N^o d'ordre : 1526

THÈSE

présentée

DEVANT L'UNIVERSITÉ DE RENNES 1

pour obtenir

le grade de : ***DOCTEUR DE L'UNIVERSITÉ DE RENNES 1***

Mention : Informatique

par

José KAHAN OBLATT

Équipe d'accueil : CCETT TMM/TDA

École Doctorale : Informatique, Traitement du Signal et Télécommunications

Composante universitaire : IFSIC / IRISA

Conception et expérimentation d'un modèle de contrôle d'accès non nominatif pour les systèmes hypermédia répartis

SOUTENUE le 19 Décembre 1997 devant la commission d'Examen

COMPOSITION DU JURY :

MM. :	Jean-Pierre BANÂTRE	Président
	Daniel HAGIMONT	Rapporteurs
	Marc NANARD	
	Kadi BOUATOUCH	Examinateurs
	Joseph FROMONT	

A la mémoire de mon Père, de Penelope et de Trisha

Remerciements

Ce travail de thèse a été réalisé au sein de l'équipe Télématique Multimédia/Traitement des Documents Audiovisuels (TMM/TDA au Centre Commun d'Etudes en Télédiffusion et Télécommunications (CCETT). Je remercie sincèrement :

- Marc Nanard, Professeur au CNAM, Paris et responsable de l'équipe Interaction Homme-Machine au LIRMM, et Daniel Hagimont, Chargé de Recherche INRIA à l'INRIA Rhône-Alpes, qui ont accepté la charge d'être rapporteurs.
- Jean-Pierre Banâtre, Professeur à l'Université de Rennes 1, pour l'honneur qu'il me fait d'avoir accepté de faire partie du jury.
- Kadi Bouatouch, Professeur à l'Université de Rennes 1, pour son support tout au long de cette thèse. Votre patience et bonne volonté ont su m'aider à surmonter tout obstacle survenu et m'ont permis de mener à bien ces travaux.
- Joseph Fromont, Ingénieur Consultant à France Telecom, pour m'avoir encadré pendant mon séjour au CCETT, me donnant l'opportunité de découvrir, d'explorer et de contribuer au monde du multimédia et de l'hypermédia. Je remercie également Pierre Rolin, ancien directeur de l'ENST-Bretagne, Rennes, pour avoir orienté mes travaux vers le domaine de l'Internet et du World-Wide Web.
- Muriel Jourdan, Laurent Tardif et Jérôme Euzenat pour leur relecture attentive de ce manuscrit.

Je tiens également à remercier tous ceux qui m'ont aidé ou soutenu tout au long de cette thèse avec une mention particulière à tous les membres de l'équipe OPERA à l'INRIA Rhône-Alpes. Je remercie également Vincent Quint et Irène Vatton pour leur patience et intérêt dans ces travaux et pour l'opportunité de me permettre de travailler au sein du Consortium World-Wide Web (W3C).

Je remercie André-Luc Beylot, Bruno Marmol, Hector Ruiz et Jesus Vázquez pour les nombreuses discussions au cours de ma recherche ainsi qu'aux membres de l'association Thésards++ du CCETT pour leur camaraderie.

Il m'est particulièrement agréable de remercier les membres du service informatique et du service bibliothèque au CCETT et à l'IRISA pour leur compétence et support. En particulier, je voudrais remercier Françoise Coudray pour son professionnalisme.

Un grand merci a V. Khi, Bruno, Christophe, Damien, Erich, Florent, Gilles, J.P., Lolo, Lydia, Marie-Noëlle, Michelle, Mickaël, Petko, Roger, Stefan, Yamina et Zazie pour avoir partagé des moments faciles et difficiles pendant toutes ces années ainsi qu'à la bande de la cité Beaulieu.

Je remercie finalement au *Consejo Nacional de Ciencia y Tecnología* (CONACyT) pour son soutien financier.

Table des matières

Liste de Figures

Liste de Tableaux

Chapitre 1

Introduction générale

Une sentinelle se tient postée devant la Loi ; un homme vient un jour la trouver et lui demande la permission de pénétrer. Mais la sentinelle lui dit qu'elle ne peut pas le laisser entrer en ce moment.

Franz Kafka
(*Le Procès*)

1.1 Motivations

Le début de cette décennie a été marqué par le développement des systèmes hypermédia répartis fonctionnant sur le réseau Internet. Le concept d'hypermédia est ambitieux : la mise en relation de données appartenant à des types différents (texte, vidéo, son, etc.) et pouvant être stockées sur différents serveurs.

Dans un système hypermédia, l'information est structurée dans des *graphes*, où les *noeuds* correspondent à des grains d'information et les arcs ou *liens* représentent des relations entre les noeuds. Deux ou plusieurs graphes peuvent partager un même sous-ensemble de noeuds.

Dans une session, on peut distinguer deux niveaux de consultation : un niveau *global* où un utilisateur consulte un graphe et un niveau *local* où l'utilisateur récupère auprès des serveurs du système les noeuds et les liens qui constituent le graphe.

Le contrôle d'accès dans un système hypermédia réparti implique la déclaration et l'évaluation des droits d'accès des utilisateurs sur l'information disponible dans le système. La structuration en graphes de l'information ainsi que sa répartition sur plusieurs serveurs posent de nouveaux besoins de contrôle d'accès :

Gestion cohérente des utilisateurs. La gestion des utilisateurs comprend la spécification, la mise à jour et la révocation des droits d'accès des utilisateurs sur une ressource. Dans un système hypermédia, la gestion des utilisateurs doit être précisée en termes de graphes. En effet, le niveau de consultation globale d'un graphe implique qu'un utilisateur doit pouvoir accéder à tous les noeuds et les liens appartenant à un graphe. Ainsi, les opérations de gestion doivent s'appliquer d'une manière cohérente sur toutes les composantes du graphe.

Organisation des droits d'accès. Une caractéristique du modèle de données hypermédia est l'inclusion d'un noeud dans plusieurs graphes. Un utilisateur peut avoir différents droits d'accès sur un noeud selon le graphe qu'il consulte. Il est important alors de différencier les droits d'accès des utilisateurs sur un noeud par rapport aux graphes où ils ont été définis.

Aujourd'hui on trouve plusieurs solutions au contrôle d'accès dans les systèmes hypermédia. On peut étudier ces solutions par rapport à la spécification des droits d'accès des utilisateurs et par rapport à la localisation de l'information de contrôle d'accès dans le système.

Dans ces solutions, on spécifie les droits d'accès des utilisateurs au moyen de deux types d'information de contrôle d'accès : les *listes de contrôle d'accès* (ACL) et les *listes de groupes d'utilisateurs*. Une ACL spécifie les droits d'accès des utilisateurs ou de groupes d'utilisateurs sur un objet (un noeud ou un lien). Une liste de groupes indique l'appartenance d'un utilisateur à un groupe.

La localisation de l'information de contrôle d'accès comprend l'emplacement de cette information par rapport aux serveurs du système. On distingue trois familles d'approches pour la localisation de l'information de contrôle d'accès dans un système hypermédia :

Approches décentralisées. Dans cette famille, chaque serveur d'objets stocke une copie des informations de contrôle d'accès associées aux objets contenus dans le serveur.

Approches centralisées. On stocke ici toute l'information de contrôle d'accès du système dans un *serveur d'autorisation*. Le serveur d'autorisation doit être consulté pour autoriser toute requête des utilisateurs.

Approches réparties. Dans cette famille, un *serveur de privilèges* contient l'information de contrôle d'accès commune à tous les serveurs (e.g., des listes de groupes d'utilisateurs) ; les serveurs d'objets contiennent l'information de contrôle d'accès qui est spécifique à chaque objet (e.g., des listes de contrôle d'accès). Au début d'une session, un utilisateur se procure auprès du serveur de privilèges une capacité[1] contenant l'information de contrôle d'accès qui lui est associée. Par la suite, l'utilisateur joint cette capacité à ses requêtes. Un serveur d'objets autorise une requêtes en évaluant la capacité ainsi que ses propres informations de contrôle d'accès.

Les solutions existantes présentent des contraintes au niveau de la spécification des droits d'accès et au niveau de l'efficacité.

1. Il s'agit d'un jeton utilisé pour l'échange de l'information de contrôle d'accès.

L'utilisation des ACL classiques s'avère insuffisante pour prendre en compte le partage des noeuds entre graphes. En effet, bien que la structure d'une ACL permet de spécifier les droits d'accès d'un utilisateur sur un objet, elle ne permet pas d'organiser ces droits par rapport aux graphes où ils sont déclarés.

On définit l'efficacité comme le nombre de serveurs devant être contactés pour effectuer une opération. Moins important sera ce nombre, plus efficace sera l'opération. Dans toutes les approches existantes pour la localisation de l'information de contrôle d'accès, il existe un choix entre *l'efficacité de la gestion des utilisateurs* et *l'efficacité de l'autorisation.* Nous mesurons l'efficacité de la gestion des utilisateurs par le nombre de serveurs d'objets impliqués par une opération de gestion sur un graphe. Similairement, nous mesurons l'efficacité de l'autorisation par le nombre de serveurs intervenant lors de l'autorisation d'une requête.

Les approches centralisées présentent une gestion des utilisateurs simplifiée car celle-ci ne concerne qu'un seul serveur. En revanche, ces approches ont une efficacité d'autorisation potentiellement réduite car toute requête pour un objet protégé implique une requête supplémentaire au serveur d'autorisation.

Les approches décentralisées ont une très bonne efficacité d'autorisation car un serveur d'objets peut autoriser un accès sans besoin de contacter aucun autre serveur. Cependant, la gestion des utilisateurs est complexe pour des graphes éclatés sur plusieurs serveurs : il faut synchroniser l'information de contrôle d'accès de tous les objets appartenant à un graphe.

Les approches réparties présentent un bon compromis d'efficacité. Nous trouvons une bonne efficacité d'autorisation car, exception faite d'une requête initiale au serveur de privilèges, l'autorisation des requêtes est effectuée indépendamment par chaque serveur d'objets. Ces approches permettent de simplifier partiellement la gestion des utilisateurs lorsque celle-ci ne concerne que l'information de contrôle d'accès hébergée dans le serveur de privilèges. En contrepartie, la gestion des utilisateurs reste toujours lourde lorsqu'elle implique également les serveurs d'objets.

1.2 Objectifs de la thèse

Notre but principal dans cette thèse est de définir un modèle de contrôle d'accès permettant de :

- prendre en compte le partage de noeuds entre graphes,
- améliorer le compromis d'efficacité présente dans les approches réparties.

L'idée maîtresse est d'utiliser deux propriétés inhérentes aux graphes hypermédia – l'appartenance d'un noeud à un graphe et les liens réliant un noeud aux autres noeuds– pour associer les objets avec des droits d'accès non nominatifs, dépendants de la structure des graphes. De plus, nous centralisons toute l'information nominative dans un serveur d'autorisation.

Au début d'une session de consultation, le serveur d'autorisation autorise globalement l'utilisateur à consulter un graphe et lui expédie une capacité lui donnant accès aux objets du système. Ensuite, les serveurs autorisent localement les requêtes de l'utilisateur en fonction de l'information contenue dans la capacité et dans le serveur.

Le modèle proposé suit deux approches qui se différencient par le nombre de capacités dont un client a besoin lors d'une session de consultation :

– Dans l'approche *une capacité par noeud*, un utilisateur a besoin d'une capacité différente pour accéder à chaque objet d'un graphe. Lors du parcours du graphe, l'utilisateur récupère des nouvelles capacités lui permettant d'accéder aux autres composantes du graphe.

– Dans l'approche *une capacité par graphe*, un utilisateur a besoin d'une seule capacité pour accéder à tout l'ensemble des composantes d'un graphe.

Les deux approches proposées sont indépendantes l'une de l'autre. Toutefois, il est possible de les combiner lors de la protection d'un graphe.

Afin de valider la faisibilité du modèle, nous avons construit CAMW3, un prototype du modèle réalisé sur le système hypermédia World-Wide Web. Le prototype intègre les deux approches du modèle et permet de mieux illustrer les propriétés, les avantages et les limitations du modèle proposé.

Nous signalons qu'il y a d'autres aspects qui relèvent *a priori* de la protection dans les systèmes hypermédia répartis. Par exemple, d'après la nature des données échangées et le niveau de confiance des réseaux interconnectant un client et un serveur, il peut être nécessaire de protéger la confidentialité et l'intégrité des données échangées. Un autre aspect concerne la protection de l'information contre sa duplication et son utilisation non autorisée des données hypermédia par un utilisateur. Nous ne traitons pas ces derniers aspects dans cette thèse. Le lecteur intéressé peut consulter [RS96b, FKK96, Cai96] pour la protection des communications et [Int94] pour la protection des données récupérées par un utilisateur.

1.3 Plan de la thèse

La suite de ce document est organisée en deux grandes parties.

La **première partie** présente une analyse des besoins pour le contrôle d'accès dans les systèmes hypermédia répartis et les principales solutions existantes à ces problèmes. L'intérêt de cette première partie est double : elle permet de familiariser le lecteur avec les systèmes hypermédia répartis et les problèmes posés pour le contrôle d'accès ; par ailleurs, elle décrit les compromis pris par les approches existantes pour la gestion des utilisateurs ainsi que pour la répartition des fonctions de contrôle d'accès. Afin de mieux pouvoir isoler la problématique spécifique au contrôle d'accès dans les systèmes hypermédia, nous avons divisé l'analyse en trois chapitres :

Chapitre 2. Ce chapitre présente d'une manière générale les concepts de base et technologies de contrôle d'accès hérités par les systèmes de protection hypermédia.

Chapitre 3. Dans ce chapitre, on présente les aspects généraux des systèmes hypermédia répartis que nous jugeons importants pour la compréhension de la problématique : la représentation des données, la navigation de l'espace d'information, les protocoles de consultation et la migration des objets.

Chapitre 4. Ce dernier chapitre décrit les principaux problèmes liés au contrôle d'accès dans l'hypermédia et effectue une critique des principales solutions existantes.

La **deuxième partie** propose un nouveau modèle d'accès non nominatif adapté à la consultation des systèmes hypermédia répartis. Cette partie est divisée en deux chapitres :

Chapitre 5. Ce chapitre présente tout d'abord les objectifs du modèle et les suppositions faites sur l'environnement. Il décrit ensuite les deux approches constituant le modèle proposé : l'approche *une capacité par objet* et l'approche *une capacité par graphe*. Le chapitre finit par une comparaison du modèle proposé avec les travaux apparentés.

Chapitre 6. Dans ce chapitre, on décrit la réalisation et l'évaluation de CAMW3, un prototype mettant en oeuvre le modèle proposé sur le système hypermédia réparti World-Wide Web.

Enfin, nous concluons ce document par un bilan du travail réalisé et les perspectives ouvertes par ce travail.

Le lecteur trouvera dans l'annexe A une liste des abreviations employées dans ce document.

Première partie

Contrôle d'accès dans les systèmes hypérmédia répartis

Cette première partie offre un panorama des principales approches pour le contrôle d'accès dans les systèmes hypermédia répartis, et met en évidence les problèmes posés par leur réalisation. Avant de traiter les spécifités propres au contrôle d'accès dans les systèmes hypermédia réparti, nous présentons les techniques empruntées aux systèmes répartis traditionnels par les approches étudiées. Nous étudions ensuite les caractéristiques générales des systèmes hypermédia, en soulevant les principaux problèmes de contrôle d'accès inhérentes à ces systèmes. Enfin, nous décrivons les différentes approches, en analysant leur comportement vis à vis des problèmes soulevés.

Chapitre 2

Le contrôle d'accès dans les systèmes répartis

La sentinelle s'efface devant la porte, ouverte comme toujours, et l'homme se penche pour regarder à l'intérieur. La sentinelle, le voyant faire, rit et dit : « Si tu en as tant envie essaie donc d'entrer « malgré ma défense. Mais dis toi bien que je suis puissant. . . . »

Franz KAFKA
(*Le Procès*)

Dans ce chapitre, nous présentons les caractéristiques du contrôle d'accès dans les systèmes répartis que l'on retrouve dans les approches pour le contrôle d'accès dans les systèmes hypermédia. Nous introduisons tout d'abord les concepts de base permettant d'identifier les éléments intervenant dans le contrôle d'accès. Nous présentons ensuite quelques mécanismes de sécurité utilisés pour la mise en oeuvre du contrôle d'accès, à savoir les mécanismes d'*authentification* et d'*autorisation*.

2.1 Concepts de base

Nous présentons dans un premier temps les concepts généraux du contrôle d'accès. Nous examinons dans un deuxième temps l'application de ces concepts aux systèmes répartis.

2.1.1 Définition et classes de contrôle d'accès

Un système informatique comporte plusieurs types de ressources, tels que des données, des applications, des services de stockage, des stations de travail etc. Le contrôle d'accès vise la protection d'un système et de ses ressources contre deux types de menaces [Bai91]. D'une part, il s'agit de protéger les ressources contre leur utilisation par des utilisateurs non autorisés. D'autre part, il s'agit de protéger le système contre l'exécution des opérations incontrôlées par des utilisateurs ayant accès à des ressources fondamentales du système.

Les contrôles d'accès se classent en deux catégories [Des91] :

- Les contrôles d'accès *administratifs et physiques*, mis en oeuvre à l'extérieur du système informatique. Ils appartiennent à ce que l'on appelle généralement la sécurité du personnel, la sécurité physique et la sécurité procédurale. Ils sont composés d'un ensemble de moyens et de règles qui limitent l'accès physique aux ressources. Ces contrôles ne sont pas abordés dans cette étude.

- Les contrôles d'accès *logiques*, mis en oeuvre à l'intérieur du système informatique. Ils concernent la protection des données et des ressources au moyen des mesures logicielles ou matérielles. Ce chapitre traite de ce type de contrôle.

Afin de mieux étudier les contrôles d'accès, il est conseillé de traiter indépendemment les concepts de politique et de mécanismes [Den82]. Une *politique de sécurité* spécifie les accès autorisés dans un système. Les *mécanismes de sécurité* réalisent la politique. Cette séparation comporte trois avantages. D'abord, on simplifie la comparaison de différentes politiques de sécurité. Ensuite, les mécanismes de sécurité peuvent évoluer sans avoir besoin de réformuler la politique. Enfin, on peut concevoir des mécanismes de sécurité pouvant être utilisés dans plusieurs politiques.

Nous allons maintenant présenter plus en detail les concepts de politique et de mécanisme de sécurité.

2.1.2 Politiques et modèles de sécurité

Une *politique de sécurité* est un ensemble de règles indicant d'abord sous quelles conditions une entité (un utilisateur ou un processus) peut accéder à une ressource du système et ensuite, quelles opérations cette entité peut appliquer sur la ressource (e.g., la lecture, l'écriture, la destruction, l'exécution, etc.) [Gas88]. Généralement, une politique de sécurité s'appliquant à la protection des données d'un système couvre un ou une combinaison des aspects suivants [And73] :

- *la confidentialité* : il s'agit de la protection des données contre leur divulgation ;

- *l'intégrité* : il s'agit de la protection des données contre les modifications accidentelles ou malignes ;

- *la disponibilité du service* : c'est assurer que les utilisateurs légitimes du système puissent accéder à l'information malgré la présence des attaques au système.

L'importance de chacun de ces aspects varie selon le système. Le contrôle d'accès concerne uniquement la confidentialité et l'intégrité des données [ECM88].

Un *modèle de sécurité* est un énoncé particulier des règles de sécurité qui constituent une politique de sécurité [RMV94]. Un modèle doit être précis, simple et facile à comprendre. De plus, le modèle ne doit pas inclure des aspects systèmes ne relevant pas de la politique de sécurité. Ceci permet de mieux adapter un modèle existant à plusieurs systèmes.

On peut spécifier un modèle de manière formelle ou informelle. Une spécification informelle a l'avantage de faciliter la compréhension du modèle. Une spécification formelle à l'avantage d'être non ambiguë. De plus, elle simplifie la vérification que les spécifications d'un système sont conformes au modèle de sécurité.

Dans la section suivante, nous décrivons le modèle de contrôle d'accès de Lampson [Lam74, LABW92]. Ce modèle fournit une infrastructure pour la description des systèmes de protection. Le lecteur pourra trouver une description plus détaillée d'autres modèles de sécurité dans [Lan81, RMV94].

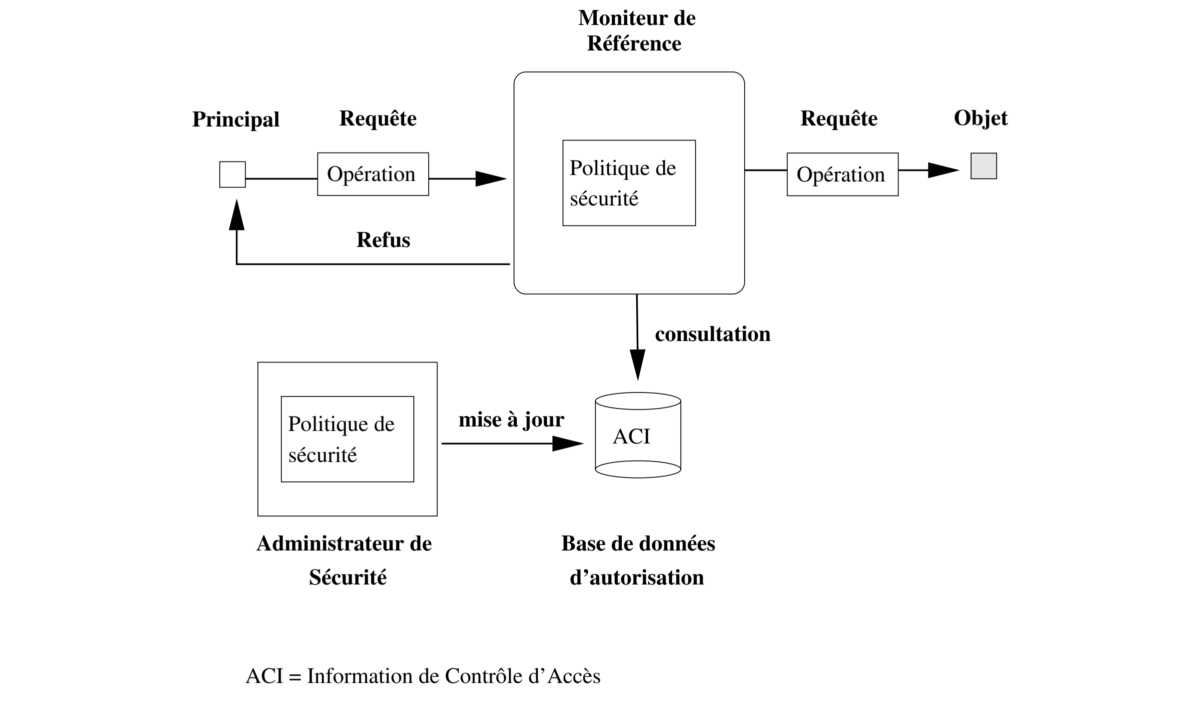


Fig 2.1: Modèle de contrôle d'accès de Lampson

2.1.3 Le modèle de contrôle d'accès de Lampson

Le modèle de Lampson [Lam74, LABW92] établi les bases pour la description des politiques de sécurité traitant de la confidentialité et de l'intégrité des données. Le modèle identifie les éléments intervenant dans un système de protection. De plus, il propose l'utilisation d'une matrice de contrôle d'accès pour la représentation de l'état de sécurité d'un système. Le modèle de Lampson a été ultérieurement raffiné et étendu par Graham et Denning [GD72] et par Harrison *et al* [HRU76].

Nous allons décrire maintenant les différents éléments du modèle et l'utilisation de la matrice de contrôle d'accès. Le lecteur pourra trouver dans [HRU76] une spécification formelle de ce modèle.

2.1.3.1 Eléments du modèle

Lampson caractérise un système de protection comme un ensemble de processus échangeant des messages entre eux. La figure 2.1 illustre les éléments du modèle de Lampson. Nous les présentons brièvement par la suite.

On appelle *objet* n'importe quelle ressource du système, définie par un nom et étant associé à un ensemble d'*opérations*.

On appelle *principal* une entité active qui demande l'exécution d'une opération sur un objet. Il peut s'agir d'un utilisateur, d'un processus client agissant en nom d'un utilisateur ou d'un processus système [Won89, ABLP92]. On associe à chaque principal un nom unique. Dans la suite, nous assimilons les termes *utilisateur* et *client* à celui de *principal* lorsque cela permettra de simplifier notre discours.

Le modèle de contrôle d'accès spécifie que tous les objets du système sont protégés par des gardes, formellement appelés *moniteurs de référence*. Il est impossible pour un principal d'accéder à un objet autrement que par l'intermédiaire du moniteur de référence associé à l'objet. Un moniteur peut contrôler l'accès à un ou plusieurs objets. Par contre, chaque objet du système est contrôlé par un seul moniteur [Won89].

Afin d'évaluer si un principal peut ou non accéder à un objet, le moniteur de référence a besoin de plusieurs types d'*informations de contrôle d'accès* (ACI) [ISO92a]; par exemple, des privilèges de sécurité représentant les droits d'accès du principal, des informations permettant d'authentifier les principaux[1], etc. On regroupe toutes ces informations dans des *bases de données d'autorisation*. Une base est gérée par un *administrateur de sécurité* selon les règles de la politique de sécurité du système.

objet / *principal*	Application_1	Fichier_1	Fichier_2
Alice	Lecture Exécution	Lecture	Ecriture
Bob		Lecture	Lecture
Trudy	Lecture		
Etudiants		Lecture	

Fig 2.2: Matrice de contrôle d'accès.

1. Il s'agit essentiellement de la vérification de l'identité des principaux. Nous traiterons le concept de l'authentification dans la section 2.2.

2.1.3.2 Représentation de l'état de sécurité d'un système

Le modèle de Lampson propose d'utiliser une *matrice de contrôle d'accès* pour représenter l'état de sécurité d'un système. Dans une telle matrice, les lignes correspondent aux principaux du système, les colonnes aux objets et les éléments aux droits d'accès des principaux sur les objets (fig. 2.2).

La valeur de la matrice d'un instant donné représente l'état de sécurité du système. Ainsi, toute modification du contenu ou des dimensions de la matrice correspond à une évolution de l'état de sécurité du système [JM91]. La politique de sécurité du système définit les états autorisés parmi l'ensemble des états possibles de la matrice ainsi que les règles permettant de passer d'un état autorisé à un autre.

Nous allons maintenant interpréter le modèle de Lampson dans le contexte des systèmes répartis.

2.1.4 Interprétation du modèle de Lampson dans les systèmes répartis

Un système réparti est constitué par un ensemble de sites indépendants interconnectés par un réseau informatique [WL92]. Ces systèmes sont souvent structurés d'après un modèle client/serveur en un groupe de processus coopérants appelés *serveurs*, qui offrent des services aux utilisateurs, représentés par des programmes *clients* [Tan92].

Le modèle client/serveur s'appuie sur un protocole simple d'interaction de type *requête/réponse*. Le client envoie une requête au serveur pour accéder à une ressource (par exemple, lire un fichier). Le serveur renvoie le résultat de la requête ou, en cas d'erreur, un code indiquant pourquoi le service n'a pas pu être rendu.

Dans les systèmes répartis, on étend le modèle de Lampson pour inclure les notions de domaine de sécurité et de distribution des éléments.

2.1.4.1 La notion de domaine de sécurité

Un système réparti peut comporter un grand nombre d'utilisateurs et de ressources. Afin de simplifier la gestion de la sécurité de ces systèmes, on les divise usuellement dans des *domaines de sécurité*[2]. Un domaine est un groupe de principaux et d'objets soumis à une politique de sécurité unique et administrés par une autorité unique [ECM89].

Dans certains systèmes répartis, tels que Sesame [PP95] et OSF/DCE [HU95], un principal peut accéder à des objets se trouvant dans un autre domaine. On parle alors d'un *contrôle d'accès inter-domaine.* [ECM89, HU95]. Dans ce document, nous ne nous intéressons qu'au contrôle d'accès intra-domaine.

2.1.4.2 Identification des éléments

Dans un modèle client/serveur, on assimile les principaux au clients et les objets aux ressources gérés par les serveurs [Won89]. Le moniteur de référence et la base de

2. Nous utiliserons le terme *domaine* pour *domaine de sécurité.*

données d'autorisation peuvent être des composantes internes au serveur ou se trouver à l'extérieur de celui-ci [Won89]. Dans la suite, nous analysons les configurations que l'on trouve le plus souvent dans les approches pour la protection des systèmes hypermédias répartis. Le lecteur pourra trouver d'autres analyses de configurations dans [Won89, ISO92a, BGS92, ECM96].

Nous étudions deux groupes de configurations complémentaires. Le premier groupe concerne la répartition des bases de données d'autorisation entre un serveur d'objets et un *serveur de privilèges* (PAS) [JM91]. Le deuxième groupe concerne la répartition des fonctions des moniteurs de référence entre les serveurs d'objets et un *serveur d'autorisation* [ISO92a].

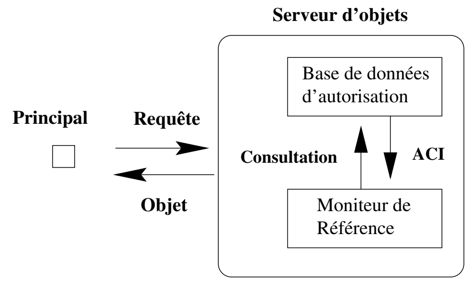


Fig 2.3: Consultation d'une base de données d'autorisation décentralisée

Bases de données d'autorisation décentralisées

Dans cette configuration, illustrée dans la figure 2.3, chaque serveur d'objets dans le système est constitué d'un ensemble d'objets, d'un moniteur de référence et d'une base de données d'autorisation précisant les droits d'accès des principaux en termes d'opérations sur les objets. Un serveur peut prendre une décision d'autorisation indépendamment des autres serveurs du système.

Bases de données d'autorisation centralisées

Dans ce type de configuration, un serveur de privilèges (PAS) héberge l'ensemble de bases de données d'autorisation du système. Le PAS est responsable de protéger l'intégrité des informations qu'il contient et de fournir l'information de contrôle d'accès (ACI) associée aux principaux. On distingue les cas des serveurs d'objets *passifs* et *actifs*, selon le type de processus, client ou serveur, qui doit consulter le PAS.

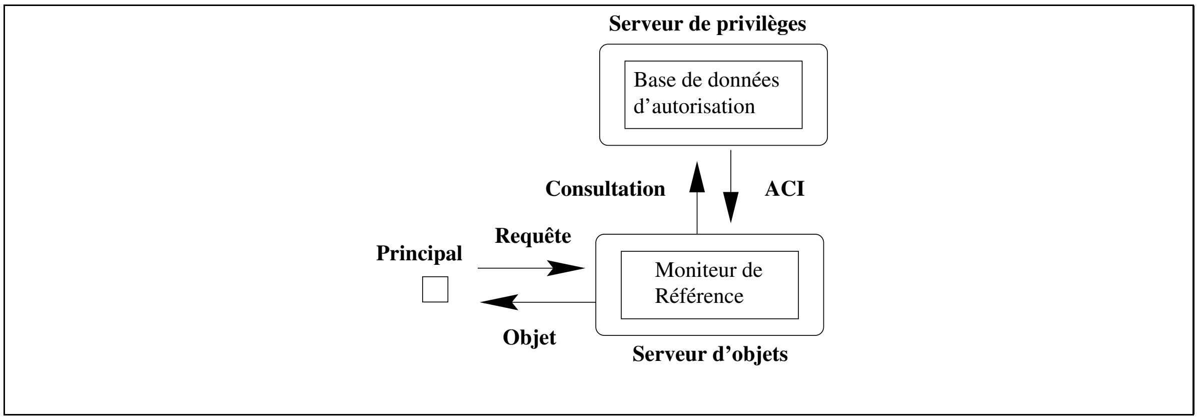


Fig 2.4: Consultation active d'une base de données d'autorisation centralisée

La figure 2.4 illustre une configuration centralisée avec un serveur d'objets actif. Dans cette configuration, lors de la réception d'une requête, le moniteur de référence envoie à son tour une requête au PAS pour connaître les droits du principal.

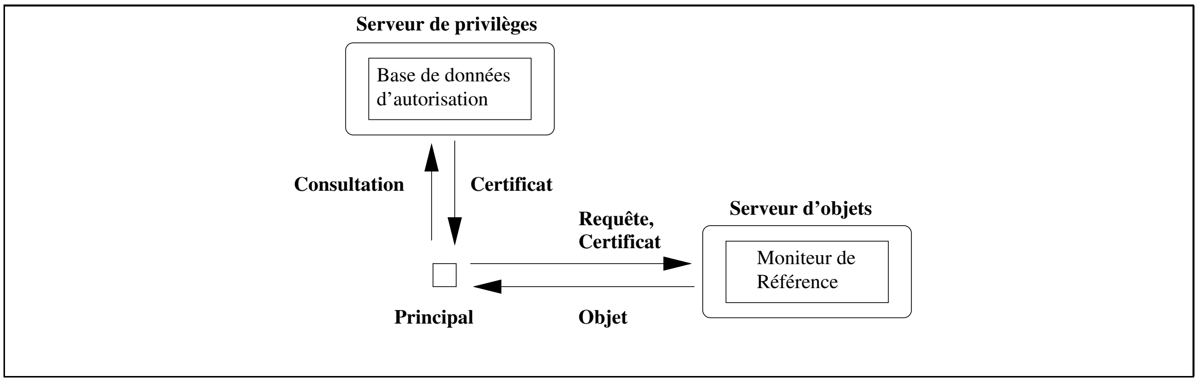


Fig 2.5: Consultation passive d'une base de données d'autorisation centralisée

Dans une configuration centralisée avec un serveur d'objets passif, illustrée dans la figure 2.5, c'est le principal, et non le serveur d'objets, celui qui contacte le PAS. Typiquement, un principal voulant accéder un objet récupère d'abord auprès du PAS un certificat contenant l'ACI nécessaire pour accéder l'objet. Ensuite, le principal inclut le certificat dans la requête qu'il envoie au serveur d'objets.

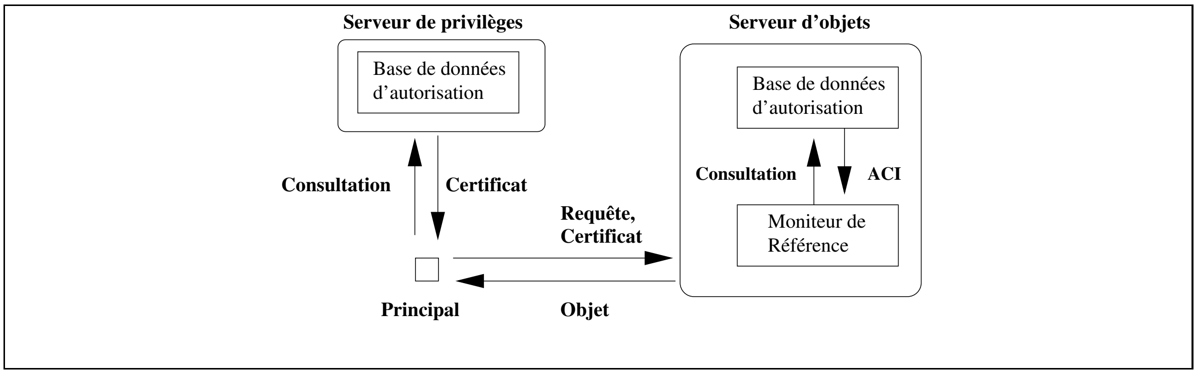


Fig 2.6: Combinaison des configurations de consultation d'une base de données d'autorisation

Bases de données d'autorisation réparties

Cette configuration est issue d'une combinaison des deux configurations précédentes. Dans une configuration répartie, la base de données d'autorisation du PAS contient l'ACI qui est commune à tous les serveurs du système et les bases de données des serveurs contiennent l'ACI qui est spécifique aux objets qu'ils administrent. Les serveurs d'objets peuvent être actifs ou passifs. La figure 2.6 montre ce dernier cas. Un moniteur de référence autorise une requête d'après l'union des données pertinentes contenues dans les deux bases.

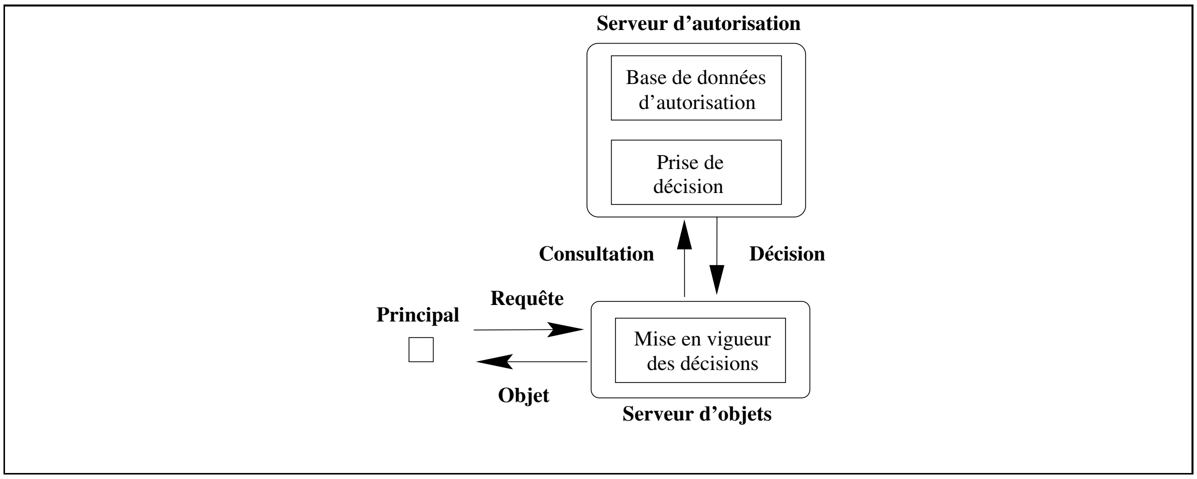


Fig 2.7: Consultation d'un serveur d'autorisation

Répartition du moniteur de référence

Dans cette configuration, les fonctions du moniteur de référence apparentées à l'autorisation sont divisées en un processus d'*évaluation d'autorisation* et un autre de *mise en vigueur de l'autorisation*. Un serveur d'autorisation intègre le premier type de processus. De plus ce serveur héberge les bases de données d'autorisation du système. Les serveurs d'objets incorporent le deuxième type de processus.

La figure 2.7 montre le protocole d'interaction avec le serveur d'autorisation. Lors de la réception d'une requête, le serveur d'objets vérifie tout d'abord l'identité du principal.[3] Le serveur envoie ensuite une requête au serveur d'autorisation lui indiquant l'identité du principal et le type d'accès demandé. Enfin, le serveur d'objets met en vigueur la décision du serveur d'autorisation.

Le concept de serveur d'autorisation ressemble à celui du PAS car les deux serveurs stockent une base de données d'autorisation commune aux moniteurs de référence du système. Toutefois, un PAS se limite à fournir l'ACI qui lui est demandé, tandis qu'un serveur d'autorisation doit fournir le résultat de l'évaluation de l'ACI.

En résumé, du point de vue du contrôle d'accès, un système réparti est divisé en domaines de sécurité. Dans un domaine, on peut assimiler les principaux et les objets aux clients et aux ressources gérés par les serveurs, respectivement. Un serveur d'objets peut incorporer un moniteur de référence et sa base de données d'autorisation, ou bien ces éléments peuvent être extérieurs au serveur. Quelques configurations correspondant à ce dernier cas incluent :

- la centralisation de la base de données d'autorisation dans un serveur de privilèges,
- la répartition de la base de données d'autorisation entre un serveur de privilèges et les serveurs d'objets,
- la répartition des fonctions d'évaluation d'autorisation et mise en vigueur de l'autorisation entre un serveur d'autorisation et les serveurs d'objets, respectivement.

2.1.5 Mécanismes de sécurité

Un mécanisme de sécurité est un ensemble de procédés techniques qui permettent de mettre en oeuvre une politique de sécurité [RMV94]. Les contrôles d'accès logiques reposent essentiellement sur trois classes de mécanismes de sécurité [ECM88] :

- les mécanismes d'*authentification* qui permettent de vérifier l'intégrité et l'origine d'une requête ou d'un message ;
- les mécanismes d'*autorisation,* qui permettent d'évaluer si le principal qui se trouve à l'origine d'une requête peut effectuer l'opération en question ;
- les mécanismes d'*audit* qui ont pour rôle de rapporter chaque action effectuée dans le système. Ce mécanisme permet à l'administrateur de s'assurer de l'application correcte de la politique de sécurité du système et d'en détecter les failles. Ce type de mécanisme n'est pas abordé dans cette étude.

