n°d'ordre 1562

USTL

LIFL Laboratoire d'Informatique Fondamentale de Lille

Thèse

présentée à

L'Université des Sciences et Technologies de Lille

pour l'obtention du titre de

Docteur en Informatique

par

Patrick TRANE

Conception et réalisation d'un système de contrôle d'accès pour la carte à micro-processeur

Soutenue le 08 septembre 1995 devant le jury :

Georges Grimonprez, Président.
Jean-Jacques Quisquater,
William Caelli, Rapporteurs.
Jean-Marie Place,
Vincent Cordonnier,
Jean-Christophe Nicolas,
Eric Alzal, Examinateurs.

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât M3 59655 Villeneuve d'Ascq CEDEX
Tél. 20.43.47.24 Fax. 20.43.65.66

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS, A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. MIGEON Michel	EUDIL
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphérique
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
M. CATTEAU Jean Pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnement
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

Avant-Propos

Les travaux réalisés et présentés ont été effectués dans le laboratoire de Recherche et Développement Dossier Portable (RD2P) appartenant au Laboratoire d'Informatique Fondamentale de Lille. Je tiens en premier lieu à remercier le Professeur Vincent Cordonnier, Directeur du laboratoire et vice-président des relations internationales de la faculté des sciences de Lille 1, de m'avoir accueilli en DEA et de m'avoir permis de poursuivre ma recherche en thèse de doctorat. Je lui suis aussi extrêmement reconnaissant de m'avoir donné la possibilité tout au long des cestrois années d'assouvir ma soif de découvertes au travers de multiples missions dans des pays aussi lointains qu'exotiques.

Je tiens aussi à remercier le Centre National de la Recherche Scientifique (CNRS) et la société Gemplus de bien avoir voulu m'accorder un cofinancement Bourse Docteur-Ingénieur (BDI) sans lequel, bien entendu, je n'aurais jamais pu prolonger cette expérience au-delà du DEA.

Plus personnellement maintenant, j'exprime toute ma reconnaissance à Jean-Marie PLACE, Maître de conférences à l'IUT de Lille, et grand humoriste devant l'Eternel, qui a eu pour dure charge de se voir confié votre serviteur qui n'a eu de cesse que de le tourmenter. Encore merci pour son perpétuel soutien, et pour avoir su dépasser les simples relations de travail et devenir un véritable et sincère ami. Il est pour beaucoup dans la valeur de ce travail. Qu'il puisse répondre aussi bien à sa récente paternité qu'il l'a toujours fait avec ses proches et son petit Antoine sera le plus comblé des enfants ...

Une partie de cette thèse a été développée au Queensland University of Technology (QUT), en Australie. Je remercie vivement Bill Caelli, directeur du Information System Research Center (ISRC) et son équipe de m'avoir permis de réaliser ce projet. Associate Professor Ed Dawson est lui aussi à associer à cette belle aventure. Votre pays est un joyau, de la grande barrière de corail aux terres rouges d'Ayers Rock, tout concorde pour en faire une sorte de petit paradis sur Terre.

J'exprime ma profonde gratitude à Georges Grimonprez, Professeur à l'université de Lille 1, pour l'honneur qu'il me fait en présidant mon jury de thèse, ainsi qu'à mes rapporteurs :

- Jean-Jacques Quisquater, Professeur à l'Université Catholique de Louvain (UCL) en Belgique

- Bill Caelli, Professeur au Queensland University of Technology (QUT), en Australie

Mes examinateurs sont eux aussi chaleureusement associés à ces remerciements :

- Jean-Marie Place, Maître de Conférences à l'université de Lille 1

- Jean-Christophe Nicolas, Maître de Conférences à l'université de Béthune

- Vincent Cordonnier, Professeur à l'université de Lille 1

- Michel Escalant, responsable du test à Gemplus qui a du remplacer Eric Alzai en dernière minute

Je suis particulièrement touché que Bill Caelli ait accepté de consacrer du temps à la lecture de ce document, lui pour qui le français est une langue inconnue, aux étranges consonances. Merci pour ce surplus de travail.

J'ai eu la chance de travailler dans un groupe convivial, qui ne s'est jamais pris au sérieux, sans pour autant en manquer. J'aimerais en remercier tous les membres avec une pensée particulière pour les doctorants et autres étudiants qui un jour ou l'autre ont été amenés à franchir notre porte. Parmi eux, je pense à Thomas Alexandre, ami de 4 ans, Thierry Peltier, Marie-Pierre Haye, les petits nouveaux David Carlier, Sylvain Lecomte, Thomas Chamussy et Patrick Biget et celui qui reste notre modèle à tous, le premier de la bande, j'ai cité Olivier Caron. Philippe Roussel qui a activement participé à la réalisation du prototype carte est lui aussi particulièrement remercié et félicité.

La présence des Gemplusiens, et notamment de Pierre Paradinas, a perpétuellement permis de garder les pieds sur Terre et les "A quoi ça sert?" ont entretenu la bonne humeur au fil des saisons.

Il est des personnes au sein du LIFL sur lesquelles on peut toujours compter en cas de problème ... Parmi elles, Annie Kaczmarek a véritablement été d'une patience angélique, répondant toujours avec précision. Elle m'aura vraiment facilité la tâche dans le dédale administratif.

Oscar Benitez Roa, decano de la faculté polytechnique d'Asunción a toujours rêvé d'ouvrir une Maestria en Informática dans son université. J'ai eu le privilège d'assurer l'ouverture des cours à Asunción pendant six semaines. Merci à tous ceux qui m'ont donné la possibilité d'effectuer cette mission. Les chutes d'Iguaçu resteront à jamais gravées en moi comme étant le plus merveilleux paysage naturel qu'il m'ait été offert de voir.

J'ai eu la chance de faire mes premiers pas dans l'enseignement il y a quatre ans. A tous les étudiants qui ont dû subir ma présence répétée en cours année après année, je vous souhaite de bien réussir dans vos entreprises futures.

Puisque qu'il est de rigueur de ne pas toujours se retourner vers le passé, je tiens à remercier aussi Takashi Nanya, Professeur à l'université de Tokyo, de m'avoir autorisé à venir travailler à ses côtés l'année prochaine. J'associe mes remerciements au ministère des affaires étrangères et au CNRS de m'offrir les moyens de réaliser ce rêve.

Cet avant-propos ne serait pas complet si je n'associais pas aussi et surtout mes amis proches de Caen, de Paris, de Lille ou d'ailleurs, ceux et celles qui, de par les aléas de la vie, se sont petit à petit faits plus discrets, mes parents et surtout celle qui partage ma destinée et qui souffre depuis des années les affres de mes mauvaises humeurs en gardant toujours le sourire, merci à toi Sophie. J'espère ne jamais décevoir votre amitié.

Merci à vous tous.

Patrick TRANE

Table des Matières

Introduction

Le monde de la carte à microprocesseur est en constante évolution. De nombreuses applications exploitent les multiples caractéristiques qui rendent la carte si spécifique: la portabilité, la sécurité, l'aisance d'utilisation et le regroupement de données personnelles sur un support unique individualisé.

En raison de la définition de nouvelles cartes et notamment des cartes multi-applicatives, les besoins en terme de sécurité ne cessent de s'affiner. Si l'un des axes de recherche s'intéresse aux techniques cryptographiques ou d'identification biométrique et comportementale à des fins diverses (authentification, confidentialité) d'autres se concentrent plus particulièrement sur le contrôle d'accès aux données. C'est sur cet aspect du problème que s'oriente notre travail.

Ce mémoire s'organise autour de six chapitres.

Le premier introduit les notions fondamentales des différents aspects de la sécurité dans le monde de la carte à microprocesseur. L'analyse détaillée de ce thème permet une meilleure approche des besoins que nous introduisons par la suite. Après avoir énoncé les besoins en matière de sécurité spécifiques à la carte à microprocesseur en fonction de son cycle de vie, nous présentons brièvement quelles en sont les fonctions de sécurité statiques et dynamiques.

Le deuxième chapitre s'attache plus largement aux concepts de sécurité dans différents systèmes informatiques. Nous justifions le choix des systèmes d'exploitation, bases de données et même langages de programmation. Une étude comparative tente de mettre en évidence des politiques sécuritaires communes et fondamentales à ces trois systèmes. De manière analogue, nous mettons en avant une certaine hétérogénéité entre ces trois systèmes qui peut être perçue comme étant une faiblesse, mais qui peut aussi nous fournir un surplus d'éléments pour notre analyse postérieure.

Le troisième chapitre est une sorte de synthèse des deux premiers. Grâce aux enseignements acquis lors de cette étude préalable, nous allons décrire un outil dont la vocation sera de proposer aux

utilisateurs un mécanisme de contrôle d'accès aux données de granularité plus fine que ceux existant déjà. Afin de mieux positionner notre travail par rapport aux autres, nous proposons une étude comparative à l'aide d'un exemple simple. Les améliorations que nous voulons apporter sont à ce moment dévoilées. Deux nouveaux exemples essaient à la fois de montrer le confort d'utilisation de l'outil et de mettre en évidence quelques innovations que ce dernier apporte.

Le quatrième chapitre constitue une sorte d'aparté voire de rupture dans la continuité du travail proposé. Avant de prolonger notre étude de cas du S-Shell, nous voulons insister sur le fait qu'à l'aide d'un vérificateur d'intégrité, nous pouvons soulager le S-Shell d'un certain nombre de contrôles, en décrivant de manière rigide les variables que nous allons utiliser. Un contrôle d'intégrité fort de ces variables est proposé dans ce chapitre, et nous insistons bien à ce propos sur les avantages que peut apporter une telle vérification au niveau du S-Shell.

Le cinquième chapitre montre que notre système peut fort bien s'adapter à des projets ambitieux et innovants. Celui que nous avons choisi l'est particulièrement; dans le cadre de l'élaboration d'un système d'exploitation orienté carte multi-applicatives, nous proposons notre travail comme outil sécuritaire. Nous nous concentrons à cet effet sur l'adaptativité de notre travail dans un nouvel environnement.

Enfin le dernier chapitre a pour vocation d'adapter notre outil au milieu de la carte. Les conséquences dues à cet environnement très spécifique et très restreint aboutissent obligatoirement à dégrader notre travail afin de le rendre directement utilisable. Une application immédiate est alors proposée.

Le but de ce travail est non seulement de faire le point sur l'état de l'art en matière de sécurité informatique orientée contrôle d'accès, mais aussi de mettre en évidence certaines lacunes. Une proposition de solution est alors soumise à la sagacité du lecteur. Nous avons voulu présenter ce mémoire de façon très didactique et espérons que sa lecture sera à la fois agréable et instructive.

Chapitre 1

L'aspect sécurité de la carte à microprocesseur

1.1 Introduction

Le but de cette introduction n'est pas de fournir au lecteur des informations générales sur la carte à microprocesseur. De nombreuses références existent déjà dans ce domaine et nous invitons le lecteur soucieux de se documenter à se rapporter aux documents [CG91 , GUQ92 , C93]. Cependant il semble important, pour bien comprendre notre démarche qui a abouti à développer un système de sécurité évolué, de décrire brièvement ce en quoi la carte à microprocesseur a d'intéressant en tant qu'élément réputé pour sa très grande sécurité.

Avant de débuter, il faut malgré tout souligner que nous entendons par sécurité les moyens mis en oeuvre afin de protéger l'accès à des informations non autorisées. La sécurité peut être alors physique ou logique. La limite à cette définition est certainement la distinction entre sécurité et fiabilité ou résistance du matériel. Ainsi donc, la destruction du support (coup de ciseau, feu, ...) ou du composant (coup de marteau, ...) ne sera pas incluse dans notre définition de la sécurité d'une carte à microprocesseur. En revanche les techniques visant à lire des informations appartiennent entièrement à ce domaine.

Moyennant ces restrictions, la carte à microprocesseur est universellement reconnue par les spécialistes comme étant un outil informatique de très haut niveau de sécurité [CL90 , Sc89].

1.2 La carte à microprocesseur

La carte à microprocesseur a été inventée par Roland Moreno en 1974 sous le nom de brevet "procédé et dispositif de commande électronique". Le but d'un tel brevet est de disposer d'un mécanisme de mémorisation des données plus sûr, de fonctions d'identification du demandeur et de commande d'actions mécaniques du terminal pour des applications [S95] telles que les cartes de crédit bancaires. Ce dispositif doit ainsi pallier aux insuffisances et aux inconvénients de la carte à bande magnétique

(faible capacité de mémorisation, usure rapide, sensibilité aux champs magnétiques, ...) mais aussi et surtout permettre de lire ou de modifier les données et de dupliquer la carte. Roland Moreno a donc créé un circuit intégré contrôlant la sécurité de la mémoire de type "Mémoire Morte Programmable" qu'il contient. Ces circuits adaptés sur un support plastique ont été appelés des cartes à logique câblée et ont été généralisés avec la télécarte française et non pour les cartes de crédit auxquelles elles étaient destinées.

L'évolution et l'amélioration de la sécurité de ces cartes et de l'étendue de leurs applications ont permis d'aboutir en 1981 à une nouvelle génération de cartes. Ce sont les cartes à microprocesseur. Cette deuxième génération de cartes dotées d'un microprocesseur possède une panoplie d'outils physiques et logiques afin d'assurer sa propre protection. La terminologie utilisée dans le monde de la carte est la suivante:

1. La **carte à mémoire simple** désigne une carte permettant uniquement le stockage d'informations. Les cartes magnétiques et les cartes laser répondent à cette terminologie
2. La **carte à micro-circuit** désigne une carte comportant un composant électronique. Ce composant contient en plus d'une mémoire une logique câblée. L'ensemble des fonctions (fonctions de calcul, sérialisation des données...) est prédéterminé et le circuit résultant, après étude par composants discrets, est "intégré"

   Les télécartes classiques (carte à logique câblée) répondent à cette désignation
3. La **carte à micro-processeur** ou plus habituellement nommée la **carte à puce** désigne, bien sûr, une carte comportant un micro-processeur et son environnement (mémoires, entrées/sorties). Par la suite, nous désignerons simplement par le terme carte une carte à micro-processeur

Il est bien difficile de dire quelles seront les cartes futures. Néanmoins il existe de nombreux prérequis qui ont été et sont toujours largement étudiés et auxquels les cartes devront à n'en pas douter répondre. Citons à titre de référence les articles traitant de l'évolution de la carte suivants [PV94 , Pe94].

Il est fondamental de bien étudier les points forts et de mettre en évidence quelques lacunes pour introduire le travail qui a été effectué dans cette thèse. C'est ce à quoi nous nous attacherons tout au long de ce chapitre.

1.3 Caractéristiques physiques

En tant qu'objet physique la carte est constituée de trois éléments principaux:

- Le support du composant
- Le micro-module
- La pastille de contacts

Le support dans sa forme la plus répandue, c'est-à-dire carte plastique ou ABS, normalisé possède une cavité pour le microcircuit. L'emplacement de la pastille ne doit par ailleurs pas empiéter sur la piste magnétique ni sur la zone d'embossage. Son degré de résistance que l'on a aussi appelé fiabilité est évidemment très faible dans le sens où il est très facile de le détruire volontairement. Cependant une résistance minimale aussi bien du support que de la pastille à des contraintes de résistance physique est aujourd'hui obligatoire (par exemple des contraintes de torsion minimale sont requises afin que le support ne soit pas détruit par simple port de la carte dans une poche arrière de pantalon). Ces contraintes matérielles sont normalisées dans [ISO1]. Les contraintes relatives à la pastille de contact sont elles aussi normalisées dans [ISO2]. Les plus courantes sont:

- Résistance mécanique de la carte et des contacts: ce sont les contraintes de torsion auxquelles il faut ajouter celles concernant l'insertion de la carte dans un lecteur et l'établissement des contacts
- Profil de surface des contacts
- Protection aux ultra-violets: ces sources électromagnétiques peuvent altérer le contenu de la mémoire de la carte
- Sensibilité aux rayons X: ils peuvent altérer le contenu de la mémoire de la carte
- Niveau d'électricité statique supporté qui peut causer un dérèglement voire une destruction du circuit intégré
- Niveau de dissipation thermique accepté dû au fonctionnement du composant

1.4 Influence de la sécurité sur le cycle de vie

1.4.1 Le cycle de vie d'une carte à microprocesseur

Le cycle de vie d'une carte est une caractéristique importante. Ce cycle de vie [GS90] est inexistant dans les systèmes informatiques classiques. Les fonctionnalités de la carte évoluent en fonction de la phase dans laquelle elle se trouve. Nous distinguons quatre phases séquentielles dans le cycle de vie d'une carte:

- La phase de fabrication
- La phase de personnalisation
- La phase d'utilisation
- La phase de terminaison

Le document [C94] aux pages 18 et 19 précise avec clarté les différentes particularités de chaque phase. Notons simplement que la carte devient "utilisable" par le porteur une fois achevée la phase de personnalisation. La phase de terminaison peut être due à des circonstances physiques particulières (perte de la carte, ...). Cependant, il est communément admis que les trois cas classiques d'entrée en phase de terminaison sont:

- zone de transaction pleine, et par conséquent impossibilité d'effectuer des opérations de mise à jour. Cette zone contient les données retraçant les opérations effectuées avec la carte et qui dépendent de l'application : références d'un paiement, d'un abonnement, etc ...
- zone d'état pleine, c'est-à-dire la zone gérant les contrôles des codes secrets, la carte est dite saturée et il est alors impossible de présenter le code secret.
- invalidation de la carte par l'application lors d'une succession de mauvaises présentations du code secret

1.4.2 Les 'pirates' informatiques

Les besoins de sécurité [Bo95] évoluent de même en fonction et de la phase dans laquelle on se trouve et des aptitudes des personnes malveillantes. En effet, afin de parfaire la sécurité de la carte, on se doit de supposer que chaque personne ou chaque institution est une entité potentielle dangereuse. Même les institutions en qui on peut avoir confiance peuvent elles-mêmes employer du personnel qui pourrait de par leurs droits d'accès aux informations les détourner à leur propre profit. J. Svigals a d'ailleurs consacré une partie de son oeuvre [Sv87] à classer les 'pirates' en fonction de leur degré de connaissance:

- Les 'pirates' occasionnels: ce sont ceux qui de par leur fonction ont accès à des informations confidentielles mais sans pour autant avoir des connaissances spécifiques
- Les 'pirates' déterminés: ce sont ceux qui, comme un étudiant acharné pour qui vaincre la résistance d'un système informatique relève d'un défi personnel, ont des connaissances réelles
- Les 'pirates' professionnels: ce sont ceux qui associent à une longue expérience de la fraude ou de l'attaque un savoir-faire préoccupant

1.4.3 Les attaques les plus courantes

Les cartes sont bien sûr exposées à toute une panoplie d'attaques possibles et il serait bien illusoire de vouloir toutes les énumérer. Néanmoins certaines plus connues ou plus préoccupantes que d'autres sont à mettre en avant. Les mécanismes de protection les plus importants doivent assurer:

- La protection contre une lecture non autorisée: c'est le secret de l'information. Une donnée ne doit être accessible que par les personnes qui ont des droits en lecture sur cette donnée. C'est le problème classique de la protection des données
- La protection contre une modification non autorisée: c'est l'intégrité de l'information. Elle concerne aussi bien les données confidentielles que les autres. Par exemple le nombre d'unités restantes sur une télécarte n'est pas un secret, par contre l'incrémentation de ce nombre ne doit pas être réalisable
- La protection contre la copie non autorisée: c'est la duplication de l'information

1.4.4 Les risques associés à ce cycle de vie

En ne se référant qu'à ce sous-groupe volontairement restreint des attaques logiques possibles sur une carte, des études sur le degré de risque que court la carte à microprocesseur à chaque phase de son cycle de vie ont été menées et ont fourni les résultats suivants [Sb89] (voir table 1):

TABLE 1 Risque d'attaques potentielles sur la carte

Cycle de vie	fabrication	perso.	utilisation	terminaison
Attaques intérieures	H	H	H	m
Attaques extérieures	m	m	H	H

H: risque haut m: risque modéré

On appelle attaques intérieures des attaques émanant de personnes malveillantes travaillant dans une institution spécialisé dans la carte (ie connaissant bien son fonctionnement et pouvant profiter de leur situation privilégiée au sein de l'entreprise).

Il est donc essentiel de bien protéger la carte tout au long de son évolution.

1.4.5 Niveau de sécurité et cycle de vie

Les applications carte à microprocesseur offrent selon les types d'utilisateurs - c'est-à-dire le fabriquant, l'émetteur et l'utilisateur final - des niveaux de sécurité différents, très fortement liés au cycle de vie de la carte. Des mécanismes de sécurité sont mis en place afin de bien pouvoir respecter cette dissociation des utilisateurs, de façon à interdire toute ingérence d'un groupe d'utilisateur dans un autre. Par exemple, le passage du bit de personnalisation à 1 parachève la phase de personnalisation de la carte et met fin à la description de la politique de sécurité de l'émetteur de la carte. On voit qu'ainsi les notions de sécurité des applications [E95 , BE95] sont très liées aux caractéristiques de la carte. Elles sont cependant, de par leur caractère statique, un obstacle aux notions innovantes de cartes multi-applicatives (au cycle de vie perturbé, se référer au chapitre 5 intitulé "Intégration dans un système d'exploitation Carte Blanche"). Nous aurons l'occasion d'y revenir plus longuement.

1.4.6 Principe d'utilisation de la carte à microprocesseur

Vis-à-vis de l'application, les mémoires internes comportent des informations secrètes (code porteur, programme de commande du microprocesseur, ...) qui ne peuvent être lues, modifiées ou détruites par l'utilisateur de la carte.

Les données liées à l'application elle-même seront complétées au cours de l'utilisation de la carte, soit automatiquement, soit par le porteur. Leur mémorisation est subordonnée à un contrôle effectué par le microprocesseur. Après ce contrôle, le microprocesseur signe les données en indiquant qu'elles proviennent du porteur (l'émetteur est aussi habilité à compléter les données sous certaines conditions, à l'aide d'un code émetteur différent). Il les enregistre (donnée et signature) et détruit aussitôt le mécanisme qui a permis leur écriture, ce qui rend impossible toute tentative de modification ou d'effacement ultérieur.

1.5 Sécurité intrinsèque

1.5.1 Eléments de sécurité physique

Le micro-module est constitué d'éléments qui sont contenus dans un substrat unique de silicium. L'ensemble des éléments doit par conséquent être réalisé dans des technologies compatibles et se contenter de 6 ou 7 contacts pour communiquer avec l'extérieur. Un des intérêts, du point de vue de la sécurité physique, de la carte monocircuit réside dans l'absence d'interconnexion processeur-mémoire. Ceci empêche ainsi toute possibilité de surveillance à faible coût ou d'espionnage par sonde entre les composants. Le terme 'microprocesseur' employé pour désigner l'ensemble des composants portés par la puce (miroprocesseur, mémoire, circuit d'échange de données) est insuffisant, il s'agit en fait d'un réel microcalculateur ou M.A.M. (Microcalculateur Autoprogrammable Monolithique) comme l'a appelé l'entreprise BULL en 1978[1]. Le terme "autoprogrammable" est utilisé car la carte possède les circuits nécessaires à un fonctionnement autonome, la possibilité de modifier sa mémoire E(E)PROM particulièrement réservée aux données.

Toujours dans le but d'optimiser les aspects sécuritaires physiques, les dimensions et la technologie utilisée répondent à ces besoins. Par exemple, afin de réduire la sensibilité aux interférences entre la bande magnétique et le circuit, la technologie CMOS 3,5µ (Complementary Metal-Oxyde-Silicon) a été retenue. Elle a aussi la particularité d'être peu consommatrice de courant. Des circuits à haute intégration, dits HCMOS 1,2µ permettent d'en réduire encore les dimensions et d'en augmenter les performances (on appelle dimension d'un circuit intégré la distance entre la source et le drain).

Le composant a de plus une surface limitée afin de se conformer aux normes de torsion et de flèche. La surface maximum autorisée est de 20 à 30 mm^2 , son épaisseur est, elle, de l'ordre de 0,28 mm.

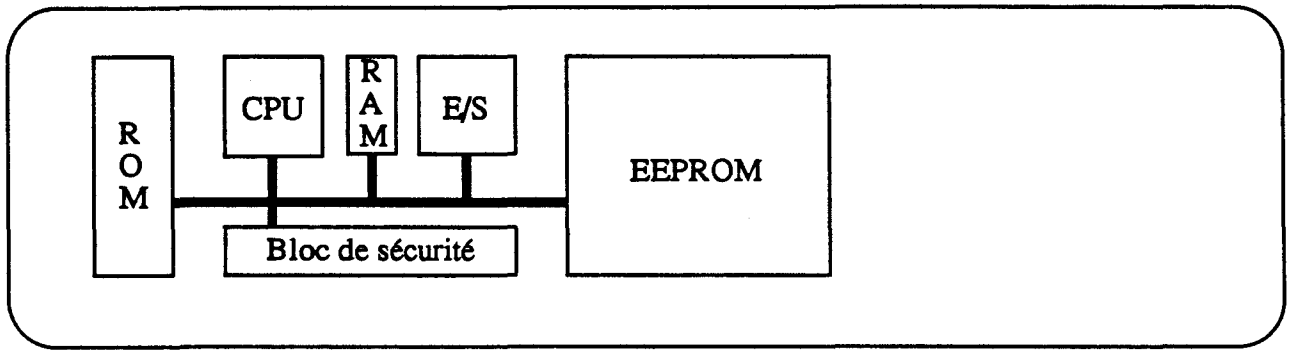
Les composants du micromodule sont:

- Le microprocesseur qui est le point de passage obligé pour toute communication entre la carte et le monde extérieur
- La mémoire de travail RAM qui est la mémoire vive de travail, la mémoire volatile dans la quelle le M.A.M est capable d'écrire lui-même sans que le monde extérieur puisse atteindre les jonctions internes ni agir sur le déroulement du programme
- La mémoire EPROM (indélébile) ou EEPROM (effaçable) qui sont les mémoires de données programmables (car elle s'enrichit en informations au cours de la vie de la carte) contenant les informations liées à l'application
- La mémoire ROM indélébile qui contient l'ensemble des programmes gravés lors de la fabrication du composant. C'est le masque du microcalculateur
- Les entrées/sorties

1. S.P.O.M. en anglais (Self-Programmable One-Chip Microcomputer)

- Le bloc de sécurité

FIGURE 1 Architecture d'un M.A.M.



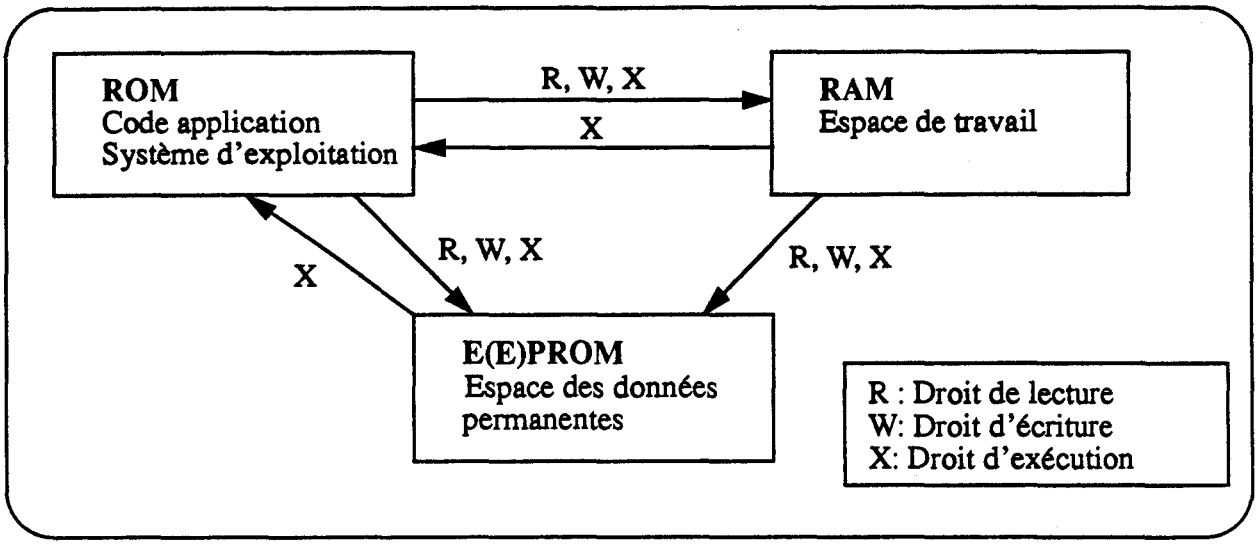
Les premiers éléments sont parfaitement étudiés dans [GRL88] et ne seront pas repris dans notre descriptif car ils n'apportent pas de caractéristiques concernant la sécurité globale de la carte particulières. En revanche, il est fondamental de détailler le rôle joué par le bloc de sécurité. Il doit assurer certains contrôles relatifs à la sécurité physique du composant.

- Détection de lumière: Le micromodule est recouvert d'une couche de résine opaque. En cas d'attaque matérielle sur cette couche, le détecteur provoque un arrêt du fonctionnement du micromodule
- Détection de passivation: Un diélectrique recouvre le module. Sa vocation est double. Il protège les éléments de la poussière et rend le sondage des données transitant sur le bus enterré ou en RAM impossible (c'est un isolant)
- Détection de variation de température, de tension et de fréquence: Une anomalie ou une malveillance basée sur un de ces trois critères sera immédiatement détectée par la carte. Son utilisation sera dès lors impossible. Le détecteur de tension vérifiera notamment que la tension est suffisante pour écrire dans la mémoire. Le détecteur de variation de fréquence vérifiera qu'on n'abaisse pas la fréquence pour augmenter le délai de garde

1.5.2 Le modèle de la matrice de sécurité

En outre, certains composants possèdent un circuit physique appelé matrice de sécurité [D82] qui a pour vocation de régir les droits de lecture, écriture et exécution des différentes mémoires dans la carte à microprocesseur. Cette matrice de sécurité supervise les accès du processeur, indépendamment du logiciel, aux différents types de mémoire. Un exemple classique de matrice de sécurité est celui de la figure 2:

FIGURE 2 Matrice de sécurité



On note que sur cette matrice de sécurité la mémoire ROM n'est accessible qu'en mode exécution, ce qui interdit le "dump" de la ROM.

Certaines autres techniques sont mises en jeu afin d'augmenter le niveau de sécurité physique de la carte. Le brouillage des adresses ROM et EEPROM permet d'éviter l'effacement d'un bloc complet d'adresses mémoire. Par ce genre de technique, la lecture optique directe de la mémoire par microscope électronique ne permet plus aujourd'hui de reconnaître les informations stockées. De plus, un mode test protégé par des fusibles empêche de revenir à un stade antérieur du cycle de vie de la carte (une fois personnalisée, un fusible est grillé une fois pour toutes et la carte rentre dans la phase d'utilisation de son cycle de vie). Une modification par l'utilisateur est donc physiquement impossible.

1.6 Les fonctions de sécurité statiques

1.6.1 La sécurité logique

Les menaces qui pèsent sur la carte quant aux attaques logiques peuvent être de plusieurs natures:

- *Confidentialité des données* (lecture d'informations protégées sans autorisation, écriture non autorisée d'informations).
  Afin de se prémunir de telles attaques, la carte est dotée de mécanismes d'authentification permettant l'accès à certaines données de la carte. Ceci s'effectue le plus souvent par présentation de clés (voir paragraphe 1.7 : "les systèmes à clés" à la page 20). Dans cette hypothèse, seule la connaissance de la clé permet de lire la carte. La plupart du temps, le porteur a trois essais pour présenter la bonne clé (mécanisme de ratification). En cas d'invalidation de la clé, le code peut être réhabilité par une autorité supérieure (banque, etc ...)

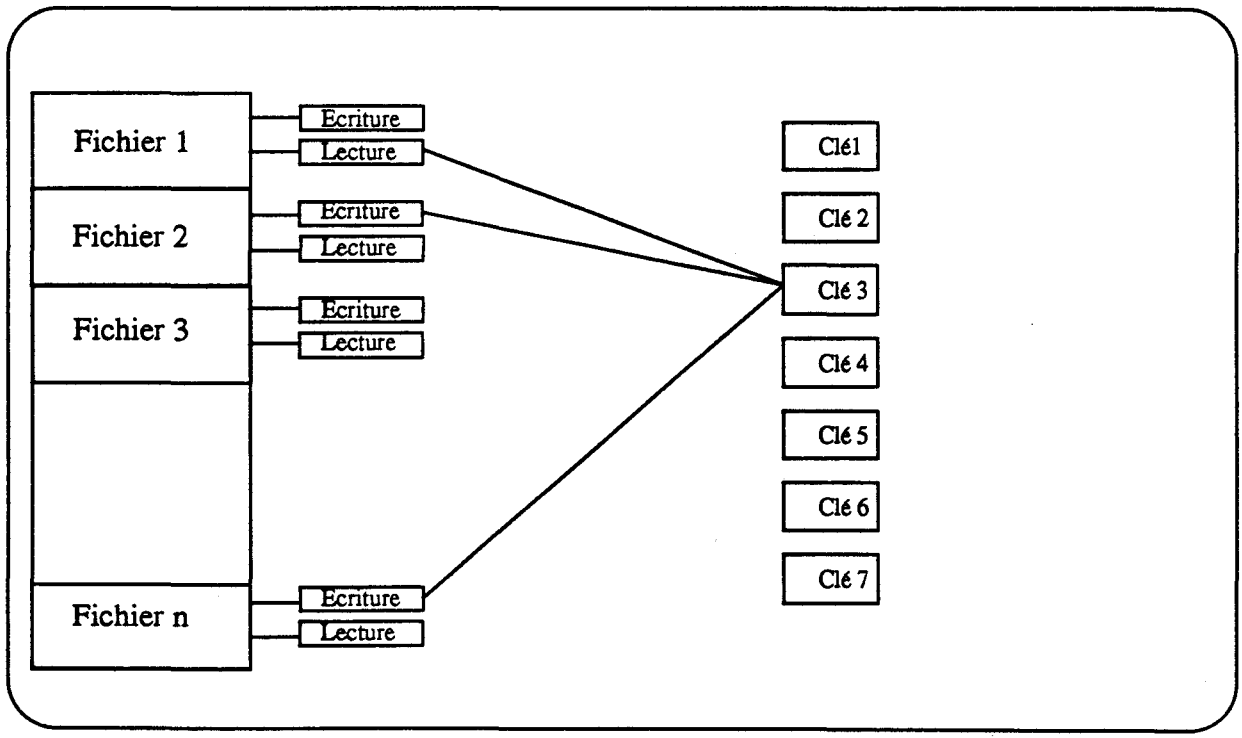
- *Intégrité des données* (altération d'informations du système ou du porteur). Ceci est assuré car l'écriture dans la carte est soit impossible, soit reconnaissable par lecture de la clé sous laquelle ont été écrites les informations. Il est de plus possible de certifier ce que l'on a écrit dans la carte (voir paragraphe 1.8.4 : "la certification" à la page 25)

Ces problèmes liés à l'analyse de risques ne sont pas nouveaux. Dans les systèmes informatiques classiques, les spécialistes de la sécurité de l'information désirent que soient à la fois préservées la confidentialité, l'intégrité et la disponibilité des données [Pa91]. Toutefois certains reproches sont aujourd'hui formulés quant à cette division en trois parties de l'analyse. Les lecteurs soucieux d'approfondir la question pourront se référer à l'article [Pa95] qui tente de mettre à jour certaines lacunes de cette classification en introduisant les notions de possession, d'utilité et d'authenticité.

1.6.2 Le cas de la carte COS

La carte COS (Card Operating System) ancêtre des cartes MCOS [Ge90] et MPCOS [Ge94] présente un profil de protection des fichiers par clés. Sur la figure 3, on remarque par exemple que seuls les détenteurs de la clé 3 pourront avoir des accès en écriture des fichiers 2 et n, et un accès en lecture du fichier 1.

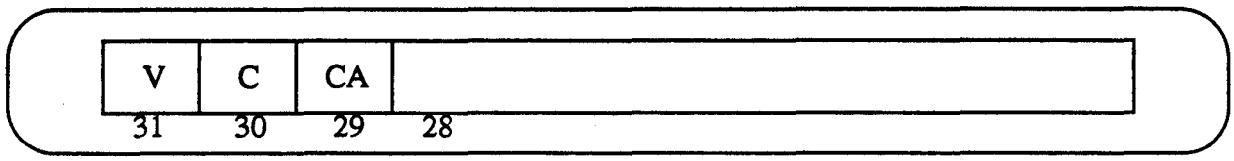
FIGURE 3 Protection des fichiers COS



1.6.3 Le cas des cartes Bull CP8 et Philips DES-D1

La taille de la mémoire EPROM des cartes Bull CP8 et Philips DES-D1 est de 8 kbits (256 mots de 32 bits). Elle est divisée en 7 zones de taille, de contenu et de mode d'accès spécifiques. Certaines zones sont personnalisées soit en lecture libre ou protégée, soit en écriture libre ou protégée. L'ensemble des zones est obligatoire hormis la zone confidentielle. Ce qu'il est important de souligner est que la taille d'un mot est de 32 bits répartis en 29 bits servant à mémoriser l'information et 3 bits système impliqués dans l'intégrité et la cohérence de l'information (voir figure 4).

FIGURE 4 Protection des cartes Bull CP8 et Philips DES-D1



Bits C-CA: Ils indiquent sous quelle clé a été faite l'écriture. Cela peut être l'émetteur primaire (clé 1A), l'émetteur secondaire (clé 1B) ou le porteur (clé 2A qui est le code confidentiel classique ou clé 2B). Les quatre clés sont contenues dans la zone secrète.

Bit V: Il permet le verrouillage définitif du mot. Ainsi, aucune modification ultérieure des bits ne sera possible. Au moment du verrouillage, il y a contrôle de cohérence entre la clé présentée et le contenu des bits C et CA.

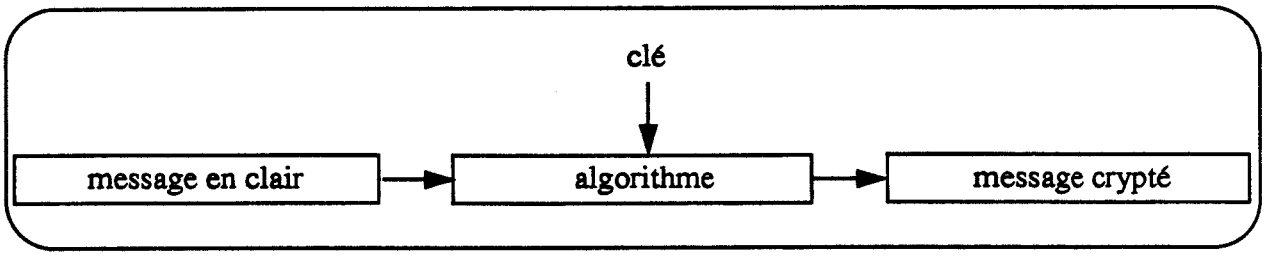
1.7 Les systèmes à clés

En parallèle avec les fonctions statiques que nous avons décrites au chapitre précédent, la sécurité des données de la carte est aussi assurée grâce à un certain nombre de techniques mises en place par utilisation de codage. Son rôle est de rendre un message confidentiel inintelligible à tout autre personne que son ou ses destinataires légitimes (confidentialité), tout en permettant à ces derniers d'être capables de reconnaître l'émetteur du message (authentification) non modifié ou non créé (intégrité) pour la circonstance et bien évidemment de le lire.

Un message en clair (c'est-à-dire non crypté) est transformé en un cryptogramme non significatif. Les caractères de ce cryptogramme sont alors regroupés en blocs de longueur constante. Le message est ensuite décodé au moyen de techniques analogues.

Outre les méthodes classiques basées sur la transposition, la permutation et la substitution de caractères, il existe certains systèmes dits à clé qui ont pour vocation de chiffrer un message clair en combinant dans un algorithme de calcul le message et une suite de chiffres appelée clés comme le montre la figure 5.

FIGURE 5 Chiffrement d'un message



Il existe à l'heure actuelle deux systèmes de chiffrement à clé:

- Les systèmes à clé secrète ou symétriques: L'émetteur et le récepteur utilisent alors la même clé secrète unique. Un ensemble d'opérations figées est ensuite exécuté dans un ordre prédéfini afin de chiffrer le message. Le déchiffrement est possible en effectuant toutes les opérations dans l'ordre inverse. Les systèmes à clé secrète les plus connus sont le D.E.S (Data Encryption Standard), Télépass de Bull CP8 ou encore le G.O.C (Générateur d'Octets Chiffrants) de France Telecom
- Les systèmes à clé publique ou dissymétriques: Les opérations de chiffrement et de déchiffrement sont cette fois totalement différentes. La procédure de chiffrement est publique alors que la procédure de déchiffrement est gardée secrète. Ceci n'altère en rien la sécurité du système. Paradoxalement un système à clé publique est réputé pour être moins vulnérable qu'un système à clé secrète. Le plus connu d'entre eux est le R.S.A (River, Shamir, Adleman). Une description de cet algorithme est disponible dans [RSA78].

Nous trouvons dans les cartes des systèmes de chiffrement à clé secrète. Le D.E.S précédemment cité a été adopté comme standard par plusieurs organisations dont l'ANSI (American National Standards Institute). Le lecteur désireux de connaître cet algorithme peut se reporter à [NBS80]. Les algorithmes à clé secrète présentent deux inconvénients : ces algorithmes sont relativement "cassables" et ils impliquent une gestion difficile des clés (distribution, renouvellement, ...). Les algorithmes à clé publique [GUQ92] présentent des résultats supérieurs et ont l'avantage de pouvoir diffuser la procédure de chiffrement sans altérer la sécurité du système (la procédure de déchiffrement reste secrète).

Tout comme les algorithmes biométriques, les algorithmes cryptographiques exigent une forte puissance de calcul difficilement compatibles avec les micro-processeurs cartes actuels. Le tableau 2 exprime les temps de calcul du D.E.S. (pour 1 Ko de données) sur différentes machines compatible PC et les compare au temps de calcul sur une carte équipée de cet algorithme. Les algorithmes à clé publique type R.S.A présentent des temps de calculs très supérieurs (exemple : 76,8 secondes pour

une génération de signatures de 512 bits sur un microprocesseur 8 bits, 5 Mhz, ces algorithmes fortement basés sur le calcul d'exponentiation en est la raison principale.

TABLE 2 Temps de calcul de l'algorithme D.E.S.

Machine	Temps (secondes)
HP VECTRA 486-33	0.93 s
HP VECTRA 486sx-20	1.45 s
HP VECTRA 386-25	1.63 s
Zenith 386 sx-16	3.08 s
Bull 286-12	6.32 s
Carte MCOS (6805)[a]	≈12 s

a. hors temps de transmission

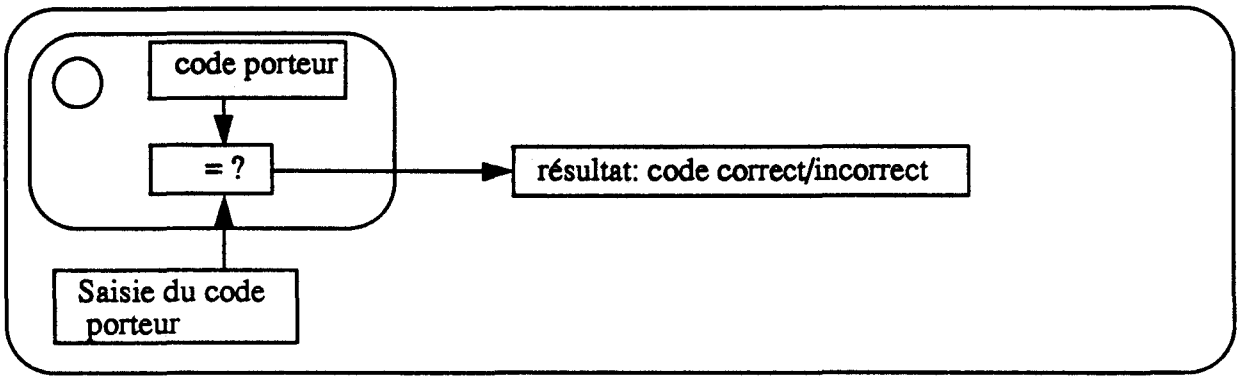
1.8 Les fonctions de sécurité dynamiques

Ce sont des fonctions typiquement logicielles. Elles s'exécutent sur un système autonome ou connectable à un autre système. Dans ce dernier cas, le porteur de la carte est demandeur d'un service. Le central communiquera avec la carte mettant ainsi en oeuvre les fonctions dynamiques. A part peut-être l'identification, toutes les fonctions que nous allons maintenant décrire font appel aux techniques de codage dont nous avons parlé dans le chapitre précédent.

1.8.1 L'identification

Lorsqu'une carte est remise au porteur, l'organisme responsable fournit en plus de la carte un code secret (ie la clé porteur) afin qu'il puisse s'identifier. La carte à microprocesseur est active dans le sens où elle va véritablement effectuer un contrôle du code présenté et vérifier l'identité numérique entre le code entré et le code secret qu'elle possède. Si le code entré correspond à celui de la carte, la légitimité du porteur est reconnue et l'application requise se poursuivra. Sinon, généralement deux nouvelles tentatives sont accordées avant de bloquer la carte. C'est la ratification.

FIGURE 6 Identification: utilisation en local



Par soucis de sécurité, le code porteur est souvent chiffré de façon à ce qu'un éventuel fraudeur, même en possession de la carte ne puisse pas retrouver le code d'identification. L'intrus devra alors connaître non seulement le code porteur, mais aussi l'algorithme et la clé de cryptage utilisée. Cette technique largement répandue est pourtant souvent critiquée. Les insécurités qu'elle présente ne font pas l'objet de cette thèse et ne seront donc pas développées ici. Cependant nous invitons le lecteur à se rapporter à [Ja89, MT79] . L'article [A93] fournit en supplément un certain nombre d'attaques ou de dysfonctionnements à base de PIN code intéressants mettant en évidence certaines faiblesses techniques ou d'utilisation courante de cette méthode d'identification.

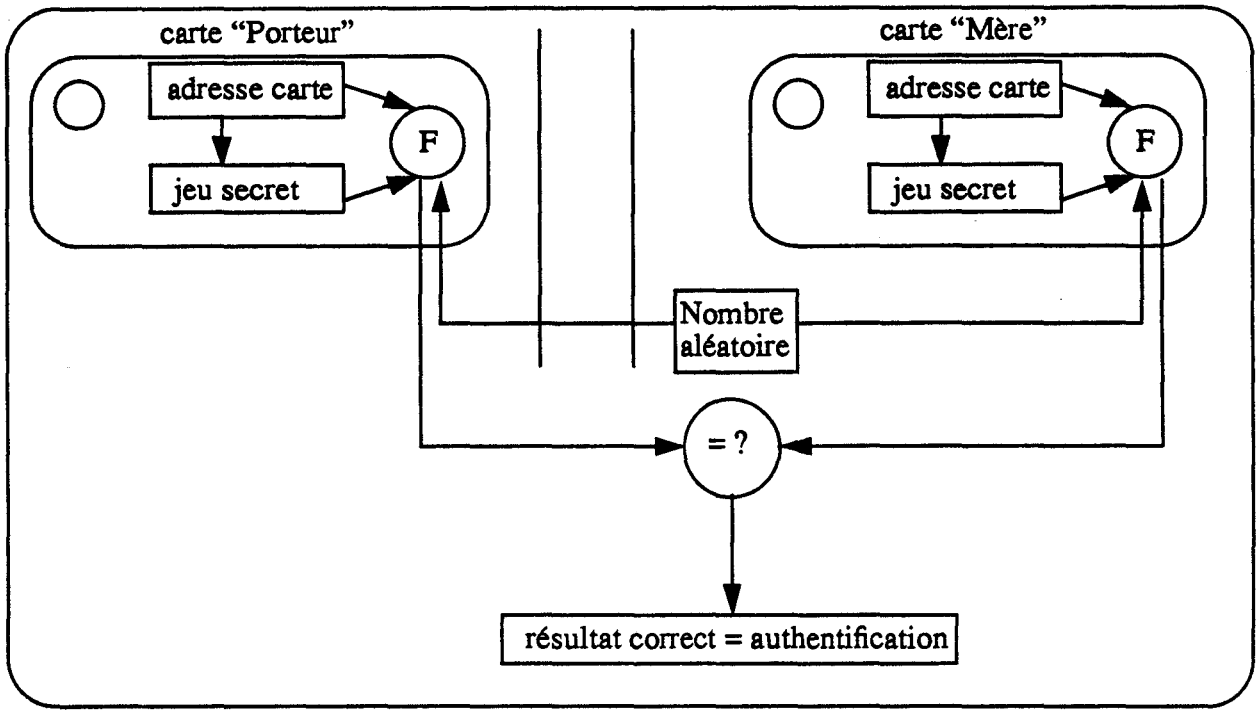
1.8.2 L'authentification

L'authentification suppose que l'émetteur et le récepteur soit distants. Le but est que le récepteur puisse authentifier l'émetteur, c'est-à-dire que l'émetteur puisse bien prouver être ce qu'il prétend. L'idée est d'échanger un nombre aléatoire entre l'émetteur et le récepteur et d'appliquer un algorithme de chiffrement sur le nombre et sur l'identifiant de la carte(numéro de série et contenu d'un mot carte réservé à cet effet).

