# Un modèle de contrôle d'accès générique et sa réalisation dans la mémoire virtuelle répartie unique Arias

Christian Damsgaard Jensen

*UNIVERSITÉ JOSEPH FOURIER–GRENOBLE I*
*SCIENCES & GEOGRAPHIE*

*N°*

# THÈSE

pour obtenir le grade de

## DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

**Discipline : INFORMATIQUE**

Présentée et soutenue publiquement

par

Christian Damsgaard JENSEN

Le 29 octobre 1999

# Un modèle de contrôle d'accès générique et sa réalisation dans la mémoire virtuelle répartie unique Arias

Directeur de thèse :

Sacha KRAKOWIAK

**COMPOSITION DU JURY :**

| | |
|---|---|
| M. Roland Balter | Président |
| M. Claude Bétourné | Rapporteur |
| M. Bertil Folliot | Rapporteur |
| M. Daniel Hagimont | Co–directeur |

Thèse préparée au sein du laboratoire Sirac de l'INPG, l'INRIA Rhône–Alpes et l'UJF

2

# Remerciements

Je tiens à remercier l'ensemble des personnes qui, par leurs conseils, leurs encouragements ou leur aide, ont contribué à l'aboutissement de ce travail :

Sacha Krakowiak, Professeur à l'Université Joseph Fourier, qui a encadré ce travail ;

Roland Balter, Professeur à l'Université Joseph Fourier et Directeur du projet SIRAC, qui m'a fait l'honneur d'accepter de présider le jury ;.

Claude Bétourné, Professeur à l'Université Paul Sabatier, et Bertil Folliot, Professeur à l'Universté Pierre et Marie Curie, qui ont accepté d'être rapporteurs de ma thèse, et par leurs critiques constructives m'ont permis de l'améliorer ;

Daniel Hagimont, Chargé de recherche à l'Institut National de Recherche en Informatique et Automatique, qui a co–dirigé cette thèse, pour l'inspiration, ses nombreux conseils sur la recherche et nos nombreuses discussions nocturnes sur le balcon. L'occasion de travailler avec Daniel et sous la direction de Sacha, m'a encouragé à poursuivre mes études dans un pays étranger et dans une langue que je maîtrise à peine ;

L'équipe qui a mis en place la plate–forme Arias : Alain, Elizabeth, Frederic, Jay, Maryannick, Olivier, Pascal et Thierry et l'ensemble des personnes du projet SIRAC pour la chaleureuse ambiance qui a régnée dans nos locaux. Je tiens en particulier à remercier mes voisins de bureau Alain, Ibaa, Maryannick et Olivier (Olaf) pour leur patience, leurs conseils dans les matières scientifiques et métaphysiques et les nombreux services qu'ils m'ont rendus ;

Enfin, et non les moindres, ma femme Lena et ma fille Michala pour leur patience pendant la préparation de cette thèse.

4

à Lena et Michala

6

# Table des matières

# Table des figures

# Liste des tableaux

# Chapitre 1

# Introduction

## 1.1 Contexte du travail

Le travail présenté dans ce document a été effectué dans le cadre du projet Systèmes Informatiques Répartis pour Applications Coopératives (SIRAC[1]) [Balter95]. L'objectif principal du projet est la conception et la réalisation des outils et supports systèmes pour la construction des applications réparties.
En particulier, l'étude présentée ici se situe dans le cadre de la conception et de la réalisation d'une plate–forme offrant un service de gestion des données partagées, complexes, structurées et à longue durée de vie.

## 1.2 Motivations

Les systèmes répartis permettent aux utilisateurs de partager l'ensemble des ressources du système, ce qui nécessite une gestion globale de ces ressources. Pendant les années 1980, la recherche dans ce domaine s'est concentrée sur la gestion globale des ressources chères comme les disques durs (les systèmes NFS [Sandberg85, Sun89] ou AFS [Satyanarayanan85, Spector89]) ou l'unité centrale (la répartition de charge a été étudiée dans le cadre des systèmes Condor [Litzkow87], Sprite [Douglis91] et Utopia [Zhou91] ou par simulation [Ferrari86, Eager88, Kunz91]).
L'arrivée des réseaux rapides et des architectures 64 bits a provoqué un intérêt pour la gestion globale de la mémoire virtuelle. Les réseaux rapides, comme ATM [Dutton95], Myrinet [Boden95] et Gigabit Ethernet [IEEE98] ou les interfaces réseaux qui permettent de consulter la mémoire à distance sans interrompre le processeur comme Memory Channel [Fillo97] et SCI [IEEE92, Omang98], réduisent le temps de communication entre machines et rendent ainsi la pagination à distance faisable. Avec ces technologies, le temps d'accès à une page en mémoire d'une machine distante devient plus économique que l'accès au disque local [Feeley95, Koch98]. Outre cette augmentation de vitesse des réseaux, la gestion d'un espace virtuel unique dans un système réparti est rendue possible par les architectures à adressage étendu (64 bits) où les adresses virtuelles ne sont plus une ressource limitée qui doit être économisée. Un calcul simple fait par Bartoli et al. [Bartoli93] montre que

[1] SIRAC est un projet commun entre l'Université Joseph Fourier, l'Institut National Polytechnique de Grenoble et l'Institut National de Recherche en Informatique et Automatique

dans le cas où dix machines allouent 10 Go/minute chacune, un espace d'adressage à 64 bits ne s'épuise pas avant 300 ans. Ce calcul indique qu'un espace d'adressage à 64 bits suffira largement pour un système réparti avec des besoins modérés d'allocation de mémoire (un système avec peu de machines ou un faible taux de allocation et fragmentation de la mémoire). Il ne dit cependant rien sur l'adéquation de cette technologie pour un système à très grande échelle. Une simulation basée sur les traces d'allocation de mémoire des applications réelles [Kotz94] suggère que les architectures à 64 bits suffiront si le système d'exploitation arrive à limiter la fragmentation et à récupérer l'espace alloué aux structures d'exécution et aux variables temporaires.
Ces deux évolutions techniques évoquées ci–dessus (arrivée des réseaux rapides et des processeurs à espace d'adressage étendu) a donné naissance à une nouvelle génération de systèmes répartis : les mémoires virtuelles réparties uniques.
La conception d'un système réparti fondé sur une mémoire virtuelle répartie unique (MVRU) et persistante a été étudiée dans le cadre de systèmes comme Angel [Murray93a, Murray93b], Mungi [Heiser93, Heiser94, Heiser98] et Opal [Koldinger92, Chase92, Chase93, Chase95]. Les aspects de désignation et d'accessibilité constituent une différence importante entre ce type de système et les systèmes d'exploitation traditionnels. Dans un système traditionnel, l'espace d'adressage est propre au processus et le droit d'accès à un objet protégé a été vérifié avant que l'objet puisse être désigné en mémoire principale. Autrement dit, la capacité de désigner un objet en mémoire principale implique le droit d'accéder à l'objet. Dans une MVRU, tous les objets sont désignés par une adresse virtuelle unique dans tous les processus et sur toutes les machines. La capacité de désigner un objet par adresse virtuelle ne peut plus avoir la même implication ; il faut donc séparer l'adressage et la gestion des droits d'accès. Chaque processus doit avoir le droit d'accéder à certaines régions de son espace d'adressage (celles qui correspondent au code et aux données du processus), mais non à celles qui sont utilisées par les autres processus. Le système donne ainsi à chaque processus une fenêtre sur la mémoire virtuelle globale ; ces fenêtres peuvent être différentes d'un processus à l'autre. Ceci implique un contrôle d'accès au niveau de la mémoire principale. L'étude d'un mécanisme de contrôle d'accès qui réalise une telle séparation entre l'adressage et le droit d'accès est le sujet de cette thèse.

## 1.3 Objectifs

Le support expérimental de ce travail a été le système Arias, un système de gestion de données persistantes et réparties, réalisé comme une mémoire virtuelle répartie unique sur un réseau de machines.
L'objectif de notre travail est de fournir un mécanisme de base qui permet la réalisation de plusieurs politiques de protection sur Arias. Ce mécanisme doit séparer la définition de la protection et le code d'application pour les raisons suivantes :

**Facilité de programmation.** Le programmeur s'occupe uniquement de l'algorithmique de l'application. La spécification de la protection est faite par un administrateur système ou un responsable de la sécurité.

**Réutilisation des composants logiciels.** Les mêmes composants logiciels peuvent être utilisés dans des contextes et des applications différents.

**Évolution de la protection.** La spécification de la protection d'une application peut changer pendant la durée de vie de l'application, sans recompilation ou ré-édition de liens.

**Protection de logiciels conçus ailleurs.** Les logiciels conçus ailleurs peuvent être encapsulés afin de protéger l'environnement local.

**Généricité du mécanisme.** La conception du mécanisme doit être générique pour permettre la réalisation des différentes politiques de protection requises par les applications. La généricité est un objectif général d'Arias.

## 1.4 Apports du travail

Les apports de cette thèse sont les suivants :

1. Une étude des modèles de contrôle d'accès et leurs réalisations dans les systèmes répartis existants.
2. La définition d'une taxinomie pour faciliter la description des systèmes à capacités répartis.
3. La proposition d'un modèle étendu du modèle à capacités cachées.
4. Une analyse (informelle) de ce modèle, pour vérifier que ce modèle permet la réalisation des politiques de protection existantes.
5. La réalisation de ce modèle dans la mémoire virtuelle répartie unique Arias.
6. Une évaluation du modèle proposé.

## 1.5 Organisation du document

Cette thèse est organisée en trois parties principales : la première partie (chapitres 2, 3 et 4) passe en revue les problèmes liés à la sécurité dans les systèmes informatisés en général et les systèmes répartis en particulier, ainsi que les modèles et mécanismes proposés pour résoudre ces problèmes. La deuxième partie (chapitres 5 et 6 décrit un modèle de contrôle d'accès proposé pour les logiciels coopératifs et sa réalisation dans un système réparti basé sur une mémoire virtuelle répartie unique. La troisième partie (chapitres 8 et 9) présente une première évaluation de ce modèle et les conclusions générales de cette thèse.

# Chapitre 2

# Sécurité dans les systèmes répartis

La sécurité des systèmes informatiques évoque normalement l'image d'un jeune pirate très doué en informatique qui, tout seul dans sa chambre, essaie de pénétrer les mécanismes de sécurité d'un système distant pour voler ou détruire des informations stockées dans le système. Cette image est largement évoquée par les romans de "science fiction" qui introduisaient le "Cyberspace" et la "Matrice" comme métaphores pour l'Internet [Gibson84] et les films populaires de science fiction comme "Wargames" [Badham83] ou "Terminator 2" [Cameron91].

Deux exemples réels de ce type de pirates sont les espions capturés en Allemagne au milieu des années 1980[1] et le groupe de pirates "Global Hell (gH)" qui en mai 1999 forçait la Maison Blanche et la police fédérale américaine (le FBI) à fermer leur site Internet [Meeks99a, Meeks99b].

Les exploits de ces pirates sont largement diffusés par la presse. De ce fait, il est généralement admis que l'objectif de la sécurité est d'empêcher ces pirates de pénétrer le système, par exemple en installant des pare–feu [Soinne98, CheckPoint99, AXENT99].

En réalité, les pirates ne sont pas responsables de plus de 30% des pertes dues aux défaillances de la sécurité [Power99]. La plupart des défaillances de sécurité sont dues aux attaques venant de l'intérieur, par exemple par des employés insatisfaits. Les attaquants internes sont normalement plus difficiles à découvrir et plus difficiles à poursuivre en justice.

Dans ce chapitre, nous étudions la sécurité informatique et en particulier la sécurité dans les systèmes répartis. D'abord, nous donnons une introduction très générale, puis nous étudions ses objectifs et ses menaces. Enfin, nous voyons plus en détail les dispositifs possibles pour assurer la sécurité et l'évaluation de la sécurité d'un système.

## 2.1 Introduction à la sécurité informatique

La définition la plus générale de la sécurité informatique est d'assurer le bon fonctionnement d'un système informatique. Toute défaillance d'un système peut être attribuée à une faille dans la sécurité. Du point de vue d'un utilisateur qui n'arrive pas à travailler ou qui a perdu ses fichiers, le problème reste le même qu'il soit causé par un pirate ou par un défaut de sauvegarde.

[1] La découverte et la poursuite de ces pirates sont décrites dans "Le nid du coucou" par Clifford Stoll [Stoll89].

Le nombre de sujets qui doivent être étudiés pour assurer la sécurité d'un système informatique est très grand. La sécurité ne couvre pas uniquement le contrôle d'accès aux données, mais aussi un grand nombre d'aspects moins prestigieux comme par exemple : déchiqueter les documents sensibles avant les mettre à la poubelle, stocker les sauvegardes hors site pour prévenir les incendies, installer des onduleurs de courant électrique, construire des bâtiments anti–sismiques, concevoir des logiciels fiables (spécification formelle et vérification de la validité du code).
Dans le cadre de cette thèse nous nous intéressons principalement à la protection de l'information. Les problèmes de sécurité physique d'une entreprise et l'installation et la gestion des machines sur un site sont traités par des agence du gouvernement. Un exposé de ces problèmes et des recommandations pour les résoudre a été publié par Le Service Central de la Sécurité des Systèmes d'Information (SCSSI) [SCSSI91a, SCSSI91b, SCSSI93]. Une autre source d'information est le "IT–Grundschutzhandbuch" publié par le "Bundesamt für Sicherheit in der Informationstechnik" [BSI97] qui traite ces problèmes et les dispositifs adéquats pour les résoudre en détail.

### 2.1.1 Terminologie de la sécurité

La sécurité dans les systèmes informatiques possède maintenant sa propre terminologie bien établie.

**Limite du système et périmètre de sécurité.** Le système n'est normalement pas une entité bien définie, il comprend l'ensemble des ressources (ordinateurs et réseaux) qui sont à la disposition des utilisateurs ou des applications. La limite du système désigne l'ensemble des ressources sur lesquelles l'administrateur exige un minimum de contrôle. Celles qui sont fondamentales à la sécurité doivent être gérées uniquement par l'administrateur derrière le périmètre de sécurité. L'administrateur ne doit jamais abandonner le contrôle d'une ressource derrière ce périmètre de sécurité.

**Moniteur de référence.** Le moniteur de référence est un médiateur incontournable dans toutes les relations entre sujets et objets (cf. Protection). Le moniteur de référence est responsable de l'autorisation de l'accès à un objet par un sujet.

**Protection.** La protection[2] consiste à contrôler l'accès aux entités passives (les données ou les ressources consommables du système — réseau, temps sur l'unité centrale, etc.) par les entités actives (les utilisateurs ou les processus du système). Les entités passives sont normalement appelées les *objets* et les entités actives sont appelées les *sujets* ou les *principaux*.[3] Remarquons qu'un sujet peut aussi jouer le rôle d'un objet, par exemple quand un processus envoie un signal/message/événement à un autre processus.

**Sous–système de sécurité.** Les systèmes informatiques se composent normalement d'un certain nombre de sous–systèmes. Certains de ces systèmes n'apportent rien à la sécurité, par

[2]Dans la littérature de la sécurité, les termes autorisation, protection et contrôle d'accès sont presque synonymes, nous ne faisons pas de distinction entre ces termes ici.

[3]Nous ne faisons pas de distinction entre sujets et principaux dans cette thèse.

contre d'autres mettent en œuvre la sécurité directement ou offrent des services pour la création d'applications sûres. L'ensemble des sous–systèmes sur lesquels la sécurité d'une application sûre repose est nommé le sous–système de sécurité ("Trusted Computing Base" ou "TCB"). Le sous–système de sécurité est normalement un sous–ensemble du système d'exploitation.
La relation entre la limite du système, le périmètre de sécurité, le sous–système de sécurité et le moniteur de référence est illustrée dans la figure 2.1[4].

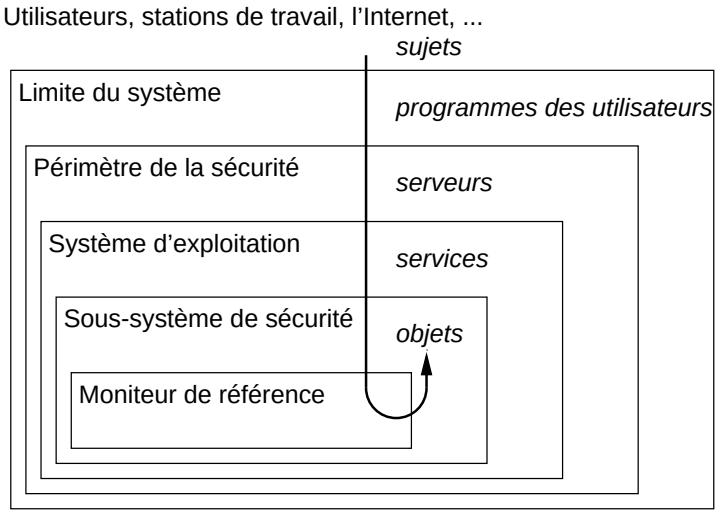



FIG. 2.1: Les périmètres de la sécurité

La figure montre comment l'accès aux objets par des sujets traverse les différents périmètres du système.
Le noyau d'un système sûr est le moniteur de référence qui contrôle l'accès à toutes les ressources du système. Le système d'exploitation fournit un ensemble de primitives qui permet de réaliser des services et les serveurs. Le sous–ensemble de ces primitives qui sont vitales pour la sécurité du système s'appelle le sous–système de sécurité. Les primitives mises en œuvre par le système d'exploitation permettent de réaliser les serveurs qui gèrent les objets. Le système d'exploitation et l'ensemble des serveurs vitaux pour la sécurité constitue la périmètre de la sécurité. Les entités derrière ce périmètre ne sont pas forcément sur la même machine, mais elles doivent impérativement être sous le contrôle de l'administrateur du système. La limite du système définit la frontière entre les entités manipulées directement par le système et le monde extérieur.

### 2.1.2 Objectifs de la sécurité

Normalement la protection est décrite dans les termes suivants :

> **La confidentialité** des informations. La confidentialité garantit que l'information dans le système est uniquement accessible aux utilisateurs autorisés. Des mécanismes doivent être présents garantissant cette confidentialité entre utilisateurs et entre utilisateurs et système.

[4]Cette figure est une adaptation pour les systèmes répartis d'une figure trouvée dans le livre "Building a Secure Computer System" par Morrie Gasser [Gasser88].

**Le bon usage** (“privacy”) des informations. Le bon usage garantit que les informations sont utilisées uniquement pour le but dans lequel elles sont données. Cette notion ressemble au secret professionnel des avocats ou médecins, où l’information obtenue ne peut pas être divulguée à quelqu’un qui n’a pas besoin de la connaître. La différence entre la confidentialité et le bon usage est que la première s’applique directement à l’information et l’autre s’applique à l’utilisation de l’information.

**L’authenticité** des informations. L’authenticité garantit qu’un sujet (utilisateur, processus, système, etc.) est celui qu’il prétend être et que les informations reçues de ce sujet sont identiques à celles fournies.

**L’intégrité** des données. L’intégrité évite la corruption ou la destruction des données traitées par le système. Il faut d’abord empêcher des utilisateurs malveillants d’accéder aux données non autorisées (assurer la confidentialité et l’isolation des informations) mais également assurer que les données sont accédées de façon cohérente (par exemple avec un accès transactionnel).

**La disponibilité** des ressources du système. La disponibilité est la garantie que les données et les services du système sont disponibles aux utilisateurs autorisés. Il faut pour cela empêcher un utilisateur malveillant d’arrêter ou de bloquer un service (“Denial of Service attacks”).

Chacun de ces objectifs est examiné plus en détail plus loin.

### 2.1.3 Menaces contre la sécurité

Afin de pouvoir analyser les dispositifs à mettre en œuvre pour assurer la sécurité, il faut d’abord analyser les menaces et les différents moyens d’attaquer un système informatique[5].

**Interception** des données. L’attaquant intercepte les informations secrètes sans rien modifier (cela s’appelle quelque fois l’écoute passive). Il a trois possibilités : lire les données stockées en mémoire secondaire (sur disque ou bande), lire les données transmises sur le réseau ou analyser le rayonnement électromagnétique des périphériques tels que l’écran ou le clavier.

**Modification** des données. L’attaquant supprime ou modifie les données stockées en mémoire secondaire ou transmises sur le réseau.

**Fabrication** des données. L’attaquant fabrique des données et les passe au système comme si elles étaient créées par quelqu’un d’autre. La fabrication menace principalement les applications qui communiquent à travers un réseau (cf. 2.1.4), mais elle menace aussi les applications qui tournent sur un seul site.

**Divulgation** des informations. L’attaquant peut avoir un complice (un utilisateur légitime) qui divulgue l’information. La prévention de la divulgation est un problème très difficile et correspond à la garantie du bon usage. Ce problème est connu dans la littérature sous le nom du problème de confinement (cf. 2.3).

**Déduction par inférence**. L’attaquant combine les informations obtenues par différents moyens (légitimes ou illégitimes) afin d’inférer de nouvelles informations. Par exemple si on ne connaît pas la formule $X$ du Coca Cola, mais que l’on connaît toutes les matières achetées par l’usine pour le produire, la composition de $X$ peut être inférée.

[5] Les menaces et les attaques listés ici sont un sous–ensemble de ceux identifiés par la SCSSI [SCSSI94].

**Mascarade** ou déguisement. L'attaquant (ou un programme qui travaille pour lui) essaie d'agir avec l'identité (et les droits d'accès) de quelqu'un autre. Le but peut être d'intercepter, de modifier ou de fabriquer des données stockées sur le site. Un cas de mascarade par un programme est généralement appelé un *cheval de Troie*[6].

Les différents paramètres de sécurité sont traités plus loin avec une analyse des menaces en termes des attaques décrites ci–dessus.

### 2.1.4 Particularités des systèmes répartis

Une particularité des systèmes répartis est qu'il n'existe pas toujours un sous–système de sécurité commun sur tous les sites. Un système réparti peut être structuré soit comme un ensemble de ressources gérées par un système d'exploitation réparti, soit comme un ensemble de systèmes individuels qui coopèrent pour supporter les applications réparties.
Dans les systèmes d'exploitation répartis, le sous–système de sécurité fait partie du système d'exploitation. Alors les informations sur les sujets et leurs droits d'accès aux objets sont répartis et accessibles sur tous les sites. Ceci a pour conséquence que le droit d'accès à un objet peut être vérifié localement, sans consulter le site qui stocke l'objet.
Les systèmes d'exploitation coopérants coordonnent les identités des sujets et des objets, mais ils gèrent chacun leurs propres informations concernant les droits d'accès. De ce fait, les objets sont accessibles partout, mais la vérification du droit d'accès se fait toujours sur le site où l'objet se trouve physiquement.
La différence entre ces deux types de systèmes est illustrée sur la figure 2.2.

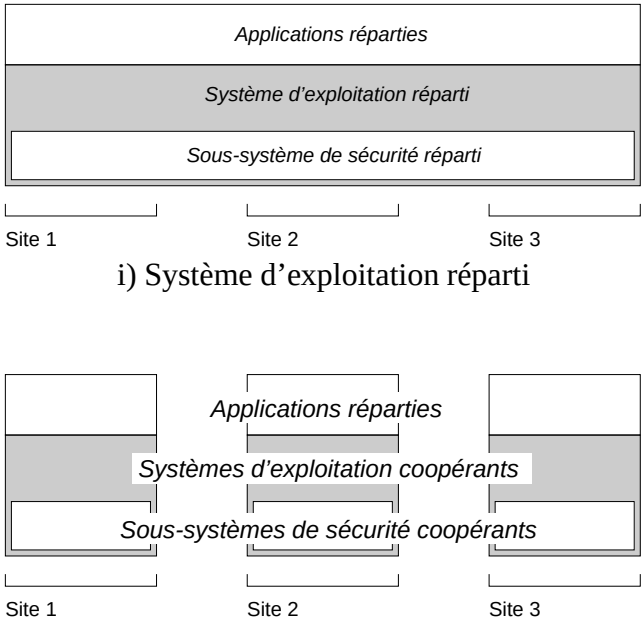



FIG. 2.2: La différence entre un système réparti et des systèmes coopérants

[6]Un cheval de Troie peut également être un programme qui parait innocent, mais introduit un problème de sécurité dans le système.

La figure montre les relations entre les applications, les systèmes d'exploitation et les sous–systèmes de sécurité dans les systèmes répartis 2.2 i) et les systèmes coopérants 2.2 ii). Pour pouvoir réaliser un système d'exploitation réparti sûr, il faut que l'ensemble des sites puisse être considéré comme une unité blindée, c'est–à–dire que l'accès direct au réseau soit impossible et que de nouvelles machines ne puissent pas être introduites arbitrairement dans le système. L'avantage d'un tel système est la possibilité de mettre en vigueur une politique de protection globale.
Dans les systèmes coopérants la décision d'autoriser l'accès à un objet par un sujet est prise localement sur la machine qui gère l'objet. Alors ces décisions peuvent différer selon les sites. Par exemple l'utilisateur `root` sous Unix peut modifier tous les fichiers stockés sur les disques locaux, mais il n'a pas les droits spéciaux pour les disques d'un autre site.
Un autre problème commun dans les deux types de systèmes répartis est le contrôle d'accès au réseau. Si l'attaquant arrive à avoir un accès direct au réseau, il peut lire, modifier et fabriquer les informations qui passent sur le réseau. Alors, il faut toujours s'assurer que l'interlocuteur distant est bien celui auquel on pense. Nous reviendrons aux solutions à ce problème dans les sections 2.8.1 et 2.8.3.

## 2.2 Confidentialité

La garantie de la confidentialité ("secrecy") des informations stockées sur un système informatique est souvent considérée comme la plus importante qualité d'un système sûr. Les utilisateurs du système doivent avoir l'assurance que le système ne divulgue pas leurs informations sensibles (secrets industriels, dossiers médicaux, secrets d'État) à des personnes non autorisées.
La confidentialité est la première préoccupation pour une installation militaire. Une grande partie de la recherche sur la mise en œuvre de la confidentialité a été financée par des projets militaires et est destinée à des installations militaires. C'est pourquoi cette recherche utilise souvent un modèle d'accès aux informations semblable à celui que l'on trouve dans le monde militaire.
Afin d'assurer la confidentialité, le système doit disposer d'un mécanisme qui permet d'établir l'identité de tous les agents.

## 2.3 Confinement

Le confinement consiste à éviter la divulgation volontaire d'information. Ce problème est donc complémentaire à la confidentialité pour assurer le bon usage des informations.
Le problème à été défini par Lampson [Lampson73] comme "le problème de confiner un programme pendant son exécution pour qu'il arrive uniquement à transmettre de l'information au programme qui l'a lancé". Alors le confinement garantit qu'un sujet n'arrive pas à divulguer le contenu des objets auxquels il a accès à quelqu'un qui n'a pas le droit d'y accéder.
Il existe trois canaux possibles pour la divulgation de l'information. Ces canaux sont :

- *Les canaux de stockage* qui permettent de transférer de l'information à travers des objets qui sont écrits en toute légalité d'un coté et lus en toute légalité de l'autre, par exemple un fichier ;

- *Les canaux de communication légitimes* qui permettent à un processus d'envoyer un message à un autre processus en utilisant les mécanismes de communication entre processus ("IPC"), par exemple un appel RPC ;
- *Les canaux cachés* qui permettent la communication entre processus en dehors de mécanismes normaux.

Si on arrive à montrer qu'un système n'a pas de canaux cachés, c'est qu'il n'est pas vulnérable aux chevaux de Troie. Nous allons examiner les canaux cachés plus en détail dans la suite.

### 2.3.1 Canaux de stockage

La divulgation à travers des objets persistants nécessite qu'il y ait une intersection non vide entre l'ensemble d'objets modifiables par celui qui divulgue et les objets lisibles par celui qui souhaite profiter de la divulgation.
Ceci s'applique aussi aux autres objets qui sont visibles à travers le système de persistance par exemple les fichiers temporaires (dans `/tmp`) dans un système Unix. Alors il ne s'agit pas uniquement de contrôler l'accès aux objets existants, mais il faut aussi contrôler les objets persistants créés par le programme confiné. Les objets créés par un programme confiné ne doivent pas être visibles à l'extérieur de la zone de confinement.
Un grand nombre de mécanismes différents de confinement pour les canaux de stockage a été proposé dans la littérature [Wahbe93, Goldberg96, Jensen98d].

### 2.3.2 Canaux de communication légitimes

Pour éviter la divulgation par les canaux de communication légitimes, le système doit vérifier tout message envoyé par un programme confiné. La politique du bac à sable ("sandbox") réalisée par le gestionnaire de sécurité par défaut dans la machine virtuelle de Java ("Java Virtual Machine" ou "JVM") [McGraw99] réalise une telle politique. Les applets (petits programmes mobiles écrits en Java) ne sont autorisées qu'à ouvrir une connexion sur le réseau avec les serveurs depuis lesquels elles ont été téléchargées. L'idée est d'éviter qu'un cheval de Troie stocké sur un serveur respectable n'arrive à envoyer de l'information à un pirate en–dehors. La politique ne garantit pas le confinement total car l'applet arrive toujours à communiquer avec le serveur depuis lequel elle a été téléchargée, mais si on n'a pas confiance dans ce serveur, il valait mieux ne pas télécharger l'applet.

### 2.3.3 Canaux Cachés

Il existe trois types de canaux cachés différents : *canaux temporels, canaux d'inférence* et *canaux de fabrication* [SCSSI94] .

#### Canaux temporels

Les canaux temporels permettent à un processus de transmettre un message à un autre processus en modulant la consommation de ses ressources systèmes afin que les variations des temps de réponse puissent être observées. Un exemple d'un canal temporel est un processus qui s'arrange pour que la charge du système soit élevée, par exemple supérieur à 15, quand il

souhaite d'envoyer le bit `1` et inférieure quand il souhaite d'envoyer le bit `0`. Il peut ainsi communiquer une information à un observateur extérieur.

#### Canaux d'inférence

Les canaux d'inférence[7] permettent à un processus de déduire de l'information à laquelle il n'a pas normalement accès.
Le message divulgué se compose de l'intersection de l'ensemble des informations non classées envoyées entre les deux interlocuteurs. Alors il ne s'agit plus uniquement de contrôler le flux des informations confidentielles, mais tous les flux d'informations peuvent être utilisés comme canaux cachés.

#### Canaux de fabrication

Les canaux dits de "fabrication" permettent de créer de l'information en formant des agrégats qui ne peuvent être obtenus directement.

#### Solution au confinement

Lampson propose une solution au problème du confinement. Il faut d'abord que le programme confiné soit sans mémoire c'est–à–dire qu'il ne laisse pas de traces entre plusieurs exécutions. Dans un système sans mémoire le confinement peut être garantit par la règle suivante :

> **Isolation totale** du programme. Le programme confiné ne doit pas pouvoir appeler un autre programme.

L'isolation totale n'est pas toujours pratique ; la condition peut être allégée si le confinement est transitif. Alors la règle de confinement allégé peut être définie :

> **Transitivité de l'isolation** du programme. Si un programme confiné appelle un autre programme dans lequel on ne peut pas avoir confiance, le programme appelé doit être également confiné.

Pour que cette règle suffise à éviter la divulgation, il faut bien évidemment d'abord maîtriser les canaux de stockage et les canaux de communication légitime, par exemple par *filtrage* ("masking") de toutes les entrées/sorties du programme confiné. Pour éviter les canaux cachés, il faut que le système mette en place des mesures coercitives ("enforcement") afin d'assurer que le comportement du programme correspond à une spécification donnée. Par exemple le système peut ralentir le programme ou générer de faux accès au disque dur pour brouiller le codage d'un message transmis par des accès au disque.
Le coût du confinement garanti peut être cher. Alors quelques fois, on se contente de limiter la bande passante d'un canal caché.

## 2.4 Authenticité

L'authenticité consiste à garantir que les données livrées à un utilisateur sont celles qu'il demandait, c'est–à–dire que l'expéditeur est bien celui qu'il prétend être et que les données ne

[7] Les canaux d'inférence sont quelque fois appelés les canaux de raisonnement.

sont pas modifiées en route entre l'expéditeur (le service de stockage secondaire ou un système distant) et le programme lancé par l'utilisateur.
Dans un système centralisé, l'existence d'un moniteur de référence et la correction du sous–système de sécurité peuvent garantir qu'il n'y a pas moyen de modifier les données entre deux entités du système.
Dans la plupart des systèmes répartis, le chemin entre le stockage secondaire et les applications et entre deux processus communicants passe par plusieurs sous–systèmes. En passant par le réseau, le message risque d'être modifié par un attaquant. Alors il faut un mécanisme qui permet d'envoyer les messages de façon à ce que :

1. l'expéditeur ne puisse pas nier l'envoi du message,
2. le récepteur puisse vérifier l'identité de l'expéditeur,
3. le récepteur ne puisse pas fabriquer le message lui même.

Ces objectifs peuvent être atteints par le chiffrement asymétrique (cf. 2.8.3) ou par les signatures digitales [Rivest78, Schnorr91, NBS94].

## 2.5 Intégrité

Une définition de l'intégrité est "la capacité du système informatique à empêcher la corruption des informations par des fautes accidentelles ou intentionnelles" [Nicomette96]. Les cas de fautes accidentelles sont normalement résolus par les transactions [Lomet77, Lampson81]. En ce qui concerne les fautes intentionnelles, les attaques visent à modifier ou supprimer des informations dans le système ou à introduire de fausses informations, normalement pour un profit personnel ou professionnel (par exemple les fraudes informatiques).
Le modèle d'intégrité le plus connu est probablement celui proposé par Biba [Biba77]. Ce modèle introduit une notation formelle (mathématique) pour décrire les restrictions nécessaires à imposer sur les modifications de données pour garder leur intégrité.
Le modèle sépare les sujets et les objets en différentes classes d'intégrité ("integrity levels") — une classification d'intégrité s'applique aux sujets et aux objets. Ensuite il définit une relation de domination, celle–ci constitue un ordre partiel qui est défini sur toutes les classes d'intégrité. Pour garantir l'intégrité des données, ce modèle définit deux conditions[8] :

1. Un sujet peut modifier un objet uniquement si la classe d'intégrité du sujet domine celle de l'objet [intégrité simple]
2. Un sujet peut lire un objet uniquement si la classe d'intégrité de l'objet domine la classe d'intégrité du sujet [intégrité confinée]

Ces conditions garantissent qu'un objet très intègre ("high integrity object") ne peut pas être contaminé par des informations obtenues par un objet moins intègre ("low integrity object"). Ces conditions sont illustrées dans la figure 2.3.

[8] Ces conditions ressemblent aux conditions de la sécurité multi–niveau proposées par Bell & LaPadula (cf. 3.4) mais la confidentialité et l'intégrité sont en réalité orthogonales et les classes de confidentialité et les classes d'intégrité sont différentes.

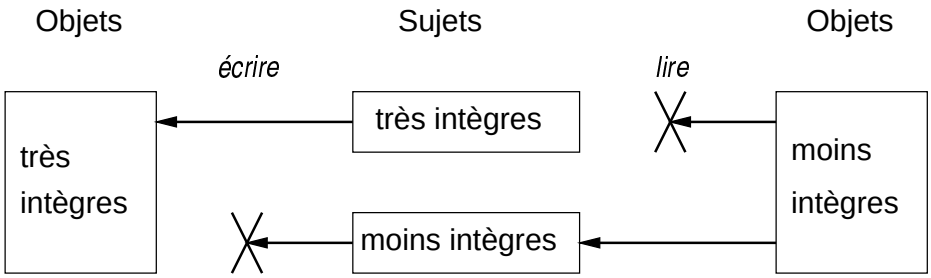



FIG. 2.3: Élimination de la contamination des informations intègres par les informations moins intègres

Il existe beaucoup d'exemples de données qui sont publiques, mais où le système dépend de leur intégrité, par exemple les fichiers de configuration d'un système Unix. Avec les mécanismes adéquats pour vérifier les conditions ci–dessus, beaucoup de tâches d'administration du système pourraient se faire sans privilège particulier. Malheureusement les systèmes d'exploitation réalisent rarement les mécanismes permettant de garantir l'intégrité des données.

## 2.6 Disponibilité

Il s'agit d'assurer la disponibilité du système lui même et des informations stockées dans le système.
L'accès au système peut être interrompu par des causes non volontaires : une faute de programmation dans une application, une panne générale du système ou bien un événement plus grave (par exemple coupure d'électricité, incendie ou tremblement de terre). Il peut également être interrompu par des actions volontaires, par exemple une surconsommation des ressources. Ces actions sont normalement appelées les perturbations ("Denial of Service" ou "DoS").
Une méthode très répandue pour garantir la disponibilité est de limiter la consommation des ressources par chaque processus et utilisateur. Si un seul utilisateur n'arrive pas à surcharger le système, suffisamment de ressources doivent rester pour pouvoir garantir la disponibilité. Il n'est pas possible de distinguer le cas où plusieurs utilisateurs coopèrent en surchargant le système du cas où le système est réellement surchargé, alors nous regardons ce cas comme un problème de conception ou configuration du système.
Unix est un bon exemple de système qui limite la consommation des ressources de chaque utilisateur pour pouvoir garantir la disponibilité. Le `quota(1)` limite l'utilisation de ressources sur le disque et l'`ulimit(1)` limite la consommation d'autres ressources comme la mémoire virtuelle, le nombre des processus qui peuvent être exécutés en parallèle.
La disponibilité peut être vue comme un paramètre de qualité de service (la non–disponibilité offre un service de très mauvaise qualité). Le problème a donc aussi été étudié dans le contexte des systèmes qui offrent une garantie par rapport à la qualité de service. Dans le système Alta [Tullman98] toutes les ressources sont allouées selon des réservations faites à l'avance par les utilisateurs (normalement en payant pour l'utilisation de ces ressources).
Le problème de la disponibilité n'a pas encore été pris très au sérieux par la communauté de la recherche en sécurité, mais l'intérêt de l'industrie pour les systèmes temps réel va probablement changer cela dans le futur.

## 2.7 Politiques de sécurité

Les politiques de sécurité définissent l'ensemble des propriétés de sécurité que l'on désire assurer dans un système ainsi que les dispositifs pour les assurer. La politique doit identifier les objectifs de sécurité du système et les menaces auxquelles le système devra résister.

> *"La politique de sécurité d'un système est l'ensemble des lois, règles et pratiques qui régissent la façon dont l'information sensible et les autres ressources sont gérées, protégées et distribuées à l'intérieur d'un système spécifique"* [ITSEC91][9].

La politique de sécurité décide de la validité de tout accès aux objets protégés. La définition de la politique de sécurité est donc vitale pour la sécurité du système. L'importance de cette définition est bien captée par l'observation suivante de Ames, Gasser et Schell :

> *"Un système donné est uniquement "sûr" par rapport à une politique de sécurité spécifique."* [Ames83]

Une politique de sécurité se divise naturellement en trois sous–politiques : la *politique de sécurité physique* qui contrôle le droit d'accès physique aux machines, la *politique de sécurité administrative* qui contrôle l'administration du système et le personnel qui l'utilise et la *politique de sécurité logique* qui contrôle les droits d'accès aux données par les sujets du système. Ces différentes sous–politiques sont décrites dans la suite.

### 2.7.1 Politiques de sécurité physique

Les politiques de sécurité physique s'occupent de tout ce qui concerne la situation physique du système. En particulier, elles définissent les mesures contre le cambriolage, les incendies, les catastrophes naturelles, et les coupures d'électricité ou d'eau (pour la climatisation).
Dans le cas de systèmes répartis, ces politiques sont de plus en plus difficiles à mettre en œuvre parce que les responsables contrôlent de moins en moins la localisation physique du système.

### 2.7.2 Politiques de sécurité administrative

Les politiques de sécurité administrative traitent de tout ce qui concerne l'administration du système et des gens qui l'utilisent. Elles définissent les responsabilités de chacun, par exemple le nom du responsable de la sécurité et le nom du responsable des postes d'incendie.
Ces politiques doivent aussi déterminer les personnes qui sont dignes de confiance pour certaines opérations, et donc à qui pourra être confiée l'autorisation de les exécuter.
Enfin les politiques de sécurité administrative s'occupent de toutes les procédures établies autour du système, par exemple pour la manipulation des entrées–sorties (imprimantes, sauvegardes, etc.), l'installation et maintenance des logiciels, l'installation des nouvelles machines sur le réseau, etc. Ces règles ne peuvent pas être vérifiées par le système lui même, alors elles doivent être vérifiées par des contrôles physiques, par exemple des verrous, des gardiens, etc.

[9] Traduction selon Vincent Nicomette [Nicomette96].

### 2.7.3 Politiques de sécurité logique

Les politiques de sécurité logique sont réalisées par le système informatique lui–même. Elles se décomposent en deux politiques distinctes : *la politique d'authentification* et *la politique d'autorisation*.

> **La politique d'authentification** détermine comment les utilisateurs (et programmes) s'identifient auprès du système et la vérification de cette identification. La politique d'authentification décide quel mécanisme d'authentification utiliser (cf. 2.8.1).
>
> **La politique d'autorisation** détermine les droits que les sujets ont sur les objets. La politique spécifie également comment ces droits peuvent être modifiés.

Nous allons étudier les politiques d'autorisation dans la suite.

### 2.7.4 Politiques d'autorisation

On distingue entre deux types de politiques d'autorisation : *les politiques discrétionnaires* et *les politiques mandataires*[10].

#### Politiques discrétionnaires

Dans le cas d'une politique d'autorisation discrétionnaire, les droits d'accès peuvent être manipulés librement par les sujets eux–mêmes. Le droit de manipuler les droits d'accès à un objet est normalement considéré comme un droit d'accès particulier qui peut être délégué à certains sujets (le propriétaire de l'information est normalement parmi ces sujets). Ainsi le droit de déléguer les droits d'accès est confié à la discrétion des sujets et se base largement sur la confiance dans les sujets.
Remarquons que l'abus de cette confiance peut amener le système dans un état non sûr, c'est–à–dire contraire à la politique d'autorisation définie). Ainsi les systèmes qui réalisent une politique d'autorisation discrétionnaire sont particulièrement vulnérables aux attaques comme le cheval de Troie.
Les politiques discrétionnaires sont très répandues dans les systèmes d'utilisation civils par exemple Unix et Windows NT.

#### Politiques mandataires

Dans le cas d'une politique d'autorisation mandataire, les interactions entre sujets et objets sont dirigées par des règles incontournables. Ces règles déterminent les droits d'accès qu'un sujet particulier peut posséder sur n'importe quel objet.
L'une des façons de spécifier ces règles est d'imposer une hiérarchie pour les sujets et pour les objets ; c'est le cas des politiques multi–niveaux appliquées par les militaires [Landwehr81]. Dans une telle politique, les informations sont classées selon leur sensibilité et les utilisateurs sont habilités à accéder à l'information jusqu'à un certain niveau de *classification de sécurité*. Ces classifications et habilitations définissent un ordre total sur les informations stockées dans le système et sur les utilisateurs.

[10] Une description des politiques discrétionnaires et mandataires similaire à celle donnée ici apparaît dans le chapitre 9 "Tolérance aux fautes, sécurité et protection" (par Y. Deswarte) dans "Construction des systèmes d'exploitation répartis" [Balter91].

Les interactions légales selon les règles suivent normalement un modèle de protection, par exemple celui proposé par Bell & LaPadula (cf. 3.4). Mais il existe aussi d'autres politiques mandataires, par exemple le modèle d'intégrité de Biba décrit ci–dessus. Dans les politiques mandataires, la confiance est remplacée par le contrôle. Comme dit le proverbe, "la confiance est bien mais le contrôle est mieux".

## 2.8 Mécanismes pour mettre en œuvre la sécurité

Il faut un certain nombre de dispositifs pour mettre en œuvre une politique de sécurité dans un système réparti. Trois mécanismes de base sont normalement identifiés dans la littérature : un mécanisme d'authentification, un mécanisme d'autorisation et un mécanisme de communication sûr.

### 2.8.1 Authentification

Il est impératif d'établir l'identité d'un interlocuteur qui souhaite avoir accès aux ressources du système et de garantir l'authenticité des requêtes et réponses entre machines.
Les buts de l'authentification ("authentication") est de vérifier l'identité de tous les utilisateurs qui souhaitent utiliser le système, de leur attribuer un identificateur système et de garantir la validité de cet identificateur entre machines, afin de pouvoir déterminer l'initiateur (utilisateur ou groupe d'utilisateurs) de tout accès aux ressources du système (y compris l'envoi de message).
La vérification consiste à s'assurer que l'utilisateur est bien celui dont il présente l'identité, au moyen de :

- quelque chose *que connaît* l'utilisateur (par exemple un mot de passe),
- quelque chose *qu'il possède* (badge, carte magnétique, carte à puce, etc.),
- quelque chose *qui lui est propre* (empreinte digitale, voix, scanographie de rétine, etc.),
- quelque chose *qu'il sait faire* (par exemple une signature).

Pour augmenter l'efficacité, une combinaison de plusieurs de ces dispositifs peut être utilisée, par exemple l'identification utilisée par la Carte Bleue utilise une combinaison d'une carte à puce et d'un mot de passe (le P.I.N. "Personal Identification Number").
La plupart des systèmes utilisent aujourd'hui le mot de passe comme seul dispositif de vérification. Par exemple le système Unix maintient un fichier avec les mots de passe chiffrés de tous les utilisateurs enregistrés. La procédure de `login(1)` exige que l'utilisateur spécifie son identité et son mot de passe, puis vérifie que les deux correspondent. Il y a deux problèmes avec ce schéma : les utilisateurs choisissent souvent des mots de passe faciles à deviner (comme "password" ou "secret") et les mots de passe sont vulnérables chaque fois qu'ils sont utilisés (beaucoup de systèmes transmettent les mots de passe en clair sur le réseau).
Pour répondre aux défaillances des mots de passe, un certain nombre de mécanismes d'authentification ont été proposés. La plupart des ces mécanismes se basent sur un protocole d'authentification proposé par Needham et Schroeder [Needham78]. Un exemple d'un tel système est le système Kerberos [Steiner88, Kohl93] — développé au Massachussets Institute of Technology (MIT) aux États Unis — qui est très répandu dans les systèmes commerciaux.
Au moment du `login`, le serveur d'identification de Kerberos envoie un certificat à

l'utilisateur, chiffré avec le mot de passe de l'utilisateur, qui doit entrer son mot de passe pour obtenir le certificat sur le poste client ; ainsi le mot passe de l'utilisateur ne passe jamais en clair sur le réseau. Ce certificat est ensuite utilisé par l'utilisateur pour s'identifier auprès des différents serveurs du système.
Un modèle de contrôle d'accès qui se base sur les certificats a été développé pour l'utilisation dans les systèmes répartis par Lampson et al. [Lampson92]. Les sujets sont identifiés par leur clé privée, c'est–à–dire leur capacité à chiffrer une chaîne de caractères qui peut ensuite être déchiffrée par la clé publique de celui qu'il prétend être. Le modèle permet à un sujet de déléguer une partie de ses droits d'accès à quelqu'un d'autre avec un certificat de délégation. Le certificat de délégation prouve que le fournisseur représente celui qui a signé le certificat. L'applicabilité de ce modèle a été montrée par sa réalisation dans le système Taos [Wobber94]. D'autres systèmes d'authentification qui se basent uniquement sur les certificats ont été proposés [Aura99, Mazieres98]. Ce type d'authentification nécessite une infrastructure qui permet aux serveurs soit de vérifier le certificat qui détermine l'identité du sujet, soit de fournir la clé publique de l'utilisateur d'une façon sûre. Des exemples de ce type d'infrastructure sont X.509 [ITU93] proposé par l'Union Internationale de la Télécommunications (ITU, auparavant CCITT) et SPKI [Ellison98, Ellison99] qui est en cours de définition par l'Internet Engineering Task Force (IETF).
L'avantage de ces schémas est de rendre la délégation des droits d'accès plus souple, il suffit d'envoyer à quelqu'un un certificat pour lui transférer le droit d'accéder aux ressources protégées.

### 2.8.2 Autorisation

L'autorisation vise à fixer les règles pour les relations entre sujets et objets dans le système et à garantir que ces règles sont respectées. Un *sujet* est une entité active (utilisateur, programme, etc.) et un *objet* est une entité passive (fichier, écran, ressource de calcul, etc.) qui peut être manipulée par des sujets autorisés.
Nous reviendrons sur l'autorisation dans le chapitre suivant.

### 2.8.3 Communication sûre

Quand des informations sensibles doivent être transmises entre deux sous–systèmes de sécurité différents, elles peuvent être interceptées ou modifiées par un attaquant. Alors il faut un moyen de rendre les informations incompréhensibles pour l'attaquant et de les protéger contre les modifications.
Il existe deux techniques différentes pour rendre les informations incompréhensibles : la stéganographie et le chiffrement [Rivest98].

> **La stéganographie** consiste à cacher un message dans un autre message beaucoup plus large. Par exemple un message peut être caché dans une image en utilisant les bits les moins significatifs de chaque pixel comme bits de message. Le message passe en clair sur le réseau, donc tout le monde peut potentiellement le lire, mais tout le monde ne sait pas comment trouver le message.
>
> **Le chiffrement** consiste à transformer des informations en clair ("clear text") en un texte chiffré ("cipher text") à l'aide d'une clé maintenue secrète et une fonction (réversible) de chiffrement. La fonction inverse s'appelle le déchiffrement.

La sténographie est rarement utilisée parce qu'elle impose un surcoût en communication pour cacher le message envoyé. Nous allons donc nous concentrer sur le chiffrement dans la suite. Il existe deux classes d'algorithme de chiffrement : chiffrement symétrique et chiffrement asymétrique.

#### Chiffrement symétrique

Les algorithmes de chiffrement symétrique, comme par exemple DES [NBS77] ou IDEA [Lai92] utilisent la même clé pour le chiffrement et le déchiffrement.

#### Chiffrement asymétrique

Le chiffrement asymétrique [Diffie76, Rivest78] utilise une clé pour le chiffrement et une autre pour le déchiffrement. La clé de chiffrement, qui est normalement connue par tout le monde (elle s'appelle aussi *la clé publique*), permet d'envoyer un message chiffré à quelqu'un qui connaît la clé de déchiffrement. Cette clé ne doit pas être divulguée par le récepteur, et elle s'appelle donc aussi *la clé secrète*.

#### L'utilisation du chiffrement

Le chiffrement était interdit par la loi en France sans permission préalable. Dans une conférence de presse le 19 janvier 1999 [Jospin99], le premier ministre annonçait la libération partielle du chiffrement, et le chiffrement avec une longueur de clé jusqu'à 128 bits est maintenant libre. La libération s'applique à l'importation et l'utilisation du chiffrement, même si les ajouts faits le 3 décembre 1998 à l'accord de Wassenaar [Wassenaar95] limitent l'exportation des produits de chiffrement à l'étranger.
Les produits de chiffrement les plus répandus sont SSL ("Secure Socket Layer") [Freier96] et PGP ("Pretty Good Privacy") [Zimmermann95, Callas98].
Pour en savoir plus sur le chiffrement, nous recommandons le livre "Applied Cryptography" de Bruce Schneier [Schneier96] qui est un ouvrage de référence fort complet.

## 2.9 Évaluation de la sécurité

Pour que les acheteurs et les utilisateurs puissent avoir l'assurance que leurs systèmes sont sûrs, un certain nombre de critères d'évaluation et de classification de la sécurité d'un système informatique ont été proposés. Nous allons étudier les plus courants pour déterminer les traits de la sécurité les plus répandus.

### 2.9.1 Le Livre Orange

Les critères publiés par le ministère de la défense américaine ("Department of Defence" ou DoD) dans le rapport "Trusted Computer Security Evaluation Criteria" [DoD85] (plus connu sous le nom de TCSEC ou "le Livre Orange"[11]) sont probablement les plus connus. En fait ces critères sont devenus la référence canonique en matière de sécurité des systèmes informatiques.

[11] Le livre orange est ainsi nommé parce que il appartenait à une série de livres sur la sécurité publiés sous des couvertures des couleurs différentes et la couverture de celui-là était orange.

Les critères fournissent une classification en sept niveaux regroupés en quatre classes de sécurité A, B, C et D (A est le plus sûr). Outre sa propre définition, chaque classe réalise aussi les traits de sécurité des classes précédentes. La classification est donnée dans le tableau 2.1.
Pour être évalué à un certain niveau (être associé à une classe de sécurité), le système doit satisfaire toutes les conditions requises du niveau et toutes celles des niveaux précédents. La plupart des systèmes installés sur les micro–ordinateurs sont classifié D ou C1, mais quelques systèmes d'Unix ont atteint une classification C2.
La politique mandataire imposée par le livre orange est celle de Bell & LaPadula (nous traitons cette politique en plus de détail dans 3.4). Cette politique réalise le modèle d'accès aux informations militaires, c'est–à–dire garantit le secret des informations. Ce modèle n'est pas forcément le plus approprié pour les systèmes civils, c'est pourquoi un nombre d'autres modèles ont été proposés pour prendre en compte l'intégrité et la disponibilité des données.
L'introduction de ces modèles a provoqué la création d'autres critères d'évaluation, notamment les ITSEC ("Information Technology Security Evaluation Criteria") adoptés par l'Union Européenne, les CTCPEC ("Canadian Trusted Computer Product Evaluation Criteria" [CTCPEC93]) au Canada et les JCSEC ("Japanese Computer Security Evaluation Criteria" [JCSEC92]) au Japon.

### 2.9.2 Les ITSEC

Les ITSEC sont le résultat d'une collaboration entre quatre pays européens : l'Allemagne, l'Angleterre, la France et les Pays–Bas [ITSEC91]. Les ITSEC introduisent la séparation entre fonctionnalité et assurance : ils définissent un nombre de classes de fonctionnalité (comme le livre orange) et un nombre de niveaux d'assurance.
Une classe de fonctionnalité décrit les fonctionnalités qu'un système doit mettre en œuvre pour être évalué à ce niveau. Les ITSEC définissent les classes F–C1, F–C2, F–B1, F–B2 et F–B3 correspondant aux classes C1 — B3 définies par le livre orange.
Un niveau d'assurance permet de décrire les preuves qu'un système doit apporter pour montrer qu'il réalise les fonctions spécifiées par la classe de fonctionnalité. Les ITSEC définissent sept niveaux d'assurance de E0 à E6. Chaque niveau rajoute des conditions de sécurité (surtout au niveau de spécification et conformité entre conception et réalisation) au niveau d'avant pour finalement arriver à une spécification formelle du sous–système de sécurité et de l'architecture générale.
Les ITSEC introduisent la notion de *cible d'évaluation* ("Target of Evaluation" ou "TOE") pour désigner l'ensemble des composants du système évalués, c'est–à–dire la politique de sécurité, les fonctions requises dédiées à la sécurité, la définition des mécanismes de sécurité requises et le niveau d'évaluation visé.