Nous allons présenter maintenant quelques mécanismes d'authentification et d'autorisation qu'on trouve couramment dans les systèmes hypermédia répartis.

3. C'est à dire, il authentifie le principal.

2.2 Mécanismes d'authentification

L'authentification est une procédure permettant de vérifier l'origine et le contenu d'un message [WL92]. Avant de développer le concept d'authentification, nous allons définir les termes que nous utiliserons, à savoir *nonce* et *canal protégé*.

2.2.1 Définitions

On appelle *nonce* une valeur ou numéro qui est utilisé une seul fois [NS78]. Il peut s'agir d'un numéro aléatoire, la date courante ou un des numéros d'une séquence. La notion de nonce se trouve à la base de plusieurs protocoles cryptographiques étudiés dans cette section.

Un *canal protégé* [VK85] est une voie de communication entre deux principaux où on peut garantir une ou plusieurs des propriétés suivantes :

- la confidentialité des données,
- l'intégrité des données,
- la vivacité des données, c'est à dire qu'un intrus n'est pas en train de retransmettre des données datant d'une session précédente.

Un canal protégé peut être du type *physique* ou *logique*. On peut réaliser le premier en protégeant les voies de communication. On peut établir le deuxième type de canal en appliquant une fonction de transformation cryptographique aux données échangées. Dans ce document, nous traitons uniquement les canaux protégés logiques. Par abus de langage, nous les appelons *canaux protégés*.

2.2.2 Concept d'authentification

L'*authentification* est une procédure de vérification. Elle peut être divisée en deux classes [WL92]. L'*authentification d'un principal* est la vérification de l'identité du principal. L'*authentification du contenu des messages* est la vérification de l'intégrité d'un message émis par un principal : s'assurer que son contenu n'a pas été modifié de manière malfaisante.

Un moniteur de référence a besoin d'effectuer plusieurs types d'authentification sur une requête avant de vérifier les droits d'accès de l'émetteur de la requête. Plus spécifiquement, le moniteur de référence doit vérifier :

- (P1) quelle est l'identité du principal qui a émis la requête (authentification du principal émetteur) ;
- (P2) que le contenu de la requête n'a pas été altéré depuis son émission (authentification de l'intégrité de la requête) ;

– (P3) que l'ordre d'émission de la requête est le même que celui de la réception (authentification de l'ordre d'émission) ;

– (P4) que la requête est récente, c'est à dire, qu'il ne s'agit pas du rejoue d'une requête précédente par un intrus (authentification de la vivacité de la requête) ;

– (P5) que l'information de contrôle d'accès émise par une autorité sous forme d'un certificat et présentée par un principal, n'a pas été modifiée par le principal ou par quelqu'un d'autre (authentification de l'intégrité d'un message asynchrone) :

– (P6) que le certificat présenté par un principal a été effectivement générée par une autorité (authentification de l'origine d'un message asynchrone).

Les propriétés (P1) à (P4) sont assimilables à l'authentification d'une session. On peut atteindre ces objectifs au moyen d'un *protocole d'authentification.*

Les propriétés (P5) et (P6) sont relatives à l'authentification des messages échangés en dehors d'une session. L'utilisation des *signatures numériques* permet de prendre en compte cette authentification.

Nous allons maintenant présenter les principes généraux des techniques des signatures numériques et des protocoles d'authentification. Le tableau 2.1 donne les notations que nous utiliserons par la suite. Certaines de ces techniques font appel à des algorithmes cryptographiques. Le lecteur trouvera une introduction à ces algorithmes dans l'annexe B.

Notation	**Signification**
$K_{A,B}$	Une clé secrète partagée par les principaux A et B.
KU_A	La clé publique du principal A.
KR_A	La clé privée du principal A.
$\{M\}_K$	Une transformation cryptographique du message M utilisant une clé K.
$M1$, $M2$	Concaténation de deux messages.
$A \rightarrow B : M$	Le principal A envoie le message M au principal B.
ID_A	L'identité du principal A.
PV	Une période de validité comprenant une date de début et une date de fin de validité.

Tab 2.1: Notation pour les échanges cryptographiques

2.2.3 Signatures numériques

Une signature numérique est l'équivalent d'une signature traditionnelle, mais appliquée à un message électronique. Si l'on note B le récepteur d'un message M signé par

A, les propriétés de la signature de A sont les suivantes [Den82] :

- B doit pouvoir valider la signature de A,
- il doit être impossible à n'importe quel principal, y compris B, de contrefaire la signature de A,
- toute modification de M doit invalider sa signature,
- si A nie avoir signé M, il doit être possible à un juge ou à une tierce partie de résoudre ce conflit de non répudiation.

Les algorithmes de signature numérique utilisent les systèmes à clé publique, les systèmes à clé symétrique, ou une combinaison de ces deux systèmes [Akl83, Sch94]. Parmi ces systèmes, on préfère dans la pratique les algorithmes à clé publique, ceux-ci permettant plus de flexibilité en termes du nombre de participants impliqués dans les opérations de signature d'un message et de validation d'une signature.

Nous examinons maintenant l'algorithme de signature à clé publique de Denning [Den82, pg. 14]. Le lecteur pourra trouver d'autres algorithmes de signature dans les références données ci-dessus.

L'algorithme de signature de Denning

Cette algorithme, comme la plupart des algorithmes de signature fondés sur un système cryptographique à clé publique[4], utilise la propriété associative des fonctions de transformation de ce système cryptographique.

Denning propose un algorithme de signature à clé publique formé de trois étapes asynchrones :

1. A signe M en calculant $\{M\}_{KR_A}$.
2. B vérifie la signature en appliquant $\{\{M\}_{KR_A}\}_{KU_A}$.
3. Un juge résout une dispute en prouvant d'abord que KU_A appartient à A et ensuite que $M = \{\{M\}_{KR_A}\}_{KU_A}$.

Cette algorithme est en apparence très simple. Cependant, il suppose que la clé publique de A est toujours valide. Si cette clé est compromise et si A change de clé, le juge ne pourra plus vérifier les anciens messages signés de A.

Akl [Akl83] propose une solution à ce problème en faisant à une autorité certifier les messages d'A. Un message certifie est composé du message d'A concatené à certaines informations de contrôle, le tout signé sous la clé privée de l'autorité :

$$\{A, M_{KR_A}, \text{Date}\}_{KR_{\text{Autorité}}}$$

4. L'annexe B donne une brève introduction aux techniques et concepts cryptographiques.

Les informations de contrôle comprennent une date et la clé publique d'A qui était valide à cette date. Ainsi, si l'autorité a toujours accès à la clé courante d'A, on peut toujours prouver que A à signé un message à une époque donnée. Notons que l'on retrouve le même problème que nous avons cité ci-dessus si la clé publique de l'autorité est compromise.

Réduction du temps de signature

Un des problèmes liés aux systèmes à clé publique est la lenteur de leur exécution (cf. section B.4). On peut diminuer le temps nécessaire pour signer M en réduisant d'abord le texte du message au moyen d'une fonction de hachage et en signant ensuite la valeur hash de M [Akl83, BD92]. La signature de M correspond alors à la concaténation de M et de la signature du hash. B doit recalculer le hash de M pour vérifier la signature de M. La section B.2 donne une description plus en détail des fonctions de hachage.

En résumé, la technique de signatures numériques permet à n'importe quel principal de vérifier à n'importe quel moment l'authenticité de l'origine et du contenu d'un message. On peut effectuer une signature numérique avec des algorithmes symétriques et avec des algorithmes à clé publique. Avec ces derniers, signer un message est équivalent à chiffrer le message avec une clé privée. Afin de diminuer le temps nécessaire pour signer un message, il est conseillé de signer un hash du message.

2.2.4 Protocoles d'authentification

Ces protocoles permettent d'authentifier une session entre deux principaux au moyen de la présentation et de la vérification des preuves d'identité. Les sections suivantes sont consacrées à la classification des protocoles d'authentification et à la description de quelques protocoles fondés sur des preuves cryptographiques. Nous introduisons tout d'abord la terminologie associée à ces protocoles. Nous présentons ensuite leurs caractéristiques générales. Nous appuyons cette présentation sur quelques exemples représentatifs se trouvant à la base de la protection des systèmes hypermédia répartis.

2.2.4.1 Terminologie

On appele *principal initiateur* le principal qui présente une preuve et *principal vérificateur* le principal qui valide la preuve.

Une *preuve d'identité* est un objet lié à l'identité d'un principal [Sha77, WL92]. Cette preuve peut être (liste non exhaustive) :

- une chose connue par le principal (e.g. un mot de passe, une clé cryptographique, etc.),
- une chose appartenant au principal (e.g. une carte magnétique, etc.),

– une chose caractérisant le principal (e.g. la voix, une adresse réseau, etc.).

On distingue deux étapes dans un protocole d'authentification [NS78] :

– La *distribution des preuves d'identité*. Il s'agit de fournir de l'information aux initiateurs leur permettant de générer une preuve de leur identité. De même, il s'agit de fournir aux vérificateurs des données leur permettant de valider les preuves d'identité. Cette distribution peut être effectuée au préalable de la session de communication ou bien lors de l'établissement de la session.

– L'*authentification d'un principal*. C'est la procédure suivie par un principal pour vérifier la preuve d'identité d'un autre principal.

2.2.4.2 Classification des protocoles d'authentification

Dans les systèmes hypermédia répartis, on distingue trois familles de protocoles d'authentification, d'après la nature de la preuve d'identité employée [Ste95, Cai96] :

Authentification par adresse réseau. Dans cette famille, l'adresse réseau du site du principal est utilisée comme preuve d'identité. Un principal vérificateur doit connaître au préalable l'adresse réseau du principal initiateur. Cette famille de protocole s'avère insuffisante lorsque plusieurs utilisateurs peuvent émettre des messages ayant la même addrèse d'origine : il est impossible alors de les distinguer les uns des autres.

Authentification par mot de passe. Ces protocoles se fondent sur l'échange d'un mot de passe en clair sur le réseau. Le principal vérificateur doit connaître au préalable le mot de passe du principal initiateur. Ce type de protocoles permet l'authentification d'un principal indépendamment de son adresse dans le système.

Authentification par protocoles cryptographiques. Dans cette famille, l'authentification s'effectue par démonstration de connaissance d'une clé cryptographique : deux principaux entament un protocole cryptographique dont la conclusion est l'authentification de l'un ou des deux principaux. Dans la plupart des cas, une exécution réussie du protocole finit par l'assignation d'une clé de session aux deux principaux. Les principaux utilisent ensuite cette clé pour établir un canal protégé de communication. La clé utilisée pour l'authentification peut être distribuée au prealable ou lors de l'établissement de la session. Ce type de protocole peut s'appliquer indépendamment de l'adresse réseau des principaux. De plus, la plupart de ces protocoles fournissent une preuve de la vivacité de l'authentification.

Les techniques d'authentification par mot de passe et par adresse réseau sont encore les plus courantes aujourd'hui, même si elles présentent plusieurs risques de sécurité. Parmi les risques les plus connus, nous pouvons mentionner la falsification des adresses réseau et l'interception des mots de passe qui circulent sur le réseau [GS96, pp. 470-471]. Certains protocoles permettent de diminuer les risques en concatenant d'abord un nonce au mot de passe et ensuite en chiffrant le tout avec une fonction à une seule voie [FHBH+96].

Un autre problème avec l'authentification par mots de passe et par adresse réseau est l'administration des preuves d'identité. En effet, le principal vérificateur doit connaître au préalable les preuves d'identité. Par conséquent, lors de l'arrivée d'un nouveau principal, il faut contacter tous les vérificateurs potentiels afin de mettre à jour leurs informations d'authentification. Dans certains cas on peut réduire le problème de mémorisation des mots de passe, si l'on utilise une base de données globale au système telle que le *Network Information System* de Sun [GS96, pp. 579-593].

La technique d'authentificaton par protocoles cryptographiques est celle qui présente la meilleure résistance aux attaques. Comme nous verrons dans cette section, cette technique peut également réduire le nombre de données que le principal vérificateur nécessite pour valider les preuves d'identité.

famille / *propriété*	Authentification par adresse IP	Authentification mot de passe	Authentification par prot. cryptographiques
Authentification de l'addresse réseau du principal initiateur	oui	non	depend du protocole
Authentification de l'identité du principal initiateur	non	oui	oui
Authentification de l'identité du principal vérificateur	non	non	dépend du protocole
Authentification de l'origine des messages	oui	non	dépend du protocole
Authentification du contenu des messages	non	non	dépend du protocole
Authentification de l'ordre d'émission de la requête	non	non	dépend du protocole
Authentification de la vivacité des échanges	non	non	oui
Distribution des preuves d'identité	au préalable	au préalable	au préalable ou pendant la session

Tab 2.2: Comparaison des principales familles de protocoles d'authentification

Le tableau 2.2 résume quelques unes des propriétés de ces familles de protocoles. Nous renvoyons le lecteur voulant avoir une analyse plus détaillée des protocoles d'authentification et de leurs propriétés aux travaux de Burrows *et al.* [BAN89], Woo et Lam [WL92], Liebl [Lie93], Gollman *et al.* [GBD93] et Carlsen [Car94].

Dans la suite de cette section, nous présentons les caractéristiques générales des protocoles d'authentification à clé symétrique et à clé publique.

2.2.4.3 Protocoles d'authentification à clé symétrique

Dans ces protocoles, un principal preuve son identité à un autre en démontrant sa connaissance d'une clé secrète partagé entre lui et l'autre principal. Dans un premier temps, nous précisons les principes pour la distribution de clés. Dans un deuxième temps, nous présentons le protocole d'authentification Kerberos, utilisé dans plusieurs approches pour la protection dans les systèmes (hypermédia) répartis.

Distribution de clés

La distribution des clés secrètes partagés peut être effectuée au préalable ou lors de l'établissement de la session de consultation.

Dans une distribution préalable, il faut octroyer une clé secrète à chaque paire de principaux pouvant échanger des messages [JM91]. Un inconvénient de cette approche est l'administration des clés secrètes : si un système contient n principaux, et si chaque principal a besoin de parler avec tout autre principal, chaque principal devra connaître $n-1$ clés.

Needham et Schroeder [NS78] proposent une protocole pour la distribution de clés à la demande des principaux. Dans cette approche, un *centre de distribution de clés* (KDC) partage une clé secrète avec tout autre principal.

Lorsqu'un principal veut contacter un autre, il demande dans un premier temps au KDC de générer une clé secrète partagée (appelée clé de session). Le KDC lui envoie au retour deux jetons. Le premier contient la clé de session chiffrée avec la clé partagée entre l'initiateur et le KDC. Le deuxième, appelé *ticket*, contient l'identité de l'initiateur et une copie de la clé de session, le tout chiffré avec la clé partagée entre le vérificateur et le KDC. Enfin, l'initiateur transmet le ticket au vérificateur. Les principaux récupère alors la clé de session et le protocole d'authentification peut alors démarrer.

On trouve le protocole de Needham et Schroeder à la base de plusieurs protocoles d'authentification [Lie93]. Le plus répandu de ceux-ci est Kerberos, que nous décrivons dans la section suivante.

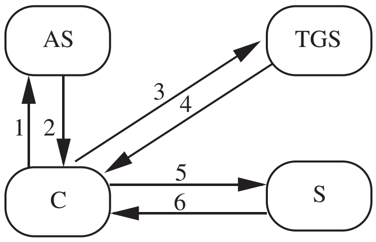
Le protocole d'authentification Kerberos

Kerberos [MNSS87, SNS88, NT94] est un système d'authentification de principaux et de distribution de clés. Il est fondé sur une variation du protocole de Needham et Shroeder qui inclut des messages horodatés [DS81]. Kerberos a suivi plusieurs évolutions. Cette section présente quelques propriétés de Kerberos V5 [KN93], la version la plus récente du système. Nous gardons la terminologie d'origine dans cette description.

Sous Kerberos, un système réparti est divisé en domaines (cf. section 2.1.4.1, page 15). Un *domaine* est constitué d'un ensemble de principaux (clients et serveurs) et d'un KDC. De plus, on suppose que les horloges des systèmes des principaux sont faiblement synchronisés.

Un KDC sous Kerberos comprend un serveur d'authentification (AS) et un serveur expéditeur de tickets (TGS)[5]. L'AS partage une clé secrète avec tous les clients. Le TGS partage une clé secrète avec tous les serveurs.

Le protocole Kerberos est constitué de trois sous-protocoles. La figure 2.8 montre un format simplifié des messages de ces protocoles.



(a) Obtention d'un ticket pour le TGS

1. $C \rightarrow AS : \mathrm{UID}_C\,,\ \mathrm{UID}_{TGS}\,,\ \mathrm{Options}\,,\ \mathrm{Nonce}_1$
2. $AS \rightarrow C : \mathrm{Ticket}_{TGT}\,,\ \{K_{C,TGS}\,,\ \mathrm{UID}_{TGS}\,,\ \mathrm{PV}_1\,,\ \mathrm{Nonce}_1\}_{K_{C,AS}}$

$\mathrm{Ticket}_{TGT} = \mathrm{UID}_S\,,\ \{K_{C,TGS}\,,\ \mathrm{UID}_C\,,\ \mathrm{AR}_C\,,\ \{\text{Privilèges}\}\,,\ \mathrm{PV}_1\}_{K_{AS,TGS}}$

(b) Obtention d'un ticket pour un serveur

3. $C \rightarrow TGS : \mathrm{UID}_C\,,\ \mathrm{UID}_S\,,\ \mathrm{Options}\,,\ \mathrm{Ticket}_{TGT}\,,\ \mathrm{Authentificateur}_C\,,$ Nonce_2
4. $TGS \rightarrow C : \mathrm{Ticket}_S\,,\ \{K_{C,S}\,,\ \mathrm{UID}_S\,,\ \mathrm{PV}_2\,,\ \mathrm{Nonce}_2\}_{K_C,TGS}$

$\mathrm{Ticket}_S = \mathrm{UID}_S\,,\ \{K_{C,S}\,,\ \mathrm{UID}_C\,,\ \mathrm{AR}_C\,,\ \{\text{Privilèges}\}\,,\ \mathrm{PV}_2\}_{K_{S,TGS}}$

$\mathrm{Authentificateur}_C = \{\mathrm{UID}_C\,,\ \mathrm{Date}_1\}_{K_{C,TGS}}$

(c) Authentification auprès d'un serveur

5. $C \rightarrow S : \mathrm{Ticket}_S\,,\ \mathrm{Authentificateur}_C$
6. $S \rightarrow C : \{\mathrm{Date}_2\}_{K_{C,S}}$

$\mathrm{Authentificateur}_C = \{\mathrm{UID}_C\,,\ \mathrm{Date}_2\}_{K_{C,S}}$

Fig 2.8: Famille de protocoles Kerberos

Kerberos définit deux protocoles pour l'acquisition des tickets. Dans une première phase, le client demande un ticket pour le TGS à l'AS. Afin de le différencier d'autres tickets, Kerberos l'appelle *ticket d'allocation de tickets* (TGT)[6]. Cet échange utilise la

5. *Ticket Granting Server.*
6. *Ticket Granting Ticket.*

clé secrète partagée entre l'AS et le client. Par la suite, le client demande des tickets pour d'autres serveurs au TGS. Cet échange utilise le TGT et la clé incluse dans celui-ci. Nous allons maintenant décrire le protocole Kerberos :

1. Dans une première étape, un client C demande à l'AS un ticket pour le TGS. La requête du client comprend le nom du TGS et un nonce. On utilise le nonce pour prouver la vivacité de la réponse de l'AS.

2. L'AS génère deux jetons. Le premier est le TGT qui contient l'identité du serveur, l'identité du client, l'adresse réseau du système hôte du client, un temps de vie, une clé de session et des champs optionnels. Le tout, sauf le premier champ, est chiffré avec la clé partagée entre l'AS et le serveur. Le deuxième jeton contient le nom du serveur et une copie de la clé de session, du temps de vie et du nonce, le tout chiffré avec la clé partagée entre l'AS et le client. L'AS envoie ensuite au client ces deux certificats. Le client vérifie le nonce inclus dans son jeton et stocke le TGT et la clé de session qui lui est associée.

3. Lorsque le client a besoin de récupérer un ticket pour un serveur, il génère un authentificateur avec la clé de session associée au TGT. L'authentificateur inclut l'identité du client et la date courante du système hôte du client, utilisée pour prouver la vivacité de l'authentificateur. Enfin, le client envoie le nom du serveur, l'authentificateur, le TGT et un nonce au TGS.

4. Le TGS utilise la clé de session du TGT pour authentifier le client. Il vérifie ensuite la date de l'authentificateur et l'adresse du système hôte du client. En cas de succès, le TGS génère un ticket pour le serveur et un jeton pour le client. Le jeton est chiffré avec la clé de session incluse dans le TGT. Le client vérifie le nonce inclus dans son jeton et stocke le ticket et la clé de session qui lui est associée.

5. Lorsque le client veut contacter un serveur S, il utilise la clé de session incluse dans le ticket du serveur pour générer un nouvel authentificateur. Le client envoie ensuite le ticket et l'authentificateur au serveur.

6. Le serveur utilise la clé de session du ticket pour authentifier le client. Il vérifie ensuite la date de l'authentificateur. En cas de besoin d'une authentification mutuelle, le serveur transmet au client un message qui contient la date de l'authentificateur chiffrée avec la clé de session. Ceci prouve au client que le serveur connaît la clé de session. Par la suite, le client et le serveur peuvent utiliser cette clé de session pour protéger le reste de leurs échanges.

La structure d'un ticket Kerberos inclut un champ pour le stockage des privilèges de sécurité d'un principal. Il existe deux manières de remplir ce champ. Dans un cas, un client demande à l'AS ou au TGS de générer un ticket, tout en leur fournissant les données que ce champ doit contenir. Dans un autre cas, si le TGS reçoit un TGT contenant des privilèges de sécurité, il les dupliquera dans le ticket demandé par le client. Cette propriété de Kerberos est notamment utilisée dans l'architecture de sécurité OSF/DCE [RKF92, HU95] , dont il sera question dans le chapitre suivant.

Nous pouvons résumer les principales caractéristiques de Kerberos comme suit :

– Un principal est authentifié initialement au moyen d'une clé qu'il partage avec les serveurs Kerberos et ensuite par la présentation d'un ticket et d'un authentificateur. Il est également possible pour un principal d'authentifier un serveur.

– Un client peut réutiliser un ticket jusqu'à ce que celui-ci expire. En revanche, un authentificateur n'est valable que pour une seule utilisation.

– L'exécution du protocole d'authentification permet à deux principaux de partager une clé de session. Cette clé est générée par les serveurs Kerberos. Cette approche suppose qu'un serveur Kerberos est une entité de confiance qui n'abusera pas de son pouvoir pour écouter une conversation entre deux principaux ou pour se faire passer pour un principal.

– Les serveurs Kerberos doivent être en ligne pour fournir des tickets à la demande.

– L'infiltration des serveurs Kerberos peut compromettre la sécurité de tous les principaux appartenant au domaine. En effet, l'intrus pourrait se faire passer pour n'importe quel principal.

– Comme les clients doivent les contacter assez souvent, les serveurs Kerberos représentent un problème potentiel de performances. Ce problème peut être partiellement résolu par la réplication de ces serveurs. Cependant, cette solution complique la protection des serveurs Kerberos.

– Kerberos utilise des horodateurs dans les tickets, pour contrôler leur temps de vie, et dans les authentificateurs, pour prouver leur vivacité. Ceci nécessite une synchronisation faible des horloges des systèmes hôtes d'un domaine. Il existe plusieurs solutions à ce problème. [RKF92] décrit l'utilisation d'un service d'horloge diffusé sur les ondes hertziennes. Lam et Gollman [LG92] décrivent la mise en place d'un service fiable et sécurisé de synchronisation opérant sur des réseaux informatiques non protégés.

En résumé, dans les protocoles d'authentification à clé secrète, l'authentification unilatérale ou mutuelle des principaux s'effectue par démonstration de connaissance d'une clé secrète. Cette clé doit être connue de l'initiateur et du vérificateur. Une distribution préalable des clés secrètes partagées est possible, mais onéreuse car tout principal doit partager une clé avec ses entités homologues. L'utilisation d'un centre de distribution de clés (KDC) permet d'effectuer une distribution sous la demande de l'initiateur et ne nécessite que la distribution préalable d'une clé secrète partagée entre les principaux et le KDC.

2.2.4.4 Protocoles d'authentification à clé publique

Dans ces protocoles, chaque principal a une paire de clés asymétriques (publique et privée). Un principal prouve son identité à un autre en démontrant sa connaissance de sa clé privé. On distingue deux étapes indépendantes dans ces protocoles [NS78, CCI87, KPS95, Sta95] :

- la *distribution de clés publiques*, qui permet à un principal de connaître la clé publique d'un autre principal,
- l'*authentification des principaux*, qui permet à un principal de vérifier qu'un autre possède bien la clé privé qui corresponde à sa clé publique.

Nous allons maintenant décrire brièvement les principes de base de ces deux étapes en donnant quelques exemples représentatifs. Le lecteur trouvera une description plus détaillée de ces concepts ainsi que d'autres protocoles d'authentification à clé publique dans [NS78, CCI87, KPS95, Sta95].

Distribution de clés

Le principal problème pour la distribution des clés publiques est de garantir qu'une clé publique appartient à un principal. La solution adoptée est de distribuer les clés publiques au moyen de *certificats de clé publique*, issus par une *autorité de certification* (CA) [CCI87]. On suppose ici que la CA a une paire de clés asymétriques et que tout principal dans le domaine connaît la clé publique de la CA.

Un certificat de clé publique est un message qui associe l'identité d'un principal (UID-principal) à la clé publique du principal (KU-principal), le tout protégé par la signature de la CA. Tout principal peut donc vérifier la signature d'un certificat et ainsi prouver son authenticité. De plus, comme il peut y avoir plusieurs CA, chaque certificat contient également l'identité de la CA qui l'a généré :

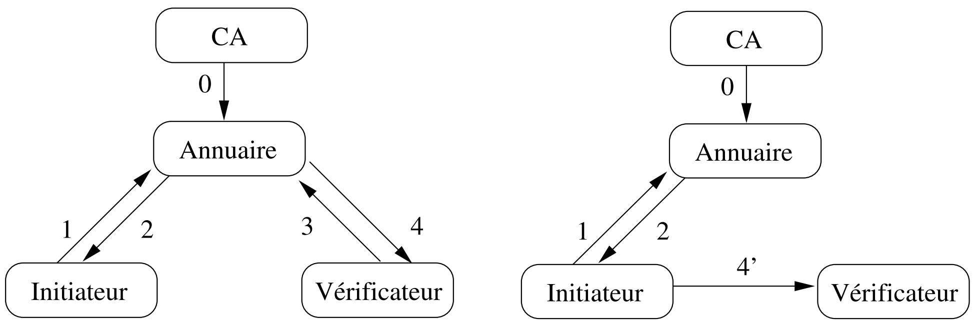
UID-principal	KU-principal	UID-CA	Signature-CA

La CA peut être un service en ligne qu'un principal contacte chaque fois qu'il a besoin de connaître la clé publique d'un autre principal [NS78, DS81]. L'avantage de cette approche est qu'un principal peut toujours obtenir la version la plus récente d'une clé publique. Un autre avantage est que la CA est le seul serveur concerné lors de l'ajout, de la modification et de la suppression des clés. Cependant, comme pour le KDC de Kerberos[7] les inconvénients de cette approche sont liés à la protection et à la disponibilité du serveur CA [KPS95, pp. 189-190].

Une autre solution est d'avoir une CA hors ligne. Il s'agit d'utiliser une CA pour générer à l'avance les certificats de clé publique de tous les principaux du domaine. La distribution des certificats peut alors être effectuée par des services non protégés,

7. cf. section 2.2.4.3.

sans aucune intervention de la CA. Par exemple, on peut utiliser les services d'annuaire [CCI87] ou de DNS [HA94] pour effectuer cette opération. Un principal contactera alors le serveur pour récupérer un certificat. Une autre possibilité est qu'un principal envoie son certificat de clé publique directement à son interlocuteur. La figure 2.9 illustre ces deux exemples, pour un principal initiateur et un principal vérificateur.



(a) Deux interactions avec le service d'annuaire

(b) Une seule interaction avec le service d'annuaire

0. $CA \rightarrow$ Annuaire : Transmission des certificats
1. $I \rightarrow$ Annuaire : UID_V
2. Annuaire $\rightarrow I$: $\{\mathrm{UID}_V\ ,\ KU_V\ ,\ ...\}_{KR_{CA}}$
3. $V \rightarrow$ Annuaire : UID_I
4. Annuaire $\rightarrow V$: $\{\mathrm{UID}_I,,\ KU_I\ ,\ ...\}_{KR_{CA}}$

4′. $I \rightarrow V$: $\{\mathrm{UID}_I\ ,\ KU_I\ ,\ ...\}_{KR_{CA}}$

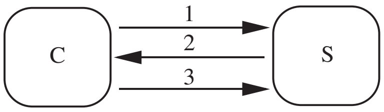
Fig 2.9: Protocoles de distribution de clés publiques avec une CA hors ligne

L'utilisation d'une CA hors ligne présente plusieurs avantages et inconvénients [KPS95, Section 7.7]. Un avantage est qu'il est possible d'isoler la CA du reste du réseau et donc de mieux assurer sa protection. Un autre avantage est que la tolérance aux pannes peut être augmentée sans aucun risque pour la sécurité en répliquant les services d'annuaire.

Le principal inconvénient de cette approche à la distribution est la révocation des certificats. Dans certains cas, le contenu du certificat d'un principal peut évoluer dans le temps et dans d'autres, le certificat lui-même peut être annulé. C'est le cas, par exemple, lorsqu'un principal veut changer sa paire de clés asymétriques (lorsque la clé privée d'un principal est compromise ou lorsqu'un principal ne doit plus être certifié par la CA). Il faut alors indiquer aux principaux vérificateurs le statut des clés publiques de leur entités homologues. Vis à vis de ces problèmes, la recommandation X.509 [CCI87] propose deux approches pour la révocation *implicite* et *explicite* des certificats.

Afin d'obtenir une révocation implicite des certificats, X.509 augmente leur structure avec une *période de validité*. Ainsi, au bout d'un certain temps, les certificats deviendront obsolètes. On suppose ici qu'il existe une synchronisation entre l'horloge de la CA et ceux des vérificateurs.

La révocation explicite d'un certificat permet d'annuler un certificat avant que sa période de validité n'expire. L'approche de X.509 pour la révocation explicite consiste en l'utilisation des numéros de série et des listes de certificats révoqués (CRL). Dans cette approche, chaque certificat inclut un numéro de série unique. Lors de la révocation d'un certificat, la CA inscrit son numéro de série dans une CRL. Dans des intervalles de temps préétablis, la CA signe et date les CRL et les met à la disposition des service d'annuaire. Comme pour les certificats, les CRL ont un champ spécifiant leur période de validité. C'est la responsabilité des vérificateurs de récupérer les versions les plus récentes des CRL



1. $C \rightarrow S : \{\text{Nonce}_1, \text{UID}_I\}_{KU_V}$
2. $S \rightarrow C : \{\text{Nonce}_1, \text{Nonce}_2\}_{KU_I}$
3. $C \rightarrow S : \{\text{Nonce}_2, \{K_{I,V}\}_{KU_V}\}_{KR_I}$

Fig 2.10: Variante du protocole d'authentification à clé publique de Needham et Schroeder

Authentification des principaux

Une fois qu'un initiateur et qu'un vérificateur obtiennent les clés publiques de leurs entités homologues, il est possible de procéder à leur authentification. Needham et Schroeder [NS78] proposent un protocole à clé publique qui permet d'établir une authentification mutuelle des principaux et une clé symétrique de session. La figure 2.10 montre une variante de ce protocole [Sta95, pages 135-136]. Afin de clarifier la description du protocole, nous assimilons les principaux à un client et à un serveur.

1. Le client envoie au serveur un message chiffré avec la clé publique du serveur. Il contient l'identité du client et un nonce, qui permet de prouver la vivacité de cet échange.

2. Le serveur envoie au client un message chiffré avec la clé publique du client. Ce message contient le nonce généré par le client et un nouveau nonce. La présence du premier nonce prouve l'identité du serveur au client.

3. Le client envoie au serveur un message signé avec sa propre clé publique. Il contient une copie du nonce du serveur et une clé de session symétrique générée par le client et chiffrée avec la clé publique du serveur, pour garantir sa confidentialité. L'inclusion du nonce du serveur prouve l'identité du client au serveur.

Par la suite, le client et le serveur peuvent utiliser la clé de session pour protéger leurs échanges.

En résumé, dans les protocoles d'authentification à clé publique, un principal s'authentifie par le biais de sa clé privée. Pour vérifier cette preuve d'identité, chaque principal doit connaître la clé publique de son entité homologue.

Le principal problème dans ces protocoles est la distribution des clés publiques. Celle-ci peut s'effectuer au moyen des certificats de clé publique. Un certificat est une structure qui associe le nom d'un principal avec la clé publique du principal, le tout protégé par la signature d'une autorité de certification (CA).

Lors de la distribution des certificats, la CA peut être un service en ligne ou hors ligne. Dans le premier cas, un principal doit toujours consulter la CA pour récupérer un certificat. Dans le deuxième cas, la CA génère à l'avance les certificats des principaux et ensuite les met à la disposition des services d'annuaires. Les principaux récupèrent alors les certificats auprès de ce service. Afin de pouvoir révoquer les certificats générés par une CA hors ligne, on inclut dans leur structure une période de validité et un numéro de série. Le premier attribut offre une révocation implicite des certificats. On utilise le deuxième attribut pour la révocation explicite des certificats : dans des intervalles de temps préétablis, la CA publie une liste contenant les numéros de série des certificats révoqués.

2.3 Mécanismes d'autorisation

Dans la section précédente, nous avons décrit différents mécanismes permettant d'authentifier l'identité d'un principal et des messages qui sont échangés dans une session. Dans cette section, nous décrivons quelques méthodes pour la représentation et pour l'échange de l'information de contrôle d'accès (ACI) et pour son association aux objets. Plus spécifiquement, nous décrivons les deux méthodes suivantes :

- les *listes de contrôle d'accès*, qui permettent d'associer à un objet les droits d'accès qu'un ensemble de principaux a sur l'objet ;
- les *listes de capacités*, qui permettent d'échanger de l'information de contrôle d'accès entre un client et un serveur.

2.3.1 Listes de contrôle d'accès

Dans la section 2.1.3, nous avons presenté l'utilisation d'une matrice de contrôle d'accès pour representer les droits d'accès des principaux sur les objets. Dans la pratique, on ne sauvegarde pas une telle matrice car elle est souvent creuse et de grande taille [SS94]. Les méthodes classiques consistent à un stockage de la matrice par lignes ou par colonnes en ne prenant en compte que les cellules non vides. Dans cette section, nous présentons le stockage par colonnes et dans la section 2.3.2 le stockage par lignes.

Le stockage par colonnes consiste à associer à chaque objet une liste de tous les principaux qui peuvent y accéder et leurs droits d'accès. Cette liste est appellé *liste de contrôle d'accès* (ACL)[8] [Lam74, SS94]. La figure 2.11 donne un exemple d'une ACL. Cette ACL est associé à l'objet `Fichier` et contient trois utilisateurs (`Alice`, `Bob` et `Trudy`).

Fichier ↩

Principal	**Droits d'acces**
Alice	Lecture, Ecriture
Bob	Lecture
Trudy	Lecture, Ecriture
Etudiants	Lecture

Fig 2.11: Liste de contrôle d'accès

L'autorisation d'une requête pour un objet protégé par une ACL est composée d'une étape d'authentification et d'une autre de parcours de l'ACL. Lors de la réception de la requête d'un principal, le moniteur de référence utilise l'identité authentifiée du principal pour vérifier si l'ACL de l'objet contient une entrée pour le principal. Si oui, le moniteur vérifie alors si l'opération demandée est permise par les droits d'accès du principal. Suite à ces deux vérifications, le moniteur autorise ou non la requête du client.

Un problème inhérent aux ACL est que leur taille croit proportionnellement au nombre de principaux qui peuvent accéder à un objet. Le parcours et l'entretien d'une grande structure de données sont onéreux et demandent beaucoup de ressources. Une solution permettant de réduire la taille d'une ACL est l'utilisation des groupes d'utilisateurs [SS94].

2.3.1.1 Notion de groupes d'utilisateurs

Un *groupe d'utilisateurs*[9] est un ensemble de principaux ayant les mêmes droits d'accès sur un ou plusieurs objets [SS94]. Au lieu d'avoir des entrées individuelles dans une ACL pour chaque principal appartenant au groupe, on a une seule entrée pour tout le groupe. Par exemple, l'entrée `(Etudiants, Lecture)` dans l'ACL de la figure 2.11 autorise tous les membres du groupe `Etudiants` à lire l'objet `Fichier`. Dans un système, un groupe peut avoir une entrée dans plusieurs ACL.

Il existe plusieurs approches permettant à un principal de démontrer à un moniteur de référence son appartenance à un groupe [KPS95]. Une approche consiste à stocker dans la base de données du moniteur une liste mettant en relation les noms des groupes et des principaux. Le moniteur doit alors authentifier le principal, trouver les noms de groupes associés au principal et enfin, vérifier les entrées de l'ACL de l'objet.[10] Un

8. Des sigles du terme anglais *Access Control List.*
9. Nous utiliserons le terme groupe pour groupe d'utilisateurs
10. Les deux dernières étapes de l'opération d'autorisation peuvent être inversées.

inconvénient de cette approche est le besoin de dupliquer les listes de groupes lorsqu'un groupe donné a accès à un ensemble d'objets répartis sur plusieurs serveurs.

Une autre approche est de stocker les listes de groupes du système dans un serveur d'autorisation (cf. section 2.1.4.2). Au début d'une session, un principal récupère auprès de ce serveur un jeton protégé ou *capacité* qui spécifie les noms des groupes auxquels appartient le principal. Par la suite, le principal joint la capacité à ses requêtes. Un moniteur valide la capacité et utilise les noms des groupes qui y sont inscrits pour parcourir la liste de contrôle d'accès. Celle-ci est notamment l'approche suivi dans OSF/DCE [RKF92, HU95] (cf. annexe C). La section 2.3.2 analyse plus en détail la technique des capacités.

2.3.1.2 Avantages et limitations des listes de contrôle d'accès

Les ACL facilitent la gestion des utilisateurs car elles sont centralisées au niveau de chaque objet. Il est relativement simple d'ajouter des nouvelles entrées à une ACL, de modifier des entrées existantes ou de révoquer les droits d'un utilisateur.

Un autre avantage d'une ACL est que son parcours permet de déterminer les principaux qui peuvent accéder à l'objet et leurs droits d'accès.

En revanche, il est relativement complexe de déterminer les droits qu'un principal a sur l'ensemble d'objets d'un domaine. En effet, il est alors nécessaire d'examiner les ACL de tous les objets du domaine. Le problème symétrique se présente lors de la révocation des droits d'accès d'un principal sur les objets du domaine.

Un autre problème avec les ACL est que le temps de parcours d'une ACL est proportionelle au nombre des entrées qu'elle contient. Ce problème peut être partiellement résolu par un tri de l'ACL permettant d'optimiser son parcours [SS94].

2.3.2 Listes de capacités

La méthode des capacités a été initialement développée pour le stockage de la matrice de contrôle d'accès par lignes [Den82, Lev84, SS94]. Cette méthode consiste à associer à chaque principal une liste des objets auxquels il peut accéder et les droits d'accès qu'il possède sur eux. Cette liste est appelée *liste de capacités* (cap-liste) et les éléments de la liste sont les *capacités*.

Dans les systèmes classiques à capacités, une capacité est un jeton qui permet à son propriétaire d'utiliser un objet avec des droits déterminés [Lev84, Gon89b]. Une capacité identifie un objet et définit les droits d'accès en termes des opérations sur l'objet.

Une autre caractéristique des systèmes classiques à capacités est que le système d'exploitation est responsable de la gestion et de la protection des capacités des utilisateurs [Lev84]. Une capacité est protégée par le système dans le sens où il n'est pas possible de la forger ou de la modifier. Les seules opérations permises sur les capacités sont l'exécution d'une opération sur l'objet désigné si elle est autorisée, la diminution de droits et le passage en paramètre d'une opération. Par conséquent, il suffit qu'un utilisateur présente une capacité afin de bénéficier des droits d'accès spécifiés dans celle-ci.

Lors de l'avènement des systèmes répartis, le concept de capacités a subi plusieurs évolutions importantes [Gon89a, Gon89b, ECM89, HU95, PP95, ECM96] :

– Une capacité peut contenir non seulement les droits d'accès d'un utilisateur, mais également d'autres privilèges de sécurité.

