SGBD embarqué dans une puce : retour d'expérience

Nicolas Anciaux, Luc Bouganim, Philippe Pucheral

SGBD embarqué dans une puce : retour d'expérience

Nicolas Anciaux* — Luc Bouganim* — Philippe Pucheral*,**

* *SMIS Project, INRIA Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay Cedex, France. Email : {prénom.nom}@inria.fr*

** *Laboratoire PRiSM, Université de Versailles/St-Quentin, 78035 Versailles Cedex, France. Email : Philippe.Pucheral@prism.uvsq.fr*

RÉSUMÉ. *La carte à puce est aujourd'hui l'objet portable sécurisé le plus répandu. Il y a 4 ans, nous avons jeté les bases d'une étude portant sur l'embarquement de techniques bases de données dans une carte à puce. Cette étude a conduit à la définition de principes de conception pour ce que nous avons appelé alors PicoDBMS, un système de gestion de bases de données (SGBD) complet intégré dans une carte à puce. Depuis, grâce au progrès du matériel et aux efforts conjoints de notre équipe et de notre partenaire industriel, les principes définis initialement ont donné naissance à un prototype complet tournant sur une plate-forme carte à puce expérimentale. Cet article reconsidère la formulation du problème initial à la lumière des évolutions matérielles et applicatives. Il introduit ensuite un banc d'essai dédié aux bases de données embarquées dans des puces et présente une analyse de performance détaillée de notre prototype. Enfin, il dresse des perspectives de recherche dans le domaine de la gestion de données dans les puces sécurisées.*

ABSTRACT. *Smart card is today the most widespread secured portable computing device. Four years ago, we addressed the problem of scaling down database techniques for the smart card and we proposed the design of what we called a PicoDBMS, a full-fledged database system embedded in a smart card. Since then, thanks to the hardware progress and to the joint implementation efforts of our team and our industrial partner, this utopian design gave birth to a complete prototype running on an experimental smart card platform. This paper revisits the problem statement in the light of the hardware and applications evolution. Then, it introduces a benchmark dedicated to Pico–style databases and provides an extensive performance analysis of our prototype, discussing lessons learned at experimentation time and helping selecting the appropriate storage and indexation model for a given class of embedded applications. Finally, it draws new research perspectives for data management on secured chips.*

MOTS-CLÉS : *Bases de données embarquées, Confidentialité et protection des données, Carte à puce, Contrôle d'accès, Modèles de stockage et d'indexation, Evaluation de requêtes.*

KEYWORDS : *Embedded databases, Tamper-resistant databases, Smart card, Access control, Storage and indexation models, Query evaluation.*

1. Introduction

En 1977, le brevet de Roland Moreno lançait l'histoire prestigieuse de la carte à puce, devenue aujourd'hui le dispositif informatique portable et sécurisé le plus répandu. La carte à puce est utilisée dans des applications aussi variées que les prestations bancaires, la télévision payante ou la téléphonie GSM, les cartes de fidélité, de santé ou de transport. Les standards multi-applications, comme JavaCard [Chen, 2000], ont révolutionné le développement et la distribution des applications carte à puce. Dès lors que les cartes sont devenues polyvalentes, multi-applications et puissantes (dotées de processeurs 32 bit et de plus d'un MB de stockage persistant) le besoin de techniques bases de données est apparu. Principalement, l'embarquement d'un évaluateur de requêtes, d'un contrôleur de droits d'accès et de techniques transactionnelles dans la puce rend le code applicatif embarqué plus compact et sûr. Cependant, la carte à puce présente des contraintes matérielles spécifiques (RAM infime, écritures très coûteuses en mémoire persistante, etc.) rendant les systèmes de gestion de bases de données (SGBD) traditionnels, ainsi que leurs versions pour dispositifs légers [Oracle Corp., 2002, Karlsson *et al.*, 2001, Seshadri *et al.*, 2000, Sybase Inc., 2000], inappropriés. Précédemment, nous avons traité du problème de l'adaptation des techniques bases de données à la carte à puce et proposé ce que l'on a appelé PicoDBMS [Pucheral *et al.*, 2001]. Ce travail préliminaire a ouvert de nouvelles perspectives de recherche orientées vers la conception de noyaux SGBD dédiés aux dispositifs informatiques ultra-légers.

Quatre ans après la publication de PicoDBMS, trois questions importantes méritent d'être posées. La première est indéniablement " l'adaptation de techniques bases de données aux cartes à puce est-elle aujourd'hui encore d'actualité ? ". En d'autres termes, il est nécessaire de s'interroger sur l'évolution de la problématique initiale face aux progrès matériels et à l'évolution des applications. " Le design de PicoDBMS est-il effectivement adapté à des plates-formes matérielles réelles ? " constitue aussi une question importante relative à la faisabilité de l'approche et introduit une discussion sur les enseignements tirés au stade expérimental. Enfin, " PicoDBMS est-il la réponse définitive à un problème ponctuel ou plutôt un premier pas vers un large champ de recherche à long terme ? " est une question légitime mettant en perspective les composants bases de données embarqués dans des puces sécurisées. Le but de cet article est de répondre à ces trois questions.

Concernant l'évolution de la problématique, les ressources matérielles des cartes à puce n'échappent pas à la loi de Moore en terme de cadence du processeur et de taille mémoire. Cependant, le problème posé dans [Pucheral *et al.*, 2001] est axé sur la dissymétrie des ressources (processeur puissant mais RAM réduite, lectures très rapides en mémoire persistante mais écritures très lentes, etc.) plutôt que sur leur insuffisance. L'équilibre des ressources n'a pas beaucoup évolué et devrait rester relativement stable à moyen terme. Il est toutefois indispensable d'estimer l'impact précis de l'évolution du matériel sur le design initial de PicoDBMS. Les besoins applicatifs ont récemment subit une mutation importante. PicoDBMS considérait principalement la gestion de dossiers portables sécurisés, et plus particulièrement de dossiers médicaux. Ces dernières années, les dossiers portables sécurisés ont été

considérés avec un nouvel intérêt par les industriels, cherchant à élargir leurs domaines d'applications (MasterCard propose de stocker des dossiers de fidélité/historique d'achats/etc. sur ses cartes bancaires [Mastercard Inc., 2002]). De plus, l'introduction de puces sécurisées dans les systèmes informatiques courants conduit à de nouvelles applications, à grande échelle. Premièrement, les puces sécurisées sont introduites dans les PC [TCPA, STMicroelectronics, 2004] et les dispositifs clients [Smartright] pour éviter le piratage et offrir des droits digitaux élaborés [XrML, ODRL] où les conditions sont évaluées sur le profil du consommateur et/ou des données historiques. Dorénavant, ces données aussi doivent être protégées. Deuxièmement, l'intelligence ambiante envahit notre vie quotidienne avec des objets informatiques traquant nos habitudes et nos préférences, menaçant les droits élémentaires des citoyens à la protection de la vie privée. Le besoin de dossiers portables sécurisés protégeant des données personnelles évolue ainsi vers celui de dossiers sécurisés fixes ou portables, assurant confidentialité et intégrité des données personnelles. La première contribution de cet article est de reconsidérer la problématique de PicoDBMS face aux évolutions matérielles et applicatives.

Un prototype complet de PicoDBMS a été initialement développé en JavaCard et démontré sur une carte à puce Palmera, plate-forme commerciale fournie par Axalto [Anciaux *et al.*, 2001]. Ce prototype valide le fonctionnement des techniques proposées, mais ses performances restent très faibles. Le prototype a été réécrit en langage C (et optimisé dans un souci de performance) et adapté à Zeplatform, le système d'exploitation servant de base à la prochaine génération de cartes à puce d'Axalto. Trois ans furent nécessaires à l'obtention de cette plate-forme et à l'adaptation (avec l'aide d'Axalto) du noyau du système d'exploitation aux exigences de performances des applications embarquées orientées données. Un banc d'essai dédié aux bases de données de type PicoDBMS a été défini et utilisé pour juger des performances relatives des différentes structures de stockage et d'indexation des données. De plus, un simulateur matériel a été utilisé pour prédire les performances de PicoDBMS sur des bases de données au volume excédant la capacité de stockage des cartes actuelles. Comme deuxième contribution, l'article expose les enseignements tirés au cours de ce long (et délicat) processus d'expérimentation.

Si PicoDBMS est une réponse appropriée au problème initial (dossiers portables sécurisés sur une carte à puce), cela ne signifie pas que la solution englobe tous les cas où la gestion de données peut bénéficier d'une puce sécurisée. Les constructeurs de puces électroniques annoncent à court terme des cartes à stockage massif [MOPASS] intégrant une puce sécurisée reliée par un bus à une mémoire persistante non sécurisée de grande capacité (plusieurs GB). Dans ce contexte, le volume de données à considérer et les nouvelles formes d'attaques à contrer imposent de reconsidérer les techniques bases de données sur la puce. Les environnements sans contact, les futures technologies de stockage persistant, le besoin de co-conception logiciel/matériel sont d'autres considérations dignes d'intérêt du point de vue de la recherche. Comme troisième contribution, cet article esquisse des perspectives de recherche liées à la gestion de données dans des puces sécurisées.

L'article est organisé comme suit. La section 2 revisite la problématique de PicoDBMS au vu de l'évolution des technologies et des usages de la carte à puce,

répondant à la première question posée en introduction, puis présente les travaux connexes. La section 3 rappelle les aspects importants du noyau de PicoDBMS (modèle de stockage et évaluateur de requêtes) décrits dans [Pucheral *et al.*, 2001]. Cette description technique donne les pré-requis à l'analyse de performance de la section suivante. La section 4 introduit les objectifs de l'expérimentation, expose les métriques choisies, et discute chaque structure de stockage envisagée face à ces métriques, répondant à la deuxième question posée en introduction. La section 5 aborde la troisième question et identifie des perspectives de recherche prometteuses dans le cadre de la gestion de données sur puce sécurisée. La section 6 conclut l'article.

2. Evolution de la problématique

Les différentes composantes d'une puce sécurisée sont intégrées sur un même dé de silicium, le tout enveloppé d'une maille sécurisée. Ainsi, la corruption/examen des informations embarquées (code et données) est inopérant et/ou aboutit à la destruction (du contenu) de la puce. L'approche PicoDBMS repose donc sur deux hypothèses : les données embarquées dans la puce sont sécurisées (le chiffrement interne des données de la base est donc inutile) ; tous les traitements qui y sont effectués le sont aussi (i.e., les résultats produits sont complets et corrects).

La puce étant considérée comme une enceinte inviolable, la puissance des mécanismes de confidentialité repose sur les capacités du SGBD embarqué à calculer des vues (la confidentialité des SGBD est obtenue en donnant droit à des vues calculées à partir des données de base). Un moteur d'évaluation de requêtes puissant permet de fonder les autorisations sur des vues finement définies. Ainsi, dans notre contexte, assurer la confidentialité des données embarquées revient à embarquer les techniques d'exécution de requêtes permettant d'évaluer des vues complexes dans la puce. C'est sur ce constat que repose le travail préliminaire présenté dans [Pucheral *et al.*, 2001], proposant des règles de conception, des structures de stockage et d'indexation, et des techniques d'exécution de requêtes adaptées aux ressources de la puce.

Nous analysons ici l'évolution des technologies et usages des puces sécurisées, reformulons le problème en conséquence et présentons les travaux existants.

2.1. Evolution des technologies carte à puce

Les puces des cartes actuelles sont sécurisées au niveau matériel par un maillage détruisant le contenu de la puce en cas d'intrusion physique (couche d'argent). Elles disposent d'un processeur 32 bit cadencé à 50 MHz, de modules mémoire dotés de 96 KB de ROM, 4 KB de RAM et 64 KB d'EEPROM, et de modules de sécurité assurant la protection physique (détection de variations anormales de voltage, détecteur de rayonnement, etc.) et logique (brouillage du bus de données, firewall protégeant la mémoire, etc.) de la puce. Un co-processeur cryptographique est souvent aussi intégré sur la puce. La ROM stocke le système d'exploitation, des données statiques et des routines standard. La RAM sert de mémoire de travail (pile

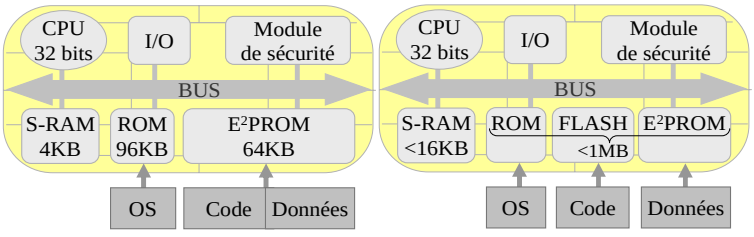
et tas). L'EEPROM stocke les informations persistantes, les données et les applications téléchargées (dans le cas de cartes multi-applications). La Figure 1.a présente l'architecture d'une carte à puce actuelle.

Les ressources internes de la puce présentent des asymétries originales. Le processeur, calibré pour effectuer des calculs cryptographiques et assurer la sécurité de la puce, est surdimensionné par rapport au volume de données embarquées. De plus, la mémoire persistante (EEPROM) partage les caractéristiques d'une mémoire RAM en terme de grain d'accès (mot) et de temps de lecture (<100 ns par mot), mais souffre d'un temps d'écriture extrêmement lent (environ 10 ms par mot). Enfin, seules quelques centaines d'octets de RAM sont disponibles pour les applications embarquées, la majeure partie de la RAM étant réservée au système d'exploitation et à la machine virtuelle (cartes Java). Cet environnement informatique original (comparé aux serveurs bases de données classiques) peut être caractérisé par les propriétés suivantes :

(P1) Processeur très puissant comparé au faible volume de données embarquées.
(P2) RAM extrêmement réduite comparée au volume de données à traiter.
(P3) Lectures rapides et écritures lentes en mémoire persistante.