### 2.9.3 Les critères communs

La diversité des critères d'évaluation rend la comparaison de la sécurité entre systèmes très difficile. Les critères communs (CC) ont été proposés pour harmoniser les critères américains (TCSEC), canadiens (CTCPEC) et européens (ITSEC). La version 2.0 de ces critères [ISO15408] a été promue Norme Internationale par l'Organisation Internationale de Normalisation (ISO) le 8 juin 1999 .

| | | |
|---|---|---|
| **D** | **Non classé** – *Les systèmes dans cette classe n'ont pas été soumis à l'évaluation ou ils n'ont pas qualifiés à une classification supérieure. Ils offrent une sécurité minimale.* | |
| **C** | **Protection discrétionnaire** – *Les systèmes dans cette classe offrent les mécanismes pour la mise en œuvre d'une politique de protection discrétionnaire. Il existe deux sous–classes :* | |
| | **C1** | **Accès discrétionnaire** – Ces systèmes offrent une séparation entre des utilisateurs et leurs données et les mécanismes pour protéger les données d'un utilisateur contre l'accès non–autorisé d'autres utilisateurs. |
| | **C2** | **Accès contrôlé** – Ces systèmes offrent l'authentification des utilisateurs, l'audit des événements de sécurité et un mécanisme de protection séparé. |
| **B** | **Protection mandataire** – *Dans cette classe la sécurité est réalisé par le sous–système de sécurité. Il doit vérifier l'accès à toutes les données et appliquer la politique de sécurité. Le moniteur de référence doit également être réalisé. Il y a trois sous–classes :* | |
| | **B1** | **Protection avec marquage** – Ces systèmes marquent tous les objets avec leur classification de sécurité et imposent que les objets peuvent uniquement être utilisés par des sujets qui ont une habilitation au même niveau ou à un niveau supérieur. |
| | **B2** | **Protection structurée** – Ces systèmes ont été réalisés dès le départ avec une politique de sécurité documentée. Tous les canaux de communication qui peuvent être exploités (même les canaux cachés cf. 2.3.3) doivent être identifiés. |
| | **B3** | **Domaines de sécurité** – Le sous–système de sécurité de ces systèmes réalise le moniteur de référence. La notion de domaine de sécurité est introduite. Un domaine de sécurité se compose de *(i)* une liste d'utilisateurs (ou groupes d'utilisateurs) qui peuvent accéder à un objet et *(ii)* une liste d'utilisateurs (ou groupes d'utilisateurs) qui ne peuvent pas accéder à l'objet. |
| **A** | **Protection vérifiée** – *Les systèmes dans cette classe ont été soumis à une conception formelle. Ils s'adressent à la gestion des données classifiées.* | |
| | **A1** | **Conception vérifiée** – Ces systèmes offrent les mêmes traits que les systèmes dans la classe B3, mais la conception formelle complète du système offre une garantie que le système soit sûr. |

TAB. 2.1: Classification du livre orange

Les CC reprennent la séparation entre fonctionnalité et assurance des ITSEC, de même que la notion de cible d'évaluation qui désigne le système à évaluer et *l'objectif de la sécurité* ("Security Target" ou "ST"). La spécification de l'objectif de la sécurité peut contenir un ou plusieurs profils de protection ("Protection Profiles" ou "PP"). Un PP définit un exemple d'exigences de sécurité et d'objectifs, indépendants d'une quelconque réalisation. Les profils de protection permettent de spécifier les objectifs de la sécurité en termes de profils connus. Ceci rend la spécification de l'objectif de la sécurité plus simple pour le développeur qui demande l'évaluation de son système et pour le client qui achète un tel système. Les nouveaux profils de protection peuvent être définis à partir des PP existants pour prendre en compte les besoins de sécurité particuliers. Une partie importante des CC est donc consacrée à la présentation détaillée des profils de protection prédéfinis.

### 2.9.4 Conclusion

L'évaluation de la sécurité d'un système permet de savoir à quel niveau on peut lui faire confiance. En général, le système évalué au niveau le plus sûr peut recevoir les informations les plus sensibles, mais cela dépend bien évidement aussi de l'environnement du système et des menaces auxquelles le système est soumis.
La séparation entre la fonctionnalité et l'assurance montre l'importance de la réalisation pour la sécurité d'un système. La fonctionnalité peut suffire pour réaliser un modèle de protection particulier, mais si le système n'est pas conçu et réalisé d'une façon sûre, cette fonctionnalité ne vaut rien.
En général les niveaux d'évaluation les plus sûrs requièrent une spécification formelle de l'architecture et de la conception du système et une vérification (également formelle) que la réalisation est conforme aux spécifications.

## 2.10 Récapitulation et discussion

Il existe aujourd'hui très peu de systèmes sûrs en dehors de l'industrie militaire ou de quelques domaines spécialisés comme l'industrie nucléaire ou les transports. Dans la plupart des systèmes disponibles sur le marché, la sécurité n'a pas fait partie de la conception du système. La sécurité est un paramètre parmi d'autres dans la conception d'un système informatique. Pour réaliser un système bien équilibré, il s'agit de trouver le meilleur compromis possible entre ces paramètres. La sécurité joue un rôle très particulier, parce que la moindre défaillance peut compromettre le bon fonctionnement du système. Si l'algorithme d'ordonnancement marche dans 95% des cas, mais n'est pas équitable dans 5% des cas, le système continue à fonctionner. Si le système de sécurité ne marche que dans 95% des cas, les 5% restants peuvent être exploités par un adversaire et compromettre toute la sécurité du système.
Dans ce chapitre, nous avons identifié un certain nombre d'objectifs pour la sécurité et des menaces contre la sécurité des systèmes répartis. Les objectifs les plus importants sont :

- La confidentialité des informations stockées dans le système
- L'intégrité de ces informations
- La disponibilité des informations et des ressources du système

Pour atteindre ces objectifs, le système doit mettre un certain nombre de dispositifs en place, notamment les mécanismes pour :

- L'identification et l'authentification des sujets du système
- Le contrôle d'accès aux ressources
- La communication sûre

Le contrôle d'accès aux ressources du système est spécifié dans la politique de sécurité du système. Cette politique spécifie les contrôles physiques, administratifs et logiques du système. Les particularités de la sécurité des systèmes répartis sont résumées dans le tableau 2.2.

| | Système central (mainframe) | Système d'exploitation réparti | Systèmes coopérants |
|---|---|---|---|
| identification | globale | globale | locale |
| authentification | implicite | dépend du système | explicite |
| autorisation | globale | globale | locale |
| communication sûre | implicite | dépend du système | explicite |
| modèle de confiance | globale | globale | locale |
| politique de sécurité | globale | globale | locale |
| conséquences d'une défaillance | globales | globales | locales |

TAB. 2.2: Résumé des particularités des systèmes répartis

Les systèmes d'exploitation répartis ont été conçus pour donner l'illusion d'un système centralisé. La répartition entre plusieurs sites nécessite des mécanismes d'authentification et de communication sûrs, pour pouvoir maintenir la transparence de cette répartition. Dans les systèmes coopérants, la répartition est explicite et les applications peuvent réaliser leurs propres mécanismes.
Dans un système d'exploitation réparti, le domaine d'application de la politique de sécurité est global. Les systèmes coopérants peuvent avoir une politique globale, mais le domaine de toutes les sous–politiques est local.

# Chapitre 3

# Contrôle d'accès

Le contrôle d'accès est indispensable pour la sécurité dans les systèmes informatiques. Paradoxalement son étude n'a pas reçu beaucoup d'attention de la part de la communauté de la recherche. Depuis le début des années 1980, la doctrine des politiques mandataires et discrétionnaires — établie dans le contexte des systèmes militaires — a lentement été remise en cause comme base appropriée pour le contrôle d'accès dans les systèmes civils [Sandhu96]. Ensuite nous avons vu l'introduction de différents modèles de contrôle d'accès (par exemple le contrôle d'accès basé sur les rôles ou le modèle de la muraille de Chine) développés pour répondre aux différents besoins de protection dans les systèmes civils.
Dans ce chapitre nous allons d'abord introduire la notion de contrôle d'accès dans un système informatique. Nous allons ensuite décrire les besoins de la sécurité militaire et les besoins de la sécurité dans les systèmes civils. Un certain nombre de modèles de contrôle d'accès ont été développés pour répondre aux problèmes de sécurité militaires (notamment le modèle de Bell & LaPadula) et civils (le modèle discrétionnaire d'Unix et deux modèles mandataires — le modèle basé sur les rôles et le modèle de la muraille de Chine) ; nous décrivons les modèles les plus répandus. Ensuite nous décrivons les différents mécanismes pour la mise en œuvre du contrôle d'accès. Enfin nous discutons le contrôle d'accès dans les systèmes répartis et présentons une récapitulation et une discussion des matières traitées dans ce chapitre.

## 3.1 Introduction au contrôle d'accès

La politique de sécurité se compose de trois sous–politiques de contrôle : d'accès physique, administratif et logique. Nous nous intéressons plus particulièrement à la politique de contrôle d'accès logique et aux modèles de protection et mécanismes nécessaires pour la réaliser.
Le modèle de base de toutes les politiques de contrôle d'accès est montré sur la figure 3.1.

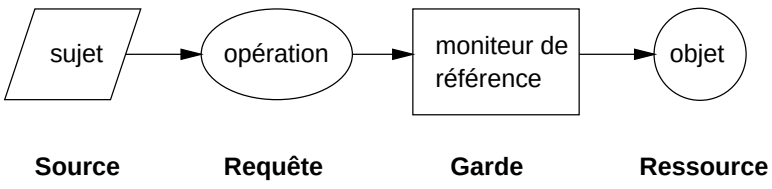



FIG. 3.1: Le modèle de base du contrôle d'accès

La figure montre un sujet qui souhaite faire une opération sur un objet. Le système transforme

l'opération en une requête qu'il passe au moniteur de référence qui contrôle l'accès aux ressources. Si le sujet est autorisé à accéder à l'objet selon la politique de sécurité en vigueur, l'accès à l'objet va être accordé et l'opération peut se dérouler normalement.
La majorité des idées de base sur le contrôle d'accès ont été fixées au début des années 1970. Elles sont largement inspirées par un article de Butler Lampson sur la protection [Lampson71]. Cet article introduit la notion d'une *matrice de contrôle d'accès* qui décrit quels sont les droits d'un sujet sur les ressources du système.
La matrice de contrôle d'accès est une structure qui contient une ligne par sujet et une colonne par objet dans le système. La cellule à l'intersection d'une ligne et d'une colonne décrit les droits d'accès du sujet sur l'objet. La figure 3.2 montre un exemple de matrice de contrôle d'accès.

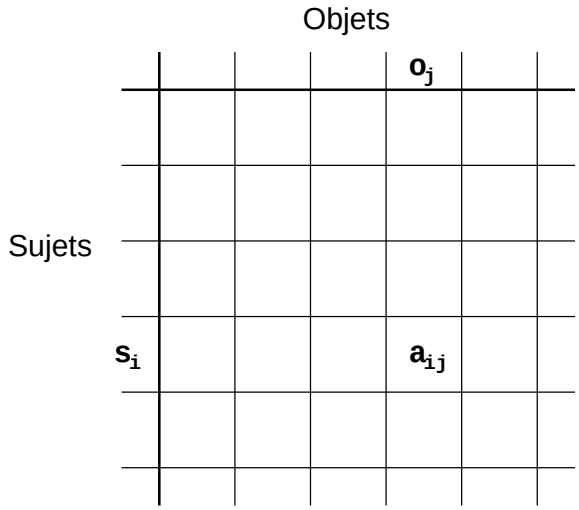



FIG. 3.2: La matrice de contrôle d'accès

La matrice représentée sur la figure ci–dessus montre un sujet $s_i$ qui possède le droit d'accès $a_{ij}$ sur l'objet $o_j$.
Le modèle de contrôle d'accès basé sur la matrice de contrôle d'accès ne prévoit aucune interprétation particulière du sujet, de l'objet ou du droit d'accès, mais ce sont normalement des entités primitives du système.
La granularité de la définition du sujet et de l'objet dépend du système. Quelques définitions répandues sont : un utilisateur, un processus ou une machine pour les sujets et un fichier, une relation dans une base de données ou une variable dans un programme pour les objets.
Remarquons que les sujets ont normalement aussi une entrée comme objet dans la matrice. Ceci permet de contrôler le droit d'envoyer un message à un utilisateur ou un événement à un processus.
Quelques exemples des droits d'accès sont : le droit de lire, écrire, rajouter à la fin ou exécuter un fichier pour un système de fichiers, le droit de lire ou modifier une relation dans une base de données, le droit d'envoyer ou recevoir un message pour un canal de communication ou le droit de lire ou modifier une variable dans un programme.
Un modèle de protection qui se base directement sur la matrice de contrôle d'accès a été proposé par Harrison, Ruzzo et Ullman [Harrison76]. Ils donnent une description formelle du modèle et définissent la sûreté d'un système comme la propriété qu'il n'existe pas de fuite des

droits d'accès dans le système, c'est–à–dire que le seul moyen d'entrer un droit d'accès particulier dans la matrice est l'insertion du droit par un sujet autorisé. Leur analyse montre que la sûreté du modèle n'est pas déterministe en général, même si la sûreté d'un système particulier dans une configuration particulière peut être prouvée. Ceci indique qu'il va probablement être difficile de prouver la sûreté d'un système de protection particulier qui se base sur la matrice de contrôle d'accès.

## 3.2 La structure de sécurité militaire

Une large partie de la recherche en contrôle d'accès a été faite dans le contexte des systèmes militaires ; nous allons donc d'abord étudier les objectifs de cette recherche, c'est à dire la structure de sécurité militaire[1].
Les informations militaires sont classées selon leur sensibilité pour la défense nationale. La classification consiste en différents niveaux (ou classes) de confidentialité ("security levels") pour l'information. Chaque classe est normalement divisée en sous–classes ("compartments") qui précise la nature de l'information. La classification utilisée en France [SCSSI91a] est montrée dans le tableau 3.1[2].

| | |
|---|---|
| Informations classifié | |
| Très secret défense | Réservée aux informations dont la divulgation est de nature à nuire à la Défense nationale et à la sûreté de l'État et qui concernent les priorités gouvernementales en matière de Défense. |
| Secret défense | Réservée aux informations dont la divulgation est de nature à nuire à la Défense nationale et à la sûreté de l'État. |
| Confidentiel défense | Réservée aux informations qui ne présentent pas elles–mêmes un caractère secret, mais dont la connaissance, la réunion ou l'exploitation peuvent conduire à la divulgation d'un secret intéressant la Défense nationale et la sûreté de l'Etat. |
| Informations non–classifié | |
| Secret | *Les indications de la confidentialité d'un document non–classifié portent normalement une mention spécifique du domaine protégé (personnel, médical, industrie, médical, industrie, commercial, etc.) et une précision de l'évolution de la classification dans le temps – à quel moment les informations doivent être déclassées* |
| Confidentiel | |
| Diffusion restreinte | |

TAB. 3.1: Classification militaire des informations

Une *habilitation* ("clearance") est associée à chaque utilisateur. L'habilitation spécifie le niveau de sécurité auquel l'utilisateur est autorisé à accéder. Une liste de sous–classes autorisées est normalement associée à l'habilitation, qui limite les informations auxquelles l'utilisateur habilité peut accéder ("need to know principle").

[1] La source principale de cette description est l'article de Landwehr dans "Computing Surveys" [Landwehr81].
[2] Il existe une hiérarchie analogue pour les informations de l'OTAN avec les classes : Très secret cosmic, Secret OTAN, Confidentiel OTAN et Diffusion restreinte OTAN.

Pour pouvoir accéder à l'information, le sujet doit avoir une habilitation égale (ou supérieure) à la classification de l'information et avoir la sous–classe de l'information dans sa liste des sous–classes autorisées. L'habilitation d'un sujet spécifie donc en même temps à quel niveau l'autorité militaire lui fait confiance et le niveau et la nature des informations qu'il a besoin de connaître.
Les documents sont normalement stockés dans un coffre–fort, donc le droit d'accéder à un document correspond au droit de le retirer du coffre fort. La classification d'un document est normalement tamponnée sur le document même et sur le cahier qui le contient. Pour pouvoir retirer le document, le sujet doit prouver qu'il possède l'habilitation nécessaire soit par son rang militaire, soit par d'autres moyens.
Un document militaire ne peut pas changer de classification mais ses informations peuvent être copiées dans un nouveau document. Dans ce cas il faut que la classification du nouveau document corresponde à la classification des informations les plus sensibles copiées dans ce document. Les informations peuvent perdre leur importance, par exemple la conception de la fortification de Grenoble (notamment le Fort St. Eynard et la Bastille) n'a plus d'importance militaire. Dans ce cas un nouveau document d'une classification inférieure est crée et les informations y sont copiées – cette procédure s'appelle la déclassification ("sanitizing") des informations. Toutes les procédures de protection sont mises en place pour protéger les informations classifiées, donc la déclassification est un processus très sensible qui suit des règles particulières.
La structure de la sécurité militaire a été développée pour résoudre deux problèmes : d'abord de contrôler l'accès aux informations secrètes (confidentialité) et ensuite d'éviter que les informations d'une classification élevée soient copiées dans les documents d'une classification inférieure (confinement).
Dans le cas traditionnel des documents stockés dans un coffre fort, la sécurité est assurée par les contrôles physiques de l'accès aux documents. Le tampon de la classification sur le document rend la recopie difficile.
Dans un système informatique, la composition, la correction et la distribution des informations sont beaucoup plus rapides et la recopie des données beaucoup plus facile. De plus, il n'existe couramment pas de moyens de faire un tampon sur les données qui ne puisse pas être effacé facilement, par exemple en recopiant le fichier qui contient le document. Alors la mise en œuvre de la politique de protection logique doit pouvoir donner les mêmes garanties qu'un coffre fort avec des gardiens armés.

## 3.3 Les besoins de la sécurité dans les systèmes civils

Comme les systèmes militaires, les systèmes civils traitent des informations sensibles (par exemple les dossiers médicaux, relevés de comptes, secrets de fabrication etc.) dont l'assurance de la confidentialité est très importante. Mais pour une grande partie des applications civiles le besoin de pouvoir garantir l'intégrité des données gérées par le système est aussi important. Par exemple, l'intégrité d'un compte en banque est plus importante que le secret du solde. Cette différence entre les systèmes militaires et les systèmes civils a été étudiée par Clark et Wilson [Clark87]. Ils prétendent que l'intégrité est normalement plus importante pour les systèmes civils que la confidentialité.
Les techniques pour la mise en œuvre de l'intégrité dans les systèmes de gestion de

l'information (avant l'introduction de l'informatique) sont les suivantes : la transaction bien formée et la séparation des fonctions entre les employés.

### La transaction bien formée

La notion de transaction bien formée est plus générale que la définition normale d'une transaction en informatique. Elle garantit que les manipulations des données suivent certaines règles pour assurer l'intégrité des informations.
Avant l'ère de l'informatique, les comptables écrivaient toujours à l'encre et les modifications étaient faites par une entrée de correction sans effacer l'entrée existante ; toute trace de modification indiquait une tentative de fraude potentielle.
Aujourd'hui le système maintient un journal de toutes les modifications des données. Un audit de ce journal permet de révéler une tentative de fraude.

### La séparation des fonctions

La séparation des fonctions a pour but de garantir que les informations stockées dans le système correspondent à la réalité. L'idée générale est d'assurer que le traitement normal des données implique plusieurs utilisateurs. Le risque que plusieurs utilisateurs soient malveillants est plus faible. Pour mieux expliquer comment cela marche, nous allons prendre l'exemple d'un magasin qui vend des téléviseurs.
Si une seule personne est responsable pour tous les achats, les paiements, le stockage et les ventes des téléviseurs, il peut manipuler les données qui représentent les prix d'achat et les prix de vente dans le système (il peut par exemple augmenter le prix d'achat et baisser le prix de vente) par rapport aux prix réels et dérober la différence au magasin. Le problème dans cet exemple est que la même personne fournit toutes les informations au système ; il devient effectivement la seule source d'information pour le système. Cet exemple montre que les seuls mécanismes de protection de l'intégrité des données n'arrivent pas à assurer l'intégrité entre les valeurs représentées et leur représentation dans le système.
La séparation des fonctions entre plusieurs employés — par exemple un qui commande les téléviseurs et un autre qui paie les factures — rend la fraude plus risquée. Si celui qui commande les téléviseurs n'entre pas le bon montant dans le système, celui qui paie les factures va tout de suite découvrir la divergence et avertir le propriétaire. Il est possible pour les deux employés de collaborer pour escroquer le magasin, mais la séparation des fonctions augmente le risque d'être découvert.
Un autre exemple de la séparation des fonctions largement répandu est le droit de signer un chèque. Normalement le directeur d'une entreprise peut signer tout seul, mais il faut la signature de deux commis de bureau pour rendre le chèque valable. Ce principe est exprimé dans le modèle de contrôle d'accès basé sur les rôles (cf. 3.6).
Un dernier exemple de la séparation des fonctions concerne les employés des sociétés de service. Pendant une mission dans une entreprise, le consultant apprend beaucoup d'informations confidentielles. Il faut donc éviter qu'il travaille tout de suite après chez un concurrent parce que ceci introduit un conflit d'intérêt. La société de service doit gérer les consultants de manière à ce que ce conflit d'intérêt ne se produise pas. Ce principe est exprimé dans le modèle de la muraille de Chine (cf. 3.7).

## 3.4 Modèle de Bell et LaPadula

Le modèle mandataire de Bell et LaPadula [Bell73a, Bell73b, Bell75] est une formalisation de la structure militaire[3] adoptée pour l'usage dans un système informatique.
Le modèle définit les sujets, les objets, les classes de confidentialité et les habilitations comme auparavant.
Il définit également une relation de domination — similaire à celle du modèle de l'intégrité de Biba (cf. 2.5)[4]. Une classe de confidentialité (ou une habilitation) $A$ domine une autre classe $B$ (écrit $A \geq B$) si le flot d'information de $B$ vers $A$ est autorisé.
Les droits d'accès définis par le modèle sont les suivants :

| | | |
|---|---|---|
| *lire* | : | permet au sujet de lire l'objet mais pas de le modifier ; |
| *écrire* | : | permet au sujet de lire et écrire l'objet ; |
| *rajouter* | : | permet au sujet d'écrire l'objet mais pas de le modifier ; |
| *exécuter* | : | permet au sujet d'exécuter le contenu de l'objet comme un programme, mais il ne peut ni lire ni écrire l'objet. |

Un attribut de chaque objet spécifie l'identité du sujet qui contrôle l'objet (le propriétaire de l'objet). Le propriétaire est normalement celui qui a créé l'objet.
Les règles de sécurité sont exprimées par les deux conditions suivantes.

> **Condition de sécurité simple :** Un sujet $s$ peut lire un objet $o$ seulement si le niveau de confidentialité de $s$ domine le niveau de confidentialité de $o$.
>
> **La propriété $\star$ :** Un sujet $s$ peut utiliser le contenu d'un objet $o_1$ pour modifier un objet $o_2$ seulement si le niveau de confidentialité de $o_2$ domine le niveau de confidentialité de $o_1$.

Pour lire un objet le sujet doit donc avoir une habilitation qui égale ou surpasse le niveau de confidentialité de l'objet.
Ces deux conditions de sécurité sont normalement résumées comme suit : un sujet "peut lire vers le bas et écrire vers le haut". La propriété $\star$ (appelée "la propriété étoile") est moins intuitive. Elle dit que le sujet ne peut pas écrire un objet d'un niveau de confidentialité inférieur au niveau de l'information le plus élevé auquel il peut actuellement accéder. Ceci empêche le sujet (ou un cheval de Troie) de "déclasser" l'information contenue dans les objets auxquels il a accès.
Pour permettre à un maréchal d'écrire une petite annonce lisible par tout le monde, le modèle introduit la notion d'habilitation courante qui correspond au niveau de confidentialité le plus élevé de l'information accédée par le sujet. Si le maréchal n'a pas encore lu de l'information très secrète, il peut donc créer/écrire un objet d'un niveau de sécurité inférieur à ce niveau. Il faut donc que le système mémorise le niveau d'information le plus haut accédé par le sujet — depuis le login d'un utilisateur ou depuis le démarrage du processus — pour pouvoir garantir que l'objet créé n'a pas un niveau de confidentialité inférieur à celui–ci[5]. Si le sujet souhaite augmenter son niveau de confidentialité pour pouvoir accéder aux informations plus confidentielles, il faut donc qu'il perde le droit d'écriture pour les objets déjà actifs (un objet actif est un objet en cours d'utilisation).

[3] Le modèle a été développé en collaboration entre l'armée de l'air américaine et le MITRE Corporation — une entreprise d'armement américaine.

[4] En réalité c'est le modèle de Bell & LaPadula qui a inspiré Biba.

[5] Le système ADEPT–50 [Weissman69] réalisait un tel modèle de contrôle d'accès (il s'appelle le modèle de la ligne des hautes eaux) avant la conception du modèle de Bell & LaPadula.

La mise en œuvre du modèle de Bell & LaPadula doit respecter les trois principes suivants :

1. **Principe de tranquillité.** Le niveau de confidentialité d'un objet actif ne peut pas être changé.
2. **Non-accessibilité des objets inactifs.** Un sujet ne peut pas lire le contenu d'un objet inactif.
3. **Initialisation des objets.** L'état initial d'un objet (après création) ne dépend d'aucune ressource antérieurement allouée à un autre objet.

Le principe de tranquillité n'a pas de justification dans le modèle d'accès aux informations militaires, mais il permet de réaliser un mécanisme plus efficace si la politique de contrôle d'accès peut être vérifiée une fois au premier accès à l'objet et non à chaque accès.
Un sujet ne doit pas avoir accès aux objets non–actifs, par exemple la mémoire ou les blocs du disque dur non–alloués.
L'initialisation des objets à pour but d'assurer qu'il n'existe pas de fuite d'information à travers de la mémoire non–initialisée. Ce principe est mis en œuvre par le plupart des systèmes courants. En générale, l'allocation d'une page de mémoire ou d'un bloc de disque dur met la page ou le bloc à zéro avant le retourner au sujet. Les principes de la non–accessibilité des objets inactifs et l'initialisation des objets à la création garantissent que la mémoire (primaire ou secondaire) non–allouée ne peut pas constituer un canal caché.

### 3.4.1 Variations du modèle

De nombreuses variations de ce modèle ont été proposées dans la littérature, notamment le modèle de flot d'information. La conception de ce modèle est normalement attribué à Denning [Denning76]. Le modèle est une généralisation du modèle mandataire qui décrit les flots d'information sûrs dans un système. Denning montre que le modèle peut être exprimé dans la théorie des treillis. Ce résultat est important parce que de nombreux autres modèles peuvent être exprimés dans cette théorie [Sandhu93].
Le modèle d'intégrité proposé par Biba est une autre variation du modèle de Bell & LaPadula qui protège l'intégrité des informations au lieu de la confidentialité.

### 3.4.2 Discussion

Le modèle de Bell & LaPadula a été réalisé dans un certain nombre de systèmes [Ames78, McCauley79, Neumann77, Feiertag79], ceci a permis d'obtenir des expériences d'utilisation de ce modèle.
La limite principale du modèle est la restriction sévère imposée par la propriété étoile. Avec le principe de la ligne des hautes eaux, ceci a pour conséquence que le niveau de confidentialité des données remonte lentement vers le niveau le plus haut. Pour remettre les fichiers à leur niveau propre, les procédures de déclassification doivent être appliquées fréquemment. Quand la déclassification devient un événement quotidien, le risque de faute causée par un manque de concentration augmente énormément.
Le gain de cette restriction en terme de sûreté n'est pas évident parce que le modèle n'empêche pas les canaux cachés (notamment les canaux temporels).

## 3.5 Modèle discrétionnaire d'Unix

Le modèle de protection d'Unix est un exemple classique de modèle discrétionnaire. Dans un système Unix traditionnel, les contrôles d'accès sont principalement associés au système de fichiers (fichiers, répertoires et "named pipes") ; il existe quelques exceptions à cette règle, notamment l'accès à un port de réseau de numéro inférieur à 1024 et l'accès à un segment de mémoire partagée dans System V.
Les meta–données associées à chaque fichier désignent le propriétaire et un groupe d'utilisateurs auxquels le fichier appartient et les permissions d'accès pour le fichier.
La notion du groupe est très importante parce qu'elle permet à un ensemble fixe d'utilisateurs de partager des fichiers entre eux.

### 3.5.1 Permissions

Les permissions sont spécifiées par un tableau de bits qui indiquent l'existence ou l'absence des droits de lire, écrire et/ou exécuter le contenu du fichier pour le propriétaire, son groupe ou tous les utilisateurs du système.
En plus de ces bits de protection, le système maintient trois méta–bits de protection : le bit Set–UID, le bit Set–GID et le bit appelé "sticky".
Les bits de permission ont une sémantique différente pour les fichiers normaux et les répertoires, ces sémantiques sont décrites par la suite.

#### Sémantique pour les fichiers

`lire` permet de lire (et copier) le contenu du fichier ;

`écrire` permet de modifier le contenu du fichier ;

`exécuter` permet d'exécuter le contenu du fichier ;

`SUID` sur un fichier exécutable, permet d'affecter l'usager (UID) associé au processus à celui du fichier pendant l'exécution du programme ;

`SGID` sur un fichier exécutable, permet d'affecter le groupe (GID) associé au processus à celui du fichier pendant l'exécution du programme ;

`"sticky"` permet d'indiquer au système qu'un binaire (un programme ou une bibliothèque partagée) doit rester dans l'espace de pagination (swap) après la terminaison du programme. Ceci permet de réduire le temps de chargement de ces binaires.

#### Sémantique pour les répertoires

`lire` permet de lister les fichiers et sous–répertoires dans le répertoire ;

`écrire` permet de modifier le contenu du répertoire, c'est–à–dire créer, renommer et supprimer les fichiers de ce répertoire ;

`exécuter` permet d'accéder aux fichiers et sous–répertoires dans le répertoire. Sans droit de lire le répertoire, le sujet doit connaître le nom du fichier à l'avance pour y accéder. Ceci permet à un groupe d'utilisateurs de partager les fichiers sans avoir le support d'un groupe en commun. La sécurité dépend du secret des noms des fichiers, si

le fraudeur arrive à deviner le nom d'un fichier dans le répertoire il peut y accéder avec les mêmes droits que les autres ; ceci s'appelle aussi la "sécurité par obscurité" ;

`SUID` et `SGID` n'ont pas de signification particulière ;

`"sticky"` avec écrire permet de créer des répertoires où tout le monde peut créer des fichiers, mais personne (sauf `root`) ne peut effacer ou renommer les fichiers des autres.

### 3.5.2 Politique de protection par défaut

La politique de protection par défaut intervient au moment de la création d'un fichier. À ce moment le système attribue au fichier le nom de l'utilisateur qui le crée et le groupe du répertoire dans lequel il est créé. Les droits deviennent ceux du répertoire, modifiés avec l'`umask(2)` de l'utilisateur de la façon suivante.
L'`umask` est un bitmap qui indique les bits à supprimer dans les droits du fichier, c'est–à–dire les bits qui sont `1` dans l'`umask` deviennent `0` dans les droits. Ainsi l'`umask` classique de `022` indique que le droit d'écriture doit être retiré pour le groupe et les autres.

### 3.5.3 Mécanismes de protection discrétionnaire

Unix réalise trois mécanismes qui permettent au propriétaire (ou l'utilisateur `root`) de modifier les meta–données de la protection :

`chown(1)` qui permet de changer le propriétaire du fichier (dans System V le propriétaire peut donner ses fichiers aux autres, dans BSD uniquement `root` a le droit de le faire) ;

`chgrp(1)` qui permet de changer le groupe associé au fichier ;

`chmod(1)` qui permet de changer les droits associés au fichier.

Les opérations `chown(1)` et `chgrp(1)` ne changent pas les droits sauf les bits spéciaux qui sont mis à zéro. L'annulation des bits spéciaux est nécessaire pour éviter qu'un utilisateur crée un programme SUID et le donne à quelqu'un d'autre, par exemple `root`.

### 3.5.4 Discussion

Dans un modèle de contrôle d'accès discrétionnaire, la confiance dans les utilisateurs est totale. Ceci nécessite des utilisateurs responsables et compétents — parce qu'il faut que chacun d'eux connaisse la politique de sécurité globale et la respecte.
Le contrôle logique du système ne permet pas de garantir certaines propriétés du modèle. L'impossibilité d'assurer le confinement en est un bon exemple :

1. si $s_1$ est un sujet s'exécutant pour le compte de l'utilisateur $u_1$ propriétaire du fichier $f_1$, il peut donner au sujet $s_2$ (s'exécutant pour $u_2$) le droit de lecture sur $f_1$.
2. $s_2$ peut créer un fichier $f_2$ sur lequel il peut donner le droit de lecture à $s_3$ (s'exécutant pour $u_3$).
3. $s_2$ peut recopier $f_1$ dans $f_2$ pour divulguer les informations de $f_1$ à $s_3$ à l'insu du propriétaire $s_1$.

## 3.6 Modèle de contrôle d'accès basé sur les rôles

La notion de propriétaire de l'information dans les modèles discrétionnaires est artificielle parce que les données n'appartiennent normalement pas à une personne, mais à une entreprise ou une organisation gouvernementale. Le droit d'accès aux données est normalement confié au personnel en raison de leur travail.
Le modèle de contrôle d'accès basé sur les rôles reflète ce fait par la séparation entre l'identité d'un utilisateur et le rôle qu'il joue dans l'organisation. Les droits d'accès aux données sont attribués aux rôles, alors que le seul privilège associé à un utilisateur est celui de pouvoir jouer un ou plusieurs rôles dans l'organisation.
Une motivation pour ce type de modèle dans les systèmes coopératifs a été décrite par Coulouris et Dollimore [Coulouris94a, Coulouris94b]. Ils montrent comment l'identification des rôles facilite l'attribution des droits d'accès pour deux cas précis : la préparation d'un examen universitaire et la préparation des informations d'un cours donné à l'Université de Londres.
Le modèle que nous allons décrire ici a été développé au sein du "National Institute of Standards and Technology" (NIST) aux États Unis [Ferraiolo92, Ferraiolo99]. Le modèle n'impose pas de mécanisme de protection particulier, mais il se base sur les notions d'un rôle actif, une liste de rôles autorisés, les transactions[6] autorisées par rôle et un prédicat *exec*$(s$ : *sujet*$, t$ : *transaction*$)$ qui est vrai uniquement si le sujet $s$ possède le droit d'exécuter la transaction $t$.

**Le rôle actif**

Le rôle actif est le rôle couramment utilisé par le sujet

$$AR(s \text{ : sujet}) = \text{le rôle actif du sujet } s$$

**Les rôles autorisés**

Les rôles autorisés sont stockés sous forme d'une liste de rôles différents que le sujet peut assumer

$$RA(s \text{ : sujet}) = \{\text{les rôles que le sujet } s \text{ peut assumer}\}$$

**Les transactions autorisées par rôle**

Chaque rôle est autorisé à exécuter une ou plusieurs transactions bien formées (cf. 3.3). Remarquons que l'autorisation n'est plus associée au sujet mais au rôle actif du sujet.

$$TA(\{r \text{ : rôle}\}) = \{\text{les transactions autorisées pour tous les rôles dans } \{r \text{ : rôle}\}\}$$

Le modèle définit les trois règles suivantes :

**Attribution des rôles**

Un sujet est autorisé à exécuter une transaction uniquement s'il a choisi ou a reçu un rôle :

$$\forall s \text{ : sujet}, \forall t \text{ : transaction} \mid exec(s,t) \Rightarrow AR(s) \neq \emptyset$$

[6] La notion de transaction utilisée ici inclut l'identité des données et la transformation sur ces données.

La procédure d'identification n'est pas considérée comme une transaction. Toutes les autres activités d'un utilisateur doivent être des transactions bien formées. Un utilisateur ne peut donc pas être actif sans assumer un rôle.
Le problème d'attribution des rôles (quels rôles un sujet est autorisé à assumer) est un problème important dans ce modèle, donc différentes extensions au modèle ont été proposées pour résoudre ce problème, par exemple le modèles de contrôle d'accès basé sur les tâches [Thomas94]. Ces modèles ont été proposés comme des modèles indépendants mais ils semblent reposer sur un modèle de contrôle d'accès basé sur les rôles de base et servent principalement comme aide à l'identification des rôles et l'attribution des droits d'accès aux rôles.

**Autorisation pour un rôle**
Le rôle actif d'un sujet doit être autorisé pour le sujet :

$$\forall s \ : \textit{sujet} \ \mid \textit{AR}(s) \subseteq \textit{RA}(s)$$

Avec la règle d'attribution des rôles, cette règle garantit qu'un utilisateur n'arrive pas à assumer un rôle pour lequel il n'a pas de droit.

**Autorisation pour une transaction**
Un sujet peut uniquement exécuter une transactions si son rôle actif est autorisé pour l'exécuter :

$$\forall s \ : \textit{sujet}, \forall t \ : \textit{transaction} \ \mid \textit{exec}(s,t) \Rightarrow t \in \textit{TA}(\textit{RA}(s))$$

Avec les deux autres règles, cette règle garantit qu'un sujet peut uniquement exécuter une transaction pour laquelle il a été autorisé.
L'"uniquement si" dans la définition ci–dessus permet d'imposer des restrictions additionnelles, par exemple certaines transactions peuvent uniquement être autorisées pendant la journée (pour éviter que la femme de ménage exploite un terminal qui a été laissé connecté).

### 3.6.1 Discussion

Le contrôle d'accès basé sur les rôles suppose que toutes les manipulations sur les données ont la forme d'une transaction bien formée. Les politiques de contrôle d'accès ne considèrent pas l'identité du sujet mais uniquement son rôle actif.
La séparation entre l'identité du sujet et ses droits d'accès dans le système offre plusieurs avantages. Par exemple l'expérience de Coulouris et Dollimore montre que l'abstraction du rôle facilite l'analyse du système et l'attribution des droits d'accès.
Le rôle constitue un niveau d'indirection supplémentaire entre les utilisateurs et les données. Cette indirection facilite la gestion du personnel d'une entreprise. Regardons brièvement l'exemple d'un conseiller de banque. Chaque conseiller a un certain nombre de clients et il a le droit d'inspecter leurs comptes, accorder les découverts, etc. Quand un conseiller de banque est promu, il n'est plus responsable de la totalité de ses clients (il peut quelquefois garder quelques clients importants) donc il ne doit plus s'occuper d'eux et par conséquent perdre l'autorisation de manipuler les données les concernant. Dans les modèles qui attribuent les droits d'accès aux utilisateurs, cela implique le parcours de tous les clients et tous leurs comptes pour retirer le droit de manipuler ces comptes pour le conseiller. Dans le modèle de

contrôle basé sur les rôles, il suffit de retirer le droit d'assumer le rôle de conseiller pour ces clients. Le rôle présente donc une vue logique sur les données et les droits d'accès.

## 3.7 Modèle de la muraille de Chine

Le modèle de la muraille de Chine [Brewer89] a été développé pour résoudre les problèmes de conflit d'intérêt dans les sociétés de service. Il est normal pour un consultant d'avoir accès à des informations confidentielles. Pour éviter qu'un consultant puisse en même temps accéder à des informations de deux entreprises concurrentes il faut une étanchéité complète entre les consultants qui travaillent pour l'une et les consultants qui travaillent pour l'autre (cette propriété a donné son nom au modèle).
Dans ce modèle la politique de contrôle d'accès ne dépend pas de l'identité du consultant — la société de service doit avoir confiance en tous ses consultants — mais uniquement de l'histoire des accès aux informations confidentielles par le consultant.
Les informations sont divisées en *classes de conflit d'intérêt*. Les informations dans la même classe de conflit d'intérêt concernent les entreprises dans le même domaine. Un consultant peut donc accéder à de l'information concernant des entreprises de deux classes de conflit d'intérêt différentes, mais jamais de deux entreprises dans la même classe de conflit d'intérêt.
Le modèle est ainsi très dynamique. Un nouveau consultant commence sa carrière sans restrictions sur les informations auxquelles il peut accéder. Au moment où le consultant accède à de l'information concernant une entreprise, toutes les informations concernant d'autres entreprises dans la même classe de conflit d'intérêt sont interdites au consultant. L'interdiction doit persister jusqu'au moment où il n'y a plus de risque de conflit d'intérêt — la durée de l'interdiction dépend donc du domaine d'affaire et la durée de vie d'un secret commercial.
La structure du modèle est illustré sur la figure 3.3.

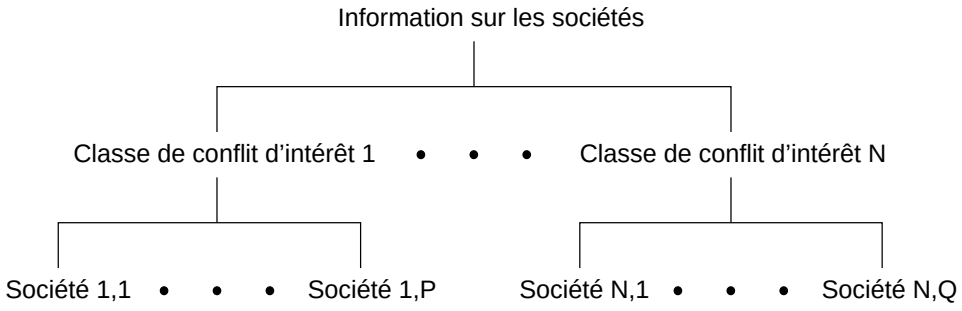



FIG. 3.3: Le modèle de la muraille de Chine

La figure montre la base de données d'une société de service qui gère des informations sur les différents clients. Les clients sont divisés en N classes de conflit d'intérêt différentes.
Avec l'introduction d'une classe de conflit d'intérêt supérieure (qui rassemble toutes les feuilles de l'arbre), le modèle peut être décrit par un treillis [Sandhu93]. Cette classe de conflit d'intérêt combine tous les conflits possibles et imaginables du système, donc aucun utilisateur normal ne doit être associé à cette classe. La classe de conflit d'intérêt maximale peut servir pour l'administration du système (comme le rôle de `root` d'Unix). Elle permet par exemple au même sujet de faire une sauvegarde de tous les fichiers sur disque.

## 3.8 Mécanismes de contrôle d'accès

Tous les modèles décrits dans ce chapitre se basent sur la matrice de contrôle d'accès. Le système doit donc représenter cette matrice d'une façon qui peut être utilisée par le moniteur de référence. Nous allons étudier différentes façons de le faire par la suite.

### 3.8.1 Matrice de contrôle d'accès

La matrice de contrôle d'accès décrite auparavant peut être stockée dans sa totalité par le moniteur de référence, mais ceci n'est pas très efficace. La plupart des sujets ont uniquement des droits d'accès sur une minorité des objets gérés par le système. La matrice est donc normalement creuse. On peut alors la représenter soit par colonnes (listes de contrôle d'accès) soit par lignes (listes de capacités).

### 3.8.2 Liste de contrôle d'accès

Une liste de contrôle d'accès ("Access Control List" ou "ACL") est associée à chaque ressource. Elle liste tous les sujets (l'identité de l'utilisateur ou du rôle) qui sont autorisés à accéder à la ressource et leurs droits d'accès. Toutes les cellules vides sont supprimées.
Le moniteur de référence autorise l'accès à la ressource si le sujet est listé dans l'ACL de celle–ci. Ceci implique que les ACLs fonctionnent uniquement si l'identité du sujet est connue sur le site qui gère la ressource. Elles ne permettent donc pas la collaboration entre les utilisateurs qui ne se connaissent pas à l'avance comme par exemple les gens qui se rencontre sur l'Internet, ou les agents (utilisateurs ou programmes) mobiles.
L'avantage principal d'une ACL est qu'elle permet facilement de répondre à la question : "qui peut accéder à cette ressource ?" Cela rend l'audit de la sécurité des ressources plus facile et plus sûr. C'est pourquoi les critères d'évaluation de la sécurité prévoient normalement l'utilisation des ACLs.
La gestion et la délégation des droits d'accès posent un problème pour les systèmes de protection basés sur les ACLs. Il faut qu'un sujet ait le droit de modifier l'ACL pour pouvoir déléguer un droit d'accès. Le droit de modifier l'ACL est très puissant, il faut donc limiter l'ensemble des sujets qui le possèdent. Ceci a pour conséquence une politique de protection très statique où les droits d'accès changent peu.
L'efficacité de la vérification du droit d'accès est un problème pour l'utilisation des listes de contrôle d'accès. Dans un système à grande échelle les ACLs peuvent devenir très grandes, ce qui implique une forte consommation de la mémoire pour le stockage et de l'unité centrale pour la recherche des entrées.

### 3.8.3 Liste de capacités

Une liste de capacités est associée à chaque sujet. Elle contient une entrée (la capacité) pour chaque ressource à laquelle le sujet peut accéder. La capacité identifie la ressource et décrit les droits d'accès. Il n'existe pas d'entrée associée à la ressource dans la liste de capacités si le sujet n'a pas le droit de l'utiliser. La possession de la capacité est normalement un critère nécessaire et suffisant pour accéder à la ressource. Il faut donc s'assurer qu'un utilisateur ne peut pas fabriquer des capacités pour les ressources pour lesquelles il n'a pas déjà le droit.

Le moniteur de référence autorise l'accès si la capacité est valable, c'est–à–dire qu'elle n'a pas été fabriquée. La capacité est stockée chez le client et elle n'impose pas de besoin de stockage chez le serveur. Ce stockage chez le client pose un autre problème ; il n'est pas facile de retirer une capacité particulière. Une solution possible est de limiter la durée de vie de la capacité, le sujet doit donc redemander une capacité de temps en temps pour pouvoir continuer à accéder à la ressource. Par exemple, cette solution a été réalisée par Kerberos (cf. 2.8.1) qui exige un renouvellement périodique des tickets.
L'avantage principal des capacités est qu'elles offrent une gestion des droits d'accès très souple et dynamique. Le droit de déléguer une capacité peut être inclus dans la capacité elle–même avec les restrictions nécessaires pour éviter une diffusion générale.
La liste de capacités permet facilement de répondre à la question : "quelles sont les ressources auxquelles un sujet peut accéder ?" Par contre il est normalement difficile de répondre à la question de qui peut accéder à une ressource donnée. Il n'est donc pas aussi simple de vérifier la sûreté d'une ressource qu'avec les ACLs.

### 3.8.4 Discussion

Les capacités offrent une souplesse et un dynamisme de la gestion des droits d'accès supérieur à ceux des ACLs.
Un exemple décrit dans la documentation de SPKI montre bien l'avantage des capacités (la définition d'un certificat d'autorisation de SPKI est indifférenciable de la définition d'une capacité).

> *"Considérons le cas d'un proxy sur une machine pare–feu qui permet l'accès de* `telnet` *et* `ftp` *entre l'Internet et un réseau des machines du Ministère de la Défense. La sensibilité du réseau militaire nécessite un contrôle d'accès strict. . . . (la discussion de l'authentification a été omise) . . . Il faut donc que le pare–feu maintienne une ACL qui liste tous les utilisateurs qui sont autorisés à traverser le pare–feu et le mode d'accès autorisé.*
> *L'ACL elle–même peut devenir très grande et consommer beaucoup de ressources pour son stockage. De plus, il faut contrôler le droit de modifier l'ACL, par exemple à l'aide d'une autre ACL. Il n'est pas possible pour quelqu'un dans l'armée de terre de décider si quelqu'un dans la marine doit avoir un accès. Il faut donc une ACL à deux niveaux pour contrôler l'accès à l'ACL du pare–feu. En réalité il faut probablement une hiérarchie des ACLs, afin de refléter la structure des forces armées.*
> *Sans besoin de stocker des ACLs, le pare–feu peut être réalisé par une machine sans disque dur, par exemple enfermé dans une petite boîte. Par contre, un système qui supporte des ACLs a besoin d'un disque dur disponible pendant la vérification des droits d'accès au pare–feu et pendant la modification des ACLs.*
> *(Description de la délégation des certificats d'autorisation omise)*
> *La structure de la délégation des certificats reflète l'organisation du Ministère de la Défense comme une hiérarchie stockée sur le disque dur du pare–feu, mais la gestion de ces certificats est répartie et ne nécessite aucune représentation de cette structure répliquée sur le pare–feu. La délégation de chaque certificat est simple et bien comprise, donc le logiciel qui la réalise peut également être simple.*" [Ellison99]

La protection par capacités permet de refléter la structure de l'organisation et le modèle de

protection en vigueur en dehors du mécanisme de protection. Ce mécanisme peut donc être beaucoup plus simple, ce qui réduit le risque d'une faute de conception ou de réalisation.

## 3.9 Particularités des applications réparties

Outre les particularités identifiées dans le chapitre 2, les systèmes répartis facilitent la collaboration entre sujets qui ne se connaissent pas à l'avance, par exemple les utilisateurs mobiles qui visitent une localité pendant un temps très court ou les groupes de travail ad hoc composés de gens qui se rencontrent sur l'Internet.
Il n'est pas toujours pratique de créer un compte pour quelqu'un qui visite un site pour un temps très court, par exemple pour faire une présentation ou démontrer un produit. Ce type d'utilisateurs a des besoins de ressources légitimes ; il faut donc trouver un moyen de lui permettre d'accéder aux resources d'un site sans une administration trop lourde.
L'Internet a provoqué la création de beaucoup de groupes de travail composés de gens qui ne se connaissent pas à l'avance, par exemple le groupe qui a créé Linux. La confiance est une propriété qui se forme lentement ; il faut donc un moyen de travailler ensemble sans avoir une confiance absolue dans ses collaborateurs.
Ces deux conditions sont particulières aux systèmes qui dépassent le domaine d'administration d'un seule site. Elles se résument par les deux définitions suivantes :

### 3.9.1 Suspicion mutuelle

Un système centralisé permet d'authentifier tous les usagers et de contrôler leur accès aux ressources. Il est donc possible de définir une politique de sécurité globale.
Les systèmes répartis se composent d'un groupement de machines gérées dans des domaines administratif différents, par exemple des machines gérées par différentes filières d'une même entreprise. La taille et la complexité d'un tel système sont beaucoup plus grandes que dans un système centralisé et la définition d'une politique de sécurité globale n'est pas toujours possible.
Pour répondre à ce problème, les différents sites peuvent être classifiés selon le mode d'interaction avec le site. Nous avons identifiés trois classes d'interaction entre deux sites (cf. figure 3.4) : la confiance absolue, la suspicion mutuelle et l'hostilité ouverte.

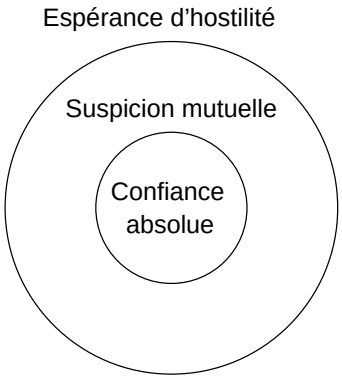



FIG. 3.4: Les différentes classes d'environnements

La confiance absolue existe en général entre différents sites dans le même domaine administratif et l'hostilité ouverte est l'espérance d'interactions prudentes entre machines connectées à l'Internet. La suspicion mutuelle couvre le spectre entre la confiance absolue et l'hostilité ouverte. Elle permet d'ouvrir le modèle de confiance d'un site et de collaborer avec quelqu'un sans lui faire totalement confiance. Ainsi, la suspicion mutuelle permet de limiter les dégâts au cas d'un abus de confiance.

### 3.9.2 Contrôle d'accès des sujets non–identifiés

Le web et le commerce électronique nécessitent un système ouvert, qui permet un accès restreint aux utilisateurs non–identifiés ou semi–anonymes, par exemple pour consulter une catalogue en ligne ou commander un produit sur le site web d'une entreprise. De plus, une entreprise peut permettre à ses meilleurs clients de consulter les nouveautés ou le nombre de produits en stock. Cette ouverture informatique devient de plus en plus un paramètre de compétitivité important. Il faut donc un moyen d'ouvrir le système aux clients sans leur faire une confiance absolue, c'est–à–dire la suspicion mutuelle.
Pour pouvoir réaliser une politique de suspicion mutuelle, il faut un mécanisme de contrôle d'accès très flexible, qui permet aux utilisateurs non–identifiés de présenter un preuve nécessaire et suffisante de leur droits d'accès à une ressource. Cette preuve doit être indépendante de l'authentification de l'identité des usagers pour faciliter l'interaction entre sites qui n'utilisent pas les mêmes agences d'authentification ("Certification Authority").

### 3.9.3 Discussion

Dans les systèmes d'exploitation répartis, les machines peuvent coopérer avec une confiance absolue. Il est donc la responsabilité du système sous–jacent au système de protection d'assurer l'identité et l'authenticité des sujets et des objets. Il n'y a alors pas de différence visible par rapport aux systèmes centralisés.
Dans les systèmes coopérants, la différence entre sites est explicite. Les problèmes de l'identification, de l'authentification et de l'autorisation des sujets et objets doivent être pris en compte par le modèle de protection et la réalisation du système.

## 3.10 Récapitulation et discussion

Depuis le début des années 1980, la doctrine des politiques mandataires et discrétionnaires — établie dans le contexte des systèmes militaires — a lentement été mise en cause comme base appropriée pour le contrôle d'accès dans les systèmes civils. L'introduction des modèles de contrôle d'accès mandataires — comme les modèles basés sur les rôles ou le modèle de la muraille de Chine — dans le monde des systèmes civils, laisse supposer qu'un mécanisme de contrôle d'accès capable de réaliser une telle politique est nécessaire aujourd'hui.
Nous avons dans ce chapitre étudié un nombre de modèles de protection différents et les mécanismes qui permettent de contrôler l'accès aux ressources. Les mécanismes de contrôle d'accès se basent généralement sur la matrice de contrôle d'accès, soit sous forme des listes de contrôle d'accès, soit sous forme de capacités.
Un système de protection à base des capacités semble mieux répondre aux besoins des applications réparties. Les capacités offrent les avantages suivants :

- Flexibilité de l'évolution des droits d'accès d'une application par délégation des capacités.
- Délégation plus simple et plus sûre parce que le droit de déléguer un droit d'accès n'implique pas le droit de déléguer tous les droits d'accès (comme le droit de modifier une ACL).
- Support pour la suspicion mutuelle par le changement de domaine qui permet d'exporter le droit d'opérer sur les données sans exporter le droit de manipuler les données.
- Contrôle d'accès des usagers non–identifiés ou semi–anonymes.
- Élimination d'une représentation complète de l'organisation dans le moniteur de référence par la gestion répartie des droits d'accès.

Le chapitre suivant présente les systèmes à capacités.