– Une capacité peut être utilisée pour accéder à plusieurs objets.

– Une capacité pour un objet peut être générée par une autorité différente de celle contrôlant l'accès à l'objet. De plus, cette autorité peut résider dans un site différente de celui où se trouve le serveur d'objets.

– Les capacités peuvent éventuellement être échangées sur un réseau informatique.

– La liste de capacités d'un utilisateur peut être stockée dans la station de travail de l'utilisateur, même si la station n'est pas protégée par un moniteur de référence.

Il y a un prix à payer pour ces évolutions. D'une part, les capacités sont plus vulnérables au vol et à la falsification. D'autre part, la révocation des capacités est plus compliquée à mettre en oeuvre. Pour repondre à ces problèmes, la structure et le mode d'echange des capacités ont subi quelques modifications.

Dans la suite de cette section, nous décrivons l'utilisation des capacités dans les systèmes répartis. Nous présentons d'abord quelques unes des techniques utilisées pour l'attribution, la révocation et la protection des capacités. Nous décrivons ensuite la *structure* des capacités proposé par la norme ECMA-138 [ECM89] qui permette de prendre en compte ces techniques. Nous concluons cette section par une synthèse des principaux avantages et limitations associés aux capacités.

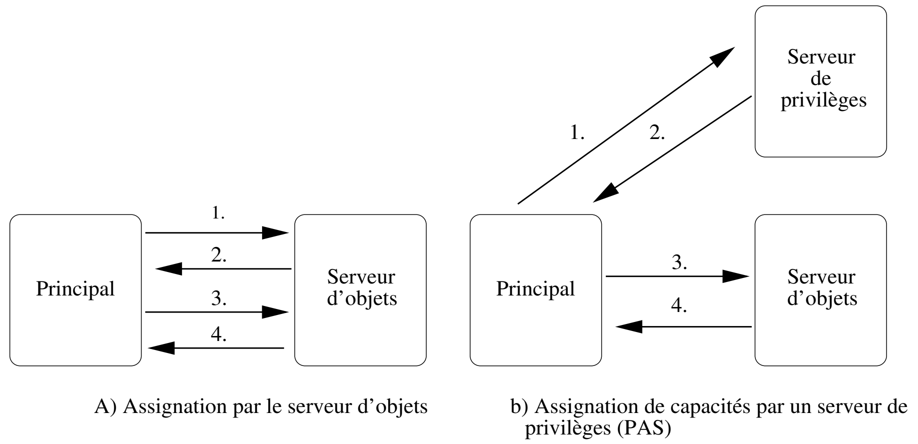
2.3.2.1 Assignation des capacités

Deux approches peuvent être distinguées pour l'attribution des capacités selon la relation entre l'autorité qui émet une capacité et celle qui valide la capacité [BGS92].

La première approche concerne l'*assignation décentralisée de capacités*. Dans cette approche, le moniteur de référence qui contrôle un objet est responsable de la génération et de la validation des capacités (fig. 2.12.a). Le moniteur décide, en fonction de sa politique de sécurité, si un principal peut acquérir une capacité et quels sont les droits d'accès compris dans cette capacité. Amoeba [TMvR86] est un exemple de système réparti qui utilise cette approche.

La deuxième approche met en oeuvre une *assignation répartie des capacités*. Dans cette approche, les fonctions de génération et de validation de capacités sont divisées entre un serveur de privilèges[11] (PAS) et les moniteurs de référence associés aux objets (fig. 2.12.b). Le PAS est responsable de générer et d'accorder des capacités aux

11. Il peut également s'agir d'un serveur d'autorisation (cf. section 2.1.4.2).



1. Demande d'une capacité
2. Envoi de la capacité
3. Envoi d'une requête accompagnée d'une capacité
4. Réponse du serveur

Fig 2.12: Attribution de capacités

principaux. Les moniteurs de référence sont responsables de la validation des capacités. Cette approche est mise en oeuvre dans les architectures de sécurité réparties SESAME [PP95] et OSF/DCE [RKF92, HU95].

Dans ces deux approches un accord doit exister entre les principaux et les expéditeurs des capacités sur la structure des capacités. En effet, un principal doit pouvoir identifier quelle capacité il lui faut présenter pour accéder à un objet. Pour ce faire, la capacité doit inclure des données précisant les objets auquelles elle est associée et le principal doit pouvoir interpreter ces données.

Similairement, lorsque l'expéditeur et le validateur d'une capacité sont deux entités différentes, un accord doit exister entre elles sur la structure de la capacité. En d'autres termes, le validateur doit pouvoir interpréter les capacités qui lui sont présentées.

2.3.2.2 Révocation des capacités

La révocation des capacités est l'opération permettant de supprimer les capacité qui appartiennent à un principal. La révocation des capacités peut être *implicite* ou *explicite*.

La révocation implicite est mise en oeuvre par l'inclusion d'une période de validité dans la capacité. L'expiration de cette période implique la révocation de la capacité. Cette technique est équivalente à celles utilisées pour l'expiration des tickets Kerberos et pour l'expiration des certificats de clés publiques. Comme dans ces autres techniques de révocation, les horloges des expéditeurs et des vérificateurs de capacités doivent être synchronisées.

La révocation explicite concerne la suppression d'une capacité avant qu'elle n'expire. Il existe plusieurs cas qui peuvent amener un administrateur de sécurité à révoquer explicitement une capacité [Kar89] :

– lors du changement des droits d'accès d'un principal possédant une capacité pour un objet, on peut être amené à révoquer la capacité de ce principal ;

– lors d'un changement du statut de sécurité d'un principal, il faut révoquer toutes les capacités possédées par le principal qui ne sont plus valables ;

– lors d'un changement du statut de sécurité d'un objet, il faut révoquer toutes les capacités qui ne doivent plus donner accès à l'objet.

La révocation explicite des capacités est complexe à mettre en oeuvre : les capacités peuvent être stockées dans plusieurs sites d'un système réparti [VAB91]. Une solution à ce problème est d'inclure des identificateurs uniques dans les capacités. On peut alors indiquer aux moniteurs de référence quelles sont les capacités obsolètes. Nous décrivons ci-dessous quelques méthodes de révocation fondées sur cette approche [TMvR86, ECM89, VAB91]. Chacune de ces méthodes s'applique à un des cas décrits précédemment.

Révocation d'une capacité d'un principal. On suppose ici que toute capacité a un numéro de série unique. Afin de révoquer une capacité, l'administrateur de sécurité communique le numéro de série de la capacité au moniteur de référence associé à l'objet, lui demandant de ne plus l'accepter. Le principal peut toutefois se servir d'autres capacités en sa possession avec le même moniteur. Le moniteur de référence doit conserver l'identificateur de la capacité révoquée, jusqu'à ce que celle-ci expire.

Révocation de toutes les capacités possédées par un principal. Dans cette technique, on suppose qu'une capacité inclut un attribut spécifiant l'identité du principal propriétaire de la capacité. L'administrateur de sécurité peut alors révoquer toutes les capacités possédés par un principal en notifiant aux moniteurs de référence de son domaine de refuser toute capacité portant l'identité du principal. Comme dans la méthode précédente, les moniteurs de référence concernés doivent stocker l'identité du principal jusqu'à ce que toutes les capacités que celui-ci possède aient expirées.

Révocation de toutes les capacités donnant accès à un objet. Si les capacités comportent un attribut spécifiant le nom de l'objet auxquel elles donnent accès, on peut alors révoquer toutes les capacités donnant accès à un objet en changeant le nom de l'objet. Les capacités désigneront alors un objet non existant.

En résumé, la révocation des capacités peut être effectuée de manière implicite ou explicite. Une révocation implicite implique l'association d'une période de validité dans les capacités ainsi qu'une synchronisation des horloges des expéditeurs et des vérificateurs des capacités. Une révocation explicite des capacités implique l'ajout d'identificateurs uniques dans les capacités et une invalidation de ces identificateurs, soit par une notification aux moniteurs de référence concernés de ne plus accepter ces identificateurs, soit par un changement du nom des objets.

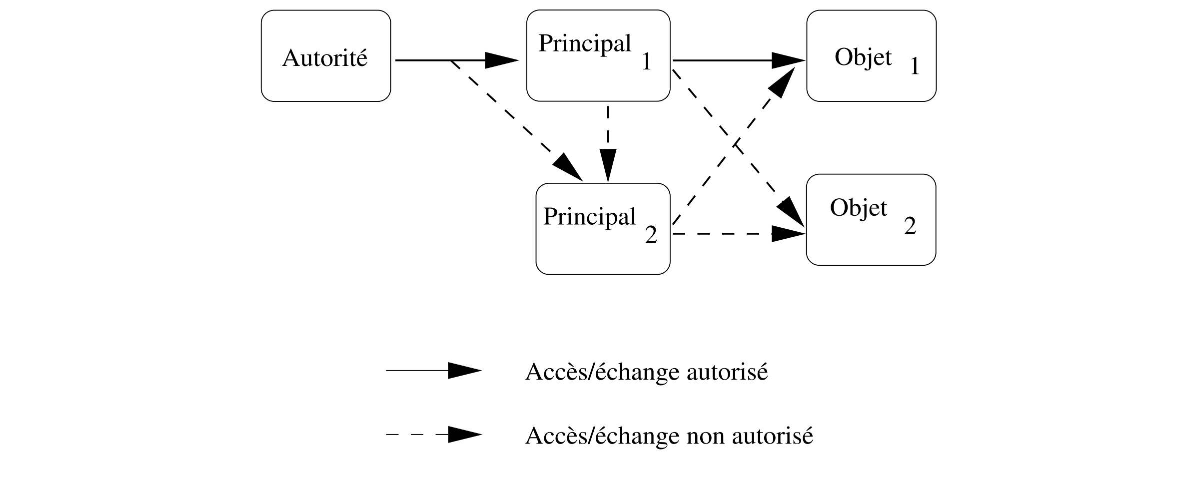


Fig 2.13: Menaces de sécurité liées à l'utilisation des capacités

2.3.2.3 Protection des capacités

Dans cette section nous identifions d'abord les principales menaces associées à l'utilisation des capacités. Nous décrivons ensuite différentes techniques de protection contre ces menaces.

Soit A une autorité qui accorde une capacité au principal P_1 pour accéder à l'objet O_1. Soit P_2 un principal qui récupère la capacité d'une manière non autorisée, par exemple, lors de sa transmission de A à P_1 ou par une propagation de la capacité de P_1 vers P_2. Sous ces conditions, nous pouvons distinguer trois classes de menaces [ECM89, FNS91] (fig. 2.13) :

- (M1) Modification non autorisée des valeurs de la capacité par P_1 ou par P_2.
- (U1) Utilisation de la capacité par P_2 avec n'importe quel objet.
- (U2) Utilisation de la capacité par P_1 avec un objet autre que O_1.

La structure des capacités, alliée aux mécanismes d'authentification constituent une solution pour répondre à ces menaces :

- (M1) peut être détectée au moyen des signatures numériques : l'autorité doit signer toute capacité qu'elle expédie et le moniteur de référence associé à l'objet doit vérifier ces signatures.

- (U1) est évitée en incluant l'identité du principal dans la capacité et en imposant l'authentification du principal lors de la présentation des capacités. Si l'identité authentifiée d'un principal ne correspond pas à celle inscrite dans la capacité, l'accès est refusé. En effet, cette authentification prouve que le principal est le propriétaire légitime de la capacité.
- (U2) peut être annulée par l'inclusion dans la capacité des attributs qualifiant le(s) objet(s) avec le(s)quel(s) on peut utiliser la capacité. Par exemple, il peut s'agir du nom d'un ou plusieurs objets, le nom d'un serveur, etc.[12] Un moniteur de référence autorise alors un accès à un objet si les attributs qualificatifs décrivent l'objet.

En résumé, les principales menaces présentes lors de l'échange des capacités sur le réseau sont la modification et l'utilisation non autorisée des capacités. Les techniques de protection contre ces menaces combinent l'inclusion des attributs dans les capacités identifiant les principaux et les objets avec l'authentification du principal et la signature des capacités par leurs expéditeurs.

Champ	**Description**
Qualificatif-objet	Identificateur des attributs de l'objet cible.
{privilèges}	Ensemble de privilèges de sécurité contenus dans la capacité.
UID-principal	Identificateur du propriétaire de la capacité.
UID-capacité	Identificateur unique de la capacité.
Période de validité	Temps de vie de la capacité.
UID-expéditeur	Identificateur de l'expéditeur de la capacité
Sig-expéditeur	Signature des champs précédents par l'expéditeur de la capacité

Fig 2.14: Structure d'une capacité

2.3.2.4 Structure des capacités

La norme ECMA-138 [ECM89] propose une structure générale pour la représentation et pour la protection des capacités[13]. Cette structure, illustrée dans la figure 2.14, permet notamment de prendre en compte les techniques d'assignation, de révocation et de protection décrites précedemmente. Nous allons donner maintenant quelques précisions supplémentaires sur les champs de la structure des capacités.

12. Le lecteur pourra trouver dans [ECM89, pp. 28-29] une liste de possibles qualificatifs pour les objets.

13. La norme ECMA utilise le terme de *certificat de privilèges* (PAC - Privilege Attribute Certificat) au lieu de capacité.

Le champ `Qualificatif-objet` spécifie les attributs des objets avec lesquels la capacité peut être utilisée. Le champ `{privilèges}` donne l'ensemble de privilèges du principal qui doit être pris en compte lors de la présentation de la capacité. Les droits d'accès du principal sont un exemple des privilèges.

Le champ `UID-principal` donne l'identificateur unique du principal autorisé à presenter la capacité. La valeur de ce champ peut correspondre à l'adresse réseau du système hôte du client, à une clé publique [LC94, ABKL90] ou à une référence à un autre certificat d'authentification, tel qu'un ticket Kerberos ou un certificat de clé publique [ECM96].

Les champs `UID-capacité` et `Période de validité` servent à la mise en place des techniques de révocation des capacités.

Les champs `UID-expéditeur` et `Sig-expéditeur` permettent d'identifier l'expéditeur de la capacité et d'authentifier les informations contenues dans celle-ci. Ce champ peut prendre deux types de valeurs, selon la localisation de l'expéditeur et du vérificateur d'une capacité :

- Lorsque l'expéditeur et le vérificateur sont deux entités distinctes, on doit utiliser une signature numérique (cf. section 2.2.3). Dans ce cas, le champ `ID-expéditeur` fournit au vérificateur de la capacité les informations nécessaires pour la validation de la signature. Par exemple, si le calcul de la signature utilise un algorithme à clé publique, le champ `ID-expéditeur` peut contenir le certificat de clé publique de l'expéditeur ou une référence à celui-ci [LC94].

- Lorsque l'expéditeur et le vérificateur sont la même entité, on peut utiliser une fonction de hachage pour générer la signature d'une capacité [TMvR86]. Avec cette méthode, un hash est calculé en utilisant les valeurs des champs de la capacité et une valeur secrète connue uniquement de l'expéditeur. L'expéditeur associe la valeur secrète à l'objet et l'utilise lors de la vérification de la capacité.

2.3.2.5 Avantages et limitations des capacités

L'utilisation des capacités apporte les avantages d'efficacité et de flexibilité. Efficacité car le temps nécessaire pour valider une capacité est relativement constant et indépendant du nombre d'utilisateurs ayant accès à un objet. Flexibilité car on peut utiliser une capacité pour transmettre des privilèges de sécurité d'un principal à un autre.

En revanche, la méthode des capacités a les inconvenients de révocation et de protection des capacités. La révocation des capacités est complexe à mettre en oeuvre car elles peuvent être stockées n'importe où dans un système. Bien que la révocation explicite des capacités soit relativement simple à mettre en oeuvre au moyen d'un mécanisme d'horodatage, la révocation explicite requiert un échange de données entre l'expéditeur et le vérificateur d'une capacité. Similairement, la protection des capacités est complexe à mettre en oeuvre car celles-ci peuvent circuler librement dans le système et peuvent être stockées dans des sites non protégés. Par conséquent, la protection des capacités fait appel aux techniques cryptographiques et aux protocoles d'authentification.

Enfin, un dernier inconvénient est qu'il est complexe de déterminer à un instant donné, quels sont les principaux qui possèdent une capacité sur un objet. En effet, il est alors nécessaire que chaque expéditeur des capacités garde une trace des capacités qu'il a octroyées.

2.4 Résumé du chapitre

L'objectif de ce chapitre était de présenter les principales techniques et principes de base du contrôle d'accès que les systèmes hypermédia empruntent aux systèmes traditionnelles. Cette présentation s'est décomposée en trois parties :

- Le modèle de contrôle d'accès de Lampson. Ce modèle général identifie les composantes intervenant dans un système de contrôle d'accès. Dans ce modèle, tout accès d'un principal (utilisateur ou processus agissant en son nom) à un objet (une ressource du système) est contrôlé par un moniteur de référence. Celui-ci est associé à une base de données d'autorisation. Le moniteur de référence autorise ou non une requête selon les informations stockées dans cette base et d'après les règles d'une politique de sécurité. La base elle-même est gérée par un administrateur de sécurité. Dans un système réparti, les principaux, les objets et les bases de données d'autorisation peuvent être localisés sur plusieurs sites.

- Les mécanismes d'authentification. Ils permettent au moniteur de référence de connaître quel principal se trouve à l'origine d'une requête. En particulier, nous avons présenté les différentes techniques que nous trouvons aujourd'hui dans les systèmes hypermédia répartis : authentification par adresse réseau, par mot de passe et par protocoles cryptographiques.

- Les mécanismes d'autorisation. Ils permettent au moniteur de référence d'évaluer quels sont les droits d'accès des principaux sur les objets. Plus spécifiquement, nous avons présenté les mécanismes des listes de contrôle d'accès et des capacités.

Chapitre 3

Les systèmes hypermédia répartis : présentation générale

Everyone - Everything - Connected [1]

6ème Conférence Internationale World-Wide Web, Santa Clara, Avril 1997

Dans ce chapitre; nous présentons les caractéristiques des systèmes hypermédia répartis que nous jugeons nécessaires à l'analyse des problèmes de contrôle d'accès. Plus spécifiquement, nous décrivons le modèle de données hypermédia, la distribution de l'information dans un système et les protocoles employés pour consulter cette information. Ces présentations sont illustrées à l'aide de plusieurs exemples de systèmes hypermédia répartis intégrant des mécanismes de contrôle d'accès.

3.1 Terminologie

Nous introduisons dans cette section la terminologie relative aux systèmes hypermédia. En particulier, les notions d'*hypermédia* et de *systèmes hypermédia répartis* sont définis.

La notion d'hypermédia

Le terme *hypermédia* est employé ici pour designer une approche à la *structuration* et à la *consultation* d'un espace d'information [Con87, TKM$^+$91, Boo92].

Dans l'hypermédia, un espace d'information est formé des noeuds (concepts) et des liens (relations) (fig. 3.1). Un *noeud* représente un fragment d'information. L'information contenue dans un noeud peut appartenir à un ou plusieurs types de média, tels que l'audio, le son, le texte, etc. Un *lien* spécifie une relation entre deux noeuds.

1. Tout le monde, toutes les ressources, connectés.

On appelle *graphe hypermédia*[2] à un ensemble de noeuds interconnectés par des liens dans un but spécifique.

La *consultation d'un graphe hypermédia* consiste en la visualisation des noeuds et en le parcours des liens existant entre les noeuds [Con87, Adi93]. Contrairement aux documents traditionnels, où un lecteur suit un parcours linéaire dans sa lecture, le parcours d'un graphe est *non linéaire* dans la mesure où l'utilisateur est libre de suivre n'importe lequel des liens partant d'un noeud vers d'autres noeuds. Par convention, on appelle *noeud racine* un point de départ dans la consultation d'un graphe. Tout graphe contient au moins un noeud racine.

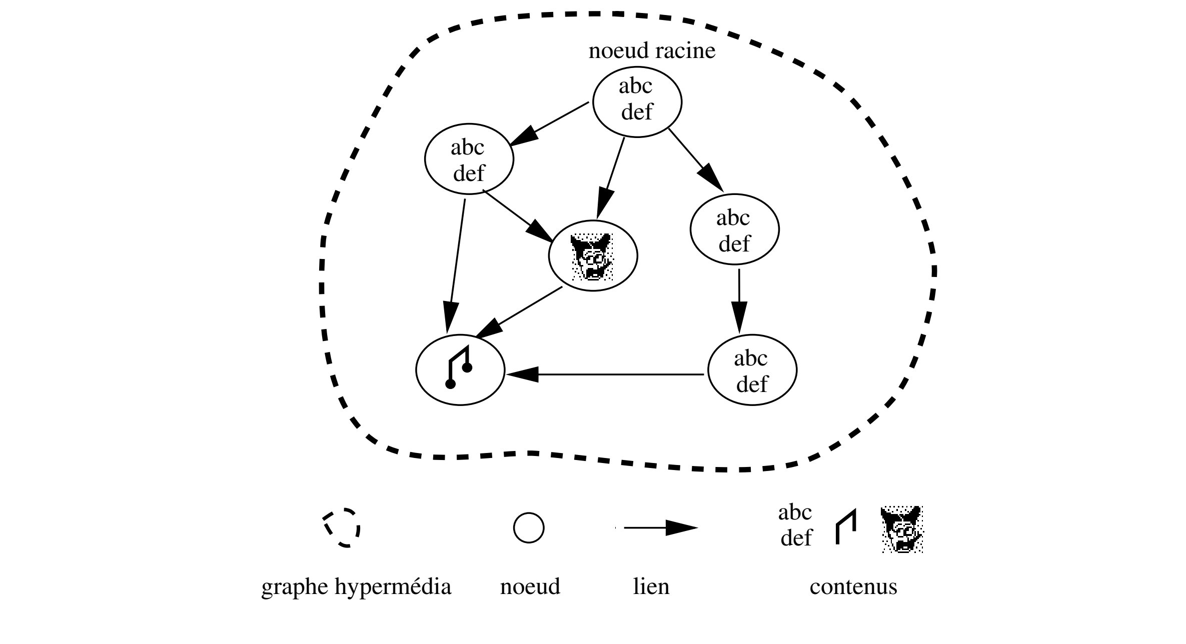


Fig 3.1: Organisation de l'information dans des graphes hypermédia.

Les systèmes hypermédia répartis

Un *système hypermédia réparti* est un système construit sur un modèle client/serveur. On appelle *hyperbase* l'ensemble de graphes stockés dans le système. Celle-ci peut être stockée sur un ensemble de serveurs distants [Adi93].

Un utilisateur consulte les composantes de l'hyperbase au moyen d'un programme client. Les systèmes World-Wide Web (WWW) [BL96], DCE/Web [LZ94, Lew95] et Hyper-G [AG95, Mau96] sont quelques exemples récentes des systèmes hypermédias répartis.

La réalisation d'un système hypermédia reparti est divisée en quatre rubriques, qui

2. Par simplification, nous utiliserons indistinctement les termes *graphe* et *graphe hypermédia* dans la suite.

sont [BLCL$^+$94] :

1. *la gestion de l'information*, qui concerne la structuration de l'information et son stockage ;

2. *la désignation de l'information*, qui concerne la localisation des données dans le système ;

3. *la consultation de l'information*, qui concerne les protocoles employés par les clients et les serveurs pour l'échange de l'information ;

4. *la migration de l'information*, qui concerne le déplacement de l'information d'un serveur vers un autre.

Nous examinons plus en détail chacune de ces rubriques dans les sections suivantes.

3.2 Modélisation et représentation des données

Comme nous l'avons mentionné au début de ce chapitre, les composants d'une hyperbase sont les noeuds, les liens et les graphes. Ces concepts peuvent être étudiés du point de vue de leur *structure* et de leur *représentation physique* dans les serveurs. L'élaboration d'un modèle général pour ces notions est encore un problème ouvert [ISO94, ISO92b, HS94, HBvR92]. Nous nous contentons ici de présenter un modèle simplifié, qui permet de rendre compte des objets d'information intervenant dans un système hypermédia et des relations inter objets. Nous renvoyons le lecteur aux références cités ci-dessus et aux études de Conklin [Con87] et de Tomet *et al.* [TKM$^+$91] pour des études plus détaillées des modèles hypermédia existants.

Notre modèle porte sur les concepts de noeud, de lien et des graphes, que nous examinons dans la suite.

3.2.1 Noeuds

Un *noeud* est une "unité atomique d'information" qui, dans l'essence, exprime une seule idée ou concept [Con87]. On peut caractériser un noeud par sa *structure* et par son *type*. Les deux paragraphes qui suivent sont consacrés à la définition de ces notions.

Structure des noeuds

Un noeud est constitué d'un *identificateur unique*, d'un ensemble d'informations de type *contenu* et d'un ensemble de *règles de présentation.*

L'*identificateur unique* du noeud permet de localiser celui-ci dans le système hypermédia. La structure des identificateurs relève de l'architecture du système, que nous décrivons plus loin dans ce chapitre.

Le *contenu* du noeud correspond à l'information stockée dans le noeud. Cette information peut appartenir à un ou plusieurs types de média, tels que "texte", "image", "audio", "vidéo," etc., comme l'illustre la figure 3.1.

Les *règles de présentation* d'un noeud précisent la manière dont le contenu doit être restitué à un utilisateur. Par exemple, la structure d'un noeud ayant des contenus du type audio et vidéo peut indiquer la synchronisation temporelle des contenus [ISO94, ISO92b].

Classification des noeuds

De manière générale, on peut distinguer entre deux grandes types de noeuds. Les noeuds *atomiques* sont des noeuds dont la restitution peut s'effectuer indépendamment d'autres noeuds. Les noeuds *composites* permettent de regrouper plusieurs noeuds (atomiques ou composites), dans le but de combiner leurs contenus dans une présentation homogène. Le cas classique est celui où une page de texte inclut des images. Si la page et les images sont des noeuds, on peut considérer les images comme des noeuds atomiques et la page comme un noeud composite.

La mise en relation entre un noeud composite et les autres noeuds qu'il inclut peut s'effectuer au moyen des *liens d'inclusion* [QV93], que nous étudierons dans la section 3.2.2.

Représentation physique des noeuds

Dans un système hypermédia reparti, les noeuds sont stockés dans des *serveurs de noeuds*. Dans ce serveur, un noeud peut prendre deux formes [BLPS96] :

– les *noeuds édités*,

– les *noeuds calculés*.

Un *noeud édité* est souvent assimilé à un fichier. Il peut être construit manuellement par un auteur, ou bien être le résultat d'un processus de conversion automatique. Ces noeuds sont statiques car leur contenu reste figé une fois que l'édition est finie.

Un *noeud calculé* est construit par un processus lors de la réception des requêtes. Typiquement, le nom du noeud comprend le nom du processus et les paramètres nécessaires pour la génération du noeud. Les noeuds calculés peuvent être statiques ou dynamiques. Si l'instanciation d'un noeud donne toujours le même résultat, on dit que le noeud est statique ; dans le cas contraire, on dit qu'il est dynamique. Une application des noeuds calculés est la présentation des données qui évoluent dans le temps ; par exemple, les données provenant d'un satellite météorologique [BLPS96].

3.2.2 Liens

Un *lien* exprime une relation entre deux noeuds. Il est dit orienté car il va d'un noeud (appelé *noeud de départ*) vers un autre noeud (appelé *noeud de destination*). Les liens servent à l'organisation d'une hyperbase en graphes et à l'indication du parcours de celle-ci [BLPS96].

On peut caractériser un lien par sa *structure*, par le type de *relation* qu'il représente et par sa *représentation physique*. Les trois sections qui suivent sont consacrées à la définition de ces notions.

Structure des liens

La structure d'un lien est usuellement constituée des quatre éléments suivants [HS94, ISO94] :

- un *identificateur unique* ;
- une paire d'*ancres* ;
- une *condition d'activation* ;
- un *ensemble d'attributs*.

L'*identificateur unique* d'un lien a la même fonction que celui d'un noeud : il permet de localiser le lien dans le système hypermédia. Nous décrivons quelques formalismes de désignation dans la section 3.3.

Une *ancre* correspond à une extrémité d'un lien. Elle repère le point de départ ou d'arrivé du lien. L'ancre est formée de deux éléments : l'identificateur d'un noeud et la spécification d'une région dans le contenu du noeud. En général, lors de la restitution d'un noeud la région d'une ancre de départ est mise en évidence par rapport au reste du contenu du noeud. Par exemple, plusieurs clients de consultation soulignent les mots d'un texte lorsque ceux-ci se trouvent à l'intérieur d'un lien.

La *condition d'activation* d'un lien spécifie quand on peut parcourir le lien. Cette condition peut dépendre des actions de l'utilisateur (e.g., sélectionner avec une souris la région d'une ancre) ou du contexte de restitution (e.g., l'utilisation d'un temporisateur associé au parcours automatique d'un lien). Dans la norme MHEG [ISO94], le lecteur peut trouver une classification générale des conditions d'activation d'un lien.

Les *attributs* associés à un lien permettent de caractériser le lien et les noeuds qui se trouvent à chaque extrémité du lien. Par exemple, dans l'architecture proposée dans [NS91], tout lien indique le type de média du contenu du noeud de destination. Si le client ne peut pas traiter le contenu, il ne parcourra pas le lien.

Classification des liens

Un lien peut représenter différents types de relations entre deux noeuds [Con87, DeR89] ; par exemple, des relation sémantiques, des progressions logiques, des références croisées, etc. Nous pouvons distinguer deux grands types de relations :

- *L'inclusion.* Cette association indique que le noeud d'origine inclut le contenu du noeud de destination. En d'autres termes, la restitution du noeud d'origine

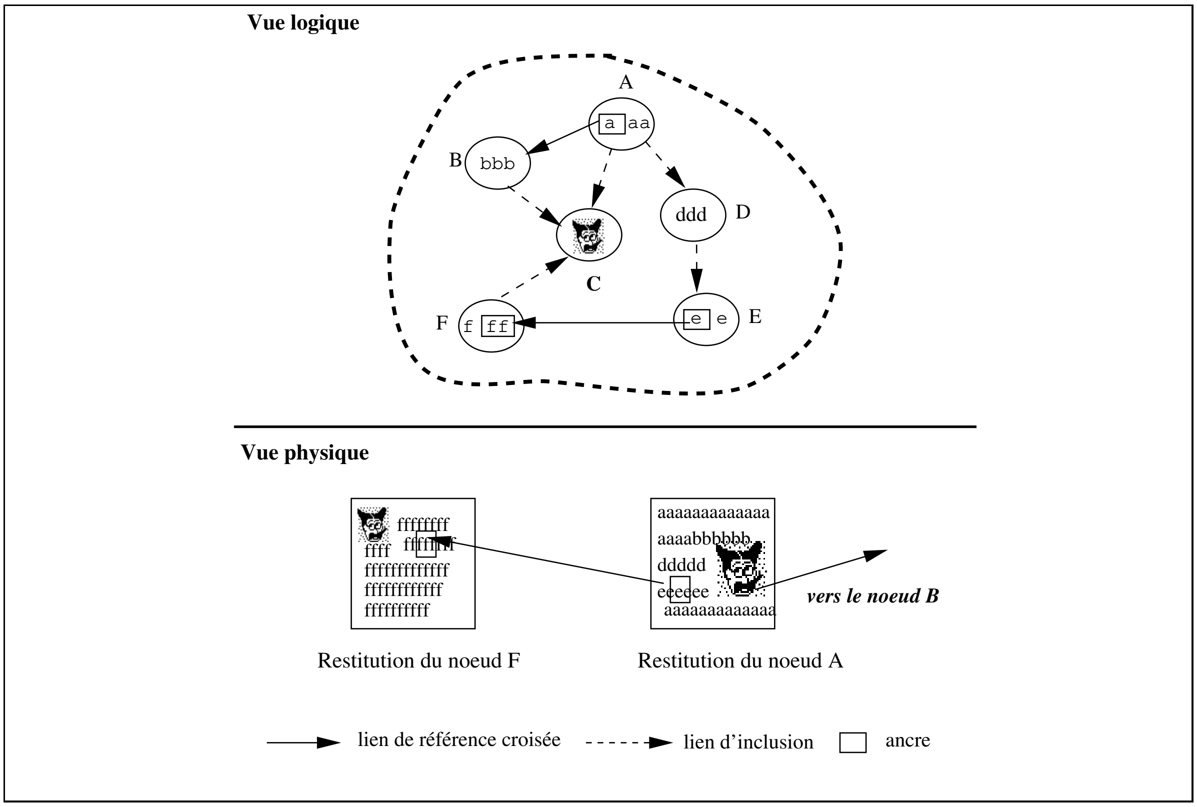


Fig 3.2: Relations pouvant être exprimées avec un lien.

comprend le contenu de ce noeud et celui du noeud de destination. On utilise notamment cette association pour associer un noeud composite à un noeud atomique.

– *Les références croisées.* Cette association indique que l'activation d'un lien conduit à la restitution du noeud destination.

La figure 3.2 illustre les différences entre les deux types d'associations. On y montre la restitution des noeuds A et F appartenant à un graphe. Notons que le noeud A inclut les noeuds C et D et que le noeud D inclut le noeud E. Ainsi, la restitution du noeud A représente une inclusion récursive de ces trois noeuds. Le résultat de cette opération est une composition du contenu des noeuds. Notons que le noeud E a un lien de référence croisée vers le noeud F. Comme le noeud A inclut le noeud E, il inclut également ce lien.

Représentation physique des liens

Les liens peuvent être imbriqués à l'intérieur des noeuds [Pla91, Rag97] ou être des citoyens de première classe dans l'hyperbase [Pea89, HKR$^+$92, AKM95]. Les deux approches se différencient dans la facilité pour ajouter un nouveau lien à un noeud existant et dans l'entretien des liens existants. Une comparaison plus détaillée de ces deux approches se trouve dans [Dav95].

Lorsque les liens sont extérieurs aux noeuds, on les regroupe dans un *serveur de liens*. Le client doit alors demander à ce serveur les liens associés au noeud. Comme dans le cas des serveurs de noeuds, les liens extérieurs peuvent être *édités* ou *calculés*.

Une liste de *liens édités* est établie manuellement par l'auteur d'un graphe. Un client récupère une telle liste en fournissant l'identificateur d'un noeud au serveur de liens. Selon le système, le serveur peut retourner la liste des liens sortant du noeud [NS91] ou, éventuellement, la liste des liens arrivant au noeud [Kap91]. Le client utilise ensuite la liste pour récupérer d'autres noeuds reliés par inclusion, pour afficher les ancres du noeud et, enfin, pour suivre un lien de référence croisée.

Une liste de *liens calculés* est générée par l'exécution d'un processus sur la base de données du serveur. La requête de l'utilisateur fournit au processus les caractéristiques des noeuds recherchés. L'intérêt principal des liens calculés résident dans la réduction du nombre des liens édités. Par exemple, dans Intermedia [HKR+92], un utilisateur peut consulter la définition de n'importe quel mot dans un noeud du type texte. Au lieu d'ajouter un lien statique entre chaque mot et sa définition, l'utilisateur envoie au serveur de liens des requêtes du type "retournez l'identificateur du noeud contenant la définition d'un tel mot." Un autre exemple est la recherche de tous les noeuds contenant un mot donné [Kap91].

3.2.3 Graphes hypermédia

Avant de décrire les propriétés d'un graphe hypermédia, nous rappelons brièvement les définitions données en début de chapitre. Un *graphe hypermédia* est une vue conceptuelle d'un ensemble de noeuds inter-reliés selon une organisation déterminée. Par exemple, le graphe d'un livre hypermédia peut être composé de plusieurs noeuds, chacun représentant une section reliée à une autre.

On appelle *noeud racine* un point d'entrée d'un graphe. Dans l'exemple précédent, un candidat pour le noeud racine pourrait être une page d'introduction donnant la table des matières du livre.

Propriétés d'un graphe hypermédia

Il existe plusieurs manières d'organiser un ensemble de noeuds dans un graphe [BD91]. La figure 3.3 montre quelques exemples d'interconnexion de noeuds à l'intérieur d'un graphe :

- Un noeud de départ peut être à l'origine de plusieurs liens vers d'autres noeuds (fig. 3.3.a).

- Un noeud peut être le destinataire de liens provenant de différents noeuds (fig. 3.3.b).

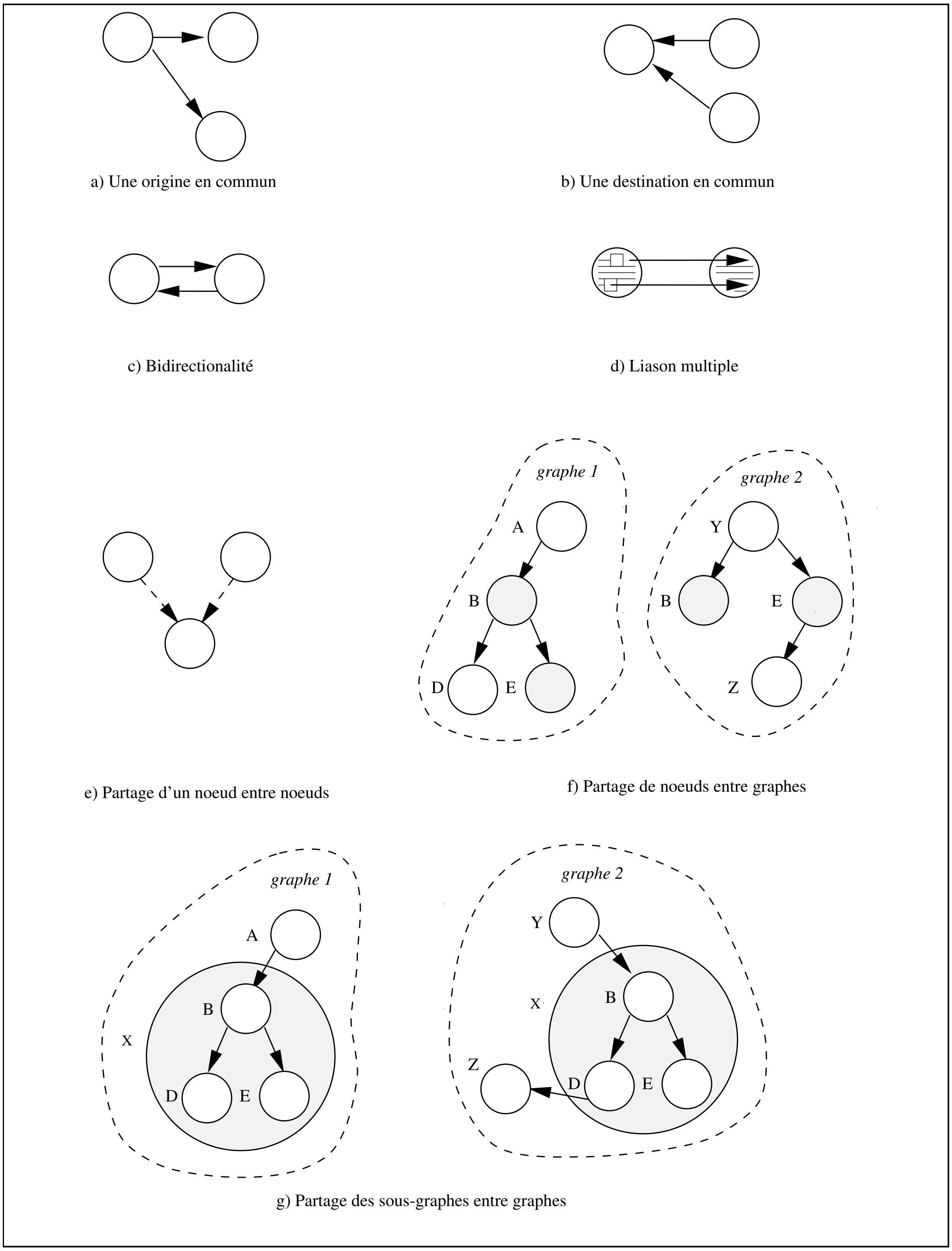


Fig 3.3: Exemples d'interconnexion de noeuds

- Un noeud destination peut avoir un lien vers le noeud à l'origine du lien, établissant ainsi une *bidirectionalité* entre les deux noeuds (fig. 3.3.c).

- Un noeud d'origine peut avoir plusieurs liens pointant sur différentes régions d'un même noeud de destination (fig. 3.3.d).

- Deux ou plusieurs noeuds peuvent inclure un même noeud (fig. 3.3.e).

- Deux graphes peuvent avoir plusieurs éléments en commun. Par exemple, deux graphes peuvent inclure un même ensemble de noeuds (fig. 3.3.f). En outre, deux graphes peuvent partager un sous-graphe (fig. 3.3.g). Notons que dans ce dernier cas, le sous-graphe peut être traité comme un noeud logique, et qu'on peut établir un lien de ce noeud vers n'importe quel autre noeud. C'est le cas du noeud X dans le $graphe2$.