Le récepteur reçoit ces deux informations et après vérification peut à son tour effectuer la même opération. Si les deux résultats (reçu et calculé) concordent, alors le récepteur aura authentifié l'émetteur. Ce mécanisme fonctionne car les deux calculs effectués par l'émetteur et le récepteur ne peuvent être accomplis qu'en connaissant l'algorithme de chiffrement et les clés utilisées.

L'authentification du récepteur par l'émetteur peut dans des conditions identiques être acquise.

FIGURE 7 Authentification: Entre un site central (carte "Mère") et un lecteur distant (carte "Porteur")

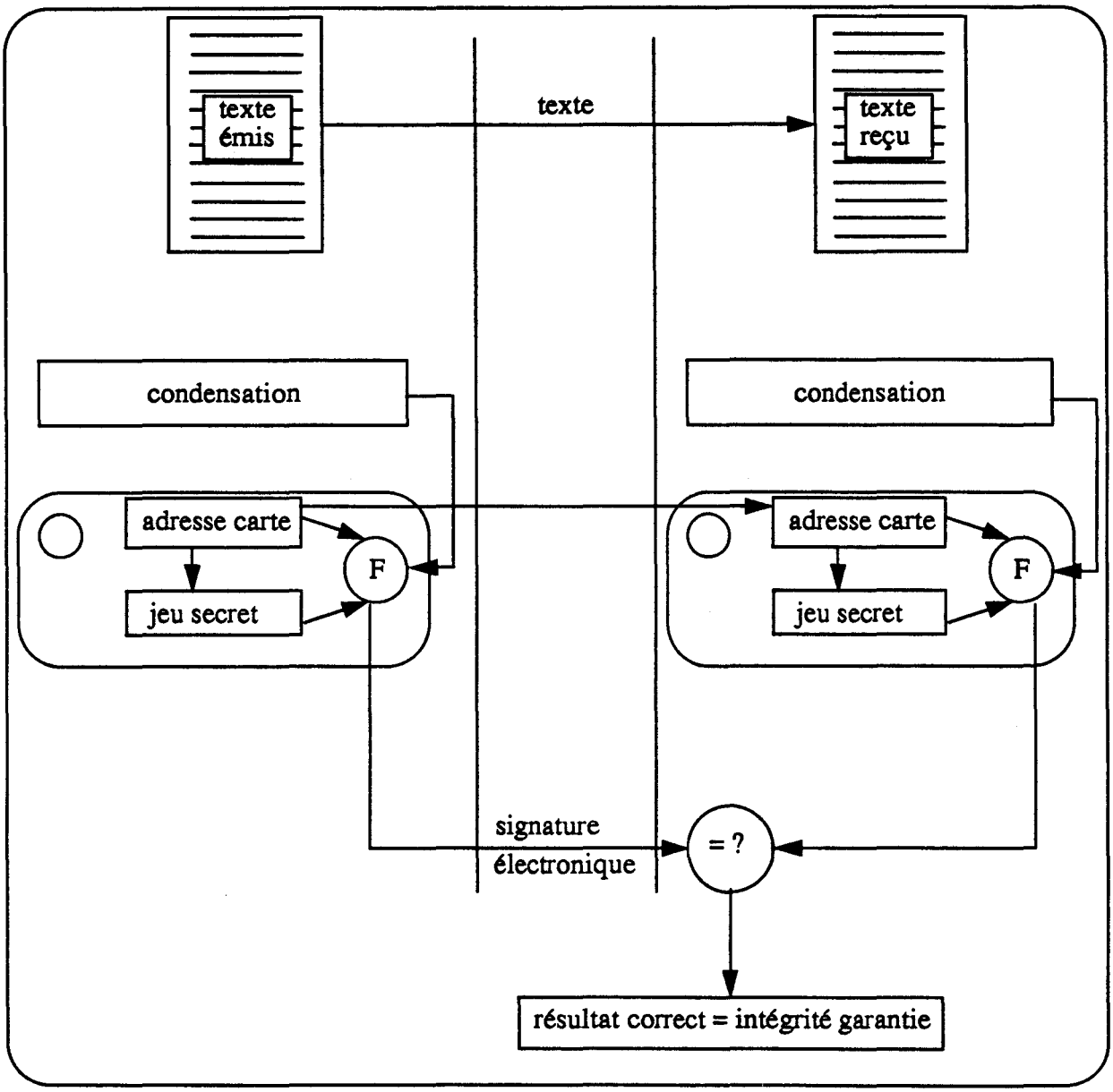


1.8.3 La signature électronique

La signature électronique [Sc93-2] sert à vérifier qu'après l'authentification des deux parties, un fraudeur n'essaie pas d'émettre de faux messages. Traditionnellement, on produit une signature numérique en appliquant une transformation cryptographique au message de façon à ce que le résultat soit de taille réduite, indépendamment du message. Pour cela, l'émetteur effectue une opération de hachage [SHS92, D90] sur les données émises en bloc, puis exécute un chiffrement sur le résultat de ce hachage et l'identifiant de la carte [Sc91]. L'homonymie avec la signature manuelle se limite à la notion de signature fournie par l'émetteur car dans le cas d'une signature électronique, la signature dépend aussi du texte signé. La notion de chiffrement évoquée ci-dessus provient du fait que la signature doit pouvoir être vérifiée par tout monde sans pour autant pourvoir être imitée. Ceci implique donc la publication d'une clé de vérification (c'est pourquoi l'algorithme est dit à clé publique) alors que la clé de production de la signature doit, elle, rester secrète (et ne doit pas pouvoir être déduite de la clé publique). La provenance des données est ainsi assurée.

Remarque: Nous allons maintenant, après avoir présenté le schéma fonctionnel de la signature électronique, parler de la certification. Or, il y a bien souvent ambiguïté entre ces deux termes. Notons simplement que les algorithmes à clé secrète ne permettent généralement que de se prémunir contre la non-altération en appliquant un sceau ou certificat au message. Sans opération de dissymétrisation, ces algorithmes appelés aussi algorithmes symétriques ne peuvent assurer la non-répudiation d'un message.

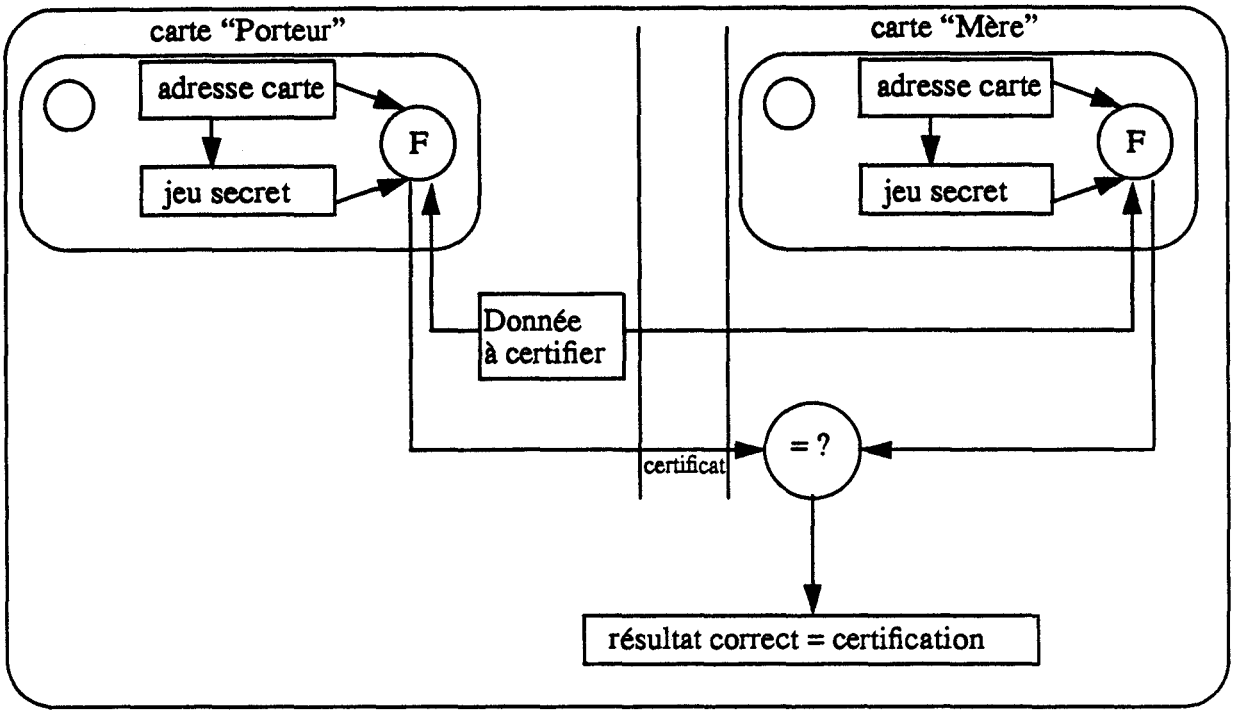
FIGURE 8 La signature électronique: schéma descriptif



1.8.4 La certification

Elle consiste à certifier chaque échange de données au cours d'une transaction. Il s'agit d'une validation par un récepteur d'une donnée transmise par un émetteur, grâce à un algorithme utilisant à la fois les données de l'émetteur et les caractéristiques du récepteur. La certification protège à la fois de la fraude et des erreurs de transmission. L'intégrité des données est alors assurée.

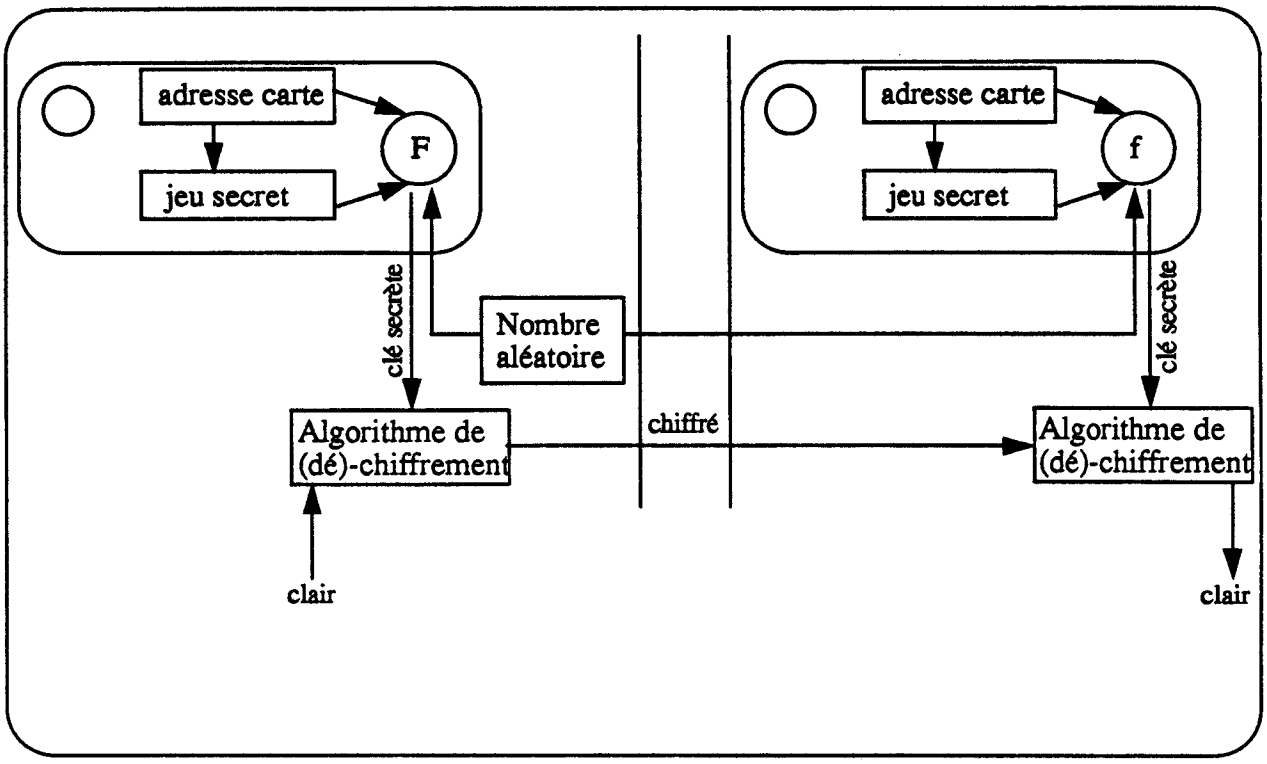
FIGURE 9 Certification: Entre un site central (carte "Mère") et un lecteur distant (carte "Porteur")



1.8.5 Le chiffrement

L'ultime étape vise à assurer la confidentialité des données. Ceci pour prévenir l'écoute d'une ligne par un fraudeur. L'intérêt du choix du chiffrement utilisé est un habile compromis entre la vitesse de transmission et la fiabilité de l'algorithme de chiffrement. Un système de chiffrement fort nécessitera un temps de transmission nettement plus long, mais préviendra efficacement d'une possible malveillance. En revanche un système de chiffrement plus simple permettra des délais de transmission intéressants mais abaissera le niveau de fiabilité de la ligne. Il faut alors changer de clé secrète assez souvent. La réalisation idéale est d'utiliser une clé par émission. A la fin de l'échange, la clé est délaissée. L'envoi de la clé est lui aussi chiffré par un algorithme de haute qualité. Eventuellement pour un échange "long" et donc dangereux (la clé peut être trouvée en cours d'échange), il faut donc être capable de remplacer la clé périodiquement. De nombreux ouvrages traitent en détail de ces problèmes de chiffrement, par exemple [Sc93-1, D82].

FIGURE 10 Chiffrement



1.9 Remarques sur les techniques d'identification

La technique usuelle utilise un code numérique permettant l'identification du porteur de la carte. Ce système présente une faille : la perte d'une carte sur laquelle le porteur aurait inscrit son code secret permettrait son utilisation par une tierce personne. La présentation du bon code numérique permet simplement de prouver que le possesseur actuel de la carte connaît le code. Elle ne permet pas de prouver que le possesseur actuel de la carte est le légitime propriétaire de celle-ci. A ce jour, seule l'utilisation de techniques biométriques associées à la carte permet de résoudre ce problème. Par la suite, nous appellerons *signature* les données issues d'un relevé biométrique, et *algorithme de reconnaissance* le programme qui permet de valider ou non une signature par rapport à une *signature de référence*. Les algorithmes de reconnaissance nécessitent une relativement forte puissance de calcul. C'est pourquoi, il n'existe pas de cartes possédant de tels mécanismes. Certains constructeurs proposent des cartes disposant de fonctionnalités biométriques. En réalité, nous ne trouvons dans ces cartes que la signature biométrique du porteur, l'algorithme de reconnaissance s'effectuant à l'extérieur de la carte. Cet aspect implique que la signature doit à un moment donné "sortir" de la carte, et donc diminuer le niveau de sécurité de celle-ci. Le cas idéal est "simple" : la carte doit contenir la signature référence, ainsi que l'algorithme de reconnaissance qui doit s'exécuter dans la carte. Ces conditions optimales et les contraintes de la carte font que la recherche actuelle se dirige vers deux voies :

- La recherche d'une codification très compacte de la signature. La recherche sur des algorithmes de compression de données apporte également des solutions à ce problème.
- La recherche sur des algorithmes de reconnaissance performants. Ces algorithmes doivent prendre en compte trois paramètres : la "faiblesse" du micro-processeur, la taille du code de l'algorithme ainsi que la taille des ressources (taille des mémoires RAM et EEPROM).

De nombreuses études sur les méthodes biométriques sont en cours. Parmi celles-ci, nous trouvons :

1. Le spectre de la voix [Te95].

   Le porteur enregistre une phrase qui est par la suite échantillonnée et traduite en une série numérique. Lors de l'identification, le porteur répète le même texte. Cependant, un état de stress ou de maladie du porteur peut "perturber" la reconnaissance de la voix et provoquer des refus d'utilisateurs légitimes.
2. L'empreinte digitale [D89 , Pi95].

   Un dispositif de lecture optique permet de coder numériquement une empreinte digitale. Ce système a pour principale qualité sa fiabilité. En effet, le taux de rejet d'un utilisateur légitime est de 2 % (taches ou blessures sur la main peuvent être des causes de rejet), et le nombre d'autorisations non légitimes est de seulement 1 pour 10000.
3. Le schéma rétinien [E94].

   L'analyse rétinienne est réalisée par un faisceau optique. La signature référence occupe seulement 40 octets et l'algorithme de reconnaissance est aujourd'hui le plus fiable de toutes les techniques biométriques (99 %). Son seul défaut est le prix élevé du système de lecture optique (environ 7000 $).
4. La reconnaissance dynamique de signatures [PP88 , Le95].

   La signature référence est basé sur l'analyse dynamique de la signature (vitesse de déplacement, pression du stylet,...). La fiabilité de ce système est importante mais demande énormément de calculs.
5. La géométrie de la main [LP84].

   Des recherches montrent que les mains possèdent des caractéristiques uniques telles que la longueur des doigts, l'épaisseur... L'intérêt de cette méthode réside dans la taille de la signature. Ainsi, une société américaine RECOGNITION SYSTEMS a conçu un système d'identification basé sur cette méthode avec la signature référence stockée sur seulement 9 octets.
6. La frappe au clavier [M94].

   Il apparaît que la manière de taper les touches ou des séquences de touche est propre à chaque individu. La vitesse et l'accélération sont des facteurs caractéristiques. L'avantage majeur d'une telle étude est son faible coût. En effet, c'est le seul système d'authentification biométrique qui ne nécessite aucun capteur particulier. Le nombre d'autorisations non légitimes est d'environ 1 pour 100.
7. Le visage [Ya95].

Il apparaît que certains traits et contours caractéristiques sont spécifiques de chaque personne. La difficulté du système réside dans la distinction de l'angle d'approche de la personne par rapport au capteur.

1.10 Conclusion

L'étude des moyens mis en oeuvre visant à assurer la sécurité de la carte à microprocesseur est on le voit très diversifiée. Cependant même s'ils sont techniquement très pointus, ils ne comblent pas certaines lacunes. Par exemple, comme nous l'avons souligné, les techniques d'identification ne prennent pas en compte que la mémorisation d'un PIN code n'est pas chose aisée pour tout le monde et qu'un fort pourcentage de personnes ont inscrit ce code secret soit au dos de leur carte, soit à proximité (dans le portefeuille le plus souvent), mettant ainsi en péril le contenu de la carte. En effet un intrus possédant le mot de passe adéquat aura toute liberté d'utilisation. Par ailleurs, la présence d'un super utilisateur en qui nous devons léguer toute confiance affaiblit notoirement le potentiel sécurité de la carte. Ces détails cumulés mettent en relief certaines faiblesses que nous viserons par la suite à renforcer. Des travaux menés en parallèle à RD2P essaieront de pallier à ces inconvénients. Les résultats obtenus en ce qui concerne la reconnaissance biométrique et sur la détection d'intrusions dans la carte (c'est-à-dire des corrections par anticipation et par erreur sur les accès aux données de la carte) sont disponibles respectivement dans [A95, C94]. La présence d'un super utilisateur n'est pas souhaitable. L'idéal serait de pouvoir s'en passer sans pour autant figer les accès aux données une fois pour toutes. C'est un des points de départ de notre réflexion. Comment se passer de sa présence tout en laissant une souplesse d'utilisation en fonction des utilisateurs, des événements et de l'évolutivité des applications? Une solution sera abordée au chapitre 3 intitulé "un nouveau modèle de sécurité : le S-Shell" à la page 49, mais pour l'heure, nous allons entrer de manière plus poussée dans le domaine de la sécurité des systèmes informatiques en général afin, là encore, d'en souligner les points forts, l'objectif étant en fin de compte de pouvoir établir notre réflexion avec plus de certitude quant à l'approche de l'outil que nous voulons concevoir.

Chapitre 2

La sécurité des systèmes informatiques

2.1 Introduction

Nous voulons dans ce chapitre mettre en évidence une certaine hétérogénéité dans la façon dont a été abordée la sécurité dans les différents domaines de l'informatique. Nous envisageons d'orienter notre étude à la fois sur les systèmes d'exploitation, les systèmes de gestion de base de données (S.G.B.D) et les langages de programmation. Pour chacun d'entre eux, nous nous appliquerons à expliquer les modèles de description des politiques de sécurité [K95 , Wa95 , La95] et souligner les forces et faiblesses qui en résultent. Parallèlement nous montrerons que cette approche diversifiée de la mise en place de la sécurité dans les systèmes informatiques aboutit à un manque de cohésion. En guise d'introduction à la sécurité, l'article [SUV93] propose un survol intéressant de différentes notions importantes.

Nous voulons prouver que ce manque de cohésion est à l'origine d'une certaine fragilité et d'une trop grande rigidité des schémas de sécurité actuels des cartes. Pour conclure ce chapitre nous proposons une solution: l'outil S-Shell que nous décrirons longuement par la suite.

2.2 Les systèmes d'exploitation

Dans le cadre des systèmes d'exploitation, la sécurité concerne différents objets tels que la mémoire, des données partagées, des programmes et sous-procédures partagés, les entrées-sorties vers les périphériques (disques, imprimantes, etc ...), mais aussi et surtout les fichiers. Ces derniers peuvent contenir des informations confidentielles [E93] et par conséquent doivent pouvoir être protégés contre des accès non autorisés. C'est ce point précis que nous allons, entre autres, développer tout au long de cette partie. En effet, il concerne plus particulièrement le contrôle d'accès qui va être le thème central du travail accompli dans le cadre de cette thèse. Les autres cas énoncés dans cette introduction sont largement décrits dans [RR83]. Dans cet article, les auteurs élucident clairement les rôles qui doivent être attribués afin de protéger les différentes entités et comment le système d'exploitation

doit gérer ces problèmes. L'ouvrage [Pf89] au chapitre 6 offre une analyse intéressante de cette problématique de la protection de la mémoire.

2.2.1 Sécurité vs Protection

Bien souvent la frontière entre les définitions de sécurité et de protection est mal définie. Cependant il est primordial de bien séparer le problème du contrôle d'accès en lecture ou en écriture des fichiers et les mécanismes mis en oeuvre par le système d'exploitation pour assurer cette sécurité. Comme le conseille Andrew Tannenbaum dans [Ta89], nous désignerons par "sécurité" le problème dans son intégralité et par "mécanismes de protection" la mise en oeuvre par le système d'exploitation pour éviter la perte d'informations.

2.2.2 La sécurité

Le terme sécurité sous-entend différents aspects. Il est généralement divisé en deux parties bien distinctes, à savoir la sécurité physique et la sécurité logique. Les caractéristiques principales de la sécurité physique sont sans aucun doute:

- la perte de données qui peut être tout aussi bien due à des circonstances exceptionnelles (incendie, inondation, ...) qu'à des erreurs matérielles ou logicielles (processeur défaillant, disques illisibles, erreurs de transmission, ...) voire humaines (perte du disque, mauvaise saisie des données, ...). La plupart de ces problèmes peuvent être résolus en effectuant des sauvegardes que l'on stockera à l'écart des données originales
- l'intrusion qui, quant à elle peut être soit passive (cas de personnes cherchant uniquement à lire des fichiers auxquels elles n'ont normalement pas accès), soit active (cas d'une personne cherchant à modifier des données non autorisées). Nous ne nous attarderons pas sur les problèmes liés au phénomène de détection d'intrusion. Signalons juste que si l'on désire créer son propre système d'exploitation il convient, afin de prévenir ce genre d'attaques, de respecter certaines règles qui permettront de renforcer la sécurité du système [SS75]. Il faut généralement se doter d'une liste des failles et des "fuites" les plus courantes dans le but de connaître les traditionnels points faibles des systèmes d'exploitation et d'effectuer des tests de sécurité en faisant souvent appel à des équipes de pénétration [H80]

Les techniques de protection visant à parer les risques de destruction ou de détérioration du matériel ne nous concerne pas directement. Il est donc convenu que nous ne nous préoccuperons pas davantage de cette problématique. En revanche, la sécurité logique attirera tout au long de cet exposé notre attention. Dorénavant, par souci de clarté nous parlerons simplement de sécurité au lieu de sécurité logique.

2.2.3 L'identification de l'utilisateur

La reconnaissance ou la vérification de l'identité de l'utilisateur par le système lors de la connexion est la base de nombreux mécanismes de protection (Unix, Multics, ...). Le plus souvent le système se base sur une information que l'utilisateur détient (mot de passe, challenge, ...), sur un objet

qu'il possède (carte à puce, ...) voire sur une caractéristique physique (empreinte digitale, voix, ...). De multiples critiques fondées ont été faites sur ces systèmes d'identification (voir "Remarques sur les techniques d'identification" à la page 27). Nous ne les décrirons pas ici, ceci n'étant pas non plus l'objet de notre propos. Nous invitons le lecteur intéressé à se reporter à [MT79 , JO89] pour le mot de passe, à [HJ94] pour le challenge-réponse et à [C94 , S87] pour l'aspect biométrie. Certains renforcements du mot de passe par des considérations biométriques sont même proposés dans [ER95].

2.2.4 Les mécanismes et domaines de protection

Comme nous l'avons préalablement souligné, nous allons maintenant nous attacher à décrire les méthodes et techniques utilisées dans les systèmes d'exploitation pour assurer la protection des fichiers.

Chaque ordinateur possède un certains nombre d'objets à protéger. Ces objets (matériel ou logiciel) portent des noms qui permettent de les identifier. Ils sont de plus associés à un ensemble d'opérations qui peut leur être appliquées. Par exemple, pour un fichier on peut envisager des opérations telles que READ ou WRITE. Le but est par conséquent de pouvoir interdire aux utilisateurs d'accéder à des objets non autorisés. Plus finement on peut aussi envisager de restreindre uniquement le jeu d'accès à un objet. Typiquement un processus peut avoir un droit de lecture sur un fichier sans pour autant avoir un droit en écriture. L'idée finale est d'associer à chaque fois un objet à ses droits pour former un domaine. Le *domaine* est finalement un ensemble de paires (objet, droit). Attachons-nous dans un premier temps à éclairer ce propos à l'aide d'un exemple: le système d'exploitation Unix.

2.2.5 L'exemple d'Unix

Unix a été initialement développé de manière assez désordonnée sous l'égide de deux programmeurs. Son évolution s'est faite sans plan précis. Elle provient néanmoins de l'initiative d'abandonner le système Multics. La première version a été réalisée et produite par Thompson qui a par ailleurs suivi le projet pendant plus d'une décennie. Un des objectifs initiaux était de fournir une boîte à outils dans laquelle un utilisateur pouvait stocker et avoir accès à une série d'outils qui pouvaient être associés pour des besoins individuels. Deux de ses atouts majeurs sont la compacité et la généralité de ses fonctions. Si sa simplicité d'utilisation permet à tout un chacun de s'y retrouver aisément, il n'en est pas de même pour les concepteurs des schémas de sécurité. Chaque décision prise rend les commandes Unix plus perméables.

Il existe un utilisateur identifié sous le nom de super-utilisateur. Ceci sera d'ores et déjà une des critiques principales sur laquelle nous nous pencherons lors de la création de notre langage de sécurité (cf chapitre 3, paragraphe 2). Ce super-utilisateur est tout-puissant et peut effectuer n'importe quelle opération dans le système. De par sa suprématie, la plupart des attaques sur Unix visent à obtenir les droits du super-utilisateur. L'obtention de ces droits même pendant une très courte durée permet au fraudeur d'installer des trappes qui lui permettront de revenir à tout moment. D'où le réel danger de l'existence du super-utilisateur.

Sous Unix, la division en domaines est beaucoup moins précise que sous Multics. Le domaine d'un processus est déterminé par le couple (numéro d'utilisateur , numéro de groupe) ou encore (*uid* , *gid*). En partant de ce couple il devient possible de répertorier l'intégralité des objets auxquels un processus peut accéder. En effet, sous Unix tous les objets - répertoires, périphériques I/O, ... - sont accessibles par la même structure. On peut alors connaître l'ensemble des opérations effectuables sur ces dits objets (lecture, écriture, exécution). Là encore, sa simplicité de conception contraste avec la difficulté de sécurisation. La permission d'accès à un fichier est vérifiée une seule fois, quand le fichier est ouvert. Ainsi, en changeant les caractéristiques du fichier préalablement ouvert, un utilisateur peut avoir accès à des données interdites.

Chaque processus est séparé en deux parties, à savoir une partie utilisateur et la partie noyau. Lors d'un appel système par un processus, ce dernier commute de la partie utilisateur à la partie noyau au moyen d'un déroutement. Le système de fichiers appelle le pilote de périphérique comme une procédure. Ainsi donc, la partie noyau peut avoir accès à un ensemble d'objets différent de celui de la partie utilisateur.

Unix fournit un niveau de sécurité 'raisonnable' pour l'environnement pour lequel il a été conçu. Cependant, le manque évident de sécurité a conduit à une ré-écriture du noyau Unix afin d'aboutir à un système qui aurait les mêmes fonctionnalités externes qu'Unix, mais avec une structure interne différente. Le niveau de sécurité souhaité est une certification B2 du livre orange [DOD85] reconnue sous le nom de protection structurée. Sa difficulté d'obtention est discutée dans [Si87].

2.2.6 Vers des systèmes d'exploitation orientés sécurité

Le système d'exploitation décrit ci-dessus n'a pas, comme beaucoup d'autres d'ailleurs (VAX/VMS, VM/370 ou IBM MVS pourtant plus sûrs), été conçu dans un objectif sécuritaire. Il est supposé être installé dans des milieux relativement sûrs. En revanche, dans des environnements plus fragiles, il est évident que les requis de sécurité vont amener à mettre en place des systèmes d'exploitation plus fiables. Les caractéristiques principales de tels systèmes vont être de pouvoir assurer certaines fonctions de sécurité telles que:

- *Authentification des utilisateurs:* comme nous l'avons déjà souligné, le mécanisme le plus répandu est celui de la comparaison avec un mot de passe
- *Contrôle d'accès aux objets*: l'utilisation d'un ou de plusieurs objets par un utilisateur ne doit pas avoir d'effets pervers sur les autres utilisateurs
- *Protection mémoire*: de manière très générale, un programme utilisateur doit fonctionner sur une portion mémoire inaccessible aux utilisateurs non autorisés
- *Contrôle d'accès aux fichiers et périphériques*: le système doit protéger les accès non autorisés
- *Partage des ressources*: il faudra donc vérifier que les ressources sont bien partagées entre les utilisateurs appropriés. Ceci implique une garantie d'intégrité et de consistance

- *Disponibilité*: un utilisateur ne doit pas attendre trop longtemps avant de recevoir le service qu'il a demandé

Ce sont ces fonctions de sécurité qui vont être à l'origine de la conception des systèmes d'exploitation sécurisés. Il faudra néanmoins veiller à bien effectuer les spécifications adéquates au but recherché, car les aspects sécuritaires apparaissant à tout niveau dans le système, il devient très difficile de gérer l'évolutivité de ce dernier. Certaines inadéquations de nouvelles caractéristiques avec la politique de sécurité déjà adoptée peuvent nuire à l'introduction de nouveaux concepts dans le système d'exploitation [GM82]. Les principes fondamentaux n'en restent pas moins constants, à savoir:

- *Moindre privilège:* chaque utilisateur et chaque programme doivent disposer du privilège le plus petit possible afin de minimiser les tentatives d'attaques. Le modèle de sécurité militaire est basé sur ce principe. Chaque élément d'information appartient à un domaine: *non classé, confidentiel, secret, top secret*. Ces domaines disjoints sont ordonnés par ordre de sensibiliité croissante de l'information. Les utilisateurs ont uniquement accès aux données sensibles strictement nécessaires à leur besoin. Un système de cloisonnement par compartiment hiérarchique permet dans ce cas précis d'assurer ce principe du moindre privilège
- *Séparation des privilèges*: il serait souhaitable que chaque accès aux objets soit assujetti à plus d'une condition. Par exemple, une authentification d'utilisateur suivie d'une présentation de clé empêcherait l'intrusion par simple connaissance d'un code secret
- *Vérification totale*: chaque accès doit être minutieusement vérifié
- *Simplicité des mécanismes*: les mécanismes de protection doivent être suffisamment simples pour être facilement compris et testés. C'est ce qui est développé sous le nom de noyau sécurisé dans [Am83]
- *Conception publique*: les mécanismes de protection doivent être publics et ne reposer que sur le secret de quelques clés
- *Accès basé sur la permission*: on préférera une définition des objets accessibles plutôt que des objets inaccessibles. Les conditions par défaut seront préférentiellement des refus d'accès
- *Séparation des objets*: les objets partagés fournissent des canaux par lesquels les informations circuleront. Des systèmes permettant des séparations physique et logique réduiront considérablement les risques découlant de ces partages. C'est le problème connu sous le nom d'isolement

2.2.7 Illustration: la structure en anneaux de Multics

Le système d'exploitation Multics [S72] assure sa sécurité par une méthode connue sous le nom de structure en anneaux. Ces anneaux permettent de spécifier quels sont les droits d'accès à un processus. Un anneau est en fait un domaine sur lequel un processus s'exécute. Les anneaux sont numérotés en partant de 0, le dit 0 étant le niveau du noyau. Chaque processus tourne sur un niveau (ie anneau) donné. Ainsi plus le numéro du processus sera petit, plus ce dernier aura d'accès aux objets du système d'exploitation. Plus formellement, un processus tournant sur un anneau i, aura tous les privilèges des anneaux j, où $j > i$.

Ce système d'exploitation utilise une forme de liste de contrôle d'accès dans laquelle chaque utilisateur appartient à trois classes de protection: *utilisateur, groupe, compartiment.* La désignation de l'utilisateur identifie un sujet particulier. La désignation du groupe permet de rassembler des utilisateurs ayant des intérêts communs. Le compartiment est utilisé afin d'isoler des objets en lesquels nous n'avons pas confiance.

Plus théoriquement, chaque zone de données est appelée un segment protégé au moyen de trois classes notées *b1, b2, b3*. La relation d'ordre entre ces trois classes est $b1<=b2<b3$. Les termes utilisés sont: support d'anneaux pour la notation *[b1, b2, b3]*, support d'accès pour *(b1, b2)* et support d'appel pour *(b2, b3)*. Ainsi, tous les éléments compris entre *b1* et *b2* représentent les processus d'anneaux pouvant accéder librement au segment. De la même manière, tous les éléments compris entre *b2* et *b3* repésentent les processus d'anneaux pouvant accéder au segment seulement en des points d'entrée précis.

Ainsi lorsque qu'un processus p, s'exécutant à un niveau k, désire appeler une zone de données ayant pour support d'anneaux *[b1, b2, b3]*, quatre cas de figure se présentent:

- Si $k < b1$: l'appel est acceptable, mais p exécute seulement une copie du processus désiré. L'accès se limite à un accès en lecture
- Si $b1 <= k <= b2$: l'appel s'effectue normalement
- Si $b2 < k <= b3$: p peut exécuter au mieux une copie du processus, seulement si l'appel est effectué au bon point d'entrée
- Si $k > b3$: l'appel n'est pas acceptable

De cette manière, un programme travaillant dans un compartiment donné ne pourra pas accéder aux objets d'un autre compartiment s'il ne détient pas une autorisation spécifique. Il s'agit ici d'une implantation de sécurité minimale. Des solutions existent afin de rendre l'utilisation de cette étroite représentation plus souple. Nous ne les détaillerons pas ici car seules les mécanismes de base nous intéressent. Le niveau de sécurité TCSEC (Trusted Computer System Evaluation Criteria) [DOD85] obtenu par le système Multics d'Honeywell est B2 (voir à ce propos le paragraphe suivant).

A titre indicatif, citons quelques exemples de systèmes d'exploitation orientés sécurité. Nous avons en tête de liste le Honeywell Scomp (qui fût le premier à avoir été classé A1), le UCLA Secure Unix, avec sa structure à trois niveaux, que nous avons déjà mentionné [Po79] et enfin le KVM/370.

2.2.8 Modèles de sécurité et certifications

En résumé, le point fondamental soulevé tout au long de cette discussion est que l'aspect fondamental à respecter dans toute sécurisation de système informatique est le contrôle d'accès aux données (confidentialité, intégrité et disponibilité). Ainsi, avant de commencer à concevoir un système d'exploitation, il est important que le concepteur bâtisse un modèle de sécurité de l'environnement à protéger et étudie différentes façons de mettre en oeuvre la sécurité dans ces modèles.

Fort heureusement, il existe déjà de nombreux et performants modèles sur lesquels on peut, en fonction de ses besoins, se baser. Nous ne nous attarderons pas sur ces points, mais à titre indicatif, nous voulons quand même les citer et donner quelques références auxquelles le lecteur pourra se reporter. Les modèles les plus courants sont:

- Le plus simple est le modèle du *moniteur*. Ce moniteur est une porte entre l'utilisateur et l'objet. L'utilisateur envoie une requête. Le moniteur consulte alors son registre de contrôle d'accès et permet ou interdit l'accès en question. Cette idée sera d'ailleurs reprise dans son esprit dans notre travail ultérieur (chapitre 3, paragraphe 1). On peut, sur ce sujet, se reporter à [GD72]
- Les limites de ce système ont été corrigées dans le modèle ultérieur dit du *flux d'information*. Là encore Denning semble être une référence appropriée [D76]
- Les deux modèles précédents, la décision finale d'accès à un objet se traduisait uniquement par une acceptation ou un refus. Une sensibilité plus fine aussi bien au niveau des utilisateurs que des objets est proposée dans le modèle connu sous le nom de *sécurité d'accès du modèle lattice* [D76]
- Citons enfin quelques autres modèles de notoriété internationale tels que le modèle de Bell-La Padula [BL73], le modèle de Biba [Bi77], le modèle de Graham-Denning [La74, GD72], le modèle de Harrison-Ruzzo-Ullman [HRU76] et enfin le système Take-Grant [Sn81]

Afin de classer les niveaux de sécurité de ces systèmes d'exploitation, un certain nombre de méthodes ont été mises en oeuvre. Elles vont de la vérification formelle (le système d'exploitation est réduit à un théorème et prouvé) à la validation (qui est plus générale que la vérification) en passant par les tests des équipes de pénétration. Tout ceci aboutit à une certification NCSC - américaine - du [DOD85] - le livre orange - qui ordonne ces niveaux de sécurité du moins sûr au plus sûr (D, C1, C2, B1, B2, B3, A1). Une certification européenne existe également. Ce sont les ITSEC. On trouvera par ailleurs dans [IT91] une liste des correspondances entre ces deux critères de certification. Ces critères de sécurité sont aussi applicables pour d'autres systèmes informatiques.

2.2.9 Conclusion sur les systèmes d'exploitation

Les systèmes d'exploitation doivent être capables de gérer le paradoxe d'aisance d'utilisation et de robustesse. Les règles à respecter visant à définir un système d'exploitation sécurisé sont de plusieurs types. L'environnement à protéger doit être clairement défini de façon à ce que les interactions entre les composants essentiels du système d'exploitation soient bien définies et de manière sécurisée. Cette sécurisation vérifie certains principes classiques tels que celui du moindre privilège. De l'expérience de cette étude, nous tenons à retenir certains principes fondamentaux à respecter lors de l'élaboration de notre outil. Par exemple, Unix a montré certaines limites, l'une d'entre elles étant l'existence même du super-utilisateur (paragraphe 2.5 de ce chapitre). Nous avons du reste bien insisté sur les inconvénients de sa présence dans un système d'exploitation. C'est pourquoi, nous allons nous efforcer lors de la description du cahier des charges de notre futur S-Shell (ce nom sera expliqué par la suite), à nous passer d'un tel utilisateur surpuissant et par là même cible d'une majo-

rité des attaques. Il est bien évident qu'il faudra pallier sa disparition et que cela rendra le schéma plus statique dans le temps. Cette limitation ne cachera pas l'étendue du gain en sécurité que nous obtiendrons alors. Néanmoins il semble déjà important de savoir que ce point faible du système Unix entre autre sera corrigé d'une certaine façon par notre travail.

Par ailleurs, certaines autres priorités à respecter nous ont été apportées grâce à l'étude des systèmes d'exploitation orientés sécurité, comme par exemples, l'accès basé sur la permission ou le contrôle minutieux d'accès aux fichiers. L'étude de la sécurité (ou du manque de sécurité) des systèmes d'exploitation a eu pour effet immédiat de mettre non seulement en évidence certaines faiblesses que ceci pouvaient présenter mais aussi et surtout de nous mettre en harmonie avec les notions fondamentales à suivre afin d'assurer un niveau de sécurité maximal dans la mise en oeuvre de notre prototype.

La démarche suivante vise cette fois les S.G.B.D (Système de Gestion de Base de Données). Nous espérons non seulement en tirer des enseignements identiques à ceux obtenus grâce aux systèmes d'exploitation mais aussi par la suite bénéficier d'une comparaison intéressante entre les atouts des uns et des autres. Mais n'anticipons pas et commençons par l'étude de la sécurité des S.G.B.D.

2.3 Les systèmes de gestion de base de données

2.3.1 Objectifs du chapitre

Dans cette section, nous allons aborder les aspects sécuritaires présents dans les systèmes de gestion de bases de données. Nous commencerons par un bref rappel de la terminologie utilisée afin d'être en accord sur les termes que nous allons employer. Nous poursuivrons par l'étude des problèmes majeurs de sécurité - intégrité et confidentialité - qui apparaissent dans le contexte des bases de données. La dernière partie mettra enfin en évidence un aspect véritablement complexe, à savoir les problèmes d'inférence.

2.3.2 L'utilisation des S.G.B.D

Notre étude se penchera tout particulièrement sur les bases de données dites relationnelles, qui sont les plus couramment utilisées. Pour les lecteurs qui seraient intéressés par une lecture plus approfondie sur les bases de données, nous pouvons conseiller la référence [Ul82]. Une base de données est en définitive un ensemble de données associé à un ensemble de règles, ces dernières permettant d'organiser les données en spécifiant certaines relations. Cette collection de données est stockée dans un central accessible à tous les utilisateurs. L'utilisation de ces bases de données permet en outre de:

- *disposer d'un accès partagé*: chaque utilisateur peut se référer à un ensemble de données centralisées
- *de minimiser la redondance des données*: chaque individu n'est pas obligé de stocker ses propres informations
- *d'assurer l'intégrité des données*: les valeurs des données sont protégées contre des changements accidentels ou malicieux

- *d'assurer la consistance des données*: le changement d'une valeur d'une donnée affecte tous les utilisateurs de cette donnée
- *d'assurer le contrôle d'accès*: seuls les utilisateurs autorisés ont un droit de lecture ou de modification sur des valeurs de données

Un système de gestion de base de données doit se conformer efficacement à ces principes. Bien souvent, les mesures visant à renforcer la sécurité de ces bases de données augmentent la complexité du système. Ainsi donc, il faudra résoudre le conflit entre niveau de sécurité et performance du système. Il faudra de toutes façons répondre à un certain nombre de critères sécuritaires que nous nous proposons d'énumérer dans le paragraphe suivant.

2.3.3 Les besoins de sécurité

Les besoins de sécurité relatifs aux S.G.B.D diffèrent notoirement de ceux évoqués dans le contexte des systèmes d'exploitation. Certains problèmes comme le contrôle d'accès ou l'authentification des utilisateurs ne sont pas nouveaux [HT95]. D'autres, en revanche sont typiquement liés aux bases de données. Nous nous proposons de les énumérer et de détailler leurs vocations:

- *Intégrité physique de la base de données*: une donnée doit être protégée contre des problèmes d'ordre physique (coupures de courant, destruction du disque, ...). Il sera ainsi possible de se prémunir contre une destruction éventuelle de la base de données. La méthode employée est la création de copies des données de tous les fichiers du système. Il est fondamental qu'une base de données puisse être reconstruite à partir du potentiel point d'erreur
- *Intégrité logique de la base de données*: la structure de la base de données est ainsi préservée. Par exemple, grâce à l'intégrité logique de la base de données, la modification d'une valeur d'un champ de la base n'affectera pas les autres
- *Intégrité des éléments*: la donnée contenue dans l'élément est ainsi correcte. Les utilisateurs se doivent de mettre des données correctes dans leurs bases. Néanmoins des erreurs peuvent survenir. Le S.G.B.D doit pouvoir être capable de détecter ces erreurs et de les corriger. Il y a trois manières de maintenir l'intégrité d'une base de données. La plus simple est la vérification de champs. On vérifie à un endroit donné que la valeur introduite est conforme au type requis. L'intégrité est aussi assurée par le contrôle d'accès. Ce point est explicité ci-dessous. Enfin, on peut maintenir l'intégrité en gardant une liste de tous les changements qui ont été effectués dans la base de données. Une erreur peut alors être corrigée par l'administrateur par une commande d'annulation de l'opération à l'origine du conflit
- *Contrôle d'accès*: seules les personnes autorisées peuvent accéder aux données avec, éventuellement, certaines restrictions (lecture seule, écriture seule, ...). C'est l'administrateur de la base qui spécifie ces contrôles d'accès à un champ ou à un enregistrement. Alternativement ce peut aussi être le propriétaire d'une table. En superficie, le contrôle d'accès dans le contexte des S.G.B.D ressemble beaucoup à celui des systèmes d'exploitation. Cependant il faut remarquer que, pour les systèmes d'exploitation les éléments à contrôler sont indépendants (objets, fichiers, ...) alors que pour les S.G.B.D les champs peuvent être liés. Il faut donc assu-

rer qu'en lisant une donnée autorisée, un utilisateur ne puisse pas en déduire des données protégées. Ce problème est connu sous le nom d'inférence. Il ne faut donc pas seulement protéger la donnée elle-même mais assurer une protection des chemins qui pourraient y mener, sans pour autant empêcher l'accès aux données autorisées. Il faut donc établir un compromis entre mettre en oeuvre une trop restrictive politique d'accès qui gênerait la bonne utilisation de la base de données et assurer malgré tout un niveau de sécurité suffisant

- *Possibilité d'audit*: on peut ainsi savoir qui a accédé aux données. Ceci peut aider à maintenir l'intégrité de la base. De plus en cas de modification non justifiée de certaines données, il est alors possible de savoir qui aurait pu effectuer ces altérations et quand. En corollaire de la remarque effectuée au point précédent sur les chemins d'accès, on peut grâce à l'audit voir si certaines personnes n'essaieraient pas, par recoupements successifs d'obtenir certaines informations confidentielles [FW72]
- *Authentification de l'utilisateur*: elle peut être propre à la base de données. Cette authentification sert et pour l'audit et pour les restrictions d'accès aux données de la base. Cette authentification peut être ajoutée à celle requise par le système d'exploitation. Ainsi le S.G.B.D se comporte comme un programme d'application se situant au-dessus du système d'exploitation. Il n'y a donc aucun chemin de confiance vers le système d'exploitation. Toutes les données reçues sont, par défaut, douteuses y compris l'authentification de l'utilisateur par le système d'exploitation
- *Disponibilité*: les utilisateurs peuvent effectivement avoir accès à toutes les données auxquelles ils sont autorisés. Un des problèmes classiques est d'assurer deux requêtes de deux utilisateurs sur le même élément en évitant d'éventuels conflits d'opérations. Un autre problème courant est d'empêcher de dévoiler des données non-protégées de façon à révéler des données protégées

En conclusion, nous pouvons affirmer que nous avons dans cette énumération non-exhaustive, mis en évidence un groupe de trois problèmes de sécurité, majeurs dans le monde des bases de données. L'intégrité des données qui peut aussi bien concerner des éléments individuels d'une base que la base dans son ensemble. La confidentialité qui devient un élément primordial du fait des problèmes d'inférences et donc de l'accès indirect aux données. La disponibilité qui permet un accès partagé aux données. Ces trois points vont maintenant être un peu plus approfondis.

2.3.4 Mécanismes d'assurance de confiance et d'intégrité des données

Pour résumer, on conçoit donc qu'une base de données soit un rassemblement de données provenant de différents horizons. Les utilisateurs d'une base de données sont en droit de douter de la valeur de leurs données. A contrario, ils sont aussi dans l'expectative que des moyens seront mis en oeuvre afin de protéger ces dites données contre la perte ou la déformation. C'est pourquoi, plus que dans aucun autre système informatique, les notions de confiance et d'intégrité des données prennent une place prépondérante dans le monde des S.G.B.D. Il est de coutume de discerner à ce propos trois niveaux d'intégrité des données:

- *L'intégrité de la base de données:* cette notion concerne principalement les éventuelles dégradations qui peuvent à tout moment survenir. Les solutions courantes envisagées s'appuient sur des contrôles d'intégrité effectués par le système d'exploitation ainsi que sur des procédures de récupération des données
- *L'intégrité des éléments de la base de données*: cette notion concerne les modifications non autorisées d'éléments de la base de données. Les moyens mis en oeuvre pour contrer ces phénomènes sont basés sur le contrôle d'accès aux données
- *L'exactitude des éléments de la base de données*: cette notion concerne la justesse des valeurs qui sont inscrites dans les éléments de la base de données. Des méthodes visant à détecter d'éventuelles valeurs incorrectes - conditions de contrainte, vérification de valeurs, ... - existent et sont largement employées

Afin de respecter ce cahier des charges, il existe différents mécanismes qu'il convient d'étudier, toujours dans l'expectative de déceler les éventuelles faiblesses et de mettre en évidence les points forts qui pourraient nous être utiles dans notre projet.