La première étude de PicoDBMS fut conduite suivant ces propriétés. Il est donc important d'examiner leur pérennité face à l'évolution des techniques matérielles.

Analyse de la propriété P1 : les processeurs embarqués sont passés en dix ans de 8 bit/2 MHz à 32 bit/50 MHz et peuvent sembler suffisamment puissants pour satisfaire les besoins des applications. Cependant, trois arguments au moins plaident pour une augmentation continue de la puissance du processeur. D'abord, l'évolution rapide du débit des cartes rend possible les traitements cryptographiques sur des flots de données sollicitant fortement le processeur, comme la télévision payante basée sur du déchiffrement par carte à puce [Potonniée, 2004]. En outre, de nombreux efforts de recherche et investissements des constructeurs visent les systèmes d'exploitation encartés multi-applications (voire multi-tâches) qui requièrent aussi un processeur puissant. Enfin, le coût d'une carte à puce dépend principalement de la quantité de mémoire embarquée (RAM et EEPROM), mais peu de la puissance de calcul. Les cartes portant sur un marché de masse, ce constat favorise l'amélioration de la puissance du processeur plutôt que la quantité de mémoire.



a. Cartes actuelles. *b. Cartes à venir (court terme).*

Figure 1. *Architectures des cartes à puces.*

Analyse de la propriété P2 : les cartes à puce sont dotées de quelques KB de RAM presque entièrement dédiés au système d'exploitation et à la machine virtuelle (cartes Java). Malgré l'évolution de la puissance du processeur, l'écart entre l'évolution de la RAM disponible pour les applications et l'évolution des autres ressources (vitesse du processeur et quantité de mémoire persistante) continue à croître pour les raisons suivantes. D'abord, les constructeurs cherchent à minimiser les ressources matérielles embarquées pour réduire les coûts de production sur les marchés de masse. Le coût d'une puce étant directement lié à sa surface, la densité faible de la RAM (16 fois moins compacte que la ROM) en fait un composant critique pour le coût de la puce [Anciaux *et al.*, 2003]. De plus, il n'existe pas de motivation forte en faveur d'une augmentation de la quantité de RAM car (i) les lectures en mémoire persistante sont directes (accès aléatoire) et rapides, rendant la mémoire tampon non obligatoire et (ii) la RAM fait concurrence aux autres composants du dé de silicium [NRC report, 2001, Gupta *et al.*, 1998]. Ainsi, l'ajout de RAM se fait au détriment de la mémoire persistante, alors que les constructeurs privilégient le volume de données embarquées. La RAM est donc strictement réservée à la pile d'exécution nécessaire aux applications embarquées, la RAM additionnelle ne se justifiant que pour supporter un système d'exploitation ou une machine virtuelle plus puissant.

Analyse de la propriété P3 : examinons l'impact des progrès matériels sur la quantité de mémoire stable et sur l'asymétrie lecture/écriture. Les puces sont basées sur une technologie matérielle éprouvée voire obsolète (0.35 micron) réduisant les coûts de production et améliorant la sécurité [Schneier *et al.*, 1999]. Cependant, la pression générée par les nouvelles applications conduit à une augmentation rapide de la capacité de stockage. La gravure à 0.18 micron permet d'embarquer jusqu'à 256 KB d'EEPROM. Parallèlement, les constructeurs visent à intégrer des mémoires plus denses, comme les FLASH de type NOR et NAND. La nature XIP de la FLASH NOR (eXecute-In-Place, i.e., permet d'exécuter le code sans le charger en RAM) la rend propice à stocker du code, mais les coûts extrêmes en modification en font un support peu adapté au stockage des données. La FLASH de type NAND constitue pour cela une meilleure option, mais son utilisation est difficile. Les lectures et les écritures sont effectuées au niveau page et les réécritures sont précédées de l'effacement complet du bloc concerné (couramment constitué de 64 pages). A long terme, les recherches menées sur les mémoires visent à développer la parfaite alternative : une mémoire compacte non volatile offrant des accès rapides et à grain fin en lecture et écriture (technologies MEMS ou PCM). Cependant, l'intégration de ces technologies dans les puces est une perspective à très long terme vu les difficultés de ce contexte (le haut niveau de sécurité force à prouver les nouvelles technologies, la contrainte du coût de production favorise des technologies éprouvées bon marché, et l'intégration des différents composants sur un même dé de silicium est difficile).

A la lumière de cette analyse et malgré les indéniables progrès matériels, nous considérons que les propriétés P1, P2 et P3 restent valides à court et à moyen termes (voir Figure 1.b). Les futures technologies mémoire modifieront probablement ces hypothèses, mais doivent être vues comme des perspectives à (très) long terme.

2.2. Evolution des usages de la carte à puce

La première application carte à puce fut développée par le système bancaire français dans les années 80 pour réduire les pertes liées à la fraude aux cartes magnétiques. Dès lors, les cartes à puce ont été utilisées avec succès dans la téléphonie GSM, la télévision payante [Potonniée, 2004], le porte monnaie électronique [MIPS Technologies Inc., 2002], l'accès à des ressources réseau, les assurances et la santé [Maloney *et al.*, 2001]. Aujourd'hui, des projets à grande échelle poussent à une acceptation encore plus large des cartes à puce entre applications, états et pays, pour améliorer l'accès aux systèmes de santé nationaux européens [Netc@rd project] ou protéger les réseaux, l'accès au lieu de travail, signer (de façon digitale) et chiffrer des documents aux USA [NIST, 2002]. Des projets de carte d'identité multi-usages [Moriyama *et al.*, 2004, SINCE, 2002] pouvant servir de passeport, permis de conduire, carte d'électeur, d'assurance et de transport, sont conduits en Europe, aux Etats-Unis [US-GAO, 2003, CardTechnology, 2003, Kim, 2004], au Japon [Veterans Affairs, 2001, NETLINK, 2000] et en Chine [APEC 2003]. Enfin, les puces sécurisées sont parfois intégrées au cœur des systèmes informatiques, comme dans les architectures TCPA protégeant les PC contre le piratage [TCPA, STMicroelectronics, 2004] ou SmartRight appliquant des droits digitaux au niveau des terminaux clients [Smartright]. La question est de savoir si des techniques bases de données sont requises dans ces dispositifs. Pour aider à y répondre, nous introduisons deux applications représentatives ne faisant rien d'autre qu'illustrer un besoin commun de techniques bases de données embarquées.

Le *dossier médical*, représentatif des dossiers personnels portables présentant d'importants besoins de gestion de données, a été l'exemple motivant l'étude de PicoDBMS. Les informations stockées dans les futures cartes santé (données d'assurance, d'urgence, prescriptions médicales) sont hautement confidentielles et peuvent être accédées par plusieurs utilisateurs (médecin, pharmacien) ayant des droits d'accès différents. Les spécifications HIPAA [OCR HIPAA, 2003] édictant les principes de protection des informations médicales considèrent la carte à puce comme le partenaire idéal dans ce contexte [Smart Card Alliance, 2003]. L'embarquement de techniques de gestion de données est exigé pour organiser et interroger les données de la puce et contrôler les privilèges respectifs des utilisateurs.

Le *profile du consommateur* stocke les habitudes des porteurs de carte. Dans ce cadre, MasterCard a introduit le standard *MasterCard Open Data Storage* (MODS) [Mastercard Inc., 2002] permettant au porteur de carte et à des fournisseurs de services (vendeurs, banques, etc.) de stocker et accéder des informations directement sur les cartes de crédits. L'intérêt pour les fournisseurs de services est de connaître les habitudes des clients, et le porteur peut bénéficier d'offres spéciales dépendant de sa consommation passée. Le prix à payer est élevé : l'activité du porteur est enregistrée et les croisements entre activités sont possibles. L'embarquement de mécanismes de droits d'accès fins constitue donc le pré-requis pour rendre au porteur le contrôle sur les données stockées. Le rapport IBM-Harris sur la protection de la vie privée souligne ce même besoin [IBM Harris, 2000].

2.3. Formulation du problème

Le problème initial de PicoDBMS [Pucheral *et al.*, 2001] était fortement influencé par l'exemple du dossier médical personnel et nécessitait d'embarquer sur la puce les données à protéger ainsi que les composants logiciels permettant leur gestion : évaluateur de requêtes, module transactionnel et de droits d'accès. Pour concevoir ces composants, 6 règles de design ont été introduites. Cette approche est encore valide et nous conservons certains points de l'article précédent pour formuler le problème, mais la trame est sensiblement différente : la section précédente discute plusieurs scénarios qui motivent aujourd'hui l'étude, et qui partagent tous le besoin fondamental d'un contrôle d'accès aux données garantissant la confidentialité. Cela différencie fortement PicoDBMS d'un SGBD traditionnel. L'interrogation de données elle-même ne constitue pas l'objectif premier, mais ce que l'on attend de PicoDBMS est d'assurer la confidentialité des données tout en maintenant une performance satisfaisante. Ceci implique que le SGBD puisse extraire la vue autorisée à l'utilisateur. Les autres calculs réalisés sur la vue (hors externalisation) ne sont considérés que s'ils entraînent une amélioration des performances.

Ainsi, l'objectif principal est de déterminer les techniques de gestion de données nécessaires pour assurer la confidentialité des données. Nous supposons un procédé d'identification classique (identifiant/mot de passe). PicoDBMS est conçu dans le contexte relationnel, le modèle de contrôle d'accès est donc celui de SQL. Par soucis de simplicité, nous nous concentrons sur le cœur du modèle, et considérons que les privilèges sont attribués sur des tables ou des vues. Nous introduisons ensuite les autorisations en lecture que PicoDBMS doit permettre d'exprimer et de garantir.

Par soucis de généralité, ces catégories d'autorisations sont exprimées sur un schéma entités associations. Les Autorisations sur le Schéma (SA) sont définies sans considérer les occurrences. Les Autorisations sur les Occurrences (OA) donnent accès à certaines occurrences en fonction de prédicat(s) sur leurs propriétés ou de l'existence d'une association (directe ou transitive) à une occurrence autorisée. Les Autorisations sur des données Calculées (CA) donnent droit à des valeurs calculées sans permettre l'accès aux occurrences participant au calcul (par exemple, une moyenne mais pas les valeurs utilisées dans son calcul).

Nous distinguons sept types d'autorisations sur une base de donnée relationnelle, dérivées de SA, OA et CA (cf. Table 1). Les autorisations de type SA reposent sur des vues basées sur l'opérateur de projection (droit d'accès P, projection). Le prédicat de OA peut s'appliquer sur des attributs d'une relation (accès SP, sélection – projection), de deux relations (association directe, accès SPJ sélection – projection – jointure), ou de plusieurs relations (association transitive, accès SPJ^n). Enfin, les autorisations de type CA reposent sur des vues impliquant des calculs d'agrégats. Le groupement peut être mono-attribut et n'engager qu'une relation (accès SPG, sélection – projection – jointure – groupement), mono-attribut et engager plusieurs relations (accès SPJG), ou multi-attributs (accès $SPJG^n$).

D'un point de vue performance, le SGBD doit être capable de minimiser la complexité du calcul de la vue en fonction de la requête utilisateur. Notamment, les

restrictions et projections contenues dans la requête doivent être évaluées dans la puce pour limiter l'extraction de données inutiles. Déléguer des opérations plus coûteuses à la puce (jointure entre vues, calcul d'agrégat ou tri sur le résultat d'une vue) ne réduit le temps de calcul que dans des cas très anecdotiques.

Une solution triviale pour implanter ces autorisations serait de matérialiser chaque vue dans un fichier séparé, externalisé à la demande. Le code serait minimal, facilitant son intégration dans la puce. Mais cette solution s'adapte mal à des données/autorisations dynamiques et entraîne de la redondance entre les fichiers, inconvénient majeur dans un dispositif à capacité de stockage restreinte. Ainsi, nous considérons l'évaluation dynamique des vues autorisées comme un pré-requis de la solution. Cela suppose d'embarquer des opérateurs bases de données et la possibilité d'évaluer des requêtes basées sur des vues. Le problème peut être formulé ainsi :

1 - concevoir un SGBD supportant les autorisations de type SA, OA, et CA ;
2 - supporter correctement les mises à jour des données et des droits d'accès ;
3 - être compatible avec les fortes contraintes matérielles de la puce.

Cette formulation du problème conduit à suivre un ensemble de règles de conception directement héritées des propriétés de la carte à puce :

– *Règle de compacité* : minimiser l'empreinte des données, des index et du code pour s'adapter à la quantité réduite de mémoire persistante ;
– *Règle de la RAM* : limiter la consommation RAM vu sa taille très limitée ;
– *Règle d'écriture* : éviter les écritures en mémoire stable (coût ≈10 ms/mot) ;
– *Règle de lecture* : profiter des lectures rapides en mémoire stable (<100ns/mot) ;
– *Règle d'accès* : tirer parti de la fine granularité et de l'accès direct à la mémoire stable pour les opérations de lecture et d'écriture ;
– *Règle de sécurité* : ne jamais externaliser de donnée sensible hors de la puce, minimiser la complexité algorithmique du code ;
– *Règle du CPU* : tirer parti de la puissance surdimensionnée du CPU, comparée au volume de données embarquées.