En résumé, dans l'organisation d'un graphe deux noeuds peuvent avoir un ou plusieurs liens entre eux. De plus, deux noeuds ou deux graphes peuvent partager un même ensemble de noeuds.

Nous avons à présent décrit la structure et les caractéristiques générales des différents éléments du modèle de données hypermédia. La section suivante est consacrée aux mécanismes de désignation de ces éléments.

3.3 La désignation dans les systèmes hypermédia répartis

La *désignation* est l'opération qui consiste d'une part à associer des identificateurs (ou noms) aux objets et, d'autre part, à interpréter les identificateurs pour permettre l'accès aux objets [Kra91]. Plus spécifiquement, à partir d'un identifiant unique, on doit pouvoir répondre aux questions suivantes :

- Quel est le nom d'une ressource ?

- Où est stockée la ressource ?

- Comment peut-on accéder à la ressource ?

Ces problèmes ne sont pas nouveaux. Plusieurs solutions ont déjà été proposées dans le contexte des systèmes répartis [Kra91, Neu92]. On peut classer ces solutions en deux types :

- la *désignation absolue*, où l'identificateur d'un objet contient la localisation de l'objet dans le système;

- la *désignation symbolique*, où l'identificateur d'un objet est transparent par rapport à la localisation de celui-ci.

Dans la suite, nous décrivons deux propositions du projet WWW, les *Uniform Resource Locators* (URL) [BLCL+94] et les *Uniform Resource Name* [BL94, SM94, RN95], pour la désignation absolue et symbolique dans les systèmes hypermédia répartis.

3.3.1 Uniform Resource Locators (URL)

Une URL [BLe94] est un schéma de désignation absolue des ressources existantes sur le réseau Internet. Dans le projet WWW, on les utilise comme identificateurs uniques des éléments d'une hyperbase.

L'URL d'une ressource stockée dans un serveur consiste en plusieurs éléments, dont :

- Le *protocole* utilisé pour accéder au serveur.
- L'*adresse Internet* du site hébergeant le serveur et le *port d'écoute* du serveur.
- Le *chemin d'accès* de la ressource à l'intérieur du serveur.
- En option, la région de l'ancre de destination.

Examinons de près un exemple typique d'une URL :

```
http://www.w3.org/pub/WWW/Welcome.html#faq
```

Cette URL désigne l'ancre de destination `faq` se trouvant dans le fichier `Welcome.html`[3], se trouvant dans le répertoire `/pub/WWW` du serveur `www.w3.org`. On peut accéder à cette ressource au moyen du protocole HTTP [BLFF96].

Le principal avantage des URL est la simplicité de leur mise en oeuvre. Cependant, elles présentent deux grands inconvénients [Mau96, pg. 97] :

- Une URL décrit une seule instance d'une ressource. Pour équilibrer la charge d'un système hypermédia, on duplique souvent une ressource sur plusieurs serveurs. Cependant, en plus des URL, il faut un contexte pour indiquer que toutes ces copies sont des instances d'une seule ressource.
- Les ressources sont reliées à leur emplacement physique. La migration des ressources d'un serveur à un autre entraîne un changement de leurs URL. A ce moment-là, toutes les références existantes aux ressources avant leur migration deviennent invalides. Pour répondre à ce problème, on doit utiliser des protocoles spéciaux pour la redirection des ressources.[4]

3. Représentant un noeud.
4. Nous examinerons un de ces protocoles dans la section suivante.

3.3.2 Uniform Resource Names (URN)

Une URN [BL94, SM94, RN95] est un schéma de désignation symbolique des ressources. A chaque ressource d'un système correspond une seule URN. Une URN est constitué de l'adresse d'un *serveur de noms* et d'un *identificateur unique*. Le serveur de noms est responsable de la conversion d'un nom symbolique en un nom absolu.

Voici un exemple d'une URN :

```
urn://www.nameserver.org/012030
```

Cette URN spécifie le nom symbolique `012030` et l'adresse `www.nameserver.org` du serveur de noms pouvant traduire le nom symbolique en une URL.

Les URN possèdent plusieurs avantages :

– Elles sont indépendantes de la localisation de la ressource. Ainsi les ressources peuvent être déplacées sans besoin de les renommer.

– Elles sont non réutilisables. Il n'est pas possible d'attribuer la même URN à deux ressources. Les URN obsolètes disparaissent à jamais.

– Elles permettent la duplication des ressources. La réponse du serveur de noms peut contenir un ensemble d'URL pointant sur plusieurs instances d'une seule ressource [LS96]. L'utilisateur peut alors choisir l'URL qui lui convient le plus.

Un des inconvénients des URN est la dépendance vis à vis du serveur de noms. En effet, celui-ci peut devenir un goulot d'étranglement et le système devient sensible aux pannes de ce serveur. Ces inconvénients peuvent être évités par redondance du serveur de noms [Kra91]. L'idée de base est d'avoir plusieurs instances du serveur de noms. La liste de ces serveurs est connue de tout utilisateur du système. En cas de défaillance d'un serveur, les clients s'adressent au serveur suivant de la liste.

Les systèmes DCE/Web [LZ94, Lew95] et Hyper-G [AG95, Mau96] intègrent la notion d'URN. Actuellement, il n'existe que des implementations expérimentales des URN sur le système WWW [PJ96, SWJF96]. La spécification d'un format général des URN est en cours au sein des plusieurs groupes de travail de l'Internet Engineering Task Force (IETF) [LS96].

En résumé, comme dans les systèmes répartis, la désignation des ressources dans les systèmes hypermédia répartis peut être absolue ou symbolique. La désignation absolue spécifie la localisation d'une ressource dans le système. La désignation symbolique spécifie une clé et l'adresse d'un *serveur de noms* pouvant traduire la clé en un nom absolu. Les deux mécanismes de désignation les plus utilisés dans les systèmes hypermédia sont les URL et les URN, pour les noms absolus et les noms symboliques, respectivement. Dans la suite de ce document, nous adoptons ces deux termes, ainsi que celui d'*Uniform Resource Identifier* (URI) [BL94] pour parler indifféremment de l'un ou de l'autre.

Algorithme 3.3.1 Algorithme pour la consultation d'une hyperbase

Soit
TAB : un tableau de noeuds;
N, n : des noeuds;

Début
Répéter
Attendre événement;
Chargement d'un noeud
Selon événement
Cas parcours d'un lien de référence croisée
N = URI noeud destination;
Cas saisie de l'URI d'un noeud
N = URI du noeud;
Cas sélection de l'URI d'un noeud dans l'historique de la session
N = URI du noeud;
Fin Selon
Si N est une URN
Récupérer l'URL auprès du serveur de noms;
N = URL;
Fin Si.
Récupérer N auprès de son serveur d'objets;
Résolution des liens
Si la liste de liens associée à N n'est pas imbriquée dans le noeud
Récupérer la liste de liens associée à N auprès du serveur de liens;
Fin Si
Initialiser le tableau TAB **à** 0;
Pour Chaque n **Dans** la liste de liens de N
Si n est relié à N par un lien d'inclusion
Insérer n dans le tableau TAB;
Fin Si
Fin Pour
Pour Chaque élément **Dans** le tableau TAB
n = élément;
Si n est une URN
Récupérer URL auprès du serveur de noms;
n = URL;
Fin Si.
Récupérer n auprès de son serveur d'objets;
Fin Pour
Restitution des données
Combiner les contenus du noeud n avec celui de N ainsi que leurs ancres de départ des liens de référence croisée;
Présenter le résultat à l'utilisateur;
Fin Répéter
Fin.

3.4 Modèle d'une session de consultation

Au long des sections précédentes, nous avons noté que l'information associée à un système hypermédia peut être stockée sur trois types de serveurs : des *serveurs de noeuds*, des *serveurs de liens* et des *serveurs de noms*. Cette section donne un algorithme caractérisant la séquence de transactions client/serveur associée à une session de consultation type.

Koegel [Koe92] propose un algorithme de consultation pour illustrer quelques propriétés de la norme hypermédia MHEG [ISO94]. Nous avons étendu son algorithme pour inclure les notions de serveur de noms, de serveur de liens et d'interaction avec l'utilisateur (cf. algorithme 3.3.1).

L'algorithme est divisé en trois types de requêtes :

- demande de chargement d'un noeud ;
- résolution des liens ;
- restitution des données ;

Demande de chargement d'un noeud. L'algorithme commence lorsque le client reçoit une ordre de parcours d'un lien de référence croisée ou une ordre de chargement d'un autre noeud (pas 3-10). Dans le premier cas, l'utilisateur a sélectionné un lien dans un noeud affiché sur son écran. Dans le deuxième cas, l'utilisateur fournit au client l'URI d'un noeud. Celle-ci peut soit provenir de l'historique des sessions du client, ou bien être celle d'un noeud jamais parcouru auparavant [VSW94]. Le noeud demandé peut être le noeud racine d'un graphe, mais il peut également être un autre noeud quelconque.

Comme nous l'avons mentionné précédemment, une URI peut être un nom absolu ou symbolique. Dans ce dernier cas, le client doit consulter un serveur de noms afin de résoudre le noms symboliques du noeud en un nom absolu (pas 11-14). Une fois que l'utilisateur connaît l'URL du noeud, il peut alors le récupérer (pas 15).

Résolution des liens. Si le système hypermédia stocke les liens à l'extérieur des noeuds, le client doit alors récupérer la liste de liens associés au noeud demandé (pas 16-18). Ensuite, si la liste des liens inclut des liens d'inclusion, le client doit les suivre et récupérer les noeuds auxquels ils pointent (pas 19-32). Si un lien d'inclusion désigne un noeud au moyen d'une URN, le client doit d'abord convertir celle-ci en une URL (pas 27-30).

Restitution des données. Enfin, le client combine le contenu des différents noeuds qu'il a rapatrié avec la liste de liens de référence croisée. Il présente ensuite le résultat à l'utilisateur (pas 33-34).

En résumé, avec cet algorithme nous avons identifié trois types de protocoles client/serveur associées à la consultation dans un système hypermédia réparti ;

- la consultation d'un serveur de noms ;
- la consultation d'un serveur de noeuds ;
- la consultation d'un serveur de liens.

La section suivante présente plus en détail les protocoles qui interviennent dans la session de consultation.

3.5 Protocoles de consultation

La consultation de l'information dans un système hypermédia réparti s'effectue par l'intermédiaire des protocoles client/serveur [Adi93]. Ces protocoles appartiennent au niveau applicatif, s'appuyant sur des protocoles de transport fiable, tels que TCP/IP. Ils ont été conçus pour offrir un accès rapide et flexible à l'hyperbase [BLFF96].

Nous allons maintenant dans un premier temps caractériser le comportement générique d'un protocole de consultation. Nous présentons ensuite les protocoles de consultation associés à une session : *consultation d'un serveur de noeuds*, *consultation d'un serveur de liens* et *consultation d'un serveur de noms*.

3.5.1 Modèle générique d'un protocole de consultation

Les protocoles de consultation d'un système hypermédia réparti se caractérisent par leur simplicité et par leur mode d'opération sans état [Adi93, BLCL+94].

Dans un protocole de consultation générique, le dialogue entre un client et un serveur se déroule en quatre étapes :

1. **Ouverture :** le client ouvre une connexion de transport vers le serveur.
2. **Demande :** le client transmet une requête au serveur.
3. **Réponse.** le serveur renvoie la réponse de la requête (il peut s'agir d'un message d'erreur).
4. **Fermeture :** cette opération est effectuée par le client ou par le serveur. Après fermeture, le serveur ne retient aucune information sur la transaction qui vient de finir.

Dans la première génération de protocoles de consultation, chaque connexion est assimilée à une seule requête [Adi93]. Bien que cette simplicité ait facilité le développement des serveurs par plusieurs équipes de recherche, elle présente des problèmes de performance. En effet, l'ouverture d'une connexion est reconnue comme une opération lente et coûteuse. Les protocoles de deuxième génération, tels que HTTP/1.1 [FGM+96] et HG-CSP [Mau96] résolvent ce problème en permettent d'envoyer plusieurs requêtes par connexion.

Dans un souci de simplification, nous ne montrons plus les étapes associées à l'ouverture et la fermeture des connexions réseaux dans la description ultérieure des protocoles de consultation.

1.	Client → serveur de noeuds	:	URL du noeud
2.	Serveur de noeuds → Client	:	noued

Fig 3.4: Protocole de consultation d'un serveur de noeuds

3.5.2 Protocole de consultation d'un serveur de noeuds

Un client utilise ce protocole pour récupérer des noeuds auprès d'un serveur de noeuds. Comme le montre la figure 3.4, la requête du client comprend l'URL du noeud. Le noeud demandé peut être du type statique ou calculé [BLPS96].

Une fois que le serveur a trouvé (pour les noeuds statiques) ou généré (pour les noeuds dynamiques) le noeud demandé, il le renvoie au client. Notons que du point de vue du client, le résultat de la requête est toujours un noeud, quelque soit la représentation physique de celui-ci dans le serveur.

1.	Client → Serveur de liens	:	URL du noeud ou paramètres de recherche
2.	Serveur d'liens → Client	:	liste de liens

Fig 3.5: Protocole de consultation d'un serveur de liens

3.5.3 Protocole de consultation d'un serveur de liens

Un client utilise ce protocole lorsque les liens sont des entités distinctes des noeuds et sont stockés dans un serveur de liens. La consultation du serveur s'effectue au moyen des requêtes dont le résultat est une liste des liens (fig. 3.5). La liste de liens peut être du type *statique* ou *dynamique* [Kap91].

Un problème particulier avec une liste de liens se présente lorsqu'un noeud est inclut dans deux ou plusieurs graphes : il faut utiliser uniquement les liens correspondant au contexte du graphe que le client est en train de consulter. Une solution à ce problème est d'inclure dans les liens un attribut spécifiant le graphe où il est utilisé. On peut alors utiliser cette attribut pour filtrer la liste des liens associés au noeud. Cette opération de filtrage peut s'effectuer au niveau du serveur de liens [HKR+92, SSS93, KPS93] ou au niveau du client [NS91].

Pour un filtrage au niveau du serveur, le client indique au serveur l'URL d'un noeud et l'identificateur du graphe que l'utilisateur est en train de consulter. Le serveur retourne ensuite une liste des liens associés au noeud dans ce contexte.

Pour un filtrage au niveau du client, le client récupère la liste complète des liens associés au noeud. Le client utilise alors l'identificateur du graphe courant pour filtrer cette liste avant de la présenter à l'utilisateur.

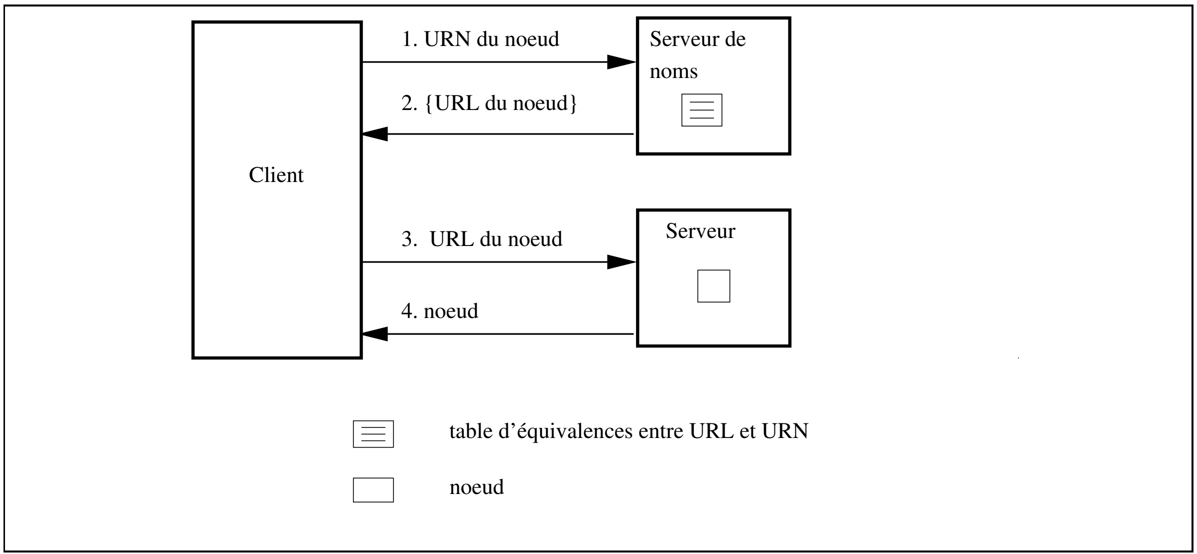


Fig 3.6: Protocole de migration par résolution de noms

3.5.4 Protocole de consultation d'un serveur de noms

L'objectif de ce protocole est de convertir l'URN d'un noeud en une URL. Plusieurs protocoles pour la résolution de noms ont été proposés pour le projet WWW [SM94, LS96], mais aucune solution n'a été retenue pour l'instant. Nous décrivons dans cette section un protocole générique, inspiré de [LS96].

La figure 3.6 montre ce protocole. Le serveur de noms gère une base de données où se trouvent les équivalences entre les URN et les URL. A chaque URN peut correspondre une ou plusieurs URL ; c'est le cas lorqu'il existe plusieurs instances d'un noeud. Un client donne au serveur une URN et récupère en retour une liste des URL équivalentes. Le client examine ensuite la liste des URL et essaie successivement chacune jusqu'à ce qu'il puisse trouver un serveur.

3.6 Techniques pour la migration de noeuds

Lors des évolutions matérielles ou de configuration d'un système hypermédia, on peut être amené à déplacer un noeud d'un serveur vers un autre [ICL96]. Il est important d'assurer que les liens qui pointent sur un noeud soient toujours valables, indépendamment des migrations éventuelles du noeud [SM94]. Il existe trois types de solutions à ce problème [ICL96] :

- la mise à jour des liens, qui traite la migration au niveau de la structure des données des objets ;
- l'utilisation des URN, qui traite la migration au niveau de la désignation des noeuds ;
- l'utilisation d'un protocole de redirection, qui traite la migration au niveau des protocoles de consultation.

Mise à jour des liens. Il s'agit de mettre à jour les liens pointant vers un noeud qui a migré. Si le système utilise un serveur de liens, la mise en place de cette solution est très simple : il faut juste mettre à jour la base de données de ce serveur [KPS93]. En revanche, cette solution est peu adaptée aux systèmes où les liens sont imbriqués à l'intérieur des noeuds [Dav95]. D'une part, les noeuds à l'origine des liens peuvent être répartis sur plusieurs serveurs. D'autre part, il n'est pas aisé de connaître l'ensemble des liens qui aboutissent à un noeud.

Utilisation des URN. Si le système hypermédia utilise des URN pour identifier les noeuds, on peut prendre en charge la migration d'un noeud par la mise à jour de la base de données du serveur de noms. Ainsi, la résolution d'une URN en une URL donnera toujours le bon emplacement du noeud.

Utilisation d'un protocole de redirection. Ce protocole, décrit dans la norme HTTP [BLFF96], utilise exclusivement des URL. La figure 3.7 montre les étapes de ce protocole. Lors de la migration d'un noeud vers un autre serveur, on laisse dans le serveur d'origine une URL pointant sur le nouveau emplacement du noeud. Lorsque le serveur d'origine reçoit une requête pour le noeud, il renvoie au client la nouvelle URL de l'objet. Le client contacte alors le serveur de destination et récupère le noeud.

Un inconvénient de cette technique est que si un noeud migre plusieurs fois, il laissera derrière une chaîne d'URL. On est obligé de garder tous les maillons de cette chaîne pendant le temps de vie du noeud. En effet, on peut établir un lien vers le vers le noeud à n'importe quel instant. En conservant la chaîne, on peut garantir que tous ces liens aboutiront au noeud. Par ailleurs, lors de la destruction du noeud, il faut nettoyer la chaîne de redirections pour éviter de laisser une trace inutile.

En résumé, la migration des noeuds dans le système peut être traitée a trois niveaux différents : structure des données, désignation des noeuds et protocoles de consultation. Parmi les techniques présentées ici, seules celle de modification des liens extérieurs aux noeuds et celle de modification des URN sont ponctuels : elles ne requièrent que la mise à jour d'une base de données centralisée. En revanche, la modification des liens

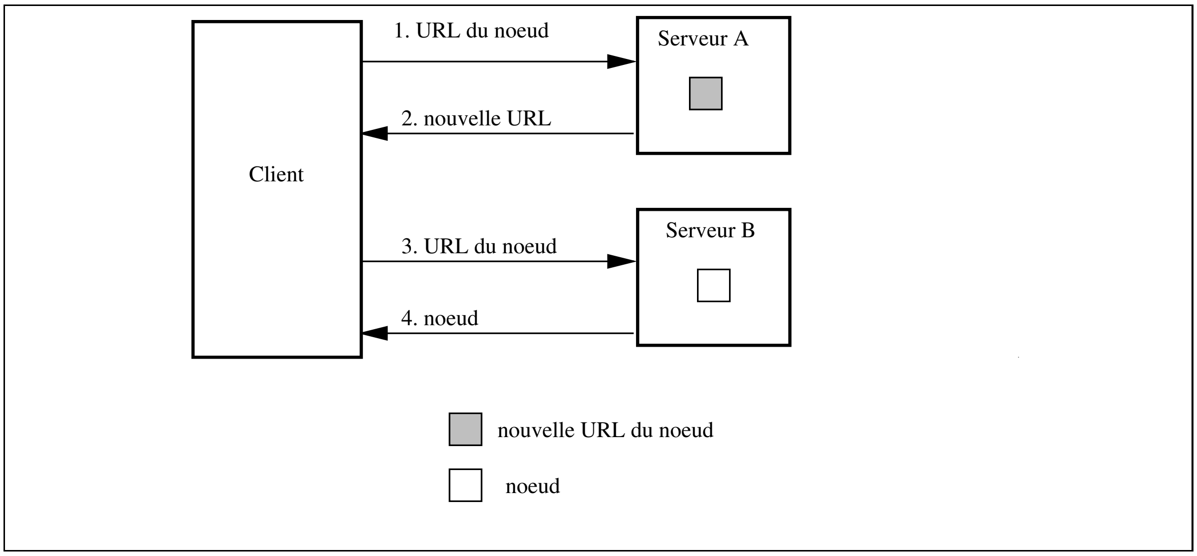


Fig 3.7: Protocole de migration par redirection

lorsque ceux-ci sont imbriqués à l'intérieur des noeuds peut impliquer la mise à jour d'un nombre indéterminé de noeuds. La migration par redirection ne concerne qu'un serveur. Par contre, lors des migrations successives d'un noeud, elle peut s'avérer une technique coûteuse en termes de requêtes et de gestion.

3.7 Résumé du chapitre

Nous avons défini dans ce chapitre un système hypermédia réparti comme un système de type client serveur permettant la consultation d'un espace d'information, appelé hyperbase, pouvant être réparti sur plusieurs serveurs. Cette présentation s'est concentrée sur quatre aspects :

– le modèle de données hypermédia,

– les mécanismes de désignation,

– les protocoles de consultation,

– la migration des noeuds dans le système.

Le modèle de données hypermédia se caractérise par une organisation de l'information dans des graphes. Les noeuds d'un graphe représentent des unités atomiques d'information, pouvant appartenir à un ou plusieurs types de média. Les arcs du graphe, appelés liens, représentent des relations entre les noeuds.

Une session est caractérisée par la consultation d'un ou plusieurs graphes. La consultation d'un graphe est équivalent à son parcours : l'utilisateur récupère d'abord un des noeuds du graphe et puis suit les liens allant de ce noeud vers les autres noeuds du graphe.

Lors de la consultation d'un graphe un client peut avoir à consulter plusieurs serveurs :

- les composantes du graphe peuvent être stockées sur différents serveurs du système ;
- on peut faire appel à un serveur de noms pour traduire un nom symbolique en un nom logique ;
- un noeud peut avoir migré d'un serveur vers un autre en laissant derrière lui sa nouvelle URL ;
- les liens peuvent être extérieurs aux noeuds et être stockés dans un serveur de liens.

Chapitre 4

Le contrôle d'accès dans les systèmes hypermédia répartis

> (L'homme) multiplie les tentatives pour qu'on lui permette d'entrer et fatigue la sentinelle de ses prières. La sentinelle lui fait subir parfois de petits interrogatoires, l'interroge sur son village et sur beaucoup d'autres sujets, ...
>
> Franz KAFKA
>
> (*Le Procès*)

Dans le chapitre précédent, nous avons examiné les caractéristiques de base des systèmes hypermédia répartis. Ce chapitre élargit cette étude aux spécificités propres du contrôle d'accès dans ces systèmes.

Peu d'expériences ont été menées dans ce domaine. Néanmoins, la comparaison des projets de recherche traitant ce sujet n'est pas simple car les hypothèses de base et les motivations de chaque projet sont souvent différentes.

L'objectif de ce chapitre n'est pas de présenter tous les projets de recherche traitant ce sujet, mais plutôt de classifier les problèmes de base et les principales solutions à ces problèmes, en les illustrant à travers des projets pertinents.

Nous allons tout d'abord présenter les différents besoins de base, puis nous proposerons une analyse comparative des solutions existantes.

4.1 Besoins de base pour le contrôle d'accès

Dans cette section, nous examinons les besoins de base posés par le contrôle d'accès dans les systèmes hypermédia répartis. Le contrôle d'accès concerne la protection des ressources dans un système. Dans un système hypermédia réparti, on s'intéresse essentiellement à la protection de l'information contenue dans son hyperbase.

La modèle de données hypermédia introduit plusieurs nouveaux besoins de contrôle d'accès. Dans ce document, nous nous focalisons sur trois besoins que nous estimons fondamentaux :

– **Identification des objets.** Quels sont les objets qui doivent être protégés ?

– **Cohérence de la gestion des utilisateurs.** Quels sont les objets concernés par une opération de gestion ?

– **Partage de noeuds protégés.** Quelles sont les dispositions à prendre en compte lors de l'inclusion d'un noeud dans plusieurs graphes ?

4.1.1 Identification des objets

Dans la section 2.1.1, nous avons défini un objet comme une ressource d'un système devant être protégée. Dans un système hypermédia on assimile les objets aux noeuds et aux liens qui forment l'hyperbase [SSS93, ZHB95, Kap91]. La motivation pour la protection de chacun de ces objets est différente :

– Les noeuds sont les dépositaires de l'information de l'hyperbase. L'application des mesures de contrôle d'accès sur eux implique donc la protection de cette information.

– Les liens expriment des relations existantes entre les noeuds. Le but de cette protection est de filtrer la vision qu'un utilisateur a des liens en sortie d'un noeud.

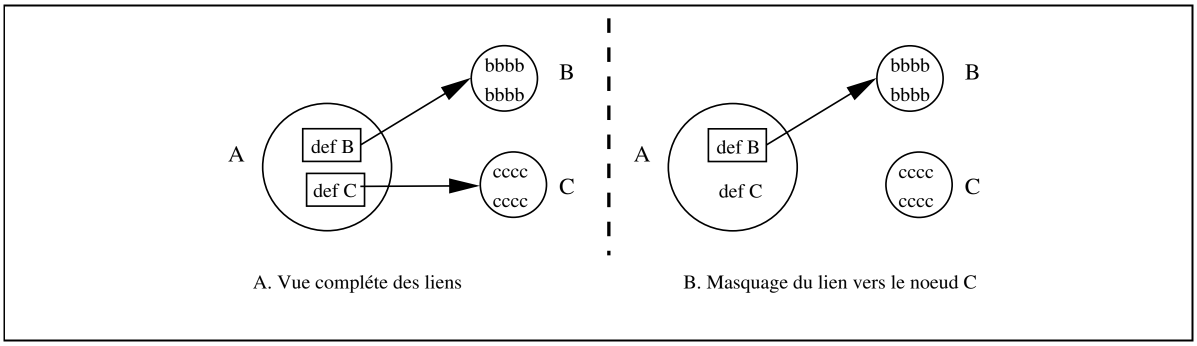


Fig 4.1: Exemples de protection des liens

Notons que lorsque l'on veut protéger l'accès à un noeud, il ne suffit pas de protéger les liens menant vers le noeud. En effet, un utilisateur peut demander un noeud sans besoin de suivre un lien. Il faut protéger explicitement l'accès au noeud et, éventuellement, celui aux liens.

La protection des liens est intéressante lorsqu'on veut cacher à un utilisateur les liens allant vers des noeuds non autorisés. En d'autres termes, il s'agit de donner à l'utilisateur une vue conforme à ses droits d'accès. La figure 4.1 en donne un exemple. Soient A, B et C trois noeuds où A à un lien vers B et C (fig. 4.1.a). Si un utilisateur n'a pas un droit d'accès sur C, on peut alors cacher le lien allant vers ce noeud (fig. 4.1.b).

Dans la pratique, nous trouvons une protection des liens uniquement dans les systèmes où les liens sont séparés des noeuds. Dans les systèmes où les noeuds et les liens sont intégrés, mettre en oeuvre cette protection est coûteux car il faut associer à chaque liens des attributs spéciaux et filtrer chaque accès à un document [SSS93]. Parmi les systèmes qui nous servent d'exemple, seul Hyper-G, un système basé sur un serveur de liens, propose une protection des liens.

Dans la suite de ce document, nous ne considérons que la protection de noeuds, celle-ci étant la plus utilisée dans les systèmes existants.

4.1.2 Gestion cohérente des utilisateurs

Nous avons vu dans la section 2.1.3.1 que l'autorisation d'une requête équivaut à l'évaluation de différents types d'informations de contrôle d'accès (ACI). En général, une partie de cette information est indexée par objet et l'autre par utilisateur. La gestion des utilisateurs concerne d'une part l'association des objets et de l'ACI des utilisateurs et d'autre part la modification et la révocation de cette information.

Dans un système hypermédia, le problème qui se pose est de propager les opérations de gestion de l'ACI à toutes les composantes d'un graphe. En effet, le modèle de données hypermédia préconise une structuration de l'information en graphes. Afin qu'un utilisateur puisse accéder à toutes les composantes d'un graphe, il faut garantir qu'il aura un droit d'accès sur elles. Du point de vue de la gestion, cette contrainte implique que l'attribution et la révocation des droits d'accès sur un graphe doivent s'appliquer à toutes les composantes du graphe.

4.1.3 Partage de noeuds protégés

Le modèle de données hypermédia permet l'inclusion d'un noeud dans plusieurs graphes. Le problème qui se pose lors du partage d'un noeud protégé est de pouvoir distinguer l'ACI spécifiée pour chaque graphe [HKR+92, SSS93, AKM95]. Sans cette organisation, on risque de répercuter les opérations de gestion ayant comme cible l'ACI d'un graphe sur l'ACI d'un autre graphe.

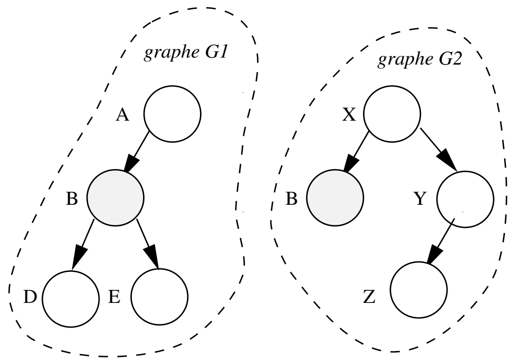


Fig 4.2: Exemples de visibilité des liens

Par exemple, soit B un noeud partagé par les graphes $G1$ et $G2$ (fig. 4.2). Supposons maintenant qu'un utilisateur possède l'autorisation pour consulter $G1$ et $G2$. Enfin, supposons que nous voulons supprimer l'ACI de l'utilisateur par rapport à $G1$. S'il n'existe pas une distinction claire entre les deux ensembles d'ACI, on ne peut avoir aucun critère pour connaître quelle est l'ACI concernée par l'opération. Dans le pire des cas, on pourrait supprimer l'ACI de l'utilisateur pour les deux graphes.

En résumé, nous venons de présenter trois besoins que nous considérons fondamentaux pour le contrôle d'accès dans les systèmes hypermédia répartis :

- *Identifications des objets* : il peut s'agir des noeuds ou des liens ;
- *Cohérence de la gestion des utilisateurs* : on doit définir les opérations d'affectation et de révocation des droits d'accès en termes de graphes. En outre, on doit appliquer ces opérations à toutes les composantes d'un graphe ;
- *Partage de noeuds protégés* : Lors de l'inclusion d'un noeud protégé dans plusieurs graphes, il faut différencier entre l'ACI des différents graphes.

4.2 Analyse comparative des approches existantes

Dans cette section, nous proposons une analyse et comparaison des principales approches pour le contrôle d'accès dans les systèmes hypermédia répartis. Le but de cette analyse est de discuter les avantages et les limitations que ces approches présentent par rapport aux besoins de gestion des utilisateurs et de partage de noeuds protégés.

La comparaison des différentes approches n'est pas simple car les hypothèses de base, les motivations et l'environnement d'utilisation pour chaque approche sont souvent différents. Notre démarche a été d'étudier la prise en compte des besoins de base par rapport à trois aspects de ces approches :

- la localisation de l'information de contrôle d'accès dans le système,
- l'organisation de l'information de contrôle d'accès associée à un objet,
- les mécanismes de défi d'autorisation qui permettent à un utilisateur de connaître quelle ACI il doit inclure dans ses requêtes.

Le premier de ces aspects permet d'analyser la gestion des utilisateurs. Les deux autres permettent d'étudier le partage de noeuds. Avant d'étudier plus en détail chacun de ces aspects, nous allons brièvement présenter les systèmes que nous avons retenus pour cette analyse.

4.2.1 Systèmes étudiés

Bien que depuis plusieurs années plusieurs modèles et systèmes hypermédia ont été proposés et mis en oeuvre, très peu parmi ceux-ci offrent des fonctionnalités de contrôle d'accès. Nous avons choisi parmi ces systèmes et modèles ceux qui reflètent le mieux les grandes tendances qui se sont dégagées ces dernières années dans la protection des systèmes hypermédia répartis. Dans la littérature, nous avons parfois trouvé des systèmes ayant les mêmes caractéristiques. Nous avons retenu dans ce cas les systèmes les plus récents.

Nous allons maintenant donner une présentation sommaire des systèmes étudiés. Le lecteur intéressé par une description complète est invité à consulter les références bibliographiques mentionnées ci-dessous.

4.2.1.1 World-Wide Web (WWW)

Le projet WWW [BLCL+94, BL96] est un système hypermédia réparti à niveau mondial adapté au réseau Internet. Le projet à débuté au CERN, Suisse en 1989 et est actuellement poursuivi par le Consortium International W3C.

WWW est organisé selon une architecture client/serveur. Chaque serveur ne gère et ne connaît que l'ensemble des objets dont il est responsable.

Le système de base WWW repose sur plusieurs normes :

- le langage de description HTML (*Hypertext Markup Language*) [Rag97],
- les formalismes de désignation absolue URL (*Uniform Resource Locator*), [BLCL+94] et de désignation symbolique URN (*Uniform Resource Name*) [BL94, SM94, RN95],
- le protocole d'échange de données HTTP (*Hypertext Transfer Protocol*) [BLFF96, FGM+96].

Dans WWW, les liens sont imbriqués à l'intérieur des noeuds sources. Lors de la réception d'un noeud, un interpréteur HTML sur la station client assure la mise en page décrite en HTML et affiche les ancres des liens au départ du noeud. Chaque lien décrit au moyen d'une URL un noeud destination et optionellement l'ancre destination. Lors de l'activation d'un lien, le client envoie une requête au serveur qui gère le noeud destination. Certains systèmes expérimentaux supportent l'utilisation des URN [PJ96, SWJF96]. Dans ce cas, le lien repère le nom d'un serveur de noms et un index. Lors de l'activation du lien le client envoie une requête au serveur de noms, qui lui envoie en retour l'URL du noeud.

Le contrôle d'accès dans WWW suit une approche décentralisée : chaque serveur intègre une base de données d'autorisation lui permettant d'évaluer les droits d'accès des utilisateurs sur les objets gérés par le serveur. Le système de base WWW supporte une authentification par adresse réseau et par mot de passe [FGM+96, Ste95]. On trouve également plusieurs protocoles cryptographiques d'authentification, dont les plus récents sont S-HTTP (*Secure HTTP*) [RS96b] et SSL (*Secure Socket Layer*) [FKK96]. Le premier est une extension au protocole HTPP tandis que le deuxième se place entre la couche HTTP et la couche transport.

4.2.1.2 DCE/Web

DCE/Web [LZ94, Lew95] est un projet en cours de l'Open Software Foundation (OSF). L'objectif de ce projet est d'intégrer le système DCE (Distributed Computing Environment) [RKF92, HU95] dans un environment WWW. Parmi les fonctionnalités que DCE/Web apporte au WWW, nous pouvons citer :

- des noms symboliques pour la désignation des objets dans le système,
- le stockage des listes de groupes du système dans un serveur de privilèges,
- un mécanisme permettant la gestion distante de l'information de contrôle d'accès des serveurs,
- un système d'authentification des utilisateurs basé sur le protocole Kerberos [1],
- un service de temps réparti pour la synchronisation des horloges physiques des différents sites. Il garantit la synchronization des horloges au sein d'un intervalle donné.

4.2.1.3 Phoenix

Le projet Phoenix [LK94], développé dans l'Université de Chicago, est un système de création et de consultation des documents hypermédia construit sur un système WWW. Phoenix apporte au système WWW l'utilisation un serveur d'autorisation centralisé et l'utilisation du protocole d'authentification Kerberos.

4.2.1.4 Hyper-G

Hyper-G est un système d'édition et de consultation de documents hypermédia développé à l'Université de Technologie de Graz, Autriche [AK94, AKM95, Mau96].

Contrairement à WWW, dans Hyper-G les noeuds et les liens sont des entités physiquement distinctes. Les noeuds sont stockés dans des serveurs de noeuds et les liens dans un serveur spécialisé appelé *serveur Hyper-G*. Outre d'avoir les fonctionnalités d'un serveur de liens, le serveur Hyper-G intègre également les fonctions de serveur de noms et de serveur d'autorisation.

Dans Hyper-G, les transactions suivent le protocole de consultation HG-CSP (*Hyper-G Client-Server Protocol*) [KP93]. Dans ce protocole, une session se caractérise par l'établissement d'une connexion entre le client et le serveur Hyper-G pour la durée de la session. Le serveur Hyper-G agit comme intermédiaire entre l'utilisateur et tous

1. Ce protocole a été décrit en détail dans la section 2.2.4.3.

les autres serveurs du système. Lorsque le serveur reçoit une requête pour un lien, il réponde directement au client ; lorsqu'il reçoit une requête pour un noeud, il redirige la requête vers le serveur de noeuds approprié.

Hyper-G suit une approche centralisée d'autorisation dans la mesure où le serveur Hyper-G est responsable de l'authentification des utilisateurs et de l'autorisation de toutes leurs requêtes. Le serveur Hyper-G supporte les protocoles d'authentification fondés sur les adresses IP et sur les mots de passe.

4.2.2 Localisation de l'information de contrôle d'accès

La localisation de l'ACI définit son emplacement par rapport aux objets du système. Dans un système hypermédia, les objets peuvent être répartis sur plusieurs serveurs. De plus, ils peuvent migrer éventuellement d'un serveur à un autre. Le problème qui se pose alors est de garantir l'accès à l'ACI associée à un objet indépendamment de l'emplacement de cet objet dans le système.

Dans les systèmes hypermédia étudiés, nous trouvons trois types d'ACI :

- des listes de contrôle d'accès (ACL) définissant pour chaque utilisateur et groupe d'utilisateurs leurs droits en terme d'opérations sur les objets du système ;
- des listes de groupes d'utilisateurs ;
- des informations permettant d'authentifier un utilisateur, tels que des mots de passe, l'adresse IP de la station de travail de l'utilisateur, des clés cryptographiques, etc.

Les systèmes étudiés utilisent tous les mêmes types d'ACI. En revanche, ils se différencient dans leur stratégie de localisation de cette information. Nous pouvons en identifier trois classes d'approches :

- les *approches décentralisées,* telles que WWW, où l'ACI se trouve dans le même serveur que les objets ;
- les *approches réparties*, telles que DCE/Web, où l'ACI est divisée en deux parties : celle qui est commune à tous les serveurs du système et celle qui est locale à chaque serveur. La première est stockée dans un serveur spécialisé et la seconde sur chaque serveur.
- les *approches centralisées*, telles que Phoenix et Hyper-G, où l'ACI de tous les serveurs d'objets est regroupée dans un serveur d'autorisation.