# Chapitre 4

# Protection par capacités

Au cours des chapitres précédents nous avons étudié la sécurité dans les systèmes répartis et plus particulièrement le contrôle d'accès. Dans ce chapitre nous allons étudier le contrôle d'accès par capacités.
Ce chapitre présente d'abord en 4.1 une brève introduction aux systèmes à capacités puis rappelle les définitions des différents types de systèmes à capacités, y compris la définition originale selon Dennis et Van Horn. Ensuite, dans la section 4.3, nous présentons quelques résultats théoriques et proposons une taxinomie des systèmes à capacités. La section 4.4 présente deux systèmes à capacités centralisés, le Cambridge CAP et Hydra, qui ont beaucoup influencé les idées sur les systèmes à capacités. Ensuite nous présentons deux systèmes à capacités plus récents Amoeba et Mach, dans la section 4.5. La conception de ces systèmes montre les différents choix de conception pour un système à capacités. La section 4.6 présente la conception de systèmes à mémoire virtuelle répartie unique (qui nous intéressent dans le cadre de cette thèse) — qui tous basent leur mécanisme de protection sur les capacités. Enfin nous présentons une récapitulation et une discussion dans la section 4.7.

## 4.1 Introduction historique aux systèmes à capacités

Les capacités ou descripteurs ont été utilisées dans les ordinateurs bien avant la première formulation de la notion de capacité. La première définition du concept de capacité a été proposée par Dennis et Van Horn en 1966 [Dennis66]. Cette définition détaillée plus loin est purement abstraite : elle ne repose sur aucun modèle d'exécution particulier, et ne présume en rien des réalisations possibles. Cette définition a ensuite été reprise et utilisée dans différents contextes matériels et logiciels, et a donné lieu a de nombreuses réalisations, notamment la "Chicago Magic Number Machine" [Fabry74] qui était parmi les premiers (sinon le premier) à proposer d'utiliser les capacités comme mécanisme de désignation générale. Le livre "Capability–Based Computer Systems" par Hank Levy [Levy84] présente un panorama historique complet des premiers systèmes à capacités que nous recommandons vivement.
Les capacités ont été utilisées dans le contexte des systèmes centralisés pendant les années 1970 et des systèmes répartis depuis le milieu des années 1980. Même si les critères d'évaluation ont forcé les systèmes commerciaux à adopter les ACLs, les capacités sont encore utilisées dans certains prototypes de recherche. Parmi les exemples les plus récents figurent Grasshopper développé à l'Université de Sydney [Dearle94a, Dearle94b], le J–Kernel développé à l'Université Cornell [Chang98, vonEicken99] et EROS développé à l'Université

de Pennsylvanie [Shapiro99].

## 4.2 Définitions des systèmes à capacités

La définition la plus générale d'une capacité est la suivante : c'est une structure de données qui

1. désigne un objet unique.
2. inclut de l'information supplémentaire pertinente à la sécurité.

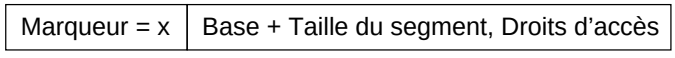
La capacité sert donc en même temps à désigner un objet et à contrôler l'accès à l'objet. Les systèmes à capacités ont normalement les propriétés suivantes :

1. La possession d'une capacité permet au sujet d'accéder à l'objet (avec les droits d'accès inclus dans la capacité).
2. La capacité permet le partage d'un objet — tous les sujets qui possèdent une capacité qui désigne un objet partagent cet objet. Différentes capacités peuvent inclure différents droits d'accès, donc les différents sujets n'ont pas forcément les mêmes droits pour manipuler l'objet.
3. Une capacité doit être impossible à falsifier, pour éviter la construction d'une capacité qui inclut les droits d'accès souhaités par le sujet mais non autorisés par le système.

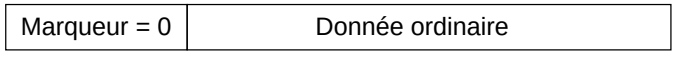
Il existe trois classes de capacités : les *capacités marquées*, les *capacités confinées* et les *capacités chiffrées*. Nous donnons une description très courte de ces classes avant de revenir à la définition originale d'un système à capacités proposée par Dennis et Van Horn.

### 4.2.1 Capacités marquées

Les *capacités marquées* ("tagged capabilities") ou descripteurs sont des variables spéciales marquées par le système. Cette marque est normalement un bit qui indique si le mot en mémoire contient une capacité ou non. Le marquage nécessite donc un support du matériel pour distinguer entre les variables normales et les capacités. De plus, il faut que le système garantisse l'intégrité des capacités, c'est–à–dire empêcher les sujets de modifier une adresse en mémoire qui contient une capacité.




*Structure protégée d'une capacité*




*Structure des capacités à marqueur*

FIG. 4.1: Le format d'une capacité marquée

De nombreux systèmes à capacités marquées ont été réalisés pendant les années 1960 et 1970 ; quelques exemples décrits par Levy sont : le Burroughs B5000, le Chicago Magic Number Machine, le CAL–TSS et le Plessey System 250.

### 4.2.2 Capacités confinées

Les *capacités confinées* (“segregated capabilities” ou “partitioned capabilities”) sont gérées par le système dans une region de la mémoire séparée, hors de portée des applications. Le système empêche ainsi les sujets de modifier une capacité. Le format d’une capacité confinée est illustré sur la figure 4.2.

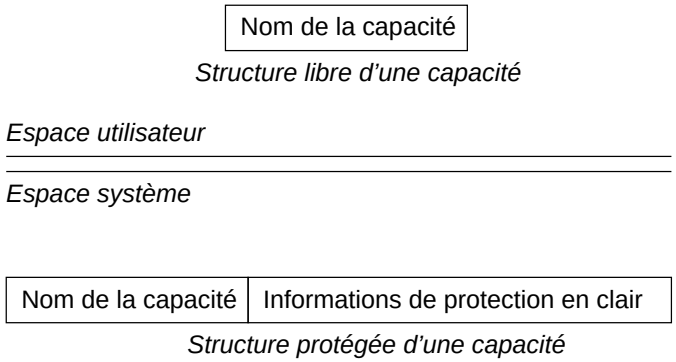



FIG. 4.2: Format d’une capacité confinée

La capacité se compose d’une partie qui se situe dans l’espace de l’utilisateur et qui lui permet de désigner l’objet et une autre partie confinée par le système (dans le noyau ou dans un serveur séparé) qui permet de localiser l’objet et vérifier les droits d’accès inclus dans la capacité.

### 4.2.3 Capacités chiffrées

Les *capacités chiffrées* (“encrypted signature capabilities”, “sparse capabilities” ou “password capabilities”[1]) utilisent le chiffrement pour protéger les droits d’accès inclus dans la capacité ou prouver l’identité de celui qui l’envoie.

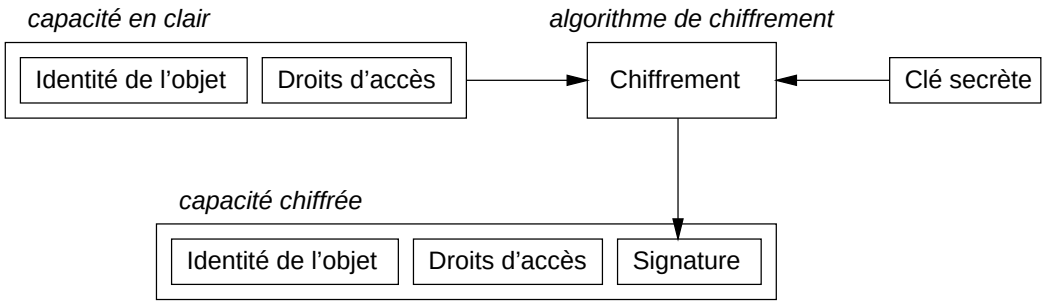



FIG. 4.3: Le format d’une capacité chiffrée

Le système stocke normalement une clé secrète avec l’objet pour pouvoir vérifier les capacités plus tard. Il faut donc connaître cette clé secrète pour faire une signature valide de la capacité.

[1]Le nom “password capability” a été utilisé pour décrire une capacité qui contient un mot de passe en clair [Anderson86]. Cette méthode n’est pas sûre dans un système réparti mais fonctionne bien dans le contexte d’un multi–processeur.

Si les capacités sont gérées par le système, celui–ci peut utiliser le chiffrement symétrique ou une fonction de hachage à sens unique pour signer la capacité. Si les capacités sont gérées par les utilisateurs, le système doit stocker une copie de la clé publique avec l'objet pour pouvoir vérifier la signature faite avec la clé secrète de l'utilisateur. Le système peut stocker plusieurs clés différentes avec l'objet pour pouvoir distinguer les droits accordés à des utilisateurs différents.

### 4.2.4 Définition originale selon Dennis et Van Horn

La définition d'un système à capacités proposée par Dennis et Van Horn se base sur les trois notions suivantes :

le *segment*, qui se définit comme une zone de mémoire contiguë

l'*activité*, qui définit un flot d'exécution quelconque

le *processus*, qui représente un contexte d'exécution, une *sphère de protection* dans laquelle s'exécutent une ou plusieurs activités

La *sphère de protection* définit le contexte de protection de toutes les activités qui s'exécutent dans la sphère. Elle définit l'ensemble des segments accessibles par les activités du processus, l'ensemble des entrées/sorties possibles, et de façon générale, l'ensemble des objets accessibles dans ce contexte.
La capacité se compose d'un nom (une chaîne de bits) unique qui identifie l'objet. Le nom permet de désigner l'objet mais pas de le localiser (le nom est indépendant de l'adresse effective de l'objet).
Les droits d'accès inclus dans la capacité dépendent du type de l'objet. De plus, la capacité indique aussi si son détenteur est propriétaire de l'objet. Le propriétaire possède les droits spéciaux comme par exemple le droit de détruire l'objet.
Une seule *C–liste* (liste de capacités) est définie pour chaque processus. Cette liste peut être partagée entre plusieurs processus. Le modèle permet de définir une hiérarchie de C–listes (et de sphères de protection correspondantes) avec des capacités différentes. Les droits normaux de manipulation des C–listes sont les suivants :

- une activité peut ajouter des capacités dans sa C–liste ou dans la C–liste d'une sphère de protection de niveau inférieure
- une activité peut activer ou arrêter un processus de niveau inférieur
- une activité peut supprimer une capacité dans une C–liste d'une sphère de protection de niveau inférieur

Une capacité d'entrée permet au sujet d'appeler une procédure (appelée une procédure protégée) dans une autre sphère de protection. Ainsi, la procédure protégée permet de changer le contexte de protection. Le processus appelant est suspendu pendant la durée de l'appel.
L'appel de procédure protégée permet également de passer une capacité en paramètre entre l'appelant et l'appelé (la capacité peut être une capacité sur une C–liste).
On retrouve les notions de capacité d'accès, capacités d'entrée et sphère de protection (souvent appelée un domaine de protection) de la définition originale d'un système à capacités selon Dennis et Van Horne dans la plupart des systèmes à capacités.

### 4.2.5 Discussion

Dans tous les types de capacités décrits ci–dessus, le format d'une capacité inclut trois types d'information : la désignation d'un objet, les droits d'accès à cet objet et les droits de manipuler la capacité elle même — par exemple de la copier ou de la déléguer à quelqu'un d'autre.

## 4.3 Quelques résultats théoriques

Les études théoriques des systèmes à capacités permettent d'analyser leurs propriétés et de classifier ces systèmes. Certaines de ces études sont présentées dans la suite.

### 4.3.1 Un modèle formel des systèmes à capacités

Les descriptions des systèmes à capacités sont souvent informelles et elles sont dominées par les détails de la réalisation. Les questions suivantes ne sont normalement pas considérées dans la description d'un système, même si elles sont souvent posées par les utilisateurs.

1. Est–ce que le système limite réellement l'accès aux ressources à ceux qui sont autorisés à y accéder ?
2. Est ce que le système supporte les modes normaux de partage de l'information ?
3. Sous quelles conditions l'information peut–elle être disséminée à l'intérieur du système ?
4. Quelle politique de contrôle d'accès est réalisée par le système ?

Cette observation a poussé Lawrence Snyder à proposer un modèle formel des systèmes à capacités [Snyder81].
Le modèle se base sur un graphe orienté où les nœuds représentent les entités du système (sujets et objets) et les arcs représentent les capacités. Pour distinguer entre sujets et objets, Snyder propose de colorer les nœuds des sujets en noir et ceux des objets en blanc. Les arcs sont annotés avec les droits inclus dans la capacité. Un exemple simple est illustré sur la figure 4.4.

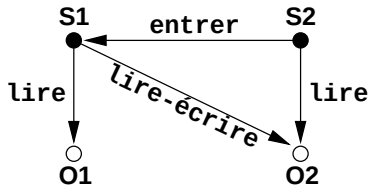



FIG. 4.4: Un système simple à capacités

La figure montre deux sujets S1 et S2 et deux objets O1 et O2. S1 possède une capacité de lecture pour O1 et une autre de lecture–écriture pour O2. S2 possède une capacité de lecture pour O2 et une capacité de changement de domaine vers S1.
Snyder définit également un ensemble des transformations légales du graphe qui correspondent à l'utilisation et à la délégation des capacités. Les transformations légales dépendent du modèle à capacités particulier.
Pour montrer la sûreté d'un système, il faut donc montrer qu'il n'existe pas une séquence de transformations légales qui transforme le graphe d'un état sûr à un état non–sûr.

### 4.3.2 Une taxinomie des systèmes à capacités

Une taxinomie des systèmes à capacités a été développée par Kain et Landwehr [Kain86]. Cette taxinomie permet de décrire des différents modèles (ou réalisations) de capacités dans les systèmes centralisés.

#### Modèle de capacités

Une capacité est une référence système qui désigne un objet et spécifie les droits d'accès accordés à celui qui présente la capacité ; un sujet sans droits sur un objet peut posséder une capacité vide.

#### Description de la taxinomie

La taxinomie est basée sur une analyse des différents événements clés dans la gestion des objets et des capacités. Kain et Landwehr ont identifié cinq questions, auxquelles les réponses servent de base à la taxinomie. Les questions sont les suivantes :

1. que se passe-t-il lors de la création d'un objet ?
2. que se passe-t-il quand les attributs de sécurité d'un objet changent ?
3. que se passe-t-il quand une capacité est copiée ?
4. que se passe-t-il quand une capacité est présentée au système ?
5. que se passe-t-il quand un sujet essaie d'accéder à une ressource protégée ?

Kain et Landwehr prétendent avoir trouvé un ensemble des réponses assez génériques pour pouvoir classifier tous les systèmes à capacités. Cette classification est présentée par la suite.

#### Classification des systèmes

La classification se base sur les réponses données aux questions ci–dessus.

**Question 1 :** À la création d'un objet une référence (capacité) est normalement retournée au créateur. Les droits d'accès du créateur peuvent être inclus dans cette capacité, sinon il faut un mécanisme pour les y inclure. Kain et Landwehr identifient les options suivantes :

a) aucun droit d'accès n'est inclus dans la capacité
b) les droits d'accès sont inclus dans la capacité

Dans le cas (b), les droits d'accès sont normalement tous les droits possibles. Le créateur d'un objet est normalement considéré comme le propriétaire de l'information et doit être autorisé à tout faire.

**Question 2 :** Kain et Landwehr identifient deux cas particuliers qui méritent une réponse séparée. Si la capacité est couramment utilisée pour accéder à un objet, elle ne peut pas être modifiée parce que cela peut invalider une référence et provoquer une erreur à l'exécution. Si la capacité n'est pas utilisée, elle peut être modifiée par le système pour refléter le changement de la classification de l'objet. Quand les attributs de sécurité d'un objet changent, trois cas sont possibles :

a) les droits d'accès ne sont pas modifiés
b) la capacité est marquée pour modification ultérieurement
c) la capacité est modifiée tout de suite.

Le problème de changement de la classification de la confidentialité est particulier au modèle de contrôle d'accès militaire. En réalité il n'est pas l'objet de beaucoup d'attention dans les systèmes à capacités et ne mérite donc pas d'être représenté directement dans une taxinomie. Kain et Landwehr supposent que les capacités doivent être modifiées pour refléter la nouvelle classification de l'objet. En réalité la plupart des réalisations des systèmes à capacités répartis reposent sur une révocation totale des capacités et une nouvelle délégation de capacités pour l'objet. Cette nouvelle délégation peut prendre en compte la nouvelle classification de l'objet.

**Question 3 :** Le modèle de capacités permet à un sujet qui possède les droits d'accès nécessaires de copier une capacité d'un objet vers un autre. Les droits nécessaires sont : le droit de lire l'objet qui contient la capacité et le droit d'écrire l'autre objet. Si l'autre objet est partagé, ceci constitue effectivement une délégation des droits d'accès. Quatre cas sont possibles :

a) les droits d'accès de la copie ne sont pas modifiés
b) les droits d'accès de la copie sont limités par une politique par défaut du système
c) les droits d'accès de la copie sont le maximum permis pour le sujet qui reçoit la capacité selon la politique de sécurité
d) les droits d'accès de la copie sont modifiés par les procédures de confiance ("trusted software")

Une politique par défaut du système peut être de permettre uniquement les copies avec des droits d'accès inférieurs à ceux de l'original. Cela implique une limite sur les délégations possibles d'une capacité.

**Question 4 :** L'utilisation d'une capacité est divisée en deux phases, la préparation et l'utilisation. Dans la phase de préparation le sujet déclare son intention d'utiliser la capacité ; par exemple il peut la charger dans un registre de l'unité centrale dédié aux capacités. Cette phase de préparation permet au système de vérifier la validité de la capacité avant l'utilisation. Trois cas sont possibles :

a) le système accorde tous les droits inclus dans la capacité
b) le système limite les droits d'accès accordés selon une politique par défaut du système
c) Le système limite les droits d'accès accordés selon la politique de sécurité en vigueur.

**Question 5 :** Quand un sujet essaie d'accéder à une ressource protégée, trois cas sont possibles :

a) aucune vérification n'est faite
b) l'opération est comparée avec les opérations permises par la capacité
c) les droits d'accès maximaux sont calculés comme fonction de la capacité et les attributs de sécurité du sujet et de l'objet ; si l'opération est incluse dans ces droits maximaux, elle peut se dérouler normalement

#### Discussion

La taxinomie de Kain et Landwehr est conçue pour classifier les systèmes de capacités centralisés. Cette classification n'est pas applicable dans le contexte des systèmes répartis.
La question 2 suppose que le système peut identifier et modifier toutes les capacités qui référencent un objet quand les attributs de sécurité de l'objet changent. Dans beaucoup d'architectures à capacités réparties, les capacités sont réalisées par des structures de données normales, indistinguables d'autres données. Ceci est par exemple le cas dans la plupart des systèmes à capacités chiffrées.
La séparation de l'utilisation de la capacité en préparation et accès permet un contrôle d'accès plus efficace dans les systèmes centralisés et les systèmes d'exploitation répartis. La préparation permet au système de vérifier la capacité et de mettre en place des structures qui décrivent les droits d'accès du sujet pour l'objet. Il suffit donc pour les contrôles d'accès suivants de vérifier si le mode d'accès correspond avec des structures mises en place pour le sujet et l'objet. L'opération sur un fichier `open(2)` d'Unix en est un bon exemple. L'appel d'`open(2)` vérifie les droits du sujet sur le fichier et met en place une entrée pour le fichier dans le tableau des fichiers du processus. Toutes les opérations suivantes se font au travers de cette entrée dans le tableau des fichiers.
Un autre problème de cette taxinomie est qu'elle suppose des ACL au niveau des objets, qui peuvent être utilisées par les politiques de sécurité. Le rôle de ces ACL n'est pas bien défini, et leur utilisation peut cacher des aspects importants du système. Par exemple l'utilisation des ACLs semble être le seul moyen de révoquer les capacités. L'impossibilité de classifier la révocation des capacités est donc un autre problème de cette taxinomie.

### 4.3.3 Une nouvelle taxinomie des systèmes à capacités répartis

Les problèmes de la taxinomie identifiés ci–dessus nous ont conduit à proposer une nouvelle taxinomie pour les systèmes à capacités répartis. Cette taxinomie est décrite dans la suivante.

#### Modèle de capacités

Nous proposons d'utiliser le même modèle à capacités que Kain et Landwehr, c'est–à–dire qu'une capacité est une référence système qui désigne un objet et spécifie les droits d'accès accordés à celui qui présente la capacité, un domaine de protection définit l'ensemble de capacités à la disposition d'un sujet et une capacité de changement de domaine permet de passer d'un domaine à un autre de manière contrôlée.

#### Description de la taxinomie

Nous suivons la même démarche que Kain et Landwehr, c'est–à–dire que nous allons identifier les événements les plus importants pour la gestion des capacités.
Dans notre taxinomie, chaque classification doit être motivée par une description courte du modèle de protection mis en œuvre par le système.
Les événements identifiés sont les suivants :

1. que se passe-t-il à la création d'un objet ?
2. comment peut-on manipuler les capacités ?

3. comment peut-on réaliser un changement temporaire des droits d'accès d'un sujet ?
4. comment peut-on déléguer une capacité ?
5. comment peut-on révoquer une capacité ?
6. comment sont vérifiées les capacités ?
7. que se passe-t-il quand un sujet essaie d'accéder à une ressource protégée ?

**Classification des systèmes**

**Création d'un objet :** Nous proposons de garder les deux classes définies par Kain et Landwehr :

a) aucun droit d'accès est inclus dans la capacité
b) les droits d'accès sont inclus dans la capacité

**Manipulation des capacités :** Les systèmes autorisent normalement les sujets à manipuler les capacités, par exemple à les copier et à les transférer entre eux. Dans le cas des capacités confinées, toute manipulation nécessite l'intervention du système. L'intervention du système n'est pas toujours nécessaire dans le cas des capacités publiques, par exemple les clients et serveurs peuvent signer et vérifier les capacités eux–mêmes. L'utilisation des capacités publiques n'exclut pas l'intervention du système (par exemple le stockage et vérification des capacités peuvent être faits au niveau de l'utilisateur) mais la création et la modification des capacités peuvent nécessiter l'intervention du système. Un point très important dans ce contexte est le transfert des capacités. Si le transfert se fait sans intervention du système, il va être difficile de réaliser une politique de protection mandataire[2].
Nous avons identifié deux classes de comportement différentes pour la manipulation des capacités :

a) les capacités peuvent être manipulées sans intervention du système
b) l'intervention du système est nécessaire pour la manipulation des capacités

**Changement des droits d'accès :** Le changement temporaire des droits d'accès correspond effectivement à un changement du domaine de protection du sujet (ou simplement un changement de domaine). Le changement de domaine a pour but de permettre une amplification ou une restriction temporaire des droits d'accès du sujet pendant l'exécution des opérations sensibles. Le changement de domaine est normalement associé à certaines opérations sur un objet ; ces opérations sont normalement appelées des procédures protégées. Le système permet au sujet de changer ses droits d'accès pendant la durée de l'appel d'une procédure protégée. Le changement du domaine peut être réalisé de trois façons différentes :

a) l'*extension* temporaire des droits d'accès du sujet
b) la *restriction* temporaire des droits d'accès du sujet

[2]La réalisation d'une politique mandataire n'est pas impossible à réaliser même si le système n'impose pas de limites au niveau de transfert des capacités. Il faut dans ce cas que le système limite les copies des capacités, par exemple par le chiffrement qui rend la capacité inutilisable pour quelqu'un qui ne doit pas avoir le droit d'accéder aux données.

c) l'*isolation* de l'opération dans un domaine de protection séparé pendant l'exécution de l'opération

Appelons la C–liste courante du sujet $C_{\text{actuel}}$, la C–liste normale du sujet $C_{\text{normal}}$ et supposons que les droits temporaires puissent être décrits par une C–liste $C_{\text{temporaire}}$.
Le changement de domaine peut donc être décrit par les formules suivantes :

a) $C_{\text{actuel}} = C_{\text{normal}} \cup C_{\text{temporaire}}$
b) $C_{\text{actuel}} = C_{\text{normal}} \cap C_{\text{temporaire}}$
c) $C_{\text{actuel}} = C_{\text{temporaire}}$[3]

La plupart des systèmes permettent de passer quelques droits d'accès supplémentaires au domaine isolé dans l'appel de la procédure protégée.

**Délégation :** La délégation est une généralisation de la copie d'une capacité utilisée dans la taxinomie de Kain et Landwehr.

a) Le transfert de la capacité. Ceci implique que la capacité soit supprimée dans le domaine de protection d'origine et insérée dans le domaine de protection qui la reçoit.
b) La copie de la capacité. Ceci implique que la capacité soit copiée du domaine de protection d'origine et insérée dans le domaine de protection qui la reçoit.

Dans les deux cas le système peut imposer les restrictions proposées par Kain et Landwehr sur le droit d'accès inclus dans la capacité et sur le droit de manipuler la capacité elle–même.

**Révocation :** La révocation est un point difficile de tous les systèmes à capacités et en particulier des systèmes à capacités chiffrées. Le système n'a que rarement les moyens nécessaires pour identifier et supprimer une capacité dans la mémoire d'un processus. La révocation est donc normalement faite au moment de la vérification de la capacité.
Nous avons identifié trois classes de révocation différentes :

a) révocation en bloc de toutes les capacités pour un objet
b) révocation sélective chez le sujet
c) révocation sélective chez l'objet

La révocation en bloc est la plus facile à réaliser. Lors de la création, un mot de passe (une chaîne de caractères ou un numéro de version de l'objet) associé à l'objet est inclus dans la capacité. La vérification de la capacité doit donc comparer le mot de passe inclus dans la capacité avec celui stocké avec l'objet ; s'il y a une différence la capacité n'est plus valable. Il suffit donc de changer ce mot de passe stocké (augmenter la version de l'objet) pour révoquer toutes les capacités qui référencent l'objet. Il faut donc que les sujets qui doivent garder leurs droits d'accès à l'objet obtiennent une nouvelle capacité pour pouvoir accéder à l'objet.
La révocation de la capacité chez le sujet n'est pas toujours possible. Elle nécessite que le site qui gère l'objet arrive à inspecter et supprimer d'une façon fiable toutes les capacités stockées sur le système. Les systèmes d'exploitation répartis qui utilisent les capacités confinées sont un cas spécial qui permet cette opération.

[3] La définition de l'isolation donnée ici ne correspond pas forcement au confinement (cf. 2.3), par exemple on n'impose pas que $C_{\text{actuel}} \cap C_{\text{temporaire}} = \emptyset$.

La dernière possibilité est de stocker avec l'objet une liste de révocation des capacités (cette liste est appelée une "capability revocation list" ou "CRL" dans le contexte de SPKI [Ellison99]). Il suffit d'inclure la capacité dans cette liste pour la révoquer. La CRL marche donc comme une ACL négative. Cette solution n'est normalement pas très satisfaisante parce qu'elle souffre des mêmes inconvénients que les ACLs. La taille de la CRL augmente avec chaque capacité révoquée ; une entrée ne peut jamais être supprimée, parce que cela rend à nouveau la capacité valable.

**Vérification :** Quand un sujet présente une capacité au système, ce dernier va d'abord vérifier la validité de la capacité puis les droits d'accès spécifiés par la capacité vont être accordés au sujet. Les actions possibles sont essentiellement celles identifiées par Kain et Landwehr pour la préparation d'une capacité, c'est–à–dire :

a) le système accorde tous les droits inclus dans la capacité
b) le système limite les droits d'accès accordés selon une politique par défaut du système
c) Le système limite les droits d'accès accordés selon la politique de sécurité en vigueur.

**Accès aux objets :** Nous proposons de garder les trois classes définies par Kain et Landwehr :

a) aucune vérification est faite
b) l'opération est comparée avec les opérations permises par la capacité
c) les droits d'accès maximaux sont calculés comme fonction de la capacité et des attributs de sécurité du sujet et de l'objet ; si l'opération est inclus dans ces droits maximaux elle peut se dérouler normalement

La séparation entre vérification et contrôle d'accès a été introduite par Kain et Landwehr pour décrire un certain nombre d'architectures de capacités centralisées, où il faut charger la capacité dans un registre spécial avant de l'utiliser. Cette distinction n'existe normalement pas dans un système réparti, parce qu'il n'existe pas de support matériel pour les capacités (les capacités sont normalement réalisées par logiciel). La séparation reste toujours intéressante parce qu'elle permet de distinguer entre la vérification initiale de la capacité et les utilisations suivantes. Le résultat de la vérification peut être la construction d'un canal de communication sûr (pour transférer l'information) entre le site du sujet et le site de l'objet. Tous les messages qui arrivent par ce canal correspondent à la capacité déjà vérifiée et peuvent être soumis à un contrôle d'accès rudimentaire (le contrôle d'accès aux objets proposé ici).

**Discussion**

L'utilité de notre taxinomie est d'abord de focaliser notre attention sur les événements les plus importants d'un système à capacités. Nous ne prétendons pas avoir prouvé que la taxinomie est complète ni pouvoir utiliser cette taxinomie pour faire des preuves formelles des caractéristiques d'un système à capacités. Ces preuves sortent du contexte de cette thèse. Un résultat facile à vérifier est que les systèmes classés " ?a ? ? ? ? ?" (toute classe peut être substituée à un ' ?') sont incapables de réaliser une politique de protection mandataire (outre la politique triviale qui interdit la communication entre sujets). Si l'intervention du système n'est pas nécessaire pour les manipulations des capacités, le système ne peut pas imposer de

restrictions sur la délégation des capacités. Le système est donc incapable d'éviter une violation de la politique de protection par un sujet qui donne une capacité à quelqu'un qui ne doit pas avoir l'accès correspondants.

### 4.3.4 Capacités augmentées

Les capacités augmentées ont été proposées par Karger et Herbert [Karger84] pour pouvoir réaliser une politique de contrôle d'accès mandataire, par exemple celle de Bell et LaPadula[4]. Le système se base sur le modèle à capacités classique avec les domaines de protection, les capacités d'accès et les capacités de changement de domaine. Les capacites sont vérifié avant l'utilisation et les capacités verifiées sont mises dans un cache de capacités. Le système définit un domaine de protection particulier (un moniteur de référence — "Security Kernel Domain") qui intervient au moment de la vérification d'une capacité, par exemple provoqué par une interruption logicielle quand le système charge une capacité dans le cache. Ce domaine de protection spécial doit s'exécuter sans restrictions pour pouvoir vérifier que toutes les utilisations sont conformes au modèle de contrôle d'accès. Remarquons que les restrictions imposées sur l'utilisation des capacités doivent être faites dynamiquement ; un domaine de protection peut donc posséder des capacités qu'il n'a pas le droit d'utiliser dans son contexte courant.
Pour pouvoir réaliser une politique de contrôle d'accès mandataire, le système classifie tous les sujets et les objets, par exemple selon la classification militaire (cf. tableau 3.1). Au moment du chargement de la capacité dans le cache des capacités, le système vérifie que les droits d'accès inclus dans la capacité sont conformes au modèle. Si les droits sont trop faibles par rapport aux droits requis, le système retourne un message d'erreur et si les droits sont trop élevés, le système limite les droits à ceux permis par le modèle (cf. figure 4.5).

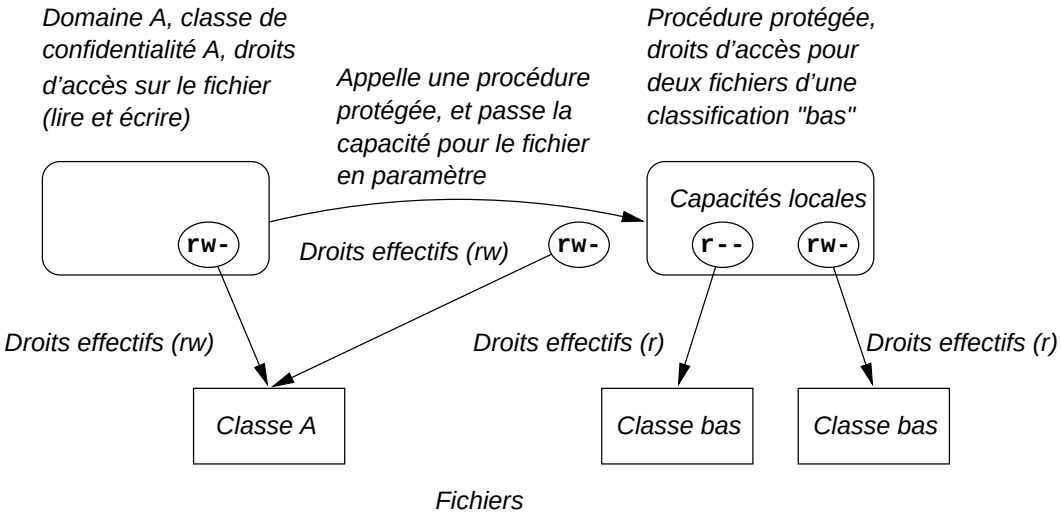



FIG. 4.5: Contrôle d'accès mandataire

La figure montre un domaine de protection qui s'exécute avec une classification de

[4]Le système proposé inclut également une cominaison des capacités et les ACL pour permettre un contrôle d'accès discrétionnaire. Nous ne détaillons pas cette extension parce qu'elle est moins générale que le modèle de capacités d'Angel (cf. 4.6.3) décrit plus loin.

confidentialité A ; classe bas $<$ classe A $<$ classe haut. Ce domaine appelle une procédure protégée qui possède des droits locaux sur deux fichiers dans la classe bas, y compris le droit d'écrire un de ces fichiers. L'écriture vers le bas n'est pas permise selon le modèle de Bell et LaPadula, le domaine de moniteur de référence va donc limiter les droits effectifs à la lecture seule.
La possession d'une capacité n'est donc plus une condition nécessaire et suffisante pour accéder à la ressource (elle n'est plus suffisante), il faut que l'utilisation de la capacité soit autorisée par la politique de sécurité.

### 4.3.5 ICAP — l'identité incluse dans la capacité

Le système ICAP est un système à capacité proposé par Li Gong [Gong89a, Gong89b] qui permet de surveiller, coordonner et enregistrer la propagation des capacités et réaliser une politique de contrôle d'accès mandataire.
ICAP utilise un format de capacité chiffrée où l'identité de celui qui est autorisé à utiliser la capacité est chiffrée avec les droits d'accès dans la signature. De plus il faut que la signature soit faite par un serveur d'autorisation ("access control server" ou "ACS"), l'ACS n'est pas forcement identique au serveur de l'objet (celui qui gère l'objet).
Quand un objet est créé par un serveur d'objets, ce serveur va envoyer le mot de passe de l'objet à l'ACS qui l'enregistre. Ce mot de passe peut ensuite être utilisé par l'ACS pour créer des nouvelles capacités.
Pour déléguer une capacité d'un sujet $s_1$ à un sujet $s_2$, il faut que $s_1$ envoie une requête au serveur pour lui demander de construire une capacité pour $s_2$. La requête contient la capacité et l'identité de $s_2$. L'ACS vérifie que la délégation est permise selon la politique de sécurité, puis il construit une nouvelle capacité uniquement utilisable par $s_2$.
La vérification de la délégation des capacités par l'ACS permet par exemple d'assurer une politique de contrôle d'accès mandataire. La politique de sécurité est donc vérifiée quand une capacité est déléguée au lieu d'être vérifiée au moment de l'utilisation comme dans les capacités augmentées. Ce choix est fait parce que :

> *"Notre intuition nous dit que le nombre de délégations est normalement beaucoup moins grand que le nombre d'utilisations. Il semble donc plus efficace de vérifier la capacité au moment de la délégation au lieu de la vérifier au moment de l'utilisation"* [Gong89a].

Dans ICAP la capacité est une condition nécessaire et suffisante pour accéder à la ressource si le sujet peut prouver qu'il est bien celui inclut dans la capacité. Cette vérification est beaucoup plus simple à réaliser dans un système réparti que la vérification de la classification des sujets et des objets nécessaire dans le contexte des capacités augmentées.

### 4.3.6 Capacités actives

Les capacités actives ont été proposées par le "System Software Research Group" à l'université d'Illinois à Urbana–Champaign [Tock94, Campbell96] pour répondre au problème de contrôle d'accès dans un réseau ouvert à grande échelle, par exemple l'Internet. Ce type de réseau se caractérise, entre autre, par une augmentation continuelle du nombre de protocoles et des types de services. Il faut donc que la sémantique des droits d'accès inclus dans la capacité puisse évoluer pour permettre des nouveaux modes d'accès.

L'idée générale d'une capacité active est de remplacer les droits d'accès traditionnels par un script de vérification des droits d'accès. Le sujet qui souhaite accéder à une ressource protégée présente une capacité active au système. Le système passe la capacité au gestionnaire de sécurité ("Security Manager") qui vérifie d'abord l'intégrité de la capacité active par sa signature digitale avant d'interpréter le script pour retirer les droits d'accès inclus dans la capacité. Après cette vérification le système passe l'identificateur de l'objet et l'opération au serveur qui gère l'objet ("Object Manager"). La structure du système est présentée sur la figure 4.6.

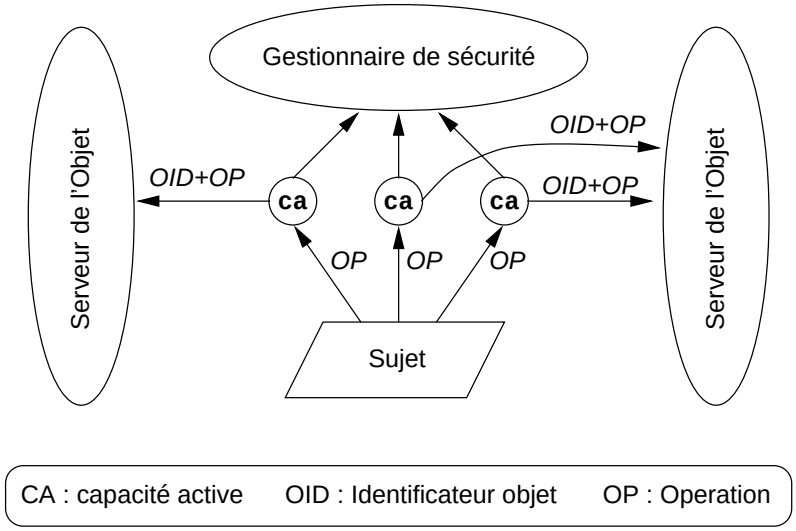



FIG. 4.6: Le modèle des capacités actives

La figure montre un sujet qui possède des capacités actives pour trois objets gérés par deux serveurs différents. Le gestionnaire de sécurité vérifie ces capacités avant de passer l'OID et l'opération au serveur qui gère l'objet.
Un avantage de cette approche est qu'elle permet d'exprimer une politique de contrôle d'accès plus complexe, par exemple interdire l'accès au serveur si la charge est trop élevée, restreindre l'accès aux heures de travail, etc. Ces politiques sont toutes difficiles à exprimer avec les capacités traditionnelles.

## 4.4 Systèmes centralisés

Les systèmes à capacités centralisés offrent une plate–forme commune aux toutes les applications. Les capacités reposent souvent sur un support matériel (capacités marquées) ou un support du noyau (capacités confinées).

**Remarque !** Un système centralisé permet de vérifier une capacité avant la première utilisation, puis associer les droits d'accès inclus dans la capacité au processus qui a présenté la capacité. Ceci permet d'amortir le coût de la vérification sur toutes utilisations de la capacité.

Nous présentons un système qui profite d'un support matériel pour la gestion des capacités et un système de capacités confinées dans la suite.

### 4.4.1 Cambridge CAP

L'ordinateur Cambridge CAP a été construit à l'Université de Cambridge en Angleterre au début des années 1970 [Needham77]. Un objectif de la conception de ce système était de rendre la manipulation des capacités transparente aux programmeurs, c'est–à–dire que la capacité sert de désignation et que le chargement et la vérification des segments de capacités se font de manière transparente.

#### Description du système de protection de Cambridge CAP

La conception du système suit le modèle à capacités proposé par Dennis et Van Horn. Le processus constitue le contexte d'exécution et de protection (la sphère de protection).
Les capacités contrôlent l'accès aux segments. Les capacités sont confinées dans des segments de mémoire d'un type particulier. La notion de segment de capacités correspond à la C–liste dans le modèle de Dennis et Van Horn.
Le système CAP fournit 16 registres de segment qui permet à un processus d'utiliser les capacités stockées dans 16 segments à la fois.
Les processus sont organisés selon une arborescence où chaque processus contrôle tous ses sous–processus. La racine de cette arborescence est une entité système qui s'appelle le coordinateur maître ("master coordinator" ou "MC").
Le contexte de chaque processus contient une liste de toutes les ressources disponibles au processus ("process resource list" ou "PRL") ; cette liste n'est pas modifiable par le processus. Les entrées de la PRL d'un processus pointent vers des capacités dans le processus qui l'a créé ; un processus ne peut donc jamais posséder une capacité qui n'est pas disponible à son parent. Une capacité pointe vers une entrée dans la PRL, qui pointe à son tour vers une capacité dans le processus père et ainsi transitivement jusqu'au MC. La liste des ressources du maître ("master resource list" ou "MRL") donne accès aux ressources physiques. Ainsi les capacités n'ont qu'une signification locale et ne peuvent pas être transférées entre processus.
L'appel des procédures protégées permet d'amplifier ou de dégrader les droits d'accès pendant l'exécution de la procédure. Les droits d'accès d'une procédure protégée dépendent de la procédure et l'identité de l'appelant et de l'appelé. Les procédures protégées sont ainsi utilisées pour contrôler l'accès aux ressources systèmes comme par exemple les entrées/sorties.

#### Format des capacités

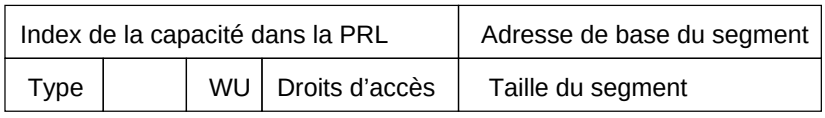



FIG. 4.7: Le format des capacités de CAP

La capacité montrée sur la figure 4.7 spécifie l'index de la capacité dans la PRL du processus, l'adresse de base du segment, le type de la capacité (capacité d'accès pour un segment, capacité d'entrée pour une procédure protégée, etc.) le bit–map WU qui indique que le segment a été modifié (W) ou utilisé (U), et la taille du segment autorisé au sujet. L'adresse de

base et la taille peuvent être utilisés pour raffiner l'accès, c'est–à–dire réduire la taille de la fenêtre sur le segment auquel le sujet peut accéder.

**Création d'un objet :** Lors de la création d'un segment de mémoire le système retourne une capacité qui inclut tous les droits sur le segment.

**Manipulation des capacités :** Il existe deux instructions dans CAP qui autorisent aux sujets à manipuler les capacités : `movecap` et `refine`.

- L'opération `movecap` permet de copier une capacité d'un segment à un autre, si le sujet possède les droits d'accès adéquats sur les segments.
- L'opération `refine` permet de faire une copie de la capacité (comme `movecap`), mais les droits d'accès de la copie sont inférieurs à ceux de la capacité. Par exemple le sujet peut créer une capacité de lecture à partir d'une capacité de lecture–écriture ou une capacité qui donne accès à une partie du segment de la capacité originale.

Dans tous les cas il faut l'intervention du système pour copier la capacité.

**Changement de domaine :** Le changement de domaine à travers des procédures protégées est contrôlé par les capacités d'entrée et de sortie. L'appel d'une procédure protégée correspond à une commutation des registrent de segment de capacité. Le système change 5 segments de capacités (sur 16) : le segment 4 (appelé *P* pour "Program") qui spécifie les capacités nécessaires pour l'exécution de la procédure protégée, le segment 5 (appelé *I* pour "Interface") qui spécifie les droits d'accès associés à l'appelant, le segment 6 (appelé *R* pour "Resource") qui spécifie les capacités de l'instance de la procédure protégée dans le contexte d'un objet protégé, le segment 2 (appelé *A* pour "Argument") qui spécifie les capacités à passer en paramètre à l'appel de la procédure protégée et finalement le segment 3 (appelé *N* pour "New argument") qui est utilisé pour construire la C–liste à passer à la procédure protégée et retourner les capacités au retour de l'appel (le segment *A* est remplacé par *N* au moment de l'appel).

**Délégation :** Il existe deux mécanismes pour la délégation dans CAP, soit par une C–liste partagée entre un processus et son sous–processus, soit par un appel de procédure protégée. Les segments de capacités sont uniquement partagés entre flots de contrôle dans le même processus ou entre un processus et l'arborescence d'un de ses sous–processus.
L'appel à une procédure protégée permet de passer les segments de capacité entre appelant et appelé comme décrit ci–dessus. Les segments , *P, I* et *R* spécifient la sphère de protection pour la procédure protégée. Les segments *A* et *N* sont utilisés pour déléguer les capacités. L'appelant et l'appelé s'exécutent toujours dans le contexte du même processus.

**Révocation :** La structure hiérarchique des capacités permet à un processus de révoquer une capacité unique dans un sous–processus et tous les sous–processus de celui–ci.

**Vérification :** Il n'y a pas de phase de préparation explicite, celle–ci est faite automatiquement par le système.

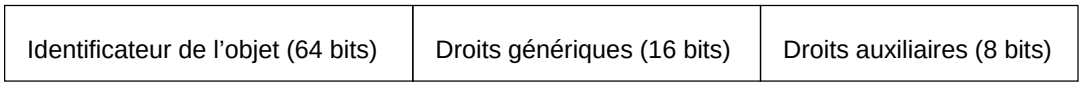



FIG. 4.8: Le format simplifié des capacités d'Hydra

**Accès aux objets :** La capacité fait partie de la désignation et elle est vérifiée à chaque utilisation.

#### Classification selon notre taxinomie

Il existe deux classifications possibles du mécanisme de changement de domaine dans Cambridge CAP. Si le système réinitialise les registres à capacités 7–16 avant l'appel, il réalise un mécanisme d'isolation, sinon il réalise une extension du domaine.

[**classification : bb(a∨c)bbab**]

### 4.4.2 Hydra

Hydra est le système d'exploitation d'un multi–processeur (le C.mmp) développé à l'Université Carnegie–Mellon aux États Unis [Wulf81]. Hydra n'est pas un système d'exploitation complet, mais fournit une base pour faire des expériences avec différents services systèmes.

#### Description du système de protection d'Hydra

Hydra réalise un système à objets actifs protégés par capacités. Les capacités sont stockées dans une C–liste propre à chaque instance de l'objet. L'instance de l'objet est appelée l'espace de désignation local ("Local Name Space" ou "LNS"). Une instance est décrite par un nom global qui identifie l'instance, un type qui détermine quelles opérations peuvent être invoquées sur l'objet et une représentation qui décrit l'état de l'objet y compris une C–liste pour accéder à d'autres objets.
La machine C.mmp n'offre aucun support matériel pour l'adressage de la mémoire par capacités, tous les objets désignés par des capacités — et les capacités elle–mêmes — doivent être confinés dans le noyau.

#### Format des capacités

Le format représenté sur la figure 4.8 montre les éléments les plus importants d'une capacité Hydra, c'est–à–dire le nom unique de l'objet, les droits d'accès génériques (ces droits sont définis pour tous les objets) et les droits d'accès auxiliaires (ces droits sont spécifiques au type de l'objet).
Les droits d'accès génériques sont réalisés par un bit–map qui spécifie les droits suivants (`1` signifie la présence du droit). Les différents droits génériques sont représentés dans la table 4.1.

| | |
|---|---|
| `GetDataRts`, `PutDataRts` et `AppendDataRts` | Spécifient le droit de lire, modifier ou rajouter des données à l'état de l'objet ; |
| `GetCapaRts`, `PutCapaRts` et `AppendCapaRts` | Spécifient le droit de copier une capacité de l'objet, modifier les capacités de l'objet ou rajouter une capacité dans la C–liste de l'objet ; |
| `DeleteRts` | Spécifie le droit de supprimer cette capacité d'une C–liste ; |
| `KillRts` | Permet de supprimer les capacités dans la C–liste de l'objet référencé par la capacité. De plus il faut que la capacité dans la C–liste inclue le `DeleteRts` ; |
| `ModifyRts` | Permet de modifier l'état d'un objet ; |
| `EnvRts` | Permet de stocker une capacité en dehors du LNS courant ; |
| `UncfRts` | Permet la modification d'un objet à travers un objet spécifié. Si la capacité ne spécifie pas l'`UncfRts`, ni la C–liste ni les données de l'objet peuvent être modifiées, ceci permet de réaliser un modèle de contrôle d'accès mandataire ; |
| `CopyRts` | Nécessaire pour appeler l'opération `$copy`. |

TAB. 4.1: Les droits génériques d'un objet Hydra

Une capacité complète d'Hydra contient des informations supplémentaires, notamment liées à la localisation de l'objet.

**Création d'un objet :** L'objet type ("type manager") est responsable de la création de tout objet de son type. La création d'un objet est normalement faite par un appel `$create` à l'objet type. Le résultat de la création dépend donc du type, mais l'objet type va normalement créer un objet, initialiser les données et la C–liste et retourner une capacité pour l'objet à celui qui a appelé `$create`. Cette capacité ne permet normalement pas de modifier l'état de l'objet directement, mais uniquement à travers des opérations définies par l'objet type.

**Manipulation des capacités :** Les capacités d'Hydra sont confinées par le noyau. Toute manipulation des capacités se fait donc à travers des appels du système.
Hydra supporte les opérations de capacité suivantes pour tout type d'objet :

`$getcapa` qui permet de copier une capacité d'une C–liste d'un autre objet dans la C–liste locale (dans le LNS).

`$putcapa` qui permet de copier une capacité locale dans une endroit précis de la C–liste d'un autre objet.

`$appendcapa` qui permet de rajouter une capacité locale à la fin de la C–liste d'un autre objet.

`$compare` qui compare deux capacités.

`$restrict` qui réduit les droits d'accès d'une capacité.

`$delete` qui permet de supprimer une capacité.

Pour pouvoir invoquer ces opérations, il faut donc que le sujet possède une capacité avec les droits adéquats.

**Changement de domaine :** Le système utilise des modèles de paramètre ("parameter templates") pour vérifier le type des capacités passées en paramètre dans l'appel d'une opération sur un objet. Un modèle de paramètre spécifie le type d'une capacité et les droits

requis par l'appel. Avant d'installer une capacité passée en paramètre dans la C–liste locale, le système vérifie qu'elle a le bon type et qu'elle spécifie au moins les droits spécifiés dans le modèle du paramètre.
Le contexte de l'objet se compose des capacités spécifiées pour le type et des capacités passées en paramètre dans un appel de méthode.

**Délégation :** Le seul moyen de déléguer une capacité à un objet est d'appeler une opération dans l'objet et de passer la capacité en paramètre. Il faut que le sujet ait le droit de copier la capacité de sa propre C–liste et de l'insérer dans la C–liste de l'objet. Les opérations `$getcapa`, `$putcapa` et `$appendcapa` sont effectivement des méthodes définies pour tous les objets.

**Révocation :** La combinaison des droits de `KillRts` et `DeleteRts` décrits ci–dessus permet de supprimer une capacité dans la C–liste d'un objet spécifique. Pour pouvoir supprimer une capacité dans sa propre C–liste il faut de plus que le domaine courant possède le droit `EnvRts`.

**Vérification :** L'invocation d'une opération sur un objet provoque la vérification de la capacité. Cela est fait automatiquement par le système.

**Accès aux objets :** Le système vérifie que toutes les manipulations d'un sujet correspondent à celles permises par la capacité utilisée pour faire l'invocation, par exemple si la capacité ne spécifie pas les droits `PutDataRts` et `ModifyRts`, les données stockées dans l'objet ne peuvent pas être modifiées.

#### Classification selon notre taxinomie

La classification selon notre taxinomie d'Hydra est donnée par la description suivante :

[**classification : bbcbbab**]

### 4.4.3 Discussion des systèmes centralisés

Le modèle de protection réalisé par CAP est très général. Il permet la réalisation des politiques mandataire et discrétionnaire. Pour réaliser un modèle mandataire, il faut que toutes les classes de confidentialité différentes soient confinées dans différents processus au niveau 2. Dans ce cas il faut que tous les passages de capacités passent par le MC au niveau 1. Le MC peut donc vérifier que les passages se font selon les règles imposées par la politique mandataire. Pour réaliser un modèle discrétionnaire, il faut que le MC n'impose aucune restriction sur la délégation des capacités.
Le chargement transparent du segment de capacités passé en paramètre est une caractéristique très intéressant du système CAP.
Le modèle de paramètres est aussi une caractéristique très intéressant du système Hydra.

## 4.5 Systèmes répartis

Les systèmes à capacités répartis n'ont pas de plate–forme en commun. La capacité doit donc être vérifiée à chaque utilisation, soit par la vérification de la capacité elle–même (capacités chiffrées), soit par l'authentification du site qui présente la capacité par un canal de communication sûre (capacités confinées).
Nous présentons deux approches différentes de réalisation d'un système à capacités réparti dans la suite.

### 4.5.1 Amoeba

Amoeba est un système réparti à objets développé au Vrije Universiteit à Amsterdam [Tanenbaum86, Tanenbaum90]. Amoeba est conçu comme un micro–noyau où différentes ressources peuvent être gérées par des serveurs dans l'espace utilisateur. La notion de système doit donc inclure les serveurs qui gèrent les ressources, par exemple le système de fichiers. Les applications sont également développées selon le modèle client/serveur.

#### Description du système de protection d'Amoeba

Les objets gérés par un serveur sont désignés par des capacités. Le serveur contrôle tous les accès à ses objets, qui ne sont normalement pas visibles de l'extérieur. Le serveur encapsule donc les objets comme les objets encapsulent leur état.
Une capacité autorise le client à exécuter certaines opérations sur un objet. L'opération prend le forme d'un appel à distance traditionnel [Birrell84]. L'appel doit passer par un point d'entrée — appelé un port — défini par le serveur.
Remarquons qu'il n'existe pas une distinction précise entre clients et serveurs. Si un client passe une capacité pour un objet en paramètre à un serveur, ce serveur va devenir le client du serveur qui gère l'objet référencé par la capacité.

#### Format des capacités

Une capacité d'Amoeba se compose de quatre champs (cf. la figure 4.9).

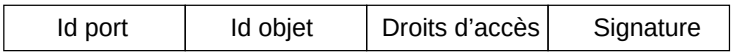



FIG. 4.9: Le format d'une capacité d'Amoeba

Les quatre champs contiennent l'information suivante :

1. le port qui sert de point d'entrée dans le serveur ;
2. un identificateur de l'objet local au serveur ;
3. un bit–map qui contient 1 bit par droit d'accès possible ;
4. une signature qui permet de vérifier les droits d'accès.

La sémantique pour les bit–map des droits d'accès et l'algorithme utilisé pour faire la signature dépendent du serveur. La clé de chiffrement est également gérée par le serveur et elle est normalement stockée avec l'objet.