Nom	Autorisa°	Type de droit d'accès	Opérateur(s)	Exemple
P	SA	Sous-ensemble des attributs	Projection	Nom des Docteurs
SP	OA	Prédicats sur une seule relation	Sélection, Projection	Nom des Docteurs non-spécialistes
SPJ	OA	Prédicats sur une relation (association directe)	Sélection, Projection, Jointure	Nom des Docteurs consultés le mois dernier
SPJ^n	OA	Prédicats sur plusieurs relations (associations transitives)	Sélection, Projection, Jointure	Nom des Docteurs ayant prescrit des antibiotiques au patient
SPG	CA	Agrégation mono-attribut sur une seule relation	Sélection, Projection, $Group^t$ (agrégat)	Nombre de Docteurs par Spécialité
$SPJG^n$	CA	Agrégation multi-attributs sur une ou plusieurs relations	Sélection, Projection, Jointure, $Group^t$ (agrégat)	Nombre de Prescriptions par Spécialité de Docteur et par mois

Table 1. *Description des autorisations sur les données embarquées.*

2.4. Travaux existants

Dès les années 90, le besoin de gestion de données est apparu dans plusieurs applications embarquées sur calculateurs légers, allant des décodeurs de télévision câblée aux systèmes d'imagerie médicale et aux contrôleurs de vol des avions. Cela a été pour les fabricants de solutions embarquées une motivation suffisante pour s'intéresser au développement de *SGBD embarqués*. L'objectif est alors d'offrir un accès efficace aux données [Singhal *et al.*, 1992, Heytens *et al.*, 1994] et des fonctionnalités transactionnelles (ACID), cohabitant avec d'autres applications internes au calculateur (notamment celles utilisant la base de données). La conception du *SGBD embarqué* est guidée par la portabilité sur différents calculateurs et/ou systèmes d'exploitation et par la minimisation de l'empreinte du code, obtenue en liant dynamiquement le module base de données à l'application embarquée. Quelques études académiques ont été menées [Olson, Ortiz *et al.*, 2000], et de nombreux produits existent comme *Pervasive SQL 2000* [Pervasive Software Inc.], *Empress Database* [Empress Software Inc.], ou *Berkeley DB* [Olson *et al.*, 1999].

Parallèlement, l'augmentation rapide du nombre de téléphones cellulaires, assistants personnels (PDA) et autres calculateurs portables de grande consommation, pousse les principaux éditeurs de SGBD traditionnels à proposer des versions légères de leur produit pour ce marché prometteur, appelées *SGBD légers* (que nous distinguons ici des *SGBD embarqués* discutés plus haut). Ainsi, *Sybase SQL Anywhere Studio* offre une option de déploiement ultra-léger [Sybase Inc., 2000] de son produit phare *Adaptive Server Anywhere*. De même, des versions légères des SGBD principaux comme *IBM DB2 Everyplace* [IBM Corp., 1999, Karlsson *et al.*, 2001], *Oracle Lite* [Oracle Corp., 2002] et *Microsoft SQL Server for Windows CE* [Seshadri *et al.*, 2000] ont été développées. Les éditeurs de SGBD portent une attention particulière à la réplication de données et à la synchronisation avec une base centrale, pour permettre l'exécution en mode déconnecté et la synchronisation lors de la re-connexion. A nouveau, l'empreinte du code est réduite, et l'accent est mis sur l'auto-administration [Chaudhuri *et al.*, 2000]. Les SGBD légers sont portables sur une grande variété de calculateurs et de systèmes d'exploitation, comme *PalmOs*, *WinCE*, *QNX Neutrino* et *Embedded Linux*. Bien que portant sur des marchés différents, il est maintenant difficile de distinguer clairement les *SGBD embarqués* des *SGBD légers* (*Sybase Anywhere* semble même appartenir aux deux classes) vu que les uns ajoutent peu à peu des fonctionnalités bases de données (support transactionnel, interrogation SQL, etc.) pendant que les autres retirent les fonctions non indispensables pour minimiser l'empreinte du code.

Le fait de considérer les puces comme un calculateur traditionnel pourrait conduire à utiliser un *SGBD embarqué*/*léger* dans cet environnement. Cependant, les puces sont dotées de composants matériels très spécifiques (compatibles avec un haut niveau de sécurité), qui présentent des asymétries originales conduisant à repenser entièrement les techniques utilisées dans les SGBD embarqués/légers.

Les réseaux de capteurs amassant des informations météorologiques ou de trafic routier ont motivé plusieurs études récentes liées à la gestion de données dans les puces. *Directed Diffusion* [Intanagonwiwat *et al.*, 2000], *TinyDB* [Madden *et al.*,

2004], et *Cougar* [Yao *et al.*, 2002] concernent l'acquisition et le traitement de flots continus de données collectées. Des techniques d'optimisation de requêtes distribuées sur les capteurs sont étudiées dans [Bonnet *et al.*, 2000, 2001]. La réduction de la consommation d'énergie liée aux communications par ondes radio, l'une des contraintes principales de cet environnement, est proposée dans [Madden *et al.*, 2002]. Bien que les capteurs et les cartes à puce partagent certaines contraintes matérielles, les objectifs des applications sont différents de même que les techniques utilisées pour atteindre ces objectifs. Dans un capteur, les fonctionnalités bases de données nécessaires sont réduites au filtrage de données et à l'agrégation pour réduire le flot de sortie et ainsi économiser de l'énergie [Madden *et al.*, 2002]. De plus, les contraintes de temps de réponse sont différentes : dans les réseaux de capteurs, les contraintes de temps peuvent être inexistantes ou temps réel, alors qu'elles sont liées à l'interaction avec l'usager dans le cas des cartes à puce. Enfin, les capteurs exécutent souvent des requêtes sur des flots de données ou sur une seule table embarquée, alors que les cartes hébergent plusieurs relations.

Un nombre réduit d'études traitent spécifiquement la gestion des données dans les cartes à puce. Les premières propositions de SGBD pour carte à puce furent CQL [Paradinas *et al.*, 1994], ISOL's SQLJava Machine et le standard ISO SCQL [ISO, 1999] spécifiant un langage bases de données pour carte à puce. Toutes deux considèrent une génération de cartes disposant de 8 KB de mémoire persistante. Bien que leur conception soit limitée à des requêtes mono-table, ces propositions illustrent l'intérêt de concevoir un SGBD dédié pour carte à puce. Plus récemment, MODS, l'initiative de Mastercard [Mastercard Inc., 2002], offre une API commune permettant aux commerçants, banques et autres organisations d'accéder et de stocker des données sur les cartes à puce des utilisateurs avec une sécurité accrue pour le porteur de carte. Cependant, MODS est basé sur des fichiers plats, des droits d'accès grossiers (au niveau du fichier), et donc propose un modèle non compact et n'offre aucune possibilité d'exécution de requêtes. Néanmoins, cette initiative montre l'intérêt du développement de techniques bases de données pour ces environnements.

Une étude récente propose des techniques de stockage spécifiques pour gérer des données dans une puce contenant de la mémoire FLASH [Bolchini *et al.*, 2003]. Là encore, la conception se limite à des requêtes mono-table et est fortement orientée par l'architecture du type de puce ciblé. Ainsi, ce travail propose de stocker les données dans de la mémoire FLASH de type NOR. Vu que les modifications dans une FLASH NOR sont très coûteuses (une simple modification impose l'effacement très coûteux d'un large bloc de données), les techniques envisagées sont orientées vers la minimisation du coût de modification (marquage de tuples[1] effacés avec un *deleted bit* et pré-allocation de tuple *dummy records*). Bien que cette étude montre l'impact des propriétés matérielles sur le noyau du SGBD, elle ne satisfait pas les contraintes des cartes et ne traite pas de l'exécution des requêtes complexes, nécessaires au calcul des données autorisées.

Enfin, remarquons que *GnatDB* [Vingralek, 2002], présenté comme un système de gestion de bases de données ultra-léger et sécurisé, peut tenir dans une puce (son

[1] Nous utilisons les mots « tuple » et « n-uplets » indifféremment dans tout le document.

empreinte est de 11 KB). En effet, GnatDB est conçu pour gérer une petite quantité de données externes, la puce étant utilisée pour protéger ces données contre les altérations accidentelles ou malicieuses. Cependant, GnatDB ne permet pas l'exécution de requêtes ni le stockage interne et l'indexation.

3. Le SGBD embarqué PicoDBMS

Le problème central de l'étude est donc d'évaluer dynamiquement les vues implantant les autorisations SA, OA et CA sur la carte. Nous rappelons les principes de PicoDBMS tirés de [Pucheral *et al.*, 2001] relatifs à cette question. Nous ne discutons pas des autres aspects, sauf lorsqu'ils jouent sur le processus d'évaluation (comme les mécanismes transactionnels), auquel cas nous donnons les informations minimales requises.

Les modules embarqués dans la puce pour garantir la confidentialité sont : un module de stockage organisant les données et les index associés dans la mémoire persistante de la puce, un évaluateur de requêtes capable de construire et d'exécuter des plans d'exécution complexes (composés d'opérateurs de sélection, projection, jointure et calcul d'agrégats), et finalement un module de droits d'accès offrant la possibilité de donner/retirer des droits sur les vues définies (évitant ainsi d'externaliser des données non autorisées). Un moteur transactionnel assurant l'atomicité de séquences de modifications doit aussi être embarqué. Les autres modules ne jouent pas sur la confidentialité. Ils peuvent donc être placés sur le terminal, dans le pilote JDBC interfaçant l'application utilisateur avec la base de données embarquée. Le schéma de la base autorisée à l'utilisateur est externalisé à la connexion, permettant au composant JDBC d'interpréter la requête et d'en vérifier la syntaxe. La clause *ORDER BY* (si elle existe) est évaluée sur le terminal. On peut noter que ces traitements externes satisfont à la règle de sécurité vu qu'ils ne concernent que des données autorisées. Ainsi, PicoDBMS permet le partage de données confidentielles entre plusieurs utilisateurs se connectant alternativement à la base. La Figure 2 illustre cette architecture.

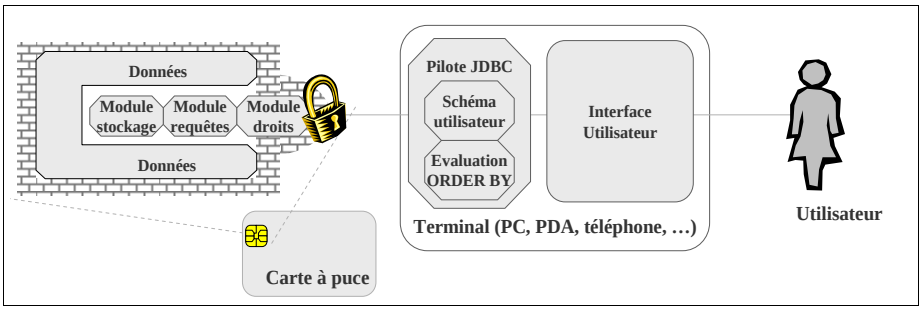


Figure 2. *Architecture de PicoDBMS.*

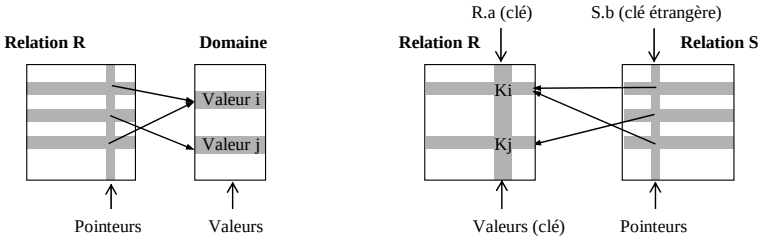
3.1. Modèle de stockage et d'indexation

La façon la plus simple d'organiser les données est le stockage à plat (FS). Les enregistrements sont placés séquentiellement sur le support de stockage, contenant les valeurs de chacun des attributs qui les composent. Le principal avantage de cette organisation est la bonne localité des données. Dans notre contexte, cette alternative présente cependant deux inconvénients majeurs. D'une part, elle est très consommatrice en espace mémoire, n'éliminant pas les doublons de valeurs. D'autre part, elle implique une exécution inefficace basée sur des opérations séquentielles en l'absence d'index. L'ajout d'index peut résoudre ce problème d'exécution, mais aggrave encore la consommation mémoire.

Le modèle de stockage d'un PicoDBMS doit garantir au mieux la compacité des données et des index. Dès lors que la localité des données n'est pas un problème dans notre contexte (règle d'accès), un modèle de stockage basé sur l'utilisation intensive de pointeurs inspiré des SGBD grande mémoire [Missikoff *et al.*, 1983, Ammann *et al.*, 1985, Pucheral *et al.*, 1990] peut favoriser la compacité. Pour éliminer les doublons, il est possible de regrouper les valeurs dans des domaines (ensembles de valeurs uniques), en utilisant le stockage en domaine (DS) comme le montre la Figure 3. Les enregistrements référencent leurs valeurs d'attributs avec des pointeurs. Ce mode de stockage est particulièrement compact pour les types énumérés, qui varient sur un ensemble réduit et déterminé de valeurs.

La mise à jour des enregistrements dont les attributs sont en domaine est plus complexe que pour des valeurs stockées à plat, mais son coût est moindre dans notre contexte simplement parce qu'elle génère moins d'écritures (règle d'écriture). Pour atténuer le léger surcoût du stockage en domaine, nous utilisons ce type de stockage uniquement pour les attributs larges (plus grands que la taille d'un pointeur) contenant des doublons. Naturellement, les attributs sans doublon (les clés) sont stockés à plat. Les attributs de taille variable (généralement plus grands qu'un pointeur) peuvent aussi bénéficier du stockage en domaine même s'ils ne présentent pas de doublon. Dans ce cas, le gain est dû à la simplification de la gestion de la mémoire (les tuples des relations sont de taille fixe) et à la compacité des journaux.

Les index doivent eux aussi être compacts. Contrairement aux bases de données traditionnelles stockées sur disque favorisant la localité des données, les puces peuvent utiliser les index secondaires (basés sur des pointeurs) de manière intensive.