Nous allons maintenant présenter les principes de chacune de ces approches ainsi que leur effet sur la migration des noeuds et sur la gestion des utilisateurs.

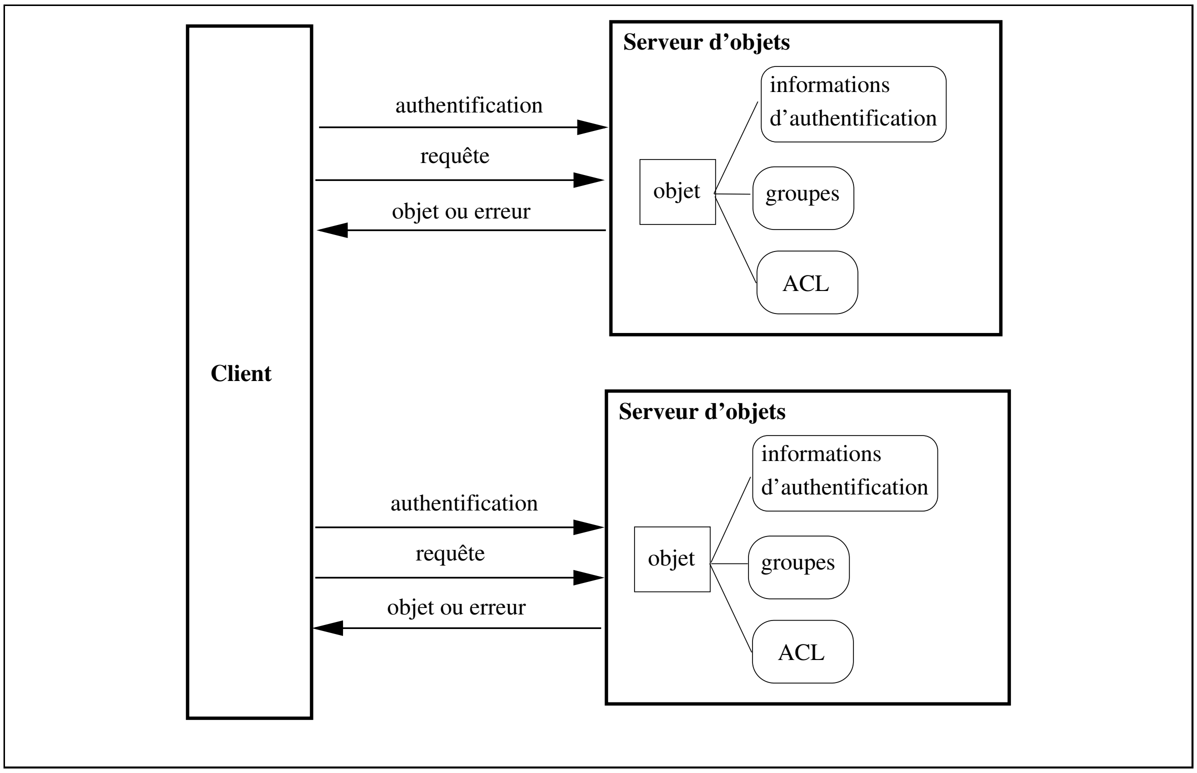


Fig 4.3: Approche d'autorisation décentralisée WWW

4.2.2.1 Approche décentralisée

Dans cette approche, chaque serveur d'objets stocke l'ACI nécessaire pour contrôler l'accès à ses objets. Chaque serveur peut donc authentifier ses clients et autoriser leurs requêtes indépendamment des autres serveurs du système (fig. 4.3). WWW est un exemple d'un système mettant en oeuvre cette approche.

Cette approche nécessite la migration conjointe d'un noeud et de son ACI. De plus, cette migration doit s'effectuer sur un canal protégé assurant la confidentialité et l'intégrité de l'ACI. Dans WWW, aucun mécanisme pour la migration physique des noeuds n'est supporté. La migration est donc à la charge des administrateurs des noeuds.

Avantages et inconvenients de l'approche décentralisée

Un des avantages de cette approche est l'indépendance d'un serveur par rapport à la disponibilité des autres serveurs du système. Il peut contrôler l'accès à ses objets indépendamment des autres serveurs du système. Si un serveur tombe en panne, les autres serveurs ne seront pas affectés.

Notons que la décentralisation de l'ACI complique toutefois la gestion des utilisateurs. Comme nous l'avons décrit dans la section 4.1.2, les opérations de gestion peuvent impliquer toutes les composantes d'un graphe. Si ces composantes se trouvent sur plusieurs serveurs, on sera obligé de mettre à jour la base de données d'autorisation de chacun de ces serveurs.

Un autre inconvénient de l'approche centralisée concerne les informations pour l'authentification des clients. On doit dupliquer ces informations dans chaque serveur hébergeant une composante du graphe. Si un intrus attaque un serveur de noeuds, il pourra éventuellement analyser ces informations et les utiliser pour accéder aux noeuds stockées dans d'autres serveurs.

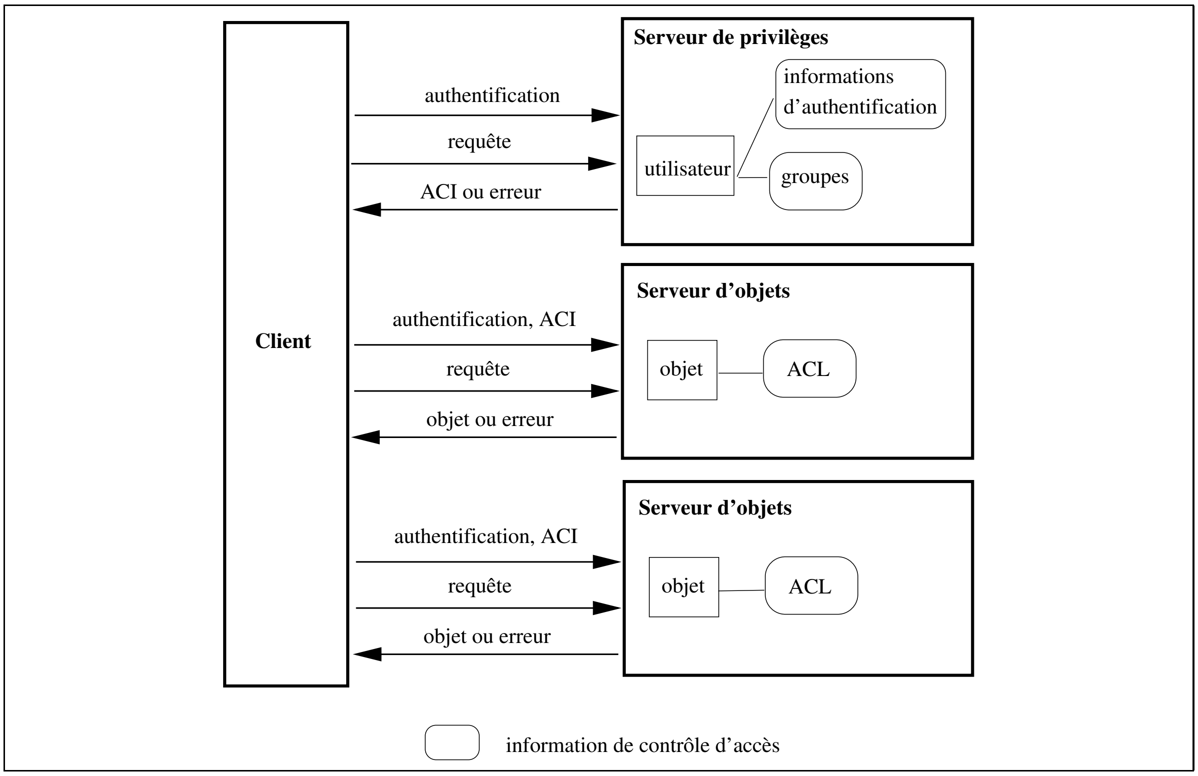


Fig 4.4: Approche d'autorisation répartie DCE/Web

4.2.2.2 Approche répartie

Le principe de base de cette approche est de regrouper dans un serveur de privilèges l'ACI commune aux serveurs d'objets du système.

Le système DCE/Web [Lew95] suit une approche répartie où le serveur de privilèges stocke les listes de groupes d'utilisateurs (fig. 4.4). Comme dans WWW, les serveurs de noeuds stockent les listes de contrôle d'accès associées aux noeuds.

DCE/Web utilise les tickets Kerberos (cf. section 2.2.4.3) comme capacités pour véhiculer les listes de groupes des utilisateurs. Au début d'une session, un utilisateur récupère un TGT Kerberos contenant une copie des groupes auxquels il appartient. La liste de groupes de l'utilisateur est dupliquée par la suite dans chaque nouveau ticket expédié par le TGS. Un serveur d'objets retrouve cette information lors de l'authentification du client. Le lecteur trouvera dans l'annexe C une présentation plus détaillée sur les mécanismes de protection dans le système DCE.

Avantages et inconvenients de l'approche répartie

Le principal avantage de cette approche est l'utilisation des groupes pour simplifier la gestion des utilisateurs. En plus d'aider à la réduction de la taille d'une ACL, un groupe peut être valable sur plusieurs serveurs. Lorsqu'un groupe a un droit d'accès sur un graphe, on peut octroyer le même droit à un utilisateur en l'incluant dans le groupe. Cette opération ne nécessite qu'une intervention dans le serveur de privilèges, ce qui réduit le nombre de serveurs devant être mis à jour. Toutefois, cette approche n'apporte aucun avantage dans la gestion des utilisateurs lors de la création et la suppression des groupes des utilisateurs. Il faut alors contacter tous les serveurs concernés et mettre à jour leurs ACL.

Un autre avantage de l'approche répartie vient de la centralisation de l'information d'authentification des clients dans un serveur d'authentification. En effet, les serveurs ne contiennent aucune information d'authentification pouvant être réutilisée par un intrus.

Bien que l'approche introduit une transaction additionnelle pour consulter le serveur de privilèges, nous ne considérons pas que celle-ci soit un inconvénient. En effet, cette transaction n'a lieu qu'au début d'une session. On peut donc considérer qu'elle est négligeable par rapport au nombre de transactions qui caractérisent une session type.

Le principal inconvénient de cette approche est l'héritage des problèmes liés à la gestion des capacités (cf. section 2.3.2, pg. 35). Notamment, lors du changement de l'information du serveur de privilèges, on peut être obligé de révoquer toutes les capacités expédiées précédemment pour éviter des accès non autorisés ; par exemple, si une des capacités valides d'un client contient le nom d'un groupe auquel le client n'appartient plus.

Dans cette approche, la consultation d'un graphe dépend initialement de la disponibilité du serveur de privilèges. Tant qu'un utilisateur ne possède pas un certificat, il ne pourra pas accéder aux composantes du graphe. En revanche, une fois que l'utilisateur récupère une capacité, l'autorisation devient indépendante de l'état de fonctionnement du serveur de privilèges.

4.2.2.3 Approche centralisée

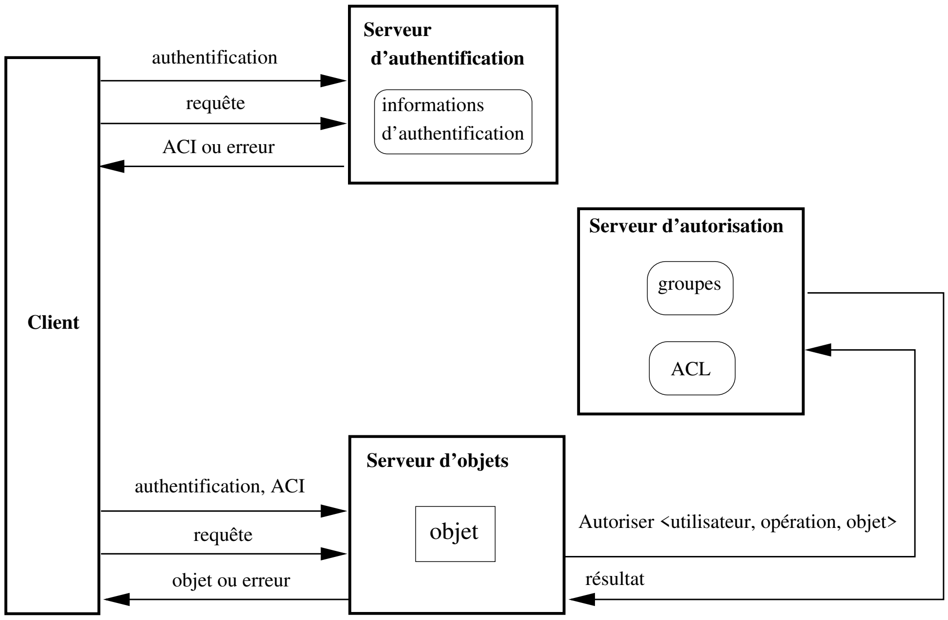
Le principe de base de cette approche est de regrouper l'ACI du système dans un seul serveur d'autorisation. Ce serveur devient alors le responsable de l'évaluation des droits d'accès des utilisateurs.

Deux variantes des serveurs d'autorisation peuvent être distinguées :

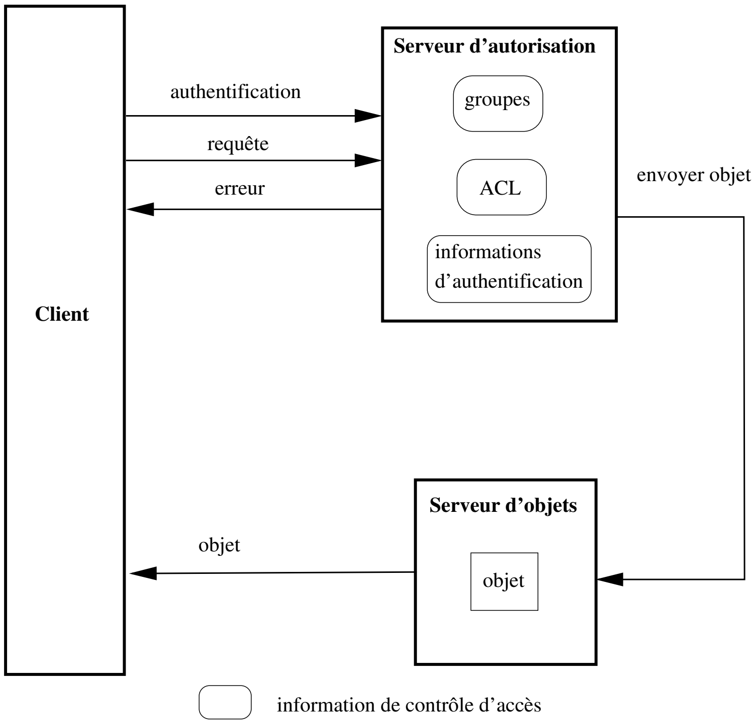
- les serveurs d'autorisation *passifs*,

- les serveurs d'autorisation *actifs*.

Dans la variante *passive* du serveur de autorisation, les serveurs d'objets contactent



(a) Serveur d'autorisation passif Phoenix



(b) Serveur d'autorisation actif Hyper-G

Fig 4.5: Deux types d'approches centralisées le serveur d'autorisation lors de la réception d'une requête (fig. 4.5.a). Dans la variante *active* du serveur d'autorisation, ce serveur agit comme un intermédiaire entre les utilisateurs et les serveurs d'objets (fig. 4.5.b). S'il autorise une requête, le serveur contactera alors le serveur d'objets approprié.

Le système Phoenix est bâti au-dessus d'un serveur d'autorisation passif et du système d'authentification Kerberos. Le serveur d'autorisation stocke les listes de contrôle d'accès et des groupes du système tandis que le serveur Kerberos stocke les informations d'authentification des utilisateurs. Grâce à Kerberos, les serveurs d'objets peuvent authentifier leurs clients sans avoir besoin de connaître *a priori* leur identité.

Le système Hyper-G intègre dans un seul serveur, appelé serveur *Hyper-G*, les fonctions de serveur d'autorisation actif, de serveur de liens et de serveur de noms. Contrairement à WWW, où un client doit s'authentifier auprès de chaque serveur d'objets qu'il contacte, dans Hyper-G, il doit s'authentifier uniquement auprès du serveur Hyper-G.

Notons qu'avec un serveur d'autorisation passif, toute requête pour un objet implique une autre requête vers le serveur d'autorisation. En revanche, un serveur d'autorisation actif permet d'économiser une requête lorsque celle-ci est refusée.

Avantages et inconvenients de l'approche centralisée

Cette approche présente les avantages d'une simplification de la gestion des utilisateurs et d'une indépendance de l'ACI vis à vis de la migration des noeuds. En effet, les opération de gestion des utilisateurs ne concernent que le serveur d'autorisation. De même, comme l'ACI se trouve dans un seul serveur, celle-ci ne doit pas être déplacée lors de la migration des objets.

Un autre avantage de l'approche centralisée est la possibilité d'ajouter les fonctionnalités d'une base de données au serveur d'autorisation. Ainsi, il est possible d'effectuer des évaluations d'autorisation très élaborées. C'est déjà le cas de Hyper-G.

Le principal inconvénient de cette approche est celui d'une sensibilité à l'indisponibilité du serveur d'autorisation. En effet, toute requête pour un objet nécessite une consultation à ce serveur. Si le serveur d'autorisation est saturé par un grand nombre de requêtes simultanées ou si le réseau reliant ce serveur aux autres serveurs du système est saturé, le temps de résolution d'une requête en subira les conséquences.

4.2.2.4 Discussion comparative des différentes approches

Nous allons maintenant comparer les trois approches de localisation d'ACI. Pour ce faire, nous utilisons deux critères :

- *l'efficacité de la gestion des utilisateurs*, qui mesure le nombre des serveurs devant être contactés pour une opération de gestion appliquée sur un graphe formé de n noeuds,
- *l'efficacité de l'autorisation*, qui mesure le nombre de serveurs devant être contactés pour autoriser l'accès à un noeud.

Plus la gestion des utilisateurs sera efficace, moins de serveurs seront concernés par une opération de gestion. Similairement, plus l'autorisation sera efficace, moins de serveurs seront impliqués lors de l'autorisation d'une requête. Le tableau suivant montre l'application de ces deux critères aux approches de localisation de l'ACI.

critère *Localisation de l'ACI*	Efficacité la gestion de des utilisateurs	Efficacité de de l'autorisation
décentralisée	n serveurs d'objets	serveur d'objets
répartie	serveur de privilèges + n serveurs d'objets [b]	serveur de privilèges [a] + serveur d'objets
centralisée passive	serveur d'autorisation	serveur d'autorisation + serveur d'objets
centralisée active	serveur d'autorisation	serveur d'autorisation + serveur d'objets

[a] Uniquement lors de la demande d'une capacité.
[b] Uniquement lors de l'ajout/destruction des groupes

Tab 4.1: Comparaison des differentes approches de localisation de l'ACI

La comparaison des trois approches met en évidence un conflit entre l'efficacité de la gestion et l'efficacité de l'autorisation. L'approche décentralisée présente la meilleure efficacité d'autorisation car tout serveur peut autoriser localement un accès. En revanche, cette approche présente la plus mauvais efficacité de la gestion car à chaque opération de gestion sur un graphe, il faut contacter tous les serveurs où sont stockés les composantes du graphe. Les approches centralisées présentent des caractéristiques contraires à celles des approches décentralisées : la gestion ne concerne qu'un serveur tandis que l'autorisation implique deux requêtes, une pour le serveur d'autorisation et une autre pour le serveur d'objets.

L'approche répartie semble offrir le meilleur compromis d'efficacité parmi les trois approches. Dans cette approche, la gestion de l'ACI commune aux système n'implique que le serveur de privilèges. En revanche, l'efficacité de la gestion de toute autre ACI est la même que celle des approches décentralisées. Par exemple, soit G un graphe composé de n noeuds. Si l'on veut donner accès au groupe d'utilisateurs *etudiants*, il faudra ajouter cette entrée dans les listes de contrôle associées aux noeuds du graphe. Pour effectuer cette opération, on a besoin de contacter tous les serveurs stockant les noeuds de G. Une fois que le groupe est défini, l'ajout des nouveaux membres au groupe n'implique que le serveur de privilèges.

L'efficacité de l'autorisation de l'approche répartie est celle de l'approche décentralisée augmentée d'une seule requête supplémentaire. Un utilisateur doit contacter une première fois le serveur de privilèges au début de sa session pour récupérer une capacité contenant une partie de son ACI. Dans la suite, il suffit que l'utilisateur présente cette capacité pour pouvoir accéder aux composantes du graphe.

Enfin, notons que dans le cas des systèmes tels que Hyper-G, où un seul serveur réunit les fonctionalités de serveur d'autorisation et de serveur de liens, une approche centralisée n'influe pas sur l'efficacité de l'autorisation. En effet, que l'accès à un noeud soit protégé ou non, il faut toujours contacter deux serveurs avant de pouvoir accéder au noeud. En revanche, si le serveur d'autorisation autorise une requête pour un noeud, il doit contacter directement le serveur de noeuds, pour lui commander de transmettre le noeud à l'utilisateur. Cela représente une charge additionnelle pour le serveur d'autorisation. En outre, cet échange entre le serveur d'autorisation et le serveur de noeuds implique l'existence d'un canal protégé de communication entre les deux serveurs. En conclusion, bien que l'approche centralisée présente un bon compromis d'efficacité de gestion et d'autorisation, elle introduit des contraintes de disponibilité de service et requiert l'utilisation des canaux protégés.

4.2.3 Organisation de l'information de contrôle d'accès

Tous les systèmes hypermédia étudiés supportent le partage des noeuds entre graphes. Comme nous l'avons décrit dans la section 4.1.3, lors de l'inclusion d'un noeud protégé dans plusieurs graphes, il faut pouvoir organiser l'ACI du noeud par rapport au graphe où elle est utilisé.

Les systèmes hypermédia qui nous servent d'exemple adoptent les schémas d'organisation de l'ACI qui sont développés pour les systèmes de fichiers classiques [Lew95, Ste95, Mau96] :

- Les listes de groupes sont spécifiés dans des fichiers suivant un ordre séquentiel.
- Les ACL énumèrent séquentiellement les utilisateurs et les groupes d'utilisateurs pouvant accéder à un objet et leurs droits d'accès.
- Enfin, on associe les ACL aux objets soit au cas par cas, soit suivant un ordre hiérarchique d'après l'arborescence du système des fichiers ou celle des URL.

Discussion sur ces travaux

Les schémas cités ci-dessus permettent d'associer l'ACI avec un objet ou avec un ensemble d'objets. Cependant, ces schémas ne fournissent aucun paramètre permettant d'identifier le contexte utilisé lors de la spécification de l'ACI. Par conséquent, il n'est pas possible de différencier l'ACI établie dans le contexte d'un graphe de celle d'un autre graphe. On ne peut pas donc partager des noeuds protégés sans éviter une ambiguïté dans la spécification de l'ACI.

4.2.4 Echange du défi d'autorisation

Selon la politique de sécurité d'un système hypermédia, un utilisateur peut être obligé de joindre à ses requêtes de l'ACI afin de pouvoir accéder aux objets. Par exemple, l'ACI pourrait être un mot de passe, une capacité définissant les groupes de l'utilisateur, un ticket Kerberos, etc. Lorsque plusieurs graphes incluent un même noeud, il se pose alors le problème du choix de l'ACI a envoyer dans le défi. Par exemple, pour le noeud B de la figure 4.2 page 65, faut-il demander l'ACI spécifié dans le contexte de $G1$ ou celui de $G2$?

On peut envoyer un défi à l'utilisateur soit au moyen d'un protocole de défi/réponse, soit en l'incluant dans les attributs des liens. Nous allons maintenant présenter chacune de ces techniques ainsi que leurs avantages et inconvénients dans le cas où un noeud est partagé par plusieurs graphes.

1.	client → serveur d'objets	:	requête
2.	serveur d'objets → client	:	erreur, défi
3.	client → serveur d'objets	:	requête, réponse

Fig 4.6: Protocole de contrôle d'accès par défi/réponse

4.2.4.1 Echange du défi au moyen d'un protocole de défi/réponse

Dans ce protocole, un utilisateur essaie d'abord d'accéder à un objet. Si l'objet est protégé et si l'utilisateur n'a pas inclut dans sa requête l'ACI qu'il faut, l'utilisateur reçoit au retour un *défi*. Celui-ci précise à l'utilisateur le type d'ACI qu'il doit présenter pour accéder à un objet protégé. L'utilisateur prépare ensuite une nouvelle requête comprenant la *réponse* au défi. Dans la pratique, un client memorise les défis qu'il récoit au cours de la session de consultation, ce qui permet de gagner une intéraction avec le serveur. Le client ne recevra alors de nouveaux défis que si ceux-ci expirent.

La spécification du protocole HTTP [BLFF96] propose la superposition du protocole défi/réponse à celui de consultation (fig. 4.6). Dans cette approche, on échange le défi à l'intérieur du message d'erreur qui indique le manque d'autorisation pour accéder l'objet. De même, on envoie la réponse au défi à l'intérieur de la requête de l'utilisateur. Cette superposition permet ainsi d'utiliser un protocole défi/réponse sans besoin de modifier les étapes du protocole de consultation.

Lors du partage de noeuds, cette technique requiert que le client indique au serveur quel graphe il est en train de consulter, ou bien que le serveur envoie au client les défis associés à chaque graphe pour que le client puisse en choisir un. Ce problème n'est pas encore traité par les protocoles défi/réponse existants. Une solution est d'imbriquer les défis à l'intérieur des liens pointant vers le noeud. Nous examinons cette technique dans la section suivante.

4.2.4.2 Echange du défi au moyen des attributs des liens

Dans cette technique, on inscrit le défi associé à un noeud dans les attributs des liens vers le noeud [AG95, RS96a, NS91]. Ainsi, dès qu'un client possède la liste des liens associés à un noeud, il peut connaître les défis associés aux noeuds destination.

Cette technique présente plusieurs avantages. D'abord, le défi pour accéder à un noeud peut être spécifié sans ambiguïté, indépendamment du partage du noeud par plusieurs graphes. Ensuite, on économise une transaction pour connaître le défi. Enfin, si le client évalue qu'il ne peut pas répondre au défi, on évite une transaction qui aurait fini par une erreur indiquant le manque d'autorisation.

Un inconvénient de cette technique est l'inscription des paramètres de sécurité à l'intérieur des liens. En principe, le temps de vie des composantes d'une hyperbase est très grande. D'autre part, le temps de vie d'un défi est nécessairement plus court, car le défi est sujet à la pérennité des clés : elles doivent être renouvellées pour ne pas être craquées. En ajoutant un attribut périmable aux liens, on sera éventuellement obligés de régénérer les liens lors d'un changement des techniques de protection. Cette opération est complexe si on a affaire à un grand nombre des liens ou si ceux-ci sont stockés sur des dispositifs non reinscriptibles, tels que des CD-ROM.

Un autre inconvénient est que le client doit d'abord connaître un des liens qui pointent vers un noeud protégé avant de pouvoir accéder celui-ci. Autrement, il ne pourra pas connaître quel est le défi à envoyer.

4.2.4.3 Discussion sur ces approches

Ces deux techniques présentent un conflit entre la flexiblité de mise à jour des défis et la facilité d'indiquer à un utilisateur le défi qu'il faut employer. Parmi ces deux techniques, seul l'échange du défi à l'intérieur des liens permet de prendre en compte le partage de noeuds. Cependant, l'imbrication du défi complique sa mise à jour. L'échange du défi au moyen d'un protocole permet une mise à jour aisée du défi. En contrepartie, cette technique nécessite une transaction additionnelle. De plus, la spécification de ce protocole ne prend pas encore en compte le partage de noeuds.

Les deux techniques que nous venons de décrire sont souvent complémentaires. En effet, lors du démarrage d'une session, on peut utiliser un protocole de défi/réponse pour connaître le défi associé à un noeud. Ensuite, les liens partant de ce noeud peuvent indiquer le défi associé aux noeuds destination.

Dans la pratique, la majorité des systèmes utilise exclusivement les protocoles défi/réponse. La technique pour l'échange du défi dans les attributs de liens n'existe que sous forme d'une proposition de norme Internet [RS96a].

4.3 Conclusion

Nous avons étudié dans ce chapitre quelques problèmes de base liés au contrôle d'accès dans les systèmes hypermédia répartis ainsi que les principales solutions à ces problèmes. Plus spécifiquement, nous avons examiné l'application du contrôle d'accès à la protection de l'information contenue dans une hyperbase.

Nous pouvons retenir de ce chapitre les points suivants :

- Les opérations de gestion des utilisateurs doivent s'appliquer d'une manière cohérente sur toutes les composantes d'un graphe.
- Les approches pour la localisation de l'information de contrôle d'accès dans le système présentent un conflit entre l'efficacité de la gestion des utilisateurs et l'efficacité de l'autorisation. Les approches réparties semblent offrir le meilleur compromis entre les deux critères d'efficacité.
- Bien que les systèmes étudiés supportent le partage des noeuds entre graphes, ils reflètent très mal ce partage au niveau de l'autorisation. Plus spécifiquement, l'information de contrôle d'accès n'est pas organisée en fonction des graphes où elle est définie. En outre, ce partage n'est pas non plus pris en compte lors de l'envoi d'un défi d'autorisation vers l'utilisateur.

Dans le chapitre suivant, nous décrivons un modèle d'autorisation fondée sur des informations de contrôle d'accès non nominatives. Ce modèle augmente l'efficacité des approches réparties et permet de prendre en compte le partage des noeuds.

Deuxième partie

Proposition et expérimentation d'un modèle de contrôle d'accès réparti non nominatif

Dans cette partie, nous proposons un nouveau modèle de contrôle d'accès réparti et non nominatif. Le modèle se base sur les propriétés des graphes hypermédia pour établir de l'information de contrôle d'accès (ACI) non nominative qui est stockée dans les serveurs du système. Un serveur d'autorisation est responsable d'autoriser l'accès global du client au graphe. Le client récupère auprès de ce serveur une capacité qui lui permet de consulter les serveurs du domaine. La combinaison de l'ACI stockée dans la capacité et dans les serveurs, permet à ces derniers d'autoriser les requêtes du client localement, sans besoin de consulter aucun autre serveur.

Cette stratégie permet d'améliorer l'efficacité de la gestion des utilisateurs sans nuire à l'efficacité de l'autorisation. De plus, elle permet de distinguer directement l'appartenance d'un noeud à un graphe, ce qui permet de prendre en compte le partage de noeuds.

Dans le chapitre 5, nous développons le modèle. Ensuite, dans le chapitre 6, nous présentons un prototype mettant en oeuvre le modèle ainsi qu'une évaluation quantitative de cette mise en oeuvre.

Chapitre 5

Un nouveau modèle de contrôle d'accès non nominatif

Ah! Si j'avais un franc cinquante
j'aurais bientôt deux francs cinquante
Ah! Si j'avais deux francs cinquante
j'aurais bientôt trois francs cinquante

Boris Vian
(*Si j'avais un franc cinquante*)

Dans ce chapitre, nous proposons un modèle de contrôle d'accès conçu pour l'utilisation des informations de contrôle d'accès non nominatives. [1]

Après une présentation des critères de conception du modèle, nous développons le modèle. Nous présentons ensuite une brève évaluation qualitative de ses propriétés. Enfin, nous positionnons notre proposition par rapport à quelques travaux apparentés.

5.1 Critères de conception du modèle

Nous avons abordé dans le chapitre précédent la nécessité d'appliquer les opérations de gestion des utilisateurs d'une manière cohérente sur tous les composantes d'un graphe hypermédia. Nous avons également étudié les principaux modèles et approches de contrôle d'accès pour les systèmes hypermédia répartis.

Dans cette section, nous présentons d'abord les motivations pour le développement d'un nouveau modèle de contrôle d'accès. Nous décrivons ensuite l'ensemble des objectifs que nous espérons atteindre avec le modèle.

1. Des versions préliminaires du modèle ont été publiées dans [Kah94, Kah95a, Kah95b].

5.1.1 Pourquoi un nouveau modèle de contrôle d'accès non nominatif ?

Dans le chapitre 3, nous avons vu que dans un système hypermédia, une session se caractérise par la consultation d'un ou plusieurs graphes hypermédias. Un graphe est constitué de plusieurs noeuds. L'utilisateur récupère ces noeuds lors du parcours du graphe. L'unité d'accès dans une session est donc un graphe hypermédia et la consultation de celui-ci implique la consultation de plusieurs noeuds.

Nous avons étudié plusieurs approches pour le contrôle d'accès dans les systèmes hypermédia répartis. Une caractéristique en commun dans ces approches est la spécification des droits d'accès des utilisateurs au moyen des listes de contrôle d'accès (ACL) classiques. Tout noeud dans le domaine est associé à une ACL. Les entrées de l'ACL spécifient le nom d'un utilisateur ou d'un groupe d'utilisateurs et leurs droits d'accès sur le noeud.

Ces approches pour la specification des droits sont du type *nominative* : on spécifie pour chaque noeud les droits d'accès de chaque utilisateur ou groupe des utilisateurs. L'utilisation des droits d'accès nominatifs et des ACL classiques présentent deux grands inconvenients :

- *Gestion des utilisateurs.* La mise à jour des droits d'accès d'un utilisateur sur un graphe implique la mise à jour de toutes les ACL associées aux noeuds du graphe.
- *Partage des noeuds.* Les entrées des ACL ne distinguent pas le contexte du graphe où les droits ont été déclarés. Supposons qu'un noeud est inclu par deux graphes et qu'un utilisateur a l'autorisation de consulter les deux graphes. Si on désire révoquer les droits d'accès de l'utilisateur sur un des graphes, on est obligé soit de révoquer tous ses droits d'accès sur le noeud, soit de laisser les droits dans l'état courant, les ACL ne comportant pas la notion de graphe.

Notons que ces deux problèmes sont indépendants de la localisation des ACL dans le domaine par rapport à celles des noeuds. En effet, ils concernent l'association des droits d'accès aux noeuds par rapport à la structure des graphes et leur organisation interne.

Dans un graphe hypermédia, les relations qui existent entre les noeuds du graphe sont généralement constantes. Afin de résoudre les problèmes cités ci-dessus, nous voulons étendre cette idée de relations constantes aux droits d'accès associés aux noeuds d'un graphe.

Il s'agit d'établir pour chaque graphe l'ensemble de droits d'accès nécessaires pour consulter les composantes du graphe. Cette ensemble de droits est alors dépendant de la structure du graphe et indépendant des utilisateurs et, donc, non nominatif.

L'utilisation des droits d'accès non nominatifs ne constitue pas encore une solution complète. Il faut encore pouvoir associer les droits d'accès aux noeuds, pouvoir les organiser par graphes et pouvoir mettre un utilisateur en relation avec ces droits. Nous proposons de suivre une approche répartie de localisation de l'ACI inspirée de

DCE/Web [LZ94, Lew95] et combinée avec une nominalisation des graphes. Nous associons un nom à chaque graphe dans le domaine. Chaque noeud dans le système est associé à une ACL dont les entrées spécifient le nom des graphes où le noeud est inclus et les droits d'accès pour chaque graphe. Un serveur d'autorisation est responsable de fournir aux utilisateurs des capacités mettant en relation les utilisateurs avec les droits d'accès accordés aux graphes.

La stratégie que nous venons d'évoquer a simplifié la gestion des utilisateurs car les serveurs hypermédia n'ont plus besoin de connaître *a prior*i leurs clients potentiels. De plus, l'utilisation des noms de graphes à l'intérieur des ACL implique une organisation des droits d'accès par des graphes. Il est possible donc de partager un noeud entre plusieurs graphes et de distinguer les droits d'accès spécifiques à chaque graphe.

Dans la suite de cette section, nous présentons les objectifs que nous voulons répondre avec le modèle. Nous divisons cette présentation en deux parties :

- les objectifs de sécurité,
- les objectifs de compatibilité fonctionelle avec les systèmes hypermédia existants.

5.1.2 Objectifs de sécurité

Nous nous sommes fixés comme objectifs de sécurité la prise en compte des besoins de base identifiés dans le chapitre 4 et le respect des contraintes d'efficacité :

- *Protection des noeuds.* Le contrôle d'accès dans un système hypermédia concerne les noeuds et les liens. Nous avons choisi de limiter nos travaux à la protection des noeuds, la protection de liens étant trop dépendante des instanciations du modèle de données hypermédia.
- *Cohérence des opérations de gestion des droits d'accès.* Un utilisateur consulte un système hypermédia en parcourant les noeuds qui composent un ou plusieurs graphes. Les droits d'accès des utilisateurs doivent être d'abord spécifiés à la granularité d'un graphe et ensuite être appliqués à toutes les composantes du graphe.
- *Partage de noeuds.* Si deux ou plusieurs graphes partagent un noeud, il doit être possible de différencier l'ACI spécifique à chaque graphe.
- *Efficacité de l'autorisation.* Afin de ne pas nuire à l'efficacité de l'autorisation, chaque serveur doit pouvoir autoriser un accès indépendamment des autres serveurs du domaine.
- *Efficacité de la gestion des utilisateurs.* Afin de diminuer le nombre des serveurs impliqués par une opération de gestion des utilisateurs, les serveurs du système ne doivent stocker que de l'information de contrôle d'accès non nominative. La seule exception est le serveur d'autorisation responsable de mettre en relation un utilisateur avec les droits d'accès accordés à un graphe.

En plus des besoins déja cités, nous avons voulu imposer au modèle le principe du moindre privilège [SS75], un des principes de bases de la conception d'un système sécurisé :

- *Application du principe du moindre privilège.* Le principe indique qu'un utilisateur ne doit pas recevoir plus de privilèges que ce qui lui faut pour effectuer une opération. Le modèle doit pouvoir adapter ce principe aux caractéristiques d'une session de consultation : un utilisateur n'a des droits qu'aux composantes des graphes qu'il peut consulter dans une session.

5.1.3 Objectifs de compatibilité

Aujourd'hui, on trouve plusieurs systèmes hypermédia. Il est donc important de pouvoir appliquer notre modèle de contrôle d'accès sur un système existant de la manière la plus transparente possible. Afin de faciliter cette intégration, nous avons fixé deux contraintes au modèle :

- *Intégration simplifiée avec les protocoles de consultation existants.* Les protocoles d'autorisation doivent pouvoir s'imbriquer facilement dans les protocoles de consultation spécifiés pour les systèmes hypermédia. Dans plusieurs cas, comme par exemple celui du WWW, les protocoles hypermédia sont très simples, se limitant à l'échange d'une requête et d'une réponse par connexion. Le modèle de contrôle d'accès doit éviter de compliquer ces protocoles.

- *Compatibilité en arrière et en avant avec les noeuds existants.* Il doit être possible de contrôler l'accès aux noeuds existants dans un système hypermédia sans besoin de les modifier. Dans plusieurs cas, l'information stockée dans un système hypermédia réparti peut avoir un temps de vie supérieure à celui des techniques utilisées pour contrôler l'accès des utilisateurs. Le modèle de contrôle d'accès doit pouvoir s'appliquer sur des noeuds existant. De plus, le modèle doit éviter toute modification de l'information qui peut affecter sa pérénité ou représenter un investissement en temps important lors de l'adoption des nouveaux modèles de contrôle d'accès.

5.2 Présentation générale du modèle

Dans cette section, nous montrons comment les techniques présentées dans la première partie peuvent s'assembler pour constituer un modèle de contrôle d'accès satisfaisant les objectifs de cette étude. Après une présentation des hypothèses de base, nous décrivons les principaux concepts du modèle.

5.2.1 Hypothèses de base

Les fondements choisis pour concevoir le modèle reposent sur plusieurs hypothèses à partir desquelles des propriétés caractéristiques du modèle peuvent être déclinées. Cette section énonce les plus importantes de ces propriétés.

On suppose que dans un système hypermédia :

- les serveurs du domaine sont regroupés dans des *domaines d'autorisation* ;
- les serveurs du domaine peuvent se synchroniser faiblement par rapport à un service d'horloge global de confiance ;
- tout noeud et tout graphe a un identificateur unique dans le domaine ;
- tout serveur dans le domaine a une paire de clés asymétriques ;
- un domaine inclut un administrateur de sécurité et un serveur d'autorisation. Ceux-ci possèdent une paire de clés asymétriques dont les clés publiques sont connues de tous les serveurs du domaine ;
- tous les serveurs du domaine ont accès à un mécanisme permettant la génération et la validation des signatures numériques ;
- l'authentification des utilisateurs peut s'effectuer au moyen d'un protocole à clé publique ou au moyen de l'adresse réseau de la station de travail de l'utilisateur.