**Création d'un objet :** Un client peut demander à un serveur de lui créer un nouvel objet. Si le client a le droit de créer des objets, le serveur crée l'objet et retourne une capacité au client. Le format de cette capacité et la sémantique pour les champs de droits d'accès et de la signature dépendent du serveur, mais ils incluent normalement le droit d'accéder à l'objet.

**Manipulation des capacités :** Les mécanismes de manipulation des capacités dépendent du serveur. Normalement les clients peuvent copier une capacité sans l'intervention du serveur et il existe également un schéma de restriction des droits d'accès qui permet au client de dériver une capacité moins privilégiée avant de la passer à quelqu'un d'autre.
Il existe deux moyens de restreindre une capacité dans Amoeba : soit le client demande au serveur de la restreindre, soit la gestion de capacité suit le format suivant.
À la création de l'objet, le serveur stocke avec l'objet le mot de passe utilisé pour créer des capacités.
Le serveur doit définir une fonction de hachage à sens unique commutative[5] par valeur dans le bit–map des droits d'accès. Pour retirer un droit spécifique dans la capacité il suffit de mettre sa valeur dans le bit–map à zéro et d'appliquer la fonction de hachage à la signature. Pour vérifier une capacité, le serveur applique toutes les fonctions de hachage qui correspondent à un droit retiré — dans le champ des droits d'accès inclus dans la capacité — au mot de passe stocké avec l'objet. Ensuite il compare le résultat avec le mot de passe inclus dans la capacité, si les deux correspondent la capacité est valable. La fonction de hachage doit être commutative pour que l'ordre dans lequel on retire les droits n'ait pas d'importance.

**Changement de domaine :** Le serveur spécifie un ou plusieurs points d'entrée dans le serveur (le domaine) qu'il exporte aux clients. Un point d'entrée est spécifié par le premier champ de la capacité. Le port est un nombre grand (normalement 48 bits), la connaissance de ce nombre peut donc être prise comme indication que le client à le droit de contacter le serveur (le serveur est libre de ne pas répondre ou de retourner un message d'erreur au client).
Le serveur va d'abord vérifier si les droits d'accès inclus dans la capacité correspondent avec ceux encodés dans la signature. Si les droits sont intègres et s'ils autorisent l'opération, le serveur invoque l'opération sur l'objet et retourne la réponse au client.

**Délégation :** Les capacités sont des structures de données gérées par les clients. Un client peut donc copier une capacité telle quelle et l'envoyer à quelqu'un d'autre. Les schémas de restriction décrits ci-dessus permettent également au client de restreindre les droits d'accès inclus dans la capacité avant de la copier.

**Révocation :** Les algorithmes de signature couramment utilisés dans Amoeba dépendent d'un mot de passe stocké avec l'objet. Il suffit donc de changer ce mot de passe pour invalider toutes les capacités.

**Vérification :** Le serveur va d'abord chercher l'objet désigné par le deuxième champ de la capacité pour trouver la clé de chiffrement (ou mot de passe) associée à l'objet. Le serveur

[5] Une fonction de hachage à sens unique ("one-way hash function") est définie par deux propriétés : l'image de la fonction est de taille fixe (normalement d'une taille inférieure à celle de départ, mais dans le cas d'Amoeba d'une taille égale) et elle doit être facile à calculer mais l'inverse doit être d'une grande complexité.

utilise cette clé pour vérifier la signature et les droits d'accès inclus dans la capacité. Si les deux correspondent, le serveur invoque l'opération demandée par le client. Le schéma de vérification dépend du format de la capacité et du schéma de manipulation des capacités utilisé par le serveur.

**Accès aux objets :** les clients n'ont pas d'accès direct aux objets, tous les accès passent donc à travers un point d'entrée dans le serveur. La vérification décrite ci–dessus est faite par le serveur à chaque accès à l'objet.

#### Classification selon la taxonomie

La classification de la révocation des capacités dans Amoeba dépend de l'utilisation du système. Si le système gère un mot de passe par utilisateur, le système peut réaliser une révocation sélective chez l'objet, sinon il est obligé de faire une révocation en bloc de toutes les capacités qui existent pour l'objet.

[**classification : bacb(a∨c)ab**]

### 4.5.2 Mach

Mach est un micro–noyau de système développé à l'université de Carnegie Mellon [Accetta86, Boykin93, Loepere92a, Loepere92b]. Il fournit les mécanismes de base permettant de construire des systèmes répartis. Il a servi notamment de base à la réalisation des systèmes OSF/1 (système Unix) et Guide–2 (système réparti à objets).
Le système Mach repose sur les notions de tâche (un espace d'adressage multi-thread) et de port (une adresse permettant l'émission et la réception de messages). Un port appartient à une tâche, cette tâche pouvant recevoir des messages sur ce port. Toute tâche ayant connaissance de ce port peut potentiellement lui envoyer un message. Les ports peuvent être échangés entre les tâches par envoi de message.

#### Description du système de protection de Mach

Dans Mach, toutes les ressources sont identifiées et protégées par des ports. Un port peut servir à contrôler le droit de coupler un segment de mémoire partagée, ou correspondre à un point d'entrée dans un serveur d'objets.
Un port peut donc être vu comme une capacité. Mach permet d'échanger des ports entre les tâches par des envois de message, comme des capacités. Les droits associés à un port sont principalement : le droit d'envoyer un unique message ou le droit d'envoyer un nombre quelconque de messages. Il est possible de restreindre les droits associés à un port avant de l'envoyer dans un message. Dans Mach, les ports sont gérés par le noyau du système. Les ports sont désignés dans une tâche par un identifiant local à la tâche. Cet identifiant est interprété par le noyau à chaque fois que la tâche présente le port au noyau, soit pour émettre ou recevoir un message, soit pour passer un port dans un message (le port doit être explicitement déclaré dans le message). Lorsqu'un port est transmis d'une tâche T1 à une tâche T2, l'identifiant local est reçu par Mach qui vérifie que le port existe effectivement dans T1, puis Mach ajoute le port dans la tâche T2 et lui associe un identifiant local à T2 qui lui est retourné. Notons que les ports sont gérés par le noyau et ne peuvent être forgés.

**Création d'un objet :** La création d'un objet dans Mach correspond en fait à la création d'un port qui représente l'objet dans un serveur. Le serveur peut alors recevoir des requêtes (messages) sur ce port et les transmettre à l'objet qu'il héberge. La connaissance de ce port peut être transmise aux clients du serveur soit en retour d'une requête, soit par l'intermédiaire d'un serveur de noms.

**Manipulation des capacités :** Les capacités sont manipulées explicitement par les programmes. Un programme qui désire envoyer un message sur un port doit construire une structure de données correspondant au message. Cette structure de données doit inclure des informations de typage des données, et notamment indiquer quelles données du message correspondent à des ports, permettant au noyau de réaliser le transfert effectif des ports passés en paramètre.

**Changement de domaine :** L'envoi de message sur un port correspond à un changement de domaine, car il permet de réaliser l'exécution d'une opération dans une autre tâche (un domaine de protection).
Le noyau vérifie lors de l'envoi de message si l'utilisation du port d'envoi est légale et si les transferts de ports en paramètre (décrits dans la structure du message) le sont également.

**Délégation :** Les droits sur les ports de Mach peuvent être échangés entre les tâches. Le droit d'envoyer un message sur un port peut être copié ou transféré, mais le droit de recevoir un message peut uniquement être transféré.

**Révocation :** Il existe une opération `mach_port_destroy` qui permet au système de retirer le droit d'envoyer ou recevoir des messages sur un port. L'appel supprime effectivement la référence au port dans la tâche.

**Vérification :** Toutes les vérifications des ports dans Mach ont lieu dans le noyau. Ceci implique qu'il n'est pas possible de forger une capacité.

**Accès aux objets :** Le droit d'envoyer ou recevoir un message sur un port est vérifié chaque fois que le port est utilisé.

#### Classification selon la taxinomie

Il existe deux façons de déléguer le droit sur un port, par copie (un port d'envoi de message) et par transfert du port (un port de réception de message).
La classification de Mach selon notre taxinomie est donnée par la description suivante :

[**classification : bbc(a∨b)bab**]

### 4.5.3 Discussion

Amoeba réalise un contrôle d'accès par capacité chiffrée qui permet aux clients de manipuler les capacités sans intervention du système (le serveur). Par contre, Mach réalise un contrôle d'accès par capacité confinée qui exigent que tous les transferts de capacités passent par le

système. La portée d'une capacité dans Mach est local. Un utilisateur est ainsi incapable de déléguer une capacité sans l'intervention du système.

**Remarque !** Le système Amoeba n'a pas besoin de connaître le format des capacités en détail, il suffit de connaître le numéro de port du serveur. La sémantique de l'identificateur d'objet, les droits d'accès et la signature dépendent du serveur. Ceci a pour conséquence que plusieurs mécanismes de contrôle d'accès différents peuvent coexister dans le système. Il est donc facile d'adapter le système de contrôle d'accès à un modèle de protection donné, même si un modèle discrétionnaire semble être beaucoup plus facile à réaliser qu'un modèle mandataire.

## 4.6 Les mémoires virtuelles réparties uniques

Les mémoires virtuelles réparties uniques sont des systèmes répartis qui gèrent la mémoire d'une façon globale, c'est–à–dire qu'une adresse virtuelle sert d'identificateur global pour les objets. La visibilité globale de toute la mémoire virtuelle nécessite une séparation entre la désignation et le contrôle d'accès.
Le contrôle d'accès à la mémoire est fait par capacités dans tous les prototypes de MVRU connus.

### 4.6.1 Opal

Opal est un système à mémoire virtuelle répartie unique développé à l'Université de Washington [Chase92, Chase93, Chase94, Chase95]. Le but principal d'Opal a été d'explorer la conception d'une mémoire virtuelle répartie unique sur des architectures à adressage étendu (64 bits).

#### Description du système de protection d'Opal

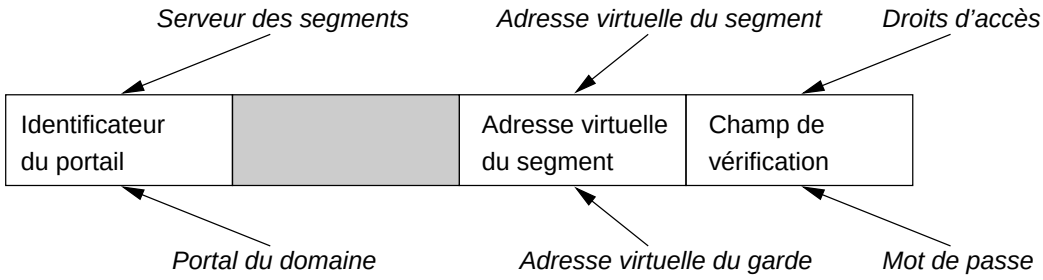
Le système de protection d'Opal réalise une protection par capacités selon le modèle de Dennis et Van Horn. Il implante deux types de capacités différents : les capacités d'accès aux segments et les capacités pour des objets protégés.
Une capacité d'accès à un segment est utilisée lors du couplage d'un segment dans l'espace d'adressage du processus[6]. La capacité spécifie les droits d'accès suivants : lecture, écriture ou exécution.
Une capacité pour un objet protégé permet d'appeler une méthode dans un objet protégé. La capacité identifie un point d'entrée ou portail ("portal") qui permet d'entrer dans le domaine. Le portail n'est pas protégé par le système, tous les sujets qui connaissent le numéro du portail peuvent donc envoyer un message au portail. La capacité inclut également l'adresse virtuelle d'une procédure garde ("guard") — qui contrôle effectivement l'accès à l'objet — et un champ de vérification utilisé pour empêcher la fabrication des capacités.
Le système d'exécution cache les portails et le passage des capacités dans des "proxies".

[6] Le couplage et découplage d'un segment est fait explicitement par les appels systèmes `attach` et `detach`.

**Format des capacités**

Les capacités d'Opal sont des structures de données gérées par les utilisateurs eux mêmes. La capacité se compose de quatre mots de 64 bit. Le premier mot identifie le service (soit le serveur des segments, soit un objet protégé), le deuxième mot n'est pas utilisé, le troisième mot spécifie une adresse virtuelle et le quatrième mot spécifie un champ de vérification ("check field"). Le format des capacités est montré sur la figure 4.10.

a) Capacité d'accès au segment




b) Capacité pour un objet protégé

FIG. 4.10: Le format des capacités d'Opal

La figure montre deux interprétations d'une capacité : une capacité d'accès au segment (cf. figure 4.10 a) et une capacité pour un objet protégé (cf. figure 4.10 b).
Le rôle précis du champ de vérification dépend des services. Les deux services par défaut d'Opal — serveur de segment et le serveur de domaine — le traitent comme un simple mot de passe pour accéder au service [Chase99].

**Création d'un objet :** Chaque appel de création d'une ressource protégée retourne une capacité pour la ressource à l'appelant.

**Manipulation des capacités :** Les capacités sont les structures de données gérées dans la mémoire par l'utilisateur. L'utilisateur est donc libre de la déléguer à n'importe qui.

> *Remarquons que la notion de confiance est transitive et que les capacités peuvent être volées ou données par maladresse à quelqu'un qui n'est pas censé la posséder (le problème de confinement)* [Chase95]

**Changement de domaine** Le mécanisme de changement de domaine est montré sur la figure 4.11.
Le proxy envoie la capacité au portail qui la renvoie au garde spécifié par l'adresse virtuelle inclus dans la capacité. Ce garde vérifie que le champ de vérification inclus dans la capacité correspond à la copie locale créée lors de la création de la capacité. Si les valeurs correspondent le garde sert de talon serveur comme dans un appel RPC normal.

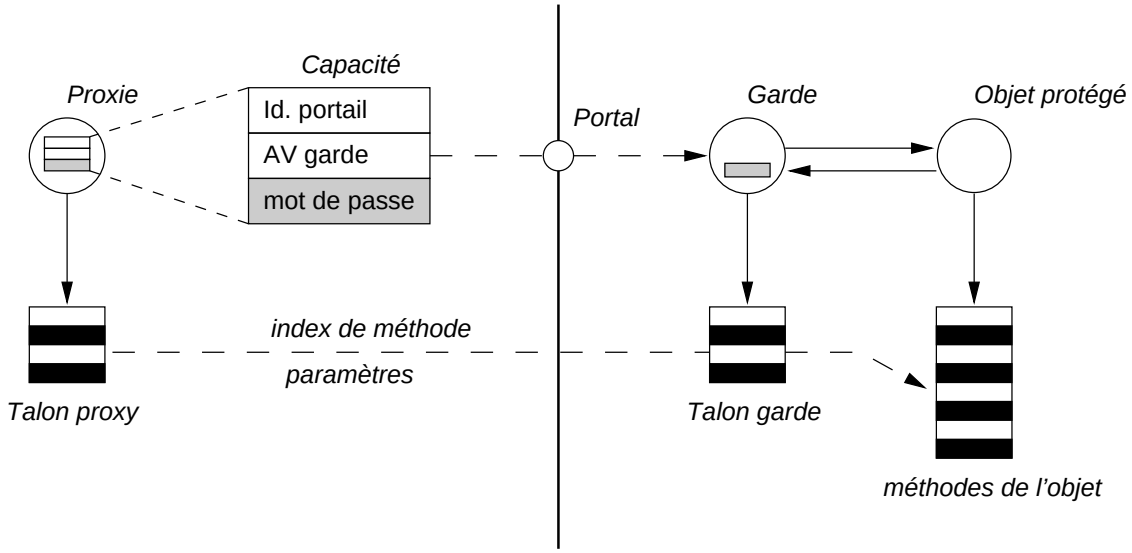



FIG. 4.11: Le changement de domaine dans Opal

**Délégation :** Il existe deux moyens de déléguer les capacités dans Opal : par la mémoire partagée ou par le passage de la capacité en paramètre à un appel de changement de domaine. Le système n'impose aucune restriction à la délégation des capacités.
Dans le cas du partage de la mémoire, deux domaines partagent effectivement la même capacité. Dans le cas du passage en paramètre, une copie de la capacité est passée entre le domaine de l'appelant et le domaine de l'appelé.

**Révocation :** La capacité est protégée par un mot de passe. Une copie de ce mot de passe est stockée chez le serveur. Le serveur peut donc facilement révoquer toutes les capacités en changeant ce mot de passe. Il n'existe aucun moyen de révoquer une capacité explicitement.

**Vérification :** La vérification de la capacité d'accès au segment provoquée par un appel à `attach` permet au serveur des segments de coupler le segment dans l'espace d'adressage de l'appelant avec les droits d'accès spécifiés par la capacité. Une fois le segment couplé en mémoire, l'appelant peut y accéder directement.
Il n'y a pas de phase de vérification pour les appels d'objets protégés, la capacité est vérifiée à chaque appel.

**Accès aux objets :** Lors de l'appel à `attach`, le serveur des segments met à jour des structures nécessaires dans la gestionnaire de la mémoire pour que celles–ci puissent contrôler l'accès au segment.

**Classification selon la taxinomie**

La classification d'Opal selon notre taxinomie est donnée par la description suivante :

[**classification : bacbaab**]

### 4.6.2 Mungi

Mungi est une MVRU développée a l'Université de
New South Wales [Vochteloo93, Vochteloo96, Vochteloo98]. Un objectif principal de Mungi a été de rendre les mécanismes de contrôle d'accès transparents pour le programmeur.

#### Description du système de protection de Mungi

Le système de protection de Mungi réalise une protection par capacités chiffrées[7]. Le système réalise les notions de capacité, objet (un objet se compose d'une suite de pages contiguës, également appelé un segment), domaine de protection et l'extension de domaine.
Mungi offre un modèle de contrôle d'accès hiérarchique. Les C–listes sont organisées selon une arborescence globale avec les C–listes publiques près de la racine et les C–listes privées près des feuilles. Un nœud dans cette arborescence — qui s'appelle un Pnode ("Protection node") — inclut un pointeur vers une C–liste et un pointeur vers son prédécesseur dans l'arborescence[8]. L'arbre est montré sur la figure 4.12.

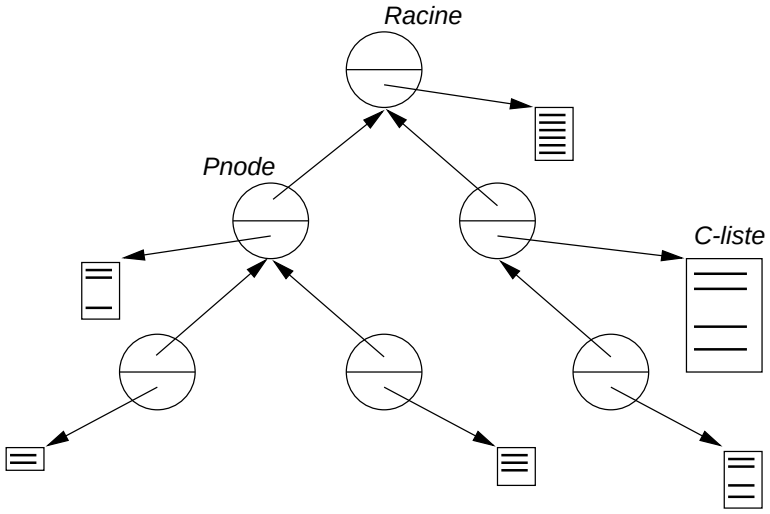



FIG. 4.12: L'arbre des capacités de Mungi

Le domaine initial d'un utilisateur est déterminé par un pointeur vers un Pnode dans cet arbre et le domaine se compose de la fermeture transitive de nœuds accessibles depuis le Pnode. Les utilisateurs sont autorisés à rajouter et supprimer les Pnodes s'ils possèdent les droits adéquats.

#### Format des capacités

Nous ne connaissons pas le format exact d'une capacité dans Mungi, mais elle inclut au moins l'adresse de l'objet (l'adresse de base du segment), les droits d'accès et le mot de passe qui correspond à ces droits d'accès. Les droits d'accès sont les suivants :

[7] La signature est en fait un mot de passe stocké avec l'objet et inclus dans la capacité.

[8] le Pnode peut également inclure un pointeur vers un "protection fault handler" fourni par l'utilisateur pour spécialiser la recherche des capacités.

`d` détruire l'objet
`r` lire l'objet
`w` écrire l'objet
`x` utiliser l'objet comme du code exécutable

**Remarque !** Le droit d'écrire existe uniquement en combinaison avec le droit de lire l'objet.

**Création d'un objet :** Après la création d'un objet, le système retourne une capacité qui donne tous les droits `drwx` au créateur. Ce type de capacité s'appelle une capacité de propriétaire ("owner capability"). Le système crée un mot de passe pour cette capacité et pour toutes les capacités dérivées, puis il les insère (avec les droits d'accès correspondants) dans un tableau géré avec l'objet. Ceci permet au système de vérifier les capacités et aux sujets de construire des versions limitées de leurs capacités existantes.

**Manipulation des capacités :** Une capacité de propriétaire permet de créer des nouvelles capacités de propriétaire. Il suffit de faire un appel système avec un nouveau mot de passe. Le système enregistre ce mot de passe et tous les mots de passe dérivés avec les mots de passe existants (dans le tableau géré avec l'objet).
Tous les autres types de capacités permettent uniquement de créer une capacité dérivée.

**Changement de domaine** Les droits d'accès d'un domaine de protection Mungi peuvent être augmentés ou réduits de façon temporaire par un appel d'extension du domaine ("protection domain extension" ou "PDX"). L'extension du domaine ne change pas le domaine de base mais permet d'exécuter une procédure particulière avec des droits supplémentaires. Il n'y a donc pas besoin de passer des capacités entre le domaine appelant et le domaine appelé. Ce mécanisme permet également l'isolation, si l'appelant fait une restriction de son domaine avant l'appel de la procédure PDX.

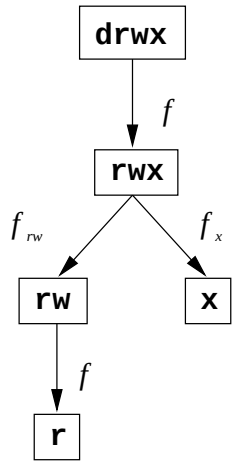



FIG. 4.13: La hiérarchie des capacités dérivées

**Délégation :** Les capacités sont des structures de données gérées par les utilisateurs qui sont libres de les copier et de les passer dans les appels d'extensions de domaine ou de les partager

dans un segment de la mémoire partagée. Une transformation connue du mot de passe de la capacité permet de dériver une version restreinte de la capacité.
L'application d'une fonction à sens unique ("one–way function") au mot de passe inclus dans la capacité de propriétaire permet de construire des capacités réduites. Trois fonctions publiques sont définies par le système : $f$, $f_{\text{rw}}$ et $f_{\text{x}}$.
Appelons la capacité de propriétaire $C_{\text{drwx}}$ ; les capacités suivantes peuvent être dérivées de cette capacité : $C_{\text{rwx}} = f(C_{\text{drwx}})$, $C_{\text{rw}} = f_{\text{rw}}(C_{\text{rwx}})$, $C_{\text{x}} = f_{\text{x}}(C_{\text{rwx}})$ et $C_{\text{r}} = f(C_{\text{rw}})$. Cette dérivation est illustrée sur la figure 4.13. La figure montre les différentes possibilités pour dériver une capacité restreinte à partir d'une capacité plus privilégiée.

**Révocation :** Une capacité de propriétaire permet de demander au système de supprimer certains mots de passe dérivés de ceux inclus dans la capacité. Ceci révoque effectivement toutes les capacités qui dépendent de ces mots de passe.

**Vérification :** La première référence à un objet protégé provoque une faute de protection. La résolution de cette faute vérifie s'il existe une capacité pour l'objet dans le domaine et compare le mot de passe inclus dans la capacité avec celui stocké avec l'objet qui donne les mêmes droits d'accès. Si les mots de passe correspondent, l'accès est accordé et le système met à jour les structures nécessaires pour coupler l'objet dans l'espace d'adressage du domaine.

**Accès aux objets :** La vérification du droit d'accès se fait au moment d'une faute de page. Le système vérifie le droit d'accès à l'objet dans une structure ("Protection Lookaside Buffer" ou "PLB") gérée de la même façon que la structure des pages en mémoire actives ("Translation Lookaside Buffer" ou "TLB"). Ces deux structures sont explorées en parallèle.

#### Classification selon la taxinomie

La classification de Mungi selon notre taxinomie est donnée par la description suivante :

[**classification : baabaab**]

### 4.6.3 Angel

Angel est une MVRU développée à City University,
Londres [Murray93a, Murray93b, Wilkinson95].

#### Description du système de protection d'Angel

Le système de protection d'Angel se base sur les capacités, les domaines de protection actifs et la communication entre domaines par interruptions logicielles.
La notion de capacité se base sur une combinaison des capacités confinées et des capacités chiffrées. Une capacité consiste de deux parties : une *description du contrôle d'accès* ("access control descriptor" ou "ACD") confiné et un biscuit (une capacité) publique qui donne le droit d'utiliser l'ACD.
Les droits d'accès fondamentaux sont les suivants :

| | | |
|---|---|---|
| *lire,* | : | le droit de lire un objet ; |
| *écrire,* | : | le droit d'écrire un objet ; |
| *exécuter,* | : | le code exécuté par le sujet ; |
| *lire droits,* | : | le droit de lire les droits associés au biscuit ; |
| *écrire droits,* | : | le droit de modifier les droits associés au biscuit ; |
| *supprimer biscuit,* | : | le droit de supprimer tout accès à l'objet utilisant ce biscuit ; |
| *créer biscuit* | : | le droit de créer un nouveau biscuit avec un nouvel ensemble des droits ; |
| *rend exécutable* | : | le droit de rendre l'objet exécutable ; |
| *"upcallable"* | : | le droit de faire une interruption logicielle vers l'objet. |

**Format des capacités**

En réalité l'ACD n'est pas une capacité confinée, parce qu'il peut être soumis à des conditions supplémentaires ; par exemple il donne le droit de lire un objet uniquement si le sujet possède des droits pour un autre objet.
Le biscuit inclut de l'information pour que le système puisse le vérifier ; ceci est probablement un mot de passe comme Opal et Mungi.
Outre cette séparation de la capacité en deux parties, nous n'avons pas trouvé de description plus précise du format des capacités dans Angel.

**Création d'un objet :** Les gestionnaires d'objet sont responsables de la création des objets. Quand le gestionnaire crée un objet, il crée également l'ACD et un biscuit qu'il retourne au utilisateur qui demandait la création. Le contenu précis de l'ACD dépend du gestionnaire d'objet. Un biscuit qui identifie l'objet à travers l'ACD est retourné au créateur.

**Manipulation des capacités :** Les ACD sont confinés par le gestionnaire d'objet hors de portée des utilisateurs. Les biscuits sont des structures de données stockées par les utilisateurs. Les droits de créer un biscuit avec des droits différents et de supprimer toute instance d'un biscuit sont contrôlés par le système.

**Changement de domaine** Il n'y a pas d'appel de changement de domaine dans Angel. Le changement de domaine est effectué par une interruption logicielle dans le domaine appelé qui passe une adresse en paramètre à l'interruption. Il faut donc que chaque domaine possède le droit de faire une interruption logicielle chez l'autre (l'appelant pour faire l'appel et l'appelé pour retourner l'appel).
L'adresse passée en paramètre doit pointer vers un objet partagé qui contient les paramètres de l'appel.

**Délégation :** Les utilisateurs peuvent passer les biscuits librement entre–eux. Ceci n'implique pas forcément le passage du droit d'accès, parce qu'il faut que le récepteur satisfasse toutes les conditions incluses dans l'ACD.

**Révocation :** Le droit de supprimer un biscuit permet de supprimer le droit d'accès pour toutes les instances du biscuit. Cela n'est pas une révocation totale de tous les droits d'accès pour la ressource, parce que d'autres biscuits peuvent exister pour le même ACD. Si un sujet qui possède un tel biscuit satisfait les conditions dans l'ACD il peut donc toujours accéder à la ressource.

**Vérification :** La vérification du droit d'accès à un objet se fait au moment de la première référence à l'objet. Le système vérifie d'abord que le biscuit est valable, puis il vérifie si le domaine appelant satisfait les conditions dans l'ACD. Si le système associe un objet à chaque utilisateur et exige uniquement les droits sur ces objets, l'ACD réalise une simple liste de contrôle d'accès. L'ACD est donc plus générale que l'ACL. Le schéma de contrôle d'accès est présenté sur la figure 4.14.

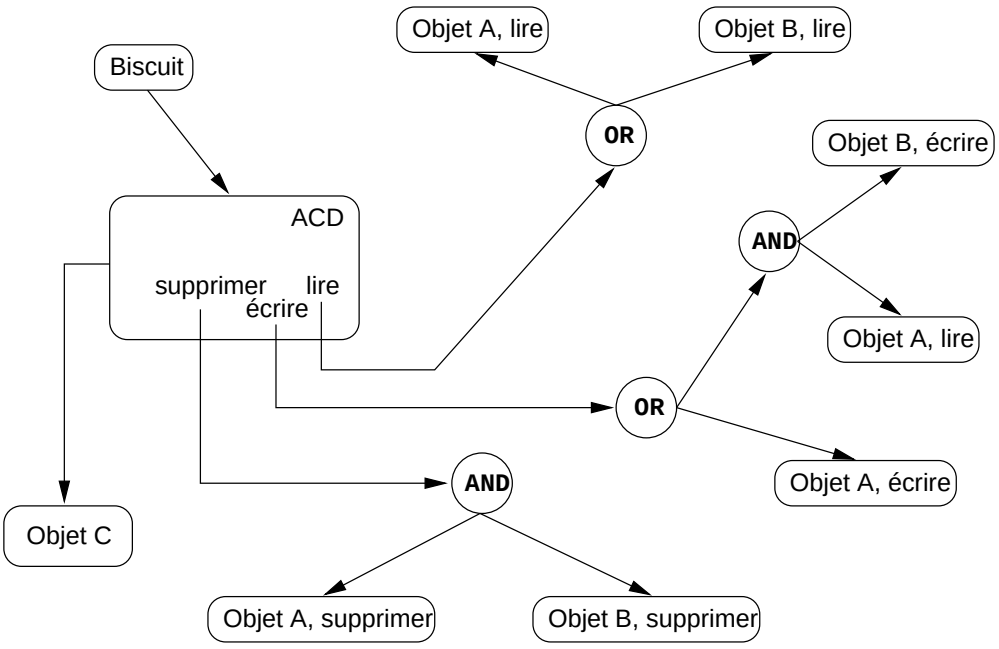



FIG. 4.14: Le schéma de contrôle d'accès d'Angel

La figure montre un biscuit qui permet d'accéder à Objet C selon les règles suivantes :

1. Le sujet est autorisé à lire Objet C si son domaine possède déjà le droit de lire Objet A ou Objet B ;
2. Le sujet est autorisé à écrire Objet C si son domaine possède déjà le droit d'écrire Objet A ou les droits de lire Objet A et d'écrire Objet B ;
3. Le sujet est autorisé à supprimer Objet C si son domaine possède déjà le droit de supprimer Objet A et Objet B.

**Accès aux objets :** La vérification du droit d'accès se fait au moment d'une faute de page comme dans Mungi. Angel utilise une structure "ddl" pareille au "PLB" de Mungi pour cette vérification.

**Classification selon la taxinomie**

Le système de contrôle d'accès d'Angel se base sur une combinaison des capacités et une ACL conditionnelle qui ne rentre pas très bien dans la taxinomie. Il y a deux possibilités pour la classification de la manipulation d'une capacité : la délégation d'un biscuit peut être faite sans intervention du système, par contre il faut l'intervention du système pour créer des nouveaux biscuits et changer les droits d'accès inclus dans un biscuit.
Il existe également deux possibilités pour la classification de la révocation d'une capacité : s'il existe un biscuit (avec son mot de passe) par utilisateur, le système peut réaliser une révocation sélective chez l'objet, sinon il est obligé à faire une révocation en bloc de tous les biscuits.

[**classification : b(a∨b)cb(a∨c)cb**]

### 4.6.4 Discussion

Opal fournit un modèle de protection par capacités chiffrées classique (la classification d'Opal (bacbaab) est presque la même que celle d'Amoeba (bacb(a∨c)ab)). Le modèle est purement discrétionnaire ; il est donc impossible de réaliser une politique de protection mandataire comme par exemple la politique de Dennis et Van Horne ou une politique basée sur les rôles.
L'extension de domaine de protection (PDX) de Mungi permet d'augmenter les privilèges lors d'un appel de procédure. Ainsi, Mungi permet d'adapter le système aux besoins des applications en utilisant des serveurs privilégiés. Par contre, le modèle n'offre aucun support direct pour des applications mutuellement méfiantes.
Le fait que le système de protection d'Angel soit très différent des autres augmente la probabilité d'une faute par mauvaise compréhension de la part du programmeur.
Il semble bizarre que le projet d'Angel propose un modèle de protection si différent des autres systèmes sans le décrire plus en détail. Il aurait été très intéressant d'apprendre leur expérience avec la programmation selon ce modèle.
Un mécanisme tel que celui d'Angel permet de réaliser les politiques mandataires, il suffit d'exiger l'existence d'un droit supplémentaire qui correspond à l'habilitation ou le rôle. Nous ne savons pas si les opérations possibles dans les ACD permettent d'interdire l'accès. Ce type de support permettrait de réaliser le modèle de la muraille de Chine.

## 4.7 Récapitulation et discussion

Les capacités offrent une grande flexibilité pour la configuration et la délégation des droits d'accès. Malheureusement, cette flexibilité se traduit souvent par une difficulté de confiner les droits d'accès, c'est–à–dire qu'un système à capacité classique ne permet pas la réalisation d'une politique de protection mandataire.
Il existe deux extensions différentes au modèle à capacités classique qui permettent de réaliser une telle politique. Le système peut vérifier chaque utilisation d'une capacité pour déterminer si l'utilisation respecte la politique mandataire (l'approche des capacités augmentées) ou le système peut contrôler la délégation des capacités (l'approche d'ICAP).
La taxinomie des systèmes à capacités proposée par Kain et Landwehr ne suffit pas pour la description des systèmes à capacités répartis. Nous avons donc proposé une nouvelle taxinomie qui permet de mieux classifier les propriétés d'un tel système. Cette taxinomie a été

utilisée pour décrire deux systèmes centralisés (Cambridge CAP et Hydra), deux systèmes répartis “classiques” (Amoeba et Mach) et trois systèmes à mémoire virtuelle répartie unique (Opal, Mungi et Angel). La classification de ces systèmes est illustrée sur le tableau 4.2.

| | CAP | Hydra | Amoeba | Mach | Opal | Mungi | Angel |
|---|---|---|---|---|---|---|---|
| Création d’un objet | b | b | b | b | b | b | b |
| Manipulation des capacités | b | b | a | b | a | a | a∨b |
| Changement des droits | a∨c | c | c | c | c | a | c |
| Délégation | b | b | b | a∨b | b | b | b |
| Révocation | b | b | a∨c | b | a | a | a∨c |
| Vérification | a | a | a | a | a | a | c |
| Accès aux objets | b | c | b | b | b | b | b |

TAB. 4.2: Classification des systèmes abordés dans ce chapitre

On voit que tous les systèmes répartis fournissent des variations d’un même modèle de base qui se classifie de la façon suivante : bacbaab.
On voit également que Mach et Angel sont les seuls systèmes à capacités répartis qui n’entrent pas dans la classification “ ?a ? ? ? ?” des systèmes qui ne permettent pas la réalisation d’une politique de protection mandataire (cf. page 69).

# Chapitre 5

# Présentation générale d'Arias

Dans ce chapitre, nous allons présenter la Mémoire Virtuelle Répartie Unique Arias qui a été développée dans le cadre du projet Sirac. Nous présentons d'abord les motivations qui nous ont amené à réaliser une MVRU, puis les objectifs de celle–ci. Nous donnons de façon succincte une vue d'ensemble de l'architecture Arias et de sa réalisation. Cette description d'Arias sert à identifier les conditions requises pour la protection dans ce système.

## 5.1 Motivations

Arias est un service de gestion de données persistantes et réparties, réalisé comme une mémoire virtuelle répartie unique. L'idée à la base d'Arias est de fournir la technologie de base pour le développement des applications réparties coopératives dans le cadre du projet Sirac. Ce choix de technologie est motivé par les arguments suivants [Balter95] :

- Le modèle de programmation fourni par la mémoire virtuelle répartie est plus simple à utiliser que le modèle client–serveur dès lors qu'il s'agit de mettre en œuvre des applications gérant des données partagées. Ce fait est maintenant largement reconnu tant par la communauté des systèmes pour le parallélisme que par celle des systèmes répartis.
- Les outils pour la réalisation de la cohérence, de la persistance et de la tolérance aux fautes d'une mémoire virtuelle répartie peuvent être rendus génériques et réutilisables pour différentes applications et environnements.
- La mémoire virtuelle répartie peut servir de champ d'expérimentation pour les travaux de construction d'applications coopératives. L'adaptation de ce service système à des techniques de coordination, de mise au point et d'observation des applications est un domaine encore peu exploré dont on peut attendre d'importantes retombées.
- Les progrès récents aussi bien dans les méthodes de mise en œuvre des mémoires virtuelles réparties que dans les techniques de réseaux laissent espérer l'obtention de bonnes performances.

Ces motivations sont principalement orientées vers l'utilisation d'une mémoire virtuelle répartie comme base pour le développement des applications réparties. Par la suite nous donnons quelques motivations pour la gestion de cette mémoire d'une façon unique et persistante.

La MVR permet aux applications qui coordonnent l'utilisation de leur espace d'adressage virtuel de partager des données en mémoire principale et d'échanger des données par simple référence (en utilisant des pointeurs vers des données stockées dans la MVR). Des systèmes de MVR traditionnels comme Ivy [Li89], Munin [Carter91], ThreadMarks [Keleher94] ou CVM [Keleher96] laissent la coordination à la charge des applications.
Pour pouvoir partager les données entre applications, il faut une désignation globale, par exemple une coordination globale des adresses virtuelles dans une MVR. La convention de gérer l'espace d'adressage des processus d'une façon unique permet l'utilisation des adresses virtuelles comme désignation globale. Les applications n'ont plus besoin de coordonner les adresses des données partagées (elles sont coordonnées par définition), par contre il faut une coordination de l'allocation et de la désallocation de la mémoire partagée.
Outre les services d'une MVR, un système MVRU fournit principalement ces services d'allocation et désallocation de la mémoire virtuelle. La réalisation de ces services dans Arias a été le sujet d'une thèse antérieure [Han96].
La gestion globale de la mémoire virtuelle permet aux applications (qui se servent de la MVRU) d'échanger des données par référence. L'espace unique permet également l'introduction de la persistance : en effet les adresses virtuelles identifient de manière unique une région de mémoire contiguë qui peut être stockée sur disque.
La persistance de la MVRU apporte les avantages suivants :

- *Un seul espace de nommage.* Il n'y a pas besoin de distinguer entre les références volatiles (les variables du programme) et les références permanentes (les fichiers ou bases de données).
- *Un seul niveau de stockage.* Il n'y a pas besoin de convertir les structures des données entre leur format à l'exécution (structures stockées dans les variables du programme) et leur représentation permanente (le flot d'octets d'un fichier séquentiel ou les tables d'une base de données).

## 5.2 Objectifs

Arias a pour but de fournir un support système générique pour la programmation des applications coopératives ou des systèmes d'exécution ("runtime systems") pour les applications coopératives. Différentes applications et différents systèmes d'exécution doivent pouvoir spécialiser Arias pour leurs besoins précis. Pour un système donné, il s'agit de permettre aux applications d'utiliser différents protocoles de cohérence et de synchronisation ou de rendre transactionnelles les opérations mémoire. On veut également pouvoir réaliser différentes politiques (ou même modèles) de protection en proposant différents modes d'installation d'Arias.
Les objectifs de la conception d'Arias sont donc :

- fournir un support de MVRU générique, qui peut être spécialisé suivant le contexte d'exécution et les besoins des applications,
- proposer un modèle de protection de base, qui permet de réaliser différents modèles de protection spécifiques (par exemple, contrôle d'accès du type Unix ou contrôle d'accès basé sur des rôles),

– fournir aux applications un ensemble de fonctions qui permettent au système d'offrir certaines garanties en cas de pannes,
– intégrer le tout à l'intérieur d'un système existant en permettant l'inter–opérabilité entre les applications d'Arias et celui-ci.

## 5.3 Vue d'ensemble d'Arias

L'architecture de base est constituée d'un ensemble de machines homogènes reliées par un réseau local. Ces machines sont des stations de travail Unix munies des services de répartition (localisation), partage (cohérence et synchronisation), permanence (journalisation et gestion du stockage) et protection (contrôle d'accès).
Certaines machines munies de disques (parfois appelées serveurs), sont chargées du stockage permanent des données gérées dans la MVRU. Les serveurs fournissent une interface homogène pour la gestion de la permanence des données en termes de volumes permanents. La notion de volumes cache les détails du stockage physique des données : les disques et les systèmes de fichiers. Parmi les services de permanence, nous trouvons également un service de journalisation, qui permet la réalisation de protocoles de reprise après panne.
La MVRU ne constitue qu'une partie de l'espace adressable d'une machine, l'autre partie étant réservée aux données privées et aux structures d'exécution de la machine (par exemple le noyau du système Unix). La MVRU sert de support d'adressage des données permanentes mais elle peut également contenir des données volatiles (données partagées persistantes n'ayant pas d'image sur disque). Les structures d'exécution sont les processus du système Unix dans lequel ces services sont intégrés. Le modèle d'exécution d'Arias sera traité plus en détail en 5.4. L'architecture globale du système est illustrée sur la Figure 5.1.

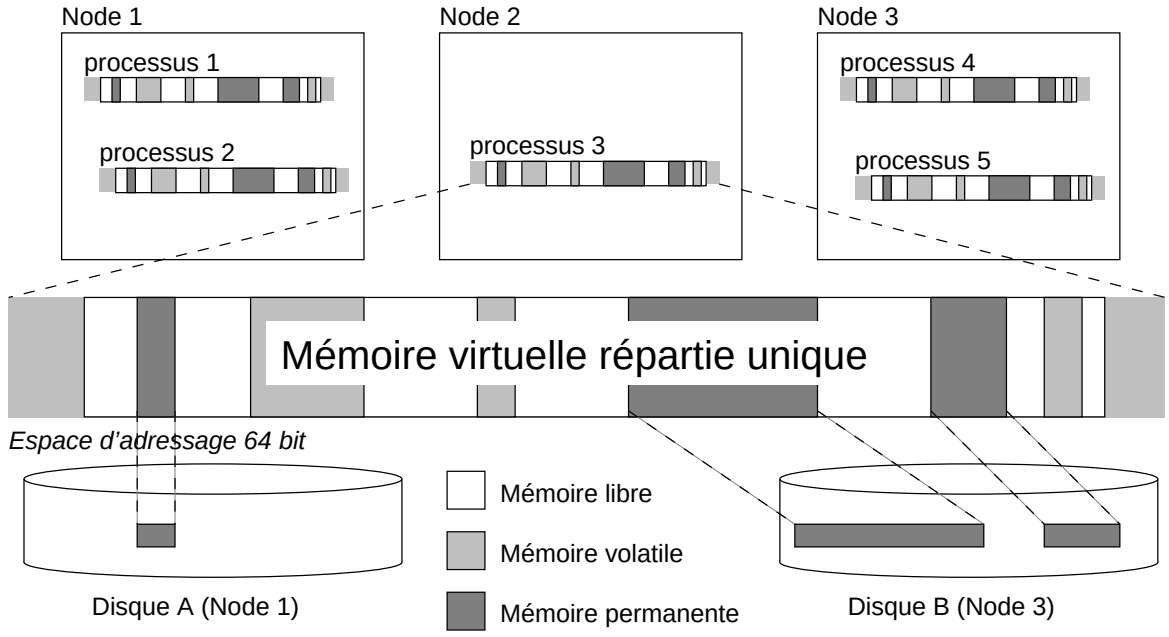



FIG. 5.1: Vue d'ensemble d'Arias

La mémoire est organisée en *segments*, de taille fixée à la création, couplés dans la mémoire à

des adresses fixes. Ainsi, les données partagées sont vues par tous les processus à la même adresse virtuelle, ce qui permet d'utiliser les adresses comme identifiants uniques.
Le segment est une suite contiguë de pages de mémoire virtuelle, identifié par son adresse de base. Il constitue l'unité d'allocation, de partage et de protection dans Arias. L'allocation et la destruction d'un segment sont explicites mais le couplage des segments dans la mémoire d'une application est implicite à la première référence. Pour diminuer le temps d'accès à la mémoire, chaque segment partagé est dupliqué sur toutes les machines qui l'utilisent. Pour traiter les problèmes de cohérence, Arias offre un service de cohérence générique, qui permet la mise en œuvre de protocoles de cohérence spécifiques adaptés aux besoins des applications. Sinon, l'application peut se servir d'une bibliothèque de protocoles préfabriqués.
Arias est conçu pour être intégré avec le système hôte (Unix), ce qui permet à une application qui se sert d'Arias d'utiliser les services et bibliothèques d'Unix et vice versa. Le démarrage d'une application se fait normalement à partir du "shell" d'Unix. Pour qu'une application puisse se servir de la mémoire Arias, il faut que le processus réserve une région de sa mémoire virtuelle pour celle-ci. Cette réservation est d'une taille fixée lors de l'installation d'Arias et doit être la même pour tous les processus. Ainsi, l'espace d'adressage des applications est divisé en deux : une partie réservée à Unix et une autre réservée par Arias (la partie Arias occupe une partie du tas ("heap") d'un processus Unix standard). Cette partition est illustrée sur la figure 5.2.

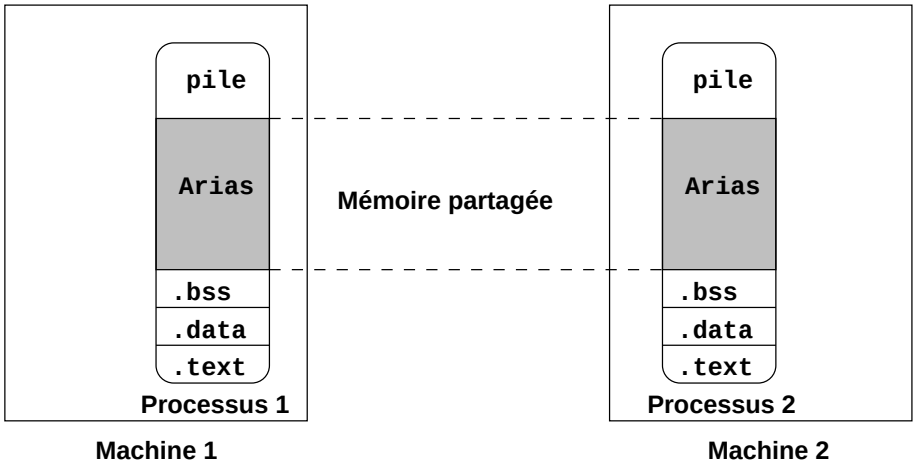



FIG. 5.2: L'espace d'adressage Arias

Le couplage des segments dans la région réservée à Arias se fait dynamiquement. La première référence à une adresse d'un segment Arias génère une exception Unix ("segmentation faut") ; le paginateur Arias couple le segment à l'adresse donnée et l'exécution peut reprendre.

## 5.4 Modèle d'exécution

Il existe deux façons d'utiliser Arias et par conséquent deux modèles d'exécution différents. Arias peut être utilisé soit comme un service traditionnel de mémoire partagée (comme les systèmes mentionnés dans la section 5.1), soit comme support pour de vraies applications à espace d'adressage unique (applications où le code et les données sont stockés dans la

mémoire répartie). Dans le premier cas, Arias n'apporte rien au modèle d'exécution d'Unix, tandis que dans le deuxième cas le modèle implique en fait une intégration entre les structures d'exécution d'Unix et celles d'Arias. Par la suite, nous nous intéressons uniquement à ce dernier type d'applications que nous appelons les applications Arias.
Une application Arias regroupe un ensemble d'activités qui coopèrent pour atteindre un but commun. Une activité consiste en un flot d'exécution quelconque qui s'exécute dans le contexte d'un processus Unix. Toutes les activités qui s'exécutent dans le même processus ont les mêmes droits d'accès comme dans les processus normaux (par exemple ceux d'Unix [Thompson74]), des tâches de Mach [Loepere92a] ou des contextes de Guide-2 [Cayuela92]). Ainsi le processus encapsule les droits d'accès et semble d'être un candidat très intéressant pour la réalisation d'un domaine de protection (cf. 6.3.2).
Arias fournit deux modes de communication entre activités dans deux processus différents, par mémoire partagée ou par appels direct ("IPC") entre activités sur la même machine. Outre ces modes de communication, l'intégration avec Unix permet aux activités de communiquer par tous les moyens de communication fournis par ce système.
Chaque activité se compose d'un ou plusieurs modules de code stocké(s) dans l'espace unique. Ces modules peuvent appeler d'autres modules Arias ou des modules Unix sous forme de librairies partagées.
Pour qu'une application Arias puisse être démarrée depuis Unix, il faut qu'elle contienne une fonction `main` standard Unix qui peut être lancée par l'`exec` standard Unix, normalement appelée par le `shell`. La fonction `main` appelle une fonction d'initialisation Arias (`AriasInit`) et au moins une fonction dans un module Arias. Le `main` constitue ainsi le pont entre le système Unix et le système Arias. Les fonctions Arias peuvent appeler des fonctions Arias ou des fonctions Unix.
Comme tous les objets Arias, une fonction Arias est désignée par son adresse virtuelle. Nous voulons que cette désignation puisse être réalisée soit statiquement par une édition de liens, soit dynamiquement au cours de l'exécution d'une application en passant à un module une adresse virtuelle désignant une fonction.
Une vue globale de ce modèle est donnée sur la figure 5.3.

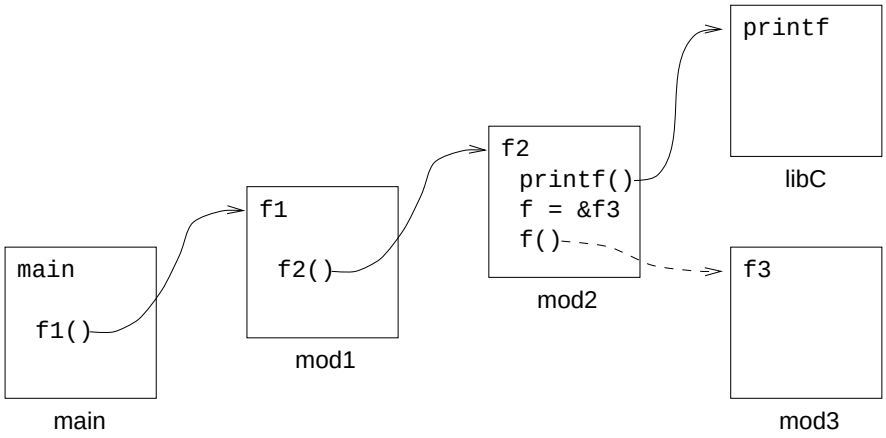



FIG. 5.3: Vue globale d'une exécution Arias.

La figure 5.3 montre comment l'application lancée par Unix (le `main`) appelle la fonction Arias `f1` dans le module `mod1`, cette fonction appelle une autre fonction Arias `f2` dans le

module mod2. Cette fonction appelle la fonction Unix `printf` suivi de l'affectation de l'adresse de la fonction `f3` au pointeur de fonction `f` ; la fonction `f` est ensuite appelée, ce qui provoque l'appel de la fonction `f3` dans le module mod3.
Cet exemple montre comment les liaisons entre modules peuvent être établies statiquement à une adresse fixe (`f1` et `f2`), statiquement à une adresse variable (l'adresse de `printf` est gérée par le chargeur Unix) ou dynamiquement à une adresse fixe (`f = &f3`). En réalité, il est aussi possible de lier des fonctions dynamiquement à des adresses variables, mais cela se passe entièrement dans l'espace Unix.

## 5.5 Réalisation d'Arias

Arias se compose d'un ensemble de services présents sur toutes les machines du système réparti. Ces services sont, pour l'essentiel, intégrés dans le noyau du système Unix par des extensions du noyau ("Loadable Kernel Modules"). Ainsi, ces services profitent de la protection de l'espace du noyau et ils ne peuvent pas être modifiés par les applications. L'architecture logicielle du système est illustrée sur figure 5.4.

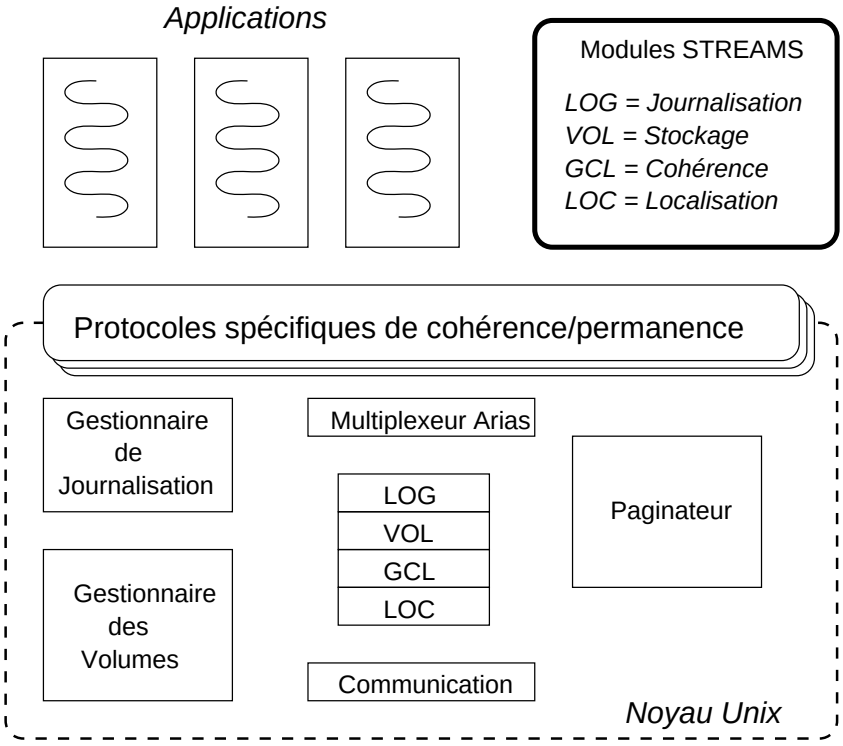



FIG. 5.4: L'architecture logicielle d'Arias

Sur cette figure, on trouve au plus haut niveau les applications qui s'exécutent en espace utilisateur et qui utilisent les services offerts par Arias. L'interaction entre les applications et Arias se fait principalement à travers des modules qui réalisent les différents protocoles spécialisés de cohérence et de synchronisation et les protocoles de permanence. Ces modules permettent d'adapter le système aux besoins particuliers des applications. Les différents modules de cohérence et de synchronisation permettent aux applications de choisir le modèle

le mieux adapté aux besoins de l'application, par exemple une cohérence à l'entrée ("entry consistency") ou une cohérence à la libération ("release consistency"). La conception et la mise en œuvre de la cohérence et de la synchronisation sont traitées plus en détail dans d'autres documents [Cortes95a, Cortes95b, Cortes96a, Cortes96b]. De la même façon, les modules de permanence permettent aux applications de choisir le modèle de permanence le mieux adapté aux besoins de l'application : par exemple ces modules fournissent des primitives pour mettre une image permanente sur disque avant de terminer ou permettre des opérations transactionnelles sur la mémoire [Knaff96].
Les modules spécialisés s'appuient sur une pile de STREAMS [STREAMS90] qui met en œuvre la plupart des services d'Arias. Cette pile des modules STREAMS réalise les services génériques de cohérence et synchronisation ("Generic Consistency Layer (GCL)") utilisés par les modules spécialisés de cohérence et synchronisation, le service générique de journalisation (LOG) et le service de gestion de volume (VOL) qui sont utilisés par les modules spécialisés de permanence, le service de localisation (LOC) qui permet de localiser un segment à partir de son adresse virtuelle.
Outre cette pile de modules STREAMS, nous trouvons quelques services de base, notamment le paginateur Arias. Il s'agit d'un gestionnaire d'exception ("exception handler") qui traite tous les défauts de segment ("segmentation faults") générés par l'application. Si l'adresse d'exception appartient à un segment Arias, il s'occupe de trouver une version courante (peut être dans un volume sur disque) de ce segment et le charge dans la mémoire de l'application. Si l'adresse n'appartient à aucun segment Arias, le paginateur fait suivre l'exception au système Unix (dans le plupart des cas, le gestionnaire d'exception d'Unix va retourner une erreur et arrêter l'application).