2.3.5 Techniques de maintien d'intégrité et de confiance

- *Assurance de protection de la part du système d'exploitation:* vis-à-vis du système d'exploitation, les fichiers de la base de données bénéficient des mêmes compétences de contrôle d'accès. Certaines vérifications sommaires d'intégrité sont également introduites, mais c'est surtout la base de données elle-même qui doit surveiller ce genre de mécanisme
- *Technique de mise à jour*: c'est le commit à deux phases. La première phase est la phase intentionnelle durant laquelle aucune opération irréversible n'est enclenchée. C'est une phase de préparation pendant laquelle la base de données récupère l'ensemble des données ou informations dont elle aura besoin pour effectuer la mise à jour. Cette opération peut être répétée autant de fois que l'on veut et une éventuelle interruption en cours de première phase ne serait en aucun cas dramatique pour l'intégrité de la base de données. La dernière étape de cette première phase est le passage par un point de non-retour: l'indicateur de commit. La seconde phase initie les changements permanents. Aucune des opérations de la première phase ne pourra alors être effectuée durant cette deuxième phase. En revanche les activités de mise à jour de la seconde phase peuvent elles être répétées autant de fois que nécessaire. En cas d'échec pendant la deuxième phase, la base peut alors contenir des informations incomplètes, mais pouvant être repérées en renouvellant toutes les activités de la deuxième phase. Une fois achevée, la base de données est de nouveau intègre
- *Redondance d'informations*: un nombre de base de données considèrent des surplus *a priori* inutiles d'informations afin de pouvoir dans certains cas détecter certaines incohérences. Un exemple pourrait être la détection d'erreur et la correction de codes. A chaque ajout ou retrait d'informations, une comparaison avec ce code est réalisée. Si une différence lors de la comparaison est décelée, on sera certain qu'une erreur est apparue dans la base de donnée. Certains codes permettent même de repérer à quel endroit l'erreur s'est produite. D'autres permettent directement de corriger cette erreur. Il faut juste savoir qu'au plus cette technique est puissante

dans la détection-correction d'erreurs, au plus elle prendra de place pour stocker les codes. Dans le même concept d'idées, des champs entiers peuvent être reproduits dans la base. Cela peut être utile, en cas d'erreurs dans l'original. Evidemment, restent toujours le problème d'espace requis et les problèmes liés à la redondance d'informations

- *Récupération*: en sus des techniques de redondance, la base de données peut aussi maintenir un registre des accès et plus précisément des changements effectués. En cas d'erreur, la base de données sauvegardée peut être chargée en lieu et place de la base erronée
- *Consistance/Concurrence*: ce point concerne l'accès multiple à la base. Il ne faut pas que deux utilisateurs sur les mêmes données interfèrent sur celles-ci. Le S.G.B.D bloque souvent l'élément sensible de façon à éviter les conflits
- *Moniteurs*: un moniteur est une unité d'un S.G.B.D responsable de l'intégrité structurelle de la base. Un moniteur peut vérifier les valeurs entrées afin d'assurer leur consistance avec le reste de la base de données ou avec les caractéristiques d'un champ particulier. Les trois méthodes suivantes sont quelques exemples de ce que l'on peut faire
- *Champ des valeurs*: ce moniteur compare chaque valeur entrée afin que ces nouvelles valeurs appartiennent à un ensemble acceptable. Si la valeur ne répond pas à cette condition, elle ne pourra être acceptée par la base de données
- *Contraintes d'état*: elles concernent l'état de la base dans son ensemble. Ainsi si elles ne sont pas, à tout moment, vérifiées, une erreur pourra survenir
- *Contraintes de transition*: contrairement aux précédentes, ces contraintes s'appliquent aux transitions d'un état à un autre. Les transitions doivent respecter certaines conditions. Ces trois derniers aspects sont d'une importance capitale et réfèrent au problème de la vérification d'intégritré que nous étudierons en détail au chapitre 4

2.3.6 Données sensibles et problèmes d'inférence

Nous définissons par donnée sensible, une donnée dont l'accès est protégé, *ie* dont l'accès n'est pas public. La décision de rendre une donnée sensible dépend uniquement du concepteur de la base. Ainsi une base peut être entièrement publique, partiellement sensible (seules quelques données de la base sont sensibles) ou entièrement sensible. Les cas extrêmes sont, on le comprend aisément, les plus simples à traiter. L'hétérogénéité du cas médian pose un certain nombre de problèmes. Il s'agit là pourtant du cas le plus intéressant du point de vue sécurité. On peut même affiner le concept en disant qu'il peut coexister dans la base plusieurs degrés de sensibilité.

Un certain nombre de techniques existent afin de définir ces niveaux de sensibilté. On trouvera notamment dans [DS83], une excellente analyse comparative de ces méthodes. Le problème d'inférence, dont nous allons maintenant parler y est aussi abordé.

Le problème d'inférence est un problème sournois, car il consiste à l'aide de données publiques ou non sensibles à déduire des valeurs de données sensibles. Ce problème est l'un des sujets de vulnérabilité des bases de données. Il existe là aussi de nombreuses méthodes qui permettent d'aboutir au résultat recherché (par addition, par comptage, etc ...). Quoique largement explorée cette recherche ne nous concerne pas directement. Nous nous contenterons donc de ne citer que quelques ouvrages de références. En plus du précédemment cité [DS79], bien qu'assez ancien, dresse un portrait intéressant des méthodes suivies par un attaquant potentiel. En résumé, les trois voies à suivre pour minimiser les risques sont:

- *Suppression des données sensibles*: facile à réaliser, mais cette tendance naturelle a le défaut d'amoindrir l'utilité de la base de données
- *Suivi des attaquants*: on maintient toutes les informations concernant tous les utilisateurs. Outre la difficulté d'un tel procédé, il faut mentionner le coût important à accorder à une telle recherche. Le point faible de ce procédé est qu'on peut difficilement faire le recoupement de ce que peuvent savoir plusieurs personnes associées
- *Cacher des données*: cette notion consiste à fournir, par des pertubations aléatoires, des informations inconsistantes ou incorrectes

2.3.7 Etude d'un exemple: SQL (Structured Query Language)

Ce langage de gestion de base de données a été adopté par l'Association de Normalisation Américaine (ANSI, American National Standards Institute) comme norme internationale par l'ISO. Ce paragraphe est destiné à montrer comment les attributs relatifs à la sécurité sont gérés. Le vocabulaire adapté à SQL ou plus généralement aux bases de données (table, ...) sera supposé connu. Une référence appropriée [Bo88 , Da89] permettra au lecteur de mieux s'orienter dans ce domaine.

Nous allons dans un premier temps parler de la manipulation de données sans curseur en considérant pour simplifier l'exposé que toutes les tables sont des tables de base. Ces opérations sont de quatre ordres: SELECT, INSERT, UPDATE (avec recherche) et DELETE (avec recherche). L'ordre SELECT permet de sélectionner tout ou une partie d'éléments situés dans une liste d'expressions scalaires à partir d'une table. En SQL normalisé la syntaxe générale du SELECT est:

SELECT [ALL | DISTINCT] sélection

INTO liste-virgule-de-cibles expression-de-table

Par défaut, l'omission de DISTINCT ou de ALL équivaut à un ALL. De la même manière, la commande INSERT permet d'ajouter de nouvelles lignes dans une table. La syntaxe de cette commande est:

INSERT INTO table [(liste-virgule-de-colonnes)] source

Ici encore l'omission de la liste revient par défaut à spécifier toutes les colonnes de la table. Le terme source représente soit une question du type SELECT, soit une clause VALUES du type: VALUES (liste-virgule-d'atomes-d'insertion). Cette dernière liste peut être une valeur nulle ou un paramètre. La commande UPDATE comme son nom l'indique, permet de mettre à jour les lignes d'une table sans utilisation de curseur. En général la mise à jour s'effectue sur plusieurs lignes. On peut donc ainsi modifier un nombre quelconque de lignes en une seule opération. Enfin, la commande DELETE permet, elle, de supprimer les lignes dans une table sans utiliser de curseur. Dans ce cas aussi plusieurs lignes peuvent être supprimées à la fois.

Par ailleurs en SQL, nous bénéficions aussi de l'existence de tables vues. Nous savons déjà que contrairement aux tables réelles dont les lignes et les colonnes sont physiquement stockées sur un support magnétique, il existe des tables dites virtuelles, c'est-à-dire qui apparaissent comme étant réelles aux yeux de l'utilisateur bien qu'elles n'aient pas d'existence propre. Une vue est définie en fonction d'autres tables et est mémorisée dans le catalogue système dont la notion ne fait pas partie de la norme SQL proprement dite. Une définition de vue contient une expression SELECT précisant l'étendue de la recherche. Ce SELECT n'est pas évalué au moment de la définition mais simplement mémorisé avec la vue dont le nom a été spécifié. Grâce aux vues, une table ne contenant qu'une partie des lignes et colonnes d'une table réelle paraît ainsi réellement exister. Nous disposons alors de processus de traduction des quatre ordres énumérés précédemment afin que l'on puisse aussi utiliser ces ordres SQL dans le cas des tables vues. Nous tairons les opérations d'extraction, de mises à jour avec option de vérification, etc ... afin de ne pas alourdir l'exposé.

Enfin, afin de bien gérer les contrôles d'accès à la base de données, il existe un DBA (Data Base Administrator) qui reçoit pleine autorité sur les bases de données relationnelles. Les principaux privilèges du DBA sont CONNECT et RESOURCE. Le premier reconnaît le droit d'utiliser le système, et le second autorise la personne à créer des tables dans des DBSPACE PUBLIC et d'acquérir des DBSPACE PRIVATE.

Ainsi le DBA peut autoriser par l'ordre GRANT un utilisateur à se connecter, à créer des tables dans un espace public ou à créer un espace privé pour un utilisateur. Il existe trois formats pour l'instruction GRANT. On peut affecter le GRANT pour des tables ou des vues (format 1), pour des autorisations d'exécution (RUN) (format 2) ou pour des autorisations spéciales (format 3). Dans ce dernier cas, il s'agit pour le DBA d'attribuer des privilèges particuliers à d'autres utilisateurs. C'est le problème de la délégation de droits que l'on traitera plus particulièrement au chapitre 5. Ceci ne peut en aucun cas être fait dans un programme.

Il peut également supprimer ces autorisations par l'ordre REVOKE. Les formats d'instruction du REVOKE sont à peu près identiques à ceux de GRANT. Le principe général est que l'on ne peut révoquer ses propres autorisations et que l'instruction REVOKE n'est pratiquement utilisable qu'avec des utilitaires particuliers SQL.

2.3.8 Conclusion sur les systèmes de gestion de base de données

Cet exposé a mis clairement à jour deux points essentiels concernant la sécurité des bases de données: d'une part le respect de l'intégrité des données et le contrôle d'accès. D'autre part, en ce qui concerne le contrôle d'accès, les mécanismes présents dans le milieu des bases de donnée ressemble étrangement à ceux des systèmes d'exploitation. L'étude a donc mis en évidence une certaine homogénéité des caractéristiques de sécurité entre les bases de donnée et les systèmes d'exploitation. En revanche, les phénomènes liés à l'inférence restent typiques des bases de données.

2.4 La sécurité dans les langages de programmation

Dans la culture informatique, il est peu courant de lier les préoccupations sécuritaires et la conception des langages de programmation. La programmation est en effet une démarche essentiellement constructive et on imagine assez peu intégrer, dans cette activité, la nécessité de se protéger d'individus malveillants. Nous commencerons donc par expliquer l'objectif de ce chapitre. Nous dirons alors comment se transposent les notions d'utilisateur, d'objet et d'opérations dans ce contexte.

La suite du chapitre décrira comment ce problème a été appréhendé par différentes générations de langages de programmation. Cette étude historique montrera les modèles de sécurité induits par ce point de vue "programmation". Nous détaillerons alors les outils nécessaires à la mise en oeuvre de ce modèle tout au long de la chaîne de développement du programme (compilation, exécution, support du matériel).

En conclusion nous présenterons les aspects originaux de ce modèle par rapport aux deux systèmes vus précédemment et nous en déduirons certaines fonctionnalités désirables pour notre modèle idéal de sécurité.

2.4.1 Justification de ce chapitre

Nous replaçons ici le problème général de la sécurité dans le contexte de l'activité de programmation. Il est apparu, au fil des ans et au fur et à mesure de l'accroissement de la complexité des ensembles logiciels produits, que la maîtrise de cette complexité passait impérativement par un découpage propre de la conception et de la programmation. Ce découpage permet de confier la réalisation de ces morceaux à des équipes ou des individus suffisamment autonomes pour limiter le surcoût de communication à des valeurs acceptables.

Ces techniques de modularité ont rejailli sur les outils de base que sont les langages de programmation. Les langages actuels permettent en effet la description puis l'utilisation d'un module/objet[1]

1. Dans ce chapitre nous appellerons module un ensemble logiciel autonome groupant des données (ou variables) et des fonctions (ou méthodes). Pour ce qui nous concerne ici, la programmation orientée objet n'apporte rien au discours

par un autre module. Le corollaire à cet état de fait est qu'il est nécessaire de limiter les interactions accidentelles entre deux modules indépendants. Par exemple par le choix malencontreux d'un même identificateur dans des modules distincts.

En résumé, nous voyons apparaître le besoin d'effectuer un contrôle d'accès aux données d'un module en vue d'assurer le bon fonctionnement de celui-ci. Les demandeurs et les propriétaires des données sont des programmes. Nous considérerons ici les outils et modèles qui permettent cette limitation.

2.4.1.1 Objectifs des règles de portée

Lors de la rédaction d'un programme, le programmeur utilise essentiellement des données et des fonctions qui peuvent être fournies par le langage utilisé ou qu'il peut avoir défini lui-même. Il se construit ainsi un environnement dans lequel chaque identificateur correspond à une donnée ou à une opération qu'il utilise. Lorsqu'il met son programme à disposition d'autres utilisateurs (programmeurs-programmes), le rédacteur le fait en "publiant" quelques unes de ses variables ou fonctions. Les règles de visibilité des langages de programmation déterminent sous quelles conditions un nom défini en un lieu d'un logiciel peut être utilisé ailleurs dans ce logiciel. Dans les langages les plus récents, il est même possible de masquer des fonctions ou des opérateurs définis par le langage puis d'en interdire l'utilisation aux autres programmeurs.

2.4.1.2 Transposition du problème

Il est intéressant d'identifier les différents éléments de cet aspect des choses par rapport au problème classique de la sécurité dans les systèmes. Là encore, nous pourrons distinguer les objets sur lesquels porteront les droits, les acteurs du système et les opérations qui seront contrôlées par les mécanismes de protection.

- Les objets à protéger

Les objets à protéger sont essentiellement des variables car ce sont les seuls éléments dont la modification peut entraîner un dysfonctionnement du système. Il est de - relativement - peu d'intérêt de protéger des constantes ou des types car ces objets statiques n'existent que durant la compilation. Notons toutefois que certaines constantes encombrantes comme les chaînes de caractères peuvent être rangées en mémoire au même titre que les variables et demandent alors certaines protections (accès en lecture seule).

Les fonctions peuvent également être réservées à une utilisation locale. Notons à ce propos qu'il existe un seul type d'accès à une fonction: l'appel[1].

- La notion d'utilisateur dans ce contexte

1. Notons qu'il existe aussi la référence à une fonction (passage en paramètre par exemple)

Dans ce schéma, il n'y a pas d'identification personnelle du programmeur; Le fait que M. Georges ait écrit à la fois le composant logiciel X et le composant Y n'a aucune influence sur la protection des objets au sens où nous l'entendons. D'ailleurs les environnements de programmation ne demandent aucune identification par eux-mêmes. En revanche, lors d'un accès, il est essentiel pour le compilateur d'identifier l'entité logicielle (bloc, procédure ou module) responsable de l'accès et l'entité logicielle propriétaire de l'objet.

Le prolongement naturel de cette considération existe dans les systèmes d'exploitation sous la notion de processus: deux processus peuvent avoir un espace de données et des privilèges différents tout en ayant le même propriétaire.

- Les opérations contrôlées

Nous avons déjà remarqué qu'un seul type d'accès existait sur une fonction: l'appel[1]. Les variables quant à elles peuvent être soumises à de nombreuses opérations. Nous regrouperons pratiquement ces opérations en deux groupes:

Les opérations créées par le programmeur pour le type abstrait que représente la donnée. Si le programmeur écrit le type de données "pile", il peut et doit simultanément définir les opérations "empiler" et "dépiler" et les proposer à tout utilisateur de sa pile. Mais il peut aussi, pour des raisons de commodité interne écrire la procédure "vider la pile" en voulant s'en réserver l'utilisation. Ces besoins sont satisfaits par les règles de visibilité.

Les opérations fournies par le langage soit sur l'objet global (comparaison, affectation) soit sur la réalisation physique de cet objet (indexation pour les tableaux, déréférence pour les pointeurs, etc.) . Ces opérations ont une syntaxe définie par le langage. La restriction de ces opérations ne peut donc reposer sur le fait que le "fraudeur" ne connaît pas la syntaxe d'appel.

2.4.2 Différentes générations de langages

Pour étayer ce discours, nous allons exposer comment trois langages de programmation de générations différentes ont traité le problème de conception et d'écriture de gros logiciels.

2.4.2.1 Le langage Basic: tout est possible

Le niveau le plus bas de la sécurité est l'absence de sécurité. Le langage BASIC n'est présenté que comme langage d'apprentissage de la programmation et ses premières versions ne structurent ni le programme lui-même ni les données du programme. Tous les traitements sont possibles sur toutes les variables. La seule restriction apportée par le langage est liée à des problèmes de représentation des

1. Nous négligerons les techniques de passage de fonctions en paramètre ou similaires qui sont régies par les mêmes règles de visibilité.

données et donc au typage. Le type d'une donnée est caractérisé par un caractère suffixe dans le nom de la variable. Le système contrôle simplement la compatibilité des opérations demandées avec le type de la variable (par exemple pour interdire des opérations chaînes sur des nombres). De fait, le langage BASIC initial ne supporte pas la programmation de gros projets.

2.4.2.2 Les langages Pascal et C

Le langage Algol d'abord, puis Pascal et le langage C ont commencé à prendre en compte le besoin de maîtriser la complexité des programmes en les structurant. La base essentielle de ces structures est le bloc qui peut être anonyme (le bloc représente alors une "région" du programme) soit identifié comme procédure ou fonction. Les blocs pouvant être imbriqués sans chevauchement sur un nombre quelconque de niveaux, il s'ensuit une structure hiérarchique du programme qui est donc vu comme le bloc unique qui contient tous les autres blocs.

- Règles de portée et de visibilité

Les règles de portée et de visibilité utilisent uniquement cette structure arborescente: Une variable du programme est visible depuis tous les blocs internes au bloc où a lieu sa déclaration. Deux blocs frères peuvent ainsi utiliser les mêmes noms pour des variables différentes sans risque de conflit. L'interaction entre deux blocs n'est alors possible que par l'intermédiaire de paramètres ou par l'utilisation conventionnelle d'une ou de plusieurs variables globales communes.

La protection apportée par ce modèle est sommaire car elle repose uniquement sur la connaissance ou l'ignorance de la donnée protégée; si la donnée est visible, toutes les opérations visibles -en particulier les opérations du langage (affectations voire opérations arithmétiques)- sont autorisées.

- Typage: vérification de la vraisemblance d'une action

Un élément supplémentaire de contrôle est introduit par le typage. Selon cette technique, chaque objet possède un type caractérisé par une représentation en mémoire mais aussi par un ensemble d'opérations plausibles sur les valeurs du type. Les types peuvent être fournis par le langage (nous avons vu l'exemple des chaînes de caractères en BASIC). Le programmeur peut également créer ses propres types et les opérations correspondantes. Dans les versions initiales du Pascal, le langage contrôle simplement si une opération est demandée, elle opérera sur des valeurs du type adéquat. Il n'est pas possible de rendre certaines opérations inaccessibles au monde extérieur. Qui plus est, certaines opérations basiques du langage comme l'affectation admettent n'importe quelle variable.

On peut à ce point établir une relation triviale avec quelques systèmes d'exploitation tels l'OS400 d'IBM - qui identifient les objets permanents par des types (fichier texte, fichier exécutable, périphérique etc ...)- et qui possèdent différents jeux de commandes selon les types d'objet (édition de texte, exécution, arrêt). Ces systèmes peuvent donc restreindre les commandes applicables à un objet en interdisant par exemple la modification binaire des exécutables. Ceci accroît d'autant le niveau de sécurité du système.

- Vers la modularité

Les techniques visées ci-dessus ont effectivement permis d'accroître la taille et la complexité des programmes tout en limitant les coûts de développement. Mais elles devinrent vite insuffisantes. On eut alors recours à un style dit programmation défensive dont l'idée essentielle est de considérer les autres programmes-programmeurs comme des intrus plutôt que comme des collaborateurs de bonne volonté. Le prolongement des techniques précédentes dans cette optique conduit à limiter au strict minimum les possibilités d'intervention du monde extérieur à l'intérieur du programme.

La prise en compte de cette idée dans les langages de programmation a donné lieu à la notion de module qui a été complètement intégrée dans les langages récents comme ADA, Modula2 et C++.

2.4.3 Modula2, Ada: Modularité, encapsulation

Les programmes se présentent actuellement sous la forme d'une collection de modules plus ou moins indépendants qui collaborent pour résoudre le problème posé. Selon le niveau d'abstraction choisi, Un module peut implémenter une variable ou une structure de donnée complexe, un type abstrait ou une classe d'objets. Pour cela, le module comprendra des définitions de constantes, de types de fonctions ou procédures, de variables ou même d'opérateurs. L'interaction avec les autres modules est limitée en groupant ces définitions en deux catégories: d'une part les éléments exportés, visibles et manipulables depuis l'extérieur et d'autre part les éléments réservés à un usage interne. Ceci donne lieu au terme d'encapsulation, propriété selon laquelle un élément peut être complètement enfermé dans un module, seuls restant visibles les éléments minimum pour l'utilisation.

Cette encapsulation peut porter soit sur de simples données (Modula2, C, Pascal avec unités) soit sur des types (types abstrait comme en ADA ou en C++). Dans ce dernier cas, il est même possible (selon les langages) d'interdire les opérations de base - type limited private en ADA [Wi89], redéfinition de l'opérateur '=' en C++ de telle sorte que les clients soient obligés de passer par ce module quelque soit l'opération.

2.4.4 Le modèle de sécurité

Le modèle de sécurité que l'on peut extraire de l'ensemble de ces techniques est fortement lié au savoir faire des compilateurs. Il consiste essentiellement un maximum de contrôles statiques sur l'accessibilité des données ou des fonctions. Pour la mise au point, le compilateur peut éventuellement adjoindre au code généré des contrôles dynamiques (valeurs des indices de tableaux et des pointeurs, etc.). La section suivante détaille les outils apportés par le système de développement à cette fin.

En un deuxième temps, nous allons nous préoccuper de la manière dont le programmeur met en oeuvre cette sécurité. Nous verrons alors que le modèle utilisé profite pleinement de la puissance du langage de programmation.

2.4.4.1 Description du modèle sous-jacent

La sécurité est décrite par le programmeur sous forme déclarative. En effet, outre les déclarations nécessaires à la compilation, il établit la liste des éléments accessibles à l'extérieur du module. Réci-

proquement, quand il utilise un autre module, il ne peut le faire qu'au travers des éléments exportés par ce module. Il s'agit donc de manière basique d'une protection en tout ou rien (ou plutôt accessible, non accessible). Notons qu'une fois le programme compilé et assemblé, le programmeur n'a plus aucun pouvoir sur le programme ou le module qu'il y a inclus.

Cette protection sommaire n'est pas suffisante. On peut désirer par exemple publier la valeur d'une variable en interdisant sa modification. Le programmeur doit alors décrire un certain nombre de primitives qu'il peut mettre à disposition des autres modules. De cette manière, il est possible de réaliser une pile d'entiers en n'autorisant que les opérations d'empilage, de dépilage et d'initialisation. Bien que plus ardu à utiliser, ce modèle apporte de nouveaux avantages: on peut par exemple interdire toute opération de dépilage si la pile est vide -ce qui n'est pas facile à réaliser sur les systèmes vus précédemment-. A l'inverse, ce modèle ne propose pas d'identification du client[1]; un programmeur désireux d'affiner cette protection devrait utiliser des techniques cryptographiques.

2.4.5 Conclusion

Nous avons vu dans cette section comment on pouvait mettre en parallèle la façon dont un programmeur pouvait protéger l'implémentation de ses modules -par rapport à un monde extérieur supposé hostile - et la sécurité des systèmes en général. Nous distinguons alors dans des différences significatives dans les modèles utilisés.

- La notion d'utilisateur est quasiment absente
- Il n'existe pas de liste exhaustive des opérations contrôlées par ce système
- La granularité des éléments protégés est extrêmement fine (1 variable ou 1 fonction)
- Le programmeur peut décrire une politique de sécurité complexe par le biais de fonctions propres. Dans ce cadre, cette sécurité tend vers un certain contrôle d'intégrité
- Le contrôle d'accès est complétement défini de manière statique (lors de la compilation). Une fois délivré, le propriétaire n'a plus aucun pouvoir sur les accès

2.5 Conclusion du chapitre

Tout au long de ce chapitre, nous nous sommes attachés à mettre en évidence que pour fiables qu'ils soient, les mécanismes de sécurité répondent en partie à des besoins généraux de sûreté d'utilisation mais sont en même temps très dépendants du système informatique concerné. Ainsi cette hétérogénéité met en relief deux phénomènes contradictoires. Les vocations sont certes différentes, néanmoins nous pouvons retenir de cette leçon que le premier phénomène mis en évidence est la faiblesse ou plus modestement l'inconvénient d'un manque de cohérence d'un système à l'autre. Le

1. Seules les langages objets permettent de faire varier la visibilité selon l'appelant et encore ceci n'est possible qu'en utilisant la hiérarchie de classe (mot clé `private` et `protected` en c++).

second est que par la plus grande diversité de ces mécanismes, nous pouvons extraire de chacun les éléments positifs de manière à les intégrer judicieusement lors de la conception de notre outil qui se veut homogène.

Le S-Shell, qui aura retenu notre attention, aura pour tâche de créer une dynamique de sécurité dans le sens où nous voulons affiner les outils de contrôle d'accès aux données valables pour tout système informatique. Bien souvent, les priorités analysées tout au long de ce chapitre serviront de modèle à la conception et au développement du S-Shell. Des références à ce chapitre aideront le lecteur à mieux situer où et comment nous les avons importées. Le chapitre suivant introduit les fonctionnalités du S-Shell, et après avoir mis en évidence le déroulement sur plusieurs chapitres, la description de l'outil, son utilisation et son intégration, une étude didactique sera proposée. Pour les lecteurs désireux de connaître plus en détail comment le S-Shell a été développé, des références en annexes permettront d'affiner l'étude de la structure de l'outil et son développement lexical et syntaxique.

Chapitre 3

Un nouveau modèle de sécurité : Le S-Shell

3.1 Introduction

Pour faire face aux problèmes évoqués dans le chapitre précédent, nous proposons un outil que nous avons baptisé S-Shell. Nous l'avons appelé ainsi, car pour aboutir à ce résultat, nous avons créé une sur-couche au shell unix. La finalité du produit étant d'assurer un niveau de sécurité de fine granularité, nous avons pensé que 'secure-shell' ou plus simplement S-Shell serait un nom satisfaisant pour cette maquette.

Le but de cet outil est de pouvoir réaliser un système de protection paramétrable qui soit capable de filtrer à tout moment des commandes émises en fonction des utilisateurs et des circonstances. Le S-Shell doit toujours rester indépendant des objets protégés. Le fonctionnement de cet outil se décompose en deux parties.

- La définition du système proprement dit (analyseur, évaluateur)
- Le schéma de sécurité (description, utilisation)

L'outil fourni offre à son utilisateur la possibilité d'utiliser l'évaluateur en ayant préalablement décrit le schéma de sécurité qui doit se présenter sous la forme d'un fichier texte. Le prototype a été réalisé sous Unix. Il a été testé en créant une sur-couche shell. Ainsi à chaque émission d'une commande Unix, le S-Shell vérifiera qu'elle est acceptable et seulement ensuite l'enverra au shell pour la faire exécuter. Il est à noter que dans l'évolution chronologique du concept S-Shell et de sa mise en oeuvre, nous avons suivi une démarche traditionnelle, c'est-à-dire découpé notre travail en plusieurs parties:

- *Prototype créé sur Unix*: c'est l'objet de ce chapitre. Nous présentons les concepts de base ainsi que la façon avec laquelle ils ont été mis en place. Nous insistons sur le fait que le S-Shell n'est pas dédié à Unix et qu'il peut fort bien s'appliquer aux domaines étudiés aux chapitres précédents

- *Etude de l'adéquation des besoins à l'aide d'exemples simples*: outre l'aspect pédagogique de cette étude (voir paragraphe 8 de ce chapitre), nous voulons montrer que ce prototype répond à la fois précisément et simplement aux lacunes évoquées dans les chapitres précédents. Une comparaison effective des résultats sera à cette occasion présentée
- *Implantation dans un environnement expérimental conséquent*: nous avons ensuite intégré notre prototype dans un projet d'actualité concret. Les raisons pour lesquelles ce travail a été réalisé sont largement décrites dans l'introduction du chapitre 2. Il est clair qu'une validation de notre travail ne pouvait se contenter de décrire quelques schémas de sécurité ne prenant effet que sur des situations d'école. Il fallait absolument inclure notre propos dans un projet mettant en évidence de nouveaux problèmes sécuritaires de façon à prouver son adéquation à des problèmes de sécurité modernes (délégations de droit, etc ...)
- *Emulations des concepts dans un système d'exploitation moderne (MPCOS)*: ceci représente en fait l'aboutissement, de ce projet de recherche. Nous avons basé note réalisation sur un système d'exploitation carte quasi-réel. En effet, ce que nous appellerons par la suite et par abus de langage MPCOS (Multiple Payment Card Operating System) sera en définitive plutôt une intersection des caractéristiques de la carte MPCOS de Gemplus et de la norme 7816-4. Nous en rappellerons plus longuement les aspects principaux au chapitre 6. En résumé, nous avons retiré les attributs de sécurité de ce système pour le remplacer par notre outil de description de politique de sécurité S-Shell. Outre l'intérêt concret de la chose, il est important de souligner les variations que va subir la première version prototype du S-Shell afin de pouvoir atteindre de tels objectifs. En effet, notre première version gourmande en place mémoire ne peut définitivement par convenir aux spécificités du monde de la carte à microprocesseur. Il devient alors fondamental de lui faire subir des modifications importantes afin qu'il puisse s'adapter aux requis de la carte sans pour autant qu'il perde de ses fonctionnalités

Pour en revenir plus précisément à notre travail, voici donc la manière dont nous avons envisagé le prototype. En fonction du schéma de sécurité décrit dans un fichier textuel constitué en dehors de l'environnement de travail, le S-Shell validera ou interdira une commande Unix entrée au clavier.

Si la commande est acceptée par l'évaluateur (cf paragraphe 4 de ce chapitre) du S-Shell, elle sera immédiatement exécutée et le S-Shell sera transparent pour l'utilisateur. En revanche, en cas de refus, le S-Shell interdira l'exécution et fournira optionnellement à l'utilisateur un diagnostic d'erreur.

Ainsi donc, à un instant donné, l'acceptation ou le refus d'une commande par le S-Shell ne dépend pas uniquement de l'utilisateur, mais aussi du concepteur du schéma de sécurité. Cela ne veut pas dire que le concepteur doit être présent au moment de l'émission de la commande. Néanmoins c'est lui qui va définir les conditions d'autorisation ou/et de refus et c'est en cela qu'il influe sur le traitement d'une commande émise.

En effet, le schéma est établi une fois pour toutes et n'est en aucun cas modifiable. Ceci permet d'envisager la suppression totale de la notion de super-utilisateur ou d'autorité absolue. Tout doit être pris en compte lors de la phase de description, et le S-Shell doit par la suite être capable de gérer seul les droits de chacun en fonction des événements. Les caractères évolutifs doivent être absolument

envisagés dès le début pour aboutir à un tel perfectionnement. Ceci peut paraître assez restrictif, mais cela renforce l'idée qu'une définition exacte des attributs d'un système reste un aspect fondamental.

Nous précisons à ce niveau que tout ce chapitre se focalisera sur l'étude qui a aboutit à la réalisation de cet outil. C'est pourquoi tout ce qui suivra ici s'appuiera sur l'environnement Unix. L'utilisation effective de ce nouveau système de sécurité dans la carte à microprocesseur sera détaillée dans le chapitre 6.

3.2 Fonctionnement du S-Shell

Une des idées de départ que nous avons eu était de concevoir un outil simple d'utilisation, c'est-à-dire qui ne demande *a priori* pas de connaissances spécifiques pour pouvoir être compris et utilisé (nous ne voulions pas recréer un nouveau langage de programmation!). Le but est de construire un langage simple et bien défini afin que l'utilisateur puisse en respectant un vocabulaire strict, restreint et précis et une grammaire souple et intuitive définir son propre schéma de sécurité.

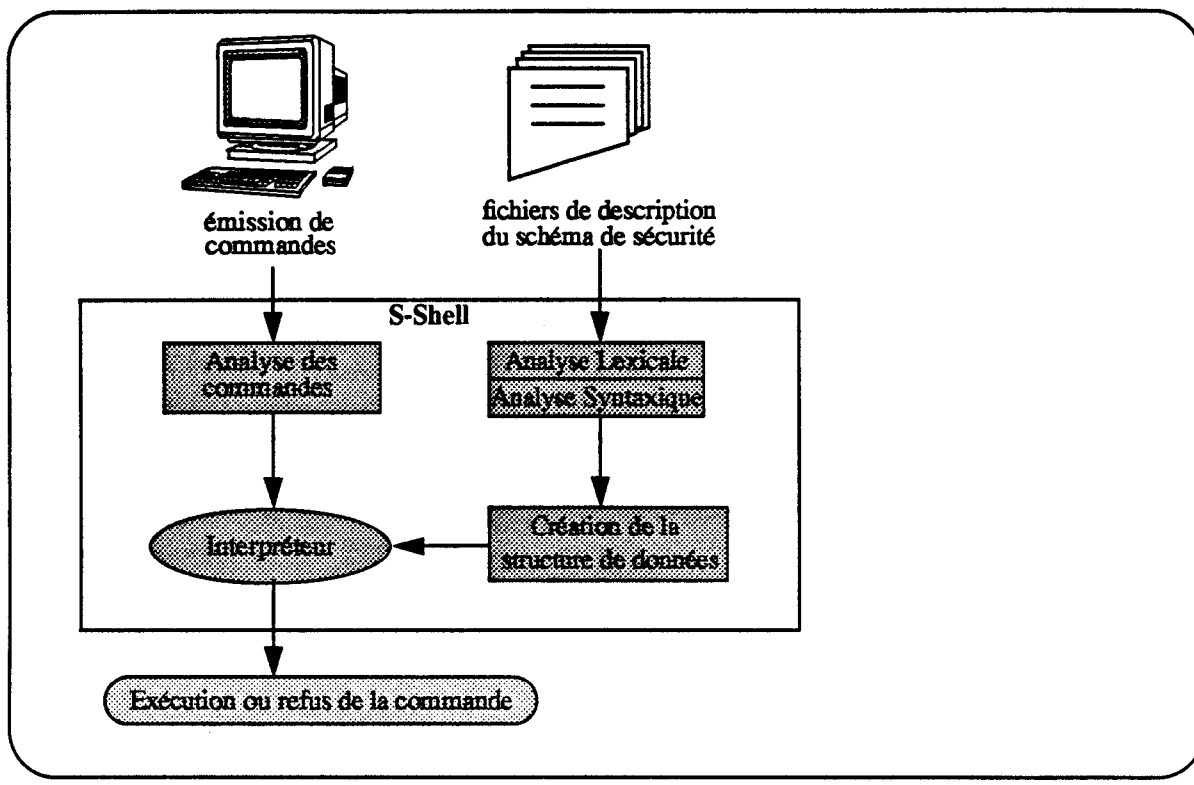
De plus S-Shell dispose d'un ensemble de caractéristiques de langage visant à simplifier au maximum l'expression du schéma. Il est par exemple possible d'avoir accès à l'heure système par la seule écriture d'un mot réservé.

Une condition qui consisterait à interdire les accès à un fichier hors des heures ouvrables doit pourvoir s'exprimer aussi simplement que:

- 'read(fic)' ou 'write(fic)' si heure entre 08h00 et 17h00

3.2.1 Vue d'ensemble du S-Shell

FIGURE 11 Structuration de l'outil S-Shell



Le S-Shell reçoit donc d'une part en entrée une commande *shell* classique venant du terminal et de l'autre un certain nombre de fichiers de description textuels. Les fichiers sont analysés lexicalement par le Lex (a Lexical Analyser Generator) et grammaticalement par le Yacc (Yet Another Compiler-Compiler) [LM90]. Si tous les mots sont reconnus et la syntaxe acceptée alors S-Shell crée la structure de données associée au schéma. L'interpréteur S-Shell analyse alors la commande reçue et en fonction du schéma de protection et du contexte (heure, ...) offert, la valide ou la rejette.

Ceci veut dire qu'il faut bien insister sur le fait que nous avons à distinguer deux phases. La première est la compilation du schéma en un passage, c'est-à-dire à la fois la validation du ou des fichiers textuels entrés par Lex et Yacc et la construction d'une structure de données (voir annexe A) associée. La seconde phase, complètement indépendante, est le contrôle des commandes envoyées qui sera analysé en fonction du ou des fichiers texte correspondant au schéma de sécurité.

Une illustration de la première phase pourrait être la description d'une autorisation d'impression de fichiers **.doc* uniquement aux heures ouvrables en respectant les règles d'écriture lexicales et syntaxiques. La deuxième phase serait alors l'envoi d'une commande d'impression d'un fichier d'exten-

sion *xls* ou d'un fichier d'extension *doc* mais à 20h00. La fin de cette phase se conclurait par le refus d'exécution du S-Shell étant donné que le fichier respectivement n'appartient pas aux fichiers d'extension *doc* ou qu'il est trop tard pour l'imprimer (20h00 n'est pas une heure ouvrable).

Pour en revenir à la commande reçue, l'analyse syntaxique effectuée par le S-Shell est classique. En fait le premier mot lu sera considéré comme étant le verbe, les autres étant les paramètres. Ceci nous a suffi pour effectuer les tests, mais il faut souligner que ce choix volontairement restreint ne remet pas en cause le fait que pour un contexte précis, nous pouvons faire appel à un ordonnancement plus complet avec entre autres options du verbe et paramètres.

3.2.2 Format d'un fichier de description S-Shell

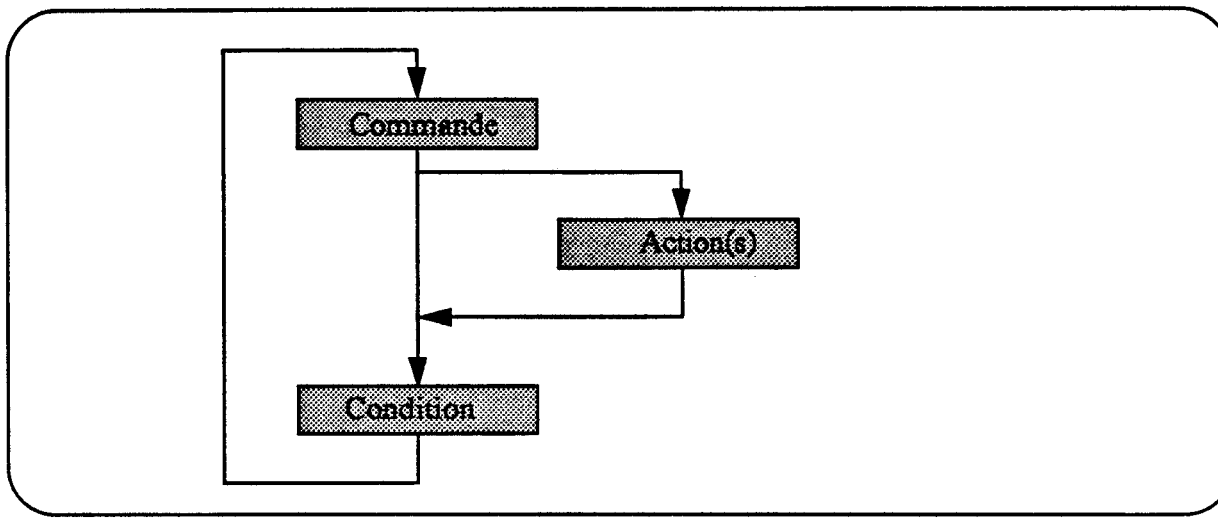
Nous avons mentionné qu'en entrée S-Shell recevait un ou plusieurs fichiers texte dont la présentation devait satisfaire un lexique et une grammaire prédéfinie.

Nous allons dans ce paragraphe décrire comment doivent se présenter ces schémas de description. Un fichier S-Shell se décomposera en un ou plusieurs groupes de trois parties distinctes répétés autant de fois que nécessaire. Ces trois parties doivent être:

1. Le nom de la commande Shell
2. La liste d'actions associées à la commande Shell
3. La liste de conditions associées à la commande Shell. Cette liste pourra se présenter de trois manières différentes: toujours, jamais ou si suivi d'une ou plusieurs condition(s)

Exception: Le nom de la commande Shell peut être le mot-clé 'default', c'est-à-dire l'ensemble des commandes qui n'aura pas été pris en compte lors de la description du schéma. La liste d'actions associée dans ce cas précis s'appellera la liste d'actions initiales.

FIGURE 12 Schéma syntaxique du fichier de description du S-Shell



Pour un fichier texte de 'n' commandes shell, nous pouvons décrire le squelette général de son contenu de la manière suivante, en sachant bien que pour un ensemble "*commande, actions, conditions*", la partie réservée à la liste des actions associées à une commande demeure optionnelle:

FIGURE 13 Description d'un fichier S-Shell

['default'] [{ liste d'actions associées }] :

[toujours, jamais ou si suivi de condition(s)] ;

[commande shell 1] [{ liste d'actions associées }] :

[toujours, jamais ou si suivi de condition(s)] ;

...

[commande shell n] [{ liste d'actions associées }] :

[toujours, jamais ou si suivi de condition(s)] .

Nous avons aussi parlé de multi-fichiers en entrée. Nous avons adopté cette politique de division possible en sous-fichiers afin de rendre la description du schéma à la fois plus simple (et plus ordonnée) mais aussi plus puissante.

Nous obtenons ainsi une structuration ordonnée ou séquence de fichiers facile à écrire.

- *Plus simple*: par analogie avec les procédures en programmation traditionnelle (Pascal, C), nous pouvons isoler un aspect spécifique d'un problème de sécurisation donné sur un fichier bien précis. Ceci aura pour but de dissocier clairement les multiples facettes que peut prendre un schéma de sécurité global en sous-problèmes. Cette clarification d'un schéma offre à la fois à son concepteur mais aussi à celui qui pourra être amené à le relire par la suite une aisance de compréhension non négligeable
- *Plus puissante*: dans certains cas complexes il peut s'avérer très compliqué, voire impossible, de pouvoir décrire son schéma de sécurité d'un seul passage. Le fait de pouvoir subdiviser un schéma par l'intermédiaire de sous-fichiers non obligatoirement indépendants permet d'obtenir une véritable séquence de fichiers dans laquelle un mot-clé spécifique d'un fichier 'i' pourra devenir un nom de commande dans un fichier 'i+1'.
  Cette dernière particularité est envisageable de la manière suivante:

Nous écrivons dans le fichier 'i' -> Ecriture : if (user = 'Write') ;
Nous écrivons dans le fichier 'i+1' ->Write : if (Fic.w =1) ;

'Fic.w' représentant le droit en écriture sur le fichier 'Fic'

On pourra se référer au cas d'étude traité dans [PT94] qui se penche sur ces deux aspects du problème en traitant un exemple d'une bourse aux livres et en proposant une étude comparative de solutions possibles entre la carte MCOS et le S-Shell.

3.2.3 Discussion quant à l'exécution des actions du S-Shell

Nous avons vu au paragraphe précédent que dans le groupe de description S-Shell, nous avons optionnellement à mettre à jour un certain nombre de données. Nous avons à ce propos un certain nombre de remarques à émettre.

La liste d'actions associée à une commande est mise à jour chaque fois que la commande est acceptée puis exécutée. Cela veut dire que, par exemple, si l'on désire incrémenter une variable compteur après chaque opération de lecture sur un fichier, l'ajout d'une unité ne sera fait qu'une fois l'ordre *read* effectué. Ceci est un choix qui peut être remis en cause.

Pour une station fixe, cet ordonnancement ne pose pas grand problème. En revanche si l'on considère une application objets nomades (comme des cartes à microprocesseur), le séquencement des opérations devient fondamental. Une interruption en cours de traitement peut empêcher (volontairement ou non) cette incrémentation d'être effectuée. Ce qui peut être une source d'erreur ou de fraude sur le système.

Etant donné qu'il ne s'agit que d'un choix effectué sur un prototype, cela ne remet pas en cause son fonctionnement. En revanche il sera indispensable de se pencher sur ce problème lors de l'intégration de ce prototype dans un projet comme celui de la Carte Blanche présenté au chapitre 5.

L'idée retenue est qu'il vaut mieux, en cas de panne ou de tentative de fraude en cours de traitement, effectuer en premier lieu l'opération elle-même et mettre les variables à jour ultérieurement. Des procédés informatiques classiques existent à ce sujet, tel le commit à deux phases (cf chapitre 2 au paragraphe 3), et il est recommandé donc de se pencher sur ces détails de sécurisation afin de rendre l'outil encore plus fiable.

3.3 La réalisation effective du S-Shell

Afin de ne pas mélanger l'aspect descriptif et pédagogique du S-Shell et sa réalisation, nous proposons en annexes A et B de trouver toute la construction lexicale, syntaxique ainsi que la structure de données et l'évaluation du schéma. Nous y expliquerons en détail la façon dont ont été abordés les différents problèmes liés à la réalisation tant au niveau des choix qu'au niveau des obligations liées à l'environnement de travail. Ainsi donc, après avoir défini le cahier des charges de notre produit et offert en annexe les plans de réalisation, nous tenons maintenant à aborder une discussion quant aux choix des décisions politiques importantes qui ont été à la base du S-Shell d'aujourd'hui. Il s'agit d'une part de l'introduction de variables simples dans l'outil, et de l'envoi des messages d'erreur.

3.4 La nécessité de variables dans S-Shell

3.4.1 La gestion des variables dans le S-Shell

Bien que rapidement énoncées lors de la présentation du Lex et du Yacc de S-Shell, le rôle des variables a été volontairement mis de côté dans la description de la structure de données et de l'évaluateur de S-Shell. Ce, pour mieux décrire l'esprit avec lequel nous avons élaboré ce prototype.

Néanmoins il convient de réserver une partie de ce 'mode d'emploi' à ces dernières afin de souligner leur rôle fondamental. Supposons, pour reprendre l'exemple donné au paragraphe 2 de ce chapitre, qu'en plus de la condition: "compilation uniquement aux heures ouvrables", nous ajoutions "maximum dix compilations par jour ouvrable". Force est alors de constater qu'il devient impossible de gérer cette simple condition sans introduire la notion de variables. Or si l'on désire réellement construire un schéma de sécurité ayant un niveau d'efficacité acceptable, il faut absolument lui ajouter une gestion de variables pour le doter de cette puissance d'expression suffisante. Dans cette programmation par événements, dans le cas cité ici, le compteur sera, chaque jour, initialisé à zéro.

Comme un des objectifs premiers de S-Shell est que ce schéma ne soit pas un langage de programmation, il faut paradoxalement se restreindre à une utilisation partielle des variables.

L'ajout de variables tel qu'il a été étudié dans S-Shell permet d'étendre la capacité de traitement sans pour autant détériorer la facilité d'utilisation.

En définitive, nous avons envisagé cet ajout pour deux raisons principales découlant finalement de l'expérience que nous avons acquise en décrivant divers schémas de sécurité. Nous nous sommes

aperçus qu'il était nécessaire d'introduire la notion de compteur dans le S-Shell. Ceci permet de décupler la puissance d'expression et d'utilisation du S-Shell sans pour autant s'éloigner de l'idée de départ qui était de ne pas recréer un nouveau langage de programmation. De plus, afin d'aboutir au caractère séquentiel des actions, il a fallu là aussi gérer ce processus par la gestion de variables.

Ce qu'il est important de retenir c'est que nous avons imposé au langage une sorte de paradoxe. Nous voulons le pourvoir des outils nécessaires pour couvrir les cas les plus répétitifs des besoins de description de schéma de sécurité, qui requièrent la gestion de quelques variables, sans permettre pour autant d'utiliser les variables comme on le ferait dans un langage de programmation. Le compromis a été d'envisager le maniement des variables dans deux cas de figure et pas plus.