Parmi ces hypothèses, seule celle de l'horloge globale peut sembler irréaliste à première vue. Or, plusieurs solutions existent à ce problème. Par exemple, Lam et Beth [LB92] proposent une série de protocoles de sécurité permettant d'avoir une horloge globale de confiance dans un réseau. Plusieurs systèmes répartis utilisent déjà un système d'horloge synchronisé globalement et sécurisé. C'est le cas de DCE [RKF92] et d'AFS [Tra]. Le premier prévoit une synchronization matériel des horloges du domaine au moyen des signaux hertziens, le second utilise un mécanisme de synchronisation logiciel.

5.2.2 Principe de protection

Notre modèle est basé sur des droits d'accès non nominatifs établis à partir des propriétés des graphes hypermédia. Un système de protection à deux niveaux permet ensuite de mettre en relation un utilisateur avec ces droits. Avant de détailler le principe de protection du modèle, rappelons tout d'abord les besoins pour l'attribution des droits d'accès.

Une session de consultation hypermédia se caractérise par le parcours d'un ou plusieurs graphes. Comme un graphe est formé de plusieurs noeuds, l'utilisateur doit avoir un droit d'accès sur les noeuds afin de pouvoir parcourir le graphe.

Notre approche de protection consiste à nommer un graphe et à accorder à cette utilisateur "graphe" des droits d'accès sur chacun de ces noeuds. On choisit les droits d'accès d'après les opérations nécessaires pour la consultation du graphe. Si un noeud est inclu dans plusieurs graphes, chaque graphe aura un ensemble de droits d'accès distincts sur le noeud.

Les informations de contrôle d'accès que l'on associe aux noeuds sont non nominatives : elles n'identifient aucun utilisateur ou groupe d'utilisteurs. On met en relation un utilisateur avec les droits d'accès associés aux graphes au moyen des capacités.

Plus formellement, dans le modèle, on organise les serveurs classiques d'un système hypermédia dans des *domaines d'autorisation*. Un domaine est composé d'un ensemble de serveurs classiques hypermédia, d'un *serveur d'autorisation* (SA) et d'un *administrateur de sécurité* (AS) (fig. 5.1).

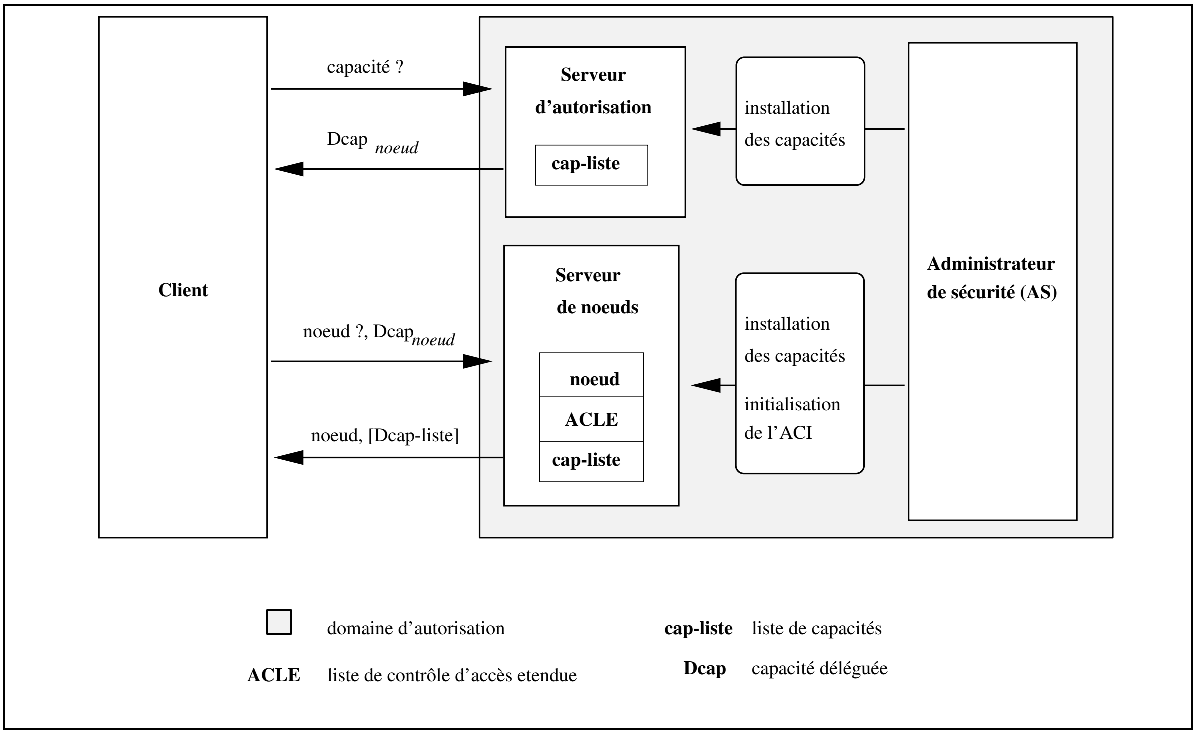


Fig 5.1: Aperçu général du modèle proposée

Dans un domaine, on emploie deux niveaux de protection :

– une protection globale par le serveur d'autorisation,

– une protection locale sur chaque site du système.

Le *serveur d'autorisation* est responsable d'autoriser l'accès d'un client à un graphe. Il fournit au client une capacité lui permettant de démarrer sa session de consultation. Cette capacité spécifie le nom du graphe.

Un *serveur de noeuds* est responsable d'autoriser l'accès aux noeuds qu'il héberge. On associe à chaque noeud du serveur une *liste de contrôle d'accès étendue* (ACLE). Celle-ci spécifie les noms des graphes où un noeud est inclu et les droits d'accès valables pour chaque graphe. Lorsqu'un client présente une capacité qui spécifie le nom d'un graphe, il acquiert les droits d'accès associés au graphe.

L'*administrateur de sécurité* (AS) est une entité active et hors ligne. Il est responsable de la génération des capacités et de l'administration des ACLE du domaine.

Notons qu'un serveur de noeuds n'a pas besoin de connaître *a priori* ses clients potentiels. Le serveur doit uniquement valider la capacité présentée et vérifier que l'opération demandée dans la requête est comprise dans les droits d'accès accordés au graphe. Ainsi la politique d'autorisation des serveurs de noeuds est d'autoriser toute requête accompagnée d'une capacité appropriée.

Deux approches pour l'assignation et pour l'utilisation des capacités

Jusqu'à maintenant, nous avons utilisé la propriété d'appartenance d'un noeud à un graphe pour assigner les droits d'accès d'un graphe sur ses composantes. Nous allons maintenant décrire l'utilisation de cette propriété et celle d'interconnexion des noeuds par des liens pour l'assignation et pour l'utilisation des capacités. Plus spécifiquement, nous proposons deux approches :

- l'approche *une capacité par noeud* (CN), basée sur l'appartenance d'un noeud à un graphe et sur les liens au départ de ce noeud ;
- l'approche *une capacité par graphe* (CG), basée sur l'appartenance d'un noeud à un graphe.

Dans l'approche **CN**, une capacité identifie un graphe et un noeud, et ne donne accès qu'à ce noeud. Un client récupère les capacités qu'il lui faut pour consulter un graphe auprès du SA et des serveurs de noeuds. A chaque noeud est associéé une ACLE. De plus, en fonction de la structure du graphe, on associe aux noeuds des *listes de capacités* donnant accès à d'autres noeuds. Au début de la session de consultation, le client récupère une capacité pour le *noeud racine* d'un graphe auprès du serveur d'autorisation. Lors du parcours d'un noeud associé à une liste de capacités, le client récupère en même temps le noeud et la liste de capacités associée au graphe. Le client utilise alors ces capacités pour accéder aux autres noeuds.

Dans l'approche **CG**, une capacité ne spécifie que le nom d'un graphe. Chaque noeud est associé à une ACLE. Le client récupère d'abord une capacité pour un graphe auprès du serveur d'autorisation. Dans la suite, le client peut accéder à n'importe quel noeud appartenant au graphe. Cette approche est similairê à l'utilisation de groupes dans DCE/Web [LZ94].

5.2.3 Résumé des propriétés du modèle

En résumé, le modèle de contrôle d'accès que nous proposons est basé sur les propriétés suivantes :

- *Domaines d'autorisation.* On regroupe les serveurs classiques hypermédia dans des domaines d'autorisation. De plus, un domaine comporte un *administrateur de sécurité* (AS) et un *serveur d'autorisation.* L'accès aux noeuds dans le domaine est conditionné à la présentation d'une capacité appropriée.

- *Etablissement des droits d'accès.* Notre approche consiste à nommer les graphes et à assigner à ce nom les droits d'accès sur les composantes du graphe. On suppose que les droits d'accès lors de la consultation d'un graphe sont les mêmes pour tous les utilisateurs.

- *Protection globale et protection locale.* On distingue deux niveaux de protection : une *protection globale* par le SA et une *protection locale* par les serveurs de noeuds. Le SA autorise un utilisateur à consulter un graphe en lui octroyant une capacité comprenant le nom du graphe. Les serveurs de noeuds sont responsables d'autoriser l'accès aux composantes des graphes. Dans un serveur de noeuds, on associe à chaque noeud une *liste de contrôle d'accès étendue* (ACLE). Celle-ci spécifie le nom des graphes qui incluent le noeud ainsi que les droits d'accès valables pour chaque graphe.

- *Affectation et utilisation des capacités.* On s'inspire également de la structure de graphes pour définir deux approches de désignation et d'utilisation des capacités : l'approche une *capacité par noeud* (CN) et l'approche une *capacité par graphe* (CG).

La suite de ce chapitre est organisée autour du cycle de vie des capacités. Dans la section 5.3, nous décrivons la structure des capacités. La section 5.4 traite de l'affectation et l'utilisation des capacités dans les deux approches. La section 5.5 examine plusieurs stratégies pour la révocation des capacités. Enfin, la section 5.7 présente une discussion générale.

Le tableau suivant donne la notation employée dans la suite :

Notation	**Signification**
ACLE	Liste de Contrôle d'Accès Etendue
AS	Administrateur de Sécurité
$\text{cap}_{objet\ o}$	une capacité donnant accès à l'objet o
$\text{Dcap}_{objet\ o}$	une capacité déléguée donnant accès à l'objet o
$\{données\}$	un ensemble de données
$\text{KU}_{principal}$	la clé publique d'un principal
SA	Serveur d'Autorisation
$\text{SIG}_{principal}$	la signature numérique d'un message par un principal
$\text{UID}_{principal}$	L'identificateur unique d'un principal

5.3 Propriétés des capacités

Le modèle utilise des capacités pour mettre en relation un utilisateur avec les droits d'accès associés à un graphe. Nous utilisons trois types de capacités :

– Des *capacités*, qui identifient un graphe et, éventuellement, un noeud ;

– Des *capacités déléguées*, utilisées pour le transfert des capacités entre clients et serveurs ;

– Des *autorisateurs*, qui spécifient l'identité des clients pouvant utiliser les capacités déléguées.

Nous allons dans un premier temps décrire les choix généraux de conception et ensuite décrire plus en détail la structure des capacités.

5.3.1 Choix de conception

Cette section présente les choix de conception concernant la structure, la génération et l'échange de capacités dans le modèle.

Comme nous l'avons vu dans la section 2.3.2, la protection d'une capacité requiert la qualification de l'objet cible, l'identification du propriétaire de la capacité et la signature de la capacité par son expéditeur. Similairement, la validation d'une capacité requiert entre autres opérations, l'authentification du propriétaire et la vérification de la signature de l'expéditeur.

Une première approche à la protection des capacités est de permettre aux serveurs de générer leurs propres capacités et de les octroyer aux clients [TMvR86]. Cependant, dans le contexte de notre modèle, un serveur doit pouvoir octroyer des capacités donnant accès à un autre serveur. Cette caractéristique nous a fait abandonner cette première approche car elle introduit plusieurs problèmes de sécurité. Tout d'abord, il devient impossible de contrôler le contenu des capacités. Ensuite, la validation des signatures et la révocation des capacités deviennent trop complexe à mettre en oeuvre. Enfin, l'infiltration d'un serveur peut donner à l'intrus la possibilité de falsifier des capacités lui donnant accès à tous les autres serveurs du domaine.

La solution que nous avons retenue est composé de plusieurs techniques :

– utilisation des capacités *auto identifiables* [LC94],

– génération hors ligne et centralisée et des capacités par l'AS,

– transfert des capacités vers les clients par *délégation*,

– authentification des clients au moyen d'un *autorisateur* généré par le SA.

5.3.1.1 Capacités auto identifiables

Dans la section 2.3.2.3, pg. 39, nous avons vu que la protection contre le vol d'une capacité ou contre son utilisation non autorisée requiert l'inclusion de l'identificateur du propriétaire de la capacité dans celle-ci. En outre, la protection de l'intégrité de la capacité ainsi que l'authentification de l'origine de celle-ci requièrent la signature de la capacité par son expéditeur. Afin de mettre en place ces deux techniques de protection, il faut qu'au moment de la validation de la capacité, le principal validateur ait accès à des données lui permettant d'authentifier le propriétaire de la capacite et de vérifier les signatures de l'expéditeur.

Comme nous l'avons signalé lors de la présentation des objectifs de sécurité du modèle, il est important de minimiser le nombre de serveurs devant être contactés pour autoriser un accès. Par extension, il est important de minimiser le nombre de serveurs devant être contactés pour valider une capacité.

La solution que nous avons retenue pour répondre à ces contraintes est une adaptation des travaux de Low et Christianson [LC94] sur les capacités auto identifiables. Une capacité auto identifiable fournit tous les paramètres permettant à un principal de vérifier une capacité sans besoin de faire appel à une autorité.

Les travaux de Low et Christianson sont basés sur des algorithmes de chiffrement à clé publique. On suppose que chaque principal dans un domaine possède une paire de clés asymétriques (privée et publique). De plus, la clé publique existe sous forme d'un certificat de clé publique généré par une autorité de certification (cf. section 2.2.4.4, pg. 30).

L'idée de base est d'identifier le propriétaire et l'expéditeur d'une capacité au moyen de leurs certificats de clé publique. On inscrit à l'intérieur de la capacité des références aux certificats de clé publique de l'expéditeur et du propriétaire. Lors de la génération des capacités, l'expéditeur signe toutes les capacités qu'il produit avec sa clé privée. Enfin, le propriétaire utilise sa clé privée pour s'authentifier lors de la présentation de la capacité, par exemple, en signant toutes les requêtes qui incluent la capacité.

La capacité est auto identifiable dans la mesure où elle identifie son expéditeur et son propriétaire. N'importe quel principal peut alors vérifier une capacité sans besoin de contacter l'expéditeur de la capacité. Il suffit qu'il récupère les certificats de clé publique nécessaires et qu'il connaisse la clé publique de l'autorité de certification. Il pourra alors vérifier d'abord les certificats de clé publique et ensuite, les signatures du propriétaire et de l'expéditeur.

5.3.1.2 Génération des capacités

L'AS est une entité hors ligne responsable de la génération et de l'installation de toutes les capacités du domaine. En diminuant le nombre des principaux pouvant générer une capacité, on simplifie la vérification de la signature de l'expéditeur d'une capacité. Un serveur octroie une capacité à un client au moyen d'une opération de délégation, que nous décrivons au paragraphe suivant.

5.3.1.3 Délégation des capacités et utilisation de l'autorisateur

La délégation est un mécanisme permettant à un principal de passer un ensemble de ses droits d'accès sur un objet à un autre principal. Dans notre modèle, nous utilisons un mécanisme de délégation de capacités pour le transfert des capacités d'un serveur vers un client. Nous employons une technique de délégation consistant en la génération d'une nouvelle capacité, dite *capacité déléguée* ou *proxy*, contenant les droits d'accès délégués [VAB91, Neu93, LC94]. Nous avons également introduit le concept d'*autorisateur* dans cette technique afin de simplifier la validation et contrôle des capacites déléguées.

Une capacité déléguée classique est composée des droits d'accès délégués, de l'identité d'un principal, de l'identité du délégateur et de la signature du délégateur. L'identité du principal indique au serveur sous quelle identité un principal doit s'authentifier pour pouvoir utiliser la capacité déléguée. L'identité du délégateur indique quel principal a délégué ses droits. Enfin, la signature du délégateur permet au serveur de noeuds d'authentifier la capacité ainsi que le délégateur.

Dans notre modèle, on interdit à un serveur de noeuds de choisir lui-même l'identité du client où la date d'expiration de la capacité déléguée. La motivation pour ce choix est de diminuer les risques provenant de l'infiltration ou de la défaillance d'un serveur de noeuds. Par exemple, le serveur pourrait déléguer une capacité pour une période indéterminée[2] ou déléguer une capacité à un client non autorisée. Afin de résoudre ce problème, nous avons dévéloppé le concept d'*autorisateur*.

Un *autorisateur* est une capacité spéciale générée par le serveur d'autorisation. On l'utilise pour identifier le propriétaire d'une capacité déléguée. Un autorisateur donne essentiellement l'identité du client, sous forme d'une adresse réseau ou d'une clé publique, et le temps d'expiration de la session de consultation.[3] Il est protégé par la signature du serveur d'autorisation. Toute capacité déléguée doit inclure un autorisateur.

Supposons que le client utilise une capacité déléguée pour consulter un serveur de noeuds. Si le serveur doit alors déléguer une autre capacité au client, le serveur réutilise l'autorisateur de la capacité présentée par le client et l'inclut dans la nouvelle capacité déleguée.

En imposant l'utilisation de l'autorisateur dans toute capacité déléguée, nous pouvons empêcher la délégation incontrôlée des capacités par les serveurs. De plus, il permet d'imposer un temps déterminé à une session de consultation.

Comme pour les capacités, une capacité déléguée a la propriété d'être auto identifiable : elle fournit tous les paramètres permettant à un serveur de vérifier l'authenticité de la capacité et les identités du propriétaire et de l'expéditeur sans besoin de contacter ce dernier.

2. Comme nous verrons dans la section 5.3.2, toutes les capacités incluent une période de vie. Ainsi, la capacité déléguée expirerait en même temps que la capacité.

3. Nous décrivons plus en détail les autorisateurs dans la section 5.3.4.

5.3.2 Capacités de base

Nous présentons dans cette section la structure d'une capacité de base. Cette structure est dérivée de celle proposée dans la norme ECMA-138 [ECM89][4] pour la protection des capacités et des travaux de Low et Christianson [LC94] pour l'utilisation des capacités auto identifiables. En particulier, nous avons défini des nouveaux attributs pour qualifier l'objet cible.

Une capacité de base, ou tout simplement capacité, identifie un graphe (dans l'approche **CN**) ou un noeud d'un graphe (dans l'approche **CG**). Lors de la présentation de la capacité, un serveur de noeuds utilise le nom du graphe inscrit dans la capacité pour trouver les droits d'accès attribués au graphe.

Un utilisateur acquiert une capacité de base par indirection. L'AS est responsable de la génération des capacités et de leur installation dans les serveurs du système. Les serveurs du système possédant des capacités les octroient à l'utilisateur au moyen d'une opération de délégation. Par abus de langage, nous appelons *délégateur* ce serveur.

Attributs	**Description**
Type	Type de la capacité
URI_{noeud}	Identificateur unique du noeud
Epoque_{noeud}	Epoque du noeud
UID_{graphe}	Identificateur unique du graphe
KU_{deleg}	Clé publique du serveur propriétaire la capacité
UID_{cap}	Identificateur unique de la capacité
Validité	Temps de vie de la capacité
UID_{AS}	Identificateur unique de l'administrateur de sécurité
Sig_{AS}	Signature de la capacité par l'administrateur de sécurité

Tab 5.1: Structure des capacités

Le tableau 5.1 montre la structure utilisée pour représenter une capacité. Celle-ci comporte quatre classes d'attributs :

- l'identificateur du type de la capacité,
- des attributs qualificateurs de l'objet cible,
- l'identificateur du serveur propriétaire de la capacité,
- des attributs pour la protection de la capacité.

Nous allons maintenant décrire plus en détail la fonction de chacun de ces attributs.

4. cf. section 2.3.2.3, pg. 39.

5.3.2.1 Type de la capacité

Dans notre modèle, nous utilisons deux types de capacités : les capacités donnant accès à un noeud et les capacités donnant accès à un graphe. On utilise une seule structure pour représenter les deux types de capacités. Une des raisons pour ce choix est de permettre la réutilisation des mécanismes de validation des capacités. La valeur de l'attribut *Type* permet de différencier l'un ou l'autre type de capacités.

5.3.2.2 Attributs qualificateurs de l'objet cible

Dans notre modèle, nous voulons utiliser une capacité pour donner accès à un noeud appartenant à un graphe. Dans l'approche **CN**, la capacité ne peut être utilisée qu'avec un seul noeud. Dans l'approche **CG**, la capacité donne accès à tous les noeuds du graphe.

Notre proposition consiste à identifier un objet cible dans l'approche **CN** en combinant les identificateurs uniques du noeud et du graphe. Similairement, nous utilisons l'identificateur unique du graphe pour identifier l'objet cible dans l'approche **CG**.

Un deuxième choix que nous faisons est l'association des *époques* aux identificateurs de noeuds et de graphes. Une époque est un compteur qui identifie une instance donnée dans le temps. On utilise les époques notamment pour pouvoir révoquer les capacités donnant accès à un objet sans avoir besoin de modifier les autres paramètres qui identifient l'objet.

Nous prenons en compte les contraintes citées ci-dessus en caractérisant un noeud au moyen de quatre attributs :

- l'*URI du noeud.* Il s'agit de l'identificateur d'un noeud dans le système. Il peut être soit une URN ou une URL ;
- l'*époque du noeud.* Il identifie une instance donnée du noeud dans le temps ;
- l'*UID graphe.* Il identifie le graphe d'appartenance du noeud.
- l'*époque du graphe.* Il identifie une instance donnée du graphe dans le temps.

On trouve deux copies de ces attributs, une associée à un noeud, et une autre inscrite dans les capacités donnant accès au noeud. Afin de pouvoir accéder a un noeud, il doit exister une correspondance entre les deux ensembles d'attributs.

Les attributs *URI noeud* et *Epoque noeud* ne sont significatifs que dans l'approche **CN** (en effet, pour **CG**, on utilise une seule capacité pour accéder à tous les noeuds d'un graphe).

5.3.2.3 Attributs pour la protection des capacités

Afin de se prémunir contre le vol et l'utilisation non autorisée des capacités, ECMA [ECM89] conseille l'inclusion d'un attribut identifiant le propriétaire des capacités. On peut alors éviter ces deux menaces en exigeant l'authentification du propriétaire lors de la présentation de la capacité. Low et Christianson [LC94] proposent l'utilisation des clés publiques pour identifier les propriétaires. Ainsi, la capacité fournit au serveur toutes les données nécessaires pour identifier et pour authentifier le propriétaire.

Nous avons retenu ce choix sous forme de l'attribut KU_{deleg}. La valeur de cet attribut donne la clé publique du serveur de noeuds qui héberge la capacité. Comme nous verrons dans la section suivante, la délégation d'une capacité consiste en l'inclusion d'une capacité et d'un autorisateur, le tout signé par la clé privée du délégateur. L'attribut KU_{deleg} permet aux autres serveurs de vérifier cette signature et, par conséquent, de prouver l'intégrité et l'authenticité d'une capacité déléguée.

Les deux attributs suivants, UID_{cap} et *Validité* sont utilisés par les mécanismes de révocation des capacités.[5] L'attribut UID_{cap} identifie de manière unique une capacité dans le domaine. Par exemple, cette valeur peut être un numéro de série qui augmente lors de la création de chaque nouvelle capacité. L'attribut *Validité* spécifie l'intervalle de temps dans lequel la capacité est valable.

Enfin, les attributs $UID_{KU_{AS}}$ et Sig_{AS} sont utilisés pour la protection de la capacité contre sa falsification et sa modification non autorisées. Comme nous l'avons indiqué précédemment, l'administrateur de sécurité est la seule entité pouvant générer des capacités dans le domaine. La signature des champs de la capacité par le serveur garantit l'intégrité des données contenues dans la capacité.

Au lieu d'inclure la clé publique de l'administrateur de sécurité, on utilise l'identificateur unique de l'AS. En effet, on suppose que tous les serveurs dans le domaine connaissent la clé publique de l'AS. Il n'est pas donc nécessaire de dupliquer cette information dans toutes les capacités. L'identificateur unique fournit un index permettant aux serveurs de trouver la clé publique de l'AS. Ce choix permet également de garantir que lors du changement de la clé publique de l'AS, toutes les capacités existantes deviendront invalides : il sera alors impossible de vérifier la signature avec une autre clé.

Notons que les capacités sont non nominatives dans la mesure où elles n'identifient pas les clients. En effet, une capacité caractérise une cible et identifie le serveur qui stocke la capacité.

Attributs	**Description**
capacité	cf. section 5.3.2
autorisateur	cf. section 5.3.4
Sig délégateur	Signature de la capacité par un serveur utilisant la clé spécifiée par l'attribut KU_{deleg} de la capacité

Tab 5.2: Structure des capacités déléguées

5. Nous décrivons ces mécanismes dans la section 5.5.

5.3.3 Capacités déléguées

La délégation est une opération permettant d'octroyer à un principal une capacité. Cette opération consiste en la création d'une nouvelle capacité qui inclut une capacité de base, un *autorisateur* et la signature de l'ensemble par le délégateur. Le tableau 5.2 illustre ces champs.

Afin de diminuer les conséquences suite à l'infiltration d'un serveur de noeuds, on contraint le serveur de noeuds à inclure un autorisateur dans toute capacité déléguée. L'autorisateur, que nous décrirons plus en détail dans la section suivante, est une capacité générée par le SA. Il identifie le client, le graphe qu'il peut consulter et la période de validité de la session. La période de validité est également utilisée comme date d'expiration de la capacité.

Le serveur de noeuds récupère d'abord l'autorisateur dans la capacité déléguée que le client a dû présenter pour accéder au noeud. Le serveur inclut ensuite l'autorisateur dans toutes les capacités déléguées qu'il fournira au client. Pour qu'une capacité déléguée soit valide, le nom du graphe inscrit dans la capacité doit être le même que celui inscrit dans l'autorisateur. Ainsi, un serveur de noeuds est contraint à la délégation des capacités associées au graphe que l'utilisateur est autorisé à consulter.

La clé privée que le serveur utilise pour signer la capacité doit correspondre à la clé publique spécifiée dans l'attribut KU_{deleg} de la capacité. Dans la suite, n'importe quel serveur peut vérifier la signature d'une capacité déléguée sans besoin de connaître *a priori* le délégateur.

Attributs	**Description**
UID$_{client}$	Identificateur unique du client
UID$_{graphe}$	Identificateur unique du graphe que le client est autorisé à consulter
Epoque$_{graphe}$	Epoque du graphe
[privilèges]	Un ensemble de privilèges de sécurité
UID$_{autorisateur}$	Identificateur unique de l'autorisateur
Validité	Période de validité de l'autorisateur
UID$_{SA}$	Identificateur unique du serveur d'autorisation
Sig$_{SA}$	Signature de l'autorisateur par le serveur d'autorisation

Tab 5.3: Structure de l'autorisateur

5.3.4 Autorisateur

Il s'agit d'une capacité définie pour les besoins du modèle et qui est imbriquée à l'intérieur des capacités déléguées. L'autorisateur identifie un client et spécifie le contexte d'une session de consultation. Le tableau 5.3 en montre la structure.

L'autorisateur est généré par le SA lorsque le client lui demande une capacité pour commencer une session de consultation. Le SA inclut dans l'autorisateur l'identité du client, l'identificateur du graphe que le client peut consulter, l'époque courante du graphe et le temps alloué à sa session de consultation. Tout autorisateur possède également un identificateur unique. La signature de l'autorisateur par le SA garantit son authenticité et son intégrité. La clé publique du SA permet de vérifier cette signature. Comme pour l'AS, on suppose que la clé publique du SA est connue de tous les serveurs du domaine. Donc, nous n'incluons dans l'autorisateur que le nom du SA pour aider aux serveurs à repérer la clé publique du SA.

Notons que l'époque des graphes est inclue dans l'autorisateur, plutôt que dans les capacités de base. La motivation pour ce choix est de permettre l'évolution de cette valeur sans avoir besoin de régénérer toutes les capacités appartenant à un graphe. En effet, tout changement de l'époque du graphe n'affecte que les capacités déléguées.

L'autorisateur est également une capacité auto-identifiable. L'attribut UID_{client} donne l'identité sous laquelle le client doit s'authentifier lors de la présentation des capacités déléguées. Cet attribut permet à un serveur de noeuds d'authentifier un client sans besoin de le connaître a priori. Comme nous verrons dans la section 5.4.2.2, cet attribut peut prendre comme valeurs soit une clé publique, soit ou une adresse réseau.

D'après le contexte d'utilisation du modèle, le SA peut ajouter d'autres privilèges de sécurité à l'intérieur de l'autorisateur. Par exemple, il pourrait s'agir d'un niveau de classification militaire, d'un rôle tel que lecteur, auteur, etc. Le modèle ne définit pas la structure ni le type de ces privilèges.

5.3.5 Résumé des propriétés des capacités

Le modèle propose une génération centralisée des capacités du domaine par l'AS. L'échange des capacités entre serveurs et clients s'effectue au moyen d'une opération de délégation où on concatène un *autorisateur* à une capacité. L'autorisateur est une capacité générée par le SA qui identifie le client et le contexte de sa session de consultation.

Afin de diminuer le nombre de principaux devant être contactés pour valider une capacité, nous avons choisi d'utiliser des capacités auto identifiables [LC94]. Il s'agit des capacités fournissant tous les paramètres nécessaires pour leur vérification. Notamment, nous utilisons des clés publiques pour l'identification des propriétaires, des expéditeurs et des délégateurs de capacités.

Le modèle utilise trois types de capacités :

– Des *capacités de base*. Elles sont générées par l'ACLE. Une capacité de base, ou tout simplement capacité, met en relation un utilisateur avec une entrée de l'ACLE d'un noeud. Dans l'approche **CN**, la capacité identifie un graphe et une instance donné d'un noeud. Dans l'approche **CG**, la capacité n'identifie qu'un graphe. Le SA et les serveurs d'objets sont les propriétaires des capacités. Ceci est indiqué par l'inclusion des clés publiques de ces serveurs dans les capacités.

- Des *capacités déléguées.* Elles sont générées par le SA et par les serveurs du système. Un serveur utilise une capacité deléguée pour octroyer temporairement une capacité à un client. Une capacité déléguée est composée d'une capacité, d'un autorisateur et de la signature du propriétaire de la capacité.

- Des *autorisateurs.* Ils sont générée par le SA. Un autorisateur inclut l'identité d'un client, l'identificateur d'un graphe et un intervalle de temps alloué à la session de consultation. Il est généré au début d'une session, lorsque le client demande au SA sa première capacité. L'inclusion de l'autorisateur dans toute capacité déléguée donne à ces capacités l'identité du propriétaire de la capacité déléguée et un temps d'expiration. L'inclusion de l'identificateur du graphe permet notamment de détecter la délégation d'une capacité n'appartenant pas au graphe.

5.4 Deux approches de protection

Dans la section précédente, nous avons présenté la structure des capacités. Cette section explore l'association, l'échange et l'utilisation des capacités dans le système hypermédia. Plus spécifiquement, nous proposons deux approches fondées sur les propriétés des graphes hypermédia :

- l'approche *une capacité par noeud* (CN), basée sur l'appartenance d'un noeud à un graphe et sur les liens au départ de ce noeud ;

- l'approche *une capacité par graphe* (CG), basée uniquement sur l'appartenance d'un noeud à un graphe.

Nous allons maintenant décrire plus en détail chacune de ces approches. Nous concluons cette section par une description sur les considérations qu'il faut prendre en compte pour combiner les approches.

Remarque. Dans les sections suivantes, nous donnons une description fonctionnelle de plusieurs protocoles de sécurité. Afin de clarifier notre discour, nous séparons les étapes d'authentification du client et de présentation des capacités. Lors de la mise en oeuvre de ces protocoles, on peut éventuellement combiner ces deux pas en une seule transaction.

5.4.1 Structures de données employées

Avant de décrire les protocoles de consultation et l'initialisation de l'ACI du domaine, nous allons présenter les structures de données utilisées pour stocker l'ACI et les capacités chez les clients et chez les serveurs. Plus spécifiquement, nous décrivons le format des structures suivantes :

- les listes de contrôle d'accès étendues (ACLE),

– les listes de capacités (cap-liste),

– les listes de capacités déléguées (Dcap-liste).

Les deux premières structures correspondent à des données stockées dans les serveurs. La dernière structure est utilisée par un client pour stocker les capacités qu'il reçoit.

5.4.1.1 Listes de contrôle d'accès étendues (ACLE)

L'ACLE est une structure qui met en relation les noms de graphes qui incluent un noeud avec les droits d'accès qui leurs sont accordés. Les entrées de l'ACLE sont des triplets

UID_{graphe}	Epoque_{graphe}	droits d'accès.

L'identificateur du graphe donne le nom du graphe associé à cette entrée de l'ACL. L'époque du graphe permet d'identifier une instance du graphe. Enfin nous trouvons l'ensemble de droits d'accès associés au graphe.

L'attribut $Epoque_{graphe}$ est utilisé lors de la révocation des capacités d'un graphe qui pointent vers le noeud. Afin de pouvoir accéder à un noeud, l'autorisateur doit avoir une époque égale ou supérieure à celle associée au noeud. Le changement de l'époque du noeud entraîne la révocation des capacités ayant une époque dépassée.

5.4.1.2 Listes de capacités (cap-liste)

On les utilise pour associer un ensemble de capacités à un noeud. Les entrées de la liste sont des triplets

UID_{graphe}	$\text{Epoque}_{cap-liste}$	{cap}.

Chaque entrée de la liste identifie un graphe et spécifie les capacités qui doivent être déléguées pour ce graphe lors de la consultation du noeud. Rappelons que l'autorisateur indique au serveur le nom du graphe qu'un utilisateur est en train de consulter.

Le nombre d'entrées de la cap-liste indique le nombre de graphes qui incluent le noeud. L'époque de la liste de capacités identifie une instance donnée de cette liste. Comme nous verrons dans la section consacrée à la consultation des serveurs de noeuds, l'époque est utilisée pour optimiser la délégation des capacités.

5.4.1.3 Listes de capacités déléguées (Dcap-liste)

Le client stocke les capacités qui lui sont attribuées dans une *liste de capacités déléguées.* Les entrées de cette liste sont des paires

$\text{Dcap}_{noeud\ n}$	$\text{Epoque}_{cap-liste\ noeud\ n}$

où $Dcap_{noeud\ n}$ correspond à une capacité déléguée qui donne accès au noeud N. Si une capacité est utilisée pour accéder à un noeud et si au cours de cette opération le client reçoit une liste de capacités, l'attribut $Epoque_{cap-liste\ noeud\ n}$ indique l'époque de cette liste. Comme nous verrons dans la consultation des serveurs de noeuds, cet attribut joue un rôle dans l'optimisation du nombre des capacités à déléguer dans une session de consultation.

5.4.2 L'approche une capacité par noeud (CN)

Dans cette approche, chaque noeud est associé à une ACLE. De plus, d'après la structure des graphes, on associe aux noeuds des listes de capacités. Celles-ci donnent accès à d'autres noeuds du graphe. Au début de la session de consultation, le client recupère une capacité pour le *noeud racine* d'un graphe auprès du SA. Lors du parcours d'un noeud associé à une liste de capacités, le client recupére le noeud et la liste qui correspond au graphe.

Dans la suite ce cette section, nous décrivons d'abord l'*initialisation de l'ACI du domaine*. Nous décrivons ensuite le protocole de consultation du serveur d'autorisation. En outre, nous décrivons l'utilisation des capacités dans chacun des protocoles de consultation hypermédia que nous avons identifiés dans la section 3.5, à savoir :

- la consultation d'un serveur de noeuds,
- le protocole de migration par redirection,
- la consultation d'un serveur de nom,
- la consultation d'un serveur de liens.

Nous concluons cette section en proposant une extension permettant de diminuer le nombre de délégations associées à une session de consultation.

5.4.2.1 Initialisation du domaine

Cette étape concerne l'association de l'ACI aux noeuds du domaine et au serveur d'autorisation.

Initialisation des serveurs de noeuds

Dans cette étape, l'AS associe aux noeuds du domaine trois types d'éléments :

- l'*époque* du noeud,
- une *liste de contrôle d'accès etendue* (ACLE),
- des *liste de capacités* (cap-liste),

L'*époque du noeud* est un compteur qui identifie une instance donnée d'un noeud. Toutes les capacités valides donnant accès à ce noeud ont la même valeur d'époque. L'intêret de cette élément est de permettre la révocation de toutes les capacités pointant vers le noeud par un simple incrément de l'époque du noeud. L'ACLE spécifie les droits d'accès des graphes sur le noeud. Enfin, la liste de capacités specifie pour chaque graphe les capacités qu'il faudra déléguer lors de la consultation du noeud.

Mise à part l'attribution d'une époque à chaque noeud du domaine, nous pouvons diviser l'initialisation de l'ACI en trois étapes :

– l'initialialisation des ACLE des noeuds d'après la structure des graphes,

– l'association de listes de capacités aux noeuds d'après une *politique d'assignation de capacités*,

– l'initialisation du SA.

L'AS initialise les ACLE de chaque noeud du domaine en ajoutant une entrée pour chaque graphe qui inclut le noeud. Dans cette entrée, il spécifie l'époque courante du graphe et les droits d'accès valables pour ce graphe lors de la consultation du noeud. Suite à l'initialisation, l'ACLE d'un noeud aura autant d'entrées que de graphes où le noeud est inclus. En outre, tous les noeuds appartenant à un graphe devront tous avoir une entrée pour le graphe et la même époque du graphe. Dans la suite, si on veut révoquer toutes les capacités déléguées qui pointent vers le noeud, il suffit d'incrémenter la valeur de l'époque des entrées des graphes concernées.

La deuxième étape d'initialisation concerne la génération des capacités et leur association aux noeuds. Pour chaque noeud d'un graphe, l'AS évalue s'il faut ou non lui associer une liste de capacités d'après une *politique d'assignation des capacités*. Si c'est le cas, l'AS génère ces capacités, utilisant la clé publique du serveur qui héberge le noeud comme valeur de l'attribut KU_{deleg}. Enfin, l'AS stocke les capacités dans une entrée de la liste de capacités du noeud et lui assigne l'identificateur du graphe et une époque. Dans la suite, à chaque évolution du contenu de cette liste, l'AS augmentara la valeur de son époque.

La *politique d'assignation des capacités* n'est pas imposée par le modèle car elle dépend du contexte d'utilisation et de la structure des graphes. Il peut exister plusieurs types de politiques. Quelques exemples comprennent :

– l'*association hiérarchique* : on associe à un noeud père une liste de capacités pour ses noeuds fils ;

– la *visite guidée* : les capacités sont associées suivant un chemin prédeterminé.

– l'*association par sous-graphe* : on associe à un noeud des capacités donnant accès à tous les noeuds d'un sous-graphe.

Nous allons maintenant illustrer quelques unes de ces politiques au moyen d'un exemple.

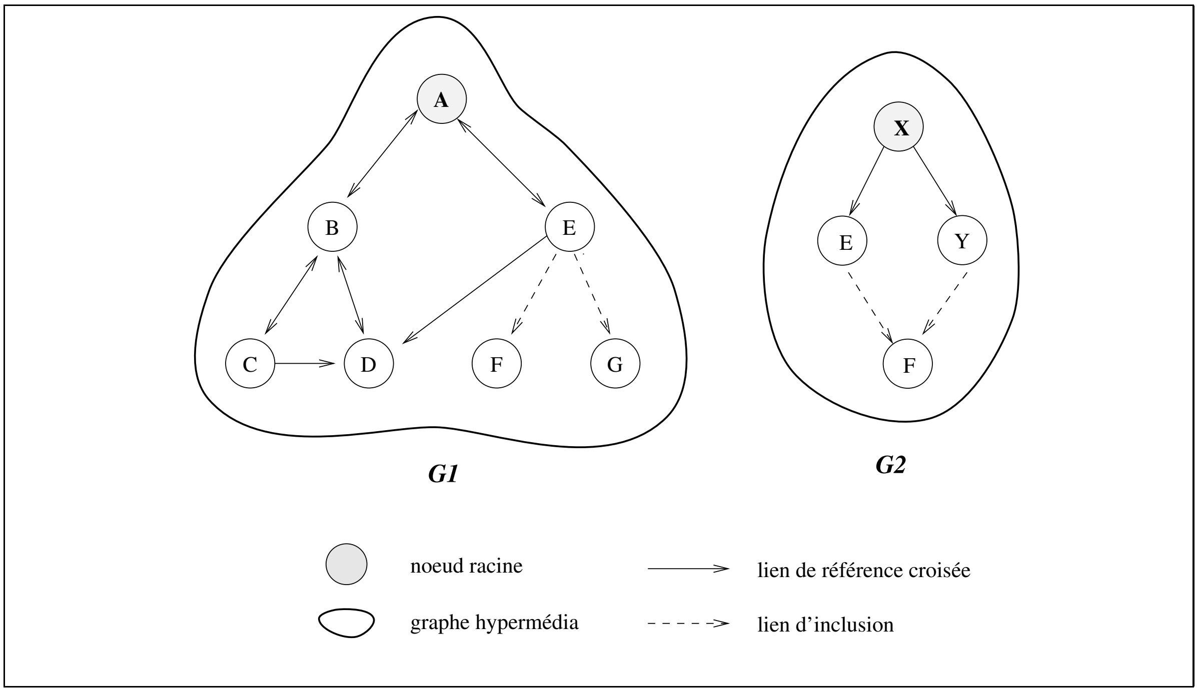


Fig 5.2: Deux exemples de graphes hypermédia

Exemple

La figure 5.2 montre deux graphes hypermédia, $G1$ et $G2$. Ces graphes ont deux noeuds, E et F en commun. Les noeuds racine A et X représentent le point d'entrée aux deux graphes.