## 5.6 Conditions requises pour la protection d'Arias

La réalisation d'une MVRU telle qu'Arias impose certaines conditions pour la protection entre applications et usagers. Ces conditions se divisent en deux groupes principaux, celles qui sont imposées par la gestion d'un espace d'adressage unique et celles qui sont imposées par la coopération entre Arias et le système hôte.
La réalisation d'un espace d'adressage unique nécessite une séparation totale entre adressage et protection. Le fait de pouvoir désigner une structure de données en mémoire ne signifie plus que l'on peut y accéder, comme c'est le cas dans les systèmes traditionnels à base de processus (par exemple Unix).
Le segment est l'unité de gestion des données : les protocoles de cohérence et de synchronisation et les protocoles de permanence sont associés aux segments. Il nous semble donc naturel de lier également la protection aux segments. L'accès à un segment est explicite lors de la création et de la destruction du segment et implicite lors du couplage du segment. Le mécanisme de protection d'Arias doit réaliser un moniteur de références qui intervient lors de ces opérations.
Les droits d'accès aux segments doivent correspondre aux modes d'accès aux segments dans une mémoire partagée normale, c'est–à–dire : lecture, écriture et exécution. Cette interprétation est conforme à la sémantique de `mmap` du système Unix BSD [McKusick96]. Normalement, les droits de lecture et d'écriture s'appliquent aux segments de données et le droit d'exécution aux segments qui contiennent du code.

Dans le chapitre 3, nous avons montré que l'autorisation doit se baser sur les sujets abstraits au lieu de l'identité de l'utilisateur qui a démarré l'application. C'est pourquoi nous proposons de baser le mécanisme d'autorisation sur la notion de domaine de protection (classique dans le modèle à capacités) au lieu de l'identificateur de l'utilisateur courant.
L'interaction possible entre Arias et Unix exige une correspondance entre les domaines d'Arias et les utilisateurs Unix, c'est–à–dire l'identificateur d'utilisateur et du groupe. Ces deux identificateurs sont des caractéristiques du processus Unix utilisées pour réaliser un domaine de protection. Nous devons donc associer à chaque domaine de protection Arias une identification secondaire qui consiste en une paire `<UID,GID>` d'Unix. Cette paire est associé au processus de base d'une application par Unix lors de la création du processus. Les valeurs sont déterminées par le mécanisme d'authentification d'Unix lors de la procédure de `login(1)`. Ces valeurs peuvent donc être utilisées par Arias dans le but d'authentifier l'utilisateur qui lance l'application. Pour les domaines d'Arias qui n'ont pas d'interactions avec Unix, nous proposons d'associer la paire `<"nobody","nogroup">` d'Unix. Nous reviendrons sur l'interaction entre Arias et Unix en 7.1

## 5.7 Récapitulation et discussion

Le modèle de programmation offert par une mémoire virtuelle répartie est plus simple à utiliser que le modèle client/serveur ou le modèle à passage de messages. La MVRU donne une transparence vis–à–vis de la répartition physique et masque la communication entre processus coopérants. Pour ne pas compromettre la transparence ainsi obtenue, le mécanisme de protection d'Arias ne doit pas la remettre en cause.
La réalisation d'une MVRU nous permet d'utiliser les adresses virtuelles comme identificateurs uniques. Ceci nous oblige, d'une part, à réaliser un mécanisme de contrôle d'accès aux données et nous permet, d'autre part, de réaliser ce mécanisme à base de capacités (l'adresse virtuelle peut servir d'identificateur de l'objet). Le mécanisme à base de capacités pour son dynamisme — la facilité avec quelle on peut faire évoluer les droits d'accès — et pour sa généricité — la possibilité de réaliser plusieurs modèles de contrôle d'accès différents — démontré plus loin (cf. 8.2).
Pour pouvoir rendre la mémoire persistante et la partager entre applications, les adresses des données doivent être fixes pendant toute la durée de vie des données.
Les opérations qui doivent provoquer l'intervention du moniteur de référence sont les suivantes : la création et la destruction explicite d'un segment et le couplage implicite d'un segment dans l'espace d'adressage d'un processus.
L'interaction avec le système Unix fournit une authentification d'utilisateur à base des mots de passe, qui se transforme en `<UID,GID>` pour la partie initiale d'une application Arias, c'est à dire la partie `main` qui s'exécute dans l'espace Unix.

# Chapitre 6

# Modèle de protection proposé pour Arias

Le chapitre précédent a présenté le système Arias d'une manière générale et a identifié un certain nombre de conditions requises pour la mise en œuvre de la protection dans une MVRU telle qu'Arias. Dans ce chapitre, nous présentons un modèle de protection pour Arias qui permet de mettre en œuvre un grand nombre de politiques de protection différentes.
La protection a déjà été le sujet de plusieurs études [Hagimont95, Saunier95] dans le cadre du projet Sirac, qui ont conduit à la définition du modèle de protection par capacités cachées [Hagimont96]. Une première thèse a étudié la gestion des capacités dans le noyau du système et l'organisation des domaines de protection [Saunier96]. Cette thèse se concentre sur la gestion dynamique des droits d'accès et sur la réalisation de différentes politiques de protection selon un modèle générique.

## 6.1 Motivations

La motivation pour la réalisation d'un mécanisme de protection dans Arias est de répondre aux problèmes de protection identifiés en 5.6 et de fournir un mécanisme de base pour la réalisation de différentes politiques de protection pour des applications et des systèmes d'exécution différents.
Les conditions requises pour les applications coopératives (le type d'application visé par Arias) sont les suivantes :

1. Les données et les applications peuvent avoir une durée de vie longue ; il est donc important que les droits d'accès puissent évoluer dynamiquement.
2. La nature interactive des applications coopératives implique le besoin de diffusion dynamique des droits entre applications ou entre différents utilisateurs de la même application.
3. Les utilisateurs des applications coopératives peuvent appartenir à différentes organisations dans la même entreprise ou à des entreprises différentes dans une entreprise virtuelle. Il faut donc que le mécanisme de protection permette de mettre en œuvre une politique de méfiance mutuelle entre utilisateurs ou services du système.
4. Pour renforcer tous les mécanismes de protection et diminuer les dégâts potentiels causés par un programme, nous proposons en plus de permettre la mise en œuvre du principe du moindre privilège ("the need–to–know principle").

## 6.2 Objectifs

Nous rappelons les objectifs du modèle de protection mentionnés en 1.3.

1. *Généricité du modèle.* La généricité a été un but principal dans la conception d'Arias. Pour cela il est naturel que le support pour la protection puisse être modifié selon le mode d'installation d'Arias.
2. *Facilité de programmation.* Le modèle doit être facile à comprendre et utiliser, sinon il ne sera pas utilisé ou les spécifications risquent d'être fausses [1].
3. *Évolutivité.* La spécification de la protection d'une application doit pouvoir changer pendant la durée de vie de l'application, sans recompilation ni re-édition des liens.
4. *Réutilisabilité des composants logiciels.* Les mêmes composants logiciels peuvent être utilisés dans des contextes et des applications différents.
5. *Possibilité d'intégration de logiciels conçus ailleurs.* Les logiciels conçus ailleurs peuvent être encapsulés par le système et leur interface avec celui–ci ou avec d'autres activités dans Arias doit être contrôlée par celui qui installe l'application.

### 6.2.1 Généricité du modèle

Arias a été conçu comme un support générique pour les applications et les systèmes d'exécution répartis. Les classes d'applications ciblées par le prototype sont très variées comme les applications coopératives, les systèmes de gestion d'une base de données répartie ou le support dans le noyau d'un système de fichier réparti.
La généricité d'Arias se manifeste par des interfaces génériques qui permettent aux programmeurs d'étendre le système avec leur propres modules qui adaptent le système aux besoins de l'application. Pour l'instant il est possible pour un programmeur d'adapter les protocoles de cohérence et de permanence de la mémoire Arias. Ceci permet de réaliser un large spectre d'applications différentes.
Ces applications ont des besoins de protection différents : les applications coopératives ont besoin de protéger les données d'un utilisateur contre les modifications non-autorisées par les autres utilisateurs ; par contre le système de fichier réparti gère lui–même le contrôle d'accès et il n'a pas besoin d'un support de protection particulier. Il semble donc naturel de prendre la même approche, c'est à dire la généricité, par rapport à la protection.
Ceci nécessite un modèle de protection capable de s'adapter aux besoins des classes d'applications différentes. Plus précisément nous voulons un modèle suffisamment général pour permettre la réalisation de mécanismes différents qui mettent en œuvre la plupart des modèles de contrôle d'accès décrits au chapitre 3.
La généricité du modèle se traduit par une approche minimaliste au niveau du modèle et cela dirige certains choix de réalisation.

### 6.2.2 Facilité de programmation

La MVRU assure une transparence vis–à–vis de la répartition des données qui facilite la programmation des applications réparties. Le modèle de protection proposé pour Arias doit

[1] Une mauvaise spécification de protection est pire qu'une spécification manquante, parce que cela donne un faux sentiment de sûreté.

faciliter la programmation des applications réparties de la même façon, c'est–à–dire rendre la protection la plus transparente possible pour le programmeur. Le programmeur doit uniquement s'occuper des aspects algorithmiques de l'application et confier la spécification de la protection à un administrateur système ou au responsable de la sécurité.

La question de transparence est une question récurrente dans la conception des systèmes répartis. Nous ne croyons pas à une transparence absolue, mais on peut espérer rendre transparente la politique de protection spécifique. Nous reviendrons à la transparence dans la section 6.3.7.

### 6.2.3 Évolutivité

L'évolution des droits d'accès doit être considérée en deux temps : pendant l'exécution d'une application et entre deux exécutions de la même application (ou deux applications différentes qui manipulent les mêmes données).

Les applications coopératives se caractérisent entre autre par un fort taux de partage entre utilisateurs et un ensemble d'utilisateurs très dynamique. L'échange des données entre utilisateurs dans un groupe de taille variable nécessite un modèle de protection flexible.

Les applications et les informations stockées dans le système Arias peuvent avoir une longue durée de vie et, pendant cette durée de vie, les données peuvent être utilisées par des applications différentes ou dans des contextes d'exécution différents. Il faut donc que les droits d'accès associés aux segments d'Arias puissent évoluer dynamiquement

En résumé, l'évolutivité demande un modèle de protection flexible et très dynamique. C'est pourquoi nous proposons un modèle de protection à base de capacités.

### 6.2.4 Réutilisabilité des composants logiciels

La recherche en génie logiciel se concentre aujourd'hui beaucoup sur la réutilisation du code et la programmation par composition des composants déjà existants. Parmi les résultats de cette recherche, on trouve les langages de configuration d'applications et de description d'architectures logicielles comme Olan [Bellissard97] ou Aster [Issarny97] ou les plates–formes pour exécuter les applications composées comme JavaBeans [Monson-Haefel99], CORBA [Geib97] et COM [Kirtland98].

Nous souhaitons profiter de notre MVRU pour pouvoir réutiliser la plupart des modules développés pour Arias. Une MVRU donne une visibilité globale aux modules de code (composants), autrement dit "un espace d'adressage unique rend le partage facile, mais rend le non–partage difficile".

Un module de code peut en même temps servir à l'intérieur d'une application ou offrir un service à des clients différents dans un scénario client/serveur. Dans le premier cas la spécification de la sécurité du module hérite de la spécification de l'application, dans la deuxième cas la spécification de la sécurité de l'interaction entre clients et serveur doit être exprimée explicitement. La spécification de la sécurité doit donc être découplée de la spécification du code.

### 6.2.5 Possibilité d'intégration de logiciels conçus ailleurs

L'intégration de logiciels conçus ailleurs peut se faire à deux niveaux : soit par l'intégration d'Arias avec Unix, soit en mettant les logiciels conçus ailleurs dans la mémoire Arias et en les utilisant comme des modules Arias. Cette dernière méthode permet de soumettre ces logiciels au modèle de protection Arias.
Les primitives de la gestion du code d'Arias permettent d'extraire le code d'une bibliothèque dynamique UNIX et de le mettre dans un module de code Arias. Ensuite, ce module peut être lié avec d'autres modules Arias pour constituer une application.

### 6.2.6 Discussion

Le modèle à capacités a principalement été choisi pour sa souplesse et son dynamisme pour la distribution des droits d'accès.
La programmation par composants introduit deux menaces contre la sécurité : le composant risque d'être remplacé par un cheval de Troie et le composant n'est pas garanti de réaliser une politique de protection adéquat. Pour protéger l'appelant contre un cheval de Troie, il faut que le système garantisse l'authenticité du composant. Le risque qu'un composant ne réalise pas une politique de sécurité adéquate, nécessite une suspicion mutuelle entre composants. De plus, le système doit impérativement respecter le principe du moindre privilège.
Nous allons d'abord rappeler les raisons pour utiliser les capacités, qui ont été données au chapitre 3, et montrer comment les capacités répondent aux objectifs listés ci–dessus.

## 6.3 Modèle des capacités cachées

Le modèle de protection par capacités cachées a été développé pour répondre aux besoins de la protection dans Arias et atteindre les objectifs listés ci–dessus. Il s'agit d'une extension du modèle classique des capacités confinées où la gestion des capacités se fait par le système.

### 6.3.1 Vue d'ensemble du modèle

Le modèle des capacités cachées est fondé sur le modèle classique de protection par capacités et les éléments du modèle d'exécution d'Arias.
Le droit d'accéder à un objet protégé est accordé à un domaine de protection en possession d'une capacité pour l'objet. Le *domaine de protection* (ou simplement le domaine) est un sujet abstrait qui correspond à la définition classique du domaine de protection (par exemple la définition originale de la *sphère de protection* par Dennis & Van Horn).
La gestion et la vérification des capacités sont faites par le système d'une façon transparente pour les applications. Le modèle respecte une version modifiée du principe de la tranquillité défini par exemple par le modèle de Bell & LaPadula où la classification d'un objet ne peut pas changer pendant que l'objet est utilisé. Dans un modèle à capacités, ceci se traduit par le fait qu'une capacité n'est pas révocable pendant l'utilisation de l'objet qu'elle désigne. Nous proposons de permettre la révocation de la capacité sans supprimer le droit d'accès déjà accordé au sujet. Ceci nous permet de réaliser un contrôle d'accès beaucoup plus efficace où les capacités sont vérifiées lors du premièr accès à un objet protégé.

L'intégration avec le modèle d'exécution d'Arias se fait de la façon suivante : une activité s'exécute à tout moment dans un domaine de protection qui définit les droits d'accès accordés à l'activité. Ces droits sont déterminés par les capacités stockées dans une liste de capacités (C–liste) associée au domaine. Au cours de son exécution, une activité particulière peut évoluer entre plusieurs domaines selon les besoins et la structure de l'application.
La définition des domaines de protection et les interactions entre domaines se fait indépendamment du code de l'application dans les interfaces de protection (voir 6.3.7 pour les détails).
Les applications peuvent avoir plusieurs activités réparties s'exécutant dans le même domaine de protection et/ou plusieurs activités s'exécutant sur la même machine mais dans des domaines de protection différents. Cette structuration est illustrée par la figure 6.1.

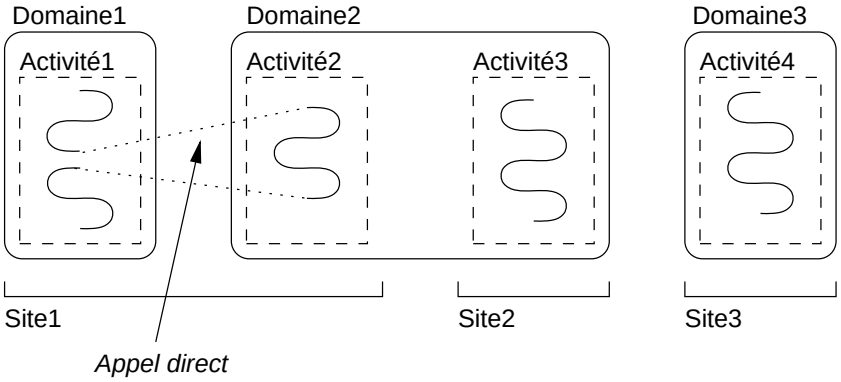



FIG. 6.1: Applications d'Arias protégées

La figure montre une application composée de quatre activités différentes s'exécutant dans trois domaines de protection différents et réparties sur trois machines. Les deux activités situées sur la même machine (Activité1 et Activité2) communiquent par appel direct tandis que la communication avec les autres activités se fait uniquement par mémoire partagée. Deux activités (Activité2 et Activité3) s'exécutent dans le même domaine de protection mais sur deux machines différentes.
L'appel direct entre deux domaines de protection permet l'amplification ou la dégradation temporaire des droits d'accès d'une activité en passant par un domaine plus ou moins privilégié.
Les listes de capacités associées à chaque domaine peuvent contenir deux types de capacités : des capacités d'accès et des capacités de changement de domaine. Les capacités d'accès permettent aux activités d'accéder aux objets protégés dans le domaine de protection courant et les capacités de changement de domaine permettent aux activités de faire un appel direct à un point d'entrée dans un autre domaine de protection. Les domaines de protection, les capacités et les appels entre domaines sont décrits plus en détail dans la suite.

### 6.3.2 Domaines de protection

Un domaine de protection constitue une enceinte de protection qui décrit l'ensemble des objets accessibles aux activités qui s'exécutent dans cette enceinte, ainsi que les droits en vigueur au sein du domaine pour tous les accès à ces objets. Le domaine définit donc notamment l'ensemble de capacités ou de C–listes disponibles pour une activité à un instant donné en

identifiant une C–liste principale au domaine. Cette liste peut inclure d'autres listes et représente ainsi par transitivité un arbre de capacités, c'est pourquoi nous appelons cette C–liste la liste racine du domaine.
La création et la destruction d'un domaine de protection sont des opérations explicites dans le modèle. La création d'un domaine se fait à partir d'une C–liste existante qui devient la C–liste racine du domaine. Une C–liste peut ainsi servir à la création d'un seul domaine.
La destruction d'un domaine supprime l'association entre la C–liste et le domaine. Même s'il n'y a plus de domaine associé à la C–liste, elle peut toujours appartenir à l'arbre de capacités d'un domaine différent ; nous ne pouvons donc pas détruire la C–liste avec le domaine. Nous remarquons que s'il n'y a pas de référence vers la C–liste, la mémoire occupée par la liste est effectivement inaccessible et peut être ramassée par le système.
Un domaine peut être réparti afin de profiter des possibilités de parallélisme physique ou pour permettre aux activités s'exécutant sur des machines différentes d'avoir les mêmes droits d'accès.
Une activité peut évoluer au cours de son exécution d'un domaine de protection à un autre. Cependant, pour conserver l'étanchéité de ces domaines de protection, il est nécessaire de contrôler le passage d'un domaine à un autre. Ce contrôle se situe à deux niveaux : les points d'entrée qui permettent aux activités de transférer le contrôle au domaine, et le droit de transférer le contrôle à travers ces points d'entrée. Ainsi une activité n'a pas le droit de franchir le domaine de protection par des points d'entrée arbitraires, mais seulement par des points d'entrée définis pour le domaine. Ce sont exclusivement ces points d'entrée qui permettent à une activité de changer de domaine et qui peuvent ainsi servir de guichets de contrôle pour valider ou rejeter l'entrée d'une activité. Le changement de domaine et sa réalisation dans Arias sont décrits plus en détail en 7.8.
La figure 6.2 donne une vision globale des domaines et des arbres de capacités.

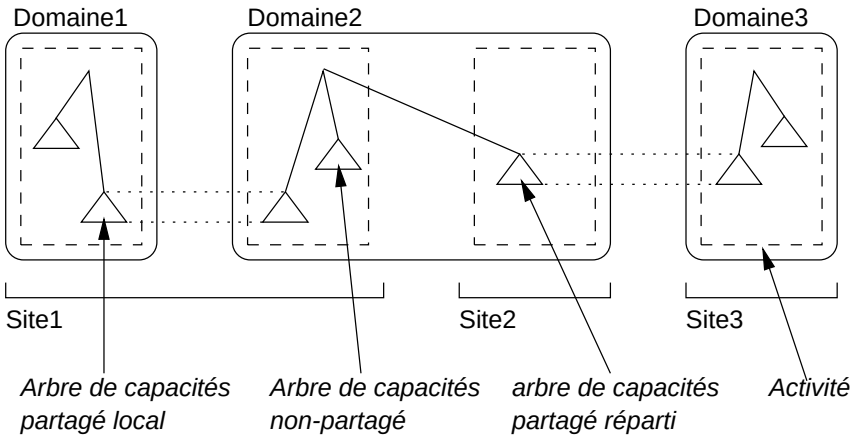



FIG. 6.2: Les domaines de protection

Cette figure montre la même application que la figure 6.1. Les deux activités sur `Site1` partagent localement un sous–arbre de capacités entre les deux domaines `Domaine1` et `Domaine2` et ils ont chacun un sous–arbre de capacités non–partagées. Les deux activités dans `Domaine2` s'exécutent sur deux sites différents mais elles ont le même arbre de capacités à leur disposition. La dernière activité s'exécute dans un domaine et sur une machine à part. Cette activité partage en réparti un sous–arbre de capacités avec `Domaine2`.

Le fait que les C–listes peuvent être partagées en réparti requiert une synchronisation et une mise en cohérence entre machines qui accèdent aux mêmes C–listes. Ces synchronisations et mises en cohérence ne font pas partie du modèle, mais doivent être réalisées par la mise en œuvre du modèle.

### 6.3.3 Changement de domaine

Le changement de domaine est un mécanisme qui permet un changement temporaire des droits d'accès d'une activité. Le changement de domaine prend la forme d'un appel de procédure qui transfère l'activité du domaine d'origine (le domaine appelant) vers un domaine qui contient les droits souhaités (le domaine appelé). L'activité s'exécute avec les droits du domaine appelé pour la durée de l'appel puis, au retour, les droits d'origine sont rétablis. Ainsi le changement de domaine est comparable à un appel à distance normal dans les architectures client/serveur. C'est pourquoi nous appelons le domaine appelant le client et le domaine appelé le serveur. Une activité peut traverser plusieurs domaines de protection au cours de l'exécution d'une application. Ceci permet à chaque instant à l'activité de s'exécuter avec les privilèges minimaux nécessaires pour réaliser sa tâche.
Le changement de domaine permet également la construction de systèmes extensibles (à partir des services protégés).
L'appel d'un service protégé nécessite souvent le transfert de paramètres entre appelant et appelé. Pour les paramètres passés par référence, le mode normal pour une MVRU, il peut aussi être nécessaire de passer le droit d'accès aux données référencées par les paramètres. Les droits à transférer avec les paramètres sont exprimés dans une interface de protection.
Le changement de domaine est une opération privilégiée qui nécessite l'intervention du système. La structure d'un changement de domaine est montrée sur la figure 6.3.

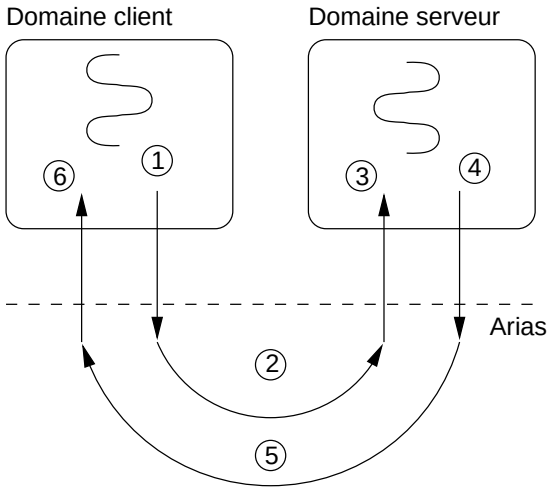



FIG. 6.3: La structure du changement de domaine

L'activité qui s'exécute dans le domaine client appelle une procédure qui doit être exécutée dans un domaine distant (le domaine serveur). Le sytème intercepte ce appel et le transforme en un appel système pour changer de domaine (1). L'appel système identifie le domaine serveur et transfère les paramètres et les droits d'accès selon l'interface de protection(2). Après avoir transféré les droits d'accès, le système fait un "upcall" dans le domaine serveur, qui

appelle la procédure appropriée avec les paramètres appropriés (3). La procédure s'exécute normalement mais le système intercepte le retour de procédure pour retourner au domaine appelant (4). Le système transfère les paramètres du retour et les droits d'accès associés à ces paramètres selon l'interface de protection (5). Puis le système retourne normalement dans le domaine appelant comme si la procédure avait été appelée localement.
La réalisation du changement de domaine est décrite plus en détail en 7.8.

### 6.3.4 Capacités d'accès

Une capacité d'accès permet à une activité d'accéder à un objet dans le domaine de protection courant. Toutes les activités qui s'exécutent dans le même domaine peuvent ainsi accéder à l'objet.
Le modèle suppose l'utilisation de capacités confinées, c'est–à–dire de capacités gérées par le système et stockées hors de portée des sujets. Cette gestion des capacités peut être réalisée par le noyau du système ou par un moniteur de référence externe au noyau.
En réalité le modèle va plus loin en se donnant pour but de permettre une transparence totale de l'utilisation des capacités par rapport aux programmeurs des applications. Le modèle propose une vérification implicite des capacités au moment du premier accès à un objet protégé. À ce moment, le système vérifie que le domaine courant de l'activité contient une capacité pour accéder à l'objet. Donc, il n'y a pas de présentation explicite de la capacité comme dans Amœba ou dans Mach, ni de besoin d'une commande explicite pour pouvoir accéder au objet (comme l'`open(2)` d'Unix ou l'`attach()` d'Opal).
La transparence de l'utilisation des capacités rend superflue une interface de programmation pour la protection et par conséquent le modèle ne prévoit aucune interface de programmation pour permettre la manipulation explicite des capacités par les activités. Ceci dit, le modèle n'interdit pas la réalisation d'une telle interface par une instance donnée du modèle.
L'utilisation des capacités d'accès dans le contexte d'Arias est décrite plus en détail en 7.7.

### 6.3.5 Capacités de changement de domaine

Le changement de domaine est une opération privilégiée. Pour pouvoir effectuer un changement de domaine, il faut que le domaine appelant possède le droit de le faire ; ce droit existe sous la forme d'une capacité de changement de domaine. La capacité de changement de domaine permet à l'activité de changer son domaine de protection à travers un point d'entrée prédéfini.
La capacité de changement de domaine identifie : le point d'entrée, le domaine de protection appelé et l'interface de protection à utiliser pour transférer des capacités entre les deux domaines.
La réalisation du changement de domaine est décrite plus en détail en 7.8.

### 6.3.6 Listes de capacités

Chaque C–liste contient un ensemble de droits qui peuvent constituer la base d'un domaine de protection. Ceci nous permet de gérer les domaines de protection et les C–listes de la même façon. Les droits d'utiliser, inspecter ou modifier les C–listes doivent être protégés par le système, c'est–à–dire par des capacités.

Les droits d'accès pour une C–liste doivent comprendre les droits suivants : utiliser une capacité, rajouter une capacité, copier une capacité dans une autre C–liste (soit la capacité entière, soit une restriction de la capacité), lire la C–liste et supprimer/retirer une capacité.
On peut également envisager de rajouter le droit de retirer certaines capacités marquées. Par exemple un domaine peut supprimer les capacités qu'il a rajoutées dans une C–liste partagée. Ceci peut aussi faciliter la tâche de retirer les capacités passées en paramètre lors d'un changement de domaine (un domaine appelant peut supprimer, au retour de l'appel, les capacités transférées à l'appel dans la C–liste du domaine appelé). Pour l'instant ces droits ne sont pas bien définis et nous avons trouvé un autre moyen de retirer les capacités au retour d'un appel de changement de domaine.
Pour rajouter une capacité dans une C–liste, il faut que le domaine possède le droit de copier la capacité dans la C–liste source et le droit de rajouter une capacité dans la C–liste destination.
Les capacités de changement de domaine posent un autre problème. Un serveur peut avoir besoin de construire une capacité de changement de domaine pour un service qui n'existait pas auparavant, mais il faut empêcher le serveur de fabriquer une capacité quelconque. Sinon la capacité fabriquée peut être utilisée pour détourner les appels selon le schéma suivant : si un domaine de protection possède le droit de rajouter une capacité dans une C–liste partagée, il peut rajouter une capacité de changement de domaine pour une fonction fréquemment utilisée (par exemple un appel au service de nom) qui remplace le domaine de destination par un domaine piraté ; toutes le capacités passées en paramètre à cette fonction sont alors transférées au domaine piraté. Pour éviter ce problème, le droit de rajouter une capacité de changement de domaine est restreint aux cas suivants : les copies des capacités de changement de domaine existantes, les capacités dérivées d'une capacité d'exécution existante (ceci permet à un domaine d'exporter des capacités mais empêche la fabrication des capacités pour les fonctions fréquemment utilisés — la fonction locale n'a pas la bonne adresse) ou une capacité générée par l'administrateur du système. L'administrateur du système est dans ce contexte quelqu'un qui possède une capacité d'écriture sur le module de code et le droit de rajouter des capacités dans la C–liste du domaine appelé.

### 6.3.7 Interfaces de protection

La transparence de la protection par rapport aux programmeurs est l'innovation principale du modèle de protection à capacités cachées. Cette transparence est obtenue par l'utilisation des capacités confinées, et par la façon de faire évoluer les droit d'accès — c'est–à–dire l'échange des droits d'accès pendant les changements de domaine.
L'échange des droits se spécifie comme une extension à la description d'interface avec l'IDL ("Interface Definition Language") utilisé pour la construction de l'application. Nous appelons cette description d'échange des droits une *interface de protection*.
L'interface de protection doit identifier le point d'entrée dans le domaine appelé, et les droits passés entre appelant et appelé. Les droits d'accès échangés, lors d'un changement de domaine, sont exprimés en termes de capacités d'accès et capacités de changement de domaine.
Pour respecter la suspicion mutuelle et conserver l'étanchéité entre les domaines, l'interface de protection doit permettre à l'appelant et à l'appelé d'exprimer chacun leur propre vue sur les droits à échanger. La liste exhaustive des contrôles sur l'échange des capacités peut s'exprimer par le tableau suivant :

| | pour le domaine appelant | pour le domaine appelé |
|---|---|---|
| à l'appel | droits offerts (capacités sortantes) | droits nécessaires (capacités entrantes) |
| au retour | droits nécessaires (capacités entrantes) | droits offerts (capacités sortantes) |

L'installation d'une interface de protection s'effectue donc avec la participation du client et du serveur, chacun spécifiant une partie des contraintes pour leur interaction. Il faut donc que l'interface de chaque coté soit incluse dans la capacité de changement de domaine. La capacité de changement de domaine fixe ainsi les droits qui peuvent être échangés lors d'un changement de domaine. Le contrôle des interfaces de protection (le contrôle des capacités de changement de domaine) permet donc la réalisation du confinement parce qu'un domaine ne peut pas passer plus de droits d'accès que l'interface de protection n'en spécifie.
La réalisation des deux interfaces séparées permet de bien réaliser la suspicion mutuelle. En pratique, les deux interfaces peuvent être spécifiées par un administrateur du système ou un responsable de la sécurité (celui qui installe l'application) qui prend en compte les intérêts de chaque partie.

### 6.3.8 Changement dynamique de domaine

Le changement de domaine décrit ci–dessus permet d'associer une procédure dans un composant avec un domaine de protection. À chaque appel de la procédure, l'activité est transférée vers le domaine spécifié.
Dans les applications coopératives, les données sont souvent structurées en différents domaines selon l'utilisateur responsable de la gestion des données. Si plusieurs utilisateurs gèrent les mêmes types de données, l'application va avoir besoin d'appeler les mêmes fonctions dans différents domaines de protection. Ceci est uniquement possible si le système arrive à faire une liste des domaines potentiels (les domaines vers lesquelles le domaine courant possède une capacité de changement de domaine) et à distinguer entre ces domaines au moment de l'appel.
La solution proposée ici repose sur la gestion des capacités et des interfaces de protection. D'abord nous proposons de permettre d'associer plusieurs capacités de domaine au même point d'entrée. Ceci permet au domaine appelant d'appeler la même procédure dans des domaines différents. La gestion des capacités est décrite plus en détail en 7.2.
Prenons l'exemple d'un correcteur d'orthographe dans un éditeur coopératif. Chaque chapitre d'un document se trouve dans un domaine de protection lié à l'usager (ou groupe) qui est responsable du chapitre. Pour faire une correction orthographique globale il faut donc appeler le correcteur d'orthographe dans les différents domaines de protection.
Pour distinguer entre domaines, il faut d'abord faire l'observation que la différence principale entre les domaines porte sur l'ensemble de droits qu'ils possèdent. Au niveau fonctionnel, il n'y a pas de différence entre appeler une procédure dans le domaine A ou le domaine B si tous les deux arrivent à accéder aux données auxiliaires (les données — outre les paramètres — nécessaires au bon déroulement de l'appel). Les deux procédures sont identiques, donc les resultats obtenus par les deux appels sont identiques. Alors la distinction doit se faire au niveau des droits sur les données auxiliaires gérés dans le domaine.
Une première solution pourrait être de faire l'appel et de revenir en arrière si l'appel ne peut pas se dérouler correctement, puis de chercher une autre capacité de domaine et refaire l'appel, etc. Cette solution nous garantit le déroulement correct, si possible avec les droits de l'appelant, mais elle n'est pas très efficace s'il y a un grand nombre de domaines à essayer et il faut que la

fonction soit idempotente. Nous avons donc tout de suite decidé d'abandonner cette piste.
Une meilleure solution se base sur l'observation que les données auxiliaires sont normalement identifiées par une ou plusieurs références passées en paramètre à l'appel. Nous proposons d'étendre la specification des interfaces de protection avec une liste des références et les droits nécessaires pour le déroulement correct de l'appel. Pour illustrer la solution, nous revenons à l'exemple d'un correcteur d'orthographe (cf. figure 6.4).

```
correcteur_orthographe(char *chapitre);
```

a) La signature de la correcteur d'orthographe

```
correcteur_orthographe(CAPA_WRITE char *chapitre);
```

b) L'interface du correcteur d'orthographe

FIG. 6.4: L'exemple d'un correcteur d'orthographe

La signature de la fonction qui réalise le correcteur d'orthographe est montré sur la figure 6.4 a). L'interface de protection étendue est montré sur la figure 6.4 b). Cette interface spécifie que le domaine appelé doit possèder une capacité d'écriture pour l'adresse passé en paramètre (le `chapitre`). Le système cherche d'abord un domaine de protection qui possède une telle capacité parmi les domaines potentiels (ceux pour lesquels le client possède une capacité de changement de domaine) avant effectuer l'appel.
La spécification complète des interfaces de protection (avec le changement dynamique de domaine) est décrite en 7.9 et la réalisation du changement de domaine en 7.8.

## 6.4 L'exemple d'un serveur d'impression

Les principes du modèle sont illustrés à l'aide d'un exemple : un serveur d'impression.
Le serveur d'impression a besoin d'avoir un accès exclusif à l'imprimante pour éviter que plusieurs processus n'impriment en même temps et que les lignes se mélangent. Pour cela, un domaine de protection est créé pour le serveur d'impression qui contient le droit d'accéder à l'imprimante. Le serveur d'impression exporte un point d'entrée qui permet aux clients d'appeler la procédure `print(char *texte)` dans le domaine du serveur.
Les clients peuvent imprimer leurs textes sur l'imprimante, à travers ce point d'entrée, en donnant en paramètre une référence au texte. Pour pouvoir imprimer le texte, le serveur d'impression doit avoir le droit de lire le segment qui contient le texte. Le serveur peut contenir le droit de lire tous les segments, mais cela est en conflit avec le principe du moindre privilège, donc nous proposons que le client fournisse une capacité de lecture avec le pointeur.
Le point d'entrée est donc constitué de l'adresse de la procédure et de l'interface de protection (la signature de la procédure et les droits à transférer entre domaines à l'appel ou au retour.)
Une capacité de changement de domaine est distribuée à tous les clients potentiels du serveur (ou dynamiquement à travers un service de désignation). Cette capacité permet aux clients de

passer de leur domaine au domaine du serveur. L'appel du serveur d'impression est montré sur la figure 6.5.

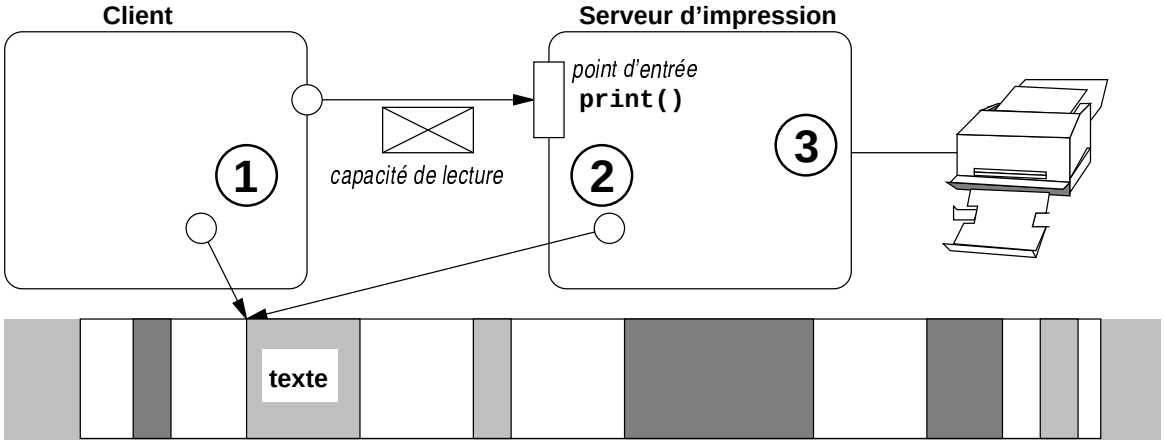



FIG. 6.5: Serveur d'impression

Quand un client appelle la procédure `print(char *texte)`, le système vérifie d'abord que le client possède la capacité de changement de domaine appropriée, puis l'interface de protection emballe les paramètres et transfère les droits d'accès nécessaires et les envoie au serveur (1). Le serveur déballe les paramètres et installe la capacité dans le domaine local (2), puis le serveur imprime le texte sur l'imprimante (3) grâce à la capacité de lecture reçue. Au retour de l'appel le système doit supprimer la capacité dans le domaine du serveur.

## 6.5 Récapitulation

Dans ce chapitre nous avons identifié les besoins de protection pour une classe d'applications visée par le système Arias : les applications coopératives réparties.
Nous avons analysé les implications pour le modèle de protection avec les objectifs mentionnés en 1.3.
La notion de domaine de protection permet de construire des services extensibles, parce que le fonctionnement d'un serveur peut être encapsulé et protégé.
La principe de tranquillité implique que le contrôle d'accès peut se faire uniquement lors du premier accès aux objets.
Il permet également un taux de partage des données plus important entre utilisateurs (domaines de protection) mutuellement méfiants, parce qu'un domaine peut contrôler les opérations sur les données en exportant une capacité de changement de domaine au lieu du droit d'accéder directement aux données.
Pour conclure, la spécification du modèle que nous avons présentée est volontairement générale. En effet, le modèle de protection proposé dans cette thèse est relativement peu dépendant de son contexte d'implantation (le système Arias) et nous verrons en conclusion que ce modèle se prête bien à d'autres environnements répartis.

# Chapitre 7

# Réalisation du modèle proposé

Ce chapitre présente un compte–rendu détaillé de la réalisation du modèle de protection dans Arias. Cette présentation est importante parce qu'elle définit et précise le modèle dans le contexte d'un MVRU, mais aussi parce que la réalisation du mécanisme de protection est décisive pour la sécurité du système (cf. 2.9).

Avant de parler de la réalisation du modèle, nous reprenons le modèle d'exécution d'Arias et étudions l'impact de la décision de réaliser Arias d'une façon interopérable avec le système hôte.

Ensuite nous analysons la gestion des capacités qui était omise dans le chapitre 6 afin de garder la généralité de la description du modèle. La réalisation du modèle dans le contexte d'Arias nécessite des précisions sur le format et la gestion des capacités, mais aussi sur la délégation et la révocation des capacités. Ces précisions sont données dans ce chapitre.

Enfin nous décrivons l'utilisation des capacités pour le couplage des segments d'Arias et les changements de domaine. Ceci nous permet enfin de caractériser le modèle selon la taxonomie introduite en chapitre 4.

## 7.1 Interopérabilité entre Arias et Unix

L'interopérabilité entre Arias et Unix a été un but important dans la conception d'Arias. Cela implique que la réalisation d'Arias soit uniquement constituée de composants Unix, principalement des bibliothèques logicielles, des extensions du noyau et des programmes Unix pour développer et installer les applications Arias.

L'intégration a pour conséquence que tout élément du modèle d'exécution Arias doit avoir un correspondant dans le système Unix qui le réalise. Dans la suite nous rappelons le modèle d'exécution d'Arias et étudions comment cette interoperabilité affecte la réalisation d'Arias.

Une application Arias est composée d'une ou plusieurs activités. Les activités exécutent le code stocké dans les segments Arias ou dans les bibliothèques partagées Unix. Les modules de code sont couplés dans l'espace d'adressage de l'activité selon les besoins.

Nous précisons dans ce paragraphe la définition des applications, domaines de protection, activités et modules de code.

### 7.1.1 Applications

Une application Arias se compose d'une ou plusieurs activités qui peuvent s'exécuter sur des sites différents et avec des droits d'accès différents.

### 7.1.2 Activités

Une activité d'Arias décrit un flot de contrôle particulier dans un domaine de protection. Elle correspond donc à la notion de "thread" dans le système AIX ; nous avons donc choisi de réaliser les activités par les "threads" natifs du système.
Une activité peut faire un changement de domaine temporaire pour exécuter certains opérations dans un domaine différent (processus).

### 7.1.3 Domaines de protection

Un domaine de protection est dit passif s'il n'y a pas d'activité associée au domaine ou actif s'il y en a. Le domaine consiste en un ou plusieurs *contextes* sur les sites ou le domaine est actif.
La notion de contexte ressemble beaucoup à la notion de processus dans le système hôte (AIX). Plusieures activités peuvent s'exécuter en parallèle avec les mêmes droits d'accès. Ces droits incluent en particulier les droits d'accès à un ensemble de pages dans la mémoire.
Un domaine peut avoir plusieurs contextes sur le même site. Ils s'exécutent tous avec le mêmes droits d'accès parce qu'ils partagent la C–liste de base du domaine. Il existe deux façons de créer un contexte, soit comme domaine initial d'une application Arias, soit comme un serveur de domaine. Ces deux types de contextes sont décrits dans la suite.

#### Domaine initial

Nous rappelons qu'une application Arias est lancée par un processus Unix. Ce processus fournit le contexte d'exécution par rapport au noyau d'Unix. L'appel d'initialisation d'Arias transforme ce processus en contexte local du domaine. Cette transformation se compose principalement de l'initialisation des structures de gestion de mémoire Arias et de l'association d'un domaine de protection initial au processus. Si l'usager démarre plusieurs applications Arias en parallèle plusieurs contextes du même domaine vont être créés. La création du domaine initial est décrite plus en détail en 7.6.1.

#### Serveur de domaine

Il existe un seul serveur de domaine par site où le contexte est actif. Le rôle du serveur est d'exécuter tous les appels à ce domaine sur le site. Pour éviter que le serveur devienne un goulot d'étranglement, il gère un nombre d'activités pour pouvoir répondre à plusieurs requêtes en parallèle. Toutes les activités dans le serveur s'exécutent avec la même C–liste de base, mais les capacités passées en paramètre au serveur sont uniquement accessibles par l'activité qui traite l'appel. La création d'un serveur de domaine est décrite en plus de détail dans 7.6.2.

**Interopérabilité Arias - Unix**

Pour pouvoir associer un domaine initial au processus, Arias gére une liste de tous les usagers et de leurs domaines de protection initiaux.
Pour qu'un serveur de domaine puisse interagir avec Unix, il faut trouver un moyen d'associer les droits d'accès d'Unix (définis par un UID et un GID) au domaine. Soit l'UID et GID de l'utilisateur créateur sont associés au domaine, soit une paire <UID, GID> est prédéfinie pour chaque domaine, par exemple à la création du domaine.
Nous avons choisi d'associer une paire <UID, GID> à chaque domaine parce que cela correspond mieux à la sémantique d'un domaine dans notre modèle — toutes les activités qui s'exécutent dans le même domaine ont les mêmes droits d'accès sur les objet Arias et Unix. En gardant l'identité de l'utilisateur créateur du domaine, différentes activités ont des droits d'accès différents.

### 7.1.4 Modules de code

Les modules de code sont comparables aux bibliothèques partagées d'Unix. La visibilité d'un module de code est globale, c'est–à–dire que le module est disponible pour toutes les applications qui possèdent une capacité pour y accéder.
La résolution des références à un module est statique, mais le chargement du module est fait dynamiquement selon les besoins de l'application.

## 7.2 Gestion des capacités dans Arias

Arias est conçu pour une grappe des machines homogènes. Nous ne sommes donc pas concernés par les motivations principales pour l'utilisation des capacités chiffrées : l'hétérogénéité des machines, les réseaux à grande échelle et les systèmes ouverts.
Nous espérons pouvoir offrir un service plus adaptable et plus efficace en gérant les capacités par le noyau sans chiffrement, c'est–à–dire en utilisant des capacités confinées.
Les capacités confinées sont entièrement gérées par le système qui connaît toutes les capacités permettant d'accéder à un segment et qui peut mettre en œuvre des mécanismes ingénieux ; par exemple le système peut gérer des compteurs de références pour savoir quand un segment n'est plus accessible (il n'existe plus de capacités) et peut être soumis au ramasse miettes.
La gestion des capacités confinées pose nombre de problèmes, notamment la création, le stockage, la délégation et la révocation des capacités. Ces problèmes sont traités dans la suite du chapitre.

### 7.2.1 Une première étude

La gestion de capacités confinées était le point central de la première thèse sur la protection dans Arias [Saunier96]. A l'époque, la première version d'Arias était en cours de réalisation et, par exemple, la gestion du code nécessaire à la réalisation du changement de domaine n'était pas encore réalisée. En plus la réalisation de Saunier se base sur l'utilisation des modules STREAMS abandonnée dans le deuxième prototype d'Arias (Arias v. 2[1]). Pour être

[1]La version 1 d'Arias a été la plate-forme de développement pour la plupart des mécanismes décrit dans cette thèse, mais la conception de la deuxième version d'Arias était déjà en cours dès le départ.

compatible avec la v. 2, nous ne pouvons pas profiter de cette réalisation, mais nous allons tirer partie des analyses faite par Saunier sur le stockage et la recherche des capacités. Les résultats de la première thèse sont décrit dans la suite.

### Stockage des capacités

Dans la réalisation de Saunier les capacités étaient stockées dans les structures du noyau et les modules de STREAMS étaient utilisé pour gérer la répartition et la cohérence des C–listes entre sites. La réalisation était entrelacée avec celle, des autres modules avec le risque de provoquer des avalanches de fautes si une modification etait introduite dans le module de cohérence.
Pour nous rendre indépendants de la version d'Arias, nous proposons de stocker des capacités dans les segments d'Arias même. Cela nous permet de manipuler les capacités selon l'API d'Arias (la probabilité de changement dans l'API est très faible). Un segment de capacités ne devra jamais être couplé dans les applications.
Le stockage des capacités dans les segments d'Arias est décrit plus en détail en 7.2.2.

### Recherche des capacités

Saunier propose de stocker les capacités dans des arbres AVL[2] pour pouvoir rapidement rechercher les capacités. Il gérait tous les types de capacités dans le même arbre. Nous adhérons à son analyse et gérons également les capacités dans des arbres AVL.
Saunier gérait les structures dans le noyau avec des pointeurs de 32bits mais nous allons gérer les capacités dans la MVRU où les pointeurs sont de 64bits. Pour éviter un gaspillage de mémoire, nous allons changer la réalisation pour utiliser l'adresse de base et un déplacement. Nous ne pouvons donc pas reprendre la réalisation de Saunier.
La réalisation du changement de domaine décrite dans ce chapitre nécessite une recherche de la capacité de changement de domaine à chaque changement. Nous avons décidé de partitionner les capacités selon leur type, c'est–à–dire qu'une C–liste contient trois sous–arbres : un pour les capacités d'accès, un pour les capacité de changement de domaine et un pour les capacités de redirection (cf. 7.2.4).

### Résolution des capacités dupliquées

L'échange des capacités pose un problème si un domaine de protection reçoit une capacité sur un segment sur quel il possède déjà une capacité. La solution proposée par Saunier est de rejeter la capacité avec les droits les plus restrictifs. Si le domaine possède une capacité de lecture et il reçoit une capacité d'écriture la capacité de lecture va être remplacée par celle d'écriture. Dans le cas inverse, la capacité de lecture va être rejetée.
Dans le cas d'une capacité de changement de domaine, plusieurs capacités peuvent exister qui permettent de passer dans des domaines différents. Le résultat dans ce cas est non–défini selon Saunier, mais il propose d'utiliser la première capacité trouvée dans le domaine. Nous avons introduit le changement dynamique de domaine et un mécanisme pour choisir la capacité de domaine à utiliser.

[2]La référence originale des arbres AVL est publiée en russe en Union Soviétique [AVL62], notre réalisation se base sur la description par Knuth [Knuth].

**Récapitulation**

Pour rendre les capacités inaccessibles aux applications, Saunier proposait de les stocker et de les gérer dans le noyau du système. Nous proposons de les stocker dans les segments d'Arias même et de les gérer dans le noyau. Le système doit s'assurer que les capacités d'accès aux segments de capacités ne sont jamais diffusées aux applications.
Nous adhérons à l'analyse qui a conduit Saunier à stocker les capacités dans un arbre AVL. Nous ne pouvons pas reprendre la réalisation de Saunier et nous avons décidé de partitionner les C–listes dans trois sous–arbres selon le type de la capacité
Nous avons repris le mécanisme de resolution des capacités dupliquées proposé par Saunier avec les modifications liées au changement dynamique de domaine décrites auparavant.

### 7.2.2 Stockage des capacités

Une service de stockage des capacités doit prendre en compte les éléments suivants :

- Persistance des capacités
- Partage des arbres de capacités entre domaines et machines
- Séparation entre l'espace d'adressage des applications et l'espace réservé aux capacités

Arias fournit des objets (segments) répartis, partagés et persistants. Il nous semble donc logique de gérer les capacités dans la MVRU elle–même. Ceci règle les problèmes de persistance et de partage des capacités. Il faut seulement trouver une solution pour la séparation entre segments utilisés par les applications (pour les données et le code) et segments utilisés par le système pour stocker les capacités. Ce problème est traité dans ce qui suit.

**Segments de capacités**

La séparation entre segments de données et segments de capacités peut être réalisée de deux façon différentes : par typage des segments ou par restriction de délégation des capacités.
Le typage associe un type à chaque segment, par exemple un segment de données partagées, un segment de code ou un segment de capacités. À la demande de couplage du segment, le système vérifie le type et la demande selon le type. En reprenant les exemples ci-dessus, le mode de couplage d'un segment de données dépend des capacités disponibles dans le domaine, le couplage d'un segment de code déclenche la liaison dynamique des librairies Unix utilisées par le code et un segment de capacités peut uniquement être couplé par l'administrateur du système (ceci est nécessaire pour réaliser des outils d'administration des capacités).
La restriction de délégation des capacités a déjà été mentionnée ci–dessus. La création d'un segment de capacités est une opération protégée qui s'effectue dans un domaine de protection particulier (le super–domaine). Toutes les créations de segments de capacités sont faites par un changement de domaine vers ce domaine. Ce domaine récupère la capacité d'accès au segment et retourne l'adresse du segment à l'appelant avec une capacité pour manipuler la C–liste stockée dans le segment. L'application peut utiliser cette C–liste pour créer un domaine de protection ou manipuler des capacités (si le système réalise une interface de programmation pour les capacités). Le système ne donne jamais de capacité d'accès sur les segments qui stockent les capacités et il n'accepte jamais de manipuler une C–liste qui ne se trouve pas dans le super domaine.

La deuxième solution montre que la gestion des segments de capacités peut être réalisée à l'intérieur du modèle (ceci est une vertu du modèle). Elle est la plus attrayante sur le plan esthétique et théorique, mais la première est plus simple à réaliser. Nous avons donc choisi, dans un premier temps, de réaliser le typage des segments.

**Remarque !** La réalisation du typage des segments n'exclut pas la restriction de délégation des capacités. Il suffit encapsuler l'appel de création de segment dans un module du code Arias (qui est uniquement accessible par le super domaine) et forcer les applications à utiliser ce module, c'est–à–dire l'appel système de création de segment doit vérifier que l'appel est issu du super domaine.