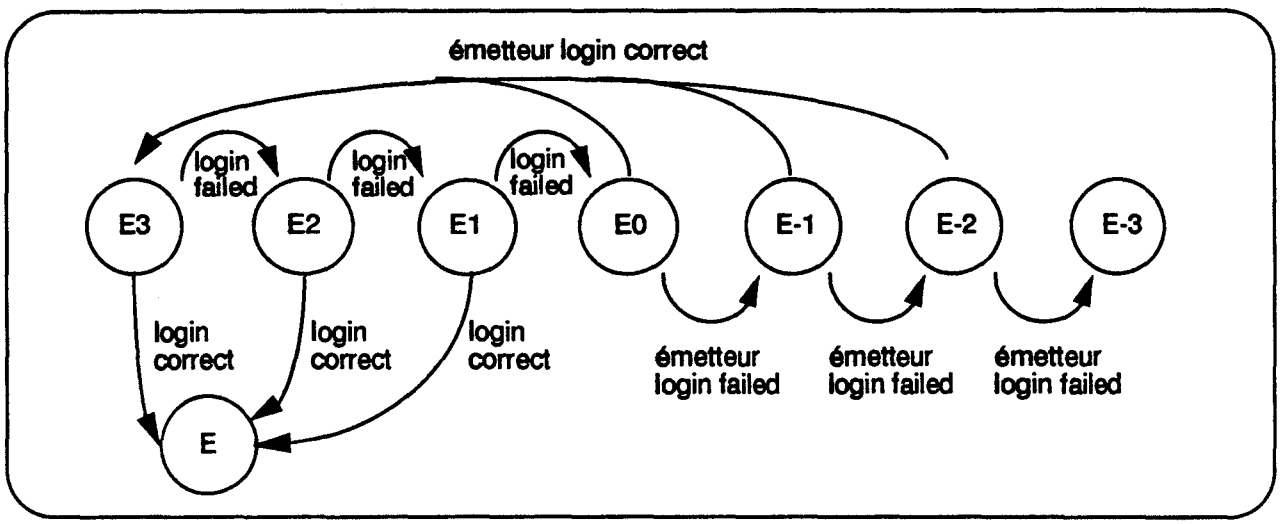
De ce fait, par abus de langage, nous considérons que nous n'avons pas de variable dans notre langage mais que par contre, nous pouvons effectuer des comptages et vérifier la séquentialité des actions, à partir de variables d'exception.

3.4.2 Une alternative coûteuse aux variables

Nous sommes bien conscients que cette utilisation simplifiée de variables dans le S-Shell peut gêner les puristes et ouvre la porte à une gestion des variables plus proche de celle des langages de programmation classique. Rien n'empêche en effet, le concepteur avisé d'un schéma de sécurité d'introduire, de par ses connaissances théoriques, des variables dont l'intérêt lui serait propre et dépasserait les notions de compteurs simples et de séquencement des actions.

Nous avons donc essayé de trouver une alternative à cet emploi de variables et avons donc tenté de simuler ces dernières à l'aide d'automates, en faisant passer ces derniers d'un état à un autre en fonction des commandes envoyées. S'il s'avère que cette approche reste théoriquement réalisable, elle est néanmoins très coûteuse en terme d'utilisation de la mémoire. Or, si l'on se souvient qu'à l'origine ce projet était désigné pour s'adapter à un environnement carte à puce et non à un système informatique classique, nous avons estimé qu'il serait plus adéquat de maintenir une gestion simple de variables minimales, plutôt que de s'orienter vers une possible solution à l'aide d'automates. Cette conclusion n'est pas apparue d'elle-même et nous avons, pour en arriver là, tenter de simuler à l'aide d'automates, le simple mécanisme de ratification de la carte bancaire. Il s'est avéré que l'espace requis pour n'effectuer que cette simple opération représentait 63 octets. En effet, comme le montre la figure 14, nous devons gérer 8 états, 6 envois de messages et 13 transactions (ratification de l'utilisateur et de l'émetteur), ce qui représente un total de (13*3)+(6*4) = 63 octets. Ce coût en espace mémoire n'étant évidemment pas acceptable, nous avons laissé de côté cette solution. Il reste qu'elle peut toujours être reprise pour quiconque désirerait reprendre ce modèle dans un système informatique dans lequel cette économie en mémoire n'est pas prioritaire.

FIGURE 14 Mécanisme de ratification d'une carte bancaire



3.4.3 Vers un S-Shell plus orienté vers l'utilisation de variables?

Nous avons toujours gardé en point de mire la conception d'un outil facile d'emploi ne présentant que par absolue nécessité certaines variables. La justification de leur utilisation était de souligner que l'amélioration dans la puissance d'expression qu'elle offrait en contrepartie de leur existence était avantageuse, qu'il eût été dommageable de vouloir s'en passer.

En définitive, dès lors que l'on a admis la possibilité d'utiliser quelques variables vers des objectifs bien définis, élargir l'utilisation de ces dernières dans le concept du S-Shell reste dans le domaine de l'envisageable. Nous y perdrons en aisance d'utilisation et en convivialité vis-à-vis des non-programmeurs, mais en revanche nous autoriserons des structures de données plus complexes. En guise d'exemple, lors de notre première tentative de simulation de la carte MCOS de Gemplus, nous aurions été bien aise de bénéficier de l'apport de variables de type tableau de façon à pouvoir gérer les bits de chaque format d'instruction.

Notre politique privilégie néanmoins comme il l'a été indiqué dans le cahier des charges la compréhension et l'utilisation par tous du S-Shell. Dans toute la suite, nous nous contenterons de l'utilisation minimale des variables en montrant qu'en nous limitant à elles seules, nous maintenons une grande liberté d'expression, dans des cas représentatifs en taille et complexité.

3.5 La gestion des erreurs dans S-Shell

Nous sommes bien conscients du fait que ce paragraphe aurait dû faire suite à l'annexe A dans son esprit très proche du travail d'implémentation. Le fait de le trouver ici se justifie car les choix que nous allons préciser maintenant sont à l'origine de la création d'un vérificateur d'intégrité qui aura

pour tâche (mais nous le verrons par la suite au chapitre 4) de suppléer en partie le S-Shell de certaines vérifications quant à l'intégrité des données au moment de leur définition et à la suite de chaque traitement.

3.5.1 Le recensement des erreurs de S-Shell

Les types d'erreur que nous pourrons rencontrer lors de l'évaluation du schéma de sécurité sont de plusieurs types bien distincts.

1. Nous pouvons envisager l'erreur de syntaxe pure. Le concepteur ne suit pas la règle proposée. De cette manière, à l'évaluation nous obtiendrons un message d'erreur de syntaxe unique. On pourrait si le besoin s'en faisait sentir implanter un véritable compilateur afin d'affiner l'impression des messages d'erreurs. Ce n'est pour l'instant pas une des priorités que l'on s'est donné pour S-Shell. Comme nous l'avons énoncé précédemment le schéma n'est pas modifiable et donc la nécessité d'un véritable compilateur ne se fait pas vraiment ressentir. On gardera en tête le fait que l'on peut à tout moment ajouter cette éventualité.

2. Comme nous avons envisagé d'introduire des variables d'exception dans le schéma afin de lui donner une puissance d'élaboration plus forte, nous devons alors envisager, de par la réalisation même de S-Shell, la possibilité d'existence d'erreurs, non plus syntaxiques mais de compatibilité. Ceci fait appel à la notion de typage et est à la source d'une idée d'outil vérificateur d'intégrité qui sera décrit par la suite (chapitre 4) et utilisée conjointement avec le S-Shell.

3.5.2 Politique de résolution

Pour résoudre ce type d'erreurs, nous avons envisagé deux types de solutions à considérer en fonction du type d'erreur.

Considérons les trois exemples suivants:

- (today + time)

- (durée + variable)

- (today + variable entière)

Dans les trois cas de figure, nous nous trouvons en présence de formulations syntaxiquement acceptables, mais qui posent le problème de compatibilité. Néanmoins le traitement des erreurs se fera de deux manières bien différentes.

En effet si l'on se penche d'un peu plus près sur ces deux écritures on peut se rendre compte qu'elles diffèrent de par leur substance. Dans le premier cas à la lecture de '(today +', on peut rien dire quant à l'apparition d'une erreur possible. En revanche la présence de '(durée +' offre à elle seule une information capitale. Nous ne pouvons jamais avoir ce type d'addition, car le sous-format *durée* de

date, s'il est utilisé le sera uniquement au niveau de l'opérande de droite. Une première lecture pourra donc amener directement un premier type de message d'erreurs. Celui-ci sera détecté et envoyé à la construction de la structure de données (dans le programme fonc.c, cf annexe A). Nous aurons ici une erreur de syntaxe.

De la même manière pour (today + time), nous aurons affaire à une erreur statique de syntaxe immédiatement détectable.

En revanche, l'addition de deux opérandes acceptables séparément mais typiquement incompatibles sera rejetée à l'évaluation (dans le programme eval.c dont on a parlé précédemment). C'est bien le cas du troisième exemple proposé, car today et variable entière sont tout deux acceptables syntaxiquement parlant, mais à l'évaluation, nous détecterons une erreur de compatibilité de type. C'est une erreur de typage dont il s'agira ici.

Nous avons donc tout intérêt afin d'éviter ce genre d'erreurs à effectuer une déclaration des variables utilisées par typage fort, qui pourrait être un premier pas vers une vérification de la cohérence du schéma. L'idée a été développée par la suite, nous en reparlerons au chapitre 4.

3.6 Messages d'erreur du S-Shell

Parallèlement aux erreurs de syntaxe décrites au paragraphe 5 de ce chapitre, nous devons faire face à un autre problème plus délicat à traiter quant à sa politique de résolution. En effet, imaginons que les accès en lecture ne soient autorisés que pendant les heures ouvrables. Un utilisateur se présente alors et après s'être identifié tente de lire un fichier en dehors de ces heures permises. Comment doit se comporter le S-Shell? Il est clair que dans tous les cas de figure la commande sera refusée, mais cela suffit-il? Doit-on fournir à l'utilisateur un message d'erreur, et si oui, de quel type?

Ce problème reste grandement ouvert, mais il faut prendre garde à l'intrus qui tentera par essais successifs et par analyse des messages d'erreur fournis de reconstituer le schéma de sécurité de l'utilisateur auquel il aura préalablement dérobé le mot de passe. Si le S-Shell fournit comme message: "Accès en lecture à ce fichier non autorisé pendant les heures ouvrables", il est certain qu'un individu mal intentionné réussira sans trop d'efforts à mettre à jour le schéma associé à l'utilisateur. Pourtant, l'utilisateur légal qui aurait devant les yeux un tel message bénéficierait d'une explication claire et explicite du pourquoi du refus d'exécution de son ordre. Ainsi donc, il devient fondamental de savoir gérer quels messages le S-Shell devra envoyer en réponse à un refus. Le compromis entre le fait de donner une explication compréhensible par le réel utilisateur et ne pas donner trop d'indications au fraudeur est fondamental dans la mise en oeuvre de ce travail.

Cet aspect du problème provient surtout d'une politique spécifique à adopter par chaque concepteur, mais il convient néanmoins de le signaler afin de prévenir ce type d'attaques détournées, qui fait immédiatement référence aux données sensibles et aux problèmes d'inférence relatifs aux bases de données (cf chapitre 2 au paragraphe 3).

3.7 Etudes d'exemples à l'aide du S-Shell

Nous voulons désormais, à l'aide d'exemples concrets, montrer l'étendue des possibilités qu'offre le S-Shell. Nous allons préalablement comparer différentes façons d'envisager la résolution d'un problème simple de sécurité et mettre en évidence la maniabilité du S-Shell. Puis, en se basant sur quelques exemples significatifs nous allons essayer de mettre en valeur un aspect nouveau ou simplement plus facile à mettre en oeuvre qu'avec un système "classique" d'un problème de sécurisation. Les deux derniers exemples de ce chapitre sont des cas concrets dont l'intérêt est de montrer que dans une situation réelle le S-Shell peut apporter une certaine amélioration en un minimum de lignes de description et toujours simplement.

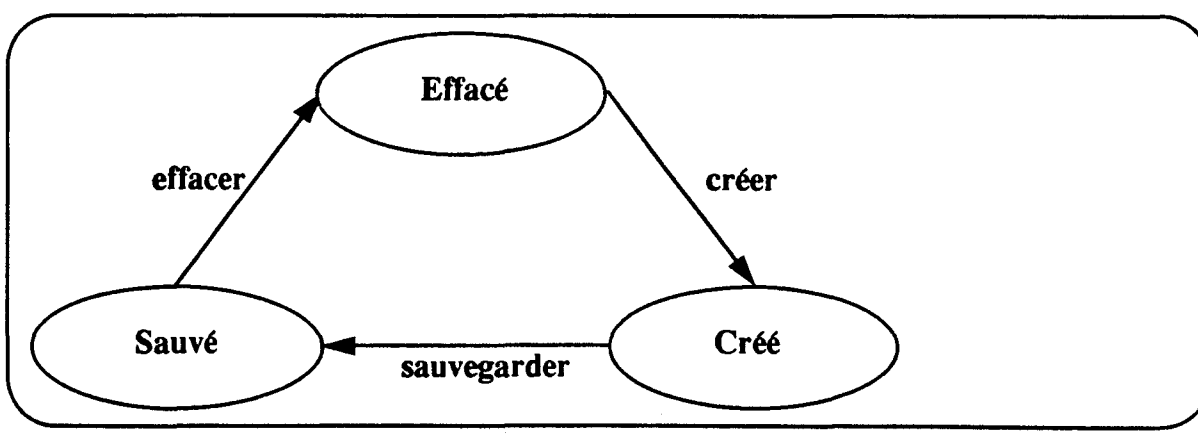
3.8 Exemple 1: Etude comparative

3.8.1 Le cas traité

Nous voulons dans ce chapitre et à l'aide d'un exemple trivial montrer comment l'on pourrait résoudre un problème de sécurité dans les domaines des bases de données et des systèmes d'exploitation. Parallèlement à cela une solution S-Shell sera aussi envisagée ainsi qu'une étude des avantages éventuels qu'apporterait le langage de sécurité.

Le cas traité est le suivant. Nous n'autorisons les opérations d'effacement d'une information que si une sauvegarde de cette information existe. Le terme 'information' est utilisé dans son sens le plus général, à savoir un fichier pour le système d'exploitation et une donnée pour le S.G.B.D. C'est-à-dire que le 'delete' des systèmes d'exploitation ou le 'clear' des bases de données suit le graphe suivant:

FIGURE 15 Opération d'effacement



- Par la voie des systèmes d'exploitation, afin d'assurer cette protection minimale, il faut successivement effectuer une opération visant à protéger le fichier contre les opérations d'écriture, et créer un programme qui vérifiera l'existence du fichier de sauvegarde du fichier que l'on veut effacer, détruira les données et enfin exécute la commande. Pour cela le programme

devra posséder les droits appropriés. On insiste donc sur le fait que cette si succincte opération de protection requiert une certaine connaissance de la programmation procédurale.

- Par la voie des bases de données, le processus est de ne pouvoir détruire une donnée qu'une fois archivée. Le concept même de bases de données impose une description exhaustive de toutes les informations que le système d'information comporte. Cette remarque vaut également pour les informations anciennes mais qui doivent être archivées car elles peuvent être encore utiles. Nous excluons en effet de ce paragraphe les dispositifs de sauvegarde de la base de données car les informations sauvegardées de cette manière ne sont pas accessibles au moyen des requêtes habituelles de la base de données (les mécanismes de restauration sont hors du langage de requêtes).
  Les remarques précédentes imposent donc que les informations sauvegardées soient décrites dans le schéma de la base. Pour résoudre ce problème, on inclut habituellement dans le schéma non seulement la donnée actuelle mais également les anciennes valeurs de cette donnée. Il est alors nécessaire d'adjoindre à cette donnée une information qui précise la durée de validité de cette information.
  Prenons comme exemple un taux d'imposition qui peut varier de jour en jour. Une mémorisation de ce taux sans historique demande simplement la description d'une table unicolonne et uniligne. Si l'on veut pouvoir retrouver la valeur du taux à n'importe quelle date, il faut à la fois autoriser plusieurs lignes (ce qui est le comportement normal d'une table dans le modèle relationnel) et ajouter une (ou deux) colonnes pour préciser la période de validité de chaque ligne. On obtient alors le schéma suivant:

TABLE 3 Schéma associé

De	A	Valeur
null	null	13.7
10/12/94	07/01/95	13.5
01/11/94	09/12/94	13.4

Dans la table, on trouve à la fois les anciennes valeurs - associées à leur période de validité - et la valeur actuelle avec des dates nulles (valeur conventionnelle).
Dans ce contexte, les opérations d'archivage et de destruction se transposent comme suit:
Utilisation de la valeur actuelle:
- Sélection de Valeur ayant date nulle.

Changement de valeur:
- Inscription de la période pour l'ancienne valeur.
- Création d'une nouvelle ligne avec pour valeur initiale (*null*, *null*, nouvelle_valeur)

La description des droits sous SQL permet de réaliser la sécurisation de ces opérations:
Pour cela, il faut créer une vue V1 qui définit le sous-ensemble réduit à la ligne active.
On peut ensuite offrir le droit de modification des colonnes `De` et `A` pour cette vue. Ceci auto-

rise l'archivage de la valeur actuelle.
On peut également définir le droit d'ajouter une ligne à la table ce qui permet de créer une nouvelle valeur actuelle (on suppose que les valeurs à la création d'une ligne sont *null* pour les colonnes De et A).

Critiques:
Ces opérations sont relativement artificielles
Il faut prendre des précautions pour qu'il ne puisse pas exister deux valeurs actuelles à un moment donné.
La sélection systématique nécessaire à l'utilisation de la valeur actuelle est assez fastidieuse.

- En utilisant le S-Shell, nous pouvons envisager deux solutions:

FIGURE 16 Description des deux étapes

1. On teste l'existence du fichier de sauvegarde par la ligne:

```
rm : if (exist("backup")) ;
```

2. On teste le processus complet par le schéma:

```
default { Status = effacé } ;
creation { Status = créé } :
if ( Status = effacé ) ;
backup { Status = sauvé } :
if ( Status = créé ) ;
delete { Status = effacé } :
if ( Status = sauvé ) ;
```

Voici donc une présentation de la manière dont on procède pour arriver au même résultat sous trois approches différentes. Les conclusions que l'on peut déduire de ce test symbolique sont de plusieurs ordres. Dans un premier temps on peut constater que l'approche base de données est certainement la moins intuitive et requiert une certaine connaissance de fonctionnement afin d'être menée à bien. Ce qui est contraire au prérequis de notre langage. En ce qui concerne l'approche système d'exploitation type Unix, nous bénéficions d'un déroulement plus logique avec néanmoins la charge d'un programme procédural à écrire, ce qui peut s'avérer être long à mettre au point et nécessite un investissement en temps chaque fois que l'on désire créer un nouveau schéma de sécurité, Par ailleurs le programme, sur un schéma plus complexe peut devenir fastidieux à élaborer. Le point de vue S-Shell décompose aussi le travail en sous-étapes mais est très systématique. La position claire du problème suffit presque à écrire les quelques lignes déclaratives uniquement de S-Shell. Il suffit de reprendre pas à pas le graphe de la figure 15.

3.9 Exemple 2 : Bourse aux livres dans une école

3.9.1 Le cas d'étude

Nous allons décrire le cas d'une bourse aux livres qui de par sa consistance et son intérêt pédagogique est devenu au fil des ans le T.P. d'application classique d'utilisation de la carte MCOS [Ge]. L'association des parents d'élèves du lycée et collège Notre-Dame propose annuellement une bourse aux livres à ses adhérents. Le principe de cette bourse est simple. Les familles peuvent acheter les livres nécessaires aux études des enfants. Ces livres pourront être repris l'année suivante (ou ultérieurement en cas de redoublement). Une décôte est appliquée selon l'état du livre (il y a trois états possibles: neuf, usagé, inutilisable). Une liste des livres demandés pour chaque enfant a préalablement été fournie aux parents en fonction des classes suivies. De toute manière une adhésion est demandée par famille et par an de 200 FF. Les livres qui n'auront pu être fournis peuvent être commandés par l'intermédiaire de l'association.

En pratique l'organisation de la bourse est la suivante: la famille va être amenée à suivre un trajet qui l'amènera à un certain nombre de postes dans la chaîne en fonction des opérations qu'elle désire y effectuer. Elle véhicule avec elle une carte à microprocesseur dans laquelle sont reportés un certain nombre d'éléments. La liste des postes est:

- Une reprise des livres des années précédentes. L'opérateur y réceptionne les livres rendus, en évalue l'état et les remet en stock. Il note les livres repris et leur état
- Un poste d'achat où l'opérateur, au vu de la liste des livres demandés, constitue la livraison à la famille. Les livres sont bien-sûr également facturés en fonction de leur état. Il est possible que les livres demandés ne soient plus disponibles, ils doivent alors être commandés. En fin de compte, la famille repart de ce poste avec les livres à emporter et, sur sa carte, la liste des livres à facturer et/ou à commander
- Un dernier poste, dit de paiement, où l'on effectue le solde des opérations d'après les renseignements contenus dans la carte à puce, sans oublier d'y inclure l'adhésion annuelle. Un ticket de caisse est alors émis et on encaisse la somme correspondante. Les livres sont payés d'avance. En outre ce poste recense les commandes à effectuer
- Un poste de retrait sera ultérieurement disponible pour que les parents puissent obtenir les livres qu'ils auront commandés et payés

3.9.2 Objectifs

De cette description du problème nous pouvons identifier deux cibles de sécurité.

1. La première est d'empêcher un intrus de lire ou de modifier des données dans la carte. Ceci est fait grâce aux droits inclus dans la carte elle-même. L'authentification utilise des méthodes traditionnelles de PIN code ou de mots de passe. Ces mécanismes sont par contre hors de nos inquiétudes car ils font partie de nos hypothèses de départ

2. Le second but est de superviser les actions des vrais utilisateurs afin de les empêcher de changer les données dans la carte pour gagner des bénéfices illégaux. Comme nous l'avons déjà noté les mécanismes bas-niveau qui agissent dans ce sens sont pour un fichier d'une carte les protections *read* et *write*. Par exemple, ceci est un moyen de prévenir le poste de reprise de faire croire que les livres sont déjà payés (qui est le privilège du poste de paiement)

Nous rappelons que ce que nous voulons faire est de permettre une description plus fine mais aussi plus naturelle de ce qui peut être fait et de ce qui doit être fait. Dans cet exemple, une parfaite description doit clairement mettre en évidence que quand un livre est marqué 'récupéré', l'état de ce livre doit aussi être rempli. Ceci introduit deux remarques:

1. La première remarque est de souligner que plus on décrit précisément le schéma de sécurité, plus proche on sera de la description des contraintes d'intégrité. Finalement nous devrions décrire toutes les règles de notre système d'informations
2. La seconde remarque est de dire que les mécanismes actuels autorisent seulement d'interdire une commande en fonction des circonstances (présentation de code, etc ...). Le responsable de la description du système d'informations ne peut pas facilement donner des obligations

Prenons pour illustrer ceci l'exemple d'un système d'informations devant se charger de deux comptes. Le système est tel que la somme des deux comptes doit rester constante (c'est le cas des porte-monnaies électroniques). Le but du système de sécurité est d'empêcher une personne de créditer un compte sans débiter l'autre du même montant. Les mécanismes traditionnels autorisent de déclarer que ce caissier seul est autorisé à changer les données dans les fichiers. Néanmoins ceci est amplement insuffisant par rapport à ce que l'on veut faire.

Une solution classique à ce problème est d'écrire un morceau de code qui se chargera à la fois de l'incrémentation d'un compte et de son opération inverse sur l'autre. Cela impose un travail lié à une cible d'implémentation (la machine, le système ou la carte à puce), ce qui peut s'avérer être très laborieux dans des cas plus complexes et être une importante surtout source d'erreurs.

Le langage que nous utilisons permet ce genre d'opérations. La façon de raisonner est alors de dire qu'un compte ne peut être crédité d'une somme donnée que si l'autre compte est diminué de la somme identique. Même si nous pensons qu'il s'agit ici d'un réel progrès, il faut constater que nous manquons d'outils adéquats pour rendre cela plus convivial. Ce point sera à la source d'une autre réflexion complémentaire que nous décrivons dans le chapitre 4. Cette réflexion porte sur les considérations d'intégrité et nous avons besoin du concept de transaction atomique étudié dans le contexte des bases de données. Nous y reviendrons.

3.9.3 Présentation

Nous voulons d'abord présenter ce que nous pouvons faire avec un système d'exploitation comme le MCOS (Multiple Card Operating System). Ensuite nous effectuerons le même travail avec notre langage à titre comparatif. Nous prouverons que:

1. Dans un exemple concret, nous montrerons qu'avec le S-Shell nous pouvons faire au moins autant que MCOS. Ceci implique la gestion de la sécurité (en termes de droits d'accès) de la bourse. Nous ne nous préoccupons cependant pas de la phase de personnalisation.
2. Si MCOS garantit l'intégrité de l'accès, il ne garantie pas l'organisation des différents postes. Nous allons montrer comment nous pouvons le faire avec ce langage en peu de lignes de description.

3.9.4 Description de la protection par le MCOS

Pour des raisons de sécurité, chaque champ sera manipulé par des postes différents avec des droits différents. Ces droits sont accordés uniquement au niveau des fichiers. Ensuite nous éclatons notre structure de données en groupes de champs en fonction des droits.

Méthodologie:

1. Nous établissons en premier la liste des utilisateurs. Dans notre exemple, nous avons cinq postes différents qui sont "Reprise", "Achat", "Paiement", "Retrait" et "Remboursement". Nous considérons que l'étape de personnalisation est une étape spéciale car elle prend effet au tout début du cycle de vie de la carte [CP93 , MP91] avant même d'avoir été donnée à la famille
2. Par la suite, nous établissons les droits associés à chaque utilisateur sur chaque champ. Ceci se résume à la figure suivante:

FIGURE 17 Droits associés à chaque station de travail

STATIONS	DROITS					
1. Reprise	R		R	R	R	W
2. Achat	R		W	W		
3. Paiement	R	U	R	R	W	R
4. Retrait	R		R	W	R	
5. Rembour.	R	U	U	U	U	U
FICHIERS	1	2	3	4	5	6
CHAMPS	- Famille - Numéro	- Année	- Livre - Condition	- Fourni	- Payé	- Retour - Condition

Remarque: Le but de cette partie n'est pas de fournir une précise description de l'implémentation MCOS. Mais pour être précis nous soulignons que normalement nous devrions transposer ce tableau en termes de valeurs de droits pour chaque fichier.

3.9.5 Description de la protection par le S-Shell

Avant de présenter la description du fichier S-Shell, nous mettons une fois encore l'accent sur le fait que nous ne cherchons nullement à optimiser le schéma ci-dessus. Un fichier texte écrit pour notre langage fournirait dans ce cas:

FIGURE 18 Représentation du fichier S-Shell

Read :	if (file = 1) or ((file = 2) and (user = 'Paiement') or (user = 'Remboursement')) or (file = 3) or (file = 4) or ((file = 5) and (user <> 'Achat')) or ((file = 6) and (user = 'Reprise') or (user = 'Paiement') or (user = 'Remboursement')) ;
Write :	if (((file =2) or (file = 5) and (user = 'Paiement') or (user = 'Remboursement')) or ((file = 3) and (user = 'Achat') or (user = 'Rembousement')) or ((file = 4) and (user = 'Achat') or (user = 'Collection') or (user = 'Remboursement')) or ((file = 6) and (user = 'Reprise') or (user = 'Remboursement'))) ;
Update :	if (((file<> 1) and (user = 'Remboursement')) or ((file = 2) and (user = 'Paiement'))) ;

Comme on peut le constater le schéma proposé est vraiment très simple dans sa conception et permet à un utilisateur d'avoir au moins les mêmes possibilités que MCOS.

3.9.6 Bénéfices supplémentaires

Maintenant, si nous nous penchons sur la manière dont la bourse aux livres fonctionne, nous remarquons qu'il existe une certaine séquentialité des actions. Par exemple il est parfaitement impossible de récupérer un livre qui n'a pas été encore acheté. Ceci veut dire que si nous voulons réellement contrôler à la fois les droits associés aux fichiers et les séquences d'actions, il faut écrire un programme en dehors des facilités offertes par le MCOS.

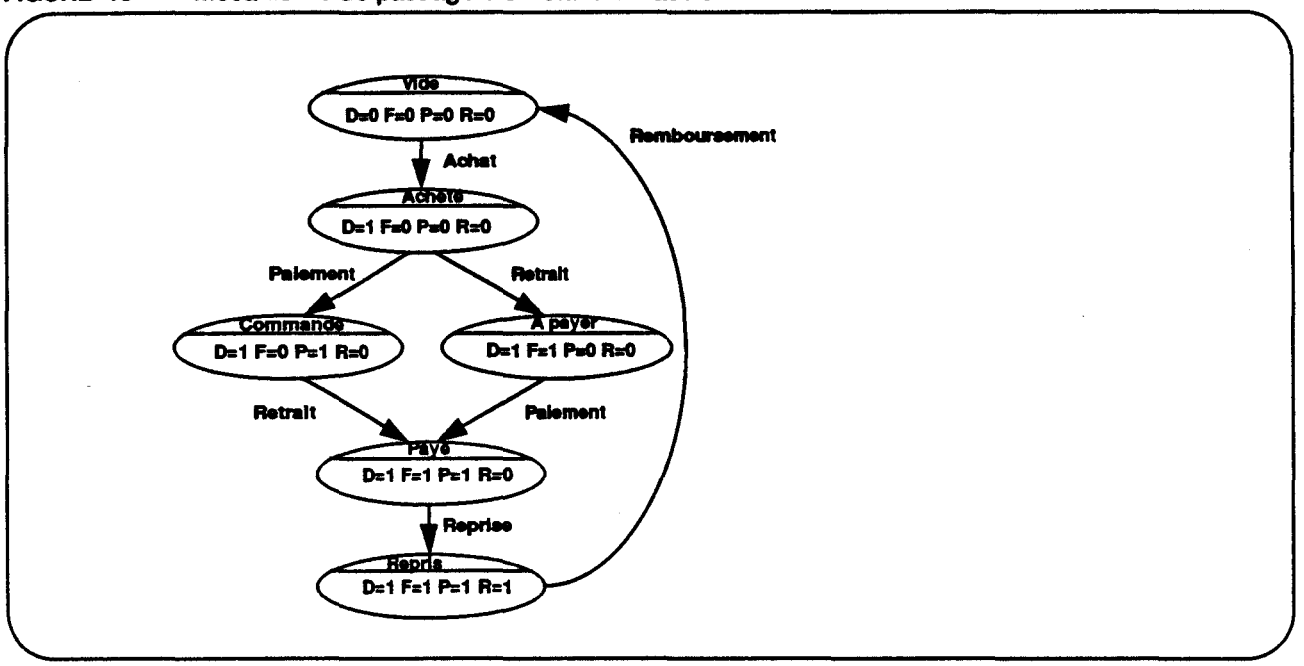
Avec notre description de la sécurité et grâce aussi à l'introduction de variables nous pouvons y arriver.

L'automate (cf figure 19) montre que l'on peut décrire l'état dans lequel nous sommes par un ensemble de quatre variables D (défini), F (fourni), P (payé) et R (repris) associé à chaque livre. Les transitions d'un état à un autre sont Paiement, Reprise, Achat, Retrait. Le but de ce que l'on voudrait faire est de valider ou d'interdire une transition en fonction de la valeur du groupe de variables (D, F, P, R).

Nous pouvons prendre l'exemple (D, F, P, R) = (1, 1, 1, 0). L'utilisateur veut alors payer. Notre vérificateur regarde la commande "Paiement" et note que cette action n'est réalisable que si nous nous trouvons dans les cas de figure (1, 0, 0, 0) ou (1, 1, 0, 0). Par conséquent la commande sera refusée et éventuellement un diagnostic d'erreur pourra être fourni. Cette fonctionnalité ne peut pas être fait avec MCOS.

Pour rendre cela plus clair voici un schéma des actions réalisables pour un livre seul:

FIGURE 19 Mécanisme de passage d'un état à un autre



Au niveau de la description avec le S-Shell, nous obtenons le second fichier texte suivant:

FIGURE 20 Second fichier S-Shell

Reprise :	if ((Book.D = 1) and (Book.F = 1) and (Book.P = 1) and (Book.R = 0)) { Book.R = 1 ; } ;
Paiement :	if ((Book.D = 1) and (Book.P = 0) and (Book.R = 0)) { Book.P = 1 ; } ;
Remboursement:	if ((Book.D = 1) and (Book.P = 1) and (Book.F = 1) and (Book.R = 1)) { Book.D = 0 ; Book.P = 0 ; Book.F = 0 ; Book.R = 0 ; } ;
Collection :	if ((Book.D = 1) and (Book.F = 0) and (Book.R = 0)) { Book.F = 1 ; } ;
Achat :	if ((Book.F = 0) and (Book.P = 0) and (Book.R = 0) and (Book.D = 0)) { Book.D = 1 ; } ;

Ainsi avec ce groupe de valeurs, en demandant une reprise, le S-Shell vérifiera que les valeurs des variables sont correctes et fera passer Book.R à 1. Ceci aboutira à l'état (D, F, P, R) = (1, 1, 1, 1).

En conclusion nous pouvons affirmer qu'en divisant bien les actions, les objets et les commandes, nous pouvons facilement écrire le ou les fichier(s) correspondant à une situation définie. Grâce au premier fichier, nous avons produit une simulation de ce que MCOS peut faire. Le second fichier offrira à l'utilisateur une séquentialité qui manque dans le système d'exploitation de la carte. Nous n'avons pas pris en considération le fait qu'une famille pouvait prendre plus qu'un livre et tout payer d'un coup. Ce problème est assez différent mais n'affecte pas notre façon de voir les choses.

Pour faire les choses correctement, nous pensons qu'il est vital de bien isoler les commandes, objets et actions. Par exemple, dans notre premier fichier les commandes étaient *Read*, *Write* et *Update*. C'est-à-dire qu'il n'y avait pas d'actions spécifiques. Ce qu'il est intéressant de noter est que les utilisateurs que nous avons considérés dans notre premier fichier (Payment, ...) deviennent des noms de commandes dans le second. Ceci renforce l'idée que nous pouvons diviser un problème de sécurité général en plusieurs sous-problèmes plus simples à résoudre.

3.10 Exemple 3 : Utilisation dans un contexte de détection d'intrusion

3.10.1 L'idée de départ

L'étude que nous voulons maintenant présenter se situe dans le cadre d'une étude qui a été menée ces dernières années à RD2P, concernant un système de détection d'intrusions dans la carte. Notre ambition s'est limitée au strict domaine du dossier portable, tels que la carte de crédit, la carte base de données ou encore le porte-monnaie électronique.

Les études axées sur la sécurité de la carte à microprocesseur ont dans la majorité des cas été basés sur les contrôles d'accès aux données. En revanche, il a rarement été pris en considération le fait que pour une raison ou une autre (fraude, négligence du détenteur, ...) un individu puisse avoir libre accès aux informations contenues dans la carte. Par exemple, une personne qui volerait une carte de crédit sur laquelle serait indiqué le code secret pourrait effectuer librement ses achats sans qu'il n'y ait aucun moyen de l'empêcher.

Par analogie, la domotique propose une ébauche de solution à ce problème. Un voleur qui pénètre par effraction dans une villa peut encore être stoppé à temps, si les habitants ont pensé à installer un radar ou une alarme. C'est sur ce constat que nous avons pensé à installer un détecteur d'intrusions dans la carte. Nous n'avons bien sûr pas l'intention de faire "sonner" la carte, mais plutôt de la bloquer (*ie* de rendre les données inaccessibles) dès lors qu'un intrus est détecté. Ainsi toutes les transactions anormales voire suspectes ne seront pas traitées. A l'instar du contrôle d'accès, cette détection est extrêmement liée à l'application. Ce détecteur ne vise donc pas à remplacer le contrôle d'accès mais bien à renforcer le schéma de sécurité déjà installé.

Le domaine d'application que nous avons choisi, afin d'illustrer notre propos, est celui de la carte bancaire. En effet, la simplicité de compréhension de l'application alliée à sa vulgarisation rendent cette application conviviale. L'idée de départ est basée sur une simple constatation: chaque détenteur d'une carte de crédit l'utilisera d'une manière qui lui est propre. En effet, on peut légitimement estimer que sur des périodes de temps fixes (par exemple mensuellement), les montants de retrait aux distributeurs automatiques ainsi que les dépenses courantes effectuées dans des domaines classiques et précis (nourriture, voiture, ...) ne varient pas de beaucoup, ou du moins restent dans les ordres de grandeur équivalents. Ces caractéristiques permettent ainsi de 'personnaliser' les types de dépense de chaque individu. Des remarques équivalentes pourraient être effectuées pour des applications porte-monnaie électronique ou pour des applications de carte CQL (Card Query Language) [P94]. Dans ce dernier cas de figure, on suppose que pour aboutir à un certain résultat, un utilisateur enverra une séquence de commandes qu'il connaît bien, plutôt que de changer fréquemment de façon de procéder. Le détecteur fonctionnera donc par séquences d'actions. C'est sur ces séquences d'actions que l'on basera la reconnaissance de l'identité de l'utilisateur. Nous parlerons à ce propos de reconnaissance comportementale.

3.10.2 Caractéristiques comportementales dans le cadre d'une application bancaire

Pour caractériser une transaction effectuée par carte de crédit, nous nous sommes donnés quatre critères de sélection: la nature de la transaction, le montant de la transaction (nombre réel positif), la date de la transaction (Jour/Mois/Année) et le lieu de la transaction (code postal par exemple). Ces quatre critères forment les bases de l'identification comportementale de chaque utilisateur. Nous avons classé les domaines de dépense les plus courants en huit sous-groupes couvrant à peu près tous les cas de figure: retrait en liquide, transport public, véhicule personnel, nourriture, logement, ameublement, loisir et divers. Les relations possibles entre les quatre entités présentées peuvent intervenir en considérant les critères individuellement, par groupe de deux, trois voire les quatre ensemble.

A partir de ces données, on peut construire un ensemble de règles pour décrire les habitudes d'un individu utilisant sa carte bancaire comme moyen de paiement. Ces règles peuvent être décrites de la manière suivante:

1. M. Van De Greef ne retire jamais plus de 500 frs à la fois et jamais plus de 2000 frs par semaine.
2. M. Van De Greef ne dépense jamais plus de 1500 frs de frais de déplacement par semaine
3. Ces dépenses s'effectue la plupart du temps dans sa ville natale (Paris) et à Lille
4. Son loyer mensuel est de 4000 frs
5. Etc...

Cet ensemble de règles permet de connaître à peu près précisément son comportement traditionnel. Il ne couvre évidemment pas des opérations extraordinaires comme des dépenses dues à un anniversaire ou à des circonstances particulières (vacances, ...). Néanmoins, notre but dans ce paragraphe n'étant pas de redéfinir les concepts de ce détecteur d'intrusion mais bien de voir comment nous pou-

vons utiliser le S-Shell dans ce contexte, nous prions le lecteur désireux de se perfectionner dans le fonctionnement interne de cet outil à la référence [AC94].

3.10.3 Utilisation du produit S-Shell

Au vu de ce qui a été dit précédemment, les habitudes de M. Van De Greef peuvent être résumées sous la forme du tableau suivant:

TABLE 4 Règles associées au profil de M. Van De Greef

Nature	Montant/Achat	Montant/Semaine	Lieu
Liquidité	500	2000	cf Table de Location
Transport Public		1500	
Véhicule Personnel	250	700	
Nourriture	800	1500	
Logement		1000	cf Table de Location
Ameublement		2000	
Loisir	1000		cf Table de Location
Divers	5000		

La table de location représente les endroits les plus fréquents d'utilisation de la carte du détenteur. Nous pouvons avoir par exemple:

TABLE 5 Table de location associée au schéma de M. Van De Greef

Lieu	Fréquence
Paris	10
Lille	3
Lens	1
Le Havre	1

Ceci veut dire que sur une moyenne de 15 opérations, 10 seront effectuées à Paris, 3 à Lille et 1 au Havre et à Lens respectivement. Possédant ces valeurs, une autorité de confiance (la banque dans ce cas) peut alors s'aider du S-Shell pour créer le schéma de sécurité de chacun de ses clients. Connaissant les règles d'écriture en S-Shell, nous pouvons proposer le fichier suivant:

FIGURE 21 Schéma d'un profil client d'une application bancaire

```
default :
if (montant/jour <= 5000 ) ;
liquidité :
if ((montant/jour <= 500) and (montant/semaine <= 2000) and Loc_Table) ;
transport_public :
if (montant/semaine <= 1500) ;
véhicule_personnel :
if ((montant/jour <= 250) and (montant/semaine <= 700)) ;
nourriture :
if ((montant/jour <= 800) and (montant/semaine <= 1500)) ;
logement :
if ((montant/semaine <= 1000) and Loc_Table);
ameublement :
if (montant/jour <= 2000) ;
loisir :
if ((montant/jour <= 1000) and Loc_Table).
```

Ainsi donc, à l'envoi d'une nouvelle commande, le système procède à une expertise des données en fonction de l'ensemble des règles associées au propriétaire de la carte. Trois résultats peuvent être obtenus:

- La transaction semble cohérente avec le schéma entré et est validée par le système. L'exécution de la commande s'en suit
- La transaction est anormale et est rejetée par le système
- Les opérations douteuses (qui s'éloignent légèrement des inflexions données par le schéma) sont détectées par le système. A ce niveau trois opérations peuvent être envisagées:

- Vérification "à la volée" avec le serveur bancaire ou un organisme habilité afin de vérifier que la carte n'a pas été préalablement déclarée volée ou perdue

- Rejet ou acceptation de la transaction. Dans ce dernier cas, la carte peut augmenter son niveau de sécurité pour éventuellement refuser une prochaine commande douteuse. Il s'agit ici d'augmenter la côte d'alerte de la carte. On comprend bien l'intérêt qu'il y a à conserver un historique des transactions déjà effectuées dans ce cas de figure

La politique qui consiste à augmenter la côte d'alerte est laissée au libre arbitre de l'organisme habilité. La discussion sur les seuils d'acceptation ou de refus comme les choix de transactions exceptionnelles ne seront pas discutées ici. En revanche, il est intéressant de voir comment le S-Shell peut gérer, une fois la politique choisie, ces niveaux de sécurité.

Supposons que le choix a été fait de refuser une transaction ambiguë après une série de transactions douteuses quand une des deux conditions suivantes apparaît être vraie:

- C1: plus de cinq transactions dans une même semaine dans un lieu inconnu
- C2: plus de trois transactions dans une même semaine au-delà d'un certain seuil (soit montant/achat, soit montant/semaine)

Ces précisions quant au schéma impliquent une description plus affinée de la partie 'action associée à une commande' du S-Shell. Par exemple, si nous associons des compteurs événementaux associés aux deux conditions imposées, nous pouvons ajouter au schéma de sécurité précédent les attributs ci-dessous:

FIGURE 22 Schéma approfondi d'un profil client d'une application bancaire

```
declaration

nb_lieu integer 0          /* condition C1 */
nb_seuil integer 0         /* condition C2 */

description

default { nb_seuil ++ } :
if ((montant/jour <= 5000 ) or (nb_seuil <= 3)) ;
liquidité { nb_seuil ++ } :
if (((montant/jour <= 500) and (montant/semaine <= 2000)) or (nb_seuil <= 3))) ;
liquidité { nb_lieu ++ } :
if ((table = FALSE) and (nb_lieu <= 5)) ;

etc ...
```

3.10.4 Conclusion

On voit qu'en plus de fournir un outil sécurisé pour la description de ces schémas, le S-Shell de par sa puissance d'expression permet d'écrire les conditions citées. De plus, le fait de pouvoir facilement découper les expressions liées aux conditions de refus des transactions douteuses permet une relecture directe du schéma sans avoir à utiliser des successions de parenthèses qui compliqueraient la description. De nombreux problèmes qui ne sont absolument pas dus au S-Shell se posent malgré tout. Comment initialise-t-on le schéma? Comment faire l'évaluation? Utilisera-t-on des méthodes de systèmes experts ou de réseaux neuronaux? Une étude comparative est effectuée dans [AT95]. Pour un réseau neuronal doit-on aussi simuler le comportement d'un fraudeur? Des réponses sont à ce propos disponibles dans [A95, chapitre 3].

3.11 Conclusions

Ce chapitre avait pour vocation d'initier le lecteur à la fois au S-Shell et à la manière dont se décrit un schéma de sécurité. En ce qui concerne les exemples d'application, il apporte en fait certaines améliorations. Par l'intermédiaire du premier exemple, il montre son intérêt quant au gain qu'il offre à la fois en complexité et en nombre de lignes de description par rapport aux deux autres approches étudiées. Les exemples suivants mettent en évidence certaines nouvelles fonctionnalités que l'on peut obtenir grâce au S-Shell sans perte de sécurité. Par exemple la séquentialité des actions est réalisée dans l'exemple 2 et la prise en compte séparée d'actions par répétition de la même commande afin d'alléger l'écriture et d'élargir le champ des possibilités dans l'exemple 3.

Même si ceci est très encourageant quant à l'utilité de notre prototype, il n'en reste pas moins qu'il n'a jamais été réellement inclus dans le cadre d'une étude beaucoup plus complexe que les cas d'étude présentés ici. C'est pourquoi il semble important de se pencher sur une véritable intégration de cet outil dans un contexte plus large et plus travaillé afin de bien se rendre compte si oui ou non le S-Shell apporte une réelle amélioration. Ceci sera fait au chapitre 5, dans un projet adapté. En effet il s'agit de la réalisation d'un prototype innovant de Carte Blanche, terme qui sera défini par la suite, dont une des préoccupations majeures sera d'assurer un niveau de sécurité très puissant. Le S-Shell ne devra pas redéfinir les concepts évolués de la Carte Blanche mais permettre de l'utiliser dans des conditions de sécurité absolues, sans quoi cette dernière perdrait toute crédibilité quant à une possible diffusion publique.

Mais avant d'en arriver à ce travail, nous voulons revenir en détail sur des remarques qui ont été émises au cours de ce chapitre quant à l'intégrité des données (cf paragraphe 9.2 de ce chapitre). Ainsi le chapitre suivant se consacrera fortement sur ces notions de respect d'intégrité. Nous voulons arriver à montrer qu'en assurant lors de la déclaration des éléments une intégrité forte, nous pouvons assurer en même temps certains contrôles auparavant effectués par le S-Shell et qu'en ajoutant ce vérificateur en amont ou en aval du S-Shell, nous pouvons soulager ce dernier d'un certain nombre de tâches, et ainsi focaliser son intérêt sur les problèmes réels de contrôle d'accès.

Chapitre 4

Un vérificateur d'intégrité pour le S-Shell

4.1 Principe

4.1.1 Intégrité vs sécurité

Parallèlement à notre réflexion sur la description des politiques de sécurité, il nous est apparu que dans le cadre d'un schéma de sécurité complexe, un nombre conséquent d'actions de l'utilisateur ne pouvait pas être jugé légal ou non d'après la seule connaissance de la commande elle-même. C'est ainsi que pour une mini-carte porte-monnaie électronique, le règlement d'un achat ne devrait être accepté que si le solde du porte-monnaie après opération reste positif.

La proposition formulée dans le chapitre précédent permet de décrire un peu plus précisément ce type de règlement, en particulier par l'usage des variables du S-Shell et par le test des paramètres des commandes. Toutefois, le parti-pris de dissocier les données propres à la sécurité des données applicatives entraîne, lors d'un tel usage, une redondance des informations; le solde du porte-monnaie se retrouve à deux endroits différents et doit être tenu à jour à la fois par le S-Shell et par l'application elle-même.

Dans ce cas de figure, il semble bien plus intéressant d'exprimer les règles d'acceptation non sur la commande elle-même ou ses paramètres ou encore d'autres valeurs circonstancielles, mais sur le résultat de cette commande et plus particulièrement sur des propriétés qui devront être vérifiées constamment sur les données applicatives. Cette description bien plus naturelle permet d'éviter un foisonnement de règles là où une seule règle est nécessaire. Il suffit pour s'en convaincre d'imaginer l'exemple du porte-monnaie électronique doté de nombreuses fonctions supplémentaires d'achat de timbres et de tickets de toutes sortes.

Cette réflexion rejoint tout un ensemble de préoccupations présentes dans le domaine des bases de données; les contraintes d'intégrité [Bi77 , DJ93]. Ce terme regroupe les idées et les techniques qui

permettent d'exprimer et de vérifier un certain nombre de propriétés sur les données d'une base. Ces propriétés peuvent être appliquées de manière plus ou moins directe à différents stades de l'exploitation d'une base de données [AF94].

- Les propriétés les plus évidentes portent sur les différentes valeurs que peut prendre une donnée (un salaire ou une date par exemple). Les indications qu'elles apportent permettent de concevoir la représentation -logique ou physique- des données de la base (application du modèle pour le système d'information, choix des clés primaires/externes)
- Certaines propriétés d'unicité de valeur ou d'identification aident le concepteur de la base à élaborer un schéma plus efficace en exhibant ainsi des liens entre les éléments de données ou en induisant des clés ou index plus ou moins complexes
- D'autres propriétés permettent de mettre en oeuvre des contrôles qui, appliqués aux manipulations de données, permettent de réduire la portée d'erreurs logicielles ou de saisies

Les idées émises dans ce chapitre sont relativement récentes. Le prototype est en cours de réalisation; par conséquence les choix d'implémentation sont toujours sujets à évolution et la complexité des algorithmes n'a pas encore été évaluée.