Le tableau 5.4 montre un extrait des ACLE et des listes de capacités version abrégée des ACLE (à gauche) et des listes de capacités (à droite) pour cet exemple.

Nous pouvons observer que tous les noeuds du graphe $G1$ incluent une entrée pour ce graphe. Les noeuds E et F ont deux entrées dans leur ACLE, une pour chaque graphe auquel ils appartiennent.

Dans cet exemple, nous illustrons deux politiques d'assignation de capacités : la "visite guidée" et l'association par sous-graphes. L'assignation des capacités donnant accès aux noeuds A, B, C et D suit une politique de "visite guidée". Le client doit initialement parcourir les noeuds suivant un ordre donné afin de récupérer toutes les capacités. Par la suite, le client peut parcourir ces noeuds dans n'importe quel ordre.

L'assignation des capacités pour accéder aux noeuds E, F et G suit une politique d'assignation par sous-graphes. Lorsque l'utilisateur récupère le noeud A, il récupère en même temps les capacités lui permettant d'accéder au sous-graphe composé des noeuds E, F et G. Par la suite, le client peut consulter dans n'importe quel ordre ces noeuds.

noeud	**époque noeud**	**graphe**	**époque graphe**	**droits d'accès**	**cap-liste**	**époque cap-liste**
A	1	G1	1	read	$\{cap_B, cap_E, cap_F, cap_G\}$	1
B	1	G1	1	read	$\{cap_C\}$	1
C	1	G1	1	read	$\{cap_D\}$	1
D	1	G1	1	read	-	-
E	1	G1	1	read	-	-
	1	G2	1	read	-	-
F	1	G1	1	read	-	-
	1	G2	1	read	-	-
G	1	G1	1	read	-	-

Tab 5.4: Approche CN : assignation des capacités pour les graphes de la figure 5.2

1.	Client → Serveur d'autorisation	:	$URI_{noeud\ racine}$, [ACI]
2.	Serveur d'autorisation → Client	:	$Dcap_{noeud\ racine}$ *ou* code d'erreur

Fig 5.3: Approche CN : protocole de consultation du serveur d'autorisation

5.4.2.2 Protocole de consultation d'un serveur d'autorisation

Un client utilise ce protocole pour récupérer une capacité pour un noeud racine. La figure 5.3 illustre les pas du protocole.

D'abord, le client envoie au serveur d'autorisation l'identificateur du noeud qu'il veut accéder. D'après la politique de sécurité du système, le client peut être obligé ou non de compléter sa requête avec de l'ACI. Par exemple, le serveur peut demander que le client présente une autre capacité ou qu'il s'authentifie au moyen d'un mot de passe afin d'autoriser la requête. Le choix de cette ACI est dépendant du contexte d'utilisation et n'est pas défini par le modèle.

Si la capacité n'est pas stockée dans le serveur ou si l'autorisation de la requête échoue, le serveur retourne un code d'erreur au client. Si le serveur héberge la capacité, il crée un autorisateur contenant le nom du graphe auquel appartient le noeud racine, l'identité du client et le temps alloué à la session de consultation. Ensuite, le serveur délégue la capacité demandée et l'envoie au client.

L'identité du principal qui est inscrite dans l'autorisateur peut être de deux types. Elle peut être soit l'*adresse réseau* de la station de travail du client, soit une *clé publique* associée à l'utilisateur, soit une combinaison de ces deux types. Cette dernière peut être soit une clé publique fournie par l'utilisateur lors d'un processus d'authentification, soit une clé publique temporaire, générée par le serveur d'autorisation et valable pendant la session de consultation. Dans ce dernier cas, le serveur doit envoyer au client par un canal de communication sécurisé la clé privée associée à la clé publique.

1.	Client → Serveur de noeuds	:	authentification du client
2.	Client → Serveur de noeuds	:	requête$_{noeud\ n}$, Dcap$_{noeud\ n}$, [époque cap-liste$_{noeud\ n}$]
3.	Serveur → Client	:	noeud n, [Dcap-liste$_{noeud\ n}$, époque cap-liste$_{noeud\ n}$] *ou* code d'erreur

Fig 5.4: Approche CN : protocole de consultation d'un serveur de noeuds

5.4.2.3 Protocole de consultation d'un serveur de noeuds

L'objectif de ce protocole est de permettre à un client de récupérer un noeud et, si elle existe, la liste de capacités associée au noeud. La figure 5.4 illustre les étapes du protocole.

Afin de pouvoir accéder à un noeud protégé, le client doit inclure dans sa requête une capacité déléguée qui donne accès au noeud. Le client s'authentifie d'abord auprès du serveur de noeuds sous l'identité inscrite dans le champ UID_{client} de l'autorisateur de la capacité déléguée. Le client envoie ensuite l'URL du noeud et la capacité déléguée. Si le client a accédé précedemment au noeud, et si le noeud est associé à une liste de capacités, le client inclut dans sa requête l'attribut *Epoque* de cette liste.

Lors de la réception de la requête, le serveur valide la capacité déléguée et extrait le nom du graphe inscrit dans celle-ci. Si la capacité déléguée n'est pas valide ou si les droits d'accès associés au graphe ne correspondent pas à la requête, le serveur retourne une erreur. Cette erreur peut avoir la forme d'un défi d'autorisation, donnant des indications sur la capacité qu'il faut présenter (cf. section 5.4.4).

Le serveur vérifie ensuite s'il existe une liste de capacités associée au noeud et s'il existe une entrée dans cette liste pour le graphe que le client peut consulter. Si c'est le cas, le serveur compare alors l'époque de la liste avec celle envoyée par le client. Si l'époque de la liste est supérieure à celle envoyée par le client, ou si celui-ci n'a pas envoyé ce paramètre, le serveur doit déléguer la liste de capacités. Pour ce faire, il extrait d'abord l'autorisateur de la capacité déléguée présentée par l'utilisateur. Le serveur délègue ensuite chaque capacité en la concatenant à l'autorisateur et en signant le tout avec sa clé privée (fig. 5.5).

Enfin, le serveur retourne au client le noeud et, si elle existe, la liste de capacités déléguées ainsi que l'époque de la liste. Enfin, si le client reçoit une liste de capacités déléguées, il les ajoute dans sa liste de capacités déléguées et met à jour la valeur de l'époque de la capacité qu'il avait inclut dans sa requête.

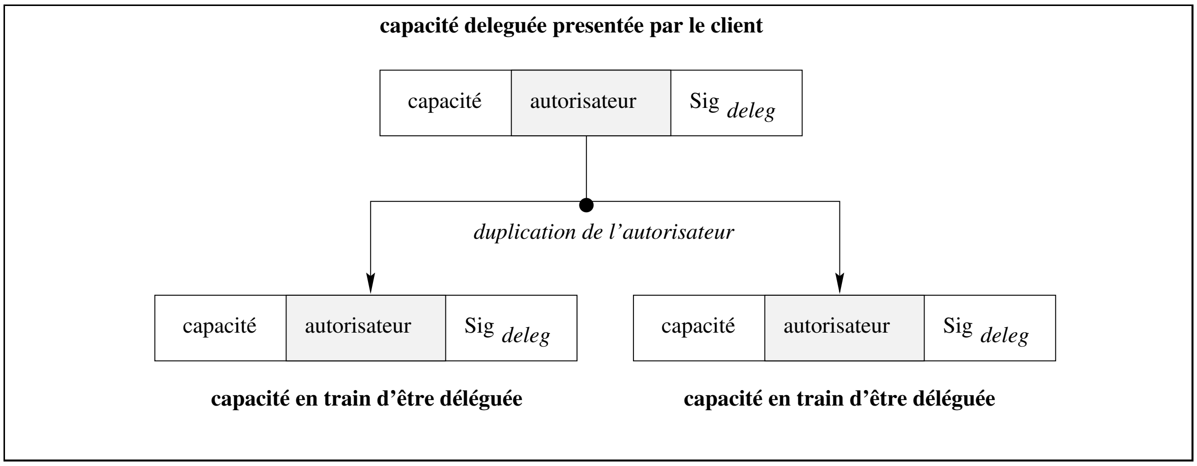


Fig 5.5: Délégation des capacités : duplication de l'autorisateur

Nous allons maintenant détailler les opérations de validation d'une capacité déléguée.

Validation d'une capacité déléguée

La validation d'une capacité déléguée consiste en deux opérations :

- la vérification de l'authenticité de la capacité,
- la comparaison des attributs de la capacité avec ceux de l'ACI associée au noeud.

L'authentification de la capacité déléguée concerne la vérification de l'origine et de l'intégrité des données de la capacité déléguée, de la capacité de base et de l'autorisateur. Un serveur de noeuds effectue ces deux opération en même temps en vérifiant la signature de ces différentes capacités. En effet, la signature constitue en même temps une preuve d'intégrité et d'authentification du signataire. Le serveur vérifie la signature de la capacité de base et de l'autorisateur en utilisant les clés publiques courantes de l'AS et du SA[6] Le serveur vérifie ensuite la signature de l'autorisateur en utilisant la clé publique courant du délégateur et qui est inscrite dans l'attribut KU_{deleg} de l'autorisateur.

Une fois que le serveur a vérifié l'authenticité de la capacité, il s'agit de vérifier si l'utilisateur peut bien utiliser la capacité pour accéder au noeud. Pour ce faire, le serveur compare les champs de la capacité et de l'autorisateur avec ceux de l'ACI associée au noeud et d'autres informations contextuelles. La figure 5.6 résume les différentes comparaisons.

Les comparaisons permettent de valider les propriétés suivantes :

- *Validité de la capacité.* La date courante du système doit être comprise dans l'intervalle de la période de validité de la capacité. Comme nous verrons dans

6. Nous rappelons qu'une des hypothèses de base est que les clés publiques de l'AS et du SA sont connues de tous les serveurs du domaine.

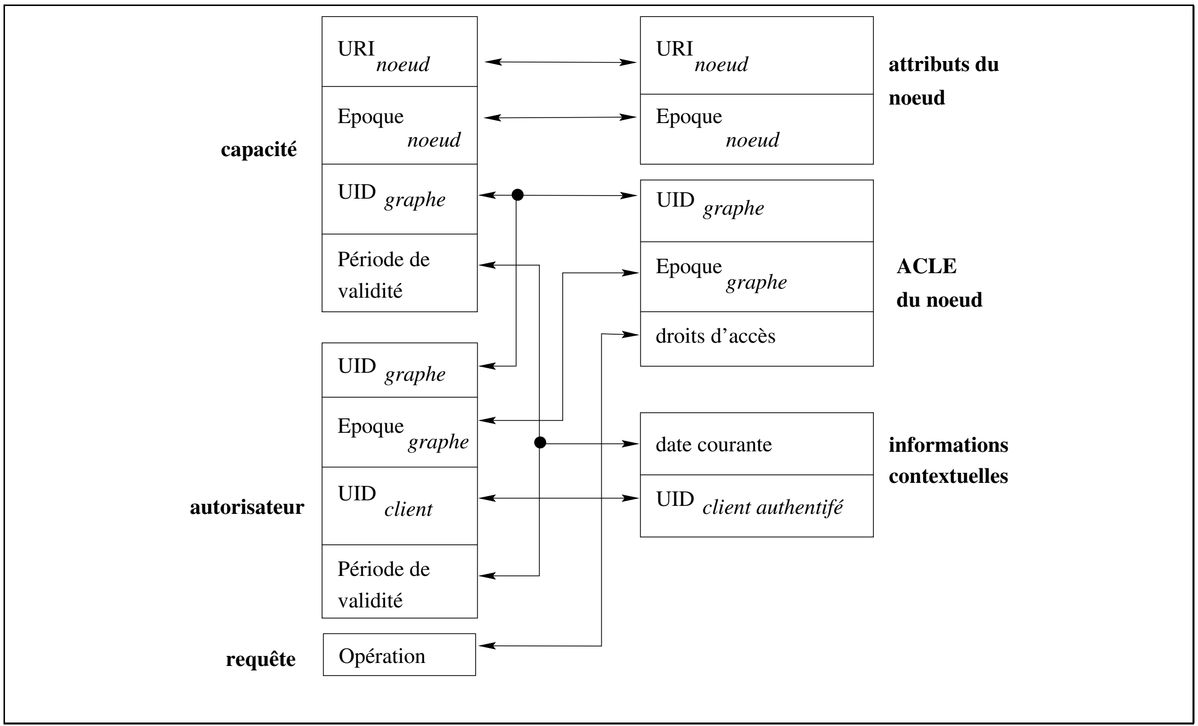


Fig 5.6: Comparaisons faites lors de la validation des capacités

la section 5.5, la révocation explicite d'une capacité implique l'incrément des attributs époque associés au noeud. La comparaison de ces attributs permet donc de savoir si la capacité a été révoquée. Une capacité déléguée valide doit avoir des attributs époque ayant une valeur égale ou supérieure à celle des époques associés au noeud.

- *Validité de la délégation.* Les attributs UID_{graphe} de l'autorisateur et de la capacité doivent être les mêmes. En effet, le client ne peut utiliser que des capacités ayant le même identificateur de graphe que celui inscrit dans l'autorisateur.

- *Validation du propriétaire de la capacité.* L'identité authentifiée du client doit être la même que celle inscrite dans l'autorisateur.

- *Droits d'accès.* La capacité doit identifier le noeud et un des graphes inscrites dans l'ACLE du noeud. De plus, l'opération demandée par l'utilisateur doit correspondre à un des droits d'accès attribués au graphe.

Dans les sections suivantes, nous allons décrire des extensions au protocole de consultation des noeuds pour prendre en compte la migration de noeuds par redirection, les serveurs de noms et les serveurs de liens.

1.	Client → Serveur de noeuds 1	:	authentification du client
2.	Client → Serveur de noeuds 1	:	requête$_{noeud\ n}$, Dcap$_{noeud\ n,\ serveur\ 1}$, [époque cap-liste$_{noeud\ n,\ serveur\ 1}$]
3.	Serveur de noeuds 1 → Client	:	URL$_{noeud\ n,\ serveur\ 2}$, Dcap$_{noeud\ n,\ serveur\ 2}$
4.	Client → Serveur de noeuds 2	:	authentification du client
5.	Client → Serveur de noeuds 2	:	requête$_{noeud\ n}$, Dcap$_{noeud\ n,\ serveur\ 2}$, [époque cap-liste$_{noeud\ n,\ serveur\ 2}$]
6.	Serveur de noeuds 2 → Client	:	noeud n, [Dcap-liste$_{noeud\ n}$, époque cap-liste$_{noeud\ n}$]

Fig 5.7: Approche CN : protocole pour la redirection de noeuds

5.4.2.4 Protocole de migration des noeuds par redirection

Le principe de base de ce protocole est que tout noeud qui se déplace dans un système hypermédia, laisse derrière lui une URL qui pointe vers son nouveau emplacement. Lorsqu'un client veut récupérer le noeud dans son ancien emplacement, le serveur lui renvoit l'URL. Le client accède ensuite le noeud dans son nouveau emplacement [BLFF96].

Nous prenons en compte ce protocole en stockant dans l'ancien emplacement du noeud une capacité donnant accès au noeud dans son nouvel emplacement. La figure 5.7 illustre le protocole lorsqu'un noeud se déplace du serveur 1 vers le serveur 2. Lorsque le client demande le noeud au serveur 1, il récupère au retour les nouvelles coordonnées du noeud ainsi que la nouvelle capacité. Le client utilise ensuite cette capacité pour accéder le noeud dans le serveur 2.

Ce protocole a l'inconvénient que chaque fois qu'un noeud migre, il faut générer une nouvelle capacité. De plus, le temps de consultation augmente d'une opération de délégation supplémentaire.

5.4.2.5 Protocole de consultation d'un serveur de noms

Comme nous l'avons mentionné dans la section 3.3, l'URI d'un noeud peut être un nom absolu (une URL) ou symbolique (une URN). Dans ce dernier cas, le client doit consulter un serveur de noms afin de résoudre l'URN en une URL.

Dans l'approche **CN**, un premier problème qui se pose est de pouvoir désigner le noeud avec lequel on peut utiliser une capacité. En effet, ce n'est qu'au moment de la résolution de l'URN que nous pourrons connaître l'URL du noeud destination. La solution à ce problème est de désigner le noeud cible au moyen de son URN[7]. Une

7. Dans la suite, nous appellons *capacité URN* et *capacité URL* une capacité désignant un noeud

capacité identifie alors de manière symbolique l'objet.

Un deuxième problème qui se pose est l'utilisation d'une capacité URN pour pouvoir accéder au noeud cible lui même. Nous envisageons deux solutions à ce problème : la *conversion* d'une capacité URN en une capacité URL et la *banalisation* des capacités URN.

1.	Client → Serveur de noms	:	authentification du client
2.	Client → Serveur de noms	:	requête$_{URN\ noeud\ n}$, Dcap$_{URN\ noeud\ n}$
3.	Serveur de noms → Client	:	URL noeud n, Dcap$_{URL\ noeud\ n}$ *ou* code d'erreur

(a) Conversion d'une capacité URN en une capacité URL

1.	Client → Serveur de noms	:	requête$_{URNnoeud\ n}$
2.	Client → Serveur de noeuds	:	authentification du client
3.	Client → Serveur de noeuds	:	requête$_{noeud\ n}$, Dcap$_{URN\ noeud\ n}$
3.	Serveur de noeuds → Client	:	noeud n, [Dcap-liste$_{noeud\ n}$ [a], époque cap-liste$_{noeud\ n}$] *ou* code d'erreur

[a] Cette liste ne contient que des capacités URN

(b) Banalisation des capacités URN pour accéder aux serveurs de noeuds

Fig 5.8: Approche CN : deux approches pour la consultation d'un serveur de noms

Conversion des URN

L'idée de base est d'utiliser le serveur de noms pour fournir une capacité URL par indirection. Lors de l'initialisation du domaine, on stocke des capacités URN dans les listes de capacités des noeuds. De plus, on associe des capacités URL aux identificateurs URL qui sont stockés dans le serveur de noms. Par exemple, si les entrées du serveur de noms mettent en correspondance une URN avec une URL, il s'agit d'avoir des triplets

$$(\mathrm{URN}_{noeud\ n},\ \mathrm{URL}_{noeud\ n},\ \mathrm{cap}_{noeud\ n}).$$

par une URN ou par une URL, respectivement.

Lorsque le client veut convertir une URN en une URL, il envoie l'URN du noeud plus la capacité URN au serveur de noms (fig. 5.8.a). Le client récupère au retour l'URL du noeud plus la capacité qui donne accès au noeud. Aussi, la capacité URN donne accès à une capacité URL. Cette dernière donne enfin accès au noeud.

Notons que le serveur de noms pourrait éventuellement n'envoyer que la capacité URL au client. En effet, celle-ci inclut déjà l'URL et le client peut la récupérer facilement.

Un avantage de cette solution est que les capacités indiquent toujours les objets cibles avec lesquelles elles peuvent être employées. Le client peut donc toujours savoir quelle capacité il faut joindre à ses requêtes.

Le prix à payer pour cette solution est un besoin d'une protection plus accrue du serveur de noms. En effet, ce serveur peut maintenant déléguer des capacités donnant accès à d'autres noeuds. Il lui faut donc une meilleure protection contre les intrusions. Un autre inconvénient de cette solution concerne la performance. Chaque requête pour un noeud nécessite une délégation et une vérification supplémentaires. En effet, un serveur de noeuds doit d'abord déléguer la capacité URN ; puis, le serveur de noms doit vérifier la capacité URN et déléguer la capacité URL, enfin, un autre serveur de noeuds doit vérifier la capacité URL. Enfin, la solution entraîne la multiplication par deux du nombre de capacités du domaine.

Cette solution requiert la modification du serveur de noms et de son protocole d'accès. On suppose donc qu'il est possible d'avoir accès au code source du serveur. La solution que nous allons présenter ci-après présente une alternative qui ne nécessite aucune intervention dans le serveur de noms.

Banalisation des URN

Cette solution consiste en la banalisation des capacités URN pour qu'on puisse les utiliser pour accéder directement aux noeuds. Pour ce faire, il suffit de stocker la valeur de l'URN d'un noeud dans l'information de contrôle d'accès qui est associée au noeud. Lorsqu'un serveur de noeuds reçoit une capacité URN, il doit d'abord vérifier que l'URN inscrite dans la capacité et celle qui est associée au noeud sont les mêmes. Si c'est le cas et si la capacité est valide, le serveur retourne le noeud demandé. La figure 5.8.b résume les échanges de messages que nous venons de mentionner.

La banalisation des URN a l'avantage qu'elle ne nécessite pas la modification du serveur de noms. De plus, elle a l'avantage que le nombre d'opérations de délégation et de vérification restent le même. Enfin, le nombre de capacités dans le domaine est le même que celui que l'on a lorsque l'on n'utilise pas de serveur de noms.

Cette solution a l'inconvénient qu'il devient plus complexe pour un client de choisir une capacité URN. En effet, le client doit mémoriser la relation entre l'URN et l'URL d'un noeud afin de pouvoir choisir la capacité URN qu'il faut joindre à sa requête pour le noeud. Un complément à cette solution pourrait être d'inclure l'URN du noeud dans le défi envoyé par le serveur.

En résumé, nous avons présenté deux solutions pour la prise en compte d'un serveur de noms dans notre modèle. Le choix d'une ou l'autre solution dépend du contexte d'instanciation du modèle et de la facilité de sélection de la capacité qui doit accompagner une requête.

1.	Client → Serveur de liens	:	authentification du client
2.	Client → Serveur de liens	:	requête$_{liens\ noeud\ n}$ Dcap$_{noeud\ n}$
3.	Serveur de liens → Client	:	liens noeud n Dcap-liste$_{noeud\ n}$

Fig 5.9: Approche CN : protocole de consultation d'un serveur de liens

5.4.2.6 Consultation d'un serveur de liens

Avant de décrire la prise en compte de ce service, nous allons rappeler les principaux concepts des serveurs de liens. Dans les systèmes où les liens et les noeuds sont des citoyens de premier classe, on stocke chacune de ces entités dans des serveurs de noeuds et dans des serveurs de liens, respectivement. On peut assimiler un serveur de liens à une base de données où chaque entrée de la base correspond à un lien [HKR+92, PJ96, Mau96]. Comme nous l'avons décrit dans la section 3.2.2, chaque lien inclut l'URI d'un noeud source et l'URI d'un noeud destination. Lorsque le client veut récupérer la liste de liens au départ d'un noeud, il envoie au serveur l'URI du noeud source. Le serveur envoie au retour la liste de tous les liens comprenant un tel noeud.

Dans l'approche **CN**, le problème qui se pose est de synchroniser les listes de capacités associées aux noeuds avec les liens existant entre les noeuds. Si on ajoute ou on enlève un lien, ou si l'URI du noeud destination change, il faut mettre à jour la liste de capacités associée au noeud.

La solution que nous avons retenue est de déplacer les listes de capacités associées aux noeuds ainsi que les responsabilités de délégation vers le serveur de liens. Plus formellement, il s'agit d'associer à chaque lien dans le serveur de noeuds l'information de contrôle d'accès qui est associé au noeud source : l'époque du noeud et l'ACLE. Enfin, d'après la politique d'assignation de capacités, on associe à chaque lien une liste de capacités. Cette liste indique pour chaque graphe quelles sont les capacités à déléguer lors de l'accès au lien.

La figure 5.9 illustre le protocole d'autorisation pour la consultation du serveur de liens. Lorsque le client demande la liste de liens, il joint à sa requête la même capacité déléguée qu'il utilise pour accéder au noeud source. Le serveur valide la capacité et, pour chaque lien ayant le noeud source, délègue toutes les capacités appartenant au graphe inscrit dans l'autorisateur. Le serveur retourne ensuite la liste de liens et la liste de capacités déléguées.

5.4.2.7 Extension : minimisation des capacités déléguées

Dans cette section nous proposons deux extensions à l'approche **CN** permettant de diminuer le nombre de capacités devant être déléguées.

Pour l'approche **CN** comme nous l'avons décrite jusqu'à maintenant, il faut présenter une capacité différente pour accéder à chaque noeud. Un inconvénient de cette approche est que si un graphe possède n noeuds, il faudra réaliser n opérations de délégation de capacités. Comme cette opération requiert l'application d'une signature numérique, elle peut avoir un effet négatif sur la performance lorsque une liste de capacités dépasse une certaine taille.

Une première solution à ce problème est l'utilisation des expressions régulières pour identifier un noeud. Si les noeuds sont stockés dans le même serveur et si les URL de ces noeuds ont la même structure hiérarchique, on peut spécifier le nom des noeuds au moyen d'une expression régulière. Par exemple, on peut utiliser l'identificateur suivant

`http://www.w3.org/Amaya/*`

pour spécifier qu'une capacité est valable pour tous les noeuds commençant par ce fragment d'URL. Cette solution est semblable à celle utilisé dans les fichiers de configuration des serveurs WWW [Ste95]. Un inconvénient de cette solution est que l'on ne peut pas différencier les attributs *Epoque* des différents noeud. On est donc obligé d'utiliser un seul de ces attributs dans la capacité et d'associer la même époque à tous les noeuds concernés.

Une autre solution au problème est d'inclure dans une capacité un ensemble de paires

$$(\mathrm{URI}_{noeud}, \mathrm{Epoque}_{noeud})$$

identifiant les objets avec lesquels on peut utiliser la capacité. Une avantage de cette solution est que l'on n'est plus contraint à avoir une structure hiérarchique des URI des noeuds. De plus, les époques des noeuds deviennent alors indépendantes. Un inconvénient de cette approche est qu'elle agrandit la taille de la capacité. Si la taille de la capacité dépasse un certain seuil, le temps nécessaire pour sa délégation et sa validation pourrait devenir trop important.

Le choix pour l'une ou l'autre solution dépendra donc du besoin de distinguer l'époque de chaque noeud et de la taille éventuelle de la capacité.

5.4.3 L'approche une capacité par graphe (CG)

Cette approche est basée sur la propriété d'appartenance d'un noeud à un graphe. Un utilisateur récupère une capacité auprès du serveur d'autorisation. Il l'utilise ensuite pour accéder à n'importe quel noeud appartenant au graphe.

Dans la suite, nous décrivons l'initialisation de l'information de contrôle d'accès dans le domaine. Nous décrivons ensuite les protocoles d'autorisation associés à une session de consultation. Ces deux phases sont très proches de celles de l'approche **CN**.

Aussi, nous ne décrivons en détail que les différences fondamentales entre les deux approches.

5.4.3.1 Initialisation du domaine

Cette phase concerne l'initialisation de l'ACI stockée dans le SA et dans les serveurs de noeuds.

L'AS stocke dans le SA des capacités donnant accès aux graphes du domaine. Chacune de ces capacités spécifie le nom d'un graphe et identifie le SA comme délégateur.

Comme dans l'approche **CN**, L'AS associe une ACLE à chaque noeud du système. Les entrées de cette liste identifient les graphes auxquels appartient le noeud. A la différence de l'approche **CG**, il n'est pas nécessaire d'associer une époque aux noeuds car les capacités n'en font référence.

Exemple

Le tableau suivant montre un extrait des ACLE utilisées lors de la protection des deux graphes de la figure 5.2 page 105. Notons qu'elles sont identiques à celles utilisées dans l'approche **CN**, sauf pour l'omission du champ $Epoque_{noeud}$.

noeud	**graphe**	**époque graphe**	**droits d'accès**
A	G1	1	read
B	G1	1	read
C	G1	1	read
D	G1	1	read
E	G1	1	read
	G2	1	write
F	G1	1	read
	G2	1	read
G	G1	1	read
X	G2	1	read
Y	G2	1	read

5.4.3.2 Protocoles d'autorisation

L'approche **CG** comporte deux protocoles d'autorisation :

– le protocole de consultation du SA,

– le protocole de consultation d'un serveur de noeuds.

Contrairement à l'approche **CN**, cette approche ne requiert pas de protocole d'autorisation pour prendre en compte la migration par redirection, la consultation d'un serveur de noms et la consultation d'un serveur de liens. En effet, comme les capacités ne désignent plus les noeuds, on n'est plus obligé de fournir une nouvelle capacité lors du changement de l'identificateur d'un noeud.

Similairement, la migration par redirection des noeuds ne requiert que le déplacement de l'ACLE du noeud. Il n'est pas nécessaire de protéger l'accès à l'URL pointant vers le nouveau emplacement du noeud source.

Protocole de consultation du serveur d'autorisation

Ce protocole, illustré dans la figure 5.10, est très proche de celui que nous avons décrit précédemment pour l'approche **CN.** La seul différence est que le client demande au SA une capacité pour un graphe, plutôt qu'une capacité pour un noeud.

Notons que même si l'autorisateur indique le nom du graphe et son époque, il est nécessaire que le serveur délègue une capacité pour le graphe. En effet, la capacité est utilisée lors de l'accès local à un serveur, tandis que l'autorisateur indique les paramètres globaux d'autorisation du client.

1.	Client → Serveur d'autorisation	:	UID$_{graphe\ g}$,
			[ACI]
2.	Serveur d'autorisation → Client	:	Dcap$_{graphe\ g}$
			ou
			code d'erreur

Fig 5.10: Approche CG : protocole de consultation du serveur d'autorisation

Protocole de consultation d'un serveur de noeuds

Ce protocole n'est utilisé que lors de la consultation d'un objet protégé. La figure 5.11 montre les différentes transactions.

Le client doit d'abord s'authentifier et joindre à sa requête une capacité déléguée pour un graphe. Si cette capacité est valide, le serveur retournera l'objet. Autrement, il répondra avec un code d'erreur.

1.	Client → Serveur de noeuds	:	authentification du client
2.	Client → Serveur de noeuds	:	requête$_{noeud\ n}$,
			Dcap$_{graphe\ g}$
3.	Serveur de noeuds → Client	:	noeud n
			ou
			code d'erreur

Fig 5.11: Approche CG : protocole générique de consultation

En résumé, dans l'approche **CG**, le seul serveur dans le domaine qui délègue des capacités est le SA. Chaque capacité identifie un graphe dans le domaine. On associe aux noeuds des ACLE semblables à celles de l'approche **CN**, mais avec un champ $\acute{e}poque_{noeud}$ vide. Cette approche comporte deux protocoles d'autorisation, un pour récupérer une capacité auprès du SA et l'autre pour consulter les serveurs de noeuds. La consultation des autres serveurs du domaine ou la migration des noeuds ne requièrent pas un protocole d'autorisation spécifique.

5.4.4 Envoi d'un défi d'autorisation

Nous décrivons dans cette section un protocole qui permet au client de connaître quelle capacité il lui faut pour accéder à un noeud. Jusqu'à présent, nous avons supposé que le client commence une session de consultation en demandant au serveur d'autorisation une capacité pour un graphe. Cependant, dans deux cas, un client est susceptible d'accéder à un noeud sans avoir de capacité :

- lorsque le client consulte un noeud sans qu'il sache qu'il est protégé,
- lorsque le client suit un lien qui pointe vers un noeud appartenant à un autre graphe protégé, sans qu'il ait une capacité lui permettant de consulter ce deuxième graphe. Notons qu'il ne s'agit pas d'un noeud partagé par les deux graphes.

Nous avons choisi d'utiliser un protocole de défi/réponse (cf. section 4.2.4) pour palier à ce problème. La motivation pour ce choix est que l'on n'a pas besoin de modifier les liens des documents pour communiquer le défi à l'utilisateur.

La figure 5.12 illustre le protocole. Si le client demande un noeud sans inclure la bonne capacité, le serveur extrait de l'ACLE associée au noeud la liste de graphes qui incluent le noeud. Ensuite, il envoie cette liste et le nom du serveur d'autorisation au client. Si le client possède la capacité, il la retourne au serveur. Autrement, il en acquiert une auprès du serveur d'autorisation.

1.	Client $\rightarrow$ Serveur de noeuds	:	URL_{noeud}
2.	Serveur de noeuds $\rightarrow$ Client	:	nom du serveur d'autorisation, liste de graphes incluant le noeud

Fig 5.12: Protocole de défi/réponse

5.4.5 Combinaison des deux approches

Les deux approches que nous venons de présenter présentent des propriétés complémentaires. L'approche **CN** permet à un serveur de noeuds de déléguer une capacité sans besoin de passer par le SA. L'approche **CG** permet d'utiliser une seule capacité pour accéder à tous les noeuds d'un graphe, ce qui permet de consulter les noeuds dans n'importe quel ordre.

Il est possible de combiner les deux approches pour profiter de leurs propriétés. En d'autres termes, il s'agit d'autoriser la présence des capacités pour un graphe dans les listes de capacités. En outre, il s'agit de généraliser la délégation des capacités par un serveur de noeuds aux deux approches. Dans la suite, nous appelons *capacités CN* et *capacités CG* les capacités utilisées dans l'approche **CN** et **CG**, respectivement.

La structure des capacités simplifie la combinaison des approches. En effet, on utilise le même format des capacités. De plus, la procédure de délégation des capacités est indépendante du type de la capacité. Il est possible donc d'inclure une capacité **CG** dans une liste de capacités et de permettre à un serveur d'objets de la déléguer.

Un problème qui se présente lors de la délégation d'une capacité **CG** par un serveur de noeuds est celui des identificateurs des graphes. Si la capacité garde le même nom du graphe où le noeud est inclu, on peut l'utiliser par la suite pour accéder à tous les noeuds du graphe. Par exemple, supposons que dans le graphe $G1$ de la figure 5.2 page 105, nous utilisons une capacité **CG** pour accéder aux noeuds F et G. Supposons également que cette capacité est associée au noeud E. Enfin, supposons que l'accès à tous les autres noeuds du graphe utilise des capacités **CN**. Le tableau suivant montre un extrait des ACLE et des listes de capacités pour cet exemple :

noeud	**époque noeud**	**graphe**	**époque graphe**	**cap-liste**	**époque cap-liste**
A	1	G1	1	$\{\text{cap}_B, \text{cap}_E\}$	1
B	1	G1	1	$\{\text{cap}_C, \text{cap}_D\}$	1
C	1	G1	1	-	-
D	1	G1	1	-	-
E	-	G1	1	cap_{G1}	1
F	-	G1	1	-	-
G	-	G1	1	-	-

La consultation des noeuds B, C et D suit les spécifications de l'approche **CN**. Si le client commence son parcours par le noeud E, il obtiendra une capacité donnant accès au graphe $G1$. Le client pourra alors accéder non seulement aux noeuds F et G mais il pourra consulter avec la même capacité tous les autres noeuds, sans respecter les contraintes que nous voulons imposer.

Une solution à ce problème est de toujours grouper dans un sous-graphe un noeud et les noeuds qu'il inclut. Nous pouvons ensuite utiliser une capacité **CG** pour accéder à ce sous-graphe. Appliquons maintenant cette solution à l'exemple précédent. Nous regroupons dans le graphe $G3$ les noeuds E, F et G. De plus, nous associons au noeud E une capacité donnant accès à ce graphe. Nous pouvons maintenant déléguer la capacité pour $G3$ sachant que l'utilisateur ne pourra pas utiliser la capacité pour accéder à d'autres noeuds qu'à ceux appartenant a $G3$.

noeud	**époque noeud**	**graphe**	**époque graphe**	**cap-liste**	**époque cap-liste**
E	-	G1	1	cap_{G3}	1
	-	G3	1	-	-
F	-	G3	1	-	-
G	-	G3	1	-	-

Un dernier problème est la mise en relation entre le nom du sous-graphe et celui inscrit dans l'autorisateur. En effet, l'autorisateur nomme le graphe que le client peut consulter et évite qu'un serveur puisse déléguer des capacités appartenant à d'autres graphes. De plus, le nom du graphe est utilisé lors de la mise à jour de l'époque du graphe.

Nous résoudrons ce problème en précisant que lors de l'inclusion d'un graphe dans un autre, il faut créer une entrée contenant la concaténation des noms des deux graphes. Dans l'exemple précédent, supposons que ce nom est $G1 : G3$:

noeud	**époque noeud**	**graphe**	**époque graphe**	**cap-liste**	**époque cap-liste**
E	1	G	1	$\text{cap}_{G1:G3}$	1
	-	G1:G3	1	-	-
F	-	G1:G3	1	-	-
G	-	G1:G3	1	-	-

Lorsque l'on concatène les identificateurs de graphes, le serveur de noeuds doit vérifier que le graphe nommé par l'autorisateur est compris dans l'identificateur du graphe qui est inscrit dans la capacité. Lors de la mise à jour de l'époque du graphe $G1$, l'AS doit vérifier si le nom des graphes déclarés dans les entrées de l'ACLE ne comportent pas la préfixe $G1$.

En résumé, il est possible de combiner les deux approches. Cela requiert d'autoriser la délégation des listes de capacités dans les deux approches. Lors de l'inclusion d'une capacité CG dans une liste de capacités, il faut :

– regrouper tous les noeuds pouvant être accédés avec cette capacité dans un nouveau sous-graphe ;

– utiliser dans les entrées des ACLE des noeuds du nouveau graphe, un nom composé de celui du graphe d'origine et celui du nouveau graphe.

5.5 Révocation des capacités

Dans les sections précédentes nous avons décrit comment les capacités sont assignées aux clients et aux serveurs. Cette section examine le problème de révocation des capacités.

Le besoin pour la révocation peut se présenter dans plusieurs cas ; par exemple, lorsque l'on veut faire expirer la session de consultation d'un utilisateur, lorsque la sécurité d'un utilisateur ou d'un serveur est compromise ou lorsque l'état de sécurité d'un noeud évolue. Dans ces trois exemples, la nature de la révocation peut être de différents types :

- révocation de toutes les capacités possédées par un serveur,
- révocation de toutes les capacités possédées par un client,
- révocation de toutes les capacités pointant vers un noeud,
- révocation de toutes les capacités d'un graphe.

Le modèle prend en compte les besoins de révocation cités ci-dessus au moyen des attributs des capacités et des ACLE. Le modèle propose deux types de révocation, *implicite* et *explicite*, que nous allons maintenant décrire.

5.5.1 Révocation implicite des capacités

Toutes les capacités ont un attribut spécifiant leur période de validité. Une capacité est donc révoquée implicitement lors de l'expiration de cette période.

Cette méthode de révocation suppose que les serveurs du domaine peuvent se synchroniser faiblement par rapport à un horloge globale et refuser les capacités expirées. Le système DCE [RKF92] propose plusieurs mécanismes permettant d'atteindre cette synchronisation.

La révocation implicite concerne les capacités en possession des clients et des serveurs. L'expiration d'une capacité déléguée est spécifiée par la période de validité de l'autorisateur. En effet, cet attribut indique le temps alloué à la session de consultation de l'utilisateur. Dans le cas des capacités, la période de validité indique une estimation du temps pendant lequel les clés du SA et du serveur peuvent être considérées fiables.

5.5.2 Révocation explicite des capacités

Ce type de révocation permet de révoquer une capacité avant qu'elle n'expire. Nous décrivons dans la suite quelques techniques de révocation satisfaisant les besoins que nous avons cités en début de section. Ces techniques sont une adaptation au modèle de celles proposées dans [TMvR86, ECM89, VAB91] et que nous avons précédemment décrit dans la section 2.3.2.2, page 37.

- **Révocation de toutes les capacités possédées par un serveur.**

  Cette méthode requiert la communication de la clé publique du serveur à tous les autres serveurs du système. Un serveur doit refuser toutes les capacités dont la valeur de l'attribut KU_{deleg} correspond à une clé publique révoquée.