### 7.2.3 C–listes

Les capacités d'une même C–liste sont stockées dans un même segment ; en fait l'association directe entre une C–liste et un segment d'Arias nous permet de nommer la C–liste par l'adresse de base du segment.
Chaque segment qui contient une C–liste débute par une en–tête contenant quelques informations administratives (par exemple l'UID et le GID Unix à utiliser si la C–liste sert de base à un domaine de protection) et la racine de chaque arbre de capacités de types différents.

### 7.2.4 Format des capacités

Nous avons besoin de quatre types de capacités : capacités d'accès (pour coupler les segments localement), capacités de domaine (pour effectuer un changement de domaine), capacités de redirection (pour outrepasser une interface de protection dans une capacité de domaine reçue dynamiquement) et capacités de listes (pour pouvoir utiliser les capacité stockées dans une C–liste). Les formats de ces types de capacités sont décrits par la suite.

#### Capacités d'accès

Le format d'une capacité d'accès est illustré sur la figure 7.1.

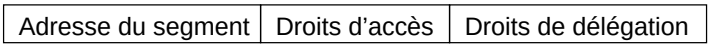



FIG. 7.1: Format d'une capacité d'accès

L'adresse du segment (8 octets) identifie le segment auquel la capacité autorise l'accès. Les droits d'accès sont spécifiés par un vecteur de bits de (2 octets) et les droits de déléguer la capacité (cf. 7.4) sont spécifiés par un vecteur de bits (2 octets). Une capacité d'accès occupe donc 12 octets.

#### Capacités de changement de domaine

Le format d'une capacité de changement de domaine est illustré sur la figure 7.2.

| Adresse de la procédure | Domaine appelé |
|---|---|
| Adresse du talon client | Adresse du talon serveur |
| Droits de délégation | |

FIG. 7.2: Format d'une capacité de changement de domaine

L'adresse de la procédure (8 octets) identifie le point d'entrée autorisé dans le domaine spécifié par le domaine appelé (8 octets)[3]. Le talon client (8 octets) identifie le segment de code à coupler à la place de la procédure, et le talon serveur (8 octets) spécifie l'adresse à laquelle le système doit faire un "upcall" dans le domaine appelé. Les droits de délégation sont les mêmes qu'avec une capacité d'accès.

### Capacités de redirection

Le format d'une capacité de redirection est illustré sur la figure 7.3.

| Adresse de la procédure | Adresse du talon client |
|---|---|

FIG. 7.3: Format d'une capacité de redirection

L'adresse de la procédure (8 octets) identifie la procédure qu'il faut surcharger et l'adresse du talon client (8 octets) identifie le segment de code qui doit être chargé à sa place. La taille d'une capacité de redirection est 16 octets.

### Listes de capacités

Le format d'une capacité[4] sur une C–liste est illustré sur la figure 7.4.

| Adresse de la C-liste | C-liste | Droits de manipulation |
|---|---|---|

FIG. 7.4: Format d'une capacité sur une C–liste

L'adresse de la C–liste (8 octets) spécifie l'adresse du segment qui contient la C-liste. Le champ C–liste (2 octet) identifie la capacité comme une capacité sur une C–liste. Outre les droits de délégation spécifiés pour une capacité d'accès, les droits de manipulation incluent les droits suivants : utilisation d'une capacité dans la C–liste, rajouter une capacité dans la C-liste, révoquer une capacité dans la C-liste et supprimer la C–liste.

### Discussion

Les formats de capacités spécifiés ici permettent de gérer les capacités dans des structures de taille fixe, soit 32 octets pour la plupart des capacités, soit 64 octets pour les capacités de changement de domaine. Ceci facilite l'allocation et la gestion des C–listes dans les segments d'Arias parce que les structures de données sont toujours alignées sur une page.

[3]Nous utilisons l'adresse du segment qui contient la C–liste de base du domaine comme identificateur unique du domaine

[4]Le format est le même que la capacité d'accès, mais les champs sont interprétés différemment.

Nous n'avons pas de notion de propriétaire d'une segment ou d'une capacité, mais la différence en taille entre les structures de gestion des capacités dans l'arbre AVL (32 octets) et les capacités d'accès et les capacités sur une C–listes (12 octet) est suffisamment grande pour qu'on arrive, si nécessaire, à inclure l'identificateur du propriétaire du domaine dans la capacité.

## 7.3 Création des capacités

Les capacités sont normalement créés lors de la création d'un segment et lorsqu'un droit d'accès est transféré entre deux domaines de protection. Lors de la création d'un segment, une capacité qui contient tous les droits sur le segment est rajoutée dans la C–liste racine de l'appelant. Le transfert de droits d'accès est réalisé par une copie de la capacité qui correspond aux droits d'accès dans le domaine appelant et l'installation de cette capacité dans le domaine appelé.

Ces mécanismes de création des capacités nécessitent un système bien configuré. Il y a donc un problème de démarrage du système (création et configuration des premiers domaines de protection). Pour résoudre ce problème, nous proposons un ensemble de primitives de création et d'administration des capacités uniquement disponible pour l'administrateur de système. Ces primitives doivent lui permettre de créer, initialiser ou manipuler des listes de capacités dans des domaines de protection différents. Ces primitives sont décrites dans l'annexe B.

L'administrateur de système obtient le droit de manipuler les C–listes parce qu'il crée les segments d'Arias qui contiennent les capacités (le stockage des capacités est traité plus en détail dans la section 7.2.2). Dans ce sens, tout le monde peut être administrateur de certaines applications ou utilisateurs (en créant les domaines de protection correspondants) et on n'a pas besoin de privilèges particuliers ou d'un utilisateur tout-puissant. Ceci permet une administration de système souple et décentralisée.

Même si l'administration se fait de manière décentralisée, le besoin d'intervenir d'une façon globale peut se présenter (par exemple pour retirer une capacité globalement). Nous proposons donc d'insérer toutes les listes de capacités dans une hiérarchie globale des capacités (les capacités ne sont pas hiérarchiques par nature, nous n'attendons donc pas que cette hiérarchie devienne très profonde). La hiérarchie globale nous permet de réaliser des mécanismes d'administration globale des capacités, par exemple la recherche des domaines qui contiennent une capacité sur un segment donné ou la révocation globale d'une capacité. Ces opérations sont normalement très difficiles dans les systèmes à capacités mais avec la hiérarchie globale, ils se réduisent à un simple parcours d'un arbre en mémoire virtuelle.

Le domaine de protection contenant la C–liste racine devient un super–domaine de même façon que `root` est un super–utilisateur dans un système Unix. Nous proposons donc d'associer ce super–domaine au super–utilisateur du système Unix qui constitue notre plate–forme de base. De toute façon le `root` d'Unix a plusieurs façons de contourner les mécanismes de protection d'Arias (il peut accéder directement aux segments permanents sur disque, il peut accéder à la mémoire directement par `/dev/kmem` et il peut assumer l'identité de chaque utilisateur du système et démarrer des applications comme cet utilisateur). Il n'y a donc pas de raisons pour limiter l'accès du `root` d'Unix aux C–listes d'Arias si cela est jugé commode.

## 7.4 Délégation des capacités

Une application qui possède les droits d'accès nécessaire peut copier des capacités entre C–listes dans son propre domaine ou transférer des capacités à un autre domaine de protection.

### 7.4.1 Le droit de délégation

Le droit de copier et de transférer une capacité est déterminé par un champ dans la capacité même (le droit de délégation). Les opérations de copie ou de transfert d'une capacité sont identiques dans le sens où une C–liste peut être partagée entre plusieurs domaines.
La restriction sur le droit de copier les capacités permet de réaliser un modèle de confiance plus ouvert, parce qu'un domaine peut déléguer une capacité à un autre domaine sans risquer la diffusion globale de la capacité.
Le droit de copier une capacité doit être vérifié par le moniteur de référence avant que la capacité soit copiée entre C–listes ou qu'elle soit passée en paramètre lors d'un appel de changement de domaine. La vérification est faite par un modul appelé module de politique dans le moniteur de référence[5]. Un module de politique est défini par l'administrateur et fixé au moment de l'installation du système Arias. Par la suite nous présentons la politique par défaut d'Arias.

### 7.4.2 La politique par défaut d'Arias

Une capacité peut être copiée telle quelle ou les droits inclus dans la capacité peuvent être restreints avant la copie. Le format d'une capacité d'accès (ou une capacité sur une C–liste) contient deux champs qui peuvent être limités avant la copie. Il y a donc cinq possibilités pour le droit de délégation : aucun droit de copier la capacité, le droit de copier la capacité avec une restriction sur les droits d'accès, le droit de copier la capacité avec une restriction sur les droits de délégation, le droit de copier la capacité avec une restriction sur les droits d'accès et les droits de délégation et le droit de copier la capacité sans restriction.

**Aucun droit de délégation.** La capacité peut être utilisée par le domaine, mais elle ne peut pas être copiée de la C–liste d'extension du domaine ou passée en paramètre à un autre domaine.

**Restriction sur les droits d'accès.** La restriction des droits d'accès implique qu'une capacité de lecture ou une capacité de changement de domaine ne peut pas être copiée, une capacité d'écriture est transformée en capacité de lecture et une capacité d'exécution est transformée en une capacité de changement de domaine vers le domaine courant. La nouvelle capacité possède les mêmes droits de délégation que l'originale.

**Restriction sur les droits de délégation.** La restriction sur les droits de délégation implique que la capacité puisse être copiée de la C–liste d'extension de domaine et installée dans la C–liste de base du domaine. La copie ne possède aucun droit de délégation.

[5]Une capacité de redirection permet à un domaine de protection de surcharger une procédure dans le domaine courant. Il n'y a pas de ressource protégée associée à la capacité ; le système permet donc toujours de copier ce type de capacité.

**Restriction sur les droits d'accès et les droits de délégation.** Les droits sont limités comme décrit ci–dessus.

**Aucune restriction.** Le domaine qui reçoit la capacité est libre de la copier entre ses C–listes et de la passer en paramètre aux autres domaines.

Les restrictions décrites ci–dessus sont vérifiées par le système, mais elles doivent également être spécifiées par les interfaces de protection. Cette spécification permet à l'application de limiter les droits des capacités passées en paramètre. Si l'interface spécifie un droit plus privilégié que celui que le domaine possède, un message d'erreur est retourné à l'application. À la création d'un segment, le système installe une capacité avec tous les droits dans la C–liste de base du domaine. Cette capacité peut servir à créer d'autres capacités et à les copier dans d'autres C–listes. Une capacité passée en paramètre est installée dans la C–liste temporaire de l'activité qui traite l'appel (cf. 7.8.4) et elle est révoquée au retour de l'appel. Si la capacité inclut la droit de la copier, le talon serveur peut demander au système de copier la capacité vers une de ces propres C–listes, par exemple la C–liste de base du domaine.

## 7.5 Révocation des capacités

Le problème de révocation des capacités est un problème classique et important dans les systèmes à capacités. Nous avons identifié deux méthodes pour retirer une capacité : la révocation automatique d'une capacité transférée lors du retour d'un appel de changement de domaine, et la révocation explicite.

### 7.5.1 Révocation automatique

Les capacités transférées lors d'un appel de changement de domaine, servent normalement pour que le serveur puisse accéder aux données nécessaires du déroulement de l'appel. Au moment du retour, le serveur n'a normalement plus besoin de ces droits d'accès et ils doivent être retirés (d'après le principe du moindre privilège).
Pour pouvoir retirer les capacités lors du retour de l'appel, le système doit garder une description des capacités reçues de chaque client (cf. 7.8).
Le système doit aussi assurer que une telle capacité ne pourra être copiée par le serveur que s'il en a reçu le droit.

### 7.5.2 Révocation explicite

Les listes de capacités (les domaines) sont organisées dans une arborescence, ce qui permet à un administrateur système (celui qui a créé la première liste) de parcourir toutes les listes de capacités et de révoquer les capacités selon ses souhaits.

## 7.6 Création d'un domaine de protection

La définition d'un domaine de protection est persistante, c'est–à–dire que le domaine existe indépendamment du processus qui le réalise effectivement. Un domaine de protection peut être associé à un processus de deux façons différentes : au démarrage d'une application Arias (le

domaine initial de l'application) et à la création d'un processus pour exécuter un appel avec changement de domaine (ce processus est appelé un serveur de domaine).

### 7.6.1 Création d'un domaine de protection initial

Les applications d'Arias sont lancées à partir du shell d'Unix, soit comme des programmes, soit par un lanceur générique des applications d'Arias. Au moment du lancement, l'application existe uniquement dans la partie Unix du système. L'appel à `AriasInit` (décrit en 5.4) va transformer le processus Unix en application Arias, c'est–à–dire qu'il va associer un domaine de protection au processus.
Cette association peut être faite interactivement par l'utilisateur (par exemple à l'aide d'un mot de passe), ou par le système, qui choisit un domaine initial à partir des paramètres du démarrage (nom de l'application et l'UID et le GID de l'utilisateur).
Pour pouvoir exécuter les applications en arrière–plan (sans intervention de l'utilisateur) nous avons choisi d'associer un domaine de protection à partir des paramètres du démarrage.
L'interprétation du GID dépend de la version du système Unix : dans System V l'utilisateur choisit son groupe courant avec la commande `newgroup(1)`, par contre dans BSD l'utilisateur s'exécute à tout moment avec tous les droits de tous les groupes auxquels il appartient. Nous avons choisi de nous reposer sur l'interprétation du GID la plus générale, c'est–à–dire celle de BSD. regarder le GID du processus.
La détermination du domaine initial à partir de la combinaison de la commande et de l'utilisateur supprime du modèle de protection une décision sur la sécurité. En plus, elle introduit plusieurs problèmes supplémentaires : par exemple comment réagir sur différents noms pour la même commande (différent liens vers le même binaire) ou qu'est-ce qui se passe si une commande est renommée ? Enfin les différentes applications vont normalement appeler différents module de code, on peut donc s'assurer que différentes applications sont exécutées dans différents domaines de protection en donnant différentes capacités de changement de domaine au domaine initial. C'est pourquoi nous avons choisi d'associer un domaine initial à chaque utilisateur. Le choix du domaine d'application à partir du domaine initial est montré sur la figure 7.5.

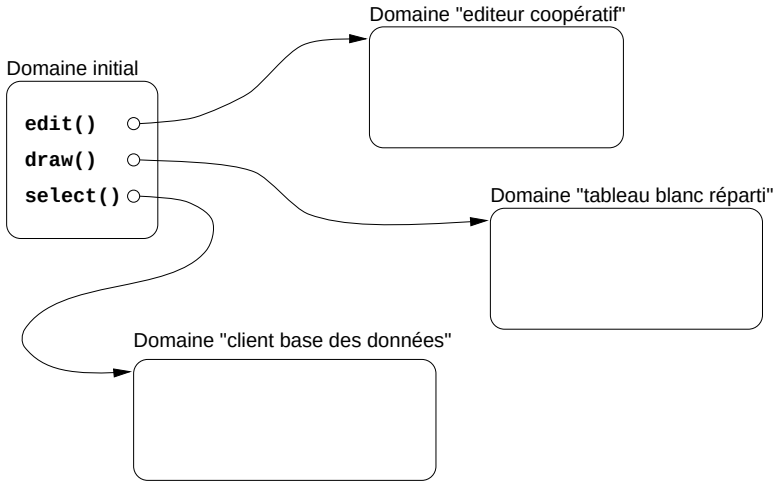



FIG. 7.5: Domaine initial et les domaines applicatifs

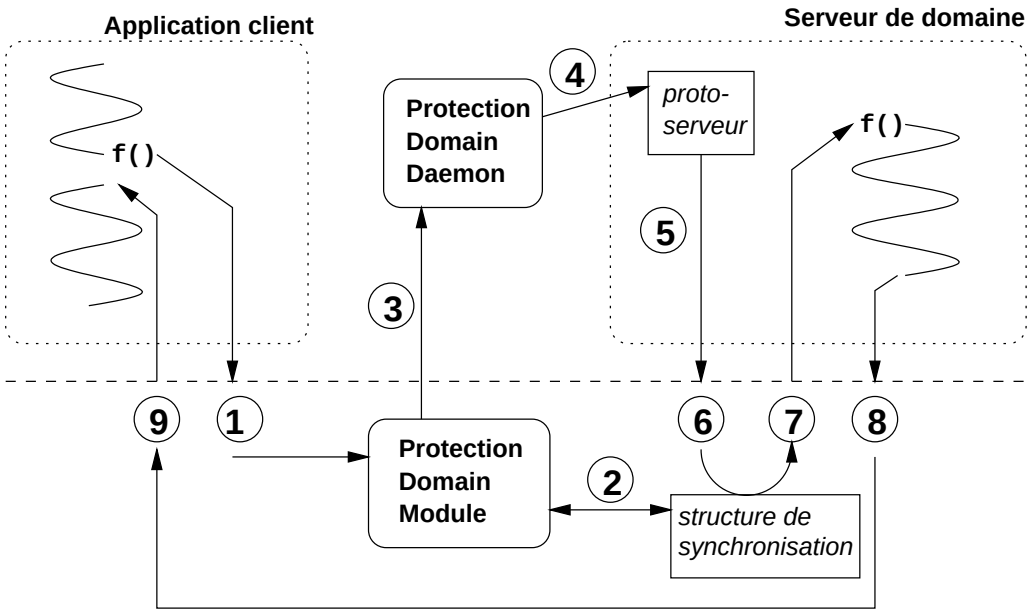



FIG. 7.6: Création d'un serveur de domaine

La figure montre un domaine initial et trois domaines applicatifs. La première fonction appelée par l'application décide vers quel domaine l'application va passer : la commande `edit()` va passer dans le domaine de l'editeur coopératif, `draw()` va passer dans le domaine du tableau blanc réparti et `select()` va passer dans le domaine de la base de données.

### 7.6.2 Création d'un serveur de domaine

Quand le client appelle une fonction qui ne peut pas être couplée dans le domaine de protection du client, le système va chercher une capacité de changement de domaine. Si une telle capacité est trouvée, l'appel va être transformé en appel de changement de domaine vers le domaine serveur identifié par la capacité. L'appel se déroule dans un processus créé spécialement pour traiter les appels de changement de domaine ; ce processus est appelé un serveur de domaine. Un serveur de domaine est crée dynamiquement au premier appel à ce domaine sur le site. Ce serveur sera utilisé pour les appels suivants. Pour éviter un goulot d'étranglement dans le serveur de domaine, chaque serveur est "multi–threadé", le nombre de "threads" étant un paramètre d'installation d'Arias. Le schéma de la création est montré sur la figure 7.6.
Le traitement d'un appel de changement de domaine cherche d'abord à identifier le processus qui réalise le serveur de domaine (1). S'il n'existe pas le système va créer un processus et l'initialiser comme serveur du domaine.
Le système crée les structures de synchronisation entre client et serveur, chaîne les paramètres de l'appel dans cette structure et interrompt l'appelant comme dans un appel de domaine normal (2).
Ensuite le système passe l'adresse de la structure en paramètre à un "upcall"[6] vers le serveur qui s'occupe de la création des serveurs de domaine. Notre version d'Unix (AIX) ne permet pas la création des processus depuis des extensions du noyau, nous sommes donc obligés

[6]Il n'y a pas un mécanisme d'"upcall" général dans Unix, donc tous nos "upcalls" sont en réalité des retours aux appels système précédents.

d'avoir un daemon Arias (le "Protection Domain Daemon" – PDD) avec comme seule tâche de créer les serveurs de domaine (3). Ce serveur a les droits de `root` pour pouvoir initialiser le processus avec le domaine de protection et les UID et GID correspondant à ce domaine.
Le serveur de création des domaines crée un processus qui exécute le programme proto–serveur (4). Le proto–serveur initialise l'utilisation d'Arias et crée plusieurs "threads" qui font un appel système pour attendre des "upcalls" (5). Un de ces appels va effectivement trouver le client suspendu dans sa structure de synchronisation (6) et retourne tout de suite pour traiter l'appel (7). Après avoir traité l'appel le "thread" va faire un appel système pour retourner la réponse (et débloquer le client) et attendre d'autres appels de changement de domaine.

**Remarque !** L'"upcall" vers le serveur de création de domaine avec l'adresse d'une structure dans le noyau, qui va plus tard être interprétée comme l'adresse d'une structure de synchronisation risque d'être détournée si quelqu'un arrive à modifier cette adresse. La sécurité de ce schéma depend donc de l'invulnérabilité du serveur de création des domaines et du binaire du proto–serveur – c'est–à–dire de l'invulnérabilité de l'utilisateur *root* sur les machines dans la grappe.

## 7.7 Couplage d'un segment

Le couplage d'un segment Arias se fait automatiquement lors de la première référence à une adresse à l'intérieur du segment. La référence provoque une faute d'adressage ("segmentation fault") dans le système hôte, qui est traitée par le paginateur Arias. Le paginateur Arias prend en charge la faute d'adressage et met à jour les structures nécessaires dans le noyau d'AIX pour que les fautes de pages suivantes puissent être prises en charge par le paginateur normal du système.
Avant de coupler le segment dans l'espace d'adressage du domaine (le processus qui le réalise) le système doit consulter le moniteur de références d'Arias pour savoir si le couplage est autorisé, c'est–à–dire si le domaine contient une capacité pour le segment. La vérification des capacités par le paginateur est montrée sur la figure 7.7.

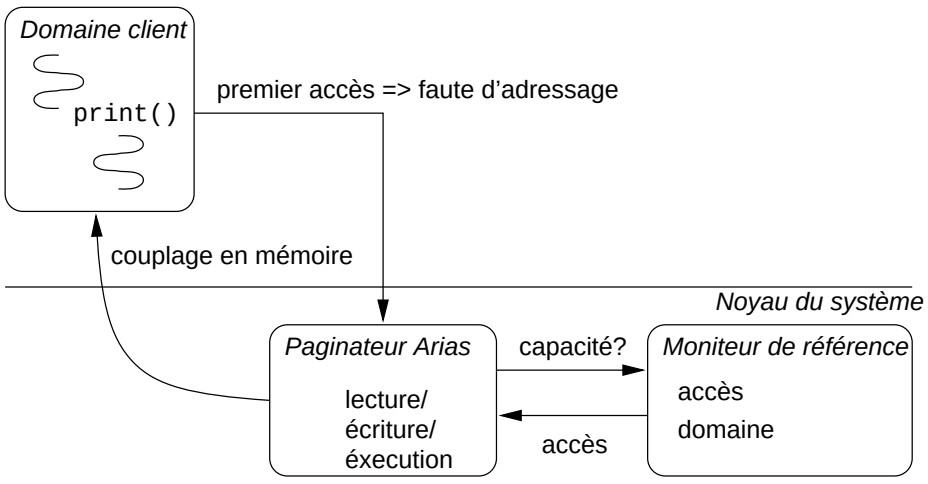



FIG. 7.7: Vérification des capacités

Quand le paginateur demande l'autorisation de coupler un segment au moniteur de références, le moniteur de références va chercher une capacité pour le segment dans la C–liste associée au domaine de protection. S'il trouve une capacité d'accès, l'autorisation est donnée et le segment est couplé dans l'espace d'adressage du domaine qui a provoqué l'exception selon le mode d'accès spécifié par la capacité (lecture, écriture ou exécution). S'il trouve une capacité de changement de domaine, un appel de changement de domaine va permettre l'exécution du code dans le domaine de protection spécifié par la capacité. Le changement de domaine est décrit en plus de détail dans la section suivante.

## 7.8 Changement de domaine

Il est important de préciser dans quelles conditions un changement de domaine peut intervenir. Toute vérification des droits d'accès est faite au moment du chargement d'un segment Arias dans un domaine de protection. L'appel de changement de domaine prend la forme d'un appel de procédure, c'est–à–dire que la vérification est faite lors du traitement de défaut du segment qui contient la procédure.

**Remarque !** Le changement de domaine lors de l'accès à un segment de données pose les problèmes suivants. Si on prend le schéma d'appel de méthode de C++, chaque objet contient un pointeur vers une table de méthodes (VTBL). On doit donc avoir accès à l'objet appelé pour calculer l'adresse de la méthode appelée et cela avant d'exécuter l'appel. Lors d'un défaut d'objet, on doit être en mesure de retrouver les caractéristiques complètes de l'appel, à savoir l'indice ou l'adresse de la méthode appelée, les paramètres, etc . . . pour recréer le contexte de l'appel dans le domaine destination. Ceci paraît difficile alors que les paramètres de l'appel n'ont peut être pas encore été empilés. Il faut donc que le système et l'environnement du langage coopèrent pour que le système puisse retrouver ces informations lors d'une faute. Ceci paraît difficile si on ne veut pas pénaliser tous les appels de méthodes et surtout si on veut pas limiter l'utilisation du système à un environnement d'exécution spécifique.

Remarquons également qu'il n'est pas évident que le changement de domaine sur une faute d'accès à un segment de données soit nécessaire pour réaliser des mécanismes de protection sur les objets. Il est peut être souhaitable de réaliser la protection à un grain plus gros au niveau de composant logiciels ou les serveurs comme dans CORBA [OMG98]. De toute façon, les langages à objet compilés sont souvent transformé en "C" avant d'être compilé en code exécutable. Dans ce contexte, il n'y a pas beaucoup de différence entre un appel de méthode standard `objet.méthode(paramètre_1, ...)` et un appel de fonction "C" `méthode(&objet, paramètre_1, ...)` outre la syntaxe.
Alors le modèle se base sur l'interception d'un appel de procedure et la transformation de cet appel en un appel de changement de domaine. Le mécanisme de changement de domaine est étudié dans la suite.

### 7.8.1 Interception de l'appel

Le premier appel à une procédure dans un module de code qui n'est pas encore couplé va provoquer une faute d'adressage. Ceci nous permet d'intervenir au niveau du traitement de la

faute.
On peut alors envisager deux schémas différents pour le déroulement de l'appel de changement de domaine : 1) soit effectuer l'appel de changement de domaine à partir du traitement de la faute, 2) soit coupler un talon à la place du vrai code. Le talon transforme l'appel local en un appel de changement de domaine de la même façon qu'un talon de RPC transforme un appel local en un appel distant.

**Traitement transparent par le système**

La transformation transparente d'un appel de fonction en appel de changement de domaine doit comprendre les étapes suivantes : vérification de la capacité de changement de domaine, identification du domaine serveur et passage des capacités selon l'interface de protection. Le domaine serveur et l'interface de protection sont identifiés par la capacité de domaine.
L'interface de protection est uniquement utilisée par Arias et doit donc être stockée par le système, soit directement dans la capacité (cela n'est pas souhaitable parce que les capacités deviennent de taille variable,) soit dans une structure référencée par la capacité.
La gestion de l'interface de protection par le système permet au système de garantir la conformité entre l'interface du client et l'interface du serveur avant de poursuivre l'appel. Cela est relativement simple pour les capacités d'accès mais beaucoup plus compliqué pour les capacités de domaine.
Si le client reçoit une capacité de domaine au retour du serveur, le système doit garantir que l'interface de protection encodée dans cette capacité correspond à la spécification de l'interface de protection du client. Alors la capacité doit contenir une spécification récursive des interfaces de protection qui est vérifiée par le système avant d'installer la capacité dans la C–liste du domaine client.
Cela a pour conséquence que le format des capacités devient entrelacé avec le code de passage des capacités, ce qui réduit la possibilité de changer un ou l'autre ultérieurement.
De plus, la gestion d'un arbre d'interfaces de protection et la vérification récursive de la conformité de ces interfaces rajoute à la complexité de la réalisation et augmente la probabilité d'introduire une faute qui compromet la sécurité du système. Ce problème est aggravé par l'environnement de programmation moins riche dans le noyau (seulement un sous–ensemble des appels systèmes sont disponibles depuis les extensions du noyau) et par la difficulté de la mise au point dans l'environnement du noyau.
Pour ces raisons nous sommes enclins à rejeter cette solution si le couplage d'un talon nous permet de bien réaliser le modèle.

**Couplage d'un talon**

La technique de remplacer une fonction par un talon est bien connue dans le contexte des appels RPC. Un talon de RPC normal s'occupe de la transformation des paramètres dans un format compréhensible pour le serveur et de la communication avec le serveur. Arias est conçu pour une grappe des machines homogènes, donc nous n'avons pas besoin de transformer les paramètres. Par contre le talon peut transformer l'appel pour réaliser l'interface de protection, c'est–à–dire rajouter une description des capacités à transférer avec l'appel et vérifier les capacités reçues.
Le talon client va être couplé dans l'espace d'adressage du client à une adresse fixe. Un client malveillant ne peut pas modifier le talon parce qu'il ne possède pas une capacité d'écriture sur

le segment (si le client possède une capacité d'écriture sur le segment à l'adresse du code, ce segment va être couplé au lieu du talon), mais il peut déchiffrer le format de l'interaction entre le talon et le système et appeler le système de la même façon. Le système n'a pas de mécanisme pour distinguer entre un appel depuis un talon proprement couplé et un cheval de Troie.
Le talon client ressemble beaucoup à la définition d'un proxy selon Shapiro [Shapiro86], il est le représentant local d'un objet distant. Shapiro propose que le proxy puisse être considéré comme une capacité, mais cela nécessite que le proxy et le serveur soient décomposables, c'est–à–dire que le proxy soit le seul moyen de communication entre client et serveur et que le client n'ait pas le moyen de changer le comportement du proxy. Dans Arias le talon client et le serveur sont deux entités séparées avec leurs propres identités (adresses virtuelles). Le client peut faire l'appel système pour le changement de domaine directement sans passer par le talon, donc le talon (le proxy) n'a pas les propriétés nécessaires pour servir comme capacité.
Cela a pour conséquence que tous les paramètres fournis par le client doivent être vérifiés avant d'être passés au serveur. Nous sommes dans un état de méfiance globale, le système se méfie des talons client et serveur, le talon serveur se méfie du talon client et le talon client se méfie du talon serveur. Les talons doivent faire confiance au système, sinon il n'y a pas de raisons d'utiliser le système.
La méfiance entre les deux talons est un élément très important dans notre réalisation du changement de domaine.

### 7.8.2 Changement de domaine : Vue d'ensemble

Le changement de domaine a déjà été traité en 6.3.3 et ci–dessus. Nous rappelons la vue d'ensemble du changement de domaine avant de décrire le traitement de l'appel du client, le coté serveur et le retour au client.

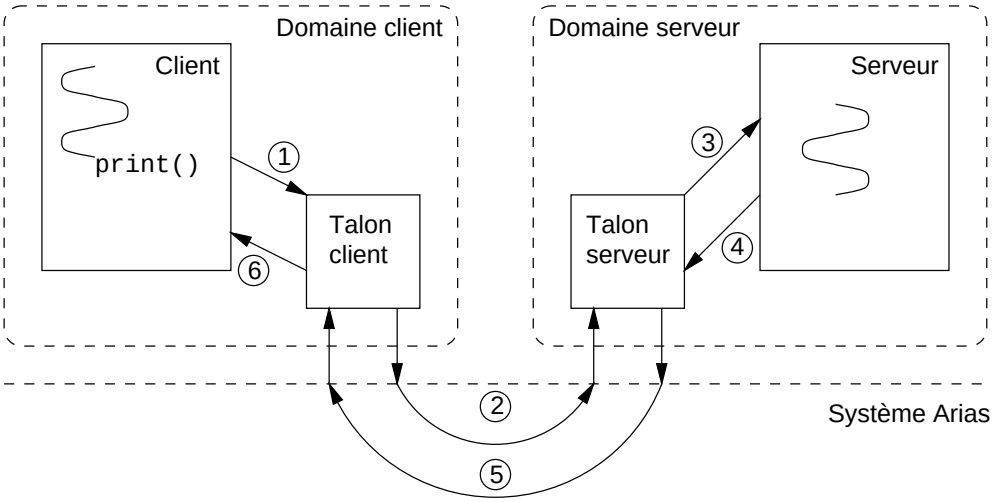



FIG. 7.8: Changement de domaine par couplage d'un talon

La figure 7.8 montre le schéma d'ensemble d'un changement de domaine. Le premier appel d'une fonction dans un module qui n'a pas encore été couplé va provoquer une faute d'adressage. Le système cherche d'abord à coupler le segment dans le domaine courant, mais

si le domaine possède une capacité de domaine au lieu d'une capacité d'accès, un talon va être couplé à la place du vrai code (1). L'adresse virtuelle permanente de ce talon est spécifiée dans la capacité de changement de domaine.
Le talon fait un appel système pour changer de domaine. Le système vérifie les paramètres de l'appel et transfère les capacités, puis un module de politique est appelé pour vérifier si le client possède le droit de transférer les capacités (2). Un "upcall" avec l'adresse de la fonction dans le talon du serveur (l'adresse du talon serveur est aussi stockée dans la capacité de changement de domaine) est effectué dans le domaine du serveur, ce qui peut provoquer une faute d'adressage dans le serveur de domaine qui va coupler le talon serveur (si le serveur n'a pas le droit de coupler le talon un erreur est retourné au client). Le talon appelle le code dans le serveur et attend le retour de l'appel (3). Au retour le talon fait un appel système pour retourner chez le client (4). Comme à l'appel, le système vérifie que le serveur peut transférer les capacités au client (5). Enfin, l'appel de changement de domaine se termine et le talon passe le résultat au client (6).
Les talons du client et du serveur peuvent réaliser les interfaces de protection du client et du serveur respectivement. Nous allons étudier les éléments de l'appel de changement de domaine plus en détail dans la suite.

### 7.8.3 Changement de domaine : l'appel de client

Le schéma de l'appel est montré sur la figure 7.9. Après avoir été couplé (1), le talon client sert d'abord à copier les paramètres de la pile locale dans un morceau de la mémoire tampon qui peut être transmis entre domaines. Puis le talon va appeler le système pour changer de domaine (2).

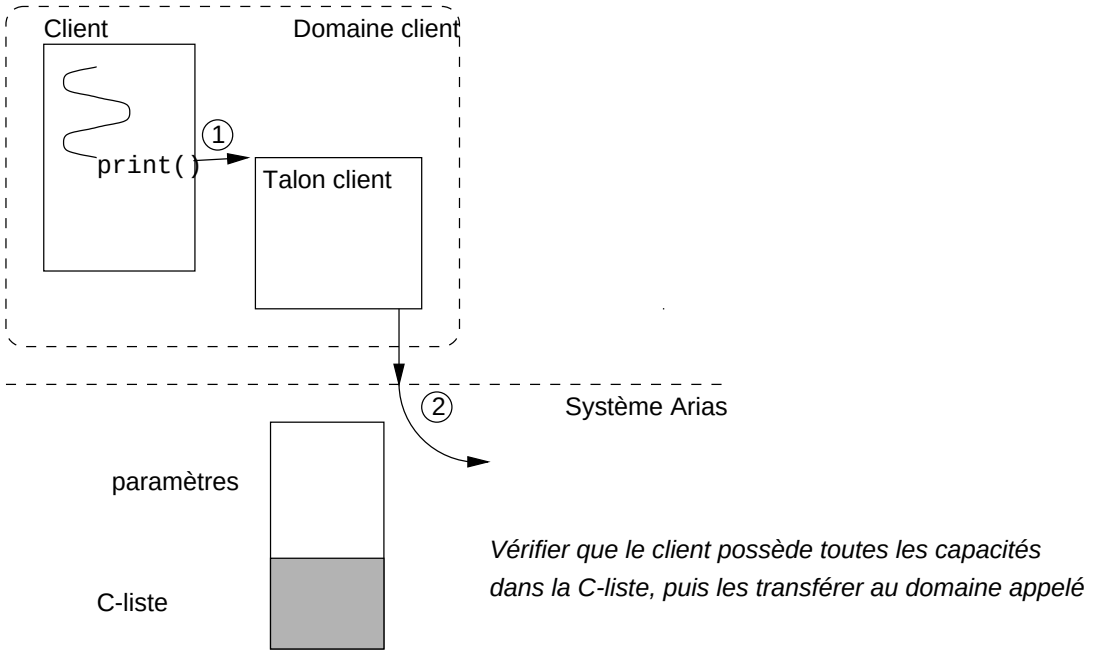



FIG. 7.9: Changement de domaine

Le talon passe l'adresse de la fonction à appeler, la mémoire tampon avec les paramètres et une liste de capacités à transférer au domaine serveur, en paramètre à l'appel de changement de

domaine. L'ensemble de la mémoire tampon des paramètres et la liste des capacités à transférer est appelé le bloc d'appel.

L'appel système ne peut pas distinguer entre un appel fait depuis un talon couplé comme résultat d'une recherche de capacité ou un appel fait depuis n'importe quel autre morceau de code. Le système doit donc vérifier l'existence d'une capacité de changement de domaine à chaque appel.

Après avoir vérifié la capacité de changement de domaine, le système vérifie que le domaine client possède toutes les capacités dans la liste spécifiée et les droits de les transférer.

Un module de vérification de la politique de protection (le Protection Policy Module — PPM) est appelé ensuite pour vérifier si le transfert des capacités est permis selon la politique de protection en vigueur. Ce module est réalisé par une extension séparée du noyau qui peut être remplacée pour réaliser différents politiques de protection[7] (par exemple les politiques mandataire ou discrétionnaire).

Si un serveur de domaine existe déjà, le bloc d'appel est chaîné dans la structure de synchronisation et l'appel système se bloque sur un sémaphore. Sinon la création de domaine dynamique est démarrée d'abord.

Une des activités créées par le serveur de domaine est débloqué pour effectuer l'"upcall" dans le domaine serveur.

### 7.8.4 Changement de domaine : Coté serveur

Le traitement du changement de domaine du coté du serveur est effectué par une autre activité (une créée par le serveur de domaine). Le schéma de déroulement de l'appel est montré sur la figure 7.10.

[7] Le seul module qui existe pour le moment réalise une politique discrétionnaire qui permet tous les transferts des capacités.

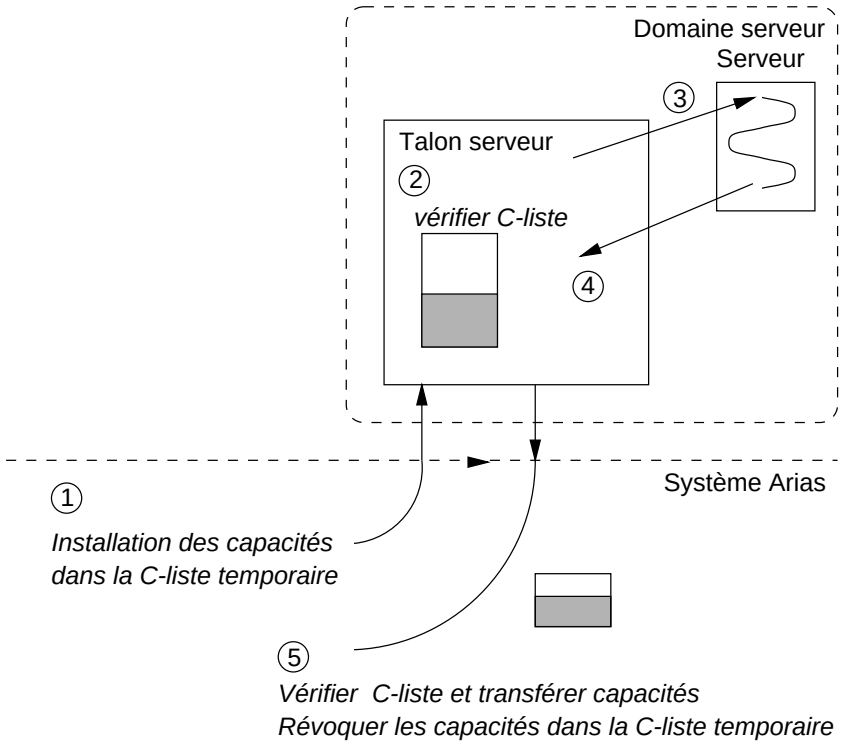



FIG. 7.10: Changement de domaine

L'activité reveillée par l'appel du client va d'abord installer les capacités transférées dans une liste des capacités temporaire du domaine. Cette liste temporaire est associé à l'activité et permet au système de restreindre l'utilisation des capacités à une activité particulière, qui traite le changement de domaine. Cette restriction ne s'étend pas à l'accès aux données parce qu'une fois couplé, le segment est accessible à toutes les activités dans le domaine, mais elle empêche une autre activité de passer la capacité à quelqu'un autre.
Le système calcule ensuite l'adresse à laquelle l'"upcall" est effectué. Ceci est nécessaire parce que le client ne connaît pas a priori l'interface de protection du serveur. Cette interface est déterminée par la capacité de domaine utilisée, c'est pourquoi l'adresse de la fonction à appeler dans le talon serveur ne peut pas être spécifiée directement, mais doit être calculée comme le déplacement de la vraie fonction dans le vrai segment du code. Notre gestion du code nous permet de garantir que le déplacement de la fonction dans le talon correspond à celui du vrai code.
L'activité revient dans l'espace d'adressage du domaine serveur et appelle la fonction à l'adresse calculée en passant tout le bloc d'appel en paramètre (1).
A ce moment le système garantit que les paramètres sont ceux reçus par le client et que toutes les capacités spécifiées par la liste dans le bloc d'appel ont été transférées au domaine du serveur. Alors le talon serveur peut vérifier que les intervalles des valeurs, le nombre et les types des paramètres correspondent a celles attendues[8]. Ensuite le talon vérifie que la liste de capacités est conforme à celle attendue — c'est–à–dire que le client fournit au moins les capacités requises par le serveur. Le client est libre de fournir plus de capacités que nécessaire

[8]La réalisation courante vérifie uniquement le nombre des paramètres, mais nous envisageons en outre de pouvoir vérifier l'intervalle de valeur des paramètre pour protéger un serveur contre un débordement de la mémoire tampon.

(2).

Si les paramètres et les capacités sont conformes à celles attendues, le talon appelle la fonction dans le serveur (3), sinon un message d'erreur est retourné au client.

Au retour de l'appel du serveur, le talon enregistre le code de retour, dépile les paramètres de retour et copie l'ensemble dans un morceau de la mémoire tampon. Ensuite le talon rajoute une liste de capacités, qui ont été transférées par le client lors de l'appel, à installer dans le domaine du serveur. Le talon pourait installer ces capacités avant d'appeler le serveur, mais cela n'est pas nécessaire (pendant l'appel ces capacités ont été disponible dans la liste temporaire) et nous préferons faire un "piggyback" sur l'installation au retour pour éviter un appel de système. Enfin le talon rajoute une liste des capacités à transferer avec les paramètres au client et appelle le système pour retourner chez le client (4).

L'appel système de retour vérifie d'abord que le serveur possède le droit d'installer les capacités transférées puis il copie les capacités dans la C–liste spécifiée par le talon serveur. Ensuite les capacités transférées pour effectuer l'appel sont révoquées (la liste des capacités temporaires est mise à zero).

**Remarque !** Les capacités ne peuvent pas être installées automatiquement parce que cela permettrait au client d'installer n'importe quoi dans le domaine du serveur. Par exemple le client pourrait installer une capacité de changement de domaine pour une fonction utilisée fréquemment, qui détourne l'appel vers un domaine sous le contrôle du client. Alors l'installation active des capacités (après avoir été inspectées par le talon serveur) est un élément nécessaire pour la suspicion mutuelle. La vulnérabilité du serveur est minimisée pendant l'appel parce que la vérification des capacités transférées est faite avant l'appel. Si le talon serveur détecte un problème un message d'erreur est rétourné au client.

Ensuite le système vérifie que le domaine du serveur possède les capacités spécifiées dans la liste des capacité à transférer au client et qu'il possède les droits de les copier. Puis le PPM est appelé pour vérifier que le transfert des capacités est en accord avec la politique de protection. Le bloc d'appel est déchaîné de la structure de synchronisation, le bloc d'appel du retour est chaîné dans la structure de synchronisation du client, l'activité du client est reveillé et l'activité du serveur se bloque sur un sémaphore dans la structure de synchronisation du serveur (5).

### 7.8.5 Changement de domaine : retour au client

La figure 7.11 montre le traitement au retour du serveur.

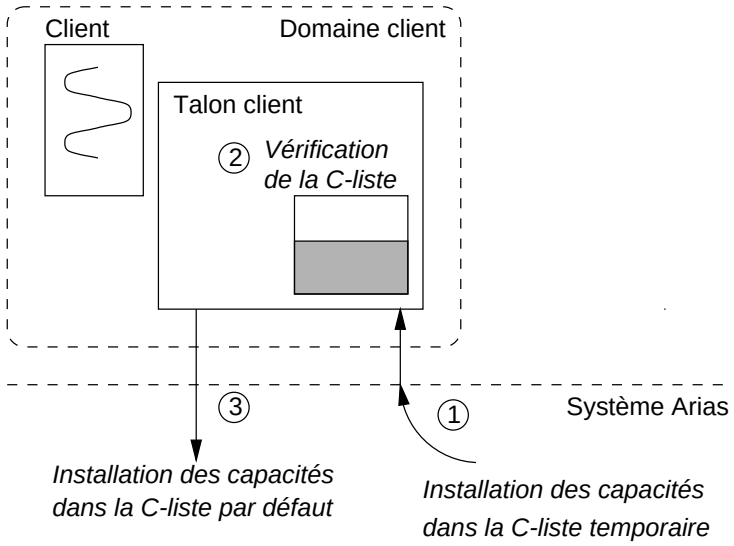



FIG. 7.11: Changement de domaine

Une fois réveillé, le flot de contrôle du client installe les capacités transférées au retour du serveur dans la liste des capacités temporaires du client. Puis les paramètres de retour sont copiés dans un tampon à transferer au talon client et l'appel de changement de domaine revient dans le talon client (1). Le talon client fait la même vérification des paramètres et les capacités que le serveur faissait à l'appel (2). Si tout est en règle le talon client appelle le système pour installer les capacités transférées par le serveur localement. Même s'il n'y a pas de capacités à installer, l'appel système est nécessaire pour vider la liste des capacités temporaires (3). Enfin les paramètres sont empilés et l'appel se termine comme s'il était un appel local.

### 7.8.6 Surcharge des interfaces de protection

Quand un domaine reçoit une capacité de domaine, cette capacité contient déjà une interface de protection du client (l'adresse du talon client). Si le client a confiance dans le serveur la capacité peut être utilisée directement, sinon il faut un moyen pour le client d'exprimer son interface de protection, c'est–à–dire de surcharger l'interface de protection encodée dans la capacité. Nous envisagerons deux méthodes différentes pour faire cela : modification de l'adresse du talon dans la capacité ou introduction d'une capacité de redirection à utiliser avec la capacité de domaine

#### Modification de la capacité

La modification de la capacité elle–même est la solution la plus simple. L'appel qui installe la capacité dans une C–liste locale va simplement remplacer l'adresse du talon client par une adresse fournie par le client. Si le client transmet la capacité vers un autre domaine de protection la capacité va être transférée avec l'interface de protection du client et pas avec l'interface par defaut du serveur inclut dans la capacité originale. Le domaine qui reçoit la capacité doit se demander s'il a confiance dans le client qui le transmet, sinon il installe sa propre interface de protection.
Dans une longue chaîne de délégations la question n'est plus de savoir si le client a confiance dans le domaine serveur, mais plutôt si un domaine a confiance dans tous les domaines qui le

précèdent dans la chaîne. En plus le domaine ne connaît pas la chaîne de délégation ; les relations de confiance deviennent donc très compliquées, et il vaut mieux installer sa propre interface de protection.
Si la capacité de changement de domaine va être stockée dans une C–liste partagée, tous les domaines vont utiliser la même interface de protection. Ceci implique aussi que la dernière interface installé va être celle utilisée par le client.

**Installation d'une capacité de redirection**

Au lieu de modifier la capacité on peut également envisager d'installer un nouveau type de capacité appelée capacité de redirection. La capacité redirige le couplage d'un segment vers un autre segment à coupler à l'adresse spécifiée par la capacité.
Quand le système cherche à coupler un talon client, il va d'abord chercher une capacité de redirection. Si une telle capacité existe, le talon specifié par cette capacité va être couplé au lieu du talon specifié par la capacité de domaine.
Les capacités de redirection doivent toujours être installées dans la C–liste de base d'un domaine. Ceci garantit que l'interface de protection souhaitée par le domaine va toujours être en vigueur.

### 7.8.7 Changement dynamique de domaine

Le changement dynamique de domaine utilise un appel système spécialisé. Outre les paramètres et la description des capacités à transférer, l'appel de changement de domaine dynamique prend un paramètre en plus — l'adresse d'un segment qui contient les paramètres auxiliaires (nous appelons cette adresse la référence décisive). La description des capacités à transférer spécifie quels droits d'accès sont associés a cette référence (si le domaine appelé doit avoir le droit de lire ou d'écrire les données dans le segment[9]).
Le système fait toutes les vérifications normales des paramètres, puis il construit une liste de capacités de changement de domaine associée à l'adresse de la fonction et disponibles dans le domaine appelant. Les capacités dans cette liste identifient les domaines de serveur potentiels. Le système parcourent ces domaines un par un, pour trouver un domaine qui possède les droits nécessaires et transfère l'appel vers le premier domaine trouvé. Si le système ne trouve pas un domaine à appeler, un message d'erreur est retourné au client.

### 7.8.8 Discussion

La suspicion mutuelle entre les deux domaines est réalisée de bout en bout. Le système n'offre aucune garantie, mais sert uniquement comme un médiateur crédible. Cela nous permet de respecter le principe de minimalité dans notre réalisation du changement de domaine.
L'appel de changement dynamique de domaine étend la notion du changement de domaine vers un modèle de programmation à objet. Un appel de méthode se réalise facilement par l'appel de changement de domaine comme mentionné au début de 7.8.

[9] On peut également imaginer d'associer le droit d'appeler une fonction dans un domaine spécifié à l'adresse d'une fonction, mais cela rend la réalisation beaucoup plus compliquée et nous n'avons pas identifié un cas ou cela serait nécessaire. Nous avons donc décidé de ne pas permettre cette fonctionnalité.

La granularité d'un appel est un segment d'Arias. La taille minimale d'un segment Arias est 4Ko, ce qui correspond à une page de la mémoire physique de la machine. Un système à objet réalisé sur Arias devrait donc réaliser un modèle à objet de gros grain ou gérer plusieurs objets dans le même segment (ceci est normalement le cas dans un serveur CORBA).
Le couplage d'un talon à la place du vrai code crée des problèmes si la même fonction doit être appelée parfois dans le domaine du client et parfois dans un autre domaine.
Si le premier appel couple le code localement, tous les appels suivants vont se dérouler dans le domaine du client. On décide donc de coupler le talon tous les cas et d'inclure dans le talon le code de la procédure que l'on vient appeler. Il faut s'assurer que le domaine ne possède pas une capacité d'exécution mais uniquement une capacité de changement de domaine dynamique.

**Remarque !** Il n'existe pas de solution simple, au problème des appels parfois local parfois distant, avec le traitement transparent par le système. Le système n'arrive pas à distinguer entre les appels locaux et les changements de domaine. Dans ce cas tous les appels des fonctions dans un autre module de code doivent être vérifiés par le système.

Dans le prototype courant, le problème peut être résolu en rajoutant le vrai code à la fin du talon[10]. L'appel de changement de domaine dynamique doit d'abord vérifier si la référence décisive peut être atteinte depuis le domaine du client et retourner un message si c'est le cas, puis le talon peut appeler le vrai code à la fin du module — ce code est stocké dans le même segment et ne provoque pas une faute d'adressage. Si les paramètres ne sont pas disponibles dans le domaine du client, le changement de domaine se déroule normalement. Cette vérification va rajouter au temps d'exécution pour tous les appels de changement de domaine dynamiques, mais elle est nécessaire pour le support d'un changement de domaine général.

## 7.9 Interfaces de protection

La spécification des interfaces de protection est compilée en deux talons différents, un pour le client et l'autre pour le serveur, par un générateur des talons modifié (nous utilisons une version modifiée de `rpcgen(1)` fourni par Sun Microsystems).
Chaque interface compilée (le talon) est stockée dans un segment différent d'Arias. L'adresse de base de ce segment peut être utilisée comme identification de l'interface dans les capacités de changement de domaine. L'adresse du talon indique au système quel segment coupler à la place du segment qui contient du vrai code dans le client et quel segment coupler pour effectuer l'"upcall" dans le serveur.
Différentes capacités de domaine peuvent spécifier différents talons qui réalisent différents traitements du coté serveur. Ceci ressemble aux capacités actives, mais le code dans la capacité reste toujours du coté serveur.
Pour l'instant nous n'avons pas des idées très précise sur la fonctionnalité qui peut être introduite dans les talons du serveur, mais notre réalisation nous donne la souplesse nécessaire pour l'introduire ultérieurement.

[10]Ces modifications n'ont pas été faites dans le prototype courant parce qu'il fallait changer la gestion de la génération des talons.

## 7.10 Récapitulation

Pour récapituler la réalisation du mécanisme de protection présenté dans ce chapitre, nous donnons son placement dans la taxonomie introduit en 4.3.3 :

1. que se passe-t-il à la création d'un objet ? Le système insère une capacité avec tous les droits dans la C–liste par défaut du domaine créateur.
2. comment peut-on manipuler les capacités ? Les capacités sont gérées dans des segments séparés qui ne peuvent pas être couplés par un processus. Toutes manipulations des capacités sont faites par des appels systèmes. L'API de ces appels système est décrit dans l'annexe B.
3. comment peut-on réaliser un changement temporaire des droits d'accès d'un sujet ? Arias fournit un mécanisme de changement de domaine qui permet d'appeler une procédure dans un autre domaine de protection.
4. comment peut-on déléguer une capacité ? Il existe deux mécanismes de délégation d'une capacité, elle peut être copiee dans une C–liste partagée ou elle peut être passée en paramètre dans un appel de changement de domaine. Il faut de toute façon l'intervention du système pour déléguer une capacité.
5. comment peut-on révoquer une capacité ? Arias fournit un service de révocation automatique des capacités passées en paramètre lors du retour de l'appel de changement de domaine. L'API de la gestion des capacités définit un appel de révocation d'une capacité dans une C–liste. Pour cela, il faut les droits adéquats sur la C–liste.
6. comment sont vérifiées les capacités ? Les capacités sont vérifiées par le système lors du traitement d'un défaut de segment.
7. que se passe-t-il quand un sujet essaie d'accéder à une ressource protégée ? Lors d'un défaut de page, le paginateur d'Arias vérifie les droits associés au segment.

La classification selon notre taxinomie du mécanisme de protection d'Arias est donc comparable à celui de Mach, c'est–à–dire la classification suivante :

[**classification : bbcbbab**]

# Chapitre 8

# Évaluation

Ce chapitre présente une évaluation du modèle proposé et sa réalisation dans la mémoire virtuelle répartie unique Arias, dans son état en juillet 1998.
L'intégartion du prototype n'est pas complet ; l'état courant permet de :

- vérifier les capacités lors du premier accès (le défaut de segment),
- coupler les talons du client et du serveur et d'effectuer un appel de changement de domaine,
- créer un serveur de domaine lors du premier appel au domaine,
- transférer les capacités passées en paramètre lors de l'appel de changement de domaine, et
- révoquer les capacités transférées lors du retour de l'appel.