4.1.2 Objectifs

A partir de ces remarques, nous avons étudié comment les contraintes d'intégrité décrites ci-dessus pouvaient être exploitées tant à des fins de contrôle d'accès [JS90] qu'afin de promouvoir la fiabilité des données. L'objectif désigné est de concevoir un outil logiciel ayant un fonctionnement similaire à celui du S-Shell. Les entrées de cet outil comporteront les éléments suivants:

- Une description du schéma de la base de données (selon un modèle relationnel)
- Une description de la représentation physique des données (les données peuvent être stockées sous différentes formes et différents codages)
- Un certain nombre de règles d'intégrité basées sur le schéma de la base

On s'appliquera donc dans un premier temps à définir rigoureusement l'environnement de travail de cet utilitaire. Un élément essentiel de cette description portera sur un langage de description des contraintes d'intégrité.

4.1.3 Applications

Nous avons imaginé deux applications à un tel utilitaire:

- Une analyse déconnectée d'un vidage mémoire (de carte ou de fichier) permettrait de détecter diverses anomalies ayant entraînés une panne logicielle
- La validation directe de transactions. Il serait intéressant d'insérer cet utilitaire dans la chaîne de modification des données, autorisant ainsi le contrôle direct et immédiat des modifications apportées, donc la certitude du respect des contraintes d'intégrité à tout moment

4.2 Différents types de contrôle d'intégrité

Nous voulons avant d'aller plus loin dans la description de notre outil, effectuer une sorte d'analyse visant à énumérer tous les types de contraintes d'intégrité [Sa91] pouvant s'exercer sur un schéma relationnel.Afin de ne pas rendre cette étude trop théorique, nous illustrerons notre propos à l'aide d'exemples. Toutes les expressions relatives aux contraintes d'intégrité seront extraites d'un unique schéma relationnel. Nous considérons donc le schéma relationnel suivant:

FACTURE (NumFact , NumClient)

LIGNE (NumLigne , NumFact , NumArticle , PrixUnitaire , Réduction , Payé)

4.2.1 Contraintes d'intégrité définies sur un attribut unique

Ceci concerne principalement les contraintes sur les valeurs possibles d'un attribut ou sur la cardinalité d'un attribut.

- Contrainte 1: `PrixUnitaire` est une valeur positive ayant au plus 2 décimales

- Contrainte 2: La `Réduction` ne peut excéder 30% tout en restant positive

- Contrainte 3: Les `Factures` sont numérotées successivement à partir de 1

4.2.2 Contraintes d'intégrité définies sur des N-uplets

Les contraintes peuvent porter sur la totalité d'un N-uplet. Ainsi pour vérifier la règle:

- Contrainte 4: Une `Réduction` ne peut dépasser 250F,

il faut considérer à la fois le prix unitaire avant remise et le taux de remise (`Réduction`).

4.2.3 Contraintes d'intégrité définies sur un schéma de relations

Les contraintes suivantes portent sur la globalité d'une table et ne peuvent être vérifiées qu'au vu de l'ensemble des données de la table. Cette catégorie comporte entre autres:

- La définition d'un schéma de relations est elle-même une contrainte d'intégrité. Chaque attribut est un composant d'au moins un schéma de relations. Chaque mise à jour de la base de données doit être faite en respectant la définition des relations
- L'unicité d'une clé simple ou composée

- Enumération d'une sous-liste d'une liste d'attributs d'une relation qui détermine l'élément obligatoire de la relation: Les valeurs de ces attributs doivent être connues.
- Spécification conditionnelle des valeurs d'un attribut d'une relation en fonction des constantes ou des valeurs d'un ou de plusieurs autres attributs
- Cardinalité de la relation

- Contrainte 5: Une instance de (`NumLigne , NumFacture`) doit être unique dans `Ligne`

4.2.4 Contraintes d'intégrité entre relations

Les contraintes suivantes portent sur la globalité d'une table et ne peuvent être vérifiées qu'au vu de l'ensemble des données de la table. Cette catégorie comporte entre autres:

- Contraintes dynamiques [WMW89]: Nous ne considérons plus les contraintes sur les valeurs ou états de leurs données à un certain moment mais sur leur évolutivité d'un état à un autre

- Contrainte 6: Une `Facture` peut être réglée à n'importe quel moment, mais une seule fois

- Spécifications des valeurs

- Contrainte 7: Le montant total d'une `Facture` doit être supérieur à 1000 francs

- Contraintes de cardinalité: Dans notre exemple, nous pouvons exprimer que le cardinal de `NumFact` de `Ligne` est au plus égal au cardinal de `NumFact` de `Facture`
- Contraintes d'inclusion: Ceci sert à spécifier que les valeurs d'un attribut d'une relation sont incluses dans l'ensemble des valeurs d'un domaine d'une autre relation.
- Contraintes d'égalité et contraintes de comparaison: Ces contraintes sont du même genre que les contraintes d'inclusion. Nous nous pencherons sur le fait que les ensembles à comparer sont obtenus par application des opérateurs relationnels.
- Intégrité référentielle et clé externe:

- Contrainte 8: Chaque `NumFact` de `Ligne` doit exister dans `Facture`

4.3 Moyens d'expression des contraintes

Notre utilitaire devra donc prendre en charge plusieurs tâches dans le processus qui permet de décrire les données et les contrôles à examiner.

La première de ces tâches est la prise en compte de la description des données [Da87]. Nous utiliserons pour cela le modèle relationnel désormais bien connu. Nous décrirons donc les données examinées par un certain nombre de relations, chaque relation résultant à son tour de l'agglomération d'attributs. Dans les SGDB classiques [Bu90], ces attributs sont caractérisés par un nom et par un

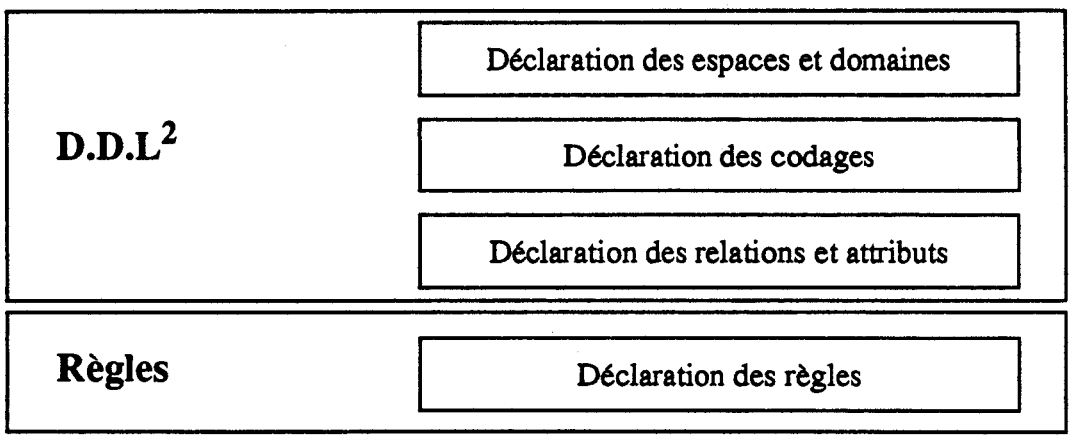
type de donnée. C'est là que l'on précise qu'un montant est représenté par un réel et non par une chaîne de caractères.

Nous irons plus loin encore dans cette idée en reprenant le concept de domaine [JS90]. Un domaine est l'ensemble des valeurs que peut prendre un attribut. Il s'agit donc au moins d'un type simple (au sens des langages de programmation), mais peut également être une restriction de type simple (entier positif, valeur de 1 à 100, jour de la semaine, etc ...). Cette première tâche peut être considérée comme remplaçant *a posteriori* le langage de description des données des SGBD ou encore les ordres "CREATE" du langage SQL. Le terme a posteriori s'explique par le fait que les données existent déjà et que la description a pour objectif de permettre l'interprétation des données[1]. La description du schéma logique de la base doit être complétée par des indications sur la manière dont sont représentées les valeurs (taille des entiers, codage des chaînes de caractères, éventuellement codage de valeurs particulières).

La deuxième tâche est l'interprétation et l'évaluation des règles d'intégrité [Co89]. Le langage établi à cet effet reprend la formulation des propriétés ensemblistes des mathématiques, comprenant les quantificateurs classiques (quelquefois, il existe, il existe un seul).

Le découpage d'un schéma d'intégrité peut ainsi être dessiné comme suit:

FIGURE 23 Découpage d'un schéma d'intégrité



1. Dans les SGBD classiques, la description des données est compilée et utilisée lors de toute manipulation (ajout, mise à jour, consultation). Celle-ci existe donc en préalable à toute donnée

2. Data Description Language: Spécification des données

4.4 Concept du langage

Le langage de description des contraintes explicite les quatre étapes mentionnées au paragraphe précédent.

4.4.1 Description des espaces et des domaines

Le concept d'espace utilisé ici est proche de la notion de type dans les langages de programmation. Toutefois, le concept d'espace est plus restrictif que la notion de type qui comporte quatre volets:

- Un type définit l'ensemble des valeurs que peut prendre une variable. ADA [Wi89] propose même la notion de sous-type qui permet de construire un nouveau type par restriction des valeurs d'un type existant
- Un type induit l'ensemble des opérations existant sur les valeurs du type (voir les types abstraits). C'est ainsi qu'on liera les opérations arithmétiques aux entiers ou aux réels ou la concaténation aux chaînes de caractères. Cet aspect n'est pas repris dans notre modèle; seules les opérations arithmétiques et logiques sur les espaces prédéfinis sont utilisables. Un nouvel espace ne sera donc doté que des opérations de comparaison
- Un type est indissociablement lié à la manière dont sont représentées ses valeurs en machine. Le langage C par exemple distingue différents types entiers ou flottants selon la précision désirée. Dans notre modèle, cet aspect n'est pas directement pris en compte par la notion d'espace mais est repris par la définition des codages. Cette méthode a l'avantage d'autoriser des représentations différentes pour des attributs associés au même domaine, donc au même type
- Le typage est un outil précieux de vérification de vraisemblance des programmes. Par une démarche proche des équations aux mesures de la physique, il est possible de détecter automatiquement certaines erreurs (additionner des francs et des mètres par exemple). De ce fait, il peut être fort utile de distinguer deux types (francs et mètres) là où un seul type serait suffisant vis-à-vis des trois autres aspects (les deux seront représentés par des réels, en flottant et disposent des mêmes opérations). Nous n'avons pas introduit cet aspect dans notre modèle par manque de temps. Mais ce pourrait être un outil performant de contrôle de vraisemblance des règles d'intégrité

Dans ces remarques, nous avons délibérément négligé un dernier aspect des types dans les langages de programmation et non le moindre; les mécanismes courants de construction de types (tableaux, structures, pointeurs) permettent la description de types complexes mémorisant des entités complètes pouvant même avoir leur comportement propre [Th90 , CL91]. Cette possibilité est redondante voire contradictoire avec la description des entités d'une base de données. Une étude plus poussée de cette équivalence pourrait d'ailleurs être intéressante (pour pouvoir modéliser simplement les relations représentées par des tableaux de données), mais tombe hors de notre propos.

Dans cette première version simple de notre prototype, nous avons inclus les espaces prédéfinis Entiers, Réels, Caractères, Chaînes et laissé au rédacteur la possibilité de définir des types énumérés.

Comme en SQL [CP89], il existe dans tous les espaces prédéfinis une valeur notée NULL qui indique l'absence de valeur utile. La représentation physique de cette valeur sera vue au paragraphe suivant.

Quand un type a été défini, il est possible de considérer un sous ensemble des valeurs du type et de l'identifier sous forme de domaine. Une fois défini le type «jour de la semaine», est possible de définir le domaine «jour ouvrable» du lundi au vendredi. Un tel domaine peut alors caractériser les valeurs possibles d'un attribut.

4.4.2 Description des codages

En toute généralité, la diversité des représentations possibles pour des types même ordinaires est très importante. En effet, si l'on considère les représentations binaires, symboliques, avec ou sans codage et toutes leurs compositions, on réalisera vite que le dénombrement des représentations possibles est un gros travail. Nous avons donc rejeté un mécanisme d'identification unique de chaque codage et nous l'avons remplacé par un mécanisme d'assemblage de codages simples et en nombre limité.

Les codages simples sont caractérisés par le domaine d'entrée et le domaine de sortie. Un domaine d'entrée (resp. de sortie) est l'ensemble des valeurs que sait coder (resp. décoder) un codage par exemple l'instruction `printf(«%d», ...)` du langage C peut être considérée comme un codage ayant l'ensemble des entiers pour domaine d'entrée et les chaînes de caractères pour domaine de sortie. Les domaines considérés appartiennent indifféremment à des espaces abstraits (entiers, réels, chaînes de caractères) ou à des espaces de représentation (binaire, texte) où se situent les représentations ultimes des données (fichier, vidage mémoire, etc). On peut par exemple déclarer qu'une instruction d'entrée-sortie binaire (fonction `write` du langage C) implémente un codage des entiers vers une représentation binaire de *n* (variable selon l'implémentation) octets.

A ces codages prédéfinis, l'utilisateur peut ajouter ses propres codages afin de décrire la représentation de types utilisateurs ou encore de préciser le codage de la valeur NULL. Il est ainsi possible de définir la représentation des jours de la semaine par la ligne:

```
Codage
   Jour = (Lundi=1,Mardi=2,Mercredi=3,Jeudi=4,
           Vendredi=5,Samedi=6,Dimanche=7,NULL=0) ;
```

Ces codages simples peuvent être assemblés entre eux pour réaliser un grand nombre de codages courants. On distingue dans notre réalisation deux sortes d'assemblages:

- L'assemblage parallèle.
  Deux codages c1 et c2 ayant des domaines d'entrée exclusifs mais appartenant à un même espace peuvent se compléter. Le codage d'une valeur de l'espace utilisera c1 si cette valeur

appartient au domaine d'entrée de c1, c2 si cette valeur appartient au domaine d'entrée de c2. Pour obtenir un codage exploitable, il faut également que l'intersection des domaines de sortie soit vide. Quand cette condition n'est pas respectée, le codage n'est pas une injection et le décodage peut être ambigu. Cette opération est notée `(c1+c2)`.
Exemple: l'assemblage parallèle des codages Jour1 et Jour2 suivants sera équivalent au codage `Jour` défini plus haut.

```
Codage
   Jour1 = (Lundi=1,Mardi=2,Mercredi=3,Jeudi=4,
            Vendredi=5,Samedi=6,Dimanche=7) ;
   Jour2 = (NULL=0) ;
```

- L'assemblage série.
  Dans une association de ce type le domaine de sortie d'un codage cA est inclus dans le domaine d'entrée du second codage cB (tous deux sont donc dans le même espace). Le domaine d'entrée du codage résultant est donc celui de cA et le domaine de sortie du codage résultant celui de cB. Cette opération est notée `(cA | cB)`.
  Exemple: l'assemblage série des codages `JourA` et `JourB` suivants sera équivalent au codage `Jour` défini plus haut.

```
Codage
   JourA =(Lundi='L',Mardi='M',Mercredi='m',Jeudi='J',
           Vendredi='V',Samedi='S',Dimanche='D') ;
   JourB = ('L'=1,'M'=2,'m'=3,'J'=4,'V'=5,'S'=6,'D'=7) ;
```

4.4.3 Description des relations et attributs

Conformément au modèle relationnel, l'ensemble de la base de données est vu comme un ensemble de relations -ou tables- composées du point de vue structurel d'un ensemble d'attributs et dont l'extension est un ensemble de N-uplets (1 N-uplet = 1 ligne de la relation, 1 attribut = 1 colonne).

On ajoutera à cette description logique un certain nombre d'indications sur la représentation physique des données (nom de fichier, format, etc ...):

Pour une relation, le type de représentation (fichier binaire, fichier texte, CQL) induira le domaine de sortie du codage employé par chacun des attributs de la relation.

Pour un attribut, on indiquera le codage employé. Le domaine d'entrée de ce codage devra contenir le domaine associé à l'attribut et le domaine de sortie du codage devra être compatible avec le type de représentation de la relation.

4.4.4 Contraintes complexes

Comme nous l'avons précisé plus haut, les règles d'intégrité complexes pourront être décrites à l'aide d'ensembles, de quantificateurs et des variables déduites. La règle courante d'unicité de la clé `NumFact` dans la relation `Fact` pourra être exprimée ainsi:

```
Forall nx In Facture (NumFact),
    ExistsOne (nf, nc) In Facture (NumFact , NumClient),
                  n=nf.
```

ou encore:

```
Forall nx In Facture (NumFact),
    CARD (Select(nf,nc) From Facture (NumFact ,NumClient)) = 1
```

Ou de manière plus simple:

```
pour chaque valeur de NumFact, il existe un seul N-uplet de Facture
possédant cette valeur.
```

On remarquera ici la présence des quantificateurs **`Forall`** et **`ExistsOne`** ainsi que l'utilisation des ensembles Facture.NumFact (extension de NumFact soit l'ensemble des valeurs prises par cet attribut). Ces quelques modèles suffisent pour exprimer une très grande part des règles habituelles.

En voici une description plus précise.

4.4.4.1 Quantificateurs

Le langage permet d'utiliser trois quantificateurs. Classiquement ces quantificateurs sont associés à trois éléments:

- Un nom de variable (cette variable est alors dite liée) qui représentera la valeur de chaque élément de l'ensemble lors du parcours
- Un ensemble
- Une proposition logique

Les quantificateurs sont:

- «Quelquesoit» (`Forall`) indique que la proposition doit être vraie pour chacune des valeurs de l'ensemble
- «Il existe» (`Exists`) indique que la proposition doit être vraie pour au moins une valeur de l'ensemble
- «Il existe un seul» (`ExistsOne`) indique que la proposition doit être vraie pour une valeur de l'ensemble et fausse pour toutes les autres

4.4.4.2 Ensembles

Les ensembles peuvent être utilisés avec les quantificateurs ou évalués comme sous-expression. Les ensembles peuvent être construits de différentes manières:

- Statiques: quand ils sont donnés explicitement dans la règle ou qu'ils résultent d'ensemble de valeurs connues à la compilation (types, domaines). Une combinaison d'ensembles statiques est statique.
- Dynamiques: quand ces ensembles sont calculés d'après les données présentes dans la base. Il s'agira en général du résultat d'une requête type SQL avec jointure, projection et restriction.

4.4.4.3 Propositions logiques.

Elles sont construites de manière assez classique avec les opérateurs logiques, arithmétiques et de comparaison.

4.5 Exemple

4.5.1 La déclaration des ensembles

Afin d'illustrer notre propos, nous avons choisi d'illustrer l'exemple présenté en section 4.2 de ce chapitre et largement développé par la suite dans les différents contextes proposés. Pour expliciter à l'aide de notre langage des règles fortes d'intégrité et fonction de ce qui a été souligné précédemment, nous suggérons le découpage suivant:

- Les espaces de travail: Nous définissons notre espace en cinq parties, dont un type énuméré, à savoir:

  ESPACE

  NoFacture = 1 .. 1 000 000 ;

  NoLigne = 1 .. 1000 ;

  NoClient = STRING(20) ;

  NoArticle = STRING(20) ;

  TypArt = (Clavier, Ecran, UC, Memoire, Disque, NULL) ;
- Les domaines de travail: il s'agit ici d'exprimer les domaines relatifs au prix net et à la remise. Nous considérerons la remise en pourcentage. Nous obtenons alors:

  DOMAINE

DPrixNet = [REAL] [0, MAXREAL] ;

DRemise = [REAL] [0.0, 100.0] ;

- Le codage a été réalisé en écrivant:

CODING

TypArt.Num = (Clavier=1, Ecran=2, UC=3, Memoire=4, Disque=5, NULL=0) ;

TypArt.Car = (Clavier='C', Ecran='E', UC='U', Memoire='M', Disque='D', Null=' ') ;

- Les relations qui découlent immédiatement de 4.2 donnent:

RELATION

```
FACT    = TEXTFILE("FACT.TXT") SEPARATORS('\t, '\n) {
            NumFact : NoFacture CODING INT.STR,
            NumClient : NoClient CODING STR,
            } ;
LIGNE = BINARY FILE("LIGNE.DAT") {
            NumLigne : NoLigne CODING INT.WORD,
            NumFact : NoFacture CODING INT.LONG,
            NumArt : NoArt CODING STR.ASCII(20),
            TypArt : TypArt CODING (TypArt.Num | INT.BYTE),
            PN : DPrixNet CODING REAL.FLOAT,
            Remise : DRemise CODING REAL.FLOAT
            } ;
```

4.5.2 La description des contraintes

Nous allons maintenant reprendre la description des huit contraintes qui ont été énoncées tout au long de ce chapitre et voir comment nous pouvons les exprimer simplement à l'aide de notre outil.

- **Contrainte 1** : Le prix net (PN) est un nombre positif avec deux décimales.

```
ForAll PN In Ligne.PN,
   INT(PN*100) - PN*100 = 0 ;
```

- **Contrainte 2** : La remise (REMISE) ne peut excéder 30 % (soit 0.30).

```
ForAll Remise In Ligne.Remise,
   PN <= 30.0 And PN >= 0.0 ;
```

- **Contrainte 3** : Les factures sont numérotées séquentiellement à partir de 1.

```
ForAll NumFact In Fact.NumFact,
   (ExistsOne NF In Fact.NumFact,
   NumFact = NF) And
   Not (Exists NF In Fact.NumFact,
   NF > CARD(Select NumFact FROM Fact) ) ;
```

- **Contrainte 4** : Une remise ne peut dépasser 250 francs.

```
ForAll (PN, Remise) In Ligne(PN, Remise),
   (PN * Remise / 100) <= 250 ;
```

- **Contrainte 5** : L'unicité d'une clé simple ou composée.

```
ForAll (NF, NL) In Ligne(NumFact , NumLigne),
   ExistOne (NF2, NL2) In Ligne(NumFact, NumLigne),
   ( ( NF=NF2) AND (NL=NL2) ) ;
```

- **Contrainte 7** : Le montant d'une facture doit dépasser 1000 francs.

```
ForAll (NF) In Fact(NumFact),
   SUM(Select PN*Remise/100 FROM Ligne),
   WHERE NumFact = NF) > 1000 ;
```

- **Contrainte 8** : Tous les numéros de facture dans LIGNE doivent exister dans FACT (intégrité référentielle).

```
ForAll (NF) In Ligne(NumFact),
   Exists NF2 In Fact(NumFact),
   NF = NF2 ;
```

4.6 Conclusion

Ainsi donc, nous voyons qu'il est possible d'associer à un schéma de sécurité un schéma d'intégrité qui peut 'soulager' notre S-Shell d'un certain nombre de vérifications (voir paragraphe 1 de ce chapitre). Par soulager, nous insistons bien sur le fait que notre vérificateur ne sert que d'aide au S-Shell en se basant sur le même genre de construction d'implémentation. Sa simplicité de réalisation

(*ie* son analogie avec le S-Shell) est pour beaucoup dans son existence même. L'idée motrice est de le placer en série avec le S-Shell, en amont ou en aval, de façon à effectuer à un moment quelconque entre l'émission de la commande et l'exécution de ce contrôle d'intégrité.

Le véritable rôle du S-Shell se limitera dès lors à juger de la légalité d'une action en fonction de la commande elle-même. Dans toute la suite de cet exposé, nous laisserons de côté cet aspect intégrité pour se concentrer plus en détail sur les fonctionnalités propres du S-Shell. Ainsi donc ce supplétif nous sera d'une réelle utilité dans la réalisation effective de l'outil. Pour l'heure, il paraît plus important de se focaliser sur les vrais problèmes que posent à la fois l'intégration du S-Shell dans un projet innovant et dans un environnement carte à microprocesseur incluant les nombreuses restrictions d'usage. Ce sera l'objet des deux chapitres suivants.

Chapitre 5

Intégration dans un système d'exploitation Carte Blanche

5.1 Intérêt de la démarche

Après avoir étudié l'approche S-Shell et montré son efficacité par l'intermédiaire de quelques exemples appropriés, nous voulons approfondir notre démarche et inclure notre schéma dans un sujet plus vaste et plus concret. Afin de mettre en évidence ses fonctionnalités en tant que prototype réalisé sous Unix, nous avons choisi de l'intégrer dans un projet de recherche mené à RD2P lui aussi créé dans le même environnement. Le projet est de concevoir un système d'exploitation [Lee95] pour Carte Blanche (c'est-à-dire vierge de toute application) [CP93]. Nous avons choisi d'adapter notre S-Shell dans ce travail de recherche afin de:

- bénéficier d'une grande compatibilité de programmation et donc de faciliter l'intégration du S-Shell
- bénéficier de l'opportunité d'un nouveau projet intégrant des problèmes essentiels de sécurité
- apporter une réelle amélioration en matière de sécurité à un outil encore non finalisé et par là-même aider à sa concrétisation plutôt que de reprendre un logiciel déjà existant dont la fusion avec le S-Shell serait plus dure à identifier pour un intérêt peut être plus minime

Afin que lecteur puisse bien comprendre ce que nous voulons effectuer, nous nous proposons de faire un rapide rappel sur les concepts fondamentaux de la Carte Blanche, sachant qu'en plus du document cité ci-dessus, la référence [Pe95-1] est aussi disponible à ce sujet et offrira au lecteur une vue plus élaborée du fonctionnement de ce nouveau système d'exploitation orienté objets nomades tel que la carte à microprocesseur. Ensuite, nous verrons en quoi le S-Shell peut concrètement répondre à des besoins de sécurité précis que nous détaillerons par la suite et comment nous pouvons l'intégrer.

5.2 Le système d'exploitation Carte Blanche

5.2.1 Présentation

Comme nous l'avons décrit dans le chapitre 1, une des spécificités de la carte à microprocesseur réside dans l'existence d'un cycle de vie comportant quatre étapes ordonnées. Durant la première étape dite de personnalisation, le super-utilisateur unique peut installer une voire plusieurs applications, une application étant une structure de données, quelques codes secrets et des droits d'accès sur ces données. Lors de cette même phase, l'identité du porteur est inscrit une fois pour toutes dans la carte. Cette dernière entre alors dans sa phase d'utilisation [B88]. Une fois cette phase de personnalisation achevée, plus personne ne peut modifier les structures de données de la carte. L'identité de l'utilisateur ne peut non plus être changée, même par le super-utilisateur.

De plus, la carte ne peut pas effectuer d'opérations complexes sur les données. Les utilisateurs peuvent avoir accès à leurs propres données grâce à un certain nombre de commandes système telles que 'read' ou 'write' en fonction des privilèges accordés par le super-utilisateur (l'authentification d'un utilisateur est faite par présentation d'un code secret). Ainsi, pour obtenir un groupe de commandes plus évoluées sur les données, ces dernières doivent d'abord sortir de la carte avant d'être traitées dans un environnement extérieur plus puissant. Le résultat de ce traitement sera ensuite inscrit dans la carte. Ceci veut dire que pour obtenir le résultat d'une commande complexe, il est nécessaire d'échanger des données entre la carte et le système. Ce protocole est bien évidemment source d'une importante perte de sécurité.

Une solution classique visant à renforcer la sécurité des échanges de données est d'effectuer une série d'opérations de chiffrement/déchiffrement sur ces données, mais dans un environnement aussi restreint ces opérations prendraient certainement plus de temps que le traitement lui-même [C90], ce qui n'est assurément pas souhaitable. Une autre solution hypothétique serait de créer un masque spécifique, c'est-à-dire un programme écrit en ROM, en addition avec de nouvelles commandes adaptées à un traitement de données particulier. Ceci est malheureusement trop coûteux et ne peut pas être toujours utilisé.

Ces caractéristiques résultant à la fois du haut degré de sécurité requis et des contraintes matérielles, imposent une certaine rigidité de la carte à puce. Ainsi les applications ne peuvent être ni mises à jour ni supprimées. Il est de plus impossible d'installer de nouvelles applications. Même si l'existence d'un code applicatif dans la carte est quelque chose de réalisable techniquement, cette solution n'est ni bon marché ni pratique.

De façon à résoudre ces problèmes, un nouveau type de système d'exploitation orienté carte a été réalisé sous le nom de Carte Blanche. Son but est entre autres de permettre l'installation et la suppression de nouvelles applications sur la carte tout en assurant un haut niveau de sécurité.

La flexibilité de la Carte Blanche est obtenue en autorisant à la fois les applications et les partenaires (un partenaire étant une entité que la carte peut identifier et authentifier) à entrer et à sortir de la

carte pendant sa phase d'utilisation. Généralement, ces opérations ne peuvent être réalisées que par le super-utilisateur durant la phase de personnalisation. Dans la Carte Blanche, le porteur demande à un émetteur d'applications (par exemple sa banque) d'installer ou de retirer une application de sa carte. Le porteur devient l'administrateur de sa propre carte. La carte évolue en fonction des besoins de l'utilisateur.

De plus, le porteur n'a pas besoin d'être enregistré par la carte. Toutefois une application peut nécessiter une identité. Dans ce cas on peut avoir recours à un tiers parti de confiance qui pourra prouver l'identité du porteur. Cette identité ne peut pas être altérée.

Le code de chaque application de la Carte Blanche représente un ensemble de services. Chaque service peut être appelé individuellement. Les données des applications sont encapsulées dans les services. En d'autres termes ceci veut dire que les données peuvent être lues, écrites ou mises-à-jour par les services de la même application mais pas par aucune autre commande ou service. Le contrôle d'accès de l'application est administré au niveau du service et non plus au niveau des données. Un utilisateur peut appeler un service d'une application, mais ne peut jamais avoir un accès direct aux données. L'avantage de ceci est que les actions effectuées sur les données des applications sont définies par l'émetteur d'application, en assurant l'intégrité de toute l'application.

Le fait de pouvoir utiliser du code utilisateur dans la carte nécessite des mesures de protection du système spécifiques (voir chapitre 1). L'accès à la mémoire doit être géré de manière à protéger une modification illégale du contenu de la mémoire d'une application frauduleuse.

5.2.2 Mise en place de nouvelles applications

La mise en place de nouvelles applications est sécurisé si ces applications vérifient les propriétés suivantes:

- Les applications doivent être indépendantes les unes des autres
- Une application est chargée par un partenaire unique. Ce partenaire sera responsable de son application et sera le seul à pouvoir la modifier. Ainsi les autres partenaires n'auront aucun droit de modification sur cette application

Installer une application, c'est en définitive charger un groupe de données initialisées, du code d'application et une description des différents services. Au chargement, le système vérifie que toutes les ressources requises sont disponibles et vérifie à la fois les données et l'intégrité du code ainsi que la complétude de l'application.

Nous allons maintenant décrire le chargement d'une application dans une carte vide. A ce niveau il n'existe qu'un partenaire prédéfini nommé PUBLIC. C'est le partenaire par défaut connecté quand la carte est mise sous tension.

L'émetteur d'application doit d'abord créer le partenaire qui gérera l'application. Cette création est réalisée par le partenaire PUBLIC. Le nouveau partenaire peut alors se connecter et installer son

application. La création de nouveaux partenaires n'est pas restreinte. La Carte Blanche assure que l'arrivée de nouveaux partenaires ne pertube ni le système, ni la sécurité du système. La création libre de partenaires est possible grâce à l'existence du partenaire PUBLIC. Ceci permet la libre installation d'applications dans la carte.

En guise de conclusion, la sécurité de la carte est assurée par les principes suivants:

- Le partenaire qui crée une application en est le propriétaire
- Seul un propriétaire peut mette à jour ou détruire son application
- Un propriétaire peut utiliser les services de son application une fois celle-ci installée
- Seul un propriétaire peut décider de faire coopérer son application avec d'autres partenaires et leurs applications

5.2.3 Coopération d'applications avec des partenaires et d'autres applications

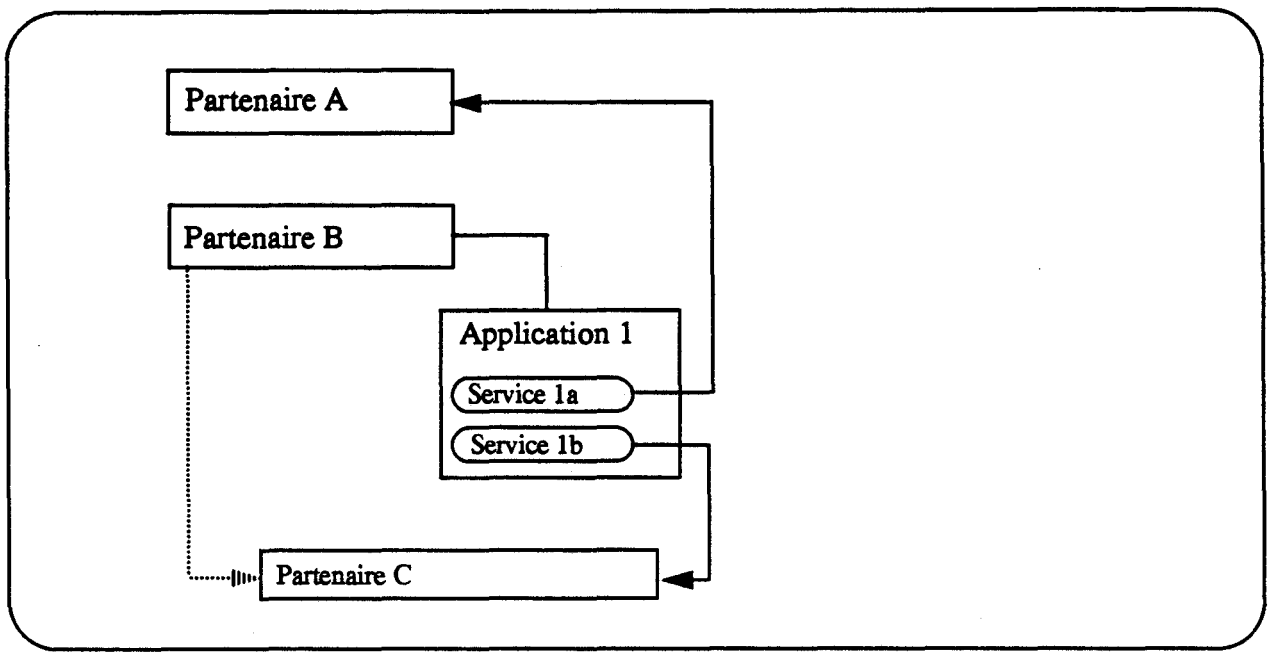
Une fois installée, une application peut être utilisée en appelant ses services. Deux types d'entité peuvent être à l'origine de tels appels:

- Un partenaire peut formuler une requête au système d'exploitation qui aboutit à un appel de service
- Une application peut - pendant son exécution - appeler un service d'une autre application. Il y aura toujours dans ce cas de figure un partenaire qui sera à l'origine de l'appel

Quand une application vient juste d'être installée, son propriétaire est l'unique partenaire qui a le droit d'utiliser les différents services proposés par l'application. Néanmoins, les objectifs d'une application sécurisée ne sont évidemment pas si restrictifs, et nous devons offrir la possibilité à d'autres partenaires de pouvoir appeler d'autres services en fonction de leurs droits respectifs sur ces derniers. En conséquence, le propriétaire d'une application doit offrir ou distribuer des privilèges aux autres partenaires en fonction des droits d'accès qu'il désire donner. De cette manière les partenaires peuvent appeler des services d'une application qui ne leur appartient pas.

Une application peut aussi posséder des services réservés à un partenaire particulier de qui elle dépend. Prenons par exemple, le schéma suivant:

FIGURE 24 Représentation d'une coopération autour d'une application



Sur ce diagramme, le partenaire B est propriétaire de l'application 1. Cette application peut être utilisée au travers des services 1a et 1b. Ici, le partenaire A pourra utiliser le service 1a proposé par le partenaire B tandis que le partenaire C pourra utiliser le service 1b. Il faut de plus souligner que le partenaire C a été créé à partir du partenaire B (ce qui est représenté par des pointillés sur la figure 24). Ainsi C est sous le contrôle de B, ce qui est passablement différent du partenariat créé sous PUBLIC [Pe95-2].

Afin de profiter de la présence de plusieurs applications sur un même support, la Carte Blanche autorise la coopération d'applications. Ce qui veut dire qu'une application peut utiliser un service provenant d'une autre application dans ses services. Par exemple, le service 1a de l'application 1 peut être utilisé par un service d'une autre application détenue par le partenaire A. Ce nouveau service peut être offert à un tiers partenaire. Cette caractéristique est nécessaire pour assurer la coopération entre applications mais est dangereuse vis-à-vis d'une assurance de haute sécurité d'utilisation. Typiquement un propriétaire veut fournir un service à un second partenaire mais ne veut pas que ce service devienne accessible aux autres (via une application du second partenaire). Pour assurer ceci, la Carte Blanche permet de créer des partenaires qui ne peuvent pas installer leurs propres applications. Ils peuvent seulement utiliser les services qui leur sont offerts. Le propriétaire peut créer ce type de partenaire et lui offrir ce service.

En référence au modèle client-serveur [B92], nous appelons un partenaire serveur celui qui peut installer une application et par conséquence offrir des services à d'autres partenaires. A l'opposé, les partenaires qui ne peuvent qu'utiliser les services offerts par les serveurs et qui ne peuvent en aucun

cas installer leurs propres applications sont appelés des clients. Un serveur peut aussi utiliser des services offerts par d'autres serveurs.

5.3 Les besoins de sécurité de la Carte Blanche

5.3.1 Concepts généraux concernant la description de la sécurité

Même si la Carte Blanche offre de nombreuses facilités dans l'installation d'applications et plus généralement dans l'utilisation de la carte à puce, nous devons prêter attention à l'aspect sécuritaire que le système d'exploitation offre au concepteur d'une application ou au porteur de la carte. Nous ne devons pas seulement offrir un niveau de sécurité identique à celui qui existe déjà dans les cartes à microprocesseur d'aujourd'hui. Nous devons élever le niveau de sécurité au niveau des facilités qui sont proposées aux utilisateurs de la Carte Blanche. Tout ce travail serait inutile si nous ne pouvions fournir au système d'exploitation quelques mécanismes qui permettent d'atteindre ce niveau.

5.3.2 Les requis des applications

D'un point de vue général, une application requiert deux principes contradictoires.

Le premier est la protection des données. Deux des principales caractéristiques de cette dernière sont la confidentialité et l'intégrité. Confidentialité veut dire que seules les entités habilitées sont à même de connaître les valeurs courantes des données. Atteindre un tel objectif nécessite des techniques telles que l'authentification qui ne sont pas l'objet de notre discours. En ce qui concerne l'intégrité des données, le concepteur espère que la valeur des données dans la carte est en harmonie avec la réalité. Les techniques usuelles pour atteindre un tel but sont d'accorder une confiance totale à un groupe réduit de personnes qui aura le droit de modifier les données. Ces altérations sur les données étant effectuées de manière consistante. Ceci est réalisé grâce aux techniques de contrôle d'accès.

Le second besoin d'un concepteur d'application est de pouvoir coopérer avec d'autres entités (application ou porteur de carte). Les paragraphes précédents ont montré qu'une meilleure utilisation de la carte pouvait être obtenue en rendant possible la coopération de plusieurs applications chargées dans la carte.

5.3.3 Le développement

Nous allons maintenant décrire les moyens offerts par le système d'exploitation pour atteindre les deux objectifs cités ci-dessus. Nous orienterons notre étude sur les outils logiques et non sur les caractéristiques physiques.

1. Par encapsulation: Ce système d'exploitation utilise les techniques classiques d'encapsulation. Les logiciels de base assure que les données atomiques dans les applications ne peuvent être atteintes qu'au travers des services (au lieu des commandes générales décrites aupara-

vant). En d'autres termes, une souple définition de tous les services d'une application peut renforcer la cohérence de ses données

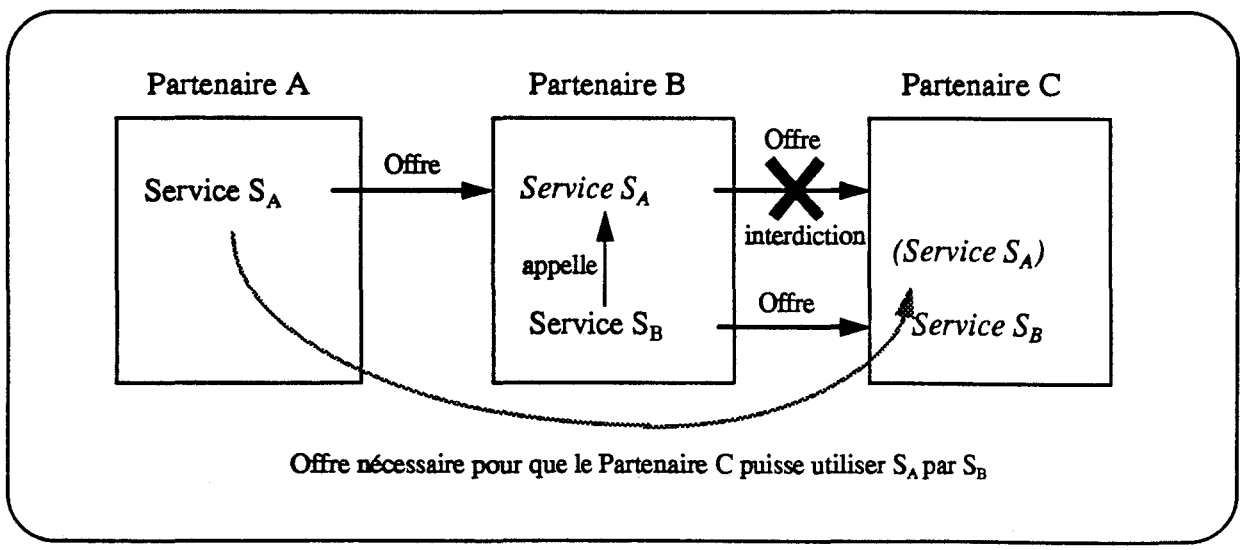
2. En contrôlant l'utilisation des services: L'étape suivante dans la mise en place de la sécurité demande que chaque appel à un service soit fait sous contrôle d'une entité autorisée. Comme nous l'avons précisé ci-dessus, ceci requiert des fonctions d'authentification afin d'identifier l'appelant. Dans le système d'exploitation Carte Blanche, ces acteurs identifiés sont appelés des partenaires. De plus, le système doit offrir à chaque application le moyen de déclarer quel partenaire peut appeler un service

5.3.4 Une première solution

Pour montrer la faisabilité des différents types de coopération avec une application, nous avons mis en place une première implémentation basée sur l'offre de service. Le système permet d'offrir indépendamment chaque service de sa propre application à un partenaire existant de la carte. Les partenaires ayant reçu une offre peuvent dès lors appeler le service offert. Les serveurs peuvent ajouter ou retirer les services à tout moment. Il est par exemple possible de fournir un service à un nouveau partenaire longtemps après l'installation de l'application. Ce même service peut lui-même être retiré sur simple décision du serveur.

Un serveur qui reçoit une offre peut utiliser le service offert dans son application. Il peut aussi offrir n'importe quel service de sa propre application à un tiers partenaire. Par voie de conséquence le service déjà offert peut très bien être de nouveau proposé. Or, nous avons émis comme hypothèse de départ que les partenaires ne devaient pas se faire confiance entre eux. Le propriétaire d'une application doit donc explicitement donner - au potentiel utilisateur final - en même temps que son service les droits associés au service. Comme ces offres peuvent être retirées, le système vérifie avant chaque appel que l'utilisateur final peut effectivement utiliser le service en question, même indirectement.

FIGURE 25 Le mécanisme d'offres



Dans ce schéma, le partenaire A offre un de ses services S_A au partenaire B. Ainsi B peut appeler S_A. B peut aussi posséder une application propre utilisant S_A par un de ses services S_B. B est alors autorisé à offrir son service S_B au partenaire C. Il n'est en revanche pas autorisé à offrir le service S_A qui ne lui appartient pas. Pour que C puisse utiliser S_A il faut que le partenaire A lui offre directement accès à S_A.

Pour définir la coopération de son application, un serveur peut seulement offrir un service aux autres partenaires. A partir de là, un utilisateur pourra ou ne pourra pas utiliser un service. Cependant, le propriétaire peut modifier ses offres. Pour cela, il doit être absolument connecté. Ainsi le contrôle de l'application ne peut jamais être modifié entre deux connections du propriétaire.

Pour conclure, cette solution de modèle de coopération est réalisable, mais nécessite une administration pesante des offres.

- Un serveur peut uniquement offrir un service à un partenaire existant. Il ne peut pas offrir un service à un partenaire futur de la carte
- Un serveur doit être connecté chaque fois qu'il désire modifier ses offres de service. En conséquence, un propriétaire d'application devra se connecter assez fréquemment et ceci ne sera possible que si le porteur le permet
- Comme le modèle suppose qu'il n'y a aucune confiance entre les entités, le propriétaire d'une application client n'est pas libre d'offrir ses services aux autres partenaires. Ces partenaires finaux devront faire une demande afin d'avoir le privilège d'utiliser les appels de service. Si deux propriétaires d'applications coopérantes se font confiance, le système devra rendre réalisable la délégation de privilèges

Ces remarques mettent en évidence certains inconvénients quant à la conception de la coopération d'une application avec les autres. Une politique de sécurité complexe qui requiert la notion de délégation de droits est difficile à mettre en place et demande de fortes restrictions d'utilisation.

5.3.5 Une autre solution: le S-Shell

Nous avons vu que les solutions proposées pour répondre aux spécifications requises par notre problème de sécurité dans le cadre de la Carte Blanche ne répondent pas entièrement à ce que l'on cherche à obtenir (ou moyennant une complexité trop importante).

Comme nous l'avons étudié précédemment, le S-Shell permet de remplacer les outils de bas niveau. Dans notre cas, il peut être utilisé pour permettre à un propriétaire d'application d'exprimer aisément ce qui peut ou ne peut pas être effectué à tout moment.

La granularité de ces spécifications est l'appel de service. L'acceptation d'un appel de service peut dépendre des valeurs des variables ou des valeurs du système. De plus, quand un appel est accepté, quelques actions peuvent être spécifiées dans le but d'être exécutées par le système pour changer des valeurs de variables.

Nous supposons que le texte de description est chargé une fois pour toutes avec l'application qu'il contrôle. Ceci implique que toute modification dans la politique de sécurité (i.e un service devient accessible à un partenaire moyennant certaines nouvelles conditions) doit être prévue avant le chargement définitif de l'application. Ces contraintes permettent au S-Shell d'éliminer le super-utilisateur qui est généralement le point faible de toute politique de sécurité. En dehors de cela, le S-Shell offre la possibilité d'effectuer certaines opérations complexes, comme de ne pas pouvoir accéder à un service plus de trois fois. La complexité du schéma reste assez petite pour être facilement et rapidement comprise. Même si la preuve n'est pas possible pour l'instant, nous pensons que sa petite taille permet une première validation de la politique de sécurité.

5.4 Le S-Shell et la Carte Blanche

Nous allons maintenant voir comment le S-Shell peut être concrètement appliqué à l'environnement de la Carte Blanche.

5.4.1 Les altérations du langage

Il existe un fichier de description par application. Ce fichier décrira la politique de sécurité nécessaire à l'application concernée, c'est-à-dire les conditions de validité d'appel à un service de l'application. Les noms des commandes du fichier de description sont en fait les noms des services de l'application. Le langage utilisé dans les fichiers de description est le même que celui du langage S-Shell, mis à part que les conditions associées aux notions de date et heure reposent sur la possibilité qu'a le système d'exploitation de posséder une horloge en qui on peut avoir confiance.