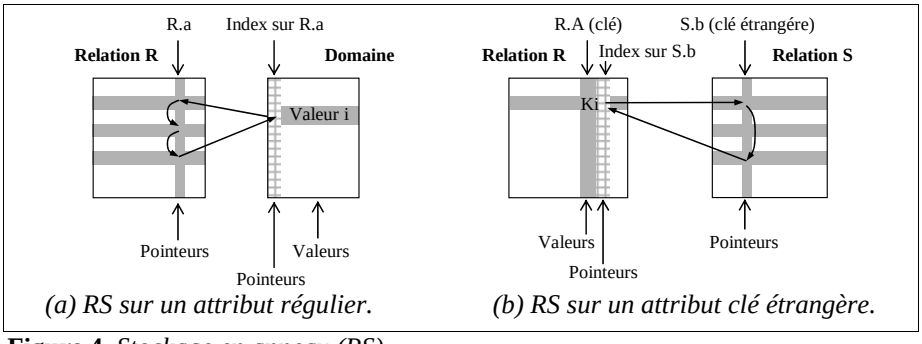


(a) Stockage d'un attribut classique. *(b) Stockage d'une clé étrangère.*

Figure 3. *Stockage en domaine.*

Tout d'abord, considérons les index de sélection. Ils sont composés de deux parties : une collection de valeurs et une collection de pointeurs reliant chaque valeur aux tuples partageant cette valeur. Si l'attribut indexé prend valeur sur un domaine, la collection de valeurs correspond au domaine rassemblant les valeurs distinctes. L'idée est de stocker la collection de pointeurs reliant ces valeurs aux tuples en lieu et place des pointeurs reliant l'attribut de chaque tuple à sa valeur de domaine. Cela nous conduit à construire une structure en anneau partant de la valeur de domaine vers les tuples (Figure 4). L'anneau de pointeurs formant l'index de sélection peut aussi être utilisé pour retrouver la valeur d'attribut des tuples, constituant ainsi un modèle de stockage que nous appelons stockage en anneau (RS). Son coût est réduit à un pointeur par valeur de domaine, quelque soit la cardinalité de la relation indexée. L'économie en stockage induit un surcoût lors de la projection (en moyenne, la moitié de la chaîne de pointeurs est suivie pour retrouver la valeur d'un attribut).

Les index de jointure [Valduriez, 1987] peuvent être construits de manière similaire. Un prédicat de jointure de type (R.a=S.b) suppose que R.a et S.b varient sur le même domaine. Stocker R.a et R.b en anneau définit un index de jointure. Ainsi, chaque valeur de domaine est reliée via deux anneaux séparés aux tuples de R et de S partageant cette valeur. La plupart des jointures se font sur un attribut clé. Soit R.a une clé primaire référencée par la clé étrangère S.b. Dans notre modèle, les clés primaires sont stockées à plat, donc R.a forme précisément un domaine, stocké à l'intérieur de la relation. Comme l'attribut S.b prend valeur dans le domaine R.a, il référence les valeurs de R.a avec des pointeurs. Ainsi, le stockage en domaine offre un index de jointure unidirectionnel naturel (Figure 3.b). Si la jointure doit être optimisée à partir de R.a vers S.b, un index bidirectionnel est nécessaire. Celui-ci peut être réalisé en définissant un anneau sur S.b (Figure 4.b). Le coût de cet index bidirectionnel est réduit à un pointeur par tuple de R, quelque soit la cardinalité de S. Cette structure d'indexation peut être améliorée en combinant RS et DS. Cela conduit à un index de jointure bidirectionnel offrant un suivi de pointeur direct (un seul suivi est nécessaire) de S.b vers R.a. Ce modèle de stockage, appelé stockage en anneau avec pointeur inverse (RIS), nécessite un pointeur additionnel par tuple de R et S. Le coût en stockage est beaucoup plus important, le gain obtenu sur les performances doit donc être très significatif pour que ce modèle soit adopté.



(a) RS sur un attribut régulier. *(b) RS sur un attribut clé étrangère.*

Figure 4. *Stockage en anneau (RS).*

3.2. Exécution de requêtes

L'exécution traditionnelle de requêtes exploite la mémoire de travail pour stocker des structures temporaires (e.g., tables de hachage) et des résultats intermédiaires, et en cas de débordement mémoire matérialise des résultats sur disque. Ceci va à l'encontre de la règle d'écriture. Pour résoudre ce problème, nous envisageons des techniques d'exécution n'utilisant pas de mémoire de travail (sauf pour stocker quelques curseurs) et proscrivant les écritures en mémoire stable.

Considérons d'abord l'exécution de requêtes SPJ (sélection – projection – jointure). PicoDBMS combine les opérateurs pour établir un plan d'exécution sous forme *d'arbre extrême droit*, conduisant à une exécution purement pipeline évitant toute matérialisation de résultats intermédiaires. Par convention, nous considérons que les opérateurs de gauche sont matérialisés et ceux de droite s'exécutent en pipeline [Schneider et al. 1990]. L'utilisation du modèle itérateur [Graefe 1993] permet d'implanter cette exécution pipeline. Le plan d'exécution est activé en partant de la racine de l'arbre d'opérateurs. Le flux de données s'établit à la demande, chaque opérateur fils rendant un tuple en réponse à un appel *next* généré par son parent.

L'opérateur de sélection vérifie le prédicat sur chaque tuple de l'entrée. Selon le modèle de stockage, les valeurs d'attribut sont lues directement dans le tuple (FS), retrouvées en déréférençant un pointeur (DS/RIS) et/ou par suivi des pointeurs formant l'anneau (RS). Avec RS/RIS, le prédicat de sélection (ou une partie, en cas de prédicat multi-attributs) peut être évalué sur les valeurs distinctes d'un domaine, les tuples qualifiés étant retrouvés en suivant l'anneau de pointeurs.

L'opérateur de projection est placé en haut de l'arbre vu qu'aucune matérialisation n'a lieu. L'opérateur de projection fabrique un tuple en copiant la valeur (FS) et/ou en déréférençant un pointeur (DS/RIS) et/ou par suivi des pointeurs formant l'anneau (RS) pour atteindre la valeur présente dans le tuple d'entrée.

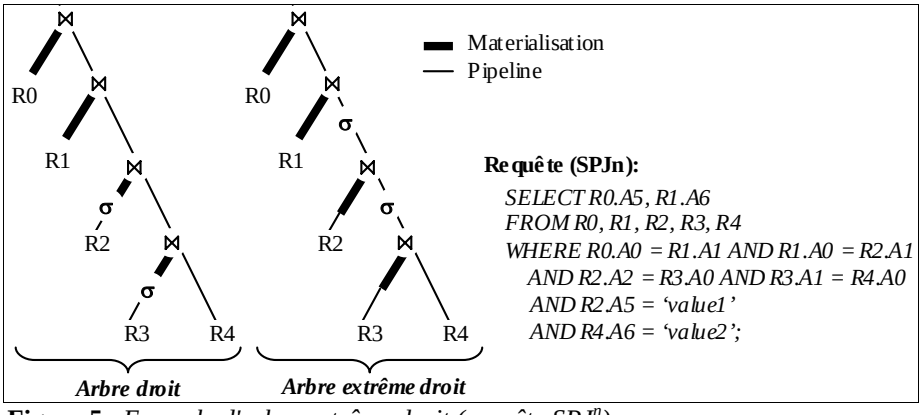


Figure 5. *Exemple d'arbre extrême droit (requête SPJ^n).*

Avec FS, les jointures sont effectuées par boucle imbriquée entre les entrées gauche et droite, dès lors qu'aucune autre technique de jointure ne permet de se passer de structures ad hoc (tables de hachage) et/ou de mémoire de travail (tri en mémoire). Chaque tuple de l'entrée droite induit une itération complète sur l'entrée gauche pour retrouver les tuples associés. Avec un index (DS/RS/RIS), le coût de la jointure dépend du sens de traversée de l'index. Considérons une jointure sur clé entre les tables R (n tuples) et S (m tuples) où S référence R avec une clé étrangère. Avec DS, le coût de la jointure est d'ordre *m* en partant de S (l'entrée droite est S), et *n*m* en partant de R. Avec RS, le coût est d'ordre *n+m* en partant de R et $m^2/2n$ en partant de S (en moyenne, la moitié de l'anneau est parcourue pour retrouver le tuple de R associé à chaque tuple de S). RIS combine les meilleurs cas de DS et RS.

Ensuite, considérons l'exécution de requêtes contenant des opérateurs de calcul d'agrégats (le tri n'est pas décrit vu qu'il peut être réalisé hors carte, sur le terminal). L'exécution pipeline parait a priori incompatible avec ces opérateurs, s'exécutant traditionnellement sur des résultats intermédiaires matérialisés. Notre solution exploite deux propriétés : (i) les calculs d'agrégats peuvent être réalisés en pipeline sur un flot de tuples groupés par valeur distincte, et (ii) les opérateurs fonctionnant en pipeline préservent l'ordre des tuples puisqu'ils consomment (et produisent) les tuples dans l'ordre d'arrivée. Ainsi, la consommation dans un ordre adéquat des tuples aux feuilles de l'arbre permet d'effectuer les calculs d'agrégat en pipeline. Si l'attribut de groupement ne fait pas partie de la relation de la feuille la plus à droite, l'arbre doit être réorganisé. Le groupement multi-attributs est plus complexe; il impose un (resp. plusieurs) opérateur de produit cartésien au bas de l'arbre pour produire les tuples ordonnés par un couple (resp. un n-uplet) de valeurs de groupement distinctes.

3.3. Techniques transactionnelles

Un seul mécanisme transactionnel a été développé, garantissant les propriétés transactionnelles dites ACID. Le coût du mécanisme est pris en compte dans l'évaluation de performances (Section 5), mais la présentation détaillée du procédé implanté n'est pas utile (ces aspects sont décrits dans [Pucheral et al. 2001]).

L'atomicité locale est assurée par un protocole ad hoc basé sur l'utilisation d'un journal d'images avant, dans lequel les pointeurs sont stockés en lieu et place des valeurs (plus le journal est petit moins il est coûteux, d'après la règle d'écriture). L'intégrité des données, bornée à un contrôle structurel (i.e., unicité et intégrité référentielle), est prise en compte dans l'étude de performance. Le contrôle de concurrence n'est pas nécessaire, PicoDBMS ayant été développé pour des plateformes mono-tâche, utilisées par un seul utilisateur à la fois. Enfin, les coûts liés à la durabilité et à l'atomicité globale (dans un environnement distribué) peuvent être intégralement délégués au réseau grâce à un protocole externe ad hoc [Abdallah et al. 2002], et n'ont pas d'impact sur les performances de PicoDBMS.

4. Mesures de performances et enseignements

Cette section évalue les performances de PicoDBMS et tire les enseignements de ce travail. Nous déterminons tout d'abord la métrique des évaluations. Ensuite, nous présentons la taille du code du SGBD embarqué et nous évaluons la compacité des données pour chaque modèle de stockage et d'indexation candidat. Puis, nous conduisons une évaluation complète des performances d'exécution de requêtes combinant des résultats de mesures et de simulations.

4.1. Métrique des évaluations

En s'axant sur le débit transactionnel, les bancs d'essai existants de SGBD (famille TPC [TPC]) sont clairement inadaptés à PicoDBMS. Ainsi, la première difficulté consiste à identifier les caractéristiques des bases de données de type PicoDBMS pour déterminer la métrique de notre étude de performances. Ces caractéristiques principales sont :

(C1) Faible borne supérieure pour l'empreinte de la base de donnée : dizaines de KB à 1 MB (cf. Section 2, Figure 1).

(C2) Profil de mises à jour de type ajout : les bases de données de type PicoDBMS gèrent souvent des historiques (modifications et suppressions sont moins fréquentes).

(C3) Droits d'accès sophistiqués : les vues autorisées peuvent combiner projections, sélections, jointures et calculs d'agrégats (cf. Section 2, Table 1).

(C4) Puce vue comme un support de stockage intelligent : l'utilisateur (logiciel ou personne physique) se connecte à la puce pour accéder (selon ses droits) aux données embarquées, comme avec tout support de stockage (disque, clé USB, etc.).

Les métriques de performances suivantes sont déduites de ces caractéristiques :

Capacité de stockage de données : cette mesure est exprimée en Ktuples/KB pour une base de données représentative. Elle capture (1) la faculté du SGBD à compresser les données embarquées et (2) l'empreinte du SGBD lui-même[2].

Taux d'insertion : cette mesure est exprimée en tuples insérés par seconde. Les coûts de modification et de suppression doivent être acceptables, mais sont moins significatifs pour les applications cibles.

Latence : cette mesure est exprimée en secondes passées à produire le premier tuple résultat pour un plan d'exécution donné (représentant une vue).

Taux de transmission : cette métrique est exprimée en nombre de tuples résultats produits par seconde. Elle est pertinente pour les applications intéressées au débit (par exemple pour remplir l'écran d'un téléphone cellulaire en un temps donné) ou

2. Dans une configuration réelle, le code du SGBD est placé en ROM alors que les données sont en EEPROM. Cependant, vu que la ROM et l'EEPROM cohabitent sur le même dé de silicium, les deux mesures sont significatives.

au temps d'exécution total (par exemple pour trier le résultat complet avant l'affichage).

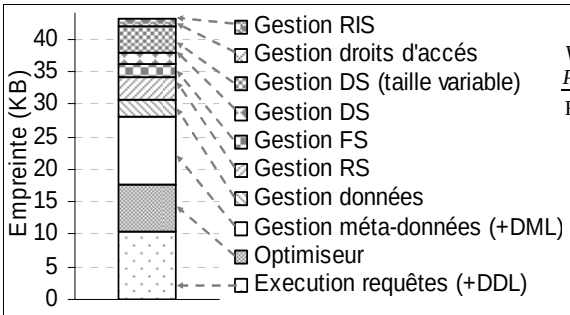
Comme cela est sous-entendu par la caractéristique C4, la latence et le taux de transmission sont comparables aux métriques utilisées pour évaluer les performances de supports de stockage traditionnels. La caractéristique C3 impose de mesurer les performances pour un large panel de requêtes (vues), représentant des droits d'accès des plus simples aux plus élaborés. Enfin, on peut noter que le débit transactionnel n'est pas pertinent dans le cas de PicoDBMS, la question étant ici de garantir un temps de réponse acceptable pour un seul utilisateur (quelques secondes est souvent considéré comme une limite acceptable pour un utilisateur humain).