L'état du prototype permet donc d'évaluer tous les éléments du modèles des capacités cachées, mais le manque d'intégration avec Arias (par exemple le manque du service de permanence) empêche le développement d'applications complètes.
L'évaluation qui suit rappelle d'abord les objectifs de cette thèse en 8.1. Ensuite nous traitons chacun des ces objectifs dans son tour : la généricité du modèle est traitée en 8.2, la facilité de programmation en 8.3, l'évolutivité du modèle en 8.4, la réutilisabilité des composants logiciels en 8.5 et la possibilité d'intégration de logiciels conçus ailleurs en 8.6. Nous allons étudier quelques aspects de la sécurité de la réalisation en 8.7 avant présenter une évaluation de performance du changement de domaine en 8.8.

## 8.1 Objectif de l'évaluation

L'objectif de l'évaluation est de déterminer si les objectifs du modèle de protection mentionnés en 1.3 sont atteints par sa réalisation dans Arias. Nous rappelons d'abord ces objectifs :

1. *Généricité du modèle.* La généricité a été un but principal dans la conception d'Arias. Pour démontrer la généricité du modèle, nous esquissons la réalisation des différents modèles de protection mentionnés en chapitre 3 sur le modèles des capacités cachées.
2. *Facilité de programmation.* Le modèle doit être facile à comprendre et à utiliser, sinon il ne sera pas utilisé ou les spécifications risquent d'être fausses. Nous montrons comment profiter des progrès récents en genie logicielle (notamment les architectures logicielles) et nous présentons deux scénarios d'application et leur réalisation possible sur Arias.

3. *Évolutivité.* La spécification de la protection d'une application doit pouvoir changer pendant la durée de vie de l'application, sans recompilation ni réédition des liens. Nous montrons comment faire évoluer les droits d'accès d'une application pendant l'exécution et comment faire évoluer la politique de protection d'une application.
4. *Réutilisabilité des composants logiciels.* Les mêmes composants logiciels peuvent être utilisés dans des contextes et des applications différents. Pour démontrer la réutilisabilité des composants logiciels, nous présentons une expérience de réutilisation d'un service de noms dans des contextes de protection différents.
5. *Possibilité d'intégration de logiciels conçus ailleurs.* Nous montrons comment les logiciels conçus ailleurs (bibliothèques partagées ou applications Unix) peuvent être encapsulés et protégées par le système Arias.

L'évaluation de ces différents objectifs sont traités dans les sections suivantes.

## 8.2 Généricité du modèle

Dans cette section, nous analysons les propriétés du schéma de capacité proposé pour savoir si nous pouvons arriver à réaliser les différents modèles de protection proposés dans le chapitre 3. Le but de cette section est de montrer que la plupart des modèles peuvent être réalisés sur le modèle de base proposé dans le cadre de cette thèse. La stratégie proposée pour arriver à ce but, est d'esquisser les grandes lignes de la réalisation d'une politique selon chaque modèle proposé dans le chapitre 3, en utilisant notre schéma de capacités cachées.

### 8.2.1 Réalisation d'une politique de contrôle d'accès mandataire

Nous avons choisi le modèle de Bell et LaPadula pour montrer la faisabilité de la réalisation d'un modèle de contrôle d'accès mandataire. Nous décrivons dans la suite les principes d'une réalisation de ce modèle.
Nous rappelons que le modèle de Bell et LaPadula se base sur deux principes fondamentaux : un sujet ne peut lire que des objets avec un niveau d'autorisation égal ou inférieur à son niveau, et il ne peut modifier que les objets qui ont un niveau d'autorisation égal ou supérieur au sien. À chaque accès à un objet, le niveau d'autorisation du sujet est comparé au niveau d'autorisation de l'objet et l'accès n'est permis que si les règles ci–dessus sont respectées.
Dans un système à base de capacités, le contrôle d'accès se fait uniquement par la possession d'une capacité. Il n'y a pas de vérification des niveaux respectifs du sujet et de l'objet, mais l'autorisation se base uniquement sur la présentation d'une capacité valable pour l'objet. Cela implique que la comparaison des niveaux d'autorisation doit se faire au moment où une capacité est rajoutée dans une liste de capacités (dans le domaine de protection) et non quand le sujet essaie d'accéder à l'objet.
Nous montrons sur la figure 8.1 comment le modèle mandataire peut être implanté sur notre modèle de protection à base de capacités. La notation utilisée ici est inspirée de celle de Snyder (cf. 4.3.1) et nous allons utiliser cette notation pour décrire les systèmes de capacités par la suite. Les flèches symbolisent des capacités, celles avec un cercle blanc symbolisent les capacités d'accès et celle avec un cercle noir symbolise une capacité de changement de domaine. Les capacités d'accès sont annotées avec le droit d'accès qu'elles peuvent accorder et la capacité de changement de domaine avec les droits d'accès qu'elle permet de transférer,

c'est-à-dire que l'interface de protection permet uniquement le transfert des capacités en lecture.

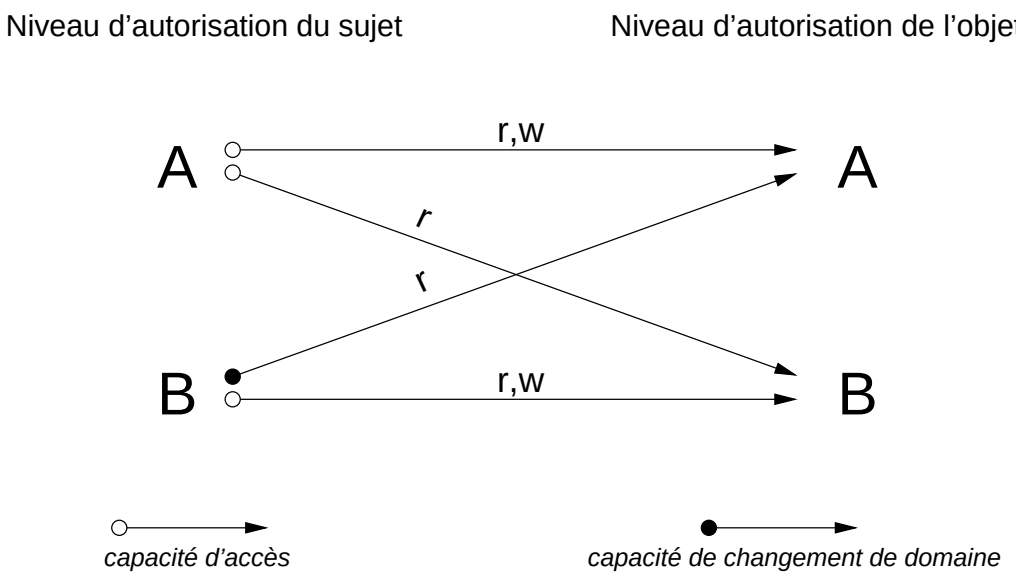



FIG. 8.1: Contrôle d'accès mandataire

La figure montre deux niveaux d'autorisation *A* et *B* où le niveau *A* est celui qui a le plus de droits. Les sujets sont représentés à gauche et les objets à droite. Un sujet habilité au niveau A doit pouvoir lire et écrire les objets au niveau A, mais seulement lire les objets au niveau B. Un sujet au niveau A a donc des capacités en lecture/écriture vers des objets au niveau A, mais que des capacités en lecture vers des objets au niveau B.
Un système de mémoire virtuelle permet de coupler un segment de mémoire en lecture ou en lecture/écriture ; il n'y a normalement pas de moyen de coupler un segment de mémoire uniquement en mode écriture. Cependant, ceci est nécessaire afin qu'un sujet au niveau B puisse écrire (donner) des objets au niveau A sans qu'il n'ait le droit de lire des objets au niveau A (modèle mandataire).
Pour implanter un modèle mandataire, nous proposons d'utiliser le changement de domaine pour réaliser des écritures vers un niveau supérieur sans donner de droit en lecture. Sur la figure 8.1, les sujets au niveau B peuvent avoir des capacités de changement de domaine vers des domaines du niveau A, en passant des capacités en lecture vers le niveau A pour donner les informations à écrire. L'opération dans le domaine appelé réalise l'écriture.
La configuration d'un domaine de protection à un certain niveau doit prendre cela en compte de la façon suivante :

1. Un domaine de protection doit uniquement posséder des capacités de lecture pour les segments avec un niveau d'autorisation inférieur à celui du domaine.
2. Un domaine de protection peut posséder des capacités en lecture ou en lecture/écriture pour les segments avec un niveau d'autorisation égal à celui du domaine.
3. Un domaine de protection ne peut modifier des segments avec un niveau d'autorisation supérieur à celui du domaine qu'à travers des capacités de changement de domaine. Ces changements de domaine ne prennent en paramètre que des capacités en lecture à l'aller.

Ces capacités de changement de domaine (et leurs interfaces protégées) sont définies par l'administrateur du système.

Pour montrer la possibilité de réaliser une politique de contrôle d'accès, nous supposons que le système part d'une configuration sûre (configurée par l'administrateur du système responsable de la protection). Nous allons alors montrer que toutes les manipulations qui changent l'état de l'arbre des capacités mènent à une configuration sûre. Dans la politique décrite ci–dessus, cela veut dire que les opérations de création de segment et changement de domaine respectent les règles de sûreté de la politique mandataire.

**Création d'un segment**

À la création d'un segment, une capacité qui donne tous les droits au segment (normalement lecture et écriture), est rajoutée à la liste de capacités du domaine créant le segment. Le domaine créant est à ce moment–ci, le seul domaine à posséder cette capacité et en conséquence le seul domaine à pouvoir modifier le segment. La règle de sûreté est alors respectée.

**Délégation des droits d'accès**

Le changement de domaine est le seul mécanisme qui permet de déléguer les droits d'accès entre domaines. Le mode d'interaction entre domaines est contrôlé par les interfaces de protection. Dans la politique de base, le changement de domaine se fait uniquement pour permettre aux domaines de modifier les données d'un niveau d'autorisation plus élevé que le domaine même ("write up"). Les interfaces interposées entre un domaine moins privilégié et un domaine supérieure impose que le passage de paramètres entrants ne contienne que des capacités en lecture.
Cela garantit que le domaine plus élevé n'arrive pas à communiquer un résultat ou à modifier les données dans un segment qui peut être lu par un domaine moins privilégié. La délégation des droits d'accès respecte alors les règles de sûreté.

**Discussion**

Les systèmes qui gèrent les capacités dans l'espace d'adressage du processus (par exemple Amoeba et Opal), ne peuvent pas servir de base d'une politique de contrôle d'accès mandataire, parce qu'il n'y a pas de mécanisme pour contrôler la délégation des droits d'accès. Le mécanisme des PDX de Mungi permet au programmeur d'application de créer une restriction du domaine de protection appelé avant l'appel effectif de ce domaine de protection. Ceci permet de réaliser l'isolation de l'appel dans le domaine appelé au lieu de l'extension de domaine. Cependant, il n'y a pas de mécanisme système permettant d'imposer une telle restriction. Il n'est donc pas possible de réaliser un modèle mandataire sur le modèle de Mungi. Enfin une politique de protection mandataire ne peut pas être réalisée sur le modèle à capacités classique, même si les capacités sont gérées par le système. En effet, il faut pour cela que les règles d'échange de capacités entre les domaines soient contrôlées par le système, ce qui est le cas avec nos interfaces protégées associées aux capacités (qui sont gérées par le système). Nous n'en apportons pas une preuve formelle ici mais cela constitue tout de même un résultat significatif.

### 8.2.2 Réalisation d'une politique de contrôle d'accès discrétionnaire

Nous avons choisi de montrer comment réaliser un modèle de contrôle d'accès discrétionnaire à la Unix parce qu'Unix est un système d'exploitation largement utilisé. Le modèle de protection d'Unix a été étudié dans la section 3.5.
L'authentification d'Unix se base uniquement sur l'UID d'utilisateur. Pour faciliter l'administration, les utilisateurs peuvent être regroupés en groupes. Les membres d'un groupe possèdent les droit définis pour ce groupe.
Nous rappelons que le modèle de ressources d'Unix est centré autour du système de fichiers. La plupart des ressources partagées ont une représentation sous forme de fichier. L'accès à un fichier est déterminé par un vecteur de bits de protection stocké parmi les meta-données du fichier (l'inode). Ce vecteur contient les droits d'accès (rwx pour lire, écrire et exécuter) pour le propriétaire du fichier, le groupe principal du fichier et l'ensemble des utilisateurs du système. Quand un utilisateur essaie d'ouvrir un fichier[1] le système vérifie le mode d'accès souhaité avec l'information stockée dans le vecteur de contrôle d'accès. Le système vérifie d'abord si l'utilisateur est le propriétaire du fichier, sinon il vérifie si l'utilisateur appartient au groupe principale du fichier et finalement il vérifie le mode d'accès avec l'entrée du vecteur pour l'ensemble des utilisateurs du système.
Un version de ce modèle de protection peut être appliquée aux segments gérés dans le système Arias. Ce modèle peut être implanté sous forme de C-listes partagées. Il faut une C-liste privée pour chaque utilisateur, qui correspond aux fichiers privés. En plus il faut une C–liste pour chaque groupe du système et une C–liste qui correspond à l'ensemble des utilisateurs du système. Un processus Unix s'exécutant pour le compte d'un utilisateur est initialisé avec un domaine de protection défini par une C–liste de capacités comprenant les capacités sur les trois C–Listes suivantes : la C–liste associée à l'utilisateur, la C–liste associée au groupe par défaut de cet utilisateur et la C–liste de l'ensemble des utilisateurs. Lorsqu'un segment est créé, le masque de création (`umask`) est utilisé afin de déterminer les capacité sur le segment qu'il faut insérer dans les trois C–liste décrite ci–dessus.

#### Vérification des droits d'accès

Il n'y a pas d'`open(2)` dans Arias, mais la vérification des droits d'accès se fait lors du premier accès au segment. L'`open(2)` d'Unix et le premier accès au segment d'Arias mettent à jour les structures du noyau qui permettent les accès ultérieurs par le sujet sans vérification.

#### Délégation des droits d'accès

Il existe deux mécanismes de délégation des droits d'accès dans Unix : les opérations `chown`, `chmod` et `chgrp` qui permettent de déléguer un droit particulier et le mécanisme de SUID/SGID qui permet de déléguer l'ensemble de droits d'un utilisateur pendant l'exécution d'un programme. Le droit de délégation est associé au propriétaire du fichier. Pour pouvoir restreindre le droit de manipuler une capacité dans une C–liste publique à un seul usager, il faut un équivalent de la notion de proprétaire. Le format d'une capacité doit donc être étendu

[1] L'`open(2)` d'Unix est un appel système qui sert en même temps à mettre à jour les structures internes du système et à vérifier les droits d'accès.

avec un identificateur du domaine propriétaire de la capacité[2]. Cela implique uniquement une modification au niveau de la réalisation du modèle.
Les opérations `chown`, `chmod` et `chgrp` peuvent être réalisées par des applications Arias qui ont le même syntaxe que leur correspondantes Unix.

> `chown` insère une capacité (ou une C–liste) dans la C–liste de base du nouveau propriétaire et la supprime dans la C–liste de l'ancien propriétaire. L'identificateur de propriétaire est également modifié.
>
> `chgrp` modifie les C–liste des groupes de la même façon que `chown` modifie les C–listes des usagers.
>
> `chmod` change les droits d'accès dans la C–liste du domaine de propriétaire, la C–liste du groupe et la C–liste globale, selon le vecteur de bits donné en paramètre.

Le mécanisme de SUID/SGID sur un fichier exécutable permet de déléguer l'ensemble des droits d'accès d'un utilisateur Unix. En prenant l'identité du propriétaire (resp. groupe) du fichier, un processus peut étendre ses droits d'accès, car il obtient en appelant cet exécutable les droits d'accès du propriétaire du fichier.
Pour réaliser l'extension des droits d'accès du processus courant, il suffit d'insérer dans le domaine de protection associé au processus une capacité de changement de domaine. Cette capacité de changement de domaine correspond à un point d'entrée du domaine (une procédure) dont une capacité (avec les droits d'exécution) se trouve dans le domaine appelé.

**Discussion**

On voit qu'ici, la fonction de base permettant de réaliser le modèle de protection discrétionnaire d'Unix est la gestion de listes de capacités partagées entre les utilisateur, auquel on ajoute le mécanisme de changement de domaine permettant de réaliser une extension dynamique des droits d'accès d'un processus.

### 8.2.3 Réalisation d'une politique de contrôle d'accès basé sur les rôles

Dans le modèle basé sur des rôles, un usager est autorisé à jouer différents rôles et nous supposons qu'une primitive du système permet à l'usager de changer de rôle. Ce rôle détermine les transactions que l'usager peut réaliser.
Le modèle de protection à base des rôles se réalise naturellement dans le modèle de protection d'Arias. Chaque domaine de protection correspond à un rôle ou une transaction et le fait d'assumer un rôle correspond à un changement de domaine.
L'ensemble des domaines de protection disponibles à un utilisateur peut être représenté sous forme d'un graphe (cf. la figure 8.2) où les nœuds représentent les domaines de protection et les arcs représentent les capacités de changement de domaine.

[2] Il est possible d'étendre le format de la capacité parce qu'il reste suffisamment d'octets dans la structure de stockage d'une capacité dans l'AVL.

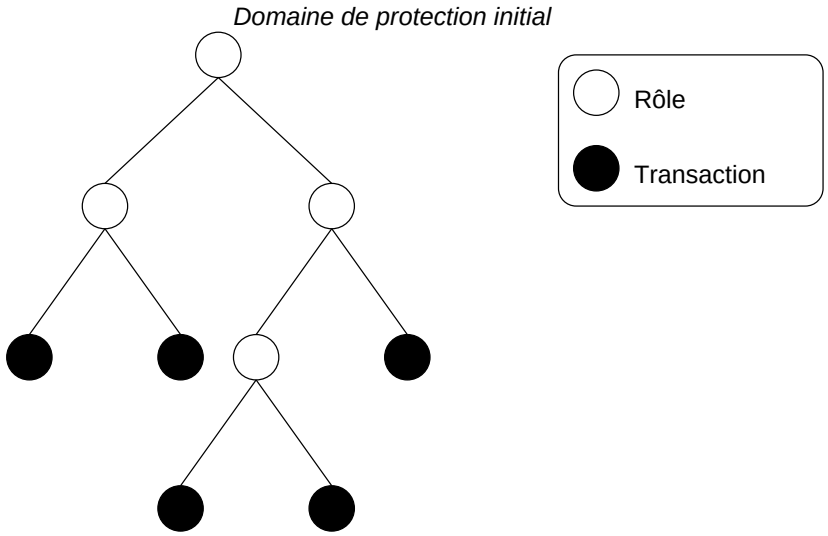



FIG. 8.2: L'arborescence de rôles

La figure montre l'ensemble de domaines qui est accessible depuis le domaine initial d'un usager (le rôle initial est normalement très général, puis les rôles deviennent de plus en plus spécialisés). L'usager est autorisé à assumer trois rôles et à exécuter cinq transactions.
Une transaction doit être représentée par un domaine de protection qui encapsule les données et les opérations sur les données. La C–liste d'une transaction contient donc des capacités d'accès, des activités d'exécution et des capacités de changement de domaine si la transaction a des sous–transactions.
Un rôle doit être représentée par un domaine de protection qui permet d'assumer d'autres rôles et d'exécuter des transactions. La C–liste d'un rôle contient donc uniquement des capacités de changement de domaine.

**Discussion**

On voit ici qu'un domaine de protection sert en même temps à encapsuler les transactions et à définir les droits d'accès autorisés à un rôle. Les appels de changement de domaine permettent d'assumer un rôle plus spécialisé ou d'exécuter une transaction.

### 8.2.4 Réalisation de la politique de la muraille de Chine

Dans le modèle de la muraille de Chine, un usager (un consultant dans notre exemple de la section 3.7) n'a au départ aucune restriction sur les données qu'il peut consulter. Lorsqu'il prend connaissance de certaines informations, il faut être en mesure de restreindre ses droits afin qu'il ne puisse pas avoir accès à des données qui, étant données les informations précédemment utilisées, génèrent un conflit d'intérêt.
En utilisant notre modèle à capacités, il suffit de gérer un domaine de protection par consultant et un domaine de protection associé à la personne qui distribue les informations aux consultants (le patron). Les consultants n'ont au départ aucun droit sur les informations. Lorsqu'une information doit être délivrée à un consultant, elle est transmise par un changement de domaine entre le domaine du patron et le domaine du consultant, en prenant en paramètre une capacité

sur l'information transférée. Le patron doit gérer dans son domaine de protection l'historique des accès permettant de décider si une information doit être transmise à ses employés.

**Discussion**

On voit ici que l'élément clé permettant de réaliser la politique de la muraille de Chine est la séparation entre domaines de protection et le passage de capacités en paramètre lors des changements de domaine.

### 8.2.5 Récapitulation et discussion

Dans ce chapitre, nous avons montré que le modèle à capacités proposé permettait de réaliser les différentes politiques de protection présentées dans le chapitre 3.
En particulier, on a vu quelles caractéristiques de ce modèle à capacités permettent de réaliser ces différentes politiques :

**Politique mandataire.** Les domaines de protection permettent de gérer les différents niveaux d'autorisation. Les changements de domaine permettent de réaliser des écritures vers les niveaux supérieurs sans que des droits en lecture soient donnés aux niveaux inférieurs. Le fait que les interfaces protégées des capacités de changement de domaine soient gérées dans le système (protégées et définies par l'administrateur) permet d'assurer que les changements de domaine vers le haut ne passent que des capacités en lecture.

**Politique discrétionnaire d'Unix.** Les listes de capacités (partagées) de notre modèle de protection permettent de gérer aisément les droits des usagers, des groupes et de l'ensemble des usagers dans le système. Le changement de domaine de protection permet de réaliser l'extension dynamique des droits d'accès (comme le fait le SUID d'Unix).

**Politique par rôle.** Un domaine de protection est associé à chaque rôle. Le changement de domaine de protection permet à un usager de changer de rôle de manière contrôlée.

**Politique de la muraille de Chine.** Un domaine de protection est associé à chaque usager. Le passage de capacité en paramètre permet d'accorder les droits dynamiquement aux usagers. Le superviseur des droits (le patron) est charger de conserver l'historique des accès et d'accorder les accès à la demande en garantissant qu'il n'y a pas de conflit d'intérêt.

En conclusion, on observe que le modèle à capacités est très générique et permet de réaliser la plupart des politiques de protection. En particulier, le modèle de protection que nous proposons se caractérise par le fait que les capacités sont confinées et que la politique de gestion des capacités (principalement comment les capacités sont échangées entre les domaines de protection) est exprimée dans la capacité de changement de domaine elle-même. Ceci implique que les échanges de capacités en paramètre lors d'un changement de domaine sont imposés par la capacité de changement de domaine utilisée. Ainsi, il est possible à l'installation d'une capacité de changement de domaine d'imposer des règles sur le passage de capacités en paramètre (par exemple seules des capacités en lecture peuvent être passées). Cette caractéristique nous permet d'implanter élégamment une politique mandataire.

## 8.3 Facilité de programmation

De nombreux chercheurs se sont intéressés et s'intéressent à la réutilisation du logiciel. La modularité des logiciels a été un grand pas dans ce sens.
De plus en plus, les applications sont développées sous la forme d'un ensemble de composants pouvant être réutilisés dans le cadre de plusieurs applications. Une application consiste en un ensemble de composants interconnectés par des connecteurs. Un connecteur définit un mode d'interaction entre des composants, par exemple un appel de procédure, de méthode ou encore l'envoi d'un message.
Dans la plupart des cas, un composant est considéré comme une boîte noire avec des interfaces clairement définies. L'implémentation du composant n'est pas visible de l'extérieur et la construction d'une application par assemblage de composants ne doit pas nécessiter une quelconque connaissance de l'implémentation des composants réutilisés.
La politique de protection d'une application est généralement définie globalement pour cette application. Ceci implique que si un composant est réutilisé dans le cadre de deux applications ayant des politiques de protection différentes, il devient nécessaire d'être en mesure d'associer des politiques de protection différentes à un même composant réutilisé dans différents environnements.
Le modèle de protection d'Arias va tout à fait dans ce sens, car il permet d'associer différentes politiques de protection à des modules, simplement en spécifiant les interfaces de protection associées aux capacités installées dans les environnements d'exécution [Jensen98b].

### 8.3.1 Architectures logicielles

Une architecture logicielle se base sur trois notions de base : les *composants*, les *connecteurs* et *la configuration* de l'application.

#### Composants

Un *composant simple* constitue l'entité basique de la composition d'une application. Quelques systèmes permet d'assembler plusieurs composants dans un même *composant composite*.
Arias réalise des composants simple par des modules de code et des composants composites par un ensemble de modules de code s'exécutent dans le même domaine de protection.

#### Connecteurs

Un connecteur permet de relier deux composants. Il identifie les points d'entrées utilisés de chaque composant avec leur interface. Quelques composants permet de transformer les données passées en paramètre entre composants, si l'interface du client ne correspond pas exactement à celle du serveur.
Arias réalise des connecteurs par les interfaces de protection. La capacité de changement de domaine identifié les deux talons utilisés pour relier les deux composants (domaines de protection). Outre le transfert des capacités, les talons du client et du serveur permet quelques transformations des données simples pour les adapter à l'interface de l'autre, par exemple transformer un `int` en un `long`.

#### Configuration

La configuration identifie les composants constituant l'application et les connecteurs utilisés pour relier les composants. La configuration doit également spécifier une politique de sécurité pour l'application.

La configuration d'une application Arias est définit par la C–liste du domaine de protection initial de l'application. La configuration d'une application correspond à définir une politique de sécurité pour l'application.

**Remarque ! :** Pour pouvoir utiliser les interfaces de protection comme connecteur entre deux composants, il faut séparer les composants dans deux domaines de protection différents. Si les composants ont besoin des mêmes droits d'accès ceci implique la création de deux domaines de protection quasi identiques, mais en générale la séparation en deux domaines permet d'appliquer la principe de la moindre privilège.

### 8.3.2 Programmation par étapes

La programmation des applications réparties reste toujours très complexe, mais la programmation par étapes ("séparation of concerns") permet de séparer l'algorithmique de l'application de la spécification de la protection [Jensen97b]. La conception de l'algorithmique peut donc être confiée à un expert du domaine et la spécification de la protection à un expert de sécurité (pour un meilleur résultat les deux experts doivent travailler ensemble).

Pour mieux montrer comment utiliser notre modèle de protection, nous présentons quelques scénarios d'application dans la suite.

### 8.3.3 Réalisation de NIS sur Arias

Ici nous allons esquisser comment réaliser deux services disponibles avec NIS ("Network Information Service") développé par Sun Microsystems[3].

#### Service de gestion des mots de passe

Le service de gestion des mots de passe se compose d'une base de données et un ensemble d'opérations qui permet de chercher et de manipuler les informations stockées dans la base de données. Le format des informations stockées en NIS est different de ceux stockées dans le fichier `/etc/passwd`, mais les operations de NIS les présentent sous forme d'une ligne dans le fichier `/etc/passwd`. Nous pouvons donc également modifier le format des informations sous condition de les présenter dans le format du fichier `/etc/passwd`.

[3] L'annexe C présente une introduction très courte à NIS.

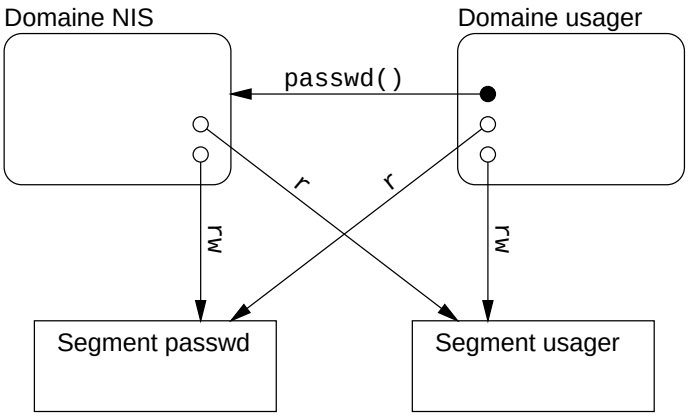



FIG. 8.4: L'architecture du service NIS sur Arias

```
<login>:<passwd>:<uid>:<gid>:<gecos>:<home>:<shell>
```

a) Format d'une entrée passwd en NIS standard

```
<login>:<passwd>:<uid>:<gid>:<adresse du segment usager>
```

b) Format d'une entrée passwd en NIS sur Arias

FIG. 8.3: Format des entrées passwd de NIS

Le format d'une entrée dans la base NIS est illustré sur la figure 8.3 a). L'entrée se compose de deux parties, une partie qui identifie et authentifie l'usager (les entrées `login`, mot de passe (`passwd`), `uid` et `gid`), et une partie qui decrits son environnement d'exécution preféré (les entrées `gecos`, `home` et `shell`). L'intégrité de la première partie est en générale garantit par le système, mais dans la plupart des systèmes les usagers sont libres à modifier la deuxième partie selon leur préférences.
La gestion des informations structurées dans un format à plat pose souvent des problèmes, il faut par exemple assurer qu'un usager n'arrive jamais à substituer son `shell` avec une chaine de caractères de format suivant "`/bin/sh\npirate::0:0::/:/bin/sh`" parce que ceci crée un compte `pirate` avec les privilèges de `root`. Pour éviter ce type de problème, nous proposons de gérer les informations modifiables par l'usager dans un segment séparé apellé le *segment usager*. Le format de la nouvelle entrée NIS est illustré sur la figure 8.3 b). Chaque usager doit avoir le droit de lire les entrées dans la base de NIS et de modifier son propre segment usager. De plus, le serveur NIS doit exporter un point d'entrée qui permet aux usagers de changer leur mot de passe. Le serveur NIS doit avoir le droit de lire tous les segments usager. Une vue très générale de l'architecture de NIS sur Arias est illustrée sur la figure 8.4.

### 8.3.4 Réalisation d'un éditeur coopératif sur Arias

Nous présentons le noyau de base d'un éditeur cooperatif pour montrer comment le modèles des capacités cachées peut être utilisé à réaliser une application coopérative.

Le modèle de l'application est le suivant : le document est réparti parmi un certain nombre d'auteurs qui sont responsable de leur propre partie, il y a un éditeur qui est chargé d'intégrer les différentes parties et de mettre une touche finale avant de publier le document.

La responsabilité d'un chapitre implique que l'auteur doit être le seul utilisateur autorisé à le modifier, c'est–à–dire que chaque chapitre doit être stocké dans un segment séparé et que le domaine de l'auteur doit être le seul domaine à posséder une capacité d'écriture pour ce segment. Le droit de lire le segment doit être diffusé à tous les auteurs.

L'editeur coopératif se compose de plusieurs modules de code qui s'occupent de l'affichage sur l'écran et la manipulation du texte. Nous nous interessons uniquement à ce dernier module.

Chaque domaine exporte également quelques points d'entrée (procédures) qui permettent aux utilisateurs autorisées de faire une opération bien définie sur le texte du chapitre. Les opérations exportées par un domaine comprennent les fonctions suivantes :

`c_o(addr_t chapitre)` qui réalise une correction d'orthographe entre une adresse donnée et jusqu'au fin du chapitre, et

`mettre_ancre(addr_t paragraphe)` qui permet de mettre une ancre dans le texte (à une adresse donnée) pour faire une référence croisée.

La figure 8.5 montre l'éditeur cooperatif avec deux auteurs et un éditeur. L'éditeur a le droit de lire tous les chapitres et de appeler le correcteur d'orthographe dans le domaine de chaque auteur. Les auteurs ont tous les droits de lire et modifier leur propre chapitre, de lire les autres chapitres et d'appeler une fonction qui met une ancre dans le chapitre de quelqu'un d'autre.

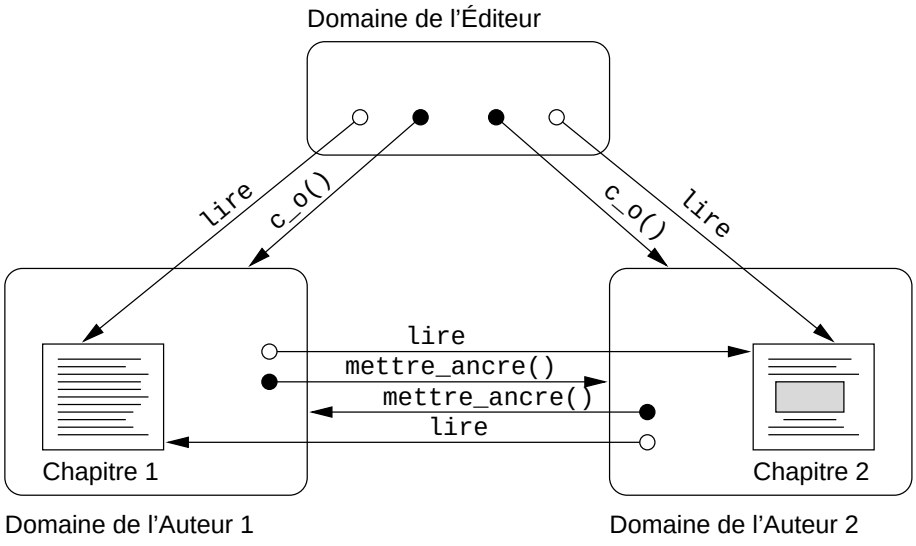



FIG. 8.5: L'éditeur cooperatif

Il faut définir deux interfaces de protection pour cette application, une utilisée entre l'éditeur et un auteur (cf. figure 8.6 a) et une autre utilisée entre auteurs (cf. figure 8.6 b).

```
c_o(CAPA_WRITE addr_t chapitre)
```

a) L'interface de protection entre l'éditeur et un auteur

```
mettre_ancre(CAPA_WRITE addr_t paragraphe)
```

b) L'interface de protection entre deux auteurs

FIG. 8.6: Interfaces de protection de l'éditeur coopératif

Ces interfaces vont faire un appel de changement dynamique de domaine dans un domaine qui possède le droit d'écriture sur l'adresse passée en paramètre.
Remarquons que les auteurs n'ont pas besoin d'appeler la fonction du correcteur d'orthographe dans un chapitre dont ils ne sont pas responsables et que l'éditeur n'a pas besoin de mettre une ancre pour les références croisées dans un chapitre.

### 8.3.5 Discussion

Le modèle des capacités cachées s'adapte bien à la programmation par composants et les architectures logicielles — deux technologies proposées pour faciliter la programmation des applications complexes. La programmatio par étapes permet également de reduire la complexité de programmation des applications sûres.
Pour démontrer comment utiliser le modèle en pratique, nous avons esquissé la réalisation de deux scénarios d'application différentes.
Le scénario NIS montre comment réalise une extension système avec le modèle des capacités cachées. Il n'existe qu'un serveur pour chaque service, nous n'avons donc pas besoin du changement dynamique de domaine.
Le scénario éditeur coopératif montre comment réaliser une application coopérative dans Arias. Pour pouvoir appeler le correcteur d'orthographe et mettre une ancre de référence croisée dans n'importe quel chapitre, nous avons besoin de l'appel de changement dynamique de domaine.

## 8.4 Évolutivité

Il y a deux aspect importants pour l'évolution des droits d'accès : l'évolution des droits pendant l'exécution de l'application et la possibilité de faire évoluer l'application elle–même. Ces deux aspect sont traités dans la suite.

### 8.4.1 Evolution des droits d'accès

La protection par capacité est largement reconnue comme le mécanisme de protection le plus flexible. Les capacités sont facilement passées entre domaines dans les appels de changement de domaine. En fait les seules limites à la diffusion des droits d'accès sont celles imposées par la politique de sécurité et réalisées par le moniteur de référence.

### 8.4.2 Évolution de la sécurité d'une application

Les architectures logicielle ont été proposées pour faciliter l'évolution des applications. Nous avons montré que le modèle de protection d'Arias correspond bien aux architectures logicielles. Les domaines de protection encapsulent les données et les opérations (comme les composants), les interfaces de protection servent en fait de connecteurs entre deux composants et l'arbre des C–listes définit la configuration de l'application.

**Remarque !** Les interfaces de protection sont spécifiées par le programmeur de chaque composant, il faut donc s'assurer que l'interface du client et l'interface du serveur correspondent. L'accord des interfaces correspond au problème de conformité de deux types dans les langages à objets [Jensen98c].

## 8.5 Réutilisabilité des composants logiciels

L'évolution vers la programmation par composition des composants existants nécessite un support pour la réutilisabilité des composants logiciels. De plus, les composants risquent d'être utilisés dans des contextes de sécurité différents (parfois à l'interieur d'un composant et parfois en interaction directe avec un utilisateur potentiellement malveillant), il faut donc avoir la capacité de changer la façon d'interagir avec le composant.
Pour illustrer cet avantage de notre modèle, nous utilisons l'exemple d'un module (composant) implantant un service de noms symboliques[4]. Ce module permet de gérer l'association entre des noms symboliques et des adresses virtuelles (un identificateur unique dans Arias).
Ce service de nommage exporte trois points d'entrée : `create`, `add` et `lookup` qui permettent respectivement de créer un nouveau service de nommage (initialisation), ajouter une une nouvelle entrée associant un nom symbolique à une adresse virtuelle et rechercher l'adresse associée à un nom dans un service de nommage. Les signatures de ces interfaces en langage C sont données ci–dessous.

```
void create(addr_t *ns,
            int elem_size, int no_elem);

void add(addr_t ns, char *name, addr_t addr);

void lookup(addr_t ns, char *name, addr_t *addr);
```

FIG. 8.7: L'interface du code

Il est alors possible d'utiliser le même module de code dans trois contextes de protection différents.
Si le module fait partie intégrante de l'application et est complètement privé à cette application, alors la configuration de la politique de protection va placer dans le domaine de protection

[4]Un article qui décrit cette expérience a été présentée au "Second Euromicro Conference on Software Maintenance and Reengineering" [Jensen98a].

d'exécution de l'application une capacité d'exécution du module de code dans ce domaine. L'exécution de ce module est locale au domaine de protection de l'application. L'application peut appeler sans aucune restriction les trois opération `create`, `add` et `lookup`.
Dans un autre contexte, ce module peut être utilisé afin de gérer un serveur de nommage protégé partagé entre un ensemble d'applications. Le serveur de protection s'exécute dans un domaine de protection qui inclut une capacité d'exécution du module de code (comme ci–dessus). Par contre, les clients de ce serveur de nommage se verront seulement donner une capacité de changement de domaine pour les points d'entrée auxquels ils ont droit. Dans ce cas, ils recevront deux capacités de changement de domaine pour aller s'exécuter dans le domaine de protection du serveur et appeler les opérations `add` et `lookup`. Par contre, il n'ont pas le droit d'appeler l'opération de création et initialisation du serveur de nom qui est partagé et protégé. Les interfaces de protection de ces deux capacités de changement de domaine sont données ci–dessous.

```
void add(IN addr_t ns,
         IN char *name, IN addr_t addr);

void lookup(IN addr_t ns,
            IN char *name, OUT addr_t addr);
```

FIG. 8.8: L'interface d'un service de noms

Enfin, dans un troisième contexte d'utilisation de ce module, nous configurons la politique de protection du module de code réutilisé pour qu'il se comporte comme un serveur de protection. L'objectif est, lorsque le serveur est consulté pour obtenir l'adresse virtuelle associée à un nom symbolique, de retourner au domaine de protection client (appelant) une capacité en lecture sur le segment Arias désigné par cette adresse virtuelle (par exemple pour consulter le document stocké dans ce segment dans le cadre de l'application). Dans ce cas, nous avons la même gestion des domaines de protection que dans le cas précédent, mis à part que les interfaces de protection associées aux capacités de changement de domaine sont les suivantes.

```
void add(IN addr_t ns,
         IN char *name, IN CAPA_READ addr_t addr);

void lookup(IN addr_t ns, IN char *name,
            OUT CAPA_READ addr_t addr);
```

FIG. 8.9: L'interface d'un serveur de protection

On voit que, dans ce cas, les capacités de changement de domaine des opérations d'ajout et de consultation du serveur de protection respectivement donnent et récupèrent une capacité en lecture pour le segment identifié par l'adresse virtuelle (unique dans Arias) passée en paramètre de l'opération.

La séparation entre la définition de la politique de protection et l'algorithmique permet de ré-utiliser un même module de code dans des contextes d'utilisation très différents sans avoir à modifier ce module. De plus, cette séparation rend la définition d'une politique de protection plus simple et plus facile à comprendre.

## 8.6 Possibilité d'intégration de logiciels conçus ailleurs

Nous n'avons pas fait une évaluation particulière de cet aspect, mais notre gestion de code nous permet de définir un module de code à partir d'une bibliothèque partagée d'Unix ; l'integration dans Arias des logiciels sous forme d'une bibliothèque partagée est donc facile à faire.

De plus, il existe aujourd'hui des techniques qui permettent d'encapsuler du code propriétaire et de l'utiliser depuis une bibliothèque partagée. Il suffit de définir un module Arias à partir de cette bibliothèque pour que l'application puisse être utilisée à partir d'Arias. Le code propriétaire s'exécute toujours dans le contexte d'un processus Unix normal, mais le module de code Arias qui permet d'y accéder peut être soumis à la politique de protection du système.

## 8.7 Sécurité de la réalisation

Nous avons identifié deux problèmes de sécurité dans la conception de notre modèle de protection dans Arias. Ces problèmes sont les suivants :

### 8.7.1 Interopérabilité avec le système hôte

L'interopérabilité avec AIX rend le problème d'analyser la sécurité du modèle très complexe (nous avons pas de noyau de sécurité globale), c'est–à–dire que notre moniteur des références est facile à identifier (en plus il est simple et petit), mais il ne marche que pour les références vers Arias, alors il est possible d'éluder ce mécanisme si on arrive à contourner les mécanismes de la protection d'AIX (par ex. si on obtient un accès à la mémoire de la machine à travers `/dev/kmem` ou un accès direct à l'image permanente sur disque)). En plus nous ne maîtrisons pas le canaux de communication d'AIX, des attaques sont donc possible par cette voie.

### 8.7.2 Communication sur le réseau

Ce travail a été réalisé avant la libéralisation du chiffrement en France. Nous n'avons donc pas eu les moyens de protéger les messages envoyés sur le réseau ou de réaliser une authentification entre machines. Les consignes de sécurité pour une installation d'Arias doit donc exiger le contrôle physique des machines dans la grappe.

Le problème de sécurité de communication sur le réseau peut être résolu, soit par un contrôle d'accès physique strict, soit par les techniques d'authentification et de communication sûre étudiées dans le chapitre 2 qui peuvent être intégrées dans Arias afin de protéger la communication sur le réseau.

### 8.7.3 Discussion

L'interopérabilité entre deux systèmes implique toujours que la sécurité de l'ensemble est celle du système le plus faible. Le sous–système de sécurité d'Arias est plus petit et moins complexe que celui d'AIX. Nous prétendons donc que la réalisation du modèle de protection d'Arias n'est pas le point faible de la sécurité

## 8.8 Evaluation de performances

Notre but est de mesurer le coût du changement de domaine et du passage de capacité en paramètre entre les domaines de protection dans le système Arias.
Dès que des domaines de protection sont créés, ils restent actifs jusqu'à ce qu'ils soient détruits explicitement ou que le système soit redémarré. Ainsi, seule une application qui passe dans un domaine de protection inactif paie le coût de création du domaine. Pour mesurer le coût du changement de domaine, nous lançons le programme une fois avant les mesures, afin d'éviter d'inclure les coûts liés à l'initialisation et la pagination.
Nous avons mesuré le temps moyen de réalisation d'un appel inter-domaine de protection. Les mesures ont été effectuées localement sur une station de travail (Bull Estrella avec AIX 4.1.4), car dans le système Arias, la protection et la répartition sont orthogonales.

### 8.8.1 Coût d'ensemble de la protection

Le coût d'ensemble de la protection est mesuré comme le coût moyen d'un appel de procédure vide dans le même module de code, entre deux modules de code dans le même domaine de protection et entre deux modules de code dans deux domaines de protection différents. Les résultats sont donnés dans la table 8.1.

| | |
|---|---|
| coût d'un appel dans le même module de code | 29 ns |
| coût d'un appel entre 2 modules de code | 47 ns |
| coût d'un appel entre 2 domaines de protection | 750 $\mu$s |

TAB. 8.1: Coût d'ensemble de la protection

Le surcoût de notre mécanisme[5] de protection a deux origines : le coût induit par la séparation des fonctions dans des modules de code séparé et le coût induit par le transfert des paramètres et du contrôle entre des domaines de protection différents.
La programmation modulaire sépare le code entre plusieurs modules de code, alors qu'il n'y aurait qu'un seul module de code dans un programme monolithique non protégé. Ainsi, le code exécuté pour un appel de procédure n'est pas toujours local. Dans le même module (et tant que la taille du module n'est pas trop grande), les procédures sont adressées relativement au compteur ordinal. Un appel à une procédure stockée dans un autre module passe à travers une table d'indirection appelée TOC (Table of Contents) associée au module. Ce surcoût n'est pas très élevé (de 29 à 47 ns) et nous ne pensons pas que les applications protégées de grande taille seront fragmentées en de nombreux modules de code.

[5]Le cout d'un RPC local est de l'ordre de 1350 $\mu$s. La performance de notre réalisation est donc respectable, même si nous ne faisons pas tous le travail d'un appel RPC standard.

En général, une application protégée inclut plusieurs domaines de protection. La troisième ligne de la table 8.1 donne le coût d'un appel inter-domaine de protection pour un appel de procédure vide, c'est à dire sans transfert de capacité. Ce coût est élevé comparé à ceux des appels locaux, mais il doit être comparé au coût d'un appel à distance (RPC) local à la machine, ce qui est fait dans la suite.

### 8.8.2 Surcoûts liés à l'implantation

Comme expliqué auparavant, le prototype Arias est basé sur des des modules STREAMS qui sont intégrés dans le noyau. L'utilisation des modules STREAMS a été un bon choix pour le prototypage, mais elle s'est avérée très coûteuse. Dans la suite, nous décomposons notre mécanisme en différents composants afin de mesurer le coût effectivement induit par l'utilisation des modules STREAMS.
Les coûts des différents composants impliqués dans le mécanisme de changement de domaine sont donnés dans la table 8.2.
Le coût d'un composant est mesuré en le court-circuitant et en mesurant la différence avec le coût global.

| | |
|---|---|
| talons client et serveur (emballage des paramètres) | 20 $\mu$s |
| code noyau (transfert des capacités) | 20 $\mu$s |
| STREAMS (RPC et changement de contexte) | 710 $\mu$s |

TAB. 8.2: Coût des composants du changement de domaine

Les résultats confirment que la plus grande partie du coût d'un changement de domaine dans notre réalisation provient de l'utilisation des modules STREAMS et du changement de contexte entre les processus Unix implantant les deux domaines de protection.
Le coût de passage à travers les talons est relativement faible comparé au coût global du mécanisme. Un talon emballe les paramètres, construit la C–liste et vérifie que les capacités fournies par l'autre domaine (l'appelant pour le talon serveur et l'appelé pour le talon client) correspondent aux spécifications de l'interface de protection.
Le code exécuté dans le noyau, qui utilise environ 20 $\mu$s, redirige les requêtes entre les domaines et transfère les capacités en fonction de la C–liste inclus dans le message de la requête.
Afin de donner une idée de l'amélioration en termes de performances que l'on peut attendre d'une ré-implementation, nous avons comparé le changement de domaine implanté avec des modules STREAMS (dans les mêmes conditions que dans notre système) avec un changement de domaine réalisé avec des sockets Unix basés sur TCP. Les résultats présentés dans la table qui suit montrent qu'un gain significatif peut être obtenu en remplaçant les modules STREAMS par un protocole spécialisé fondé sur des fonctions de communication de base.

| | |
|---|---|
| appel inter-domaine basé sur des STREAMS | 710 $\mu$s |
| appel inter-domaine basé sur des sockets (TCP) | 250 $\mu$s |

TAB. 8.3: Coût de différents mécanismes d'appel inter-domaine

## 8.9 Récapitulation

Dans ce chapitre, nous avons présenté une argumentation pour la généricité du modèle. Nous avons montré comment les différents modèles de protection présentés en chapitre 3 peuvent être réalisés sur le modèle à capacités cachées.
La programmation par composants et les architectures logicielles ont été proposées pour faciliter la programmation des applications complexes. Nous avons montré une correspondance directe entre les entités du modèle à capacités cachées et celles des architectures logicielles. Nous avons présenté deux scénarios d'application qui montrent comment utiliser notre service de protection. Le scénario NIS montre que le modèle original [Hagimont96] s'adapte bien aux extensions du système. Le scénario "éditeur coopératif" montre la nécessité du changement dynamique de domaine pour la réalisation des applications coopératives.
La programmation par composants et les architectures logicielles facilitent également l'évolutivité des applications et permettent de réutiliser les composants dans d'autres applications. L'expérience avec le service de noms montre comment le même module de code peut être utilisé dans trois contextes de protection différents.
L'évaluation de performance de l'appel avec changement de domaine indique une efficacité comparable à celle d'un RPC standard. De plus, le remplacement des modules STREAMS par une implantation basée sur des sockets permet un gain de performance significatif. Les structures de gestion des capacités sont essentiellement les mêmes que celles utilisées par Saunier, l'efficacité et le passage à l'échelle de notre réalisation doivent être comparables à ceux rapportés dans sa thèse (cf. pages 114–120 de la thèse de Saunier [Saunier96]).

# Chapitre 9

# Conclusions

## 9.1 Principaux résultats

### 9.1.1 Rappel des objectifs

L'objectif de ce travail était de concevoir et réaliser un mécanisme de base pour la réalisation de la protection dans un système de gestion de données persistantes réparties réalisé sous la forme d'une mémoire virtuelle répartie unique (*single address space*). Ce mécanisme devait avoir les qualités suivantes (voir détails en  1.3).

- Généricité
- Facilité de programmation
- Facilité de réutilisation des composants logiciels
- Facilité d'évolution de la spécification de la protection
- Protection de logiciels récupérés ailleurs

Ces exigences nous ont conduits à une expression de la définition de la protection séparée du texte des programmes d'applications, utilisant des interfaces de protection et mise en œuvre par un mécanisme de "capacités cachées". Sur ces bases, nous avons conçu et réalisé un système de protection répondant aux besoins exprimés. Ce mécanisme a été intégré à la plate-forme Arias, service de gestion de données persistantes réparties réalisé sur un réseau local de serveurs. Nous l'avons utilisé pour des applications simples.
Dans la suite de cette section, nous examinons dans quelle mesure les objectifs de qualité décrits ci-dessus ont été atteints. Nous indiquons ensuite les apports de notre travail.

### 9.1.2 Satisfaction des qualités requises

#### Généricité du modèle

La généricité a été un objectif principal dans la conception du service de protection. Nous avons présenté une argumentation pour la généricite du modèle dans la section 8.2. Nous avons montré que le modèle permet de réaliser la plupart des politiques de protection, y compris une politiques mandataire, discrétionnaire, par rôle, et la politique de la muraille de Chine.
Le confinement des capacités par le système et le contrôle de la délégation rendu possible par les interfaces de protection sont des élément clés pour ce résultat.

#### Facilité de programmation

En raison de sa modularité, le modèle s'adapte bien aux architectures logicielles et à la programmation par composants. Il s'intègre donc bien dans les tendances du génie logiciel d'aujourd'hui.
De plus, la séparation de la spécification de la protection et de l'algorithmique de l'application facilite la programmation par étapes et fournit une certaine transparence de la protection vis–à–vis du programmeur.

#### Évolutivité

La spécification de la protection d'une application dans la configuration du domaine initial et dans les interfaces de protection permet de faire évoluer la protection d'une application sans modifier celle–ci. L'inclusion des interfaces de protection dans les capacités de changement de domaine facilite cette évolution, parce qu'il n'y a pas de talons à modifier comme dans un système RPC standard.

#### Réutilisabilité des composants logiciels

Notre expérience avec différentes politiques de protection pour le service de noms montre la réutilisabilité des composants logiciels dans des contextes de protection différents.
Il faudrait davantage d'expérience avec le système pour savoir si les modules de code vont vraiment être réutilisés.

#### Possibilité d'intégration de logiciels conçus ailleurs

Nous prétendons que des logiciels conçus ailleurs peuvent être encapsulés et protégées par le système Arias, mais nous ne l'avons pas vérifié expérimentalement.

### 9.1.3 Apports du travail

#### Etude et classification des systèmes à capacités

Nous avons réalisé une étude synthétique des modèles de contrôle d'accès et leurs réalisations dans les systèmes répartis existants. Nous avons plus particulièrement étudié les mécanismes à base de capacités logicielles pour les systèmes répartis, et nous en avons proposé une nouvelle taxinomie.

#### Proposition et analyse d'un modèle étendu

Nous avons proposé un modèle étendu du modèle des capacités cachées. Ce modèle utilise deux types de capacités (accès et changement de domaine), et réalise une transparence totale de la manipulation de la capacité par rapport aux programmeurs d'applications. Comme indiqué plus haut, une analyse (non formelle) de ce modèle nous a permis de vérifier qu'il permettait la réalisation des principaux modèles de protection existants.

**Réalisation du modèle dans Arias**

Nous avons fait une réalisation du modèle proposé, intégrée au service de gestion de données réparties Arias, utilisant une mémoire virtuelle partagée à espace unique d'adressage pour l'ensemble des données du système. Le service de stockage des capacités utilise les capacités de stockage persistant du système Arias lui-même et gère de manière partitionnée les diférents types de capacités. Les opérations de changement de domaine utilisent également les fonctions d'Arias pour le couplage de talons dans la mémoire virtuelle. La modularité de la réalisation permet l'installation de différentes politiques de protection.
Nous avons fait une évaluation qualitative et quantitative du modèle proposé et de sa réalisation. Les résultats font l'objet de la section suivante.

## 9.2 Évaluation

L'évaluation decrite dans le chapitre 8 nous a permis de faire un certain nombre d'expériences avec le modèle des capacités cachées. Ces expériences nous ont permis de vérifier que les principales qualités requises étaient effectivement obtenues (9.1). Quelques remarques supplémentaires sont données ci-dessous.

### 9.2.1 Expression transparente de la protection

Un but du modèle des capacités cachées a été de rendre transparente la spécification de la protection. Nous avons obtenu une transparence au niveau du transfert des droits, mais pas au niveau de l'encapsulation. Le programmeur doit toujours être conscient de la structuration de l'application en modules qui peuvent s'exécuter avec des droits d'accès différents.
Le modèle s'applique bien à la programmation par assemblage où les composants sont souvent très autonomes. Le même argument d'autonomie indique que le modèle doit bien s'appliquer aux environnements de programmation par agents mobiles. Cet aspect est le sujet d'une autre thèse en cours dans le cadre du projet Sirac.