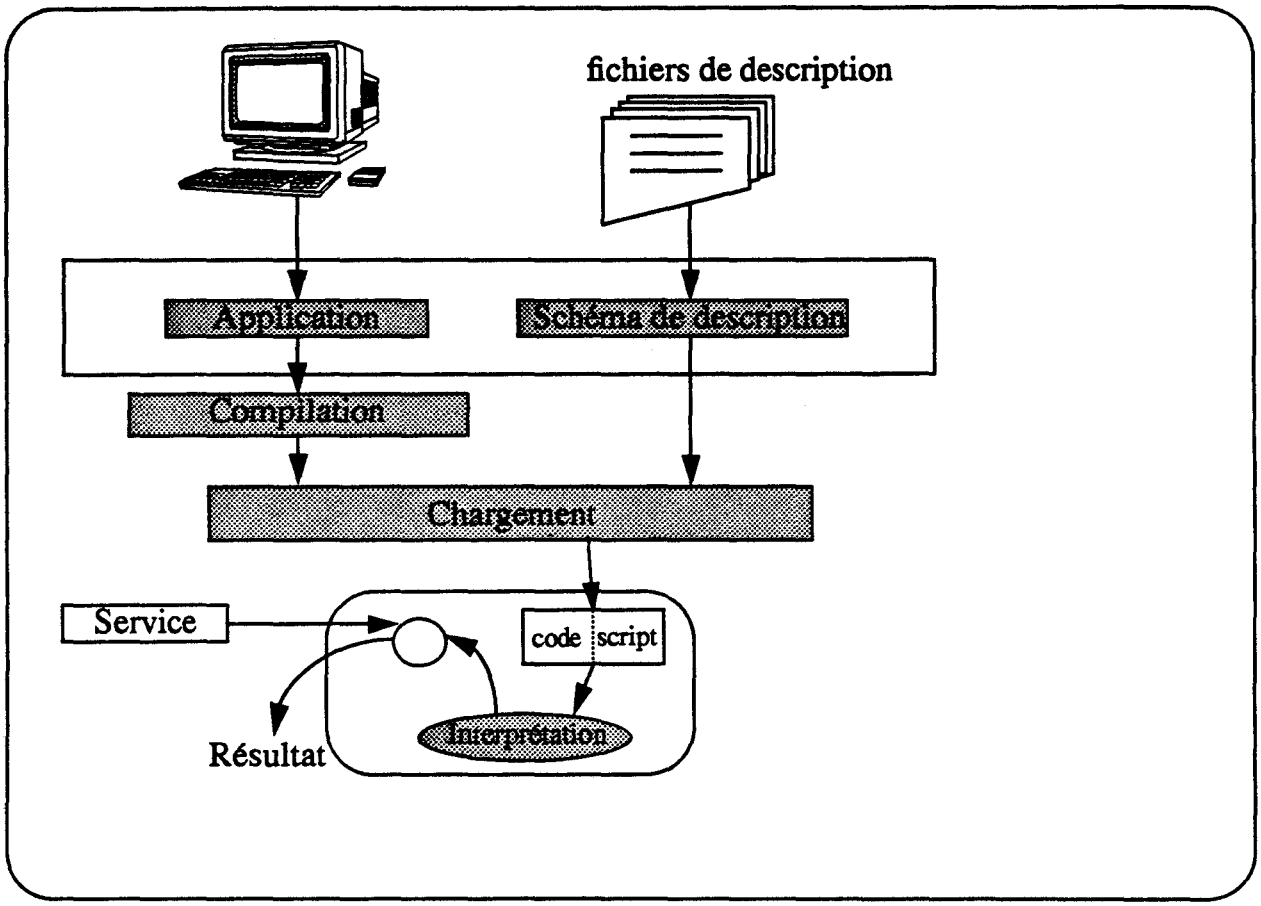
Les conditions d'acceptation d'une commande peuvent reposer aussi sur la mise en oeuvre de deux mot-clés *user* et *caller*.

Le mot-clé *user* indique l'utilisateur qui est vraiment connecté au moment de l'appel. Le mot-clé *caller* se rapporte au propriétaire de l'application qui appelle directement le service concerné. Quand un utilisateur appelle directement un service, sa valeur est "système".

5.4.2 Intégration dans la Carte Blanche

La description de la politique de sécurité d'une application est chargée avec l'application elle-même. Ces fichiers de description ne sont pas chargés directement, mais sont bien entendu compilés et optimisés à la fois pour bénéficier d'un gain de place et d'efficacité. Pendant l'installation de l'application, ces fichiers sont vérifiés afin de respecter des normes de cohérence et d'intégrité. Le schéma S-Shell est alors évalué par un interpréteur interne activé à chaque appel de service. Ceci est représenté sur la figure 25.

FIGURE 26 Description de son intégration dans la Carte Blanche



5.5 Etude d'exemples

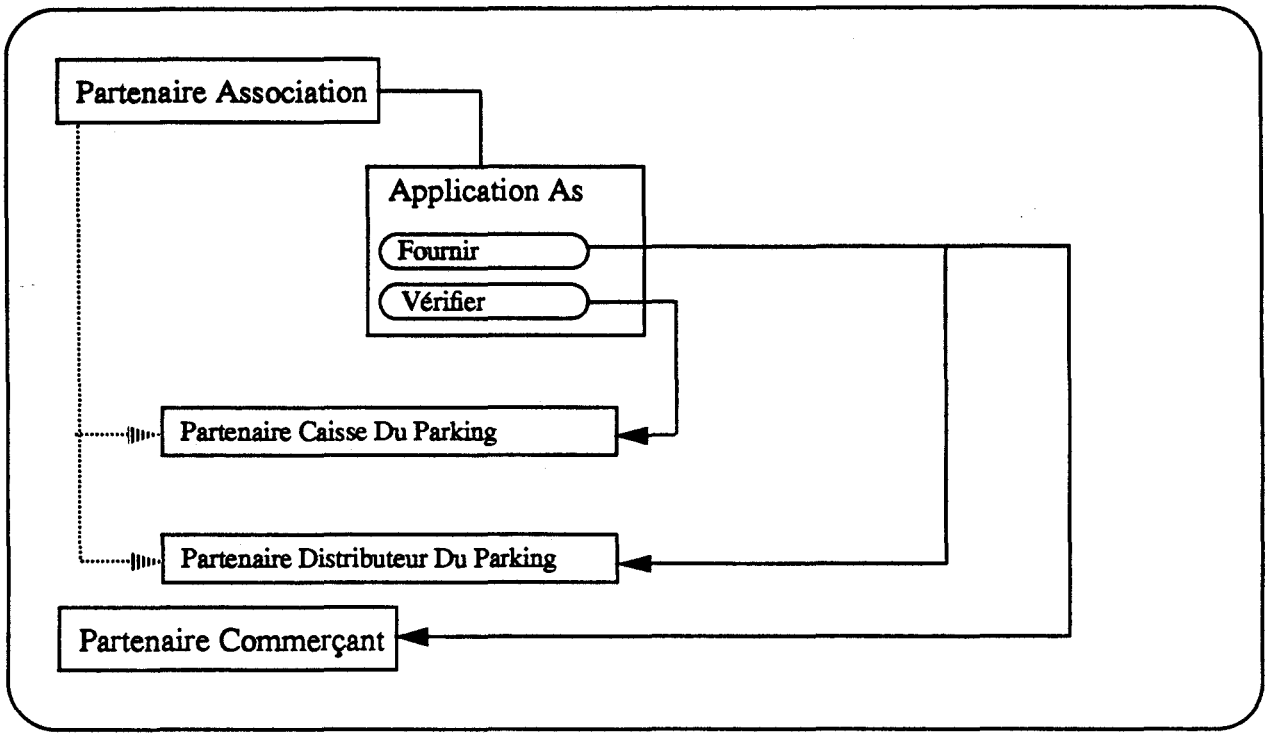
Afin d'illustrer notre propos, nous avons choisi une carte multi-applicative basée sur une application commerciale. Nous avons choisi comme exemple un groupe de commerçants gérant leurs propres applications (cartes de réduction, soldes, ...). Ces commerçants ont décidé de créer une association dont le but est de gérer un parking commun et d'offrir à leurs clients une carte de fidélité globale.

5.5.1 Exemple 1: Gestion d'un parking

L'association vient donc de mettre en place pour leur clientèle un parking afin de faciliter leur accès aux magasins. Le parking n'est pas gratuit, cependant chaque commerçant a la possibilité d'offrir à ses clients des jetons de parking gratuits afin que ceux-ci ne payent pas le stationnement s'ils ont effectué des achats dans la journée. La seule restriction imposée à ce schéma est que chaque particulier ne puisse pas posséder plus d'un jeton gratuit dans sa carte.

Le diagramme représentatif des droits d'utilisation de services entre les intervenants et le suivant:

FIGURE 27 Représentation de l'exemple 1



En effet, l'association a besoin de deux services dans son application pour répondre au cas décrit. Le premier service que nous avons appelé 'fournir' sert à distribuer des jetons. Ce service est offert au partenaire 'distributeur du parking' pour pouvoir acheter ces jetons. Il est de plus offert au partenaire commerçant pour que ce dernier puisse en distribuer gratuitement. Le second service appelé 'vérifier' ne sert à vérifier les conditions de distribution. Il est donc fourni au partenaire 'caisse du parking'. Au niveau de la description de ce schéma sécuritaire pour la Carte Blanche, nous pouvons écrire en S-Shell, les lignes suivantes:

FIGURE 28 Représentation de l'exemple 1

```
Default { free_token = 0 ; } : never ;
Vérifier { free_token = 0 ; } : if (user = "Caisse") ;
Fournir : if (user = "Distributeur") ;
Fournir { free_token = 1 ; } : if ((user = "Commerçant") et (free_token = 0)) .
```

Ainsi à l'initialisation, on précise qu'il n'y a aucun jeton gratuit dans la carte. Ne pouvant posséder plus d'un jeton gratuit, le passage à la caisse du parking fait que ce jeton est directement consommé.

C'est pourquoi, l'action associée au passage à la caisse est une remise à zéro du nombre de jeton gratuit contenu dans la carte. De plus, l'action associée au service 'fournir' étant différente selon le type d'utilisateur, le S-Shell permet une division de ce service en deux de façon à pouvoir traiter ce cas plus facilement. Ainsi, un achat de jeton se faisant au distributeur, le service 'fournir' doit dans cette hypothèse être toujours satisfait. En revanche, si c'est un commerçant qui désire fournir un jeton à son client, le S-Shell vérifie d'abord que ce client ne possède pas déjà un jeton gratuit. Si cette condition est satisfaite, le service 'fournir' est autorisé et la variable nombre de jeton gratuit passe à 1.

5.5.2 Exemple 2: Application bancaire

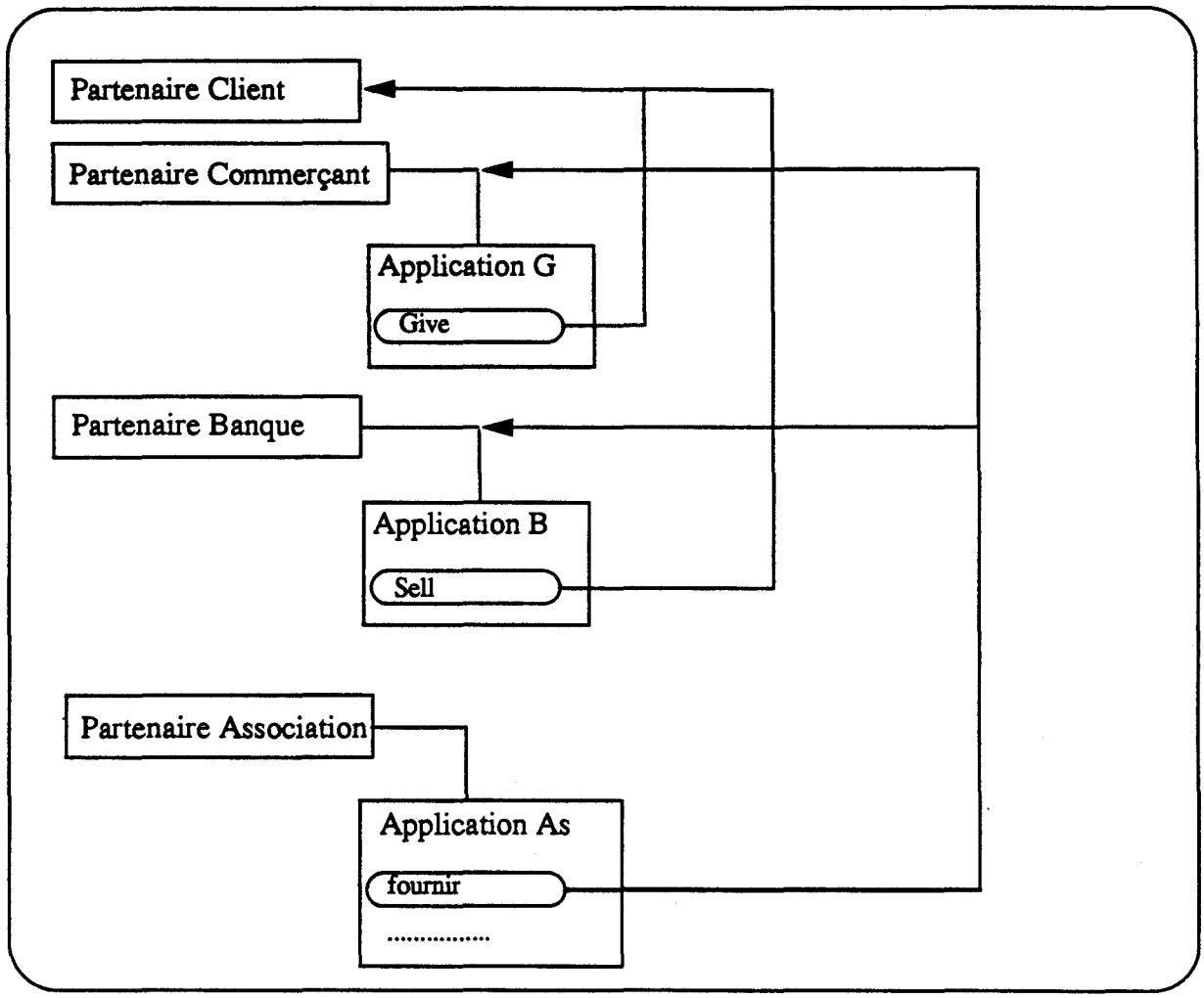
L'association fournit toujours des tickets de parking au commerçant qui possède maintenant sa propre application. En addition, la banque offre la possibilité d'acheter des jetons à son distributeur. On suppose que la banque a déjà acheté sa propre application et que l'application bancaire peut être mise dans la carte.

Le problème que nous voulons mettre en évidence maintenant est celui de la délégation de droits. D'une part la carte veut vendre des jetons par l'intermédiaire de son application. Pour faire cela, l'association doit fournir à la banque le droit de distribuer des jetons. Ces droits doivent être encore valides dans les services de l'application bancaire même si le partenaire connecté est le client.

D'autre part, l'association veut donner au commerçant le droit de distribuer des jetons (quand il est personnellement connecté), elle peut ne pas avoir assez confiance en lui au point de lui donner les mêmes droits.

Le schéma ci-dessous offre une vue synthétique (même si les différences entre les deux cas ne sont pas dessinées) du problème.

FIGURE 29 Représentation de l'exemple 2



La Carte Blanche doit être assez puissante pour permettre une expression des deux problèmes aussi simplement que possible.

En résumé nous voulons autoriser la banque à vendre des jetons de parking dans son application tout en empêchant le magasin de s'autoriser à mettre des jetons dans sa propre application. Les différences entre les deux cas apparaissent en considérant l'utilisateur qui est connecté au moment de l'appel de service.

En ce qui concerne l'application bancaire, nous pouvons supposer que le porteur de la carte est connecté de façon à pouvoir discuter avec l'application bancaire. Le comportement traditionnel du magasin est de se connecter à la Carte Blanche afin d'offrir des jetons au porteur de la carte. Le commerçant ne doit pas donner son code secret au client (sinon le client pourrait gagner des jetons à tout moment).

La description S-Shell de l'application de l'association inclura alors les lignes suivantes:

FIGURE 30 Résolution du second exemple

```
...
fournir : if ((user = "commerçant") or (caller = "banque")) ;
```

Ce schéma de description autorise l'appel du service 'fournir' si le commerçant est personnellement connecté ou si l'application bancaire est en fonctionnement. Un utilisateur qui a le droit d'utiliser 'fournir' (par exemple le commerçant) peut obtenir un jeton grâce au service du commerçant.

5.6 Conclusion

Ce cas d'étude lié à la Carte Blanche montre une évidente utilité du S-Shell. On remarque que non seulement des schémas qui peuvent se révéler très précis peuvent être réalisés par ce prototype, mais qu'ils le sont d'une manière toujours aussi simple. Cette simplicité de description fait la force du S-Shell car elle permet de condenser des droits de fine granularité en un nombre de lignes de description très réduit sans pour autant affaiblir la sécurité du système. Des problèmes réels, tels que la délégation de droits au sein de la Carte Blanche ne serait sans le S-Shell vraiment pas facile à réaliser. En utilisant le S-Shell, on voit qu'il suffit de rajouter, dans le cas de figure décrit ci-dessus, une seule ligne pour y parvenir. Cette réalisation est donc plus qu'encourageante et nous a conforté dans l'idée que nous avions de créer une réelle simulation de carte à microprocesseur, en adaptant le S-Shell au besoin de réalisation plus concrète. C'est l'objet du chapitre suivant intitulé "Une implantation distribuée du S-Shell: une architecture de sécurité pour la carte".

Chapitre 6

Une implémentation du S-Shell

6.1 Introduction

Le but de ce chapitre est de réaliser une maquette de carte qui permette d'expérimenter l'idée du S-Shell. Nous avons préalablement étudié les mécanismes mis en oeuvre par le S-Shell et proposé quelques exemples d'application de problèmes de schémas de sécurité concrets que l'on pouvait obtenir grâce aux fonctionnalités de notre outil (chapitre 3).

Nous voulons maintenant nous attacher au problème de l'implémentation de ce S-Shell sur un prototype de carte à microprocesseur. Nous nous attacherons tout au long de ce chapitre à élucider les points sensibles d'un tel travail. Nous mettrons notamment en évidence les restrictions à apporter au S-Shell afin de pouvoir réellement atteindre ce but.

6.2 Le système d'exploitation retenu

Dans ce chapitre nous voulons présenter le système d'exploitation que nous avons choisi et énumérer les différentes commandes de base que nous avons retenues et que nous comptons implanter dans le simulateur de carte. Par système de base, nous entendons le système sans sécurité.

6.2.1 Justification des choix

Nous avons étudié la carte MCOS [Ge90] de Gemplus, la carte TB100 [Ph90] de Philips et enfin la norme ISO/IEC 7816-4 [ISO4]. De cette étude nous pouvons faire les remarques suivantes:

- La norme ISO/IEC 7816-4 est largement postérieure aux cartes MCOS et TB100 et nous apparaît comme une synthèse et une généralisation des deux cartes précédemment citées au niveau de l'organisation des donnés et des clés

- La norme ISO/IEC 7816-4 se contente de normaliser l'accès aux fichiers et la présentation des clés, mais sans jamais rien imposer quant à la création de répertoires, de fichiers et de clés. En définitive la norme laisse entière liberté pour implémenter le schéma de sécurité
- A contrario, les cartes TB100 et MCOS offrent un schéma de sécurité parfaitement défini et sont plutôt à considérer comme des exemples de ce l'on pourrait obtenir à l'aide d'une carte sur laquelle on aurait implémenté S-Shell

Ainsi donc, de ces trois remarques nous pouvons conclure que nous allons essayer surtout de nous approcher le plus possible de la norme ISO/IEC 7816-4 pour implémenter les fonctionnalités de base de notre carte. Cependant le but n'étant pas de réaliser une implémentation de la norme sur une carte mais de développer un prototype de recherche, de nombreuses fonctionnalités présentes dans la norme ne seront pas considérées, comme les considérations liées à la cryptographie par exemple.

6.2.2 Organisation des données

1. L'organisation logique des fichiers est la structure arborescente classique composée de fichiers répertoires appelés Dedicated File (DF) dans la norme et de fichiers élémentaires Elementary File (EF). Le répertoire racine ou Master File (MF) est obligatoire, tous les autres DF étant optionnels. Il existe deux types de fichiers élémentaires, à savoir les fichiers élémentaires internes (dans lesquels se trouvent les données analysées et utilisées par la carte), et de travail (dans lesquels se trouvent les données qui seront utilisées par le monde extérieur). La figure 31 suivante montre l'organisation en fichiers dans la carte
2. Les méthodes de référence aux fichiers sont de plusieurs types, cf [ISO-4]
3. A tout moment, il existe un répertoire actif et un fichier actif. Une commande permet de désigner ou de sélectionner le fichier actif. Toutes les commandes de lecture ou d'écriture se rapportent au fichier actif. La norme souligne à ce propos qu'une sélection réussie d'un fichier positionne ce fichier courant à un canal logique. Ces canaux sont au nombre de quatre [ISO-4]

6.2.3 Organisation de la sécurité

Les mécanismes de sécurité envisagés par la norme sont de plusieurs ordres. Nous avons l'authentification d'entité par mot de passe, l'authentification d'entité par clé, l'authentification de données et le cryptage de données. Par souci de simplification, nous ne traiterons dans notre prototype que l'authentification d'entité par mot de passe. Ainsi pour pouvoir accéder aux données de la carte, l'utilisateur devra fournir des mots de passe.

Ces derniers pourront être attachés aux répertoires. Ils seront numérotés de 0 à N pour chaque répertoire. La norme prévoit de pouvoir présenter un mot de passe global (le mot de passe est alors comparé aux mots de passe attachés au répertoire racine) ou local (le mot de passe est alors comparé aux mots de passe attachés au répertoire courant). Dans notre prototype nous admettrons que, pour cette commande, si le répertoire courant ne contient aucun mot de passe, la recherche se poursuivra dans le répertoire parent et ainsi de suite.

La présentation réussie de mots de passe aura pour effet de modifier le *statut de sécurité* de l'utilisateur. Pour autoriser une opération la carte compare le statut de sécurité courant (les droits acquis) avec les "attributs de sécurité" associés à chaque fichier et qui en contrôlent l'accès. On distingue "le statut de sécurité global" et "le statut de sécurité local" qui est lié au répertoire courant et pourra être 'oublié' par la carte si un répertoire qui n'est pas un descendant du répertoire courant devient actif.

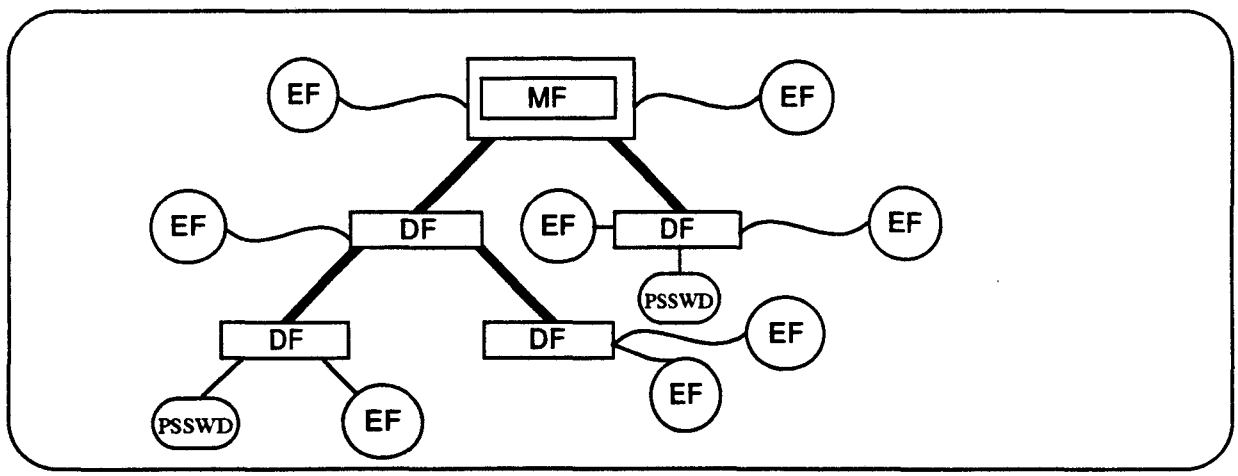
6.2.4 Définition d'une application

Nous n'avons pas clairement défini ce que nous entendions par "application". Nous admettons qu'il s'agit d'un sous-ensemble de fichiers, dont l'accès est régi par un sous-ensemble de mots de passe, qui est globalement la propriété d'un organisme "émetteur d'application", et qui concourt à assurer le bon fonctionnement d'une application du monde réel.

C'est donc par abus que l'on appelle application ce qui n'est en fait que la structuration de données utilisée dans une carte pour les besoins d'une vraie application.

Au vu des paragraphes précédents une application est matérialisée sur la carte par un répertoire et tout ce qu'il contient. On notera qu'une application peut contenir des sous-applications et organiser une forme de coopération entre celles-ci.

FIGURE 31 Organisation des données dans la carte



6.2.5 Références des commandes de base

Ce paragraphe vise à énumérer les commandes de base qui seront utilisées dans notre prototype tout en indiquant quels seront les effets de ces commandes sur les données de la carte.

Avant de débuter cette description il faut savoir que la carte tient les informations suivantes à jour:

- Un pointeur sur le fichier élémentaire courant (initialement non défini)

- Un pointeur sur le fichier répertoire courant (initialement le répertoire racine)
- Les statuts de sécurité local et global courant

Les commandes que nous allons maintenant décrire sont les suivantes:

- SELECT <path>
  <path> est une chaîne d'identificateurs de fichiers commençant par l'identifiant du répertoire racine (référencé selon la norme par 3F00) ou par un fichier/répertoire se trouvant dans le répertoire courant et finissant par l'identifiant du fichier lui-même.
  Si un fichier élémentaire est sélectionné, il devient le fichier élémentaire courant et son répertoire parent devient le fichier répertoire courant.
  Si un fichier répertoire est sélectionné, il devient le fichier répertoire courant et le fichier élémentaire courant n'est alors pas défini.
  Le statut de sécurité local peut être affecté par cette commande.
- MK_FILE <FileIdent> <ByteSize>
  Cette commande crée un fichier élémentaire dans le répertoire courant.
  <FileIdent> est un numéro de deux octets qui ne doit pas déjà avoir été dans le répertoire courant. La taille du fichier est fixée une fois pour toutes lors de la création. Le fichier est initialement dans un état "effacé". En cas de succès, le fichier créé devient le fichier élémentaire courant.
- MK_DIR <FileIdent>
  Cette commande crée un fichier répertoire dans le répertoire courant.
  <FileIdent> est un numéro de deux octets qui ne doit pas déjà avoir été dans le répertoire courant.
- MK_PWD <NumPasswd> <Passwd>
  Cette commande crée un nouveau mot de passe et l'associe au répertoire courant. Il ne doit pas déjà exister un mot de passe sous le numéro <NumPasswd>
- CHG_PWD <NumPasswd> <OldPasswd> <NewPasswd>
  Cette commande permet de changer un mot de passe dans le répertoire courant. Pour cela l'ancien mot de passe doit être fourni.
- CHK_PWD <Local/Global> <NumPasswd> <Passwd>
  La variante "global" recherche le mot de passe dans le répertoire racine de la carte. La variante "local" recherche le mot de passe dans le répertoire courant ou éventuellement un répertoire parent si le répertoire courant est dépourvu de mot de passe.
  <Passwd> est comparé au mot de passe associé au répertoire sous le numéro <NumPasswd>. En cas de succès, le statut de sécurité est mis à jour. En cas d'échecs successifs répétés, le mot de passe peut être bloqué. Le S-Shell gère les conditions liées à ce blocage.

6.3 Un S-Shell non distribué

6.3.1 Une synopsis possible du S-Shell dans le cadre du prototype

Dans notre étude le S-Shell a pour mission de:

- Accepter ou refuser une commande provenant de l'utilisateur
- Exécuter ou non la commande en fonction de la commande et de ses paramètres, de l'état courant du S-Shell et de quelques données propres au système d'exploitation (date, heure, ...)
- En cas de refus de la commande, fournir une justification auprès de l'utilisateur par un code d'erreur
- En cas d'acceptation de la commande, la faire exécuter par le système d'exploitation
- En fonction du résultat (commande refusée ou acceptée, avec dans ce dernier cas prise en compte du code retour renvoyé par le système d'exploitation), mettre à jour par le S-Shell de son état courant

Ce squelette du S-Shell peut se résumer à un algorithme général suivant:

```
main ()
{
Initialisation des variables d'état du S-Shell;
Répéter indéfiniment

  {
  Saisie de la commande ;
  Analyse syntaxique de la commande
  Selon la commande :

    {
    Cas READ_FILE <DataByteOffset> <DataByteLength>:

    Déterminer si la commande est autorisée
    Si oui la faire exécuter par le système d'exploitation
    Mettre à jour les variables d'état du S-Shell

    Cas WRITE_FILE <DataByteOffset> <DataByteLength> <Data>:

    .....

    Cas UPDATE_FILE <DataByteOffset> <DataByteLength> <Data>:

    .....
```

```
        Cas ERASE_FILE <DataByteOffset> <DataByteLength> <Data>:
        .....
        .....
        }
      Envoi du résultat de la commande à l'utilisateur
      }
    }
```

On remarque que l'adaptation du S-Shell au milieu de la carte maintient son efficacité au détriment de l'aisance de description. Un pseudo-C doit alors être envisagé.

6.3.2 Implémentation du S-Shell en programmation orientée objet

Nous allons présenter dans ce paragraphe une façon différente d'implémenter le S-Shell, dans laquelle ce dernier est un objet; Cette implémentation permet donc de bénéficier de tous les avantages de la programmation objet (abstraction, encapsulation, ...) et apporte en outre les avantages suivants:

- Le S-Shell est clairement séparé du reste, c'est-à-dire de la boucle de saisie de messages et du système d'exploitation
- Nous avons cassé une fonction très grande en nombreuses petites fonctions, puisque chaque commande filtrée par le S-Shell fera l'objet d'une ou de deux fonctions membres
- Cette décomposition ouvre la voie à la distribution du S-Shell, en effet on peut imaginer facilement plusieurs de ces objets, spécialisés chacun sur un sous-ensemble des données et/ou des commandes

L'objet S-Shell est désormais appelé deux fois. La première fois il devra décider d'autoriser ou de rejeter la commande. La seconde fois il mettra son état à jour. Les amateurs de programmation pourront se référer à l'annexe C afin de découvrir les algorithmes correspondant à ce chapitre.

6.4 Principe d'un S-Shell distribué

Nous avons mis en évidence dans le chapitre précédent que le S-Shell pouvait être implémenté sous forme d'objet contrôleur, et que pour chaque commande reçue par la carte on pouvait envoyer un message à cet objet contrôleur.

Il est dès lors naturel d'envisager le S-Shell distribué comme un ensemble de contrôleur.

6.4.1 Attacher un contrôleur aux données

- *Un contrôleur pour chaque répertoire*: Pour mettre en oeuvre un schéma de sécurité spécifique à chaque application, il suffit de prévoir une architecture bi-contrôleur. Le contrôleur global carte, qui gère les droits sur les données globales à plusieurs applications, coopérant avec un contrôleur local sélectionné en fonction de l'application courante. Ceci nécessite de pouvoir attacher un contrôleur à chaque application. Ce contrôleur n'est alors activé qu'au moment où l'application est sélectionné.
  De cette manière en considérant, comme il l'a été clairement souligné précédemment (paragraphe 2.4), qu'un répertoire est une application en puissance, c'est donc naturellement qu'il convient d'attacher ces contrôleurs aux répertoires. Le contrôleur aura alors la tâche de:

- superviser les opérations qui se rapportent à ce répertoire et à ses sous-répertoires (création de fichiers, création de mots de passe, accès aux données, ...)

- mémoriser dans son état interne la liste des mots de passe locaux présentés par l'utilisateur (ou ensemble des droits acquis par ce moyen)

- *Un contrôleur pour chaque fichier élémentaire:* En généralisant ce qui a été dit, il paraît souhaitable de pouvoir associer des contrôleurs non seulement aux répertoires, mais également aux fichiers.
  Le fait de pouvoir associer un contrôleur à un fichier permet en effet de varier le schéma de sécurité selon les fichiers à l'intérieur d'une même application, mais surtout c'est un moyen d'attacher des données de contrôle à un fichier, car les données encapsulées dans l'objet contrôleur sont un excellent moyen de coder les "attributs de sécurité" d'un fichier (droits requis pour lire, écrire, verrouillage éventuel du fichier, etc ...).
- *Un contrôleur pour chaque mot de passe:* En suivant la démarche précédente, on attache de la même manière un contrôleur à chaque mot de passe. Ce contrôleur a pour vocation de:

- mémoriser dans son état les droits associés à ce mot de passe

- gérer le mécanisme de la ratification (blocage du mot de passe après plusieurs tentatives infructueuses)

6.4.2 Attacher plusieurs contrôleurs par donnée

Pour justifier ce choix, nous avançons deux raisons d'ordre différent.

La première est de dire que les mécanismes de sécurité peuvent être dissociés dans la carte. Par exemple au niveau d'un fichier on pourra trouver un premier contrôleur qui interdira l'accès si un certain mot de passe n'a pas été présenté, et un second contrôleur qui permettra un verrouillage en lecture ou en écriture du fichier. On notera que tous les fichiers n'ont pas obligatoirement besoin de ces deux mécanismes et par conséquent il est préférable d'attacher à chaque contrôleur un rôle unique. De plus de par le choix d'une programmation en langage orienté objet, cela facilite grandement l'aisance de description des classes.

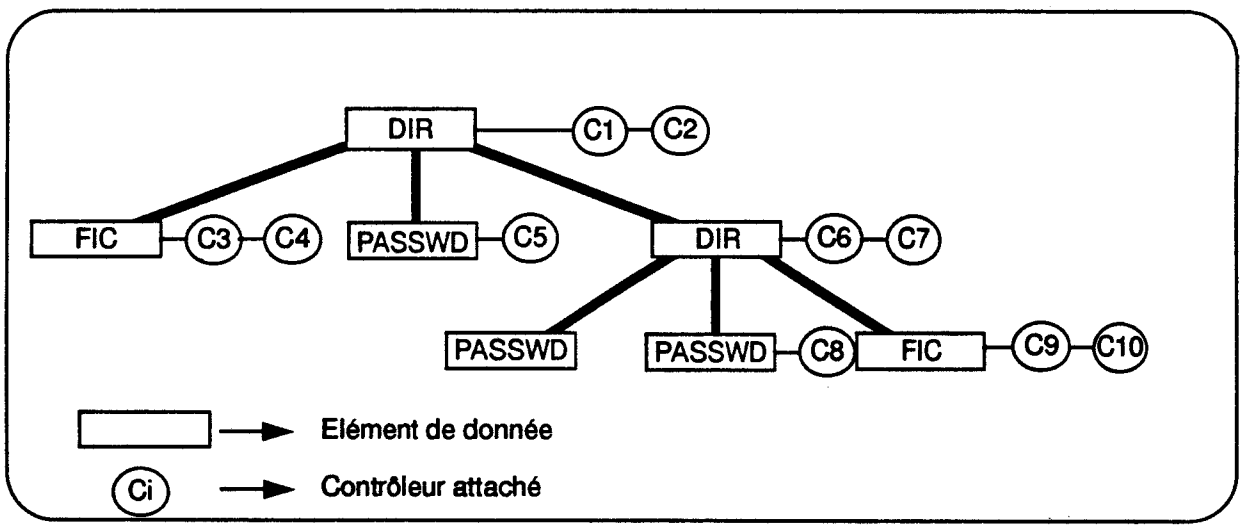
La seconde raison est de souligner que les contrôleurs peuvent provenir d'origines différentes. Par exemple, *l'émetteur de la carte* crée un répertoire application et lui associe un contrôleur qui limite les droits globaux des utilisateurs de cette application. Par la suite *l'émetteur de l'application* met en place sa propre architecture de sécurité et pour cela peut lier de nouveaux contrôleurs à ce même répertoire. Ceci permet donc de rendre indépendant les besoins de sécurité.

En adoptant cette théorie, on doit donc introduire la notion de liste de contrôleur. A chaque élément de donnée carte (répertoire, fichier, etc ...) sera donc associé une liste de contrôleur.

La commande permettant d'ajouter un nouveau contrôleur sera:

- BIND <ClasseContrôleur> <Données d'initialisation>
  La classe de contrôleur indiquée est instanciée, l'instance est initialisée à l'aide des données fournies puis ajoutée au bout de liste sur l'élément donné comme l'indique la figure 32.

FIGURE 32 Liste des contrôleurs attachée à un élément de donnée



6.4.3 Arbitrage

Après avoir établi cette liste des contrôleurs, il reste à mettre en place un arbitrage entre tous les contrôleurs. Un contrôleur se contentant de donner son avis si on le lui demande, la tâche d'arbitre va donc consister à:

- déterminer en fonction de la commande l'ensemble des contrôleurs qui peuvent donner leur avis
- arbitrer entre les contrôleurs qui se prononcent pour ou contre la commande

6.4.4 Sélection des contrôleurs

De par les choix effectués précédemment, et notamment à cause de l'importance des liens de parenté, le principe retenu est celui du chemin. Si une commande concerne une donnée (fichier, mot

de passe), les contrôleurs associés à cette donnée et à tous les fichiers répertoires qui se trouvent sur son chemin depuis le répertoire racine seront associés à la commande.

Le fait qu'un contrôleur de répertoire se prononce sur une opération de lecture de fichier élémentaire situé à un niveau plus bas dans la hiérarchie est le bienvenu. En effet cela permet:

- le verrouillage de répertoire ou d'application
- une gestion simplifiée en cas de protection identique pour tous les fichiers d'un répertoire
- la gestion séparée des protections inter et intra-application

6.4.5 Les réponses possibles des contrôleurs

En réponse à une commande, un contrôleur interrogé dispose de trois types de réponse.

- oui (ACCEPT)
- non (VETO)
- ne se prononce pas (DMIND)

Les deux premiers cas sont triviaux. Le dernier peut survenir pour les raisons suivantes:

- il n'est pas prévu dans sa classe de méthode pour répondre à la commande en question
- la méthode renvoie explicitement la valeur 'ne se prononce pas'

L'avis d'un contrôleur qui ne se prononce pas n'est pas pris en compte dans l'arbitrage. Dans le cas où aucun contrôleur ne se prononce, l'arbitre refuse la commande. Dans le cas où plusieurs contrôleurs répondent par des 'oui' et d'autres par des 'non', la priorité va au 'non'. Cette suprématie du 'non' sur le 'oui' permet par exemple à un contrôleur qui s'occupe du verrouillage au niveau d'un fichier ou répertoire d'empêcher la lecture d'un fichier verrouillé, même si un autre contrôleur autorise cette lecture en fonction des droits acquis. L'annexe D propose un algorithme simplifié de l'arbitrage.

6.5 Discussion sur les contrôleurs

Dans ce paragraphe, nous allons examiner en détail l'anatomie des contrôleurs et de leur cycle de vie; leur intégration au système d'exploitation carte fera l'objet du chapitre suivant.

6.5.1 Classe de contrôleur

Le modèle retenu pour l'implémentation des contrôleurs est le modèle objet avec les notions d'abstraction et d'encapsulation, mais sans la notion de hiérarchie.

Comme dans le modèle à objets classiques, la classe de contrôleur est une abstraction, qui formalise la structure et le comportement communs à plusieurs objets contrôleurs attachés à divers éléments de données de la carte.

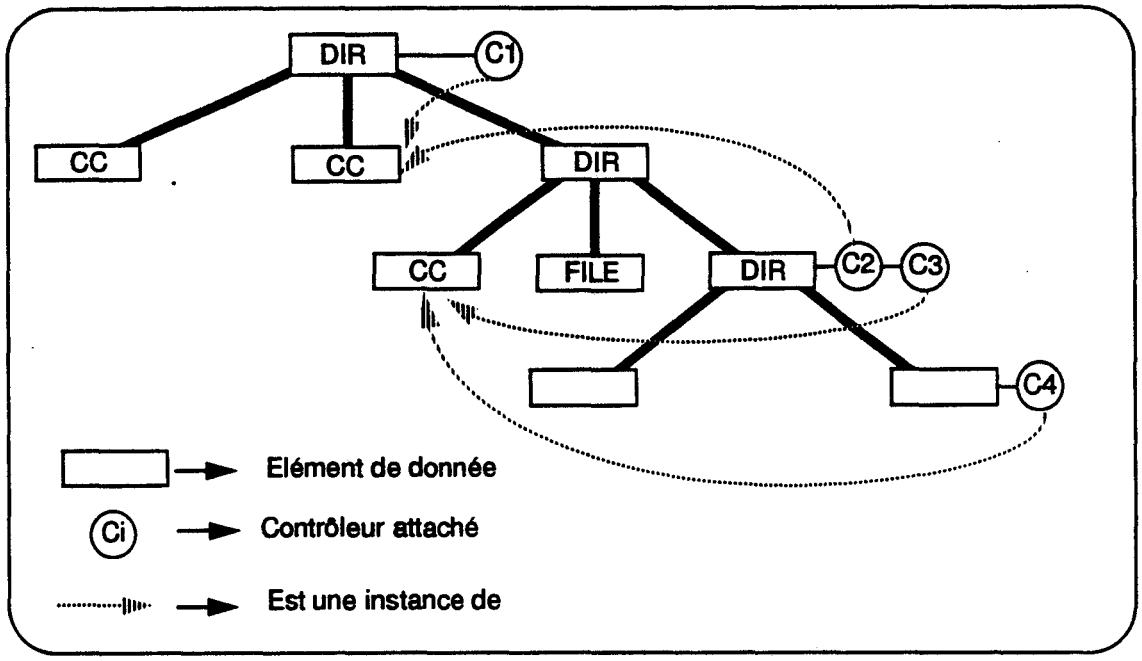
Cette structure et ce comportement sont représentés par un ensemble de variables et de méthodes. Les variables permettent de coder l'état d'un objet, tandis que les méthodes sont utilisées par les objets pour répondre aux messages qui leur sont envoyés.

6.5.2 Une classe de contrôleur est un élément de donnée

Une instance de contrôleur ne contient qu'une indication de classe, ainsi que les données qui représentent son état. Par contre une classe de contrôleur contient la description des variables servant à coder l'état d'un objet, ainsi que les méthodes utilisées par toutes les instances de la classe.

Pour des raisons de sécurité évidentes, le code de ces méthodes doit résider dans la carte, et nous proposons de stocker dans la carte les classes de contrôleur sous la forme de fichiers élémentaires d'un type particulier. Il faut donc considérer les classes de contrôleur comme des éléments de donnée de la carte, au même titre que les répertoires, les fichiers ou les mots de passe. Par conséquent, le mécanisme de protection par contrôleur s'applique aux classes de contrôleur, comme on peut le voir sur la figure suivante:

FIGURE 33 Mécanisme de protection par objets contrôleur



6.5.3 Pas d'héritage

La notion d'héritage présente dans la plupart des langages de programmation orientés objets, et qui permet de dériver des classes plus spécialisées en héritant des caractéristiques de classes plus générales, n'est pas prévue dans le premier prototype. En effet:

- La gestion de l'héritage est relativement complexe à mettre en oeuvre, alors que les contrôleurs carte ne seront a priori pas très complexes; il n'est dès lors pas évident que les gains réalisés dans l'expression des classes de contrôleur compenseront l'accroissement de la complexité de la carte
- Si l'on avait imposé un seul contrôleur par élément de donnée, l'héritage multiple aurait été nécessaire, avec par exemple le mixin 'Objet_Verrouillable'. On peut considérer que le fait de permettre plusieurs contrôles par élément de donnée, allié au mécanisme d'arbitrage entre contrôleurs, constitue un cas primitif et particulier d'héritage, au reste adapté à la sémantique d'un contrôleur

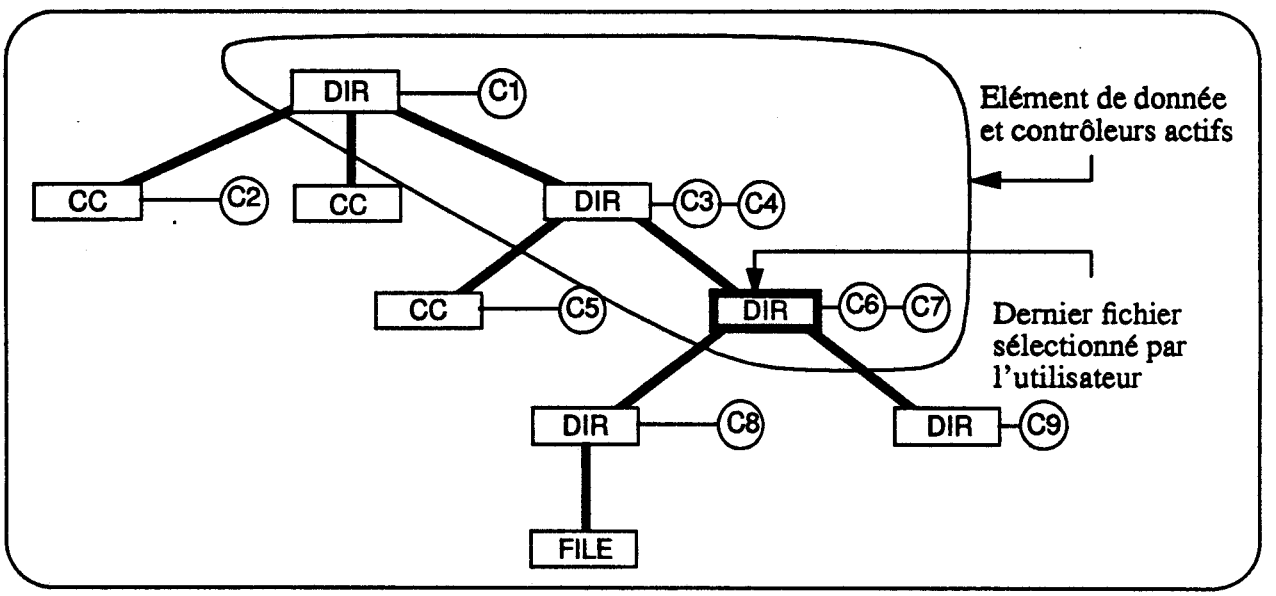
6.5.4 Contrôleur actif/inactif

Nous allons introduire la notion de contrôleur actif et de deux notions qui y sont rattachées, la restriction des envois de messages aux seuls contrôleurs actifs et l'état volatile d'un contrôleur.

- *Définition d'un contrôleur actif*: Nous avons vu au premier paragraphe la définition de contrôleur actif et de fichier élémentaire actif. Nous avons aussi étudié (paragraphe 4.4 de ce chapitre) l'arbitrage entre les contrôleurs situés sur le chemin menant de la racine jusqu'à l'élément

de donnée concerné par la commande. Nous dirons qu'un contrôleur est actif si et seulement si il est attaché à un élément de donnée actif. Un élément de donnée est actif si et seulement si il se trouve sur le chemin qui mène du répertoire racine à l'élément de donnée (fichier répertoire, fichier élémentaire) actif qui se trouve le plus bas dans la hiérarchie. Les mots de passe et les classes de contrôleur doivent être considérés comme des fichiers élémentaires d'un type particulier, comme le montre la figure ci-dessous:

FIGURE 34 Mécanisme de protection par droits



6.5.5 Etat volatile d'un contrôleur

Une partie de l'état d'un contrôleur réside en EEPROM, tandis que l'autre réside en RAM. La partie résidant en RAM est appelée l'état volatile du contrôleur: Cette partie est remise à zéro non seulement lors de la coupure d'alimentation de la carte, mais aussi le contrôleur devient inactif.

On déclare au niveau de la classe de contrôleur les variables qui seront volatiles:

```
class Toto {
        ram int champ1 ;
        eeprom int champ2 ;
        ...
        };
```

L'état non volatile est une nécessité, car en son absence les attributs de protection des fichiers élémentaires ne pourraient survivre à une coupure d'alimentation;

L'état volatile est également très pratique pour stocker les droits acquis au niveau du répertoire:

- les droits acquis sont susceptibles de changer lors de chaque présentation de mot de passe et il serait dommage de faire une écriture EEPROM à chaque fois

- la remise à zéro des droits acquis est automatique lors du Reset

- la norme 7816-4 précise que les droits locaux acquis dans un répertoire seront perdus lorsqu'on sort du répertoire[1]; d'ailleurs le mécanisme des canaux multiples est une réponse à cette limitation

6.5.6 Ordre d'activation et de désactivation des contrôleurs

Lorsque l'utilisateur sélectionne un nouvel élément de donnée, le système d'exploitation commence par désactiver tous les contrôleurs situés sur la branche "morte" jusqu'à la jonction vers le nouvel élément actif, puis active tous les contrôleurs qui deviennent actifs.

Un contrôleur est toujours activé une fois que les contrôleurs attachés à des éléments de données situés plus haut dans la hiérarchie aient été activés; les désactivations ont lieu dans l'ordre inverse, c'est-à-dire se propagent du bas de la hiérarchie vers le haut.

Seul un contrôleur actif peut recevoir et émettre des messages, puisque ses données membres volatiles n'existent pas lorsque le contrôleur est inactif.

6.5.7 Envoi de messages aux contrôleurs

Nous avons vu que le noyau du S-Shell envoyait des messages aux contrôleurs, en particulier pour savoir ci ceux-ci autorisent une commande utilisateur. Nous allons introduire dans cette section d'autres messages envoyés par le noyau, ainsi que deux autres expéditeurs de messages.

- *Messages du noyau à un contrôleur*: Pour chaque commande du système d'exploitation, les contrôleurs qui sont sur le chemin de l'élément de donnée donnent dans un premier temps leur avis sur la commande, puis dans un deuxième temps ont une chance de modifier leur état. Tout ceci se fait au moyen d'une seule méthode par commande du système d'exploitation, de prototype général:

```
enum { Oui, Non, PasDeRéponse } Verdict ;
Verdict <Contrôleur>:: Before <OS-Command>(<Paramètres de la commande>)
{
Déterminer si la commande est autorisée
Si non autorisée CodeRésultat <-- ???
return Yes or No or ...
}
```

1. Cette propriété est une caractéristique de la norme actuellement implémentée et non au S-Shell lui-même

```
void <Contrôleur>::After <OS-Command>(<Paramètres de la commande>)
{
Mettre à jour les variables d'état
}
```

Ces deux méthodes sont facultatives, une classe de contrôleur peut définir zéro, une seule ou les deux pour chaque commande du système d'exploitation.