4.2. Empreinte du code de PicoDBMS

La Figure 6.a présente la taille respective des modules de PicoDBMS et la Figure 6.b donne l'empreinte totale des alternatives d'implantation principales. Ces figures conduisent aux deux remarques suivantes.

D'abord, la minimisation de l'empreinte du code est souvent considérée comme un objectif majeur, notamment par les constructeurs de SGBD légers. Notre expérience contredit ce point pour les trois raisons suivantes : (1) la différence entre la version complète (*Full-Fledged*) et la version la plus réduite (*Flat-Based*) de PicoDBMS est moins importante que prévue ; (2) une cellule de ROM est 4 fois plus dense qu'une cellule d'EEPROM, ce qui réduit d'autant les bénéfices obtenus par la réduction du code ; (3) la minimisation du code conduit à stocker les données à plat (FS) ce qui dilate fortement l'empreinte des données (voir section suivante).

Ensuite, la Figure 6.a montre que la taille du module de gestion des méta-données est importante par rapport aux autres. En effet, plusieurs plans d'exécution de requêtes ne peuvent cohabiter simultanément en RAM. Cela proscrit le stockage des méta-données dans des relations, accédées par des requêtes bases de données au cours de l'exécution. Les méta-données sont donc stockées dans des structures ad hoc et sont accédées par appel de procédures, ce qui explique l'importance de l'empreinte du composant correspondant.



(a) Version complète (Full-Fledged).

Version de PicoDBMS	*Stockage*	*Indexation*	*Taille (KB)*
Flat-Based	FS	No	26
Domain-Based	FS/DS	Jointure	30
Index-Based	FS/DS/ RS	Jointure & sélection	39
Full-Fledged	FS/DS/ RS/RIS	Jointure & sélection	42

(b) Variations de PicoDBMS.

Figure 6. *Taille du code de PicoDBMS.*

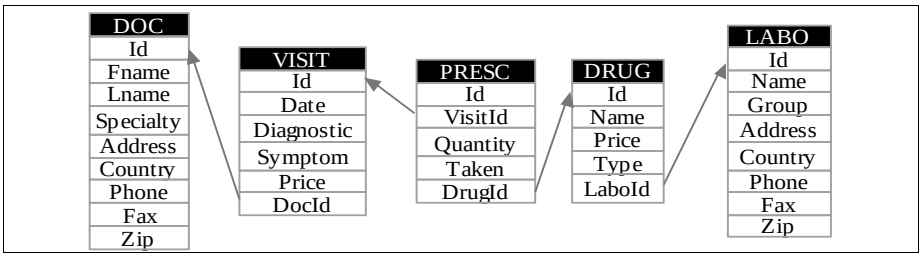
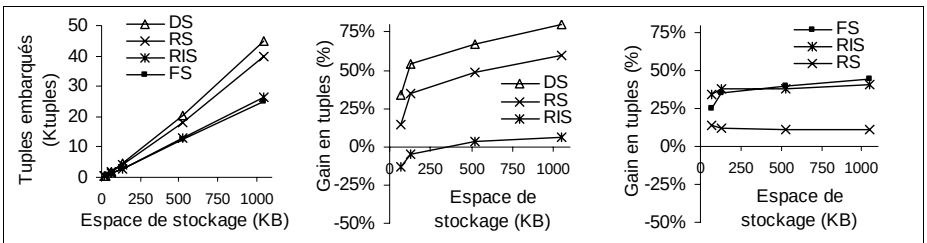


Figure 7. *Schéma de la base de données.*

4.3. Empreinte de la base de données

Cette section évalue la compacité de la base obtenue avec les quatre modèles de stockage et d'indexation candidats (stockage à plat FS, en domaine DS, en anneau RS, et en anneau avec pointeur inverse RIS décrits Section 3.1). L'évaluation ne peut être menée en utilisant un jeu de données synthétique, incapable de retranscrire la distribution et les doublons présents dans les cas réels. Nous avons utilisé un jeu de données médicales fourni par des partenaires médicaux (cf. schéma Figure 7).

Le jeu de données sur lequel sont basées les expérimentations de la section 4.3 contient 1276 médecins, 6442 visites, 32210 prescriptions, 863 médicaments et 48 laboratoires. Un sous-ensemble de ce jeu de données a été utilisé dans les mesures de la Figure 8. La taille moyenne des n-uplets considérés, stockés à plat, est respectivement de 75, 53, 23, 32, 68 octets pour les tables DOC, VISIT, PRESC, DRUG, LABO. La Figure 8.a présente la capacité de stockage de chaque modèle, et les Figures 8.b et 8.c traduisent les rapports de capacités entre ces modèles. RIS, très proche de FS, est beaucoup moins compact que DS et RS. Ainsi, RIS doit être adopté seulement si le gain obtenu sur l'exécution de requêtes est très significatif. DS et RS présentent des résultats proches, ce qui souligne l'extrême compacité des index en anneau. En outre, plus le nombre de tuples embarqués est important, moins le coût de l'indexation est significatif. Cette remarque indique que pour des cartes de grande capacité, le coût des index est moindre. On remarque aussi que le nombre total de tuples embarqués ne dépasse pas 50.000 pour une carte de capacité de stockage de 1 MB. Nous n'examinons donc pas dans la suite les requêtes traitant plus de données.



(a) Tuples embarqués. *(b) Gain par rapport à FS.* *(c) Perte par rapport à DS.*

Figure 8. *Capacité de Stockage de PicoDBMS.*

4.4. Performances de PicoDBMS

Cette section donne les performances du PicoDBMS en mise à jour (insertion de tuples) et en interrogation (évaluation de vues). Deux plates-formes fournies par Axalto ont servies aux expérimentations. Nous présentons ici ces deux plates-formes, les données et requêtes considérées dans notre banc d'essai, ainsi que les résultats.

4.4.1. Plate-forme d'expérimentation

Le prototype de carte réelle utilisé est doté d'un processeur 32 bit cadencé à 50 MHz, de 64 KB d'EEPROM, 96 KB de ROM et 4 KB de RAM (quelques centaines d'octets restant libres à l'application). Un timer interne mesure le temps de réponse à chaque APDU[3] entrante. Le code du PicoDBMS étant stocké dans l'EEPROM du prototype, l'espace restant pour les données est réduit à 25 KB.

Pour apprécier les performances sur de plus gros volumes de données, nous utilisons la plate-forme de simulation présentée Figure 9. Elle est constituée d'un simulateur matériel de carte à puce connecté à un PC de contrôle, et relié à un PC client connecté à un lecteur de carte standard (norme ISO 7816). Ce simulateur matériel donne le nombre exact de cycles processeur effectués entre deux points d'arrêts du code embarqué. Nous avons vérifié la précision du simulateur en confrontant les résultats sur de petites bases à ceux de la carte réelle.

Une version modifiée du système d'exploitation ZePlatform tourne à la fois sur le prototype de carte réelle et sur le simulateur. En effet, grâce à nos expérimentations, Axalto a modifié le noyau de ZePlateform pour améliorer le support d'applications orientées données (cas d'application inhabituel sur une puce). Entre autre, les accès directs à la mémoire persistante ont été optimisés. Les modifications de ZePlateform et le portage de PicoDBMS en C ont abouti à gagner 2 ordres de grandeur par rapport aux performances du prototype démontré à VLDB'01 [Anciaux *et al.*, 2001].

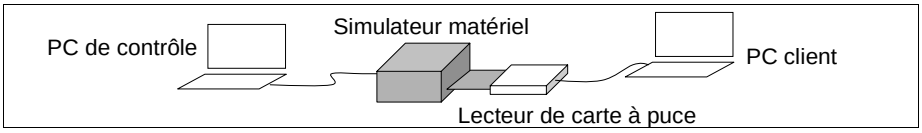


Figure 9. *Plate-forme de simulation de carte à puce*

4.4.2. Jeu de données et jeu de requêtes

Nous évaluons les performances de PicoDBMS sur un schéma synthétique (différent de celui utilisé pour évaluer la compacité) pour contrôler la sélectivité de chaque attribut (Figure 10). Les ratios de cardinalités des relations sont proches des bancs d'essais TPC-R/H/W : une table (R0) référence deux tables (R1 et R2) 6 fois plus petites, chacune référençant une autre table (resp. R3, R4) 6 fois moins peuplée.

3. L'APDU (Application Protocol Data Unit) est l'unité de communication entre la carte et le lecteur définie par la norme ISO 7816. Le coût de communication entre la carte et le lecteur n'est pas pris en compte dans les mesures, ce coût n'étant pas une limitation à long terme.

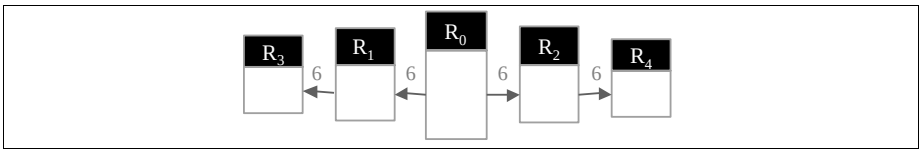


Figure 10. *Schéma de base de données synthétique.*

La Table 2 détaille le contenu de la table R0. Les autres tables sont basées sur le même modèle, mais ne contiennent qu'une seule clé étrangère (R1 et R2) ou aucune (R3 et R4). Notons que l'attribut A0 est stocké en FS car il est unique (ne peut profiter de DS, RS ou RIS). A3 et A4 ne sont pas stockés en DS, les valeurs étant plus petites que la taille d'un pointeur.

Nom des attributs	*Contrainte*	*Type du contenu*	*Type de stockage*	*Valeurs distinctes*	*Taille (Bytes)*
A_0	clé primaire	numérique	FS	$\|R_0\|$	4
A_1, A_2	clés étrangères	numérique	FS/DS/RS/RIS	$\|R_0\|/6$	4
A_3, A_4		numérique	FS/RS/RIS	$\|R_0\|/100$	4
A_5, A_6, A_7, A_8, A_9		alpha- numérique	FS/DS/RS/RIS	$\|R_0\|/10$	20 (moyenne)

Table 2. *Schéma de la Relation R_0.*

La Table 3 présente le jeu de requêtes utilisé. Ces requêtes sont représentatives des droits d'accès introduits dans la Section 2. Les attributs de projection, sélection, jointure et groupement ne sont pas spécifiés car chaque requête est évaluée sur différents attributs. Ceci nous permet de présenter le meilleur (appelé *bon* sur les courbes) et le *pire* des cas lorsque cela se justifie. Les courbes présentées dans la suite montrent les performances pour différentes valeurs de sélectivité de sélection et de jointure, et pour différents taux de groupement. Lorsque ces valeurs ne sont pas spécifiées, c'est que nous utilisons les valeurs par défaut suivantes : le nombre d'attributs projetés est fixé par défaut à 2 attributs, la sélectivité par défaut de la restriction est fixée à 1 %, la jointure par défaut est l'equi-jointure sur clé (la cardinalité du résultat est égale à celle de la relation référençant) et le taux de groupement par défaut est fixé à 1 % (chaque groupe agrége 100 tuples de l'entrée).

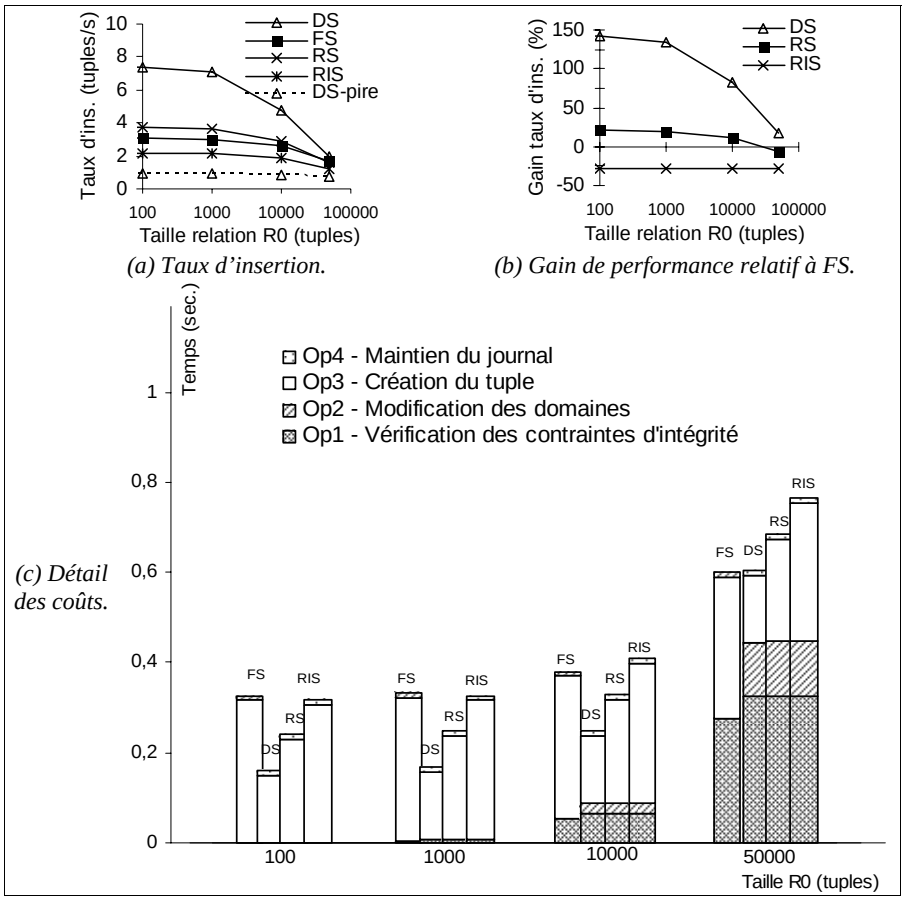
Acronyme	*Description du droit d'accès*	*Requête représentative correspondante*
P	Sous-ensemble des attributs	Projection de R_0
SP	Prédicats sur une seule relation	Sélection de R_0
SPJ	Prédicats sur 2 relations (association directe)	Sélection de R_0, jointure de R_0 et R_1
SPJ^n	Prédicats sur n relations (ass. transitive)	Multi-sélections, jointure R_0-R_1-R_2-R_3-R_4
SPG	Groupement et calcul d'agrégat mono-attr.	Grp^t mono-attribut de R_0, calcul d'agrégat
SPJG	Plusieurs relations, groupement et calcul d'agrégat mono-attribut	Jointure de R_0, R_1, R_3, groupement mono-attribut, un calcul d'agrégat
$SPJG^n$	Groupement et calcul d'agrégat multi-attributs	Jointure de R_0, R_1, R_3, grp^t sur 2 attributs de 2 relations distinctes, calcul d'agrégat

Table 3. *Jeu de requêtes des évaluations de performance.*

4.4.3. Insertion de tuples

La caractéristique C2 fait de l'insertion de données un aspect important de l'évaluation. Dans les mesures, les tuples sont insérés dans la relation R0. En effet, la plupart des insertions sont faites dans R0 vu que c'est la relation du schéma ayant la plus grande cardinalité. De plus, les insertions dans cette relation représentent le pire des cas en terme de performance vu qu'elles impliquent le plus grand coût de contrôle d'intégrité : deux contraintes d'intégrité référentielle doivent être vérifiées et le coût de la contrainte d'unicité augmente avec la cardinalité de la relation. Plus précisément, l'insertion met en jeu quatre opérations principales :

(Op1) Vérification des contraintes d'intégrité : la relation cible (ici R0) doit être parcourue complètement pour vérifier la contrainte d'unicité, et chaque relation référencée doit être explorée (ici parcours moyen de 1/6 de la relation) pour vérifier chaque contrainte d'intégrité référentielle.