### 9.2.2 Support pour les applications coopératives

Le besoin d'introduire les changements de domaine dynamiques indique que le modèle n'est pas bien adapté à la programmation des applications coopératives dans une MVRU. Par contre les exemples du serveur d'impression (cf. 6.4) et du service de noms (cf. 8.5) montrent que le modèle est bien adapté aux services construits comme des extensions du système.

### 9.2.3 Performances

L'évaluation quantitative est restée sommaire, d'autant plus que la recherche de performances élevées ne figurait pas dans les objectifs initiaux. Elle a néanmoins montré que le coût supplémentaire induit par la protection était acceptable dans les situations pratiques où la protection s'exerce entre composants à relativement gros grain. Concernant les surcoûts directement liés à notre implantation, qui sont relativement élevés, nous avons pu vérifier qu'ils sont dus en majeure partie à l'usage de modules STREAMS (justifié dans une réalisation prototype par des considérations de facilité de construction), et qu'une réimplémentation sur

un support plus efficace (sockets Unix) pouvait apporter un gain significatif (de l'ordre d'un facteur 3).

## 9.3 Perspectives

Il y a quatre continuations naturelles du travail presenté dans cette thèse : la continuation de l'intégration du prototype dans la version 2 d'Arias, l'application du modèle des capacités cachées à d'autres systèmes, une continuation du travail sur la taxinomie et une modélisation formelle du modèle à capacités cachées et la preuve formelle de la généricité du modèle. Ces continuations possibles sont décrites dans la suite.

### 9.3.1 Application du modèle dans Arias

Le prototype réalisé a été partiellement integré dans la première version d'Arias. Pour pouvoir faire plus d'expériences avec la programmation selon le modèle, il aurait fallu une intégration complète dans Arias v. 2 (version pré-industrielle). Cette intégration n'a pas été possible en raison du calendrier du projet.

#### Changement dynamique de domaine

La réalisation du changement dynamique de domaine permettrait de faire des expérience avec des applications coopératives. Pour faire cela il faudrait réaliser un support pour un système à objets sur Arias ou sur un système présentant des fonctions équivalentes, par exemple une JVM répartie sur une grappe de machines. Nous envisageons de mener une telle expérience.

#### API de capacités

L'API courante se compose d'un ensemble de fonctions nécessaires pour la mise en œuvre du prototype. Un ensemble d'applications plus étendu nous permettrait de vérifier si l'ensemble est complet et si toutes les fonctions ont des bonnes interfaces.

#### Réalisation d'un SGBD sur Arias

Un système de gestion d'une base de données (SGBD) est en cours de réalisation sur Arias v. 2 [1]. Ce système aurait besoin du service de protection et les expériences avec ce SGBD nous permettraient de vérifier que le système passe bien à l'échelle.

### 9.3.2 Application du modèle dans d'autres systèmes

La protection est un aspect non–fonctionnel des programmes. L'idée de programmer la protection en dehors des modules de code proposée par le modèle semble être bonne. Cette idée a déjà été appliquées dans d'autres contextes par exemple dans le contexte de CORBA [Hagimont97a], dans le contexte des agents mobiles [Jensen97a, Hagimont97b], dans le contexte des architectures de logiciels [Jensen98b] et dans le contexte d'un système d'Unix

[1] La réalisation de ce SGBD est le sujet d'une autre thèse dans le cadre conjoint du projet Sirac et du laboratoire IMAG-LSR

standard [Jensen98d]. L'idée est maintenant en cours de réalisation pour des systèmes auto–configurables programmés dans le contexte de Jini [Jensen99].

### 9.3.3 Application de la taxinomie

L'application de la taxinomie aux autres systèmes à capacités répartis enfin de pouvoir classifier et comparer ces systèmes.
On pourrait ainsi analyser les propriétés d'un système à capacités qui sont déterminées par la classification du système, c'est–à–dire quelles sont les forces et les faiblesses qui peuvent être identifiées à partir de la classification du système.

### 9.3.4 Preuve formelle de la généricité du modèle

Dans cette thèse nous avons présenté une argumentation pour la généricité du modèle. Une description formelle du modèle à capacités cachées permettrait de faire une preuve formelle de ce résultat.

# Annexe A

# Langage de définition des interfaces de protection

Nous avons fait quelques modifications au langage de définition des interfaces de protection (LDIP), par rapport à celui décrit par Saunier. Nous présentons les motivations pour ces modifications et le LDIP dans la suite.

## A.1 Rappel des objectifs

Les objectifs du langage sont les suivants[1] :

- le LDIP doit en premier lieu permettre de préciser la signature des procédures protégées ;
- ce langage doit de plus permettre de spécifier les modalités de transfert éventuel de capacités lors d'un changement de domaine ;
- enfin le langage doit permettre de spécifier quel paramètre détermine le domaine appelé lors d'un appel de changement dynamique de domaine.

La spécification des modalités de transfert éventuel de capacités concerne potentiellement chaque paramètre qui représente une adresse virtuelle, et pour chacun elle doit préciser si une capacité doit accompagner cette adresse ou pas. Si oui, la spécification doit exprimer quel type de capacité doit accompagner l'adresse virtuelle passée en paramètre (capacité d'accès ou capacité de changement de domaine) et le droit de copier et déléguer cette capacité.
Le choix de langage de définition d'interface se base largement sur la grammaire proposée par Saunier. La définition originale du LDIP permet de spécifier à partir de quelle C–liste une capacité doit être transférée (la clause `from` capa–list–ident) et dans quelle C–liste la capacité transférée doit être insérée (la clause `install` capa–list–ident). Notre décision d'identifier une C–liste par l'adresse virtuelle du segment qui contient la C–liste introduit le risque de se tromper parce que le nom n'est pas significatif. Le problème peut être résolu par l'utilisation d'un nom symbolique dans la spécification de l'interface de protection, mais cela ouvre une nouvelle voie d'attaque à travers le service de noms. Nous avons donc choisi de supprimer ces deux clauses dans la spécification du langage [2].

[1] Les deux premiers objectifs ont été identifiés par Saunier.

[2] La supression de ces deux clauses enlève l'ambiguïté dans la grammaire de Saunier qui contient deux définitions différentes de "direction".

Nous présentons d'abord la grammaire du langage de base avant de décrire les extensions nécessaires pour le support du changement dynamique de domaine.

## A.2 Grammaire du langage

La grammaire du langage de spécification des interfaces de protection est décrite par le tableau A.1.

| | | |
|---|---|---|
| **definition–list** | $\longrightarrow$ | definition ';' |
| | \| | definition ';' definition–list |
| **definition** | $\longrightarrow$ | struct–definition |
| | \| | typedef–definition |
| | \| | proc–definition |
| **struct–definition** | $\longrightarrow$ | 'struct' struct–ident '{' arias–declaration–list '}' |
| **arias–declaration–list** | $\longrightarrow$ | arias–declaration ';' |
| | \| | arias–declaration ';' arias–declaration–list |
| **typedef–definition** | $\longrightarrow$ | 'typedef' simple–declaration |
| **proc–definition** | $\longrightarrow$ | 'procedure' proc–ident '(' parameter–list ')' |
| **parameter–list** | $\longrightarrow$ | $\perp$ |
| | \| | parameter ',' parameter–list ) |
| **parameter** | $\longrightarrow$ | direction arias–declaration |
| **direction** | $\longrightarrow$ | 'in' \| 'out' \| 'inout' |
| **arias–declaration** | $\longrightarrow$ | simple–declaration |
| | \| | capa–declaration |
| **simple–declaration** | $\longrightarrow$ | type–ident variable–ident |
| | \| | type–ident variable–ident '[' value ']' |
| | \| | type–ident '*'variable–ident |
| | \| | type–ident '(*' function–ident ')' |
| **capa–declaration** | $\longrightarrow$ | capa–type copy–righttype–ident '*' variable–ident |
| | \| | 'CAPA_DOMAIN' ($\perp$ \| copy–right) type–ident '(*' function–ident ')' |
| **capa–type** | $\longrightarrow$ | 'CAPA_READ' |
| | \| | 'CAPA_WRITE' |
| | \| | 'CAPA_EXECUTE' |
| **copy–right** | $\longrightarrow$ | 'COPY_NO' |
| | \| | 'COPY_REDUCE_CAPA_RIGHT' |
| | \| | 'COPY_REDUCE_COPY_RIGHT' |
| | \| | 'COPY_REDUCE_ALL_RIGHT' |
| | \| | 'COPY_ALL' |

TAB. A.1: La grammaire du langage de définition des interfaces de protection

Nous utilisons le symbole $\perp$ pour désigner une chaîne de charactères vide.

## A.3 Changement dynamique de domaine

Le support pour le changement dynamique de domaine nécessite une modification dans la définition d'une procédure, c'est–à–dire dans la 'proc–definition' (cf. A.2).

| | | |
|---|---|---|
| **proc–definition** | $\longrightarrow$ | 'procedure' proc–ident '(' domain parameter–list ')' |
| **domain** | $\longrightarrow$ | $\perp$ |
| | $\mid$ | 'domain' capa–type adresse |

TAB. A.2: Modifications nécessaires pour le support du changement dynamique de domaine

La modification de la grammaire décrite ci–dessus permet de spécifier une seule adresse virtuelle, ceci correspond à un identificateur d'objet. Il suffit de répéter la spécification de l'adresse pour pouvoir supporter une liste d'adresses avec des capacités associées.

## A.4 Récapitulation

Le langage de définition des interfaces de protection présenté ici correspond largement à celui proposé par Saunier. En fait, nous utilisons une version modifiée du générateur de talon de Saunier. Nos modifications visent principalement à simplifier la génération des talons pour les rendre plus efficaces.
Les modifications par rapport au langage proposé par Saunier sont les suivantes :

- changement de la définition d'une procédure pour permettre l'appel de procédure sans paramètres ;
- ajoût de la spécification du droit de délégation pour une capacité ;
- suppression de la spécification de la C–liste d'origine ou de destination d'un transfert de capacités (les clauses "`install` et "`from`").

# Annexe B

# Interface de programmation des capacités

Cette annexe décrit l'interface (l'API) qui permet de manipuler les capacités. Cette API se compose d'un ensemble de fonctions pour lesquelles nous avons identifié un besoin. Nous ne prétendons pas que cette API soit complète, mais elle nous semble être cohérente.
Les fonctions de l'API sont préfixées par l'acronyme du module auquel elles appartiennent, c'est–à–dire :

**PDM – Protection Domain Module.** Le module responsable de la gestion des domaines de protection.

**PDD – Protection Domain Daemon.** Le démon responsable de la création dynamique des domaines de protection.

**CL – Capability List.** Le module responsable de la gestion des capacités dans les C–listes.

## B.1 Administration du système

La création d'une C–liste insère les droits de manipulation dans la C–liste par défaut du domaine créateur. L'administrateur du système obtient donc le droit de manipuler toutes les C–listes par transitivité et du fait qu'il a créé la première C–liste du système.

## B.2 Domaines de protection

Les fonctions qui permettent de créer et manipuler les domaines de protection sont décrites dans la suite.

```
int PDM_PDCreate(AGt_Addr clist, uid_t uid, gid_t gid)
```

Crée un domaine de protection à partir d'une C–liste, un UID et un GID passés en paramètre. L'adresse de la C–liste devient l'identificateur du nouveau domaine. Il faut que l'appelant possède tous les droits sur la C–liste et que la C–liste ne soit pas la C–liste de base d'un autre domaine. Enfin, il faut être `root` pour spécifier un autre UID ou GID que ceux de l'appelant.

`PDMt_Desc PDRead(AGt_Addr pdid)`

Retourne le descripteur du domaine (l'UID et le GID du domaine par défaut).

`int PDDump(AGt_Addr pdid)`

Imprime le descripteur du domaine et l'arborescence des capacités du domaine sur le `stdout`.

`int PDDestroy(AGt_Addr pdid)`

Supprime un domaine de protection. Il faut que le domaine appelant possède une capacité qui permet de supprimer la C–liste de base du domaine.

## B.3 C–listes

Le segment d'une C–liste contient un entête qui identifie la racine des arborescences des capacités d'accès, de changement de domaine et d'indirection. Cette structure est la même que celle d'un domaine de protection, mais l'UID et GID sont ceux de `nobody` et `nogroup`. Les fonctions qui permettent de créer et manipuler les C–listes sont décrites dans la suite.

`AGt_Addr PDM_CLCreate(AGt_Addr *segment, int size)`

Crée une C–liste vide et insère tous les droits sur cette liste dans la C–liste de base du domaine appelant. La fonction retourne l'adresse du segment créé comme identificateur de la C–liste.

`int CLAdd(AGt_Addr clist, capa_t capa)`

Rajoute la capacité dans la bonne arborscence de capacités dans le segment identifié par `clist`. Il faut que l'appelant possède le droit de rajouter des capacités dans la C–liste. Une valeur de `clist = 0` signifie que la capacité doit être insérée dans la C–liste de base du domaine.

`int CLDel(AGt_Addr clist, capa_t capa)`

Supprime la capacité `capa` dans la C–liste `clist`. Il faut une correspondance exacte entre les droits d'accès et les droits de délégation spécifiés par `capa` et la capacité dans la C–liste pour supprimer la capacité.

`int CLDump(AGt_Addr clist)`

Imprime l'arborescence des capacités du domaine sur le `stdout`.

`int CLDestroy(AGt_Addr clist)`

Supprime la C–liste. Il faut que le domaine appelant possède une capacité qui permet de supprimer la C–liste.

# Annexe C

# NIS : Network Information Service

Le but de cette annexe est de décrire en général comment fonctionne NIS et comment NIS peut être utilisé pour faciliter l'administration des utilisateurs dans un projet tel que SIRAC.

## C.1 Description générale du NIS

NIS est un système développé par Sun Microsystems, qui facilite la maintenance d'une description de la configuration d'un réseau de stations de travail de façon cohérente. L'implementation du NIS repose sur une base des données distribuées, utilisant la technologie client/serveur.
Une base NIS contient normalement les informations suivantes : descriptions des utilisateurs (`/etc/passwd` sans "shadow", `/etc/group` et `/etc/aliases`), adresses IP (`/etc/hosts`), services offerts (`/etc/services`), description des serveurs de boot (`/etc/ethers`, `/etc/bootparams`, etc.). Il n'est pas nécessaire de définir tous les services et on peut sans doute en définir d'autres.
Un système NIS est organisé selon une hiérarchie des serveurs NIS, avec un seul serveur "master" et un client sur tous les machines. Il est possible de definir plusieurs serveurs intermédiaires (appelé serveurs "slave") pour augmenter les performances où la disponibilité du service NIS. Le serveur master est la seul machine qui peut changer les informations dans NIS. Après chaque mise à jour, le "master" diffuse les nouvelles informations à tous les "slaves".

## C.2 La mécanique du NIS

NIS est normalement utilisé comme un supplément aux configurations locales (certaines utilisateurs (root, daemon,...) sont toujours definis dans le `/etc/passwd` local pour assurer le fonctionnement de la machine indépendamment du NIS. L'utilisation du NIS est normalement lancée par une ligne dans le `/etc/passwd` qui contient un "+".
Pour que les programmes qui utilisent les routines pour examiner le `/etc/passwd` (`getuid`, `getpwent`, ...) continuent à marcher il faut installer une version speciale de la bibliothèque C qui comprend les entrées NIS dans `/etc/passwd` et qui sait interroger le serveur NIS pour obtenir l'information demandée.

Certains programmes comme `login`, `ftp`, `rsh`, ... doivent être liés statiquement, donc il faut installer les nouvelles versions de ces programmes adaptées au NIS.

## C.3 Avantages du NIS

NIS permet de maintenir une base de données des utilisateurs et des machines cohérente sur un réseau, c'est–à–dire qu'une seule définition suffira pour que un utilisateur puisse utiliser un ensemble de machines — l'ensemble peut être defini par des groupes internes du NIS ("netgroups") qui peuvent restreindre l'ensemble des utilisateurs qui ont le droit d'accéder à une certaine machine).
L'utilisation des netgroups permet de definir les resources disponibles pour un utilisateur selon un modèle ("template"), par exemple un stagiaire Arias a accès à choiseul, apia et les machines de crash).
Quand l'information du NIS est mise à jour elle est disponible sur toutes les machines instantanément, sans intervention de l'administrateur local.

## C.4 Problèmes lié au NIS

Le "master" NIS est un point faible du système (on ne peut pas mettre à jour l'information dans NIS pendant un crash du server "master").
Les informations definies dans NIS sont généralement visibles partout. Ça veut dire qu'un utilisateur malveillant local peut obtenir une copie des mots de passe de tous les utilisateurs et utiliser un programme comme `crack` ou `john the ripper` pour trouver le mot de passe de quelqu'un d'autre. Il existe de mécanismes pour cacher les mots de passe ("shadow password"), mais ils ne sont pas standardisés.
Un dernier problème est le fait qu'il faut être `root` pour modifier une entrée dans NIS. Pour changer son mot de passe l'utilisateur appelle un programme SUID qui lui permet uniquement de changer son mot de passe. Il n'a plus la possibilité de changer son shell ou l'entrée dit gecos ; les programmes SUID sont normalement très compliqués et Sun a préféré de garder les programmes simples (ils ont eu beaucoup problèmes avec la fonction de changement de shell au début, alors ils ont renoncé à étendre la fonctionalité de NIS).

# Bibliographie


[Accetta86] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, et M. Young, Mach : A New Kernel Foundation for UNIX Development, dans *Proceedings of the Summer 1986 USENIX Conference*, pages 93–112, juillet 1986.

[AVL62] G. M. Adel'son-Velskii et Y. M. Landis, An Algorithm for the Organization of Information, *Dokl. Akad. Nauk SSR*, 146 :263–266, 1962, Traduit en anglais dans "Soviet Math." (Dokl.) 3 (1962), pp. 1259–1263.

[Ames83] S. R. Ames, M. Gasser, et R. R. Schell, Security Kernel Design and Implementation : An Introduction, *IEEE Computer*, 16, no. 7 :14–22, juillet 1985.

[Ames78] S. R. Ames et D. R. Oestreicher, Design of a message processing system for a multilevel secure environment, dans *Proceedings of AFIPS National Computer Conference*, volume 47, pages 765–771, Arlington, Virginia, États–Unis, 1978, AFIPS Press.

[Anderson86] M. Anderson, R. D. Pose, et C. S. Wallace, A Password Capability System, *The Computer Journal*, 29, no. 1 :1–8, 1986.

[Aura99] T. Aura, Distributed Access–Rights Management with Delegation Certificates, dans J. Vitek et C. Jensen, éditeurs, *Secure Internet Programming*, numéro 1603 dans Lecture Notes in Computer Science, pages 211–236, Springer Verlag, juin 1999.

[AXENT99] I. AXENT Technologies, Raptor Firewall 6.0 White Paper, Rapport technique, AXENT Technologies, Inc., 1999.

[Badham83] J. Badham, Wargames, Metro Goldwyn Mayer, 1983.

[Balter95] R. Balter et S. Krakowiak, Objectifs et plan de travail du projet Sirac, Rapport technique Sirac 1–95, Laboratoire IMAG–LSR, juin 1995.

[Balter91] R. Balter, J.-P. Banâtre, et S. Krakowiak, *Construction des systèmes d'exploitation répartis*, Institut National de Recherche en Informatique et Automatique, 1991.

[Bartoli93] A. Bartoli, S. J. Mullender, et M. van der Valk, Wide–Address Spaces — Exploring the Design Space, *Operating Systems Review*, 27, no. 1 :11–17, janvier 1993.

[Bell75] D. E. Bell et L. J. LaPadula, Secure Computer System : Unified Exposition and Multics Interpretation, Rapport technique 2997, MITRE Corporation, juillet 1975.

[Bell73a] D. E. Bell et L. J. LaPadula, Secure Computer Systems : Mathematical Foundations, Rapport technique 2547, Volume I, The MITRE Corporation, mars 1973.

[Bell73b] D. E. Bell et L. J. LaPadula, Secure Computer Systems : A Mathematical Model, Rapport technique 2547, Volume II, The MITRE Corporation, mai 1973.

[Bellissard97] L. Bellissard, *Construction et Configuration d'Applications réparties*, Ph.D. thesis, Institut National Polytechnique de Grenoble, 1997.

[Biba77] K. J. Biba, Integrity Considerations for Secure Computer Systems, Rapport technique, US Air Force Electronic Systems Division, 1977.

[Birrell84] A. D. Birrell et B. J. Nelson, Implementing Remote Procedure Calls, *ACM Transactions on Computer Systems*, 2, no. 1 :39–59, février 1984.

[Boden95] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, et W. Su, Myrinet — A Gigabit–per–Second Local–Area Network, *IEEE Micro*, 15, no. 1 :29–36, février 1995.

[Boykin93] J. Boykin, D. Kirschen, A. Langerman, et S. LoVerso, *Programming under Mach*, Addison-Wesley Publishing Company, 1993.

[Brewer89] D. F. C. Brewer et M. J. Nash, The Chinese Wall Security Policy, dans *Proceedings of the IEEE Symposium on Security and Privacy*, pages 329–339, mai 1989.

[BSI97] *IT–Grundschutzhandbuch 1997 : Maßnahmenempfehlungen für den mittleren Schutzbedarf*, numéro 7252 CD dans BSI, Bundesamt für Sicherheit in der Informationstechnik, 1997.

[Callas98] J. Callas, L. Donnerhacke, H. Finney, et R. Thayer, OpenPGP Message Format, Request for Comments (RFC) 2440, Network Working Group, 1998.

[Cameron91] J. Cameron, Terminator 2 : Judgement Day, Carolco International Inc., juillet 1991.

[Campbell96] R. H. Campbell, T. Qian, W. Liao, et Z. Liu, Active Capability : A Unified Security Model for Supporting Mobile, Dynamic and Application Specific Delegation, Rapport technique, University of Illinois, Urbana–Champaign, février 1996.

[Carter91] J. Carter, J. Bennet, et W. Zwaenepoel, Implementation and performance of Munin, dans *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 152–164, Ashevill, North Carolina, États–Unis, octobre 1991.

[Cayuela92] J. Cayuela, P. Y. Chevalier, A. Duda, A. Freyssinet, D. Hagimont, S. Lacourte, P. Ledot, M. Riveill, et X. Rousset de Pina, La machine Guide 2, Bull–IMAG note de travail interne K16.archi.

[Chang98] C.-C. Chang, G. Czajkowski, C. Hawblitzel, D. Hu, et T. von Eicken, Security versus Performance Tradeoffs in RPC Implementations for Safe Language Systems, dans *Proceedings*, pages 158–162, Sintra, Portugal, septembre 1998.

[Chase99] J. Chase, Communication privée, août 1999.

[Chase95] J. Chase, *An Operating System Structure for Wide-Address Architectures*, Ph.D. thesis, University of Washington, août 1995.

[Chase94] J. Chase, H. M. Levy, M. J. Feeley, et E. D. Lazowska, Sharing and Protection in a Single Address Space Operating System, *ACM Transactions on Computer Systems*, 12, no. 4 :271–307, 1994.

[Chase93] J. Chase, H. M. Levy, et V. Issarny, Supporting Distribution in Single-Address Space Operating Systems, dans *Proceedings of the 5th ACM SIGOPS European Workshop*, septembre 1992, *Also appeared in Operating Systems Review, ACM SIGOPS, April 1993.*

[Chase92] J. Chase, H. M. Levy, E. Lazowska, et M. Baker-Harvey, Opal : A Single Address Space System for 64-Bit Architectures, dans *Proceedings of IEEE Workshop on Workstation Operating Systems*, avril 1992.

[CheckPoint99] C. Point, Check Point FireWall–1 : Technical Overview, Rapport technique P/N 31400000010, Check Point, avril 1999.

[Clark87] D. D. Clark et D. R. Wilson, A Comparison of Commercial and Military Computer Security Policies, dans *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, California, États–Unis, avril 1987.

[Cortes96a] E. Pérez-Cortés, J. Han, et J. Mossière, Construction de protocoles de cohérence sur une interface générique de mémoire répartie partagée, dans *Journées sur la Mémoire Partagée Répartie (MPR'96)*, Bordeaux, mai 1996.

[Cortes96b] E. Pérez–Cortés, *La cohérence sur mesure dans une mémoire virtuelle partagée répartie*, Ph.D. thesis, Institut National Polytechnique de Grenoble, 1996.

[Cortes95a] E. Pérez-Cortés, P. Dechamboux, et J. Han, Generic Support for Synchronization and Consistency in Arias, dans *5th. Workshop on Hot Topics in Operating Systems*, pages 113–118, mai 1995.

[Cortes95b] E. Pérez–Cortés, Cohérence et synchronization dans une mémoire virtuelle partagée répartie, Rapport technique Sirac 3–95, Laboratoire IMAG–LSR, octobre 1995.

[Coulouris94a] G. Coulouris et J. Dollimore, Requirements for security in cooperative work : two case studies, Rapport technique 671, Queen Mary and Westfield College, University of London, mai 1994.

[Coulouris94b] G. Coulouris et J. Dollimore, A security model for cooperative work, Rapport technique 674, Queen Mary and Westfield College, University of London, août 1994.

[CTCPEC93] *The Canadian Trusted Computer Product Evaluation Criteria, V. 3.0e*, Canadian System Security Center, Communications Security Establishment of Canada, janvier 1993.

[Dearle94a] A. Dearle, R. di Bona, J. Farrow, F. Henskens, A. Lindström, J. Rosenberg, et F. Vaughan, Grasshopper : An Orthogonally Persistent Operating System, *Computing Systems*, 7, no. 3 :289–312, 1994.

[Dearle94b] A. Dearle, R. di Bona, J. Farrow, F. Henskens, D. Hulse, A. Lindström, S. Norris, J. Rosenberg, et F. Vaughan, Protection in the Grasshopper Operating System, dans *Proceedings of the 6th International Workshop on Persistent Object Systems*, Tarascon, France, septembre 94.

[Denning76] D. E. Denning, A Lattice Model of Secure Information Flow, *Communications of the ACM*, 19, no. 5 :236–243, mai 1976.

[Dennis66] J. B. Dennis et E. C. Van Horn, Programming Semantics for Multiprogrammed Computations, *Communications of the ACM*, 9, no. 3 :143–155, mars 1966.

[Diffie76] W. Diffie et M. E. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, IT–22, no. 6 :644–654, novèmbre 1976.

[DoD85] *Trusted Computer Security Evaluation Criteria*, numéro 5200.28-STD dans Department of Defence Standard, Department of Defence, Décembre 1985.

[Douglis91] F. Douglis et J. K. Ousterhout, Transparent Process Migration : Design Alternatives and the Sprite Implementation, *Software — Practice and Experience*, 21, no. 8 :757–785, août 1991.

[Dutton95] H. Dutton et P. Lenhard, *Asynchronous Transfer Mode (ATM) Technical Overview*, Prentice Hall, 1995.

[Eager88] D. L. Eager, E. D. Lazowska, et J. Zahorjan, The Limited Performance Benefits of Migrating Active Processes for Load Sharing, dans *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 63–72, Santa Fe, NM, Etats Unis, 1988.

[Loepere92a] K. L. (Ed.), *OSF Mach Kernel Principles*, Open Software Foundation, Inc. & Carnegie Mellon University, janvier 1992.

[Loepere92b] K. L. (Ed.), *OSF Mach Kernel Interfaces*, Open Software Foundation, Inc. & Carnegie Mellon University, juillet 1992.

[Power99] R. P. (Ed.), 1999 CSI/FBI Computer Crime and Security Survey, Computer Security : Issues & Trends Vol. V, No. I, Computer Security Institute, 1999.

[Ellison99] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, et T. Ylonen, SPKI Certificate Theory, INTERNET-DRAFT <draft-ietf-spki-cert-theory-05.txt>, Internet Engineering Task Force (IETF), mai 1999, Disponible sur le site web de l'IETF, URL="`http ://www.ietf.org/internet-drafts/draft-ietf-spki-cer`

[Ellison98] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, et T. Ylonen, Simple Public Key Certificate, INTERNET-DRAFT <draft-ietf-spki-cert-structure-05.txt>, Internet Engineering Task Force (IETF), mars 1998, Disponible sur le site web de l'IETF, URL="`http ://www.ietf.org/internet-drafts/draft-ietf-spki-cer`

[Fabry74] R. S. Fabry, Capability–Based Adressing, *Communications of the ACM*, 17, no. 7 :403–412, juillet 1974.

[Feeley95] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, et C. A. Thekkath, Implementing Global Memory Management in a Workstation Cluster, dans *Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 201–212, Copper Mountain, Colorado, États–Unis, Décembre 1995.

[Feiertag79] R. J. Feiertag et P. G. Neumann, The Foundations of a Provably Secure Operating System (PSOS), dans *Proceedings of AFIPS National Computer Conference*, volume 48, pages 329–334, Arlington, Virginia, États–Unis, 1979, AFIPS Press.

[Ferraiolo99] D. F. Ferraiolo, J. F. Barkeley, et D. R. Kuhn, A Role Based Access Control Model and Reference Implementation within a Corporate Intranet, *ACM Transactions on Information and Systems Security*, 1999.

[Ferraiolo92] D. F. Ferraiolo et D. R. Kuhn, Role–Based Access Control, dans *Proceedings of 15th National Computer Security Conference*, 1992.

[Ferrari86] D. Ferrari et S. Zhou, A Load Index for Dynamic Load Balancing, dans *Proceedings og 1986 Fall Joint Computer Conference*, Dallas, TX, Etats Unis, 1986.

[Fillo97] M. Fillo et R. B. Gillet, Architecture and implementation of MEMORY CHANNEL, *Digital Technical Journal*, 9, no. 1 :27–41, 1997.

[Freier96] A. O. Freier, P. Karlton, et P. C. Kocher, The SSL Protocol : Version 3.0, INTERNET-DRAFT <draft-freier-ssl-version3-02.txt>, Internet Engineering Task Force (IETF), 1996.

[Gasser88] M. Gasser, *Building a Secure Computer System*, Van Nostrand Reinhold Company, Inc., New York, états Unis, 1988.

[Geib97] J.-M. Geib, C. Gransart, et P. Merle, *CORBA : des concepts à la pratique*, Masson, Paris, France, octobre 1997.

[Gibson84] W. Gibson, *Neuromancer*, Victor Gollancz Ltd., 1984.

[Goldberg96] I. Goldberg, D. Wagner, R. Thomas, et E. A. Brewer, A Secure Environment for Untrusted Helper Applications, dans *Proceedings of the 6th USENIX Security Symposium*, pages 1–13, 1996.

[Gong89a] L. Gong, On Security in Capability–Based Systems, *Operating Systems Review*, 23, no. 2, avril 1989.

[Gong89b] L. Gong, A Secure Identity–Based Capability System, dans *Proceedings of the IEEE Symposium on Security and Privacy*, pages 56–63, Oakland, California, États–Unis, mai 1989.

[Hagimont97a] D. Hagimont, O. Huet, et J. Mossière, A Protection Scheme for a CORBA Environmen, dans *ECOOP Workshop on CORBA*, juin 1997.

[Hagimont97b] D. Hagimont et L. Ismail, A Protection Scheme for Mobile Agents on Java, dans *3rd ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, septembre 1997.

[Hagimont96] D. Hagimont, J. Mossière, X. Rousset de Pina, et F. Saunier, Hidden Software Capabilities, dans *16th International Conference on Distributed Computing Systems*, pages 282–289, Hong Kong, mai 1996.

[Hagimont95] D. Hagimont et F. Saunier, La protection dans un service de gestion de données persistantes partagées, Rapport technique Sirac 6–95, Laboratoire IMAG–LSR, octobre 1995.

[Han96] J. Han, *Conception et réalisation d'une mémoire partagée répartie*, Ph.D. thesis, Institut National Polytechnique de Grenoble, 1996.

[Harrison76] M. A. Harrison, W. L. Ruzzo, et J. D. Ullman, Protection in Operating Systems, *Communications of the ACM*, 19, no. 8 :461–471, août 1976.

[Heiser98] G. Heiser, K. Elphinstone, J. Vochteloo, S. Russell, et J. Liedtke, Implementation and Performance of the Mungi Single-Address-Space Operating System, *Software : Practice and Experience*, 28, no. 9 :901–928, juillet 1998.

[Heiser94] G. Heiser, K. Elphinstone, S. Russell, et J. Vochteloo, Mungi : A Distributed Single Address-Space Operating System, dans *Proceedings of the 17th Australasian Computer Science Conf* , pages 271–280, Christchurch, New Zealand, janvier 1994.

[Heiser93] G. Heiser, K. Elphinstone, S. Russell, et G. R. Hellestrand, A Distributed Single Address-Space Operating System Supporting Persistence, Rapport technique, University of New South Wales, 1993.

[IEEE92] IEEE, IEEE Standard for Scalable Coherent Interface (SCI), Standard 1596, The Institute of Electrical and Electronics Engineering, 1992.

[IEEE98] IEEE, IEEE STandard for Gigabit Ethernet, Rapport technique, The Institute of Electrical and Electronics Engineering, 1998.

[ISO15408] *Les Critères Communs d'évaluation de la sécurité des technologies de l'information*, numéro 15408 dans Norme internationale ISO, International Standards Organisation (ISO), juin 1999.

[Issarny97] V. Issarny, Configuration-Based Programming Systems, dans F. Plasil et K. G. Jeffery, éditeurs, *Proceedings of SOFSEM'97 : Theory and Practice of Informatics*, volume LNCS 1338, pages 183–200, Milovy, Czech Republic, 1997, Springer Verlag.

[ITSEC91] *Information Technology Security Evaluation Criteria*, European Comunities, juin 1991.

[ITU93] T. S. S. of ITU, *Information Technology — Opens Systems Interconnection — The Directory : Authentication Framework*, numéro X.509 dans ITU–T Recomandation, International Telecomunication Union, novembre 1993, Standard international ISO/IEC 9594–8 : 1995 (E).

[JCSEC92] *The Japanese Computer Security Evaluation Criteria, Draft V1.0*, Le ministère de l'industrie et du commerce, août 1992.

[Jensen99] C. Jensen et M. Haahr, Towards a Security Framework for Ad Hoc Applications, Soumis au deuxième workshop de Distributed Object Security, OOPSLA'99.

[Jensen98a] C. Jensen et D. Hagimont, Protection Reconfiguration for Reusable Software, dans *Second Euromicro Conference on Software Maintenance and Reengeneering*, pages 74–81, Florence, Italy, mars 1998.

[Jensen98b] C. Jensen, Secure Software Architectures, dans *Proceedings of the Eighth Nordic Workshop on Programming Environment Research*, pages 239–246, Ronneby, Suéde, août 1998.

[Jensen98c] C. Jensen et D. Hagimont, Mutual Suspicion in a Generic Object-Support System, dans *Joint Proceedings of the ECOOP Workshop Workshop on Distributed Object Security and the 4th Workshop on Mobile Object Systems (Secure Internet Mobile Computations)*, pages 15–20, Brussels, Belgium, juillet 1998.

[Jensen98d] C. Jensen et D. Hagimont, Protection Wrappers : A Simple and Portable Sandbox for Untrusted Applications, dans *Proceedings of the 8th ACM SIGOPS European workshop*, pages 104–110, Sintra, Portugal, septembre 1998.

[Jensen97a] C. Jensen et L. Ismail, Capability Based Protection for Hosting Mobile Code, dans *Proceedings of the 2nd European Research Seminar on Advances in Distributed Systems*, pages 234–240, 1997.

[Jensen97b] C. Jensen, Reducing Complexity of Distributed Application Protection, Presenté au 4th Cabernet Radicals Workshop, Rithmynon, Crete, septembre 1997.

[Jospin99] L. Jospin, Conférence de presse de Monsieur Lionel Jospin, Premier ministre, à l'issue du Comité interministériel pour la société de l'information, Hôtel de Matignon, Conférence de presse, janvier 1999, Disponible sur le web, URL="`http ://www.premier-ministre.gouv.fr/PM/D190199.HTM`".

[Kain86] R. Y. Kain et C. E. Landwehr, On Access Checking in Capability–Based Systems, dans *Proceedings of the IEEE Symposium on Security and Privacy*, avril 1986.

[Karger84] P. A. Karger et A. J. Herbert, An Augmented Capability Architecture to Support Lattice Security and Traceability of Access, dans *Proceedings of the IEEE Symposium on Security and Privacy*, 1984.

[Keleher96] P. Keleher, *CVM : The Coherent Virtual Machine*, University of Maryland, novembre 1996.

[Keleher94] P. Keleher, A. L. Cox, S. Dwarkadas, et W. Zwaenepoel, ThreadMarks : Distributed shared memory on standard workstations and operating systems, dans *Proceedings of the Winter 1994 USENIX Conference*, pages 115–132, San Francisco, California, États–Unis, janvier 1994.

[Kirtland98] M. Kirtland, *Designing Component–Based Applications*, Microsoft Press, 1998.

[Knaff96] A. Knaff, *Conception et réalisation d'un système de stockage fiable extensible pour un système à objets persistants répartis*, Ph.D. thesis, Université Joseph Fourier (Grenoble I), 1996.

[Knuth] D. E. Knuth, *The Art of Computer Programming, Volume III*, chapitre Searching, pages 458–478, Addison Wesley, deuxième edition, 1998.

[Koch98] P. Koch, E. Cecchet, et X. Rousset de Pina, Global Management of Coherent Shared Memory on a SCI Cluster, dans *Proc. SCIEurope'98, a Conference Stream of EMMSEC'98,*, Bordeaux, France, septembre 1998.

[Kohl93] J. Kohl et C. Neuman, The Kerberos Network Authentication Service (V5), Request for Comments (RFC) 1510, Network Working Group, septembre 1993.

[Koldinger92] E. Koldinger, J. Chase, et S. Eggers, Architectural Support for Single Address Space Operating Systems, dans *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, octobre 1992.

[Kotz94] D. Kotz et P. Crow, The Expected Lifetime of "Single–Address–Space" Operating Systems, dans *Proceedings of SIGMETRICS '94*, pages 161–170, Santa Clara, California, Etats Unis, mai 1994.

[Kunz91] T. Kunz, The influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, *IEEE Transactions on Software Engineering*, 17, no. 7, juillet 1991.

[Lai92] X. Lai, *On the Design and Security of Block Ciphers*, Hartung–Gorre, 1992.

[Lampson92] B. Lampson, M. Abadi, M. Burrows, et E. Wobber, Authentication in Distributed Systems : Theory and Practice, *ACM Transactions on Computer Systems*, 10, no. 4, novembre 1992.

[Lampson81] B. W. Lampson, *Distributed Systems — architecture and Implementation*, chapitre Atomic Transactions, pages 246–265, numéro 105 dans Lecture Notes in Computer Science, Springer Verlag, 1981.

[Lampson73] B. W. Lampson, A Note on the Confinement Problem, *Communications of the ACM*, 16, no. 10 :613–615, octobre 1973.

[Lampson71] B. W. Lampson, Protection, dans *Proceedings of the 5th Princeton Symposium on Information Sciences and Systems*, pages 437–443, mars 1971, réimprimé dans *Operating Systems Review, 8, 1* janvier 1974 pages 18–24.

[Landwehr81] C. E. Landwehr, Formal Methods for Computer Security, *ACM Computing Surveys*, 13, no. 3 :247–278, septembre 1981.

[Levy84] H. M. Levy, *Capability–Based Computer Systems*, Digital Press, 1984.

[Li89] K. Li et P. Hudak, Memory coherence in shared virtual memory systems, *ACM Transactions on Computer Systems*, 7, no. 4 :321–359, novembre 1989.

[Litzkow87] M. J. Litzkow, M. Livny, et M. W. Mutka, Condor : A Hunter of Idle Workstations, Computer Science Technical Report 730, Computer Science Department, University of Wisconsin-Madison, 1987.

[Lomet77] D. B. Lomet, Process Structuring, Synchronization, and Recovery Using Atomic Actions, *SIGPLAN Notices*, 12, no. 3 :128–137, 1977.

[Mazieres98] D. Mazières et M. F. Kaashoek, Escaping the Evils of Centralized Control with self–certifying pathnames, dans *Proceedings of the 8th ACM SIGOPS European workshop*, pages 118–125, Sintra, Portugal, septembre 1998.

[McCauley79] E. J. McCauley et P. J. Drongowski, KSOS : The Design of a Secure Operating System, dans *Proceedings of AFIPS National Computer Conference*, volume 48, pages 345–353, Arlington, Virginia, États–Unis, 1979, AFIPS Press.

[McGraw99] G. McGraw et E. W. Felten, *Securing Java : Getting Down to Business With Mobile Code*, chapitre 2, John Wiley & Sons, 1999.

[McKusick96] M. K. McKusick, K. Bostic, M. J. Karels, et J. S. Quarterman, *The Design and Implementation of the 4.4 BSD Operating System*, Addison–Wesley Publishing Company, Inc., 1996.

[Meeks99a] B. Meeks et A. Boyle, White House Web site shut down, MSNBC, 12 mai 1999, Disponible sur le web, URL="`http ://www.zdnet.com/zdnn/stories/0,4586,2257784,00.htm`

[Meeks99b] B. Meeks, A. Boyle, et B. Sullivan, Hack attack knocks out FBI site, MSNBC, 26 mai 1999, Disponible sur le web, URL="`http ://www.zdnet.com/zdnn/stories/0,4586,2266648,00.html`".

[Monson-Haefel99] R. Monson-Haefel, *Enterprise Javabeans*, O'Reilly & Associates, juin 1999.

[Murray93a] K. Murray, T. Stiermerling, T. Wilkinson, et P. Kelly, Angel : Resource Unification in a 64–bit Micro–Kernel, dans *Proceedings of th 27th Hawaii International Conference on System Science*, septembre 1993.

[Murray93b] K. Murray, T. Wilkinson, P. Osmon, A. Saulsbury, T. Stiermerling, et P. Kelly, Design and Implementation of an Object–Orientated 64–bit Single Address Space Microkernel, dans *2nd USENIX Symposium on Microkernels and other Kernel Architectures*, 1993.

[Needham78] R. M. Needham et M. D. Schroeder, Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM*, 21, no. 12 :993–999, Décembre 1978.

[Needham77] R. M. Neeedham et R. D. H. Walker, The Cambridge CAP Computer and its protection system, dans *Proceedings of the 6th ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–10, novembre 1977.

[Neumann77] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, et L. Robinson, A Provably Secure Operating System : The System, its Applications, and Proofs, Rapport technique, SRI International, février 1977.

[Nicomette96] V. Nicomette, *La protection dans les systèmes à objets répartis*, Ph.D. thesis, Institut National Polytechnique de Toulouse, 1996.

[Omang98] K. Omang, Performance of a Cluster of PCI based UltraSparc Workstations Interconnected with SCI, dans Springer Verlag, éditeur, *Proceedings of Network–Based Parallel Computing, Communication, Architecture, and Applications (CANPC'98)*, volume 1362 de *Lecture Notes in Computer Science*, pages 232–246, janvier 1998.

[OMG98] *CORBAservices : Common Object Sercices Specification*, chapitre Security Service Specification, Object Management Group, décembre 1998, Disponible sur le web : URL="`http ://www.omg.org/homepages/secsig/secrtf.html`".

[Rivest98] R. L. Rivest, *Chaffing and Winnowing : Confidentiality without Encryption*, MIT Lab for Computer Science, avril 1998, Publié sur le Web, URL="`http ://theory.lcs.mit.edu/ rivest/chaffing.txt`".

[Rivest78] R. L. Rivest, A. Shamir, et L. Adleman, On a Method for Obtaining Digital Signatures and Public Key Cryptosystems, *Communications of the ACM*, 21, no. 2 :120–126, fevrier 1978.

[Sandberg85] R. Sandberg, D. Goldberg, K. S, D. Walsh, et B. Lyon, Design and Implementation of the Sun Network File System, dans *Proceedings of the Summer 1985 USENIX Conference*, pages 119–130, Portland, Oregon, États–Unis, juin 1985.

[Sandhu96] R. Sandhu, Access Control : The Neglected Frontier, dans *Proceedings of the First Australasian Conference on Information Security and Privacy*, Wollongong, Australia, juin 1996.

[Sandhu93] R. S. Sandhu, Lattice–Based Access Control Models, *IEEE Computer*, 26, no. 11 :9–19, novembre 1993.

[Satyanarayanan85] M. Satyanarayanan, J. H. Howard, D. A. Nicols, R. N. Sidebotham, A. Z. Spector, et M. J. West, The ITC Distributed File System : Principles and Design, dans *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 35–50, 1985.

[Saunier96] F. Saunier, *Protection d'une mémoire virtuelle répartie par capacités implicites*, Ph.D. thesis, Institut National Polytechnique de Grenoble, 1996.

[Saunier95] F. Saunier, Service de Protection d'une Mémoire Virtuelle Répartie, dans Journée des Jeunes Chercheurs, *Réseau Doctoral en Architecture des Systèmes et des Machines Informatiques*, Rennes, octobre 1995, IRISA.

[Schneier96] B. Schneier, *Applied Cryptography*, John Wiley & Sons, Inc., deuxième edition, 1996.

[Schnorr91] C. P. Schnorr, Efficient Signature Generation for Smart Cards, *Journal of Cryptology*, 4, no. 3 :161–174, 1991.

[SCSSI94] S. C. de la Sécurité des systèmes d'Information (S.C.S.S.I.), *La ménace et les attaques informatiques*, numéro 650 dans DISSI/SCSSI, République Française, premier ministre, mars 1994.

[SCSSI93] S. C. de la Sécurité des systèmes d'Information (S.C.S.S.I.), *Protection des informations sensibles ne relevant pas du secret de défense. Recommandations pour les postes de travail informatiques*, numéro 600 dans DISSI/SCSSI, République Française, premier ministre, Secrétariat général de la défense nationale, 1993.

[SCSSI91a] S. C. de la Sécurité des systèmes d'Information (S.C.S.S.I.), *Fiche d'expression rationelle des objectifs de sécurité des systèmes d'information*, numéro 150 dans DISSI/SCSSI, République Française, premier ministre, février 1991.

[SCSSI91b] S. C. de la Sécurité des systèmes d'Information (S.C.S.S.I.), *Recommandations d'installation des sites et systèmes des informations sensibles ne relevant pas du secret de défense protection des informations sensibles contre les signaux compromettants*, numéro 400 dans DISSI/SCSSI, République Française, premier ministre, octobre 1991.

[Shapiro99] J. S. Shapiro, *EROS : A Capability System*, Ph.D. thesis, Université de Pennsylvanie, 1999.

[Shapiro86] M. Shapiro, Structure and Encapsulation in Distributed Systems : The Proxy Principle, dans *Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 198–204, Cambridge, Massachusetts, États–Unis, juin 1986.

[Snyder81] L. Snyder, Formal Models of Capability–Based Protection Systems, *IEEE Transactions on Computers*, C–30, no. 3 :172–181, mars 1981.

[Soinne98] F. Soinne, SecurWare Netwall Version 3.3, Rapport technique, Bull, juin 1998, Disponible sur le web, URL="`http ://www-frec.bull.fr/OSBU2_0/wp_netwall.htm`".

[Spector89] A. Z. Spector et M. L. Kazar, Wide Area File Services and the AFS Experimental System, *Unix Review*, 7, no. 3, 1989.

[NBS94] N. B. of Standards, *Digital Signature Standard*, numéro 186 dans NBS FIPS PUB, U. S. Department of Commerce, mai 1994.

[NBS77] N. B. of Standards, *Data Encryption Standard*, numéro 46 dans NBS FIPS PUB, U. S. Department of Commerce, janvier 1977.

[Steiner88] J. G. Steiner, B. C. Neuman, et J. I. Schiller, Kerberos : An Authentication Service for Open Network Systems, dans *USENIX Conference Proceedings*, pages 191–202, février 1988.

[Stoll89] C. Stoll, *Le nid du coucou*, Albin–Michel, 1989, Titre original : "The Cuckoo's Egg".

[Sun89] Sun Microsystems Inc., NFS : Network File System Protocol Specification, Request for Comments (RFC) 1094, Network Working Group, mars 1989.

[Tanenbaum90] A. S. Tanenbaum, R. van Renesse, H. van Staveren, G. J. Sharp, S. J. Mullender, A. J. Jansen, et G. van Rossum, Experiences with the Amoeba Distributed Operating System, *Communications of the ACM*, 33, no. 12 :46–63, décembre 1990.

[Tanenbaum86] A. S. Tanenbaum, S. J. Mullender, et R. van Renesse, Using Sparse Capabilities in a Distributed Operating System, dans *Proceedings of the 6th International Conference in Computing Systems*, pages 558–563, juin 1986.

[Thomas94] R. K. Thomas et R. S. Sandhu, Conceptual Foundation for a Model of Task-based Authorisations, dans *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 66–79, Franconia, New Hampshire, États–Unis, juin 1994.

[Thompson74] K. Thompson et D. M. Richie, The UNIX Timesharing System, *Communications of the ACM*, 17, no. 7 :365–375, juillet 1974.

[Tock94] T. D. Tock, *An Extensible Framework for Authentication and Delegation*, Master's thesis, University of Illinois at Urbana–Champaign, 1994.

[Tullman98] P. Tullman et J. Lepreau, Nested Java Processes : OS Structure for Mobile code, dans *Proceedings of the 8th ACM SIGOPS European workshop*, pages 111–117, Sintra, Portugal, septembre 1998.

[STREAMS90] UNIX Software Operation, *UNIX SYSTEM V RELEASE 4 Programmers Guide : STREAMS*, Prentice-Hall, Inc., 1990.

[Vochteloo98] J. Vochteloo, *Design, Implementation and Performance of Protection in the Mungi Single Address Space Operating System*, Ph.D. thesis, University of New South Wales, juin 1998.

[Vochteloo96] J. Vochteloo, K. Elphinstone, S. Russel, et G. Heiser, Protection Domain Extensions in Mungi, dans *Proceedings of the 5th IWOOS*, pages 161–165, Seattle, WA, USA, octobre 1996.

[Vochteloo93] J. Vochteloo, S. Russel, et G. Heiser, Capability–Based Protection in the Mungi Operating System, dans *Proceedings of the 3rd IWOOS*, pages 108–115, Asheville, NC, USA, Décembre 1993, *A version of this article is available as Technical Report number SCS&E 9309 from the University of New South Wales, March, 1993.*

[vonEicken99] T. von Eicken, C.-C. Chang, G. Czajkowski, C. Hawblitzel, D. Hu, et D. Spoonhower, J–Kernel : A Capability–Based Operating System for Java, dans J. Vitek et C. Jensen, éditeurs, *Secure Internet Programming*, numéro 1603 dans Lecture Notes in Computer Science, pages 369–394, Springer Verlag, juin 1999.

[Wahbe93] R. Wahbe, S. Lucco, T. E. Anderson, et S. L. Graham, Efficient Software–Based Fault Isolation, dans *Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP'93)*, pages 203–216, Décembre 1993.

[Wassenaar95] *The Wassenaar Arrangement on Export Controls for Conventional Arms and Dual–Use Goods and Technologies*, Secretariat of The Wassenaar Arrangement, Vienna, Austria, décembre 1995.

[Weissman69] C. Weissman, Security Controls in the ADEPT–50 Time Sharing System, dans *Proceedings of AFIPS Fall Joint Computer Conference*, volume 35, pages 119–135, Arlington, Virginia, États–Unis, 1969, AFIPS Press.

[Wilkinson95] T. Wilkinson et K. Murray, Extensible, flexible and secure services in Angel, a single address space operating system, dans *Proceedings of the 1st International Conference on Parallel Architectures and Algorithms*, avril 1995.

[Wobber94] E. Wobber, M. Abadi, M. Burrows, et B. Lampson, Authentication in the Taos Operating System, *ACM Transactions on Computer Systems*, 12, no. 1 :3–32, février 1994.

[Wulf81] W. Wulf, R. Levin, et S. P. Harbinson, *HYDRA/C.mmp : An Experimental Computer System*, McGraw–Hill, 1981.

[Zhou91] S. Zhou, X. Zheng, J. Wang, et P. Delisle, Utopia : A load sharing system for large, heterogeneous distributed computer systems, CSRI Technical Report 257, University of Toronto, 1991.

[Zimmermann95] P. R. Zimmermann, *The Official PGP User's Guide*, MIT Press, 1995.

# Résumé

Un système à mémoire virtuelle répartie unique permet l'utilisation des adresses virtuelles comme identificateurs globaux uniques. Il faut donc séparer la résolution des adresses et le contrôle d'accès, parce qu'une adresse virtuelle est potentiellement visible par toute application dans le système.

Nous proposons un modèle de contrôle d'accès pour une mémoire virtuelle répartie unique qui se base sur le modèle à capacités cachées. Le modèle se base sur les notions suivantes : la capacité (un droit d'accès simple), le domaine de protection (définit le contexte de protection par l'ensemble des capacités disponibles dans le domaine) et l'appel de changement de domaine (qui permet d'appeler une procédure désignée dans un autre domaine de protection). Deux principes de base sont très importants pour le modèle : l'utilisation des capacités confinées et la délégation contrôlée à travers des interfaces de protection. Les interfaces de protection permettent une séparation entre la spécification de la protection et le code de l'application. L'évaluation de notre modèle indique qu'il permet de réaliser la plupart des modèles de contrôle d'accès existants y compris le modèle mandataire de Bell & LaPadula.

Le modèle à capacités cachées a été réalisé dans Arias, une mémoire virtuelle répartie unique conçue et développée au sein du projet SIRAC. Les expériences avec cette réalisation montrent que la séparation entre la spécification de la protection et le code de l'application facilite la réutilisation logicielle et l'évolution de l'application. L'efficacité d'un appel de changement de domaine correspond à celui d'un appel RPC standard.



# Abstract

Single Address Space Operating Systems allow virtual addresses to be used by applications as globally unique references. This means that address resolution and access control have to be separated because virtual addresses can be referenced by all applications on all nodes in the system.

We propose an access control model for a Single Address Space Operating System based on Hidden Software Capabilities. The model is based on the notions of capabilities (elemental access rights), protection domains (containers for access rights) and cross domain calls (allow the flow of control to change from one protection domain to another). The model builds on two important principles : segregated capabilities and the controled delegation of capabilities through protection interfaces. Application code and protection interfaces are defined separately. The evaluation of this model indicates that the majority of existing access control models, including the mandatory model proposed by Bell & LaPadula, can be based on Hidden Software Capabilities.

The Hidden Software Capability model has been implemented in the Arias Single Address Space Operating System. Experience with this implementation, shows that separation of protection definition and application code facilitates software reuse and evolution of applications. The performance of the implemented cross domain call is roughly equivalent to a standard RPC.