys* (December 1996), vol. 28, No. 4.

[43] Wahbe, R., Lucco, S., Anderson, T. E., and Graham, S. L. Efficient Software-Based Fault Isolation. In *the 14th ACM Symposium on Operating Systems Principles (SOSP'93)* (Dec. 1993), pp. 203–216. `http://www.ifi.uio.no/~oddvar/bib/papers/WLAS93.ps`.

[44] Zuberi, K. M., Pillai, P., and Shin, K. G. EMERALDS: A small-memory real-time microkernel. In *the 17th ACM Symposium on Operating Systems Principles (SOSP'99)* (December 1999), pp. 34(5):277–291. `http://kabru.eecs.umich.edu/rtos/emeralds.html`.