(a) Taux d'insertion.

(b) Gain de performance relatif à FS.

(c) Détail des coûts.

Figure 11. *Performances d'insertion.*

(Op2) Modification des domaines : pour RS, DS et RIS, la valeur de chaque attribut inséré doit être retrouvée dans le domaine correspondant et ajoutée le cas échéant. La Figure 11 suppose que les valeurs sont déjà présentes (cas le plus fréquent).

(Op3) Création du tuple : une zone est allouée pour le nouveau tuple dans la relation cible, et les valeurs et pointeurs sont écrits dans la zone.

(Op4) Maintien du journal : cette opération correspond au coût de la journalisation basée sur des pointeurs comme nous l'avons présentée Section 3.3.

La Figure 11.a montre que le taux de transmission est convenable pour chaque modèle de stockage. Sur la Figure 11.c, le bénéfice de DS et RS par rapport à FS décroît avec la cardinalité de la relation. Bien que la compacité du modèle de stockage réduise le coût des écritures en mémoire stable (règle d'écriture), la cardinalité de la relation cible a un effet négatif sur le coût de Op1 et Op2. Notons que l'opération Op1 pourrait être accélérée (arbre-B sur les clés primaires), mais ce gain doit être pondéré par le coût de maintien de l'index (écritures en EEPROM) et le surcoût en espace de stockage. Vu que le débit transactionnel n'est pas significatif, la conclusion principale est délivrée par la Figure 11.a. Ainsi, chaque modèle de stockage satisfait le taux d'insertion requis, rendant tout effort d'optimisation inutile.

4.4.4. Projection

La Figure 12.a présente le taux de transmission des requêtes de projection, représentatives du droit d'accès de type P. Les courbes correspondent à chaque modèle de stockage candidat en fonction du nombre d'attributs projetés. Les conclusions suivantes peuvent être tirées :

Le taux de transmission est indépendant de la cardinalité de la relation : en effet, le coût de projection est constant pour chaque tuple de l'entrée.

RS ralentit la projection : RS induit en moyenne la traversée de la moitié de l'anneau pour atteindre la valeur d'un attribut. Avec notre jeu de données, la longueur moyenne d'un anneau est de 10 pointeurs, ce qui conduit à une dégradation des performance d'un facteur 5 (remarquons l'échelle logarithmique de l'ordonnée des courbes de la Figure 12.)

La projection ne représente pas un frein aux performances : le débit est supérieur à 5000 tuples par seconde quelque soit le modèle de stockage considéré.

4.4.5. Sélection

Les Figures 12.b et 12.c présentent le taux de transmission des requêtes de sélection mono-table, représentatives du droit d'accès de type SP. Le taux de transmission est représenté en fonction de la sélectivité de la requête (Figure 12.b) et en fonction de la complexité (nombre d'attributs impliqués dans le prédicat) de la sélection (Figure 12.c). Ces courbes conduisent aux remarques suivantes :

Le taux de transmission est indépendant de la cardinalité de la relation : comme l'opération de projection, la sélection est effectuée à coût constant pour chaque résultat produit.

Le taux de transmission profite d'une faible sélectivité : en effet, plus la sélectivité est élevée, plus le nombre de tuples parcourus et testés avant la qualification d'un résultat est faible.

Les modèles RIS et RS se comportent mieux que DS et FS : RIS/RS offre un index de sélection naturel. Le gain en performance est déterminé par le ratio entre les cardinalité des domaines et des relations (fixé ici à 10). Lorsque la sélection est multi-attributs, l'écart de performance entre les modèles indexés et non indexés diminue, soulignant le fait que le processus d'exécution de PicoDBMS ne peut profiter que d'un seul index de sélection pour une requête donnée.

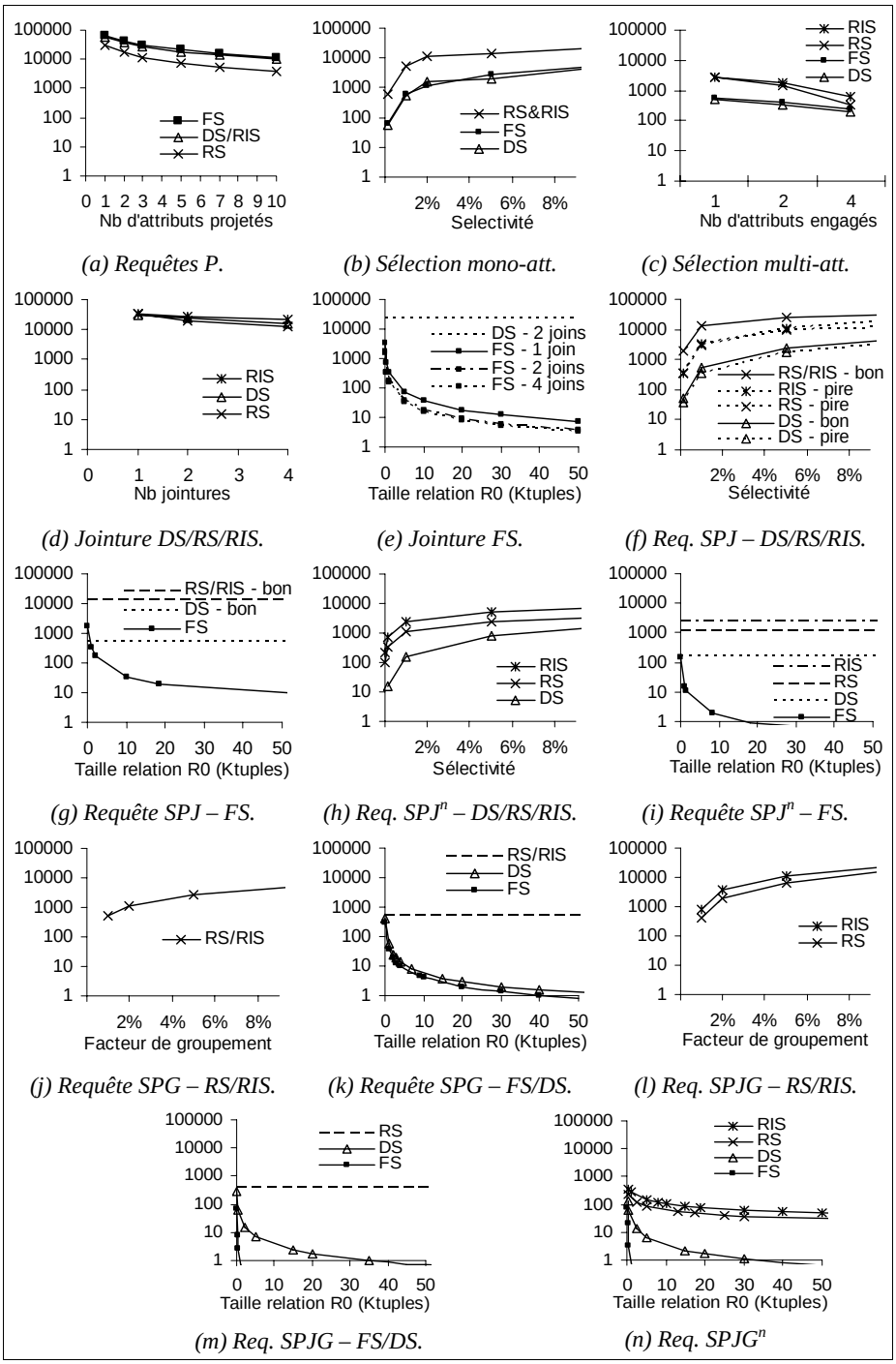
4.4.6. Jointure

Les Figures 12.d et 12.e exposent les performances de jointure, en fonction du nombre de jointures effectuées. Les Figures 12.f à 12.i présentent des requêtes représentatives des droits d'accès de type SPJ et SPJ^n, impliquant sélections et jointures. Ces deux séries de mesures sont nécessaires pour saisir l'incidence de la sélection sur les requêtes de jointures. Considérons d'abord les enseignements tirés des Figures 12.d et 12.e :

Taux de transmission élevé pour RIS, RS et DS : grâce à leur nature indexée, ces modèles présentent des débits de jointure élevés, quelque soit la taille de la base et le nombre de jointures. DS implémente un index de jointure unidirectionnel entre un attribut clé étrangère et la clé primaire correspondante (voir Section 3.1). RS et RIS rendent cet index bidirectionnel. L'ordre de jointure optimal privilégie la traversée des index de la clé étrangère vers la clé primaire, permettant de produire chaque tuple résultant de la jointure de N relations en traversant N-1 pointeurs de domaine, d'où l'écart faible entre RIS/RS et DS sur la Figure 12.d. Cependant, ce cas favorable ne peut pas toujours être atteint, notamment lorsque d'autres sélections forcent à adopter un autre ordonnancement des jointures. FS présente des performances beaucoup plus faibles et s'adapte mal lorsque la taille de la base augmente (Figure 12.e).

Considérons les Figures 12.f à 12.i concernant les requêtes SPJ et SPJ^n. Le taux de transmission est présenté en fonction de la sélectivité de la requête pour RIS, RS et DS (modèles établis indépendants de la cardinalité de la base), et en fonction de la cardinalité de la base pour FS (avec une sélectivité par défaut de 1%). Le plan d'exécution de la requête peut favoriser différents modèles selon l'ordonnancement des jointures, représentés sur les figures par les taux de transmission minimum et maximum pour chaque modèle (lorsque l'écart existe).

Les résultats sont clairs, mais méritent tout de même les deux remarques suivantes. D'abord, la performance de la jointure augmente lorsque la sélectivité des sélections décroît. Comme pour les requêtes de sélection, plus la sélectivité est élevée, plus le nombre de tuples parcourus et testés pour rechercher un tuple qualifié est faible. D'autre part, RS et RIS sont nettement plus efficaces que DS en présence de sélection. Comme nous l'avons mentionné plus haut, DS impose un ordre unique des jointures dans le plan d'exécution de la requête, empêchant l'utilisation d'index de sélection. Ainsi, le choix de RS ou RIS pour stocker les clés étrangères est judicieux.



(a) Requêtes P. *(b) Sélection mono-att.* *(c) Sélection multi-att.*

(d) Jointure DS/RS/RIS. *(e) Jointure FS.* *(f) Req. SPJ – DS/RS/RIS.*

(g) Requête SPJ – FS. *(h) Req. SPJ^n – DS/RS/RIS.* *(i) Requête SPJ^n – FS.*

(j) Requête SPG – RS/RIS. *(k) Requête SPG – FS/DS.* *(l) Req. SPJG – RS/RIS.*

(m) Req. SPJG – FS/DS. *(n) Req. SPJG^n*

Figure 12. *Performances d'interrogation en taux de transmission (tuples/s, Log.).*

4.4.7. Groupement et agrégation

Les calculs d'agrégats implantent les autorisations de type CA, donnant accès à des valeurs calculées sans autoriser les occurrences prenant part au calcul. Les Figures 12.j à 12.n présentent les performances des requêtes correspondant aux droits d'accès SPG, SPJG et SPJG^n et conduisent aux remarques suivantes :

DS et FS supportent mal l'agrégation : même pour l'agrégation mono-attribut, le taux de transmission est faible et dépend fortement de la taille de la base (Figures 12.k et 12.m). En effet, DS introduit un produit cartésien entre le domaine sur lequel porte le groupement et la relation, et FS nécessite des parcours successifs de la relation pour produire le résultat. Lorsque la requête fait intervenir des jointures, les performances de DS restent stables (grâce à un débit de jointure élevé), mais s'effondre avec FS (moins d'un tuple par seconde).

RIS et RS supportent aisément l'agrégation mono-attribut : grâce aux anneaux, la présence de jointures dans la requête a une faible influence sur les performances (Figures 12.j et 12.l). Bien que la performance soit indépendante de la taille de la base (comme pour les requêtes de jointure), le facteur de groupement a une influence importante. Par exemple, un facteur de groupement de 10% réduit la performance d'un facteur 10 par rapport à une requête sans agrégation.