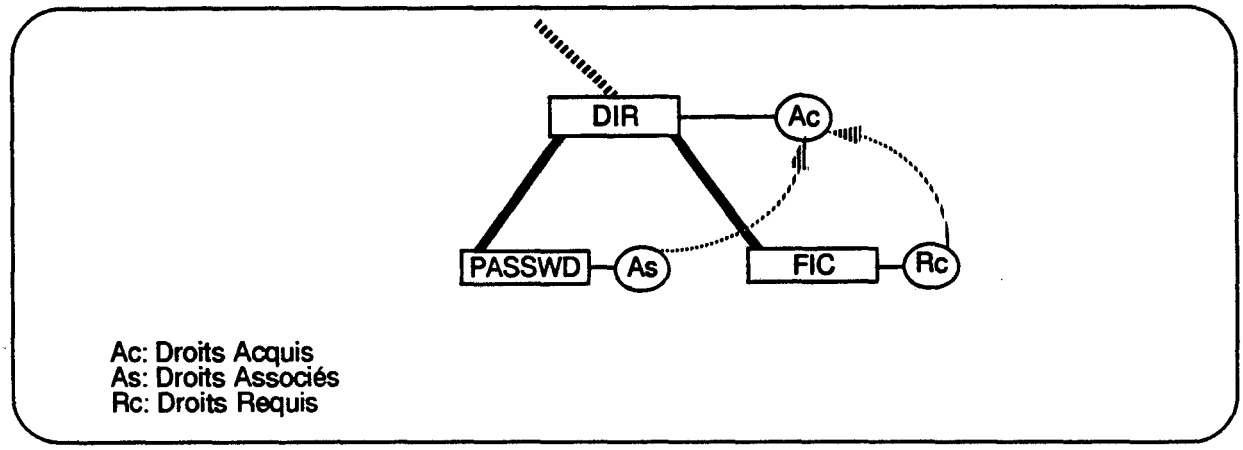
- *Messages 'OnBind' et 'On(De)Activate' du noyau à un contrôleur*: Nous considérons pour ces envois de messages le prototype suivant:

```
void <Class-Contrôleur> :: OnBind (<paramètres selon Class-Contrôleur>) ;
void <Class-Contrôleur> :: OnActivate();
void <Class-Contrôleur> :: OnDeactivate();
```

Le message 'OnBind' est l'équivalent du constructeur d'objet dans le langage C++. Il est envoyé par le système d'exploitation au contrôleur qui vient juste d'être créé par la commande BIND. Les arguments de la commande BIND sont liés aux paramètres de la méthode 'OnBind', dont la tâche consiste donc essentiellement à initialiser les variables EEPROM du contrôleur. Le message 'OnBind' est le tout premier message envoyé à un contrôleur. Avant d'être appelé, tous les champs du contrôleur sont initialisés à zéro.
Le message 'OnActivate' est envoyé à un contrôleur chaque fois que celui-ci est activé. C'est une occasion pour initialiser les données membres RAM. A l'inverse le message 'OnDeactivate' est envoyé juste avant que le contrôleur ne soit désactivé. Il faut tout de même être conscient qu'en cas de Reset ou d'interruption de l'alimentation électrique de la carte, les contrôleurs seront tous désactivés de fait sans que 'OnDeactivate' ait pu être appelé.

- *Messages de contrôleur à contrôleur*: Considérons un répertoire contenant plusieurs mots de passe et plusieurs fichiers élémentaires. Si l'on souhaite implémenter une protection par 'Droit', dans laquelle l'accès à chaque fichier nécessite certains droits et où à chaque mot de passe est associé une liste de droits, il est facile de se rendre compte que la solution la plus simple consiste à maintenir, au niveau d'un contrôleur du répertoire, la liste des droits acquis.

FIGURE 35 Mise en place d'une protection par droits acquis localement

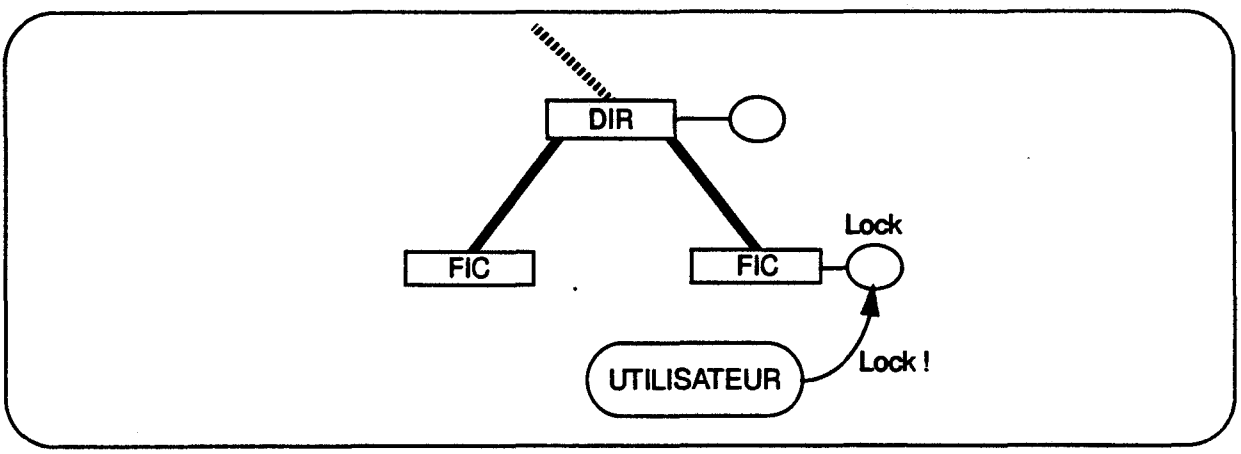


Ainsi lors de la présentation d'un mot de passe, un des contrôleurs du mot de passe va envoyer un message au contrôleur du répertoire qui aura ainsi connaissance des éventuels nouveaux droits acquis. Lorsqu'un ordre de lecture d'un fichier sera reçu, le contrôleur du fichier interrogera le contrôleur du répertoire pour savoir si le droit correspondant a été acquis.

L'interaction entre contrôleurs ne se limite pas à l'envoi du message, mais permet aussi de consulter et de modifier directement les champs de donnée d'un autre contrôleur.

- *Messages de l'utilisateur à un contrôleur*: Considérons le contrôleur qui permet de verrouiller un fichier, c'est-à-dire d'empêcher tout ordre d'écriture lorsqu'il est dans un certain état. Il faut que l'utilisateur puisse envoyer une commande pour verrouiller le fichier, or le système d'exploitation sous-jacent ne connaissant pas la notion de verrouillage, la commande "LOCK_FILE" n'existe pas.
  La solution la plus simple à ce problème consiste à permettre à l'utilisateur d'envoyer directement un message à un contrôleur, au moyen d'une commande "SEND". Par exemple:
  SEND <Contrôleur> <Msg> <ParamLength> <Params>
  En réponse à cette commande, le système d'exploitation enverra le message correspondant au contrôleur visé qui répondra grâce à la méthode "On<Msg>(Param1, Param2, ...)".

FIGURE 36 Envoi de message à un contrôleur par un utilisateur



Naturellement, comme toute commande du système d'exploitation la commande SEND passe au crible du S-Shell et les contrôleurs peuvent filtrer cette commande au moyen de méthodes 'BeforeSend' et 'AfterSend':

"Verdict BeforeSend(datelt ClassContrôleur, int Msg, int P1, int P2)"
"void AfterSend(datelt ClassContrôleur, int Msg, int P1, int P2)"

6.6 Restrictions d'adressage

Nous venons de voir qu'un contrôleur ou un utilisateur peuvent envoyer un message ou accéder à un champ d'un autre contrôleur: Dans ce chapitre il nous faudra préciser la manière de nommer les autres contrôleurs et résoudre deux problèmes potentiels, l'envoi de message à un contrôleur inactif et la possibilité de restreindre les appels de méthodes aux seuls contrôleurs amis (voir le deuxième paragraphe de ce chapitre).

6.6.1 Adressage des contrôleurs

Le mode d'adressage des contrôleurs fait que ceux-ci ne peuvent envoyer de messages qu'aux contrôleurs situés plus haut dans la hiérarchie.

Ainsi les seuls destinataires possibles d'un message envoyé par un contrôleur actif sont forcément actifs et donc à même de traiter des messages.

Cette restriction garantie un cloisonnement horizontal de l'arborescence de fichiers et permet donc de mieux assurer la sécurité. Dans une méthode de contrôleur, la référence à un autre contrôleur destinataire de message se fait au moyen de deux champs:

- Le premier champ désigne un élément de donnée de la carte, de manière relative et non absolue. Pour une commande SEND, il s'agit du dernier fichier sélectionné. Pour un envoi de mes-

sage depuis un contrôleur, il s'agit d'une expression bâtie à l'aide du mot-clé **'this'**.
this : désigne l'élément de donnée auquel est rattaché le contrôleur qui envoie le message
Super(this) : le répertoire contenant **this**
App(this) : l'application contenant **this**
etc ...

- Le second champ indique une classe de contrôleur. La seule mention de la classe suffit pour désigner un éventuel contrôleur sans ambiguïté car il sera interdit d'attacher deux instances de la même classe de contrôleur à un élément de donnée. Un exemple est proposé en annexe E

6.6.2 Contrôleurs amis

Il est évident dans l'exemple précédent que les classes de contrôleur CtrlDroitsAppli et CtrlPassDroit sont liées, qu'il existe une relation de confiance entre ces contrôleurs et que pour garantir le schéma de sécurité il est nécessaire de restreindre l'accès du champ 'DroitsAcquis' à CtrlPassDroit et l'interdire à tout autre contrôleur.

Pour qualifier cette relation de confiance, nous parlerons de classes de contrôleur amies. Nous dirons que le champ 'DroitsAcquis' est un champ restreint de 'CtrlDroitsAppli' en ce sens qu'il pourra être référencé par d'autres contrôleurs, à condition que ceux-ci appartiennent à une classe de contrôleur amie.

Pour déterminer si deux classes de contrôleur sont amies, nous avons adopté le critère suivant: "Deux classes de contrôleur sont amies si et seulement si elles sont situées dans le même répertoire".

Ce critère est très simple et peut être vérifié en un temps très court, autorisant éventuellement un contrôle dynamique de l'accès: Par ailleurs se protéger des classes de contrôleur pirates revient tout simplement à protéger un répertoire contre la création d'éléments de données indésirables.

Enfin nous dirons que "deux contrôleurs sont amis si et seulement si leurs classes respectives sont amies". Deux contrôleurs attachés à des éléments de donnée appartenant à deux applications distinctes peuvent donc néanmoins être amis, mais pour qu'ils puissent réellement communiquer, il faut toutefois la condition supplémentaire qu'ils soient attachés sur une même branche. Une application doit donc être sous-application de l'autre.

6.6.3 Portée des champs d'une classe de contrôleur

Afin de qualifier la portée des champs et méthodes des classes de contrôleur, nous avons introduit quatre modificateurs de portée: **private, restricted, public** et **shell**. Les trois premiers modificateurs s'appliquent à la fois aux données membres et aux fonctions membres d'une classe de contrôleur, le modificateur 'shell' ne s'applique, pour sa part, qu'aux fonctions membres.

- **private**: ce modificateur indique que la donnée ou la fonction membre ne peut être référencée qu'à l'intérieur des fonctions membres de la classe elle-même. Tout comme en C++, il désigne les membres strictement privés d'une classe de contrôleur

- **public**: ce modificateur indique que la donnée ou la fonction membre peut être référencée librement depuis n'importe quelle fonction membre de n'importe quel contrôleur. La signification est la même qu'en C++
- **restricted**: ce modificateur indique que la donnée ou la fonction membre peut être référencée par les fonctions membres appartenant à des classes de contrôleur amies, mais pas par des classes de contrôleur étrangères
- **shell**: ce modificateur s'applique aux fonctions membres qui ne peuvent être appelées que par le noyau: OnBind, OnActivate, OnDeactivate, Before<X>, After<X>

Un exemple est là encore disponible en annexe F.

6.6.4 Conclusions

Les possibilités d'interaction entre contrôleurs ont été volontairement limitées, mais cela a trois avantages:

- c'est plus facile à implémenter

- le schéma tient compte des caractéristiques de la carte, à savoir une écriture en EEPROM coûteuse et une RAM minuscule (seul un petit nombre de contrôleur est actif à un moment donné)

Le cloisonnement induit par le mode d'adressage permet de mieux assurer la protection entre applications

6.7 Commandes étendues et problèmes d'amorçage

6.7.1 Contrôle de l'instanciation des classes de contrôleur

Puisque des contrôleurs amis attachés à des sous-applications peuvent interférer avec les contrôleurs d'une application, il est important que l'application contrôle la dissémination des contrôleurs amis dans les autres applications. Autrement dit une application doit contrôler l'instanciation et la liaison de ses classes de contrôleur.

Cette instanciation se fait au moyen de la commande étendue BIND après avoir sélectionné l'élément de donnée auquel on veut attacher le nouveau contrôleur:

BIND <Chemin D'une Classe Ctrl><Données D'initialisation Du Contrôleur>
Cette commande échoue si un contrôleur de la même classe est déjà attaché à l'élément de données; Au S-Shell d'autoriser ou non cette commande.

6.7.2 Problèmes d'amorçage

- *Personnalisation de la carte*: Si l'on part d'une carte vierge sans aucun contrôleur, il est facile de se rendre compte que toutes les commandes seront refusées, puisqu'il n'y aura aucun contrôleur pour répondre 'Oui' et qu'une commande est refusée si elle n'est pas autorisée par au moins un contrôleur.
  Pour résoudre ce problème, la carte est créée initialement avec un répertoire racine 'Root' vide et un algorithme S-Shell désactivé. Tant que le S-Shell n'est pas activé, toutes les commandes de l'utilisateur sont autorisées et il n'est pas tenu compte de l'avis des contrôleurs installés. La personnalisation de la carte se fait au moyen de la commande 'SShell' qui a pour effet d'activer la protection par contrôleurs. A ce stade, il faut souhaiter que les contrôleurs mis en place assureront une protection efficace, sans tomber dans l'excès qui consiste à interdire les commandes utiles
- *Mise en service de nouveaux contrôleurs*: Lorsque l'on vient tout juste de créer une classe de contrôleur et passée la phase de personnalisation de la carte, il faut qu'au moins un contrôleur autorise la création d'instances de ce contrôleur. Pour résoudre ce problème, il faut prendre soin, lors de la phase de personnalisation de la carte, d'attacher à Root un contrôleur ayant la méthode suivante:
  verdict BeforeBind(datelt ctrl, ...) {return App(ctrl)== **SelElt** ? ACCEPT : DMIND ; }
  Cette méthode autorise le BIND d'une classe de contrôleur donnée au répertoire application à laquelle elle appartient
- *Mise en service de nouveaux contrôleurs*: Lorsque l'on veut, passée la phase de personnalisation de la carte, mettre en place une nouvelle architecture de sécurité qui nécessite l'interaction entre plusieurs contrôleurs, on peut être confronté au besoin de mettre en place d'un seul coup plusieurs éléments de donnée avec leurs contrôleurs.
  Par exemple un contrôleur de répertoire application autorise la création d'utilisateurs, mais cette création est réservée à certains utilisateurs privilégiés. C'est le problème de la poule et de l'oeuf. Une technique utile est alors la personnalisation du contrôleur application: La création d'utilisateur est autorisée si un utilisateur privilégié s'est manifesté ou si le contrôleur n'a pas encore été personnalisé.
  La personnalisation consiste à envoyer un message utilisateur au contrôleur pour qu'il positionne à 1 un champ bit eeprom

Autres détails d'implémentation: L'identification des éléments de donnée se fait au moyen:
Soit d'un chemin, comme sous DOS à partir de l'élément courant ou de la racine, en spécifiant les 'filenames'. Par exemple: '\Root\Office\MyApp', 'Ctrls\MyCtrl', '..\Users\toto'.
Soit d'un identifiant entier unique
typedef int datelt ; // Data element id (never 0)
datelt Root ;
Le système d'exploitation permet de manipuler l'arborescence des éléments de donnée au moyen de fonctions prédéfinies:
Le système d'exploitation permet de manipuler l'arborescence des éléments de donnée au moyen de fonctions prédéfinies:

```
typedef enum { TPwd, TDir, TApp, TBin, TClass } dattyp;
dattyp Type(dateelt);
datelt Super(datelt); // returns father, Super(Root) == 0
BOOL Bound(datelt data, datelt class) // was class bound to data ?
```

La création des classes de contrôleur se fait en plusieurs étapes:

- compilation préalable à l'extérieur de la carte;

- création d'un élément de donnée vide de type Tclass et de taille définitive:
MK_CLASS <FileName><ByteSize>;

- remplissage de cet élément de donnée en plusieurs fois éventuellement (WRITE_FILE)

Les variables globales du S-Shell sont:

```
datelt SelElt ;          // mis à jour par la commande SELECT "path"

WORD ReturnCode;  // mis à jour par BeforeX en cas de refus, ou par l'OS lors de l'exécution
```

6.7.3 Résumé des commandes étendues

On utilisera les commandes étendues suivantes:

MK_CLASS <FileName> <ByteSize>;

BIND <Chemin D'une Classe Ctrl> <Données D'initialisation De La Carte>

```
SSHELL ;          // Personnalise la carte

SEND ;            // Message utilisateur
```

6.8 Simulation de la carte MCOS

Les exemples suivants montrent comment nous pouvons programmer les contrôleurs de façon à reproduire le comportement des codes secrets de la carte à microprocesseur. Un code une fois correctement présenté offre des droits d'accès en lecture et en écriture à un fichier binaire.

6.8.1 Une solution proposée (définition des contrôleurs)

La solution proposée met en jeu trois contrôleurs amis associés à trois éléments de donnée: le mot de passe, le fichier binaire et l'application. Cette dernière détient le statut de sécurité et est mis à jour par le contrôleur des mots de passe. La ratification est entièrement gérée par ce même contrôleur. La

seule action effectuée par le contrôleur du fichier binaire est d'accepter ou de refuser une commande *read* ou *write* sur le fichier binaire.

6.8.2 La programmation des contrôleurs

Un résumé de la programmation des trois contrôleurs étudiés dans cet exemple est proposé ci-dessous. Comme nous l'avons précédemment signalé, nous pouvons constater que la syntaxe est très proche de celle du langage C++. Les mot-clés sont écrits en gras.

- Mot de passe

Voici une partie du programme concernant le contrôleur lié au mot de passe. La méthode *BeforeCheck* renvoie une valeur de verdict (*ie* yes, no ou don't mind). Cette méthode accède aux contrôleurs d'application afin de vérifier le statut de sécurité courant. Pour être bien comprise, la notation (*App(this)!CSS.HasRight*) doit être découpée comme suit: *App(this)* indique l'application concernant l'élément courant. Pour l'application, la notation *!CSS.HasRight* permet l'accès au seul contrôleur dont la classe est CSS et appelle alors la méthode *HasRight*. Le bit 14 du statut de sécurité permet de déverrouiller les mots de passe. Un autre programme pourrait aussi utiliser un autre bit afin d'avoir par exemple une ratification supérieure à trois essais successifs infructueux. Après présentation le code de retour de la commande est testé et le compteur de ratification est augmenté d'une unité si le code présenté est un code d'erreur.

```
class CPassWord {

private:

eeprom BYTE ratif ; // 0..5 bad presentation

                    // 10 : canceled

eeprom WORD associatedRights ;


shell:

...

Verdict BeforeChkPwd()

{

   if (ratif < 3 ||(ratif<6 && App(this)!CSS.HasRight(14)))

   return ACCEPT ;
```

```
    else
    return Veto(PWD_LOCKED) ;
}
AfterChkPwd()
    {
    if (ReturnCode==PWD_FALSE) ratif++ ;
    if (ReturnCode==PWD_GOOD) {
    ratif=0 ;
    App(this)!CSS.SetRights(associatedRights) ;
    }
}
```

- Application

Il y a deux points importants dans ce texte:

Il existe une variable eeprom appelée *bpers* dont le but est identique au bit de personnalisation de la carte au niveau applicatif. Un autre résultat lié à l'installation d'un contrôleur est la faculté de passer des paramètres pendant l'instanciation d'un contrôleur. Pour chaque classe, il existe une méthode similaire aux constructeurs du C++. Ainsi, une même classe de contrôleur peut être appelée quelque soient les bits utilisés pour les opérations de lecture/écriture.

Les N variables font partie du mécanisme implanté de façon à rendre des droits globaux. Ces droits peuvent être obtenus à un niveau inférieur et sont conservés en remontant dans l'architecture arborescente des données.

```
class CSS {
private:
ram WORD securityStatus;// 16 local rights
                        // 15 create application
                        // 14 create & unlock pwd
```

```
eeprom BYTE alias1,alias2,alias4;// alias to upper rights
eeprom BYTE bpers;// appli Bpers

void setRights(WORD r)
  {
    securityStatus |= r ;
    WORD rsup= 0;
    if (r&1) rsup|= 1<<alias1; // transitive to upper rights
    if (r&2) rsup|= 1<<alias2; // transitive to upper rights
    if (r&4) rsup|= 1<<alias4; // transitive to upper rights
    if (rsup) App(Super(this))!CSS.setRights(rsup);
  }
.....
}
```

- Donnée

L'unique point intéressant ici est la présence de paramètres dans la méthode *BeforeReadBinary*. Ces paramètres peuvent être pris en compte lors de la décision du statut de sécurité pour cette commande. Par exemple, il est possible d'écrire un contrôleur qui autorise la lecture binaire si l'offset est un multiple de 4.

```
class CAC // access control
{
private:
eeprom BYTE rightRead;
eeprom BYTE rightWrite;
verdict beforeXX(BYTE r)
```

```
    {
    return App(SelElt)!CSS.HasRight(r) ? ACCEPT : VETO;
    }
  shell:
  verdict beforeReadBinary(int Offset, int Length)
    {
    return beforeXX(rightRead) ; }
    }
```

Pour les lecteurs qui seraient intéressés de découvrir un algorithme plus détaillé de ce que l'on peut faire grâce à ce S-Shell distribué, nous proposons en annexe G un exemple de programme visant à simuler la carte MCOS de Gemplus. Nous espérons que les commentaires de ce programme suffiront à sa bonne compréhension.

Conclusion

Pour conclure cette thèse de doctorat, nous retiendrons surtout que notre travail s'est particulièrement intéressé à des approches innovantes de la sécurité au sein de l'environnement cartes à microprocesseur. Par une étude systématique, nous avons pu retenir de différents systèmes informatiques les points forts que présentaient chacun d'entre eux. Par ailleurs, l'hétérogénéité des notions sécuritaires a elle aussi permis d'identifier des points intéressants que nous avons repris dans notre travail. La recherche s'est ensuite focalisée sur le contrôle d'accès en essayant de prouver qu'à faible coût, il était possible d'améliorer grandement la finesse et la puissance d'expression des schémas de sécurité. De surcroît, certaines notions telles que la séquentialité des actions ont pu être introduites. Nous avons donc abouti à un prototype de langage, que nous avons voulu simple et utilisable par n'importe quel responsable.

Des tests ont étayé nos propos, montrant la véracité de nos espérances. Même si l'étendue des possibilités du S-Shell semble être trop puissante dans le contexte de la carte à puce, il apparaît qu'un sous-ensemble "utile" du S-Shell peut fort bien convenir à des modèles de carte telle celle qui a été choisie dans cette thèse. Il est de plus à souligner que si nous travaillons dans le domaine de la carte d'autres peuvent fort bien reprendre le S-Shell tel qu'il est actuellement et l'adapter à leurs besoins.

De par sa maniabilité, le S-Shell pourra être facilement intégré dans des systèmes d'exploitation futurs. L'avenir semblant être tourné vers les cartes multi-applicatives (projet Carte Blanche, projet européen Cascade), l'implémentation que nous avons détaillée semble bien répondre aux besoins et aux difficultés que poseront ces cartes futures.

Annexe A

Réalisation complète du S-Shell

A.1 Le vocabulaire de S-Shell

Cette annexe très formelle a pour objectif d'identifier le lexique reconnu par S-Shell. Comme nous l'avons souligné auparavant, au chapitre 3, l'analyse lexicale de S-Shell s'appuie sur l'utilitaire Lex capable de reconnaître des chaînes de caractères. La description du vocabulaire utile à S-Shell peut se scinder en plusieurs catégories.

1. Les opérateurs: Nous possédons les caractères classiques suivants avec leurs sémantiques traditionnelles.
   - , -- , + , ++ , = , <> , < , > , <= , >= .

2. Les délimiteurs de champ: Au nombre de neuf. Ils sont spécifiques à chaque zone du fichier de description.
   , . ; : ( ) { } CR

3. Les mots prédéfinis: Ils sont utilisés tels quels avec pour les 6 derniers une parenthèse ouvrante due à l'attente d'un complément d'informations concernant ces mots-clés. Nous étudierons par la suite comment ces mots seront analysés syntaxiquement, mais nous pouvons déjà annoncer que tout mot inscrit n'appartenant pas à la liste des mots prédéfinis, s'il est syntaxiquement acceptable sera considéré comme étant un nom de variable. La liste est la suivante:
   if , and , or , not , never , always , default , time , today , matches , between , user , owner exist , read , write , execute , modified.
   Soit dix-huit mots prédéfinis acceptables en minuscule ou en majuscule indifféremment.

4. Les formats littéraux: Nous les avons classé en trois grandes catégories.

- Le format utilitaire qui se subdivise lui-même en plusieurs sous-formats, à savoir:

Format	Type	Exemple
Paramètre	${digit}	$3
Entier	{digit}*	12
Variable	{letter}({letter}|{digit})*	chainA1
Commande	"{"({digit}|{letter}|$|.|*|)*"}"	{ mv }
Chaîne	" ' "({digit}|{letter}|.|*|?|_)*" ' "	' prg.* '

Exemple: Si le programmeur entre la chaîne de caractères $2, l'analyseur lexical de S-Shell identifiera cette information comme étant du type *paramètre*.

- Le format heure se présente comme suit:

Format	Type	Exemple
Heure	{digit}{2}":"{digit}{2}	12:06

Remarque: A l'analyse d'une chaîne du type 12:34, S-Shell reconnaîtra un format heure (2 digits, le symbole ';' et de nouveau 2 digits) et mettra chaque paire de digits dans la structure *tm* d'Unix. Un algorithme permet de vérifier si l'heure entrée est cohérente (afin de ne pas se retrouver en présence d'heures du type 25:70), et de pouvoir l'exploiter dans les fonctionnalités de S-Shell.
Il est à noter de plus que l'appel de 'time' permet d'obtenir l'heure courante système.

- Le format date qui contient deux sous-formats:

Format	Type	Exemple
Date	{digit}{2}"/"{digit}{2}"/"{digit}{2}	25/04/70
Durée Jour	({digit})*(" jour")	15 jour *ou* 15 JOUR
Durée Mois	({digit})*(" mois")	2 mois *ou* 2 MOIS
Durée Année	({digit})*(" annee")	4 annee *ou* 4 ANNEE

Remarque: En ce qui concerne le format date, on peut faire la même remarque que pour le format heure, ainsi une date du type 32/13/90 sera détectée comme étant une date incorrecte. Unix permet de calculer à partir d'une date donnée de nombre de secondes qu'il s'est écoulé depuis le premier janvier 1970. La fonction réciproque existe aussi. Nous avons donc pour chaque date entrée calculé ce nombre de secondes et la nouvelle date correspondante. Si la date calculée est identique à celle entrée, elle est acceptable, sinon elle est fausse. Un message d'erreur préviendra l'utilisateur de l'entrée dans le schéma d'une date illisible. Le format durée pour sa part identifiera une entrée du type '5 jour' comme une date ayant pour valeur 05/00/00. De la même manière '5 mois' sera reconnu comme étant une date de valeur 00/05/

00 et '5 an' comme étant une date de valeur 00/00/05.
De la même manière que pour les heures, l'appel de 'today' donne la date courante.

A.2 La grammaire de S-Shell

Cette partie vise à décrire la façon dont on va utiliser les léxèmes décrits dans le paragraphe précédent. Nous allons tenter de mettre en valeur les différentes possibilités offertes par le S-Shell afin de permettre une conception aisée du schéma de sécurité voulu.

Les règles de lecture sont les suivantes:

- En *italique souligné* apparaîtront les mots devant figurer tels quels
- En italique apparaîtront les mots devant figurer tels qu'ils ont été définis au niveau de Lex dans le fichier
- En caractère droit apparaîtront les non-terminaux définis ailleurs dans la grammaire

Un diagramme syntaxique est proposé en annexe B à la suite de l'analyse grammaticale.

A.3 Exemple de schéma S-Shell

Avec la structure syntaxique que nous avons détaillée, nous pouvons aisément permettre l'impression de fichiers d'extension *doc* uniquement aux heures ouvrables en écrivant:

FIGURE 37 Exemple de description d'un schéma S-Shell

```
'lpr' :

if ((time between 08:00 and 17:00) and ($1 matches '*.doc')).
```

A.4 Les outils offerts par le S-Shell

Par soucis de clarté, nous avons classé les outils offerts par champ d'action.

Les outils relatifs aux dates

Rappel: Le format date a été défini dans la structure de données de la manière suivante:

FIGURE 38 Définition de la structure relative aux dates

```
typedef struct
{ int YY; int OO; int DD; } /* YY=Year, OO=Month, DD=Day */
SDate ;
```

Toutes les données provenant d'outils relatifs aux dates seront fournies avec cette structure

1. Le premier outil considéré permet d'obtenir la date actuelle par l'appel de *today* au niveau de Lex qui appellera dans la partie concernant l'évaluation une procédure Tool_Today
2. Le deuxième outil permet d'additionner deux dates par Tool_Dadd, en écrivant *(date + date)* ou *(date + duree)*
3. Le troisième outil permet de comparer deux dates par Tool_RDat, en écrivant *(date op_rel date)*

Les outils relatifs aux heures

Rappel: Le format heure a été défini dans la structure de données de la manière suivante:

FIGURE 39 Définition de la structure relative aux heures

```
typedef struct
{ int HH; int MM; } /* HH =Hour , MM=Minute */
STime ;
```

Toutes les données provenant d'outils relatifs aux heures seront fournies avec cette structure

1. Le premier outil considéré permet d'obtenir l'heure actuelle par l'appel de *time* au niveau de Lex qui appellera dans la partie concernant l'évaluation une procédure Tool_Time
2. Le deuxième outil permet d'additionner deux heures par Tool_Tadd, en écrivant *(heure + heure)*
3. Le troisième outil permet de comparer deux dates par Tool_RTim, en écrivant *(heure op_rel heure)*
4. Le quatrième outil offre la possibilité de vérifier qu'une heure est bien située entre deux heures données. Cette procédure appelle Tool_Betw à la lecture de (*heure* between *heure* and *heure*)

Les outils relatifs aux fichiers

Ils sont classiques. Ils concernent pour les fichiers:

1. Le droit de lecture par Tool_Read avec l'appel de read(*expression*)
2. Le droit d'écriture par Tool_Write avec l'appel de write(*expression*)
3. Le droit d'exécution par Tool_Execute avec l'appel de execute(*expression*)

Nous noterons de plus la possibilité de connaître:

4. L'existence d'un fichier par l'appel de exist(*expression*)
5. La date de dernière modification d'un fichier par Tool_Modi. On remarquera que l'on aurait aussi bien pu classer ce dernier outil relatif aux fichiers dans le paragraphe des outils concernant les dates. On appelle modified(*expression*)

Les outils relatifs aux personnes

1. Le premier outil permet de connaître l'utilisateur courant par Tool_User, en écrivant user
2. Le second donne le propriétaire d'un fichier par Tool_Own, en écrivant owner(*expression*)

Les outils relatifs aux chaînes de caractères

1. Nous possédons la comparaison entre deux chaînes de caractères par Tool_RStr, à l'appel de (*expression op_rel expression*)
2. Enfin nous pouvons vérifier si une expression est incluse ou pas dans une autre à l'appel de (*expression* matches *expression*)

A.5 La structure de données de S-Shell

Le descriptif

Nous définissons ici les différents types à associer à la grammaire Yacc. Ces types seront décrits dans un fichier *sd.h* déclaré en *include* dans le programme YaccShell de reconnaissance grammaticale de S-Shell.

En guise d'exemples aux éléments liste_spécification, contrainte, expression_logique, facteur et terme seront associés les structures TSpec, TComm, TCont, TLog, TFact et TTerme.

FIGURE 40 Exemple de définitions de types de la structure de données du S-Shell

```
typedef struct TSpec *PSpec;

typedef struct TComm *PComm;

typedef struct TCont *PCont;
```

Ainsi donc, sachant qu'un (not) terme peut être soit une expression_logique, soit une expression_-relationnelle, soit read, write, execute ou exist d'une expression, on peut construire la structure de données relative à TTerme.

La partie déclarative sera définie comme suit:

FIGURE 41 Extrait de la partie déclarative de la structure de données du S-Shell

```
#define V_SUBEXP, V_RELEXP, V_READ, etc...

typedef struct TTerme

{

int TypeTerme;

union {

PLog SubExpr;

PRel RelExpr;

PString StrExpr;

PTerme Terme;

} u
```

Il reste alors à créer la structure de données comme suit:

FIGURE 42 Extrait de la structure de données du S-Shell

```
PTerme Creer_V_Rel (Log) /* Cas de l'expression logique */
PLog Log;
{
PTerme tmp;
tmp=(PTerme) malloc(sizeof(struct TTerme));
tmp->TypeTerme=V_SUB;
tmp-> u.SubExpr=Log;
return(tmp);
}
```

Et ainsi de suite pour chaque pointeur de la structure. Cette programmation structurée permet de pouvoir aisément d'évaluer l'espace mémoire requis au stockage de la structure de données.

A.6 L'évaluateur de S-Shell

Cette partie du programme n'effectue qu'un appel des outils d'évaluation (programme eval.c) en fonction de la structure appelée.

En reprenant la partie traitée ci-dessus, nous obtenons:

FIGURE 43 **Extrait du programme de l'évaluation**

```
int Eval_Terme(Terme)
PTerme Terme;
{
switch (Terme->TypeTerme)
{
case V_SUBEXP : return(Eval_Log(Terme->u.SubExpr));
case V_ RELEXP: return(Eval_Rel(Terme->u.RelExpr));
case V_READ : return(Tool_Read(Eval_Str(Terme->u.StrExpr)));
case V_WRITE : return(Tool_Write(Eval_Str(Terme->u.StrExpr)));
case V_EXECUTE : return(Tool_Execute(Eval_Str(Terme->u.StrExpr)));
case V_ EXIST: return(Tool_Exist(Eval_Str(Terme->u.StrExpr)));
case V_ NOT: return(!Eval_Terme(Terme->u.Terme));
}
```

Schéma récapitulatif

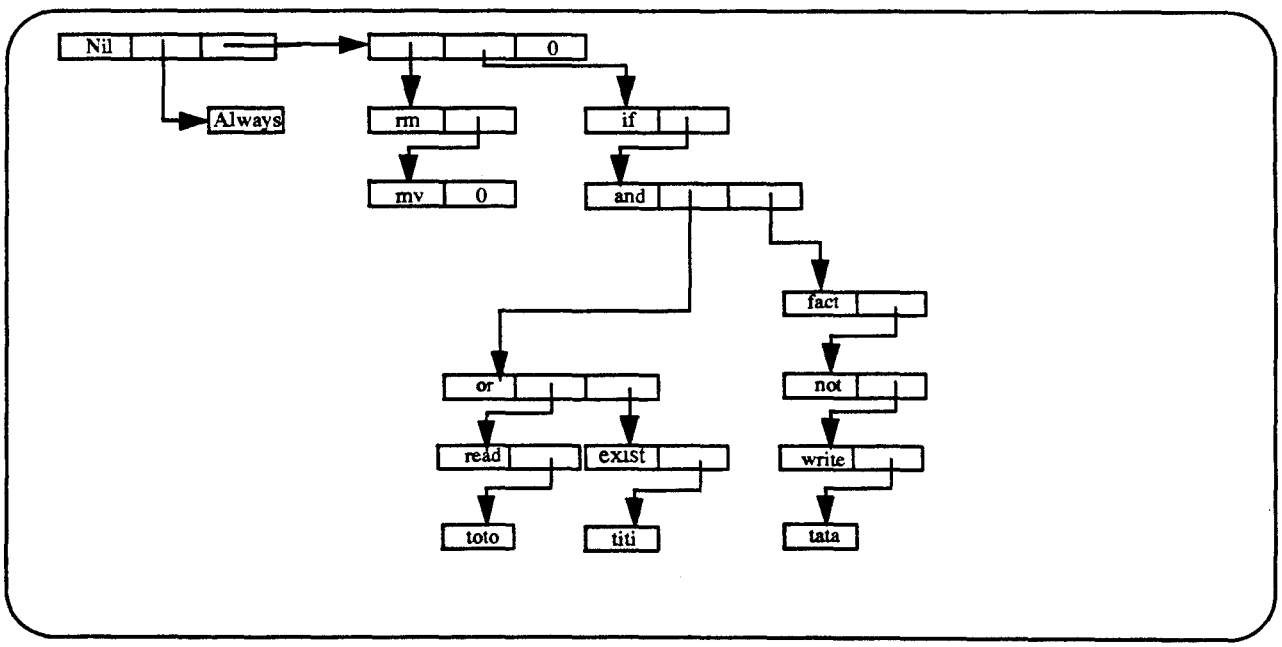
D'après ce qui a été dit dans les chapitres antérieurs, si l'on se donne le schéma ci-dessous, nous pouvons en déduire l'arborescence qui découlera du modèle:

FIGURE 44 **Modèle simple de schéma S-Shell**

```
default :

always ;

'rm' 'mv' :

if read('toto') or exist('titi') and not write('tata') .
```

On obtient dans ce cas précis la configuration arborescente suivante:

FIGURE 45 **Construction de l'arborescence**



Annexe B

Grammaire complète du S-Shell

B.1 Définition de la grammaire

Ainsi qu'il l'a été signalé en annexe A, nous proposons ici la grammaire complète du S-Shell telle que nous l'avons envisagée. Le tableau de description de la grammaire de S-Shell est le suivant. Nous rappellons les règles de lecture préalablement énoncées en A.2:

- En *italique souligné* apparaîtront les mots devant figurer tels quels
- En italique apparaîtront les mots devant figurer tels qu'ils ont été définis au niveau de Lex dans le fichier
- En caractère droit apparaîtront les non-terminaux définis ailleurs dans la grammaire

FIGURE 46 Grammaire du S-Shell

fichier	:==	*default : cr* contrainte liste_spécification
		default partie_actions *: cr* contrainte liste_spécification
liste_spécification	:==	.
		: cr liste_commande *: cr* contrainte liste_spécification
		partie_actions *: cr* liste_commande *: cr* contrainte liste_spécification
partie_actions	:==	*{* liste_actions *}*
liste_commande	:==	expression
		expression , liste_commande
contrainte	:==	*if* expression_logique
		always
		never

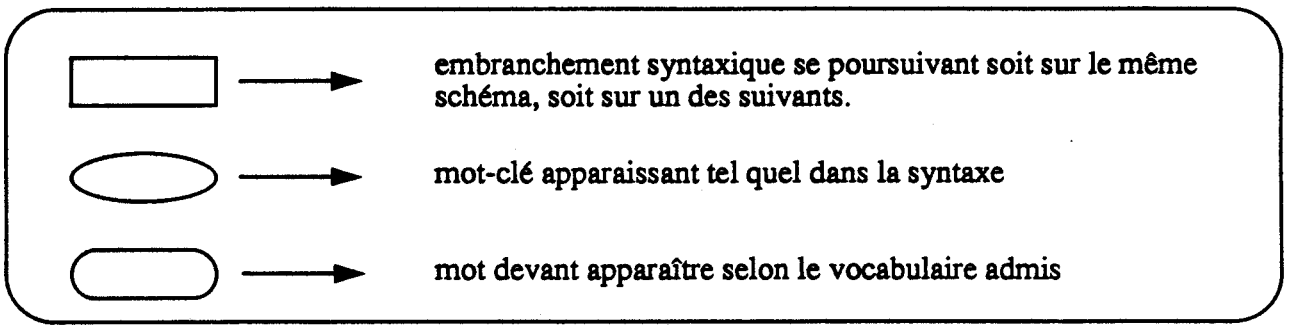
expression_logique	:==	facteur *and* facteur
		facteur
facteur	:==	terme *or* terme
		terme
terme	:==	(expression_logique)
		expression_relationnelle
		read(expression)
		write(expression)
		execute(expression)
		exist(expression)
		not terme
liste_actions	:==	*variable* = expression ;
		liste_actions *variable* = expression ;
		variable - - ;
		liste_actions variable - - ;
		variable ++ ;
		liste_actions variable ++ ;
expression_relationnelle	:==	(expression op_rel expression)
		(expression *matches* expression)
		(expression *between* expression *and* expression)
		cmd
op_rel	:==	=
		<>
		<
		>
		<=
		>=
expression	:==	user
		owner(expression)
		paramètre
		variable
		chaîne
		entier

		(expression ± expression)
		(expression - expression)
		date
		today
		modified(expression)
		durée_jour
		durée_mois
		durée_année
		durée_jour durée_mois
		durée_mois durée_jour
		durée_jour durée_année
		durée_année durée_jour
		durée_mois durée_année
		durée_année durée_mois
		durée_année durée_jour durée_mois
		durée_année durée_mois durée_jour
		durée_mois durée_jour durée_année
		durée_mois durée_année durée_jour
		durée_jour durée_mois durée_année
		durée_jour durée_année durée_mois
		heure
		time

B.2 Description syntaxique

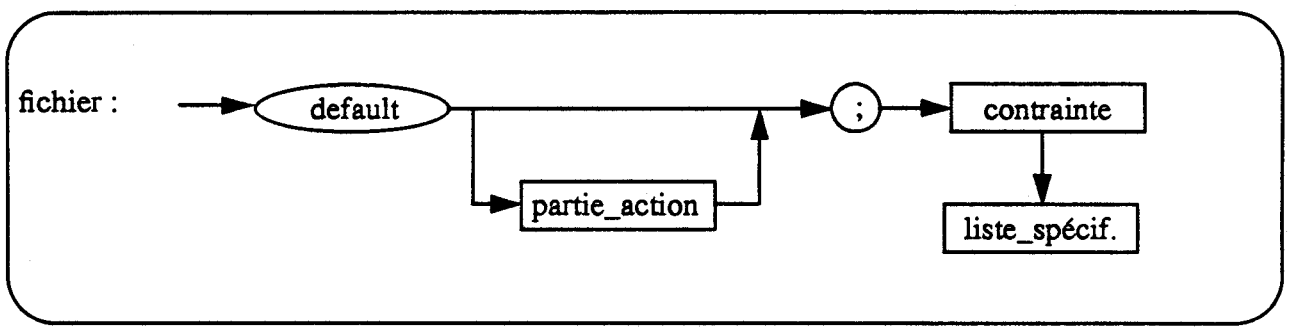
En parallèle à cette écriture, nous proposons une description syntaxique reprenant la sémantique de tous les éléments importants de notre arborescence. Chaque élément en bout de chaîne ne doit pas être considéré comme un élément terminal et il convient à chaque fois de se reporter à la figure suivante pour voir comment il s'exprime syntaxiquement.

FIGURE 47 Légende des formes employées dans la description syntaxique'



Selon ces normes de lecture, la description syntaxique est donc:

FIGURE 48 Syntaxe de l'exception 'default'



Comme précisé dans la légende, les parties_actions, les contraintes et les listes de spécification seront développés selon la même règle par la suite.