RIS et RS supportent correctement l'agrégation multi-attributs : le taux de transmission dépend de la cardinalité de la base quelque soit le modèle de stockage (Figure 12.n). En effet, seul un attribut impliqué dans la clause *GROUP BY* peut bénéficier du stockage en anneau. Les performances restent cependant acceptables avec encore 80 tuples résultats produits chaque seconde.

4.4.8. Conclusion sur les taux de transfert

Le taux de transmission atteint pour chaque classe de requête est fortement dépendant du modèle de stockage choisi. Pour aider à comparer ces modèles, la Table 4 exprime leur performance relative en terme de rapports pour chaque classe de requête. Pour chaque colonne de la table, la cellule grisée sert de référence à l'établissement des rapports. Les taux de transmission ont été calculés en fixant la cardinalité de la relation R_0 à 1000.

DS donne le meilleur taux de transmission en insertion et est meilleur que FS lorsque des jointures interviennent dans la requête (1 à 3 ordres de grandeur). L'ajout d'anneaux apporte un bénéfice majeur pour les évaluations de jointures et les calculs d'agrégats (1 à 2 ordres de grandeur par rapport à DS). Cependant, le gain obtenu grâce à RIS est relativement décevant par rapport à RS (au plus un facteur 2 lorsque

Type d'opération / *Modèle de stockage*	*Insertion*	*Interrogation*						
		P	SP	SPJ	SPJ^n	SPG	SPJG	SPJG^n
FS	1	1	1	0,08	0,01	0,7	0,002	0,002
DS	4,8	0,9	0,9	1	1	1	1	1
RS	1,1	0,5	8,7	23,4	7,1	136	118	19
RIS	0,7	0,9	8,7	23,4	15,2	136	228	31

Table 4. *Rapports des taux de transmission.*

plusieurs jointures sont impliquées), surtout lorsque l'on considère la perte qu'il engendre en terme de compacité de la base. Néanmoins, on peut noter que pour des schémas de bases de données particuliers, comme le schéma en étoile, où une grande relation centrale référence de nombreuses petites relations par l'intermédiaire de grands anneaux de pointeurs, le modèle de stockage RIS peut devenir intéressant.

4.5. Conclusion et étude de cas

Cette section conclut les mesures en mettant en pratique les évaluations de performances pour sélectionner le modèle de stockage le plus adapté à une application donnée. Nous considérons ici un profil d'application s'intéressant au temps d'exécution total des requêtes, cas fréquent lorsque les tuples résultats ne peuvent être délivrés en pipeline pur (par exemple, lorsque qu'une clause *ORDER BY* est appliquée au résultat final). Nous supposons que le temps de réponse requis dans ce profil d'application est d'une seconde. D'autres profils d'applications sont envisagés dans [Anciaux, 2004]. Nous nous concentrons sur les requêtes d'interrogation vu que les insertions sont acceptables (cf. Section 4.4.3) quelque soit le type de stockage.

La Table 5 présente le pourcentage du résultat total produit en une seconde pour chaque classe de requête, selon la cardinalité de la base et le modèle de stockage sélectionné. Notons que la cardinalité de la base est elle-même déterminée par la capacité de stockage de la carte à puce et par le modèle de stockage choisi. Par exemple, la cardinalité potentielle de la base pour une carte d'1 MB est de 24 Ktuples (avec FS) à 44 Ktuples (avec DS). Les cellules répondant au besoin de l'application sont grisées. La Table 5 montre que FS ne peut satisfaire aux besoins de l'application pour les requêtes d'agrégat, même dans de petite cartes disposant de 64 KB (seul 11 % du résultat est délivré dans la seconde). Alors que DS satisfait les besoins pour des cartes de 64 KB, RS et RIS sont nécessaires pour des cartes plus grosses. De plus, la différence entre RS et RIS n'est pas significative.

Capacité de la puce	*Tuples stockés*	*FS*						*Tuples stockés*	*DS*					
		SP	*SPJ*	*SPJ^n*	*SPG*	*SPJG*	*SPJG^n*		*SP*	*SPJ*	*SPJ^n*	*SPG*	*SPJG*	*SPJG^n*
64KB	1428	100	100	100	100	**11**	**11**	1906	100	100	100	100	100	100
128KB	2978	100	100	**39**	**95**	**0**	**0**	4593	100	100	100	**59**	**61**	**52**
512KB	12293	100	**40**	**1**	**5**	**0**	**0**	20590	100	100	100	**2**	**2**	**2**
1MB	24864	100	**9**	**0**	**1**	**0**	**0**	44730	100	100	**48**	**1**	**0**	**0**
		RS							*RIS*					
64KB	1637	100	100	100	100	100	100	1247	100	100	100	100	100	100
128KB	4015	100	100	100	100	100	100	2829	100	100	100	100	100	100
512KB	18250	100	100	100	100	100	**43**	12736	100	100	100	100	100	100
1MB	39650	100	100	100	100	100	**13**	26426	100	100	100	100	100	**41**

Table 5. *Pourcentage de tuples du résultat transféré en 1 seconde.*

A la lumière de ces résultats, nous pouvons tirer les conclusions globales suivantes. Les requêtes de sélection et de projection ne posent jamais de problème, quelque soit le modèle de stockage considéré. La difficulté principale est donc de supporter efficacement les jointures et agrégations. Pour cela, RS est généralement requis, sauf pour des cartes de faible capacité où DS peut être acceptable. La question finale concerne la généralité de ces conclusions. En d'autres termes, pouvons nous imaginer d'autres modèles de stockage et d'indexation ? Nous avons la conviction que cette étude couvre les principaux compromis en terme de techniques de stockage et d'indexation, à savoir le mode de stockage brut (plat) vis à vis du modèle compressé (domaine) et le modèle non indexé vis à vis des structures accélérant les sélections et les jointures. Bien sûr, différentes variations peuvent être envisagées, mais les écarts de performance entre nos techniques et leurs éventuelles variantes restent faibles dans un contexte proche de celui des bases grande mémoire. La différence de performance entre les modèles RS et RIS n'est que la confirmation de cette allégation. Au mieux, des structures d'indexation multi-attributs pourraient être conçues pour accélérer les requêtes d'agrégation multi-attributs, ce qui constitue une forme particulière de requêtes pré-calculées.

5. Perspectives de recherches

De nombreux problèmes de recherche s'inscrivent dans la suite de PicoDBMS. Sans prétendre à l'exhaustivité, cette section introduit des perspectives intéressantes de cette étude et, plus généralement, du contexte bases de données embarquées.

Utilisation de mémoire externe : l'une des limitations principales empêchant d'élargir le champ d'application des cartes à puce est leur faible capacité en terme de mémoire de stockage persistant. De nombreux fabricants poussent vers de nouvelles architectures combinant une puce sécurisée à de gros modules de mémoire stable (et non sécurisés) de type FLASH. Par exemple, la ‘‘carte à puce’’ SUMO fabriquée par Gemplus [Gemplus, 2002] est dotée de 224 MB de mémoire FLASH intégrée dans le support plastique de la carte. La carte X-Mobile [Renesas Tec., 2004] combine sous forme de MMC (Mass Memory Card) une puce sécurisée et une grosse mémoire FLASH. Ces recherches sont conduites dans le cadre du consortium Mobile PASSport [MOPASS]. A très court terme, plus de 8 GB de mémoire sont attendus pour ce type d'architecture. Dans ce contexte, le problème est d'utiliser cette mémoire externe tout en conservant la sécurité offerte par la puce. La mémoire externe étant non sécurisée, quatre catégories d'attaques doivent être considérées : l'examen des données (lecture directe ou inférence d'information sensible) ; la détérioration (modification de données non autorisées) ; la substitution (des blocs de données valides sont remplacés par d'autres blocs de données valides) ; et le rejeu (des blocs de données valides sont remplacés par une ancienne version de ces blocs). Le porteur du dispositif peut être susceptible d'attaquer ses propres données (e.g., pour rejouer le remboursement d'une consultation grâce à son dossier médical). Pour éviter de divulguer des données sensibles et pour protéger leur intégrité, des techniques cryptographiques (chiffrement, hachage sécurisé, etc.) doivent être utilisées. Certaines méta-données (clés de chiffrement, valeurs de hachage, numéros

de versions, secteur de démarrage, etc.) ainsi que l'évaluateur de requête doivent être embarqués dans la puce sécurisée pour délivrer le sous ensemble autorisé des données à l'utilisateur. Le problème est donc de combiner techniques cryptographiques et exécution de requêtes de manière à satisfaire les trois objectifs (souvent contradictoires) que sont la performance, la sécurité et la compatibilité avec les fortes contraintes matérielles de la puce. Dans [Anciaux, 2004], nous conduisons une étude préliminaire de ce problème dans le cadre d'une puce connectée à une grande quantité de mémoire FLASH.

Environnement sans contact : les constructeurs et les organisations gouvernementales [SINCE] portent un intérêt grandissant aux puces sans contact lié à leur facilité d'utilisation. Bien que PicoDBMS puisse être intégré tel quel dans une carte sans contact, l'environnement diffère par de multiples aspects, ce qui peut conduire à une conception différente. Par exemple, le temps d'exposition au lecteur est extrêmement court dans un environnement sans contact. La remise en cause de l'objectif en terme de temps de réponse (considéré jusqu'alors comme étant de l'ordre de la seconde), impose de nouvelles techniques d'exécution. Ceci peut conduire à d'intéressantes problématiques visant à exécuter des requêtes très rapidement, quitte à fournir un résultat approché (évitant au porteur de stationner devant le lecteur).

Utilisation de nouvelles technologies de mémoire persistante : nous nous sommes concentrés sur la technologie EEPROM lors de l'étude de PicoDBMS. Cependant, l'EEPROM ayant atteint sa limite de passage à l'échelle d'après certains constructeurs, de nouvelles technologies comme la FLASH font leur apparition dans les puces. La mémoire de type FLASH présente ses propres caractéristiques qui doivent être prises en compte dans le design des composants embarqués. La perspective intéressante consistant à adapter les composants embarqués à la mémoire FLASH a fait l'objet d'une première étude [Bolchini *et al.*, 2003] s'intéressant exclusivement aux écritures sur un petit volume de données (centaines de tuples). Une étude considérant de plus larges volumes et orientée vers l'évaluation de droits d'accès complexes serait utile. De même, les technologies alternatives à long terme (PCM, OUM ou Millipèdes), pourraient être envisagées. Ces mémoires exhibent aussi des spécificités ayant un impact direct sur le modèle de stockage et d'indexation des données. Des travaux existent dans un contexte plus général sur les mémoires Millipèdes [Yu *et al.*, 2003]. La combinaison de ces propriétés mémoires avec les autres contraintes des puces constitue clairement un problème intéressant.

Calibrage des ressources : les ressources matérielles embarquées ont un impact immédiat sur le coût de la puce, surtout lorsque celle-ci concerne un marché de masse. Il est donc particulièrement important de calibrer au plus juste les ressources matérielles à intégrer. Nous avons mené dans ce contexte une première étude du calibrage de la RAM [Anciaux *et al.*, 2003]. En effet, la RAM présente une cellule de très faible densité, occupant en moyenne le tiers de la puce, et représentant le tiers de son prix (le coût d'une puce est directement lié à sa taille). Cette ressource est donc cruciale. Les recherches futures menées sur ce point devraient viser à calibrer la puce selon ses trois dimensions principales (processeur, RAM, mémoire persistante) selon le besoin de l'application. La consommation processeur peut être

minimisée par l'utilisation intensive d'index et la matérialisation, et la quantité de mémoire persistante peut être modulée par compression des données de base et des structures accélératrices. Bien sûr, les différentes dimensions du problème ne sont pas indépendantes. Tous cela fait du co-design complet de la puce face au besoin de l'application un défi particulièrement intéressant.

Traitements de données XML : l'étude présentée ici est conduite sur le modèle relationnel. XML devenant un standard de facto pour décrire des données hétérogènes et les échanger entre les applications, gérer ce type de données dans la puce fait sens. Par exemple, XML peut être un modèle de données approprié pour gérer des profiles personnels partagés par de nombreuses applications (comme dans le cas de l'environnement personnel virtuel). Des problèmes nouveaux sont générés par un modèle de données XML. D'abord, bien que de nombreuses études propose des techniques de stockage et d'indexation de données XML, ces propositions (à notre connaissance) ne sont pas dédiées aux puces sécurisées. La complexité vient uniquement de la nature semi structurée et hiérarchique des données. D'autre part, les modèles de contrôle d'accès aux documents XML sont basés sur des règles et non sur des vues. Ainsi, l'adaptation à XML nécessite d'implanter des politiques de contrôle d'accès basées sur des règles, compatibles avec les contraintes matérielles de la puce. Une contribution dans ce sens propose un moteur de contrôle d'accès sur un flux de données XML embarqué dans une carte à puce [Bouganim et al. 2004]. Enfin, les modèles de droits digitaux [XrML, MPEG-REL 2004, ODRL], exprimant des conditions complexes sur des méta-données XML, introduisent des problèmes liés à la garantie des politiques d'accès sur des flux de données (multimédia).

6. Conclusion

Quatre ans après la publication du design initial de PicoDBMS, les objectifs de ce papier sont de répondre aux trois questions importantes posées en introduction.

La première question s'interroge sur les éventuelles modifications de la problématique initiale à la lumière des évolutions matérielles et applicatives. Nous avons montré dans la section 2.1 que bien que la carte à puce n'échappe pas à la loi de Moore, l'asymétrie des ressources caractérisant les puces (à la base du design de PicoDBMS) n'est pas remise en question pour des raisons technologiques et économiques. En outre, l'introduction de puces sécurisées (ou assimilés) au sein de l'infrastructure informatique élargit le domaine d'application de PicoDBMS à la gestion de dossiers portables et sécurisés complexes, à des modèles avancés garantissant des droits digitaux, ainsi qu'aux dispositifs Hippocratiques. La première contribution de ce papier a été de revoir la position du problème en intégrant cette évolution, repositionnant PicoDBMS sur la gestion des droits d'accès.

La deuxième question était liée à la faisabilité de l'approche. La technologie carte à puce disponible en 2001 (à la fois matérielle et logicielle) n'était clairement pas adaptée aux applications embarquées orientées données [Anciaux et al., 2001]. Trois ans d'efforts conjoints de la part de notre équipe (réécriture optimisée de PicoDBMS en C) et de notre partenaire industriel Axalto (nouvelle plate-forme matérielle,

adaptation du système d'exploitation) ont été nécessaires pour obtenir un prototype convaincant. Un banc d'essai dédié aux bases de données de type PicoDBMS a été défini et utilisé pour déterminer les performances relatives des modèles de stockage et d'indexation candidats, à la fois sur une carte réelle et sur un simulateur matériel. Cette étude de performance permet de sélectionner le modèle de stockage et d'indexation approprié pour une application donnée et un certain volume de données embarquées. Elle constitue la seconde contribution de cet article.

La troisième question concernait les perspectives de recherche ouvertes par PicoDBMS. Indubitablement, la gestion de données relationnelles sur carte à puce est maintenant bien comprise. Cependant, les puces sécurisées envahissent notre vie quotidienne sous de nombreuses apparences : cartes sans contact, clés de sécurité USB, cartes de stockage de masse (PCMCIA), objets intelligents sécurisés, etc. La technologie matérielle évolue parallèlement : large volume de stockage non sécurisé, débit de communication élevé, futures (long terme) technologies de mémoire persistante, etc. De nouvelles techniques de gestion de données embarquées sont nécessaires pour s'adapter à ces évolutions : support d'un grand volume de données, protection des données contre de nouveaux types d'attaques et tirant avantage du débit de communication élevé pour externaliser des données et/ou des traitements. La co-conception est bien sûr un problème très important dans ce contexte. Enfin, on peut même imaginer que des puces sécurisées soient intégrées dans des infrastructures non sûres (par exemple un serveur bases de données) et constituer le tiers de confiance ultime sur lequel résiderait la sécurité. En conclusion, nous espérons que cet article contribue à dessiner les prémisses d'un axe de recherche prometteur autour des techniques de gestion de données sur puces sécurisées.

7. Bibliographie

Abdallah M., Guerraoui R., Pucheral P., « Dictatorial Transaction Processing: Atomic Commitment Without Veto Right », *Int. Conf. on Distributed and Parallel Databases, 2002*.

Agrawal R., Kiernan J., Srikant R., Xu Y., « Hippocratic Databases », *Int. Conf. on Very Large Data Bases*, 2002.

Ammann A. C., Hanrahan M., Krishnamruthy R., « Design of a memory resident DBMS », *IEEE Int. Conf. Compcon*, 1985.

Anciaux N., Bobineau C., Bouganim L., Pucheral P., « PicoDBMS: Validation and Experience », *Int. Conf. on Very Large Data Bases*, demo session, 2001.

Anciaux N., Bouganim L., Pucheral P., « Database Components on Chip », *ERCIM news*, 2003.

Anciaux N., Bouganim L., Pucheral P., « Memory Requirements for Query Execution in Highly Constrained Devices », *Int. Conf. on Very Large Data Bases*, 2003.

Anciaux N., Database Systems on Chips, Thèse de doctorat, Université de Versailles, 2004.

APEC, Telecommunications and Information Working Group, Policy and Regulatory Update of Hong Kong, China, 2003. http://unpan1.un.org/intradoc/groups/public/documents/apcity/unpan008982.pdf

Bobineau C., Bouganim L., Pucheral P., Valduriez P., « PicoDBMS: Scaling down Database Techniques for the Smart card », *Int. Conf. on Very Large Data Bases*, 2000. Récompensé par le « Best Paper Award ».

Bolchini C., Salice F., Schreiber F., Tanca L., « Logical and Physical Design Issues for Smart Card Databases », *ACM Transactions on Information Systems*, 2003.

Bonnet P., Seshadri P., « Device Database Systems », *Int. Conf. on Data Engineering*, 2000.

Bonnet P., Gehrke J., Seshadri P., « Towards Sensor Database Systems », *Int. Conf. on Mobile Data Management*, 2001.

Bouganim L., Dieu N., Pucheral P. « MobiDiQ: Mobile Digital Quietude », Gold Award of the *Simagine International Contest* organized by Sun, Axalto and Samsung, 2005.

CardTechnology, Omaha Hospitals To Accept Patient Smart Card, Press release, 2003. http://www.cardtechnology.com/cgibin/readstory.pl?story=20031104CTDN157.xml

Chaudhuri S., Weikum G., « Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System », *Int. Conf. on Very Large Data Bases*, 2000.

Chen Z., *Java Card Technology for Smart Cards: Architecture and Programmer's guide*, Addison-Wesley, 2000.

Gemplus, A 224MB microprocessor Smart Card, 2002. http://www.gemplus.com/smart/enews/st3/sumo.html

Graefe G., « Query Evaluation Techniques for Large Databases », *ACM Computing Surveys*, 1993.

Gupta R., Dey S., Marwedel P., « Embedded System Design and Validation: Building Systems from IC cores to Chips », *International Conference on VLSI Design*, 1998.

Henderson N.J., White N.M., Hartel P.H., « iButton Enrolment and Verification Requirments for the Pressure Sequence Smart Card Biometric », *E-Smart Conf.*, 2001.

Heytens M., Listgarten S., Neimat M., Wilkinson K., Smallbase: A Main-Memory DBMS for High-Performance Applications, HP Lab., 1994.

IBM Corp., DB2 Everywhere – Administration and Application Programming Guide, IBM Software Documentation SC26-9675-00, 1999.

IBM Harris, Multinational Consumer Privacy Survey, 2000. http://www.pco.org.hk/english/infocentre/files/westin.doc

IFPI. International. Federation of Phonographic Industry, consulté en juin 2005. http://www.ifpi.org/

Intanagonwiwat C., Govindan R., Estrin D., « Directed diffusion: A scalable and robust communication paradigm for sensor networks », *Int. Conf. on Mobile Computing and Networking*, 2000.

ISO, International Standardization Organization, Integrated Circuit(s) Cards with Contacts - Part 7, ISO/IEC 7816-7, 1999.

Karlsson J., Lal A., Leung C., Pham T., « IBM DB2 Everyplace: A Small Footprint Relational Database System », *Int. Conf. on Data Engineering*, 2001.

Kim W., « Smart Cards: Status, Issues, US Adoption », *Journal of Object Technology*, 2004.

Madden S., Franklin M.J., Hellerstein J., Hong W., « TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks », *Int. Conf. on Operating Systems Design and Implementation*, 2002.

Madden S., Hellerstein J.M., « Distributing Queries over Low-Power Wireless Sensor Networks », *ACM Int. Conf. on Management of Data*, 2002.

Madden S., Hellerstein J.M., Hong W., « TinyDB: In-Network Query Processing in TinyOS », Tutorial, *Int. Conf. on Data Engineering*, 2004.

Maloney D. L., « Card Technology in Healthcare », *Int. Conf. CardTech/SecurTech*, 2001.

Mastercard Inc., MasterCard Open Data Storage, 2002. http://www.mastercardintl.com/

MIPS Technologies Inc., Smart Cards: the computer in your wallet, White Paper 2002. http://www.mips.com/content/PressRoom/TechLibrary/techlibrary

Missikoff M., Scholl M., « Relational Queries in a Domain Based DBMS », *ACM Int. Conf. on Management of Data*, 1983.

MOPASS Consortium, consulté en juin 2005. http://www.mopass.info/english/

Moriyama S., Tanabe H., Sasaki S. « Toyomura S. Traceability and Identification Solutions for Secure and Comfortable Society », Hitachi Review, 2004.

MPEG-REL, MPEG-21 Right Expression Language (MPEG-REL), ISO/IEC 21000-5:2004 standard, 2004. http://www.contentguard.com/MPEGREL_home.asp

Netc@rd Project, consulté en juin 2005. http://www.netcards-project.com/new/

NETLINK 2000, consulté en juin 2005. http://www1.va.gov/card/docs/

NIST, U.S. Government Smart Card Interoperability Specification, Intern Report 6887, 2002.

NRC report, « Embedded Everywhere, A research agenda for networked systems of embedded computers », *National Academy Press*, 2001.

OCR HIPAA Privacy. « General Overview of Standards for Privacy of Individually Identifiable Health Information », 2003.

ODRL, The Open Digital Rights Language Initiative, consulté en juin 2005. http://odrl.net/.

Olson M.A., « Selecting and Implementing an Embedded Database System », *IEEE Computer Magazine*, 2000.

Olson M.A., Bostic K., Seltzer M.I., « Berkeley DB », *USENIX Conf.*, 1999.

Oracle Corp., Oracle 9i lite: Release Notes - Release 5.0.1, 2002.

Ortiz Jr. S., « Embedded Databases Come out of Hiding », *IEEE Computer Magazine*, 2000.

Paradinas P. Vandewalle J.-J., « A Personal and Portable Database Server: the CQL Card », In Application of Databases, volume 819 of LNCS, 1994.

Pervasive Software Inc., Pervasive.SQL v8, consulté en juin 2005. http//:www.pervasive.com.

Potonniée O., « A decentralized privacy-enabling TV personalization framework », *European Conference on Interactive Television: Enhancing the Experience (euroITV)*, 2004.

Pucheral P., Bouganim L., Valduriez P., Bobineau C., « PicoDBMS: Scaling down Database Techniques for the Smart card », *Very Large Data Bases Journal*, 2001.

Pucheral P., Thevenin J.-M., Valduriez P., « Efficient main memory data management using the DBGraph storage model », *Int. Conf. on VLDB*, 1990.

Renesas Tec, X-Mobile Card Overview, 2004, consulté en juin 2005. http://www.x-mobilecard.com/products/technology.jsp

Schneier B., Shostack A., « Breaking up is hard to do: Modeling Security Threats for Smart Cards », *USENIX Symposium on Smart Cards*, 1999.

Seshadri P., Garrett P., « SQLServer For Windows CE – A Database Engine for Mobile and Embedded Platforms », *Int. Conf. on Data Engineering*, 2000.

SINCE, Secure and Interoperable Networking for Contactless in Europe, Interoperable European Electronic ID / Public Service Cards, Project Deliverable, 2002.

Singhal V., Kakkad S., Wilson P., « Texas: An Efficient, Portable Persistent Store », *Int. Workshop on Persistent Object Systems*, 1992.

Smart Card Alliance, HIPAA Compliance and Smart Cards: Solutions to Privacy and Security Requirements, 2003.

Smart Card Alliance, Privacy and Secure Identification Systems: The Role of Smart Cards as a Privacy-Enabling Technology, 2003.

Smartright, The Smartright content protection system, consulté en juin 2005. http://www.smartright.org/

STMicroelectronics, ST's Trusted Platform Module Provides Complete TCG-Enabled Security Solution for Desktop and Laptop PCs, 2004.

Sybase Inc., The Next Generation Database for Embedded Systems, 2000.

TCPA, Trusted Computing Platform Alliance, consulté en juin 2005. http://www.trustedcomputing.org/

TPC, Transaction Processing Database Council, consulté en juin 2005. http//:www.tpc.org

US-GAO, United States General Accounting Office, Electronic Government: Progress in Promoting Adoption of Smart Card Technology, 2003.

Valduriez P., « Join Indices », *ACM Transactions on Database Systems*, 1987.

Veterans Affairs, G-8 Healthcare Data Card Project, 2001. http://www.va.gov/card/

Vingralek R., « Gnatdb: A small-footprint, secure database system », *Int. Conf. on Very Large Databases*, 2002

Vogt H., Rohs M., Kilian-Kehr R., *Middleware for Communications*, Chapter 16: Middleware for Smart Cards, John Wiley and Sons, 2003.

XrML, eXtendible rights Markup Language, consulté en juin 2005. http://www.xrml.org/.

Yao Y., Gehrke J., « The Cougar Approach to In-Network Query Processing in Sensor Networks », *SIGMOD Record*, 2002.

Yu H., Agrawal D., El Abbadi A., « Tabular Placement of Relational Data on MEMS-based Storage Devices », *Int. Conf. on Very Large Data Bases*, 2003.

Article reçu le 28 juillet 2005

Article accepté le 23 novembre 2006

Nicolas Anciaux *est chargé de recherche à l'INRIA Rocquencourt, au sein du projet SMIS (Secured and Mobile Information Systems). Il a travaillé en tant que post-doctorant à l'Université de Twente aux Pays-Bas, puis a rejoint l'INRIA en 2006. Ses travaux actuels portent sur la confidentialité des données et l'intimité des individus dans un environnement d'intelligence ambiante.*

Luc Bouganim *est Directeur de Recherche à l'INRIA Rocquencourt, responsable permanent du projet SMIS. Il a été Maître de Conférences de 1997 à 2002, année où il a rejoint l'INRIA. Ses travaux passés sont axés sur le moteur des Systèmes de Gestion de Bases de Données (SGBD), en particulier l'exécution et l'optimisation de requêtes. Depuis 2000, Luc est fortement engagé dans des travaux liés à la gestion de données en environnement ubiquitaire et à la confidentialité des données.*

Philippe Pucheral *est Professeur à l'Université de Versailles, actuellement en détachement à l'INRIA Rocquencourt où il dirige le projet SMIS. Son domaine de recherche couvre les aspects systèmes des bases de données (modèles de stockage et d'indexation, évaluation de requêtes, transactions), les composants base de données embarqués dans des puces et la protection matérielle des bases de données. Plus généralement, le projet SMIS a deux objectifs : i) concevoir des composants bases de donnée embarqués compatibles avec les contraintes matérielles spécifiques des calculateurs ultra-légers; ii) concevoir de nouvelles architectures préservant la confidentialité des données en combinant techniques cryptographiques et logiciels embarqués dans des puces sécurisées.*