FIGURE 49 Syntaxe de la liste des spécifications

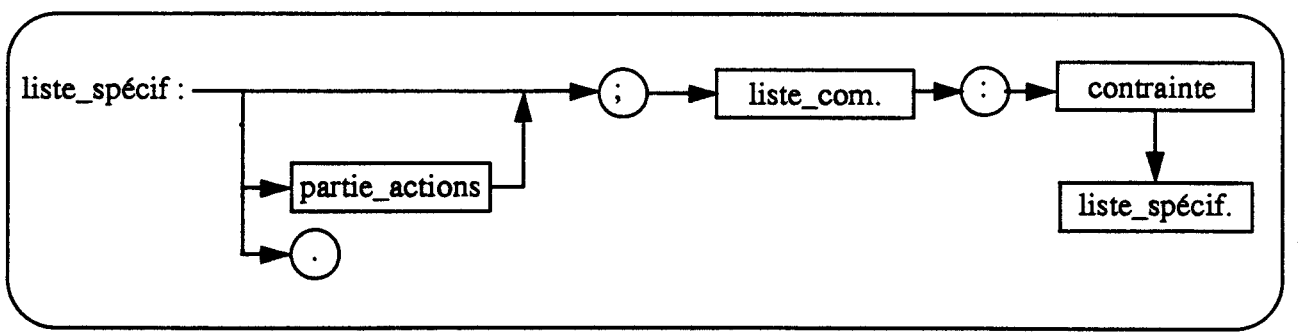


FIGURE 50 Syntaxe de la liste des attributs utilisés pour les spécifiations

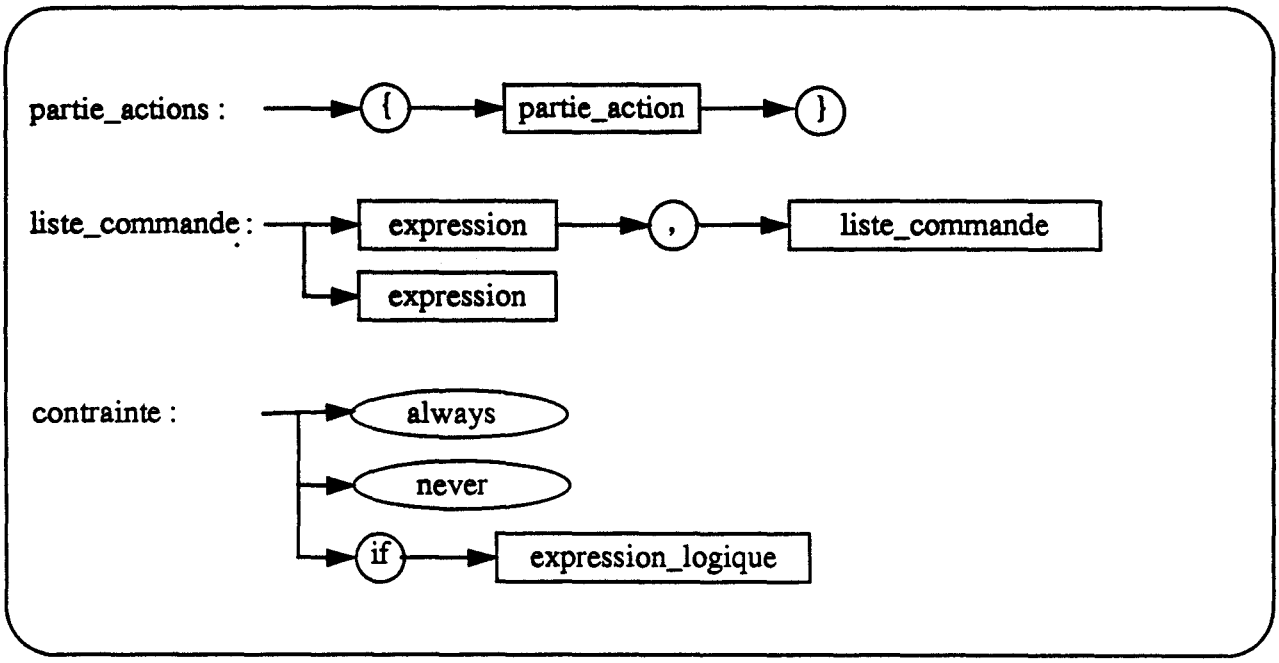


FIGURE 51 Syntaxe de la liste des attributs utilisés pour les spécifiations

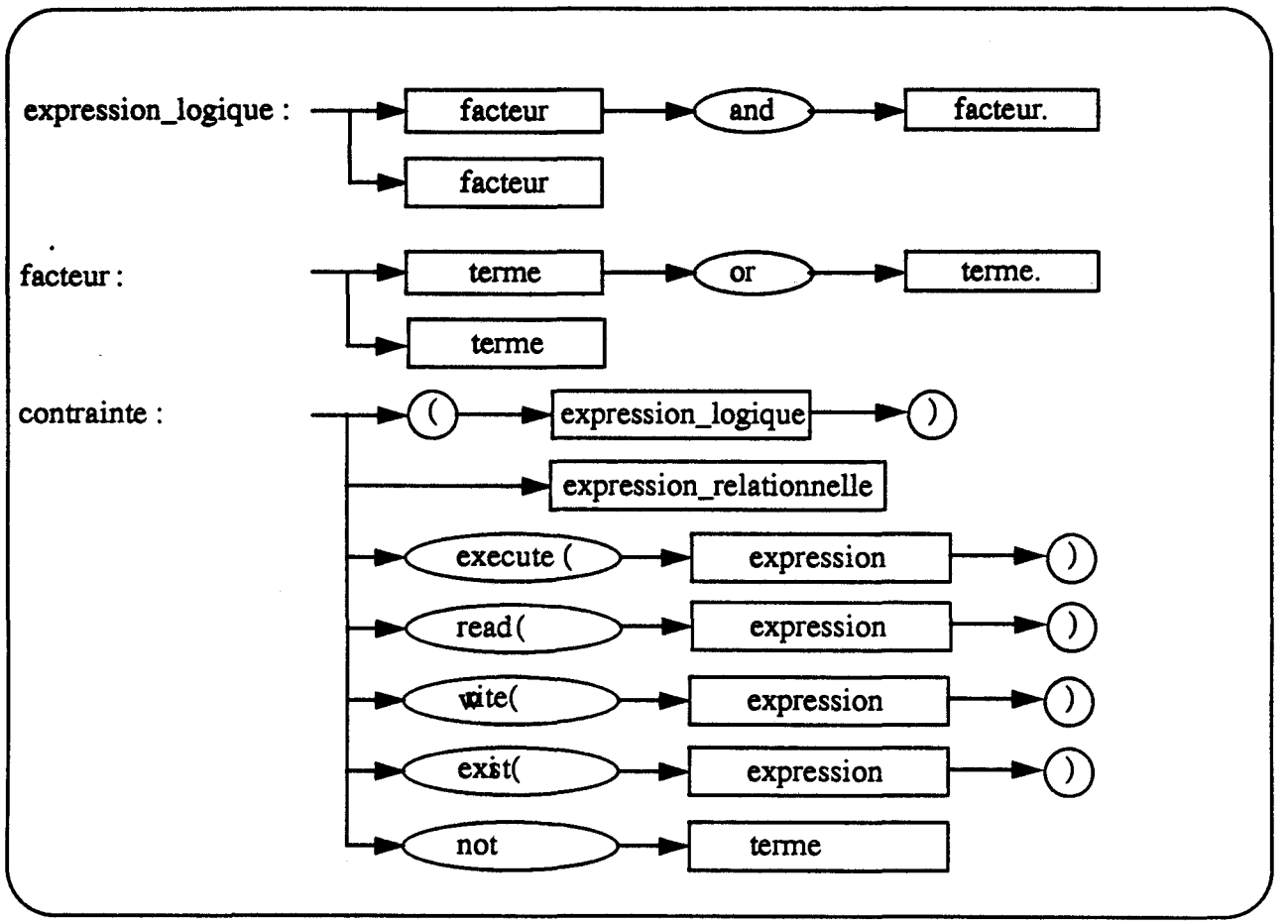


FIGURE 52 Syntaxe de la liste des actions

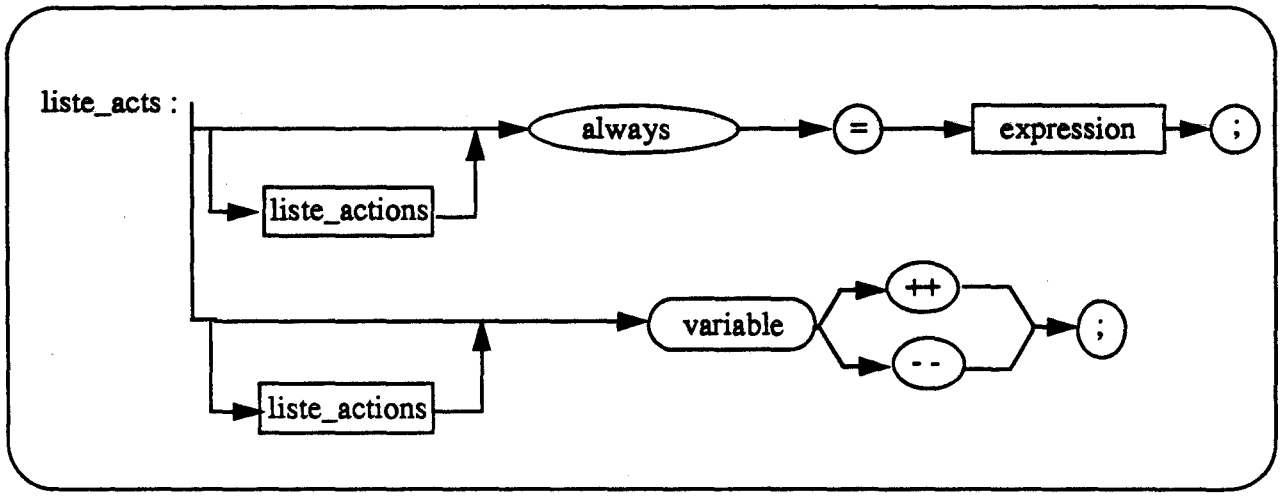


FIGURE 53 Syntaxe des listes d'action

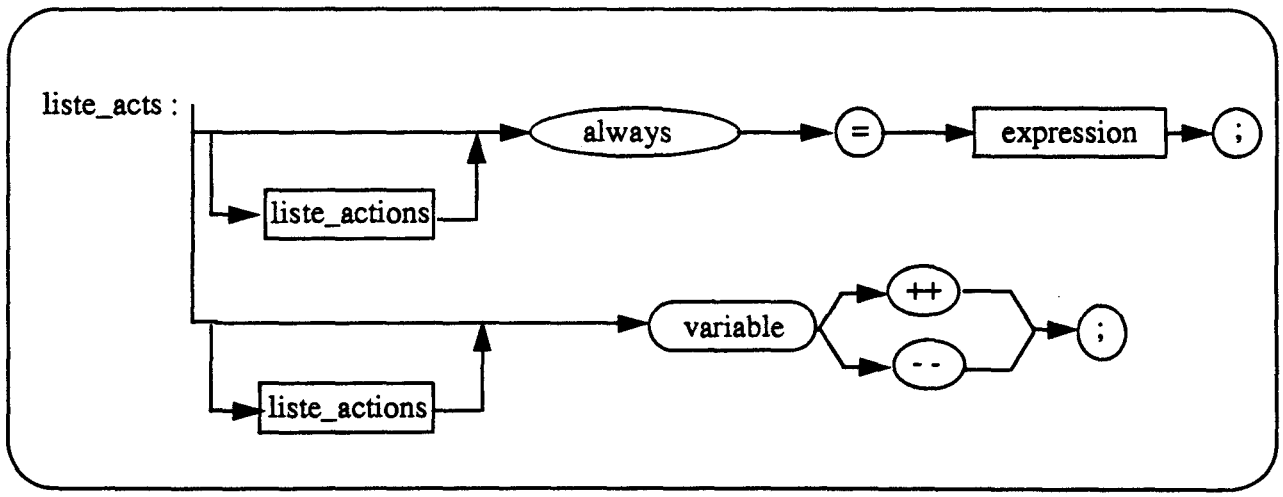


FIGURE 54 Syntaxe des expressions relationnelles

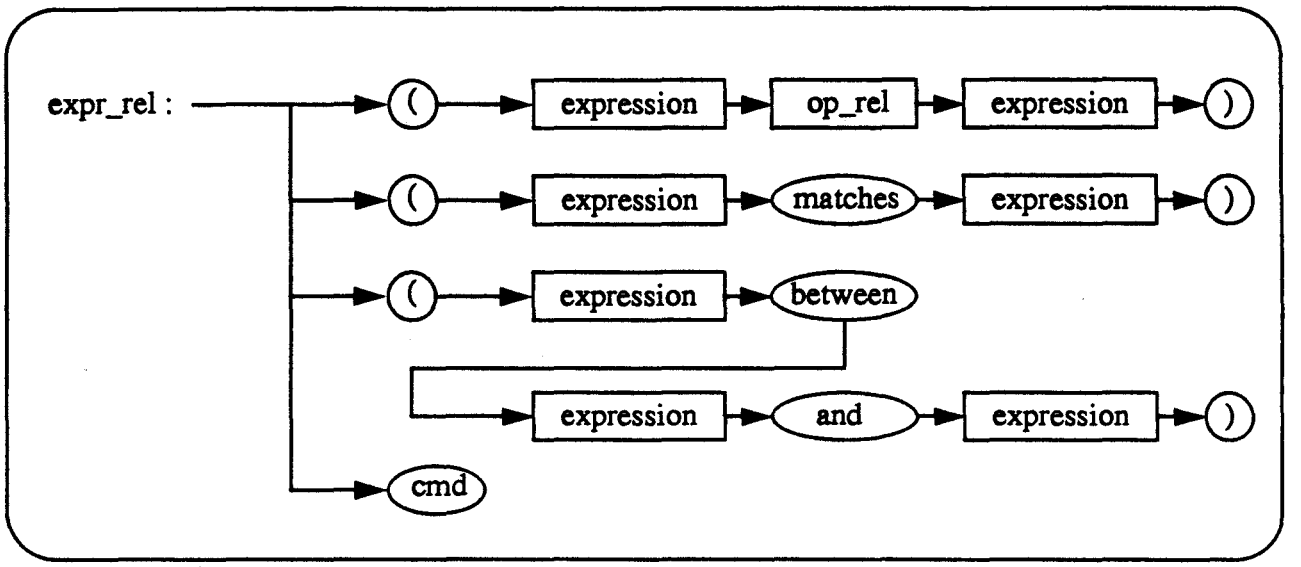


FIGURE 55 Syntaxe des opérateurs relationnels

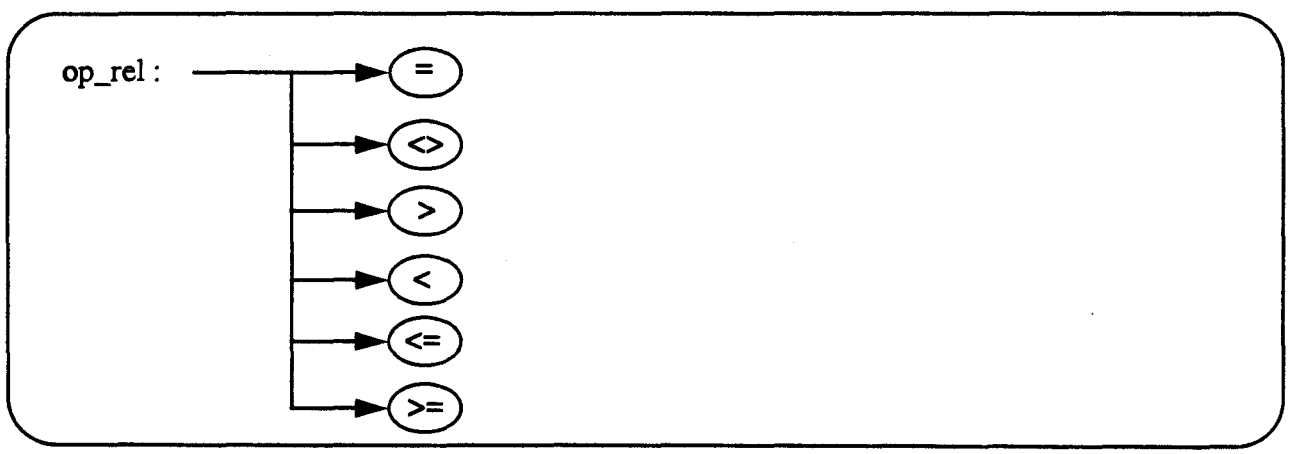
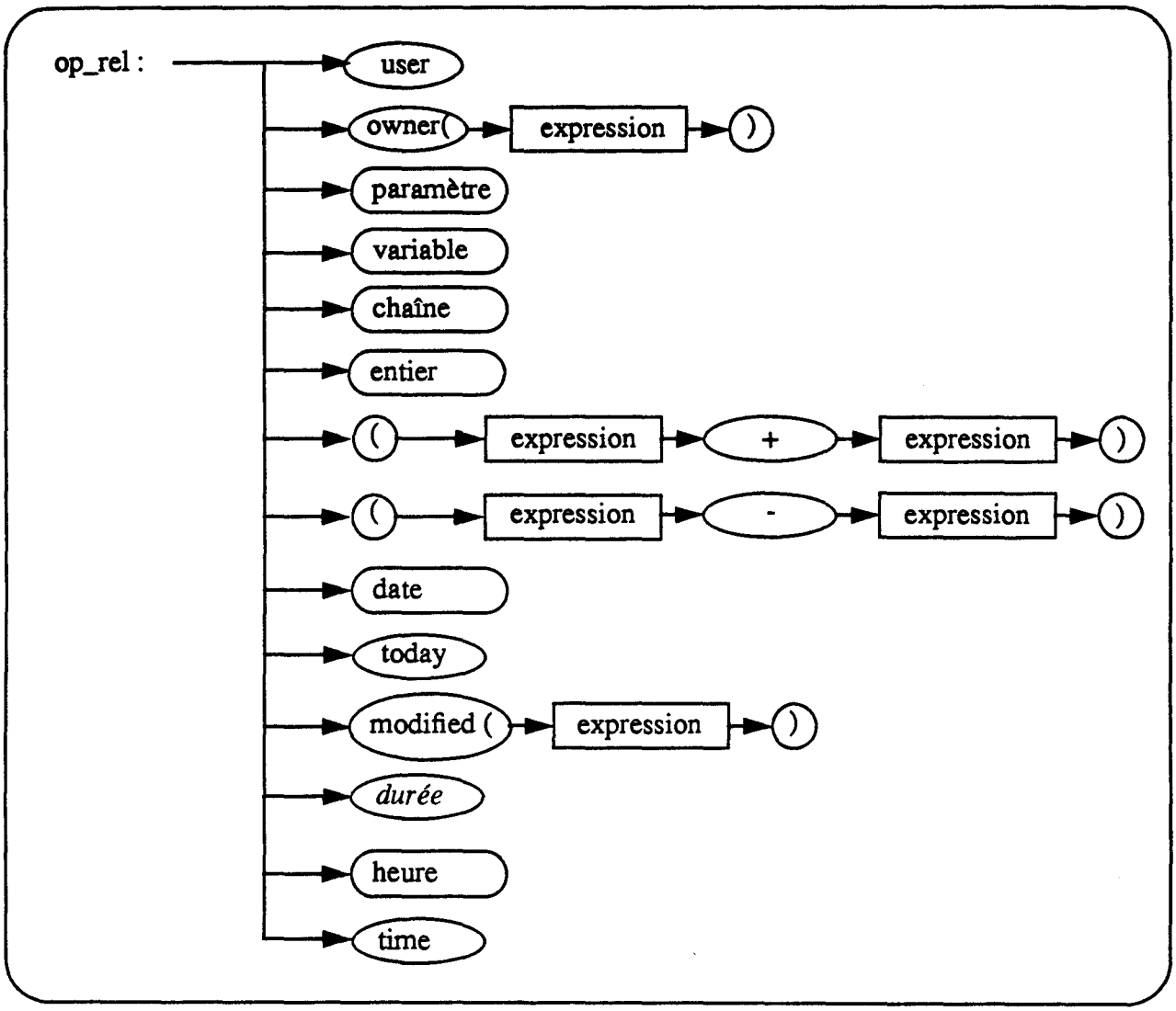


FIGURE 56 Syntaxe des opérateurs relationnels



Afin de ne pas alourdir ce diagramme, en écrivant *durée*, nous voulons simplement signifier l'ensemble de tous les formats durée_jour, durée_mois et durée_année seuls ou associés entre eux. Ce choix a été fait afin de faciliter la lecture de ce schéma. Une écriture répétitive des 15 structure de durée ne pourrait qu'en rendre plus difficile la compréhension.

Annexe C

Algorithme du S-Shell objet

```
typedef enum { ACCEPT , VETO } verdict;
int CodeResultat;

class Contrôleur {
privé:
    les variables d'état du S-Shell
    les méthodes privées associées au schéma de sécurité retenu
public:
    verdict BeforeReadFile(int Offset, int Length);
    verdict BeforeWriteFile(int Offset, int Length);
    AfterReadFile(int Offset, int Length);
    AfterWriteFile(int Offset, int Length);
    ....

verdict Contrôleur:: BeforeReadFile(int Offset, int Length)
```

```
{
   Déterminer si la commande est autorisée
   V <-- ACCEPT or VETO
   CodeResultat <- si V == VETO
   return V;
}

main ()                                    // Cette fonction ne fait pas partie du S-Shell
{
   var S-Shell : Contrôleur;
   var Verdict : verdict ;
   Répéter indéfiniment
      {
      Saisie de la commande ;
      Analyse syntaxique de la commande
      Verdict <-- S-Shell.Before<Command>(<paramètres de la commande>)
      Si (Verdict == ACCEPT) {
      faire exécuter la commande par le système d'exploitation
      CodeResultat <-- Code résultat du système d'exploitation
      }
S-Shell.On<Command>(<paramètres de la commande>);
Envoyer le résultat de la commande à l'utilisateur
}
}
```

Annexe D

Algorithme d'arbitrage

L'algorithme d'arbitrage peut se résumer à la réécriture de la fonction main proposée au chapitre 6, paragraphe 3 pour un S-Shell mono-contrôleur. On notera de plus que l'algorithme met en évidence que seuls les contrôleurs sélectionnés lors d'un premier appel seront de nouveau appelés au second tour.

```
main ()                                    // Cette fonction ne fait pas partie du S-Shell

{

   var Verdict : verdict ;
   Répéter indéfiniment

   {
   Saisie de la commande ;
   Analyse syntaxique de la commande

                                           // Premier appel des contrôleurs

   Verdict <-- DMIND//Verdict de l'arbitre

   CLast <-- Nul//Dernier contrôleur interrogé au second tour

   Soit LPath la liste des éléments de données depuis le répertoire racine jusqu'à l'élément de
   donnée concerné par la commande
   Foreach E in LPath

   Soit LCont la liste des contrôleurs attachés à E

   Foreach C in LCont
```

```
    {
    CLast <-- C
    Selon C-> Before <Command>(<Paramètres de la commande>)
        {
        cas ACCEPT :
        Verdict <-- ACCEPT

        cas VETO :
        Verdict <-- VETO
        Sortir des deux boucles

        cas DMIND :

        }
    }
}
                                        // Exécution éventuelle de la commande
Si (Verdict == ACCEPT)
{
Faire exécuter la commande par le Système d'exploitation
CodeRésultat <-- Code résultat du Système d'exploitation
}
                                        // Deuxième appel des contrôleurs
Foreach E in LPath
{
Soit LCont la liste de contrôleurs attachés à E
Foreach C in LCont
{
```

C-> After<Command>(<paramètres de la commande>);

Si (C == CLast) Sortir des 2 boucles

}

}

Envoyer le résultat de la commande à l'utilisateur

}

}

Les règles d'arbitrage que nous avons explicité sont figées et seront implantées en ROM. L'utilisateur de la carte peut créer des classes de contrôleurs et associer leurs instances aux éléments de données, mais il ne peut pas remettre en cause l'algorithme d'arbitrage.

Annexe E

Exemple d'adressage de contrôleurs

Afin d'illustrer notre propos concernant l'adressage de contrôleurs, voici un exemple intéressant:

```
class CtrlDroitsAppli { // Attacher à une application
ram droitsAcquis;
}

class CtrlPassDroit { // Attacher à un mot de passe
eeprom int droitsAssociés;
eeprom char motPasse[8];
eeprom int ratif;
void AfterChkPwd();
}
```

```
void CtrlPassDroit :: AfterChkPwd(char *pwd)
{
if (CodeResultat == GOOD) {
                                        // lever les droits au niveau du contrôleur d'application
App(this) : CtrlDroitsAppli.droitsAcquis OR = droitsAssociés;
}
else ratif=ratif+1;
}
```

Annexe F

Portée des champs de contrôleurs

Voici en guise d'illustration un exemple de restrictions envisageables:

```
class CtrlDroitsAppli { // Attacher à une application

restricted:
ram droitsAcquis;

}


class CtrlPassDroit { // Attacher à un mot de passe

private:

eeprom int droitsAssociés;

eeprom char motPasse[8];

eeprom int ratif;

shell:

void AfterChkPwd();

}
```

Annexe G

Simulation de la carte MCOS

```
************************************
* OPERATING SYSTEM COMMANDS *
************************************

BIND ControllerClass InitdataLength Initdata          // on SelElt
CHKPWD string8                        // Type(SelElt) == TPwd
USRCMD CtrlClassName CmdNum p1 p2
RDBIN Offset Length
WRTBIN Offset Length Data
UPDBIN Offset Length Data

#define PWD_LOCKED 0x9840
#define PWD_FALSE 0x9804

**********************
```

```
* PREDEFINED TYPES *
**********************

typedef int datelt;                          // data element id (0 <--> not found)
typedef enum { TPwd , TDir , TApp , TBin , TClass } dattyp;
typedef enum { ACCEPT , VETO , DMIND } verdict;

*****************************************
* PREDEFINED CONSTANTS & FUNCTIONS *
*****************************************

dattyp Type(datelt);
datelt Root;                                 // Type(Root) == TApp
datelt Super(datelt);                        // returns father; Super(Root) == 0
datelt LookDir(datelt eltDir, char *eltName);
// search a data element in a directory (if not found) ->)
BOOL Bound(datelt data, datelt clss)         // was class bound to data ?

BOOL IsApp(datelt e) { return Type(e) == TApp; }

datelt App(datelt E)
{
while(E && Type(E) != TApp)E = Super(E);
```

```
return E;
}

************************************
* GLOBAL PREDEFINED VARIABLES *
************************************

datelt SelElt;                          // current user selected data element id
WORD ReturnCode;

verdict Veto(WORD Cod) { ReturnCode = Cod; return VETO; }

***************
*class CDefault *
***************

Class CDefault {
shell:
verdict BeforeBind(datelt ctrl, ... ) { return App(ctrl) == SelElt ? ACCEPT : DMIND ; }
} ;

******************
* class CPassWord *
```

```
**********

#define CPASSWORD_USER_CANCEL 1

class CPassWord {
private:
eeprom BYTE ratif;                    // 0,1,2, -> 3,4,5 count of bad presentations
                                      // 10 when cancelled
eeprom WORD associatedRights;

shell:
OnBinding(WORD Rights)
{ ratif = 0; associatedRights = Rights; }

Verdict BeforeChkPwd()
{ return (ratif<3 || ratif<6 && App(this)!CSS.HasRight(14)) ?
ACCEPT :Veto(PWD_LOCKED); }

AfterChkPwd()
{ if (ReturnCode == PWD_FALSE) ratif ++;
if (ReturnCode == PWD_GOOD) {
ratif = 0;
App(this)!CSS.Setrights(associatedRights);
```

```
}
}

verdict BeforeUserMsg(int msg, int param1, int param2)
{ if (msg == CPASSWORD_USER_CANCEL) return ACCEPT; }

AfterUserMsg(int msg, int param1, int param2)
{ if (msg == CPASSWORD_USER_CANCEL) ratif = 10;}
};

**********
* class CSS*
**********

#define CSS_USER_SETBPERS 1

class CSS {

private:
ram WORD securityStatus;          // b16 local rights
                                  // b15 : can create applications
                                  // b14 : can create and unlock passwords
```

```
eeprom BYTE alias0, alias1, alias2, alias3;
                                        // upper alias to local b0 .. b3 right bits
eeprom BYTE pers;

restricted:
void SetRights(WORD r)
{
securityStatus |= r;
WORD rsup |= 0;
if (r&1) rsup |=1 << alias0;
if (r&2) rsup |=1 << alias1;
if (r&4) rsup |=1 << alias2;
if (r&8) rsup |=1 << alias3;
if (rsup) App(Super(this))!CSS.SetRights(rsup);
}

BOOL HasRight(WORD b)
{ return(securityStatus & (1<<b)) !=0 ; }

shell:
OnBind(BYTE r0, BYTE r1, BYTE r2, BYTE r3);
{alias0 = r0; alias1 = r1; alias2 = r2; alias3 = r3; }
```

```
verdict BeforeBind(datelt ctrl , ...)
{
                                        // control binding of CSS :
if (ctrl == thisclass) {
if Type(SelElt)!= TApp) return VETO;    // bind only to Applications
datelt BedApp = App(Super(SelElt));     // embedding application
if (BedApp == this) return HasRight(15) ? ACCEPT : VETO;
                                        // b15 enables application creation and rights mapping
return Bound(BedApp,CSS) ? DMIND : VETO;
                                        // only embedding application security server should
                                        // authorize
}

                                        // control binding of CPassWord :
if (ctrl == CPassWord) {
if (Type(SelElt)!= TPsw) return VETO;
                                        // bind only to passwords
if (App(SelElt) == this) return !bpers || HasRight(14) ? ACCEPT :VETO;
                                        // b14 enables password creation when bpers ...
return Bound(App(SelElt),CSS) ? DMIND : VETO;
                                        // only embedding application security server should
                                        // authorize
}
```

```
                                        // control binding of CAC : accept iff existing security
                                        // server
if (ctrl == CAC) return Bound(App(SelElt),CSS) ? ACCEPT : VETO;
return DMIND ;
}

verdict BeforeUserMsg(int msg, int param1, int param2)
{ if (msg == CSS_USER_SETBPERS) return ACCEPT; }

AfterUserMsg(int msg, int param1, int param2)
{ if (msg == CSS_USER_SETBPERS) bpers= TRUE; }
};

**************************
* class CAC - Access control *
**************************

class CAC {

private :
BYTE rightRead ;                        // requested rights for ...
BYTE rightWrite;
BYTE rightUpdate;
```

```
verdict beforeXX(BYTE r)
{ return App(SelElt)!CSS.HasRight(r) ? ACCESS : VETO; }

shell:

OnBinding(BYTE rRd, BYTE rWrt, BYTE rUpd)
{ rightRead = rRd; rightWrite = rWrt; rightUpdate = rUpd; }

verdict BeforeReadBinary(int Offset, int length)
{ return beforeXX(rightRead); }
verdict BeforeWriteBinary(int Offset, int length, ...)
{ return beforeXX(rightWrite); }
verdict BeforeUpdateBinary(int Offset, int length)
{ return beforeXX(rightUpdate); }

};

**********************
* Initialization sequence *
**********************

                                    // 1 : creation of controllers somewhere
```

```
rapp = SELECT "\Rott\ ... \MyApp" ;
srep = MKDIR "DSceu";
SELECT srep
css = MKCTRL "CSS" <size>
->write it, valid it
cpw = MKCTRL "CPassWord" <size>
-> write it, valid it
cac = MKCTRL "CAC" <size>
-> write it, valid it
SELECT rapp;
BIND cs 0 0 0 0                    // CDefault Accepts it ...
prep = MKDIR "DPwds"               // the passwords directory
SELECT prep
uph = MKPWD "Phil" "OBOULO"
SELECT uph
BIND cpw 0xFF                      // grant phil all rights! accepted because CSS not pers.
SELECT rapp                        // MyApp
USRCMD css 1 0 0 0                 // CSS_USER_SETBPERS
```

Bibliographie

[A90] AFCET, European Symposium On Research In Computer Security, 1990

[Af92] AFCET, "*Glossaire Didactique De La Sécurité Des Systèmes D'information*", Comité Technique "Sécurité et sûreté informatiques", version 1.2, Oct. 1992

[Al95] T. Alexandre, "*Manipulation De Données Multimédia Dans La Carte A Microprocesseur: Application A L'identification Biométrique Et Comportementale*", Thèse D'informatique LIFL, Feb. 1995

[Am83] S. Ames et al., "*Security Kernel Design And Implementation: An Introduction*", in Computer, v16 n7, pp 14-23, Jul. 83

[An92] R.J. Anderson, "*UEPS - A Second Generation Electronic Wallet*", ESORICS 92, number 648 in Lecture Notes in Computer Science, pp 411-418 , Springer-Verlag, 1992

[An93] R.J. Anderson, "*Why Cryptosystems Fail*", in Proceedings of the 1993 ACM Conference on Computer and Communication Security, p p215 - 227, 1993

[AC94] T. Alexandre, V. Cordonnier, "*The Radar Concept*", in Proceedings of the Cardtech/ Securtech '94 International Conference, Arlington, Virginia, USA, April 10-13, pp 87-98, 1994

[AF94] Y. Amghar, A. Flory, "*Integrity Constraints In Object-Oriented Databases*" , *Engineering of Informtation Systems*", Vol. 2, No 5/1994, Ed. Hermes, ISSN 1247-0317, AFCET, 1994

[AT95] T. Alexandre, P. Trane, "*Detecting Intrusions In Smart Card Applications Using Expert Systems And Neural Networks*", in Proceedings of the IFIP TC11, Conference on Information Security, IFIP/Sec "95, Cape Town, South Africa, pp 507-519, 9-12 May 1995

[Bi77] K. J. Biba, "*Integrity Considerations For Secure Computer Systems*", Technical Report ESD-TR-76-372, ESD/AFSC, Hanscom AFB, Bedford, Mass., 1977.

[Ba87] R. Baldwin, "*Rule Based Analysis Of Computer Security*", in Proceedings 1987 Compcon, pp 227-233, 1987

[Be83] D. Bell, "*Secure Computer System : A Retrospective*", in Proceedings IEEE Symposium Security &Privacy, IEEE Comp. Soc., pp 125-131, 1983

[Bo88] C. Bonnin, "*SQL Bases Relationnelles*", Ed. Eyrolles, 1988

[Bo94] J. P. Boly, "*The ESPRIT Project CAFE - High Security Digital Payment Systems*", ESORICS 94, number 875 in Lecture Notes in Computer Science, pp 217-230 , Springer-Verlag, 1994

[Bo95] E. Bovelander, "*Evaluations Of Smart Card Based Security Systems: Is Your Smart Card Really Secure?*", Project Manager at TNO Evaluation Center For Instrumentation & Security Techniques, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 149-154, 1995

[Br88] R. Bright, "*Smart Cards: Principles, Practice, Applications*", Ellis Horwood Books in Information Technology, Halsted Press : A division of John Wiley & Sons, 1988

[Bu90] R. K. Burns, "*Integrity And Secrecy: Fundamental Conflits In The Database Environment*", in Proceedings of the 3rd RADC Database Security Workshop, 1990

[Be92] A. Berson, "*Client/Server Architecture*", Mc Graw-Hill Ed., 1992

[BE95] H. Booysen, J. Eloff, "*A Methodology For The Development Of Secure Application Systems*", in Proceedings of the 11th International Conference on Information, IFIP/ SEC '95, pp 230 to 244, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[BL73] D. Bell, L. LaPadula, "*Secure Computer Systems : Mathematical Foundations And Model*", MITRE Report MTR 2547, v2, Nov. 1973

[BM84] JM. Busta, S. Miranda, "*Introduction Aux Bases De Données*", Ed. Eyrolles, pp 11 to 22, 1984

[C94] O. Caron, "*Méthodologies De Conception Et D'évaluation D'architecture R.I.S.C. Adaptées Aux Futures Cartes A Microprocesseur*", Thèse de Doctorat en Informatique du L.I.F.L., No 1256, Chapitres 1 et 2, Jan. 1994

[Ca94] Bob Carter, "*The Present And Future State Of Biometric Technology*", CardTech SecurTech '94, Conference Proceedings, pp 401-415, 1994

[Co89] R. Courtney, "*Some Informal Comments About Integrity And The Integrity Workshop*", in Z. Ruthberg and W. Polk Editors. Report of the Invitational Workshop on Data Integrity, Gaithersburg, 1989

[Cr90] J. Mc Crindle, "*Smart Cards*", IFS Publications/ Springer-Verlag, 1990

[CL91] D. Colnet, D. Léonard, "*Les Langages A Objets*", InterEditions, Paris, 1991

[CEC92] Commission of the European Communities, "*Security Investigations*", Information Security '92, Document reference DGXIII/F-GE1190/G1, revision Jan. 1992

[CG91] V. Cordonnier, G. Grimonprez, "*Smart Cards And Portable Data Files : A Glance At The Future*", Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol. 13, N03, pp 1387, 1991

[CK90] R. C. Connor, G. N. Kirby, "*Protection In Persistent Object Systems*", in Security and Persistence, J. Rosenberg & J. L. Keedy (ed), Springer-Verlag, pp 48-66, 1990

[CL90] T.Y. Chua, G. Lisimaque, "*Smart Cards Provide Very High Security And Flexibility In Subscribers Management*", in Proceedings of the IEEE Transactions on Consumer Electronics, Vol. 36, No 3, August 1990

[CP93] V. Cordonnier, T. Peltier, "*Taxonomy Of Smartcards*", Publication LIFL, 1993

[CP89] U. Chouchena, P. Pons, "*Guide Rapide SQL*", Presse Pocket/ P.S.I, 1989

[Da87] C. Date, "*Introduction to Databases 2*", Ed. InterEditions, 1987

[Da89] C. Date, "*Introduction au standard SQL*", Ed. InterEditions IIA, 1989

[D76] D. Denning, "*A Lattice Model Of Secure Information Flow*", Comm ACM, v19 n5, pp 236-243, 1976

[DS79] D. Denning, Schwartz, "*The Trackers: A Threat For Statistical Database Security*", ACM Trans DB Sys, v4 n1, pp6 76-96, Mar. 1979

[D82] D. Denning, "*Cryptography And Data Security*", Addison-Wesley Publishing Company, pp 191-208, 1982

[D89] S. Duval et al., "*Use of Fingerprints As Identity Verification*", in Proceedings of the fifth IFIP International Conference on Computer Security, Gold Coast, Queensland, Australia, Ed. William J. Caelli, May 19-21, 1989

[D90] I. B. Damgard, "*A Design Principle For Hash Functions*", Advances in Cryptology, in CRYPTO '89 Proceedings, Berlin: Springer-Verlag, pp 416-427, 1990

[DJ93] R. Demolombe, A. Jones, "*Integrity Constraints Revisited*", 4th international workshop on the deductive approach to information systems and databases, Universitat Politecnica de Barcelona, Lloret Catalonia, Sept. 1993

[DOD85] US Department of Defense, "*Trusted Computing System Evaluation Criteria*", DOD5200.28-STD, December 1985

[DS83] D. Denning, J. Schlörer, "*Inference Controls For Statistical Data Bases*", Computer, v16 n7, pp 69-82, Jul. 1983

[DS91] J. A. Mc Dermid, Qi Shi, "*A Formal Model Of Security Dependency For Analysis And Testing Of Security Systems*", Proceedings of The Computer Security Foundations Workshop IV (Cat. No.91TH0383-0), pp 188-200, June 1991

[E87] S. Even, "*Secure Offline Electronic Fund Transfer Between Non-Trusting Parties*", IFIP, TC-11.6, SmartCard 2000, Laxenburg 87

[E89] G. Eizenberg, "*Mandatory Policy : Secure System Model*", in AFCET editor, European Workshop On Computer Security, Paris, 1989

[E93] J. E. Ettinger, "*Information Security*", Chapman & Hall, 1993

[E94] EyeDentify Inc., "*Reference Manual*", 1994

[E95] C. Eckert, "*Matching Security Policy To Application Needs*", in Proceedings of the 11th International Conference on Information, IFIP/SEC '95, pp 212 to 229, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[ER95] J. Eloff, W. de Ru, "*Reinforcing Password Authentication With Typing Biometrics*", in Proceedings of the 11th International Conference on Information, IFIP/SEC '95, pp 520 to 532, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[F86] R.C. Ferreira, "*On The Utilisation Of Smart Card Technology In High Security Applications; Perspectives For The Future*", IFIP/Sec 86, pp 487 to 503, Dec 1986

[FH86] L. Farinas, A. Herzig, "*Reasoning About Database Updates*", Workshop on foundations of deductive databases and logic programming", Jack Minker editor, 1986

[FW72] M. Farr, K. Wong, "*Security For Computer Systems*", The National Computer Centre Limited, 1972

[GPV94] A. Gamache, P. Paradinas, J-J. Vandewalle, "*Worldwide Smart Card Services*", in Proceedings of the first smart card research and advanced application conference, IFIP, Cardis 94, pp 141-148, Lille 1994

[HJ94] R. Hauser, P. Janson, "*Robust And Secure Password And Key Change Method*", ESORICS 94, number 875 in Lecture Notes in Computer Science, pp 108-122, Springer-Verlag, 1994

[HRU76] M. Harrison et al., "*Protection In Operating Systems*", Comm. ACM, v19 n8, pp 461-471, Aug. 1976

[HS94] C. O'Halloran, C. Sennett, "*Security Through Type Analysis*", ESORICS 94, number 875 in Lecture Notes in Computer Science, pp 75-89, Springer-Verlag, 1994

[HS75] T. H. Hinke, M. Schaeffer, "*Secure Data Management System*", Technical Report RADC-TR-75-266, System Development Corporation, 1975

[HT95] R. Holbein, S. Teufel, "*A Context Authentification Service For Role Based Access Controls In Distributed Systems*", in Proceedings of the 11th International Conference on Information, IFIP/SEC '95, pp 245 to 260, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[HW88] M. E. Haykin, R. B. Warnar, "*Smart Card Technology: New Methods For Computer Access Control*", Security Technology Group, Institute for Computer Science and Technology, NIST, Gaithersburg, MD, USA, 1988

[G89] L.C. Guillou, "*La Normalisation Internationale Des Cartes A Micro-Circuit A Contacts*", Arts et Manufactures, La Revue Des Centraliens, No 409 - Juin-Juillet-Août 1989. pp 23-28, 1989

[GD72] G. Graham, D. Denning, "*Protection. Principles And Practice*", in Proceedings Spring Joint Comp. Conference, pp417-429, 1972

[GG92] E. Gordons, G. Grimonprez, "*A Card As An Element Of Distributed Database*", IFIP, WG8.4, Workshop, Ottawa 92

[GM82] J. A. Goguen, J. Meseguer, "*Security Policies And Security Models*", in Proceedings of the 1982 IEEE Symposium on Security And Privacy, pp11 to 20, 1982

[GRL88] F. Guez, C. Robert, A. Lauret, "*Les Cartes A Microcircuit*", Ed. Masson, 1988

[GS90] R. Ganne, B. Salomon, "*La Carte A Mémoire*", Ed. Eyrolles, 1990

[GUQ92] L.C. Guillou, M. Ugon, J.J. Quisquater, "*The Smart Card: A Standardized Security Device Dedicated To Public Cryptology*", Gustavus J. Simmons: Contemporary Cryptology - The Science of Information Integrity; IEEE Press, Hoes Lane 1992, 561-613

[Ge90] Gemplus, "*MCOS Reference Manual*", 1990

[Ge94] Gemplus, "*MPCOS Reference Manual*", version 1.0, Sept. 1994

[Gr90] R. Graubart, "*A Integration Of Security Considerations With DBMS Development*", IFIP WG 11.3 North-Holland, pp 167 to 189, 1990

[GM82] J. Goguen, J. Meseguer, "*Security Policies And Security Models*", Proceedings 1982 IEEE Symposium on Security and Privacy, Oakland, 1982

[H80] B. Hebbard, "*A Penetration Analysis Of The Michigan Terminal System*", Operating Systems Review, Vol. 14, pp 07-20, Jan. 1980

[ISO1] Normes internationales ISO/IEC, 7816-1, "*Caractéristiques Physiques*"

[ISO2] Normes internationales ISO/IEC, 7816-2, "*Dimensions et emplacements des contacts*"

[ISO4] Normes internationales ISO/IEC, 7816-4, "*Commandes inter-industrielles pou l'échange*"

[IT91] Information Technology Security Evaluation Criteria (ITSEC), "*Provisional Harmonised Criteria*", June 1991

[Ja88] J. Jacob, "*Security Specifications*", Proceedings of the 1988 IEEE Symposium on Security and Privacy, pp 14-23, IEEE Comput. Soc. Press, Washington, DC, USA, 1988

[Ja89] R.B. Jack (chairman), "*Banking Services: Law And Practice Report By The Review Committee*", HMSO, London, 1989

[Jo92] H. L. Johnson, "*Integrity And Assurance Of Service Protection In A Large Multipurpose Critical System*", in Proceedings of the 15th National Computer Security Conference, Baltimore, 1992

[JS90] S. Jajodia, R. S. Sandhu, "*Integrity Mechanisms In Database Management Systems*", Proceedings of the 13 rd NIST-NCSC, National Computer Security Conference, Washington, Oct. 1990

[JO89] D. L. Jobusch, A. E. Oldehoeft, "*A Survey Of Password Mechanisms : Weaknesses And Potential Improvements. Part 1 And Part 2*", Computers & Security, 8 (1989), pp 587 to 604 and pp 675 to 689 respectively, 1989

[K95] W. Kühnhauser, "*On Paradigms For Security Policies In Multipolicy Environments*", in Proceedings of the 11th International Conference on Information, IFIP/SEC '95, pp 386 tp 400, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[La74] B. Lampson, "*Protection*", in Proceedings of the 5th Princeton Symp. in Operating Systems Review, v8 n1, pp18-24, 1974

[La95] B. Lau, "*Framework For Access Control Models*", in Proceedings of the 11th International Conference on Information, IFIP/SEC '95, pp 469 to 491, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[Le90] J. Mc Lean, "*Security Models And Information Flow*", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, May 1990

[Le95] A. Lewcock, "*The Development Of A Dynamic Signature Verification System By AEA Technology*", Business Development Manager, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 163-172, 1995

[Lee95] P. Lee, "*Smart Card Operating System*", President of Applied System Technology, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 263-266, 1995

[Lo93] A. S. Lovering, "*Issues In Using Formal Methods For Specifying Security Systems*", NPL report DITC 224/93, National Physical Laboratory, Teddington, Oct. 1993

[LM90] J. Levine, T. Mason, "*Lex & Yacc*", chapters 2 and 3, O'Reilly & Associates, Inc, 1990

[LP84] F. Lamarche, R. Plamodon, "*Segmentation And Feature Extraction Of Handwritten Signature Patterns*", in Proceedings of the Seventh International Conference on Pattern Recognition (Montreal, Canada, July 30- August 2, 1984), IEEE Publ. 84 CH2046-1, pp756-759, 1984

[M94] B. L. Miller, "*Biometric Identification: The Power To Protect People, Places And Privacy*", in Proceedings of the Cardtech/Securtech '94 International Conference, Arlington, Virginia, USA, April 10-13, pp 193-201, 1994

[MT79] R. Morris, K. Thompson, "*Password Security: A Case History*", Bell Laboratory, Communication of th ACM, vol. 22, pp 595 to 601, nov. 1979

[MP91] C. Macon, A. Pillot, "*Téléinformatique Et Systèmes D'exploitation*", Tome 2, Nathan, ch 6-4, pp 243-244, 1991

[NBS80] National Bureau of Standard, DES modes of operation, Federal Information Processing Standard. Ed. US Department of Commerce, FIPBS Pub 46, Washington, 1980

[Pa91] D. B. Parker, "*Restating The Foundation Of Information Security*", in Proceedings of the 14th Natl. Comp. Sec. Conf., 1991

[Pa95] D. B. Parker, "*A New FrameWork For Information Security To Avoid Information Anarchy*", in Proceedings of the 11th International Conference on Information, IFIP/

SEC '95, pp 135-144, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[Pe94] P. Peyret, "*RISC-Based, Next-Generation Smart Card Microcontroller Chips*", in Proceedings of the Cardtech/Securtech '94 International Conference, Arlington, Virginia, USA, April 10-13, pp 9-36, 1994

[Pe95-1] T. Peltier, "*Operating Systems For Blank Cards*", in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 107-116, 1995

[Pe95-2] T. Peltier, "*La Carte Blanche : Système D'exploitation Pour Objets Nomades*", Thèse de doctorat en informatique, Laboratoire d'informatique fondamentale de Lille, 1995

[Pf89] Ch. P. Pfleeger, "*Security In Computing*", Chapters 1,6 and 7, Prentice Hall International Editions, 1989

[Ph90] Philips Telecommunicatie en Data, "*Smart Card TB100*", 1990

[Pi95] O. Pieper, "*Advances In Fingerprint Image Capture*", President of Identificator Technology, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 195-200, 1995

[Po79 G. Popek et al., "*UCLA Secure Unix*", AFIPS Proceedings NCC, pp355-364, 1979

[PP88] R. Plamodon, M. Parizeau, "*Signature Verification From Position, Velocity And Acceleration Signals: A Comparative Study*", in Proceedings of the Ninth International Conference on Pattern Recognition (Rome, Italy, November 14-17, 1988), Computer Society Press, Washington, pp260-265, 1988

[PP91] J.M. Place, T. Peltier, "*Les Cartes A Microprocesseur, TP D'application*", 1991, Publication LIFL

[PT94] J.M. Place, P. Trane, "*A Security Language For The Card*", in Proceedings of the first smart card research and advanced application conference, IFIP, Cardis 94, pp 33-47, Lille 1994

[PPT95] T. Peltier, J.M. Place, P. Trane, "*Secured Cooperation Of Partners And Applications In The Blank Card*", in proceedings of the GMD-SmartCard Workshop, Ed. Struif, Darmstadt, 31 January - 01 February 1995

[R91] H. P. Reiss, "*Modeling Security In Distributed Systems*", Computer Security and Information Integrity, Proceedings of the sixth international conference on computer security in our changing world, pp 371-381, Netherlands1991

[RR83] J. Rushby, B. Randell, "*A Distributed Secure System*", in Computer, v16 n17, pp55-67, July 1983

[RSA78] R. L. Rivest, A. Shamir, A. Adleman, "*A Method For Obtening Digital Signature And Public Key Crypto-System*", Ed. Communication of the ACM, Feb. 1978

[Sa91] R. S. Sandhu, "*On Four Definitions Of Data Integrity*", Proceedings of IFIP WG 11-3, 1991

[Sb89] I. Schaumuller-Bichl, "*Card Security : An Overview*", in Proceedings of the Second International Smart Card 2000 Conference, Amsterdam, The Netherlands, pp 19-27, 4-6 October 1989

[Sc72] M. Schroder and al., "*A Hardware Architecture For Implementing Protection Rings*", Comm ACM, v15 n3, pp 157-170, Mar. 72

[Sc89] P. Schnabel, "*A New High-Security Multi-Application Smart Card, Jointly Developed By BULL And Philips*", in Proceedings of the Second International Smart Card 2000 Conference, Amsterdam, The Netherlands, pp 9-15, 4-6 October 1989

[Sc91] C. P. Schnorr, "*Efficient Signature Generation for Smart Cards*", Journal of Cryptology, vol. 4, No 3, pp 161-174, 1991

[Sc93-1] B. Schneier, "*Applied Cryptography*", Ed. John Wiley and Sons, Inc. , 1993

[Sc93-2] B. Schneier, "*Digital Signatures*", in BYTE, pp 309-312, November 1993

[Se95] S. Seidman, "*Introduction to Smart Card Technology And Applications*", Editor & Publisher, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 1-16, 1995

[Si87] W. Sibert et al., "*Unix And B2 : Are They Compatible?*", in Proceedings 1987 NBS/ NCSC Computer Security Conference, pp 142-149, 1987

[Sn81] L. Snyder, "*Formal Models Of Capability-Based Protection Systems*", IEEE Trans-Comput, pp 172-181, Mar. 1981

[Sp87] J. C. Spender, "*Identifying Computer Users With Authentication Devices*", Computer & Security, 6 (1987), pp385 to 395, 1987

[Sv87] J. Svigals, "*Smart Cards : The New Bank Cards*", Revised Edition by Macmillan Publishing Company, A Division of Macmillan Inc., chapters 5 and 11, 1987

[SHS92] Federal Information Processing Standards Publication, "*Secure Hash Standard*", Draft, National Institute of Standards and Technology, August 19, 1992

[SS75] J. H. Saltzter, M. D. Schroeder, "*The Protection Of Information In Computer Systems*", Proc. IEEE, vol. 63, pp1278-1308, Sept. 1975

[SUV93] H. Saiedian, E. A. Unger, R. B. Vaughn Jr, "*A Survey Of Security Issues In Office Computation And The Application Of Secure Computing Models To Office Systems*, Computer & Security", vol. 12, pp 79-97, 1993

[Ta84] A. Tannenbaum, "*Structured Computer Organization*", chapter 6, InterEditions, 1984.

[Ta89] A. Tannenbaum, "*Operating Systems. Design And Implementation*", Prentice Hall, Inc., Englewood Cliffs, N.J. 07632, chapter 5, 1989

[Te95] W. Tetschner, "*Voice Technology Applications In Identification*", President of Voice Information Associates, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 173-178, 1995

[Th90] M.B. Thuraisingham, "*Security In Object-Oriented Database System*" , 1990

[Tr92] P. Trane, "*Protection Et Système D'information*", Mémoire de DEA, Publication LIFL, 1992

[TW89] P. Terry, S. Wiseman, "*A New Security Policy Model*", Proceedings of the IEEE Symposium on Security and Privacy, Oakland CA, May 1989

[Ul82] J. Ullman, "*Principles of Database Systems*", Computer Science Press, 1982

[Wa95] A. Warman, "*Developing Policies, Procedures And Information Security Systems*", in Proceedings of the 11th International Conference on Information, IFIP/SEC '95, pp 427 to 438, Ed. Jan HP Eloff & SH von Solms, CapeTown, South Africa, 1995

[Wi89] B. A. Wichmann, "*Insecurity In The ADA Programming Language*", NPL report DITC 137/89, National Physical Laboratory, Teddington 1989

[Wi92] S. Wiseman, "*On The Problem Of Security In Databases*", in Spooner and C. Landwehr editors, Database Security 3: Status and Prospects. North-Holland, results of the IFIP WG 11.3 Workshop on Database Security, 1990

[WMW89] R. Wieringa, J. J. Meyer, H. Weigand, "*Specifying Dynamic And Deontic Integrity Constraints*", Data And Knowledge Engineering 4, North-Holland, 1989

[Ya90] K. Yazdanian, "*Relational Database Granularity*", AFCET, Esorics, pp 15 to 19, 1990

[Ya95] W. Yang, "*A Real-Time Face Recognition System Using Machine Vision Techniques*", Havard University, in Proceedings of the Cardtech/Securtech '95 International Conference, Washington DC, USA, April 10-13, pp 179-194, 1995