- **Révocation de toutes les capacités possédées par un client.**

  Cette méthode est similaire à la précédente, sauf que l'on transmet aux serveurs de noeuds l'identificateur de l'autorisateur.

  Notons que dans le cas d'une authentification des clients par leur adresse réseau, la révocation des capacités impliquera la révocation de toutes les capacités provenant du même adresse réseau. Tant que cette révocation reste en place, aucune autre capacité déléguée ne pourra pas être utilisé depuis cette adresse.

- **Révocation de toutes les capacités pointant vers un noeud.**

  Cette méthode est relativement simple. Il s'agit d'incrementer la valeur de l'époque du noeud. En effet, comme nous l'avons décrit précédemment, un serveur refuse les capacités ayant une époque inférieure à celle du noeud.

- **Révocation de toutes les capacités d'un graphe.**

  Cette méthode est similaire à la précédente, sauf que l'on change l'époque du graphe dans toutes les ACLE associées aux composantes du graphe. De plus, il faut mettre à jour l'époque du graphe dans le serveur d'autorisation. Notons que l'avantage de cette méthode est que l'accès au noeud dans le contexte d'autres graphes n'est pas affecté. Un autre avantage de cette approche est que la révocation d'un accès n'implique pas la régénération des capacités stockées dans les serveurs.

Les méthodes que nous venons de décrire présentent quelques inconvénients. Les premières deux méthodes requièrent que les serveurs d'objets stockent une liste de clés publiques ou d'identificateurs uniques d'autorisateurs ayant expiré. Les serveurs doivent maintenir cette liste jusqu'à que les capacités révoquées expirent.

Dans toutes ces méthodes de révocation sauf la troisième, l'AS doit mettre à jour d'une manière cohérente tous les serveurs impliquées. La mise en oeuvre de cette opération peut être assez compliquée et lente. En effet, elle dépend de l'infrastructure de communication et de la disponibilité des serveurs. Il s'agit d'ailleurs d'un problème classique de révocation dans les systèmes répartis à capacités [VAB91] mais qui est compliqué par le nombre de serveurs devant être mis à jour.

Plusieurs études sur les systèmes à capacités montrent que la révocation explicite des capacités est une opération rare [VAB91]. Cela diminue le problème de stockage d'un grande nombre d'informations de révocation. Néanmoins, le problème de propagation des identités de capacités révoquées peuvent rendre ces méthodes peu fiables.

5.6 Analyse informelle des propriétés de sécurité du modèle

Dans cette section nous considérons quelques unes des attaques les plus importantes pouvant être appliquées à une instance du modèle. Plus spécifiquement, nous analysons les attaques où un intrus essaie d'utiliser les capacités du domaine. Nous considérons trois grandes types d'attaques :

- l'infiltration d'un serveur par un intrus,
- l'infiltration d'un client par un intrus,
- l'infiltration des clients,
- le vol des capacités déléguées pendant leur échange,
- le rejeu des requêtes.

Nous décrivons pour chacune de ces attaques les mesures de prévention inhérentes au modèle, les conséquences de l'attaque et quelques stratégies permettant de récupérer l'état de sécurité du système.

5.6.1 Infiltration des serveurs

Nous définissons l'infiltration d'un serveur comme la prise en main de toutes les ressources d'un serveur par un intrus. Nous analysons en particulier le cas où l'intrus essaie de se servir des capacités stockées dans un serveur pour accéder aux noeuds stockés dans d'autres serveurs.

Tout serveur possédant une liste de capacités peut subir ce type d'attaque. Dans l'analyse qui suit, nous analysons la conséquence d'une attaque dans un serveur de noeuds et dans un serveur d'autorisation, respectivement.

Infiltration d'un serveur de noeuds

Considérons maintenant le cas d'un intrus ayant infiltré un serveur de noeuds. La signature des capacités protège celles-ci contre leur modification non autorisée. En effet, tout ce que l'intrus pourra réaliser avec les capacités d'un serveur est d'essayer de les déléguer, les capacités non déléguées ne pouvant pas être utilisées pour accéder à un serveur.

Afin que l'intrus puisse déléguer une capacité, il faut que plusieurs conditions soient satisfaites :

- l'intrus doit connaître la clé privée du serveur de noeuds,
- l'intrus doit posséder un autorisateur,
- l'autorisateur et la capacité doivent identifier le même graphe.

Si l'on protège l'accès à la clé privée du serveur, l'intrus ne pourra plus se servir des capacités pour accéder aux autres serveurs du domaine. Si l'intrus arrive à connaître la clé privée du serveur, il lui faudra avoir un autorisateur approprié pour pouvoir déléguer une capacité.

Supposons maintenant que l'intrus connaît la clé privée du serveur. Supposons également que l'intrus a acquis un autorisateur qui inclut un des noms des graphes inscrits dans les capacités du serveur.

L'intrus ne pourra déléguer une capacité que si celle-ci et l'autorisateur identifient le même graphe. De plus, si l'intrus ne connaît pas la clé privée du propriétaire de l'autorisateur, il ne pourra pas se servir de la capacité déléguée. Notons que dans ces deux cas, l'intervention de l'intrus sera sans effet : il n'aura fait que substituer le serveur de noeuds.

Lorsque un intrus découvre la clé privée d'un serveur de noeuds, il faut alors révoquer la clé publique de ce serveur, révoquer toutes les capacités possédées par ce serveur et, éventuellement, régénérer des nouvelles capacités. Notons que comme toute capacité inclut la clé publique d'un serveur de noeuds, la révocation de cette clé implique la révocation de toutes les capacités où elle est déclarée.

Infiltration du serveur d'autorisation

Le SA est plus sensible à cette attaque qu'un serveur de noeuds. En effet, le SA est responsable d'autoriser l'accès global aux graphes du domaine. Si l'intrus arrive à connaître la clé privée du serveur, il pourra autoriser à n'importe quel principal l'accès à n'importe quel graphe du domaine. Pour ce faire, il suffira que l'intrus génère un autorisateur et qu'il délègue une des capacités stockées chez le SA.

Les mesures à suivre lors de la détection de ces attaques sont similaires à celles prises lors de l'infiltration d'un serveur de noeuds : la révocation de la clé publique du SA et la régénération de toutes les capacités stockées dans le SA.

En résumé, l'infiltration d'un serveur de noeuds peut facilement être contrecarrée par les attributs inscrits dans l'autorisateur et dans les capacités. L'infiltration d'un serveur d'autorisation peut impliquer l'ouverture d'un accès non autorisé à tous les noeuds du domaine. Les mesures à prendre dans les deux types d'infiltration sont les mêmes : révocation de la clé publique du serveur infiltré et la régénération des capacités possédées par les serveurs.

5.6.2 Infiltration des clients/vol des capacités échangées

Nous allons maintenant examiner les attaques où un intrus essaie de récupérer et d'utiliser une capacité déléguée. L'intrus peut obtenir cette capacité soit en infiltrant le client, soit en l'interceptant lors de son échange sur le réseau.

Afin que l'intrus puisse utiliser une capacité déléguée, il lui faudra connaître la clé privée du client. Si l'on identifie additionellement le client par son adresse réseau, l'intrus devra émettre ses requêtes depuis cette adresse.

Pour récupérer l'état de sécurité, il faut empêcher que l'intrus puisse continuer à s'authentifier sous l'identité du client. Pour ce faire, il faut indiquer à tous les serveurs du domaine de ne plus accepter des capacités déléguées incluant la clé compromise. Il faudrait également accorder un nouveau paire de clés asymétriques au client.

5.6.3 Rejeu des requêtes

Dans cette attaque, on suppose qu'une requête comprend une capacité et une preuve d'authentification du client (cf. section 2.2.4.4 page 32). On suppose également que ces deux éléments sont liés de manière indissociable à la requête par une signature numérique. L'attaque consiste en l'enregistrement des requêtes et en leur rejoue ultérieur.

Il existe deux possibles solutions contre cette type d'attaques. La premier solution concerne l'authentification dynamique du client. Il s'agit d'inclure une différente preuve d'authentification dans chaque requête du client. Par exemple, il peut s'agir d'un nonce envoyé par le serveur et que le client doit inclure dans sa requête signée. Le serveur pourra alors détecter le rejoue des requêtes. La deuxième solution consiste en chiffrer les données envoyés. On n'interdit pas ici le rejoue, mais on empêche à l'intrus la lecture de la réponse du serveur.

5.6.4 Résumé de l'analyse des attaques

Le tableau suivant résume les différentes attaques présentées précédemment, leurs effets et les mesures nécessaires pour récupérer l'état de sécurité du domaine.

attaque	**effet**	**mesures de récupération**
Infiltration d'un serveur de noeuds. Infiltration du SA.	Compromis de des données du serveur. Compromis des graphes du	Révocation globale des clés asymétriques du serveur. Régénération des capacités du serveur. Révocation globale de la clé publique du SA. Régénération des capacités du SA.
Infiltration du client / vol des capacités déléguées.	Si l'intrus peut s'authentifier comme le client, il pourra accéder aux mêmes données que celui-ci.	Révocation globale de toutes les capacités déléguées comprenant l'identificateur de l'autorisateur compromis. Changement des clés asymétriques du client.
Rejoue des requêtes vers les serveurs de noeuds.	L'intrus peut voir la réponse du serveur de noeuds.	Si le protocole d'authentification n'offre pas un défi dynamique et si les données échangées ne sont pas chiffrées, révoquer toutes les capacités déléguées comprenant l'identificateur du client.

5.7 Synthèse et discussion

Après une brève synthèse des principales caractéristiques du modèle, nous présentons une discussion organisée autour des deux points suivants :

- limitations du modèle,
- travaux apparentés.

5.7.1 Synthèse

Nous avons proposé dans ce chapitre un modèle de contrôle d'accès non nominatif adapté aux systèmes hypermédia. Ce modèle s'inspire des approches réparties pour la localisation de l'information de contrôle d'accès dans le système. Par rapport à ces approches, notre modèle présente l'avantage d'avoir une meilleure efficacité pour la gestion des utilisateurs que celle des autres modèles. De plus, le modèle permet l'inclusion d'un noeud protégé dans plusieurs graphes, un besoin qui n'avait pas été traité auparavant dans les modèles existants.

Dans notre modèle, on organise les graphes dans des *domaines d'autorisation*. Chaque graphe dans le domaine a un nom unique. Un administrateur de sécurité associe à chaque noeud une liste de contrôle d'accès étendue dont les entrées spécifient les noms de graphes qui incluent le noeud et leurs droits d'accès. Ces droits sont non nominatifs car ils n'identifient aucun utilisateur ou groupe d'utilisateurs.

Afin de pouvoir consulter un noeud, un utilisateur doit inclure dans sa requête une capacité qui spécifie le nom d'un graphe. Le modèle utilise un système de protection à deux niveaux pour l'acquisition et l'utilisation des capacités. Le premier niveau correspond à une *protection globale* où un serveur d'autorisation octroie à l'utilisateur une première capacité. Le deuxième niveau est une *protection locale* où un serveur d'objets valide une capacité et vérifie que les droits d'accès associés au graphe prennent en compte la requête.

Nous avons considéré deux approches pour l'assignation des capacités aux utilisateurs : l'approche *une capacité par noeud* (CN) et l'approche *une capacité par graphe*. Dans l'approche **CN**, l'administrateur de sécurité assigne à certains noeuds d'un graphe des capacités donnant accès aux autres noeuds. Lorsque l'utilisateur parcourt un noeud, il récupère en même temps sa liste de capacités. Dans l'approche **CG**, l'utilisateur peut consulter tous les noeuds du graphe avec une seule capacité. Ces deux approches sont indépendantes, mais peuvent être combinées.

Afin de diminuer les risques d'infiltration d'un serveur de noeuds, toutes les capacités dans le domaine sont générées à l'avance par un administrateur de sécurité. Un serveur octroie alors ces capacités à ses clients au moyen d'une opération de délégation. Les capacités sont révoquées de manière implicite par l'inclusion d'une date d'expiration dans leur structure. Le modèle prévoit également plusieurs techniques de révocation des capacités avant que celles-ci n'expirent.

5.7.2 Limitations du modèle

Cette section décrit les limitations les plus importantes de notre modèle de contrôle d'accès

Vulnérabilité du serveur d'autorisation. Le compromis de la sécurité de ce serveur peut impliquer un accès non autorisé à tous les graphes du domaine. En effet, le SA est responsable de la génération de l'autorisateur, un des éléments fondamentaux pour la protection des capacités.

Eventuelle manque de performance. La validation et la délégation des capacités requièrent plusieurs opérations cryptographiques. Ces opérations étant onéreuses, le temps de réponse d'une requête dépend de la qualité de mise en oeuvre du modèle et de sa plateforme cryptographique. L'approche **CN** peut présenter également des problèmes de performance : une session de consultation peut nécessiter la délégation de plusieurs capacités. Afin de minimiser ces opérations, nous avons proposé deux extensions à l'approche **CN** dans la section 5.4.2.7, page 114.

Régénération des capacités. Lorsqu'une capacité expire ou est révoquée, il faut une intervention de la part de l'administrateur de sécurité pour générer une nouvelle capacité. Si plusieurs capacités expirent en mêmes temps, les limites matérielles de l'administrateur pourraient être vite dépassés. Un cas plus pessimiste se présente lorsque la clé d'un serveur est compromise. Il faudra alors régénérer éventuellement un grand nombre de capacités avant que le service ne soit normalisé.

Annotations. Dans une session de consultation type, un utilisateur peut choisir de stocker l'URI d'un noeud dans un fichier d'annotations. Il pourra ultérieurement ouvrir ce fichier est parcourir n'importe laquelle des URL qu'il contienne. Dans les approches proposées, il faut que le client ait une capacité appropriée pour pouvoir consulter un noeud. Si l'utilisateur a une capacité pour un noeud et choisit de sauvegarder l'URI de celui-ci, il pourra y revenir autant de fois qu'il veut avant que la capacité expire. Par la suite, il lui faudra renouveler la capacité afin de réactiver son annotation. Dans l'approche **CG**, cela ne représente aucun problème car une capacité est utilisé pour consulter tous les noeuds d'un graphe. Cependant, dans l'approche **CN**, l'utilisateur pourrait bien être forcé de refaire une partie du parcours du graphe avant d'obtenir la capacité désirée. Cette limitation peut être partiellement évitée en définissant plusieurs points d'entrée dans un graphe et en stockant des capacités y autorisant l'accès dans le SA.

Evaluation de la durée d'une session de consultation. Un autorisateur indique le temps que le SA a accordé au client pour la consultation d'un graphe. Il n'est pas simple d'estimer cette durée : il faut prendre en compte le temps que l'utilisateur prend à assimiler l'information de chaque noeud du graphe, la charge du réseau et des différents serveurs, etc. Si l'on n'estime pas correctement la période de validité de l'autorisateurt, un utilisateur pourrait n'avoir assez de temps pour parcourir tous les noeuds appartenant à un graphe. Notons que cette limitation affecte principalement l'approche **CN**, où l'utilisateur doit suivre un parcours déterminé. Dans l'approche **CG**, il suffit que l'utilisateur récupère une nouvelle capacité pour continuer sa session de consultation. Une solution à cette contrainte est l'utilisation de durées de vie relativement longues pour les autorisateurs.

5.7.3 Travaux apparentés

Dans cette section, nous comparons notre modèle avec deux travaux apparentés :

- le système DCE/Web, sur lequel nous nous sommes inspirés pour la localisation répartie de l'ACI,
- les approches de protection par rôles, qui présentent également la notion de droits d'accès non nominatifs.

Pour chacun de ces systèmes, nous décrivons les principaux points en commun avec notre modèle et les avantages que celui-ci apporte par rapport à la gestion des utilisateurs et au partage des noeuds. Enfin, nous donnons quelques suggestions sur comment notre modèle peut être combiné avec ces travaux.

Comparaison avec DCE/Web

DCE/Web [LZ94, Lew95] présente un bon compromis d'efficacité car il distingue entre deux types d'ACI, celle qui est locale aux serveurs de noeuds et celle qui est globale au système. L'ACI locale spécifie les droits d'accès des utilisateurs ou des groupes des utilisateurs, tandis que l'ACI globale spécifie les groupes auxquels appartient un utilisateur et son identificateur unique. Ces deux types d'ACI sont donc nominatifs.

Dans notre modèle, l'ACI locale est exclusivement non nominative. Elle représente les droits d'accès accordés à chaque graphe pour chaque objet. L'ACI globale spécifie les graphes qu'un utilisateur peut consulter.

Dans certains cas, l'utilisation des groupes simplifie de gestion des utilisateurs : on peut attribuer des droits d'accès à un utilisateur en incluant son nom dans un groupe. Cependant, la création d'un nouveau groupe d'utilisateurs peut nécessiter la mise à jour de l'ACI de plusieurs serveurs du système.

Dans notre modèle, tout spécification des droits d'accès d'un utilisateur ou d'un groupe d'utilisateurs peut s'effectuer dans le serveur d'autorisation. Il ne faut pas mettre à jour l'ACI des serveurs que lors de la création des nouveaux graphes.

Dans DCE/Web, une ACL est associé à chaque objet et définit pour chaque utilisateur ses droits en terme d'opération sur cet objet. Nous avons démontré dans le chapitre précédent que cette spécification ne permet pas donc de prendre en compte le partage de noeuds protégés.

Notre modèle se sert également des ACL pour spécifier les droits d'accès sur les objets. Par opposition à DCE/Web, on substitue les noms des utilisateurs dans une ACL par les noms des graphes qui incluent l'objet. L'inclusion du nom du graphe permet d'organiser automatiquement les droits d'accès de chaque graphe qui inclut le noeud. Nous pouvons ainsi partager un noeud protégé sans ambiguïté.

Il est possible de combiner l'approche **CG** a DCE/Web, si (1) on ne stocke que des groupes dans les ACL et (2), si on assimile un nom de graphe par groupe. On peut alors profiter de l'architecture DCE/Web, tout en ayant une simplification de la gestion des utilisateurs et la possibilité de partager des noeuds partagés

En résumé, notre modèle spécifie une organisation de l'ACI par graphes, un échange des capacités dans les requêtes et une autorisation globale par un serveur de sécurité. DCE/Web propose des mécanismes de contrôle d'accès plus généraux qui ne prennent pas en compte la structure des graphes hypermédia. Il est possible d'utiliser l'approche **CG** de notre modèle dans DCE/Web si l'on assimile un graphe à un groupe.

Comparaison avec les modèles de contrôle d'accès basés sur des rôles

L'approche que nous adoptons pour simplifier la gestion des utilisateur au moyen des droits d'accès non nominatifs est similaire à celle prise dans les modèles de contrôle d'accès basés sur des rôles (RBAC) [FCK95, SCFY96]. Un rôle décrit une fonction au sein d'une entreprise ou d'une organisation ; par exemple, `président`, `secrétaire`, `chercheur`, etc. On associe à chaque rôle l'ensemble droits d'accès sur les objets du système qui lui sont nécessaires pour exercer ses fonctions. Ces droits peuvent être spécifiés sous forme des ACL associées aux objets, ou bien être inclus à l'

Comme pour les groupes, la mise en relation d'un utilisateur avec un rôle permet à l'utilisateur de profiter des droits d'accès attribués au rôle. Un utilisateur peut appartenir à un ou plusieurs rôles. On stocke dans une base de données d'autorisation les correspondances entre utilisateurs et rôles.

Quelques travaux de recherche récents proposent l'intégration du concept de rôles au WWW [BCF+97, GNE+97]. Ces travaux se sont appliqués jusqu'à maintenant aux problèmes de mise en place des mécanismes permettant l'association des rôles aux utilisateurs et l'évaluation des droits associés à un rôle lors de la réception d'une requête. Dans leur état actuel, ces travaux présentent deux inconvénients importants :

Unité de protection. La définition des droits d'accès d'un rôle est très générale et ne considère que les noeuds comme des unités de protection. Il est donc à la charge de l'administrateur de sécurité de garantir qu'un utilisateur aura un droit d'accès sur tous les éléments d'un graphe.

Partage de noeuds. Les approches n'organisent pas les droits d'accès des rôles sur un objet selon le graphe où ils sont spécifiés. Comme pour les groupes, cette manque introduit une ambiguïté lorsqu'un rôle a des droit d'accès sur deux graphes qui incluent un même noeud. En effet, on ne peut pas différencier alors entre les droits spécifies pour un graphe ou pour l'autre.

Notre modèle et les modèles RBAC ont plusieurs points en commun. Les deux modèles utilisent des droits d'accès non nominatifs pour la simplification de la gestion des utilisateurs. En outre, toutes les informations nominatives, et donc, susceptibles d'évoluer rapidement dans le temps, sont stockées dans un seule base de données.

Notre modèle et les modèles RBAC diffèrent dans la spécification des droits d'accès et dans l'organisation de ces droits. Dans le modèle proposé, nous considérons un graphe comme l'unité de protection. Nous spécifions donc les droits d'accès non nominatifs par rapport à l'appartenance des noeuds dans un graphe. Cette spécification permet le partage de noeuds protégés car elle organise implicitement les droits d'accès par graphes.

Tandis qu'un modèle RBAC spécifie les droits d'accès nécessaires pour réaliser une fonction au sein d'une organisation, notre modèle ne précise que les droits d'accès dont on a besoin pour consulter un graphe. Nous pouvons donc considérer notre modèle comme un modèle RBAC dégénéré où un rôle ne permet que la consultation d'un graphe.

Il est possible de combiner les deux modèles si l'on définit les droits d'accès d'un rôle en termes de graphes. Un rôle devient alors une spécification d'haut niveau qui indique l'ensemble de graphes qui peut être consulté. Les graphes devient des spécifications de niveau intermédiaire qui indiquent les noeuds qui peuvent être consultés. Cette combinaison des deux modèles permet de partager des noeuds protégés et de garantir qu'un rôle pourra accéder à tous les composantes d'un graphe.

En résumé, bien que notre modèle ressemble aux modèles RBAC, notre modèle spécifie et organise les droits d'accès sur les noeuds en termes de graphes. En comparaison, un modèle RBAC spécifie les droits d'accès associés à un rôle à l'intérieur d'une organisation. Il est toutefois possible de combiner les deux modèles si l'on définit les droits d'accès d'un rôle en termes de graphes.

Chapitre 6

Prototypage et expérimentation

EN PRATIQUE en fait, dans l'exécution.
Des décisions qu'il faut mettre en pratique,
exécuter, réaliser, concrétiser.

(*Micro-Robert, Dictionnaire de Langue Française)*

Ce chapitre décrit la mise en oeuvre de CAMW3 [1], un prototype qui applique les concepts de notre modèle de contrôle d'accès au système World-Wide Web. L'objectif de cette réalisation est d'avoir une première idée sur les problèmes liés à la mise en oeuvre du modèle sur un système existant. En particulier, nous voulons évaluer les problèmes liés au traitement et à l'échange des capacités, tant au niveau interface qu'au niveau performance. Le système WWW nous a semblé un bon environnement de tests puisqu'il s'agit d'un système hypermédia réparti à échelle mondiale.

Nous donnons d'abord une présentation générale de l'architecture logicielle choisie pour la réalisation. Nous décrivons ensuite la réalisation des différents composants de CAMW3, en insistant sur les aspects d'association des capacités aux objets et sur le traitement de ces capacités. Un bilan de la réalisation conclut ce chapitre.

6.1 Présentation générale de CAMW3

Nous allons maintenant présenter l'architecture générale de CAMW3. CAMW3 est construit sur le système WWW développé au National Center for Supercomputing Applications (NCSA) de l'Université d'Urbana Champaign, Illinois [VSW94, YM96]. Le système NCSA/WWW est constitué d'un client graphique, appelé `Mosaic`, et d'un serveur HTTP, connu par l'acronyme de `httpd`. Nous avons choisi ce système en raison de la disponibilité du code source et de la présence d'une API cryptographique.

Dans la suite de cette section, nous donnons d'abord quelques caractéristiques du système NCSA/WWW. Nous décrivons ensuite l'architecture générale de CAMW3.

1. CAMW3 a été présenté dans le Colloque Francophone sur l'Ingénierié des Protocoles (CFIP'95) à Rennes, France.

6.1.1 Rappels sur le système NCSA/WWW

L'objectif de cette section est de décrire les caractéristiques du système NCSA/WWW qui sont essentielles pour la réalisation de CAMW3. On trouvera dans [YM96] une description plus détaillée du projet NCSA/WWW. Stein [Ste95, Chaps. 3-4] donne plus de précisions sur la configuration de ce système.

Nous nous concentrons sur trois aspects de NCSA/WWW :

- les caractéristiques générales du système : le modèle de données, la désignation des URL et la migration des noeuds ;
- la mise en place du contrôle d'accès ;
- le déroulement d'une transaction.

6.1.1.1 Caractéristiques générales

Le système NCSA/WWW intègre plusieurs des concepts des systèmes hypermédia que nous avons exploré dans le chapitre 3. Nous en rappelons ici ceux qui sont les plus importants du point de vue de CAMW3 :

- *Modèle et représentation des données.* Comme tout système WWW, NCSA/WWW suit le modèle de données défini par la norme HTML [Rag97]. Dans HTML, les liens sont imbriqués à l'intérieur des noeuds sources. Les noeuds peuvent être du type statique ou calculé. Dans le cas des noeuds statiques, un noeud est représenté par un fichier. Dans le cas des noeuds calculés, un noeud est généré par un script invoqué par le serveur.
- *Désignation des noeuds.* Elle suit le formalisme des URL [BLe94], où le nom d'un noeud spécifie l'adresse du serveur qui héberge le noeud et le chemin vers le noeud à l'intérieur du serveur. Dans le cas des noeuds calculés, un lien précise le nom d'un script et peut inclure zéro ou plusieurs paramètres d'exécution.
- *Migration des noeuds.* La migration des noeuds suit le protocole de redirection, décrit dans la section 3.6. Lorsqu'un noeud migre vers un autre serveur, il laisse derrière lui une URL pointant vers sa nouvelle destination. Cette déclaration s'effectue à l'intérieur du fichier de configuration du serveur.

Notons que toutes ces caractéristiques font partie de la spécification de base WWW. On les retrouve donc dans la plupart des systèmes WWW qui existent aujourd'hui.

6.1.1.2 Contrôle d'accès

Nous allons maintenant présenter les aspects concernant la spécification des droits d'accès des utilisateurs et l'authentification des utilisateurs.

Spécification des droits d'accès

NCSA/WWW suit une approche décentralisée, où chaque serveur stocke l'ACI nécessaire pour contrôler l'accès à ses objets. On trouve trois types d'ACI : des informations pour l'authentification des utilisateurs [2], des listes de groupes d'utilisateurs et des listes de contrôle d'accès (ACL).

On spécifie les droits d'accès des utilisateurs au moyen des *règles de protection* [Ste95, pp. 131-144]. Une règle de protection comprend essentiellement trois éléments :

- le nom d'un fichier de groupes,
- le nom d'un répertoire dans le serveur,
- une ACL.

L'ACL spécifie les droits d'accès des utilisateurs et des groupes d'utilisateurs sur le contenu du répertoire, y compris toute arborescence qui y découle. Un répertoire fils peut avoir des règles de protection différentes à celles de son répertoire père.

Authentification des utilisateurs

Depuis sa conception, le système NCSA/WWW a été une plateforme d'expérimentation pour divers protocoles d'authentification [Cai96]. Nous nous intéressons en particulier à la mise en oeuvre du protocole expérimental d'authentification à clé publique PGP/PEM [McC]. Il s'agit d'un protocole simple où le client signe ses requêtes et où le serveur chiffre ses réponses avec la clé publique du client. On suppose que le serveur connaît *a priori* la clé du client et peut ainsi vérifier la signature. Le lecteur trouvera dans l'annexe D une description détaillée de ce protocole.

La mise en place de PGP/PEM s'appuie sur un système cryptographique externe au client et au serveur. Les appels à ce système s'effectuent au moyen d'une API spécialisée. Nous nous inspirons de cette API pour ajouter une couche d'autorisation supplémentaire aux clients et aux serveurs.

6.1.1.3 Déroulement d'une transaction dans NCSA/WWW

Les clients et les serveurs NCSA/WWW sont composés de plusieurs modules. Nous pouvons citer les module de restitution de données, les modules de protocoles de communication (HTTP, FTP, NNTP, ...), les modules d'authentification, les module d'autorisation, etc.

Une transaction se propage à travers les différents modules du client et du serveur sous forme d'un flot de données. Chaque module analyse le flot de données et, si c'est nécessaire, effectue un traitement sur celui-ci.

2. Par exemple, des mots de passe, des clés publiques, etc.

On distingue deux étapes dans une transaction : la *requête* et la *réponse* (fig. 6.1). Nous allons maintenant décrire brièvement chacune de ces étapes, en soulignant l'interaction avec le module d'authentification PGP/PEM et le module d'autorisation du serveur.

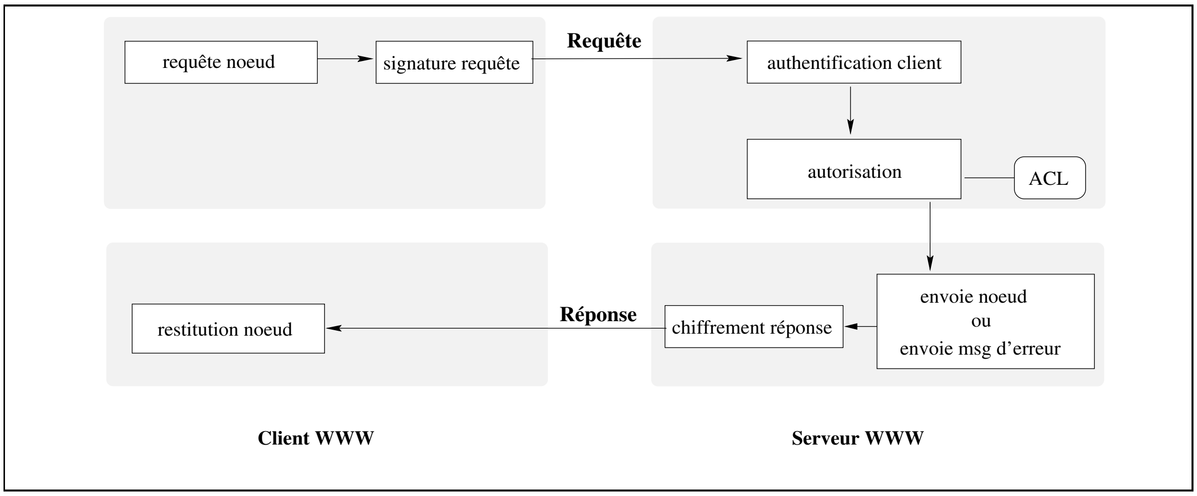


Fig 6.1: Modèle d'une transaction dans NCSA/WWW

La requête. Lorsque l'utilisateur demande le parcours d'un noeud n, le client forme d'abord une requête HTTP classique. Le client envoie ensuite cette requête à un module de signature, où la requête est signée avec la clé privée de l'utilisateur. Le client utilise ensuite le message signé comme le corps d'une autre requête HTTP. La figure 6.2 résume ces deux étapes.

Lors de la réception de la requête, le serveur authentifie l'utilisateur en vérifiant la signature de la requête originale HTTP. Dans NCSA/WWW, on suppose que le serveur connaît la clé publique de l'utilisateur. Le serveur vérifie ensuite si l'utilisateur a le droit d'accéder à l'objet en comparant l'ACL associée à l'objet avec l'opération demandée par l'utilisateur.

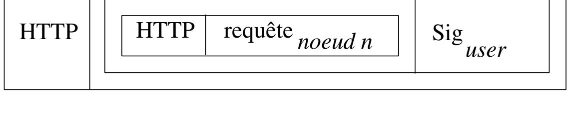


Fig 6.2: Requête HTTP-PGP/PEM

La réponse. Si l'authentification ou si l'autorisation échouent, le serveur retourne au client un message d'erreur. Autrement, le serveur prépare une réponse HTTP contenant le noeud demandé. Si le noeud a migré vers un autre serveur, la réponse inclut alors la nouvelle URL du noeud. Le serveur chiffre ensuite la réponse, quelqu'elle soit, avec la clé publique de l'utilisateur. La figure 6.3 montre la réponse du serveur lorsque celle-ci contient un noeud.

Enfin, le client déchiffre la réponse avec sa clé privée, Le client traite alors le contenue du noeud et le présente à l'utilisateur.

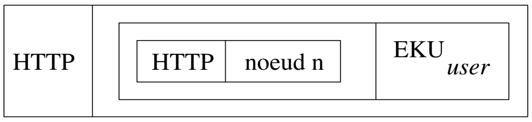


Fig 6.3: Réponse HTTP-PGP/PEM

En résumé, le protocole PGP/PEM tel que nous l'avons brièvement décrit, est composé de deux opérations : la signature des requêtes avec la clé privée du client et le cryptage des réponses avec la clé publique du client. Une variation de ce protocole permet l'authentification serveur. Il s'agit de chiffrer les requêtes avec la clé publique du serveur et de signer les réponse avec la clé privée de ce dernier.

Bien que ce protocole d'authentification soit vulnérable aux attaques de rejeu des messages, il est intéressant du point de vue applicatif. Il s'agit d'un exemple simple qui montre l'intégration modulaire des protocoles d'authentification. Le lecteur trouvera dans l'annexe D une description plus détaillée du protocole PGP/PEM.

6.1.2 Architecture générale de CAMW3

CAMW3 reprend les concepts de modules et de communication entre modules par des flots de données. Nous avons défini pour les clients et pour les serveurs des nouveaux modules d'autorisation pour le traitement des capacités. Nous avons également étendu le protocole PGP/PEM pour enlever la contrainte dans laquelle les serveurs doivent connaître les clés publiques de leurs clients potentiels.

La réalisation de CAMW3 comprend quatre classes d'acteurs (fig. 6.4) :

- des clients,
- des serveurs d'objets,
- un serveur d'autorisation,
- un administrateur de sécurité.

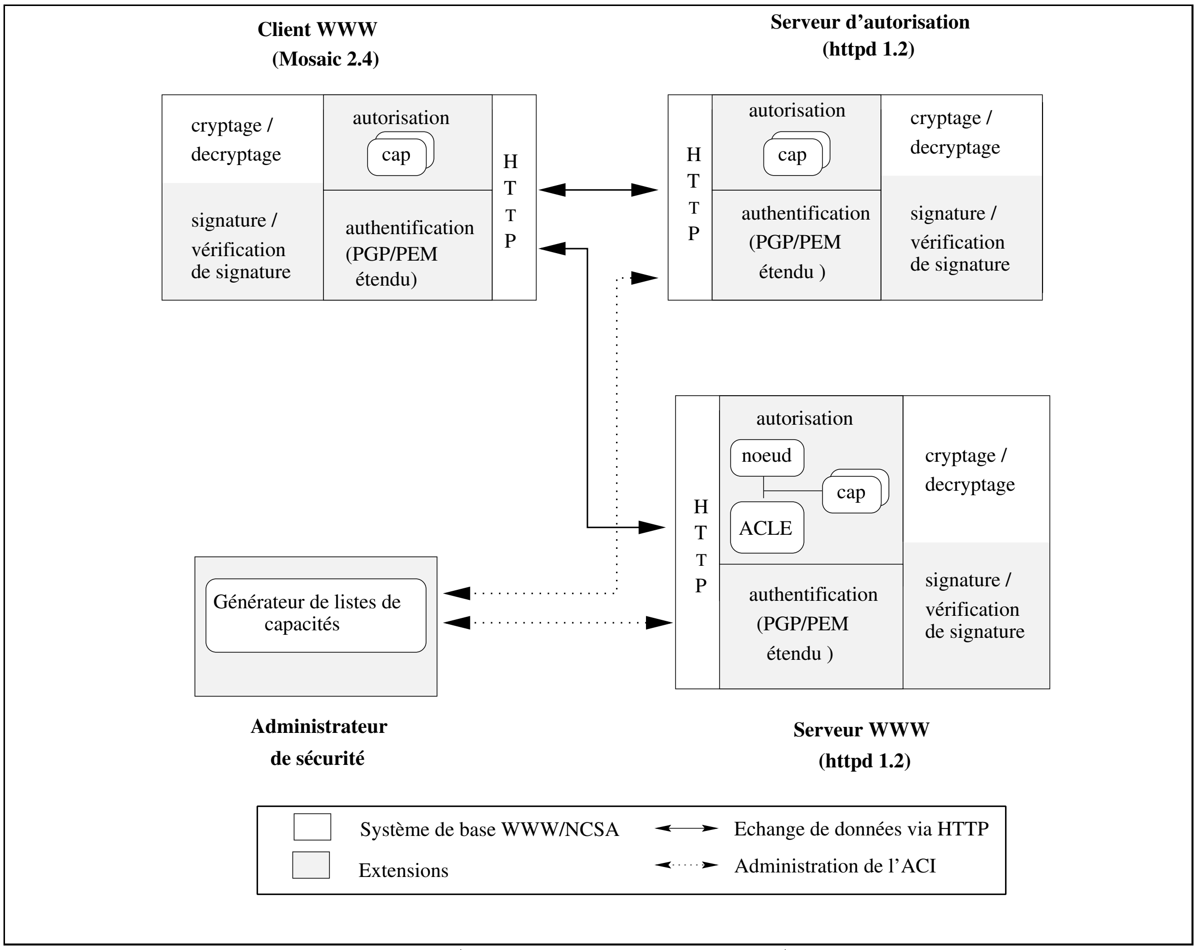


Fig 6.4: Architecture globale de CAMW3

L'administrateur de sécurité est responsable de la génération des capacités et de leur installation dans les différents serveurs du système. Il est également responsable de la mise à jour des ACLE associées aux noeuds. CAMW3 ne propose qu'un outil pour l'administrateur de sécurité : un script générateur de listes de capacités. L'installation de celles-ci dans les différents serveurs est réalisée manuellement, en utilisant des protocoles classiques tels que NFS et FTP.

Le client et les deux classes de serveurs sont ceux proposés par le système NCSA/WWW, augmentés des trois extensions suivantes :

- un nouveau module d'autorisation pour les serveurs et pour les clients, responsable de l'échange et du traitement des capacités.
- une API pour le module d'autorisation ;
- l'échange des capacités dans le corps du message PGP/PEM.

Toutes ces modifications ne sont actives que lors de l'accès à un objet protégé par des capacités. Dans tous les autres cas, le client et le serveur opèrent d'après la spécification NCSA/WWW.

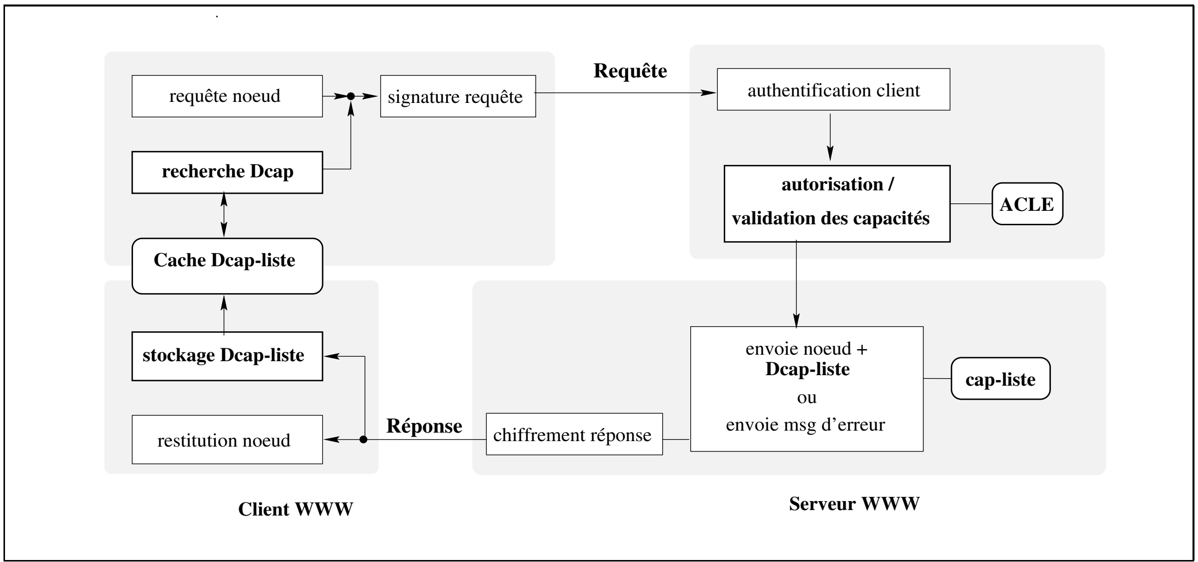


Fig 6.5: Modèle d'une transaction dans CAMW3

6.1.3 Déroulement d'une transaction dans CAMW3

CAMW3 introduit lors des transactions l'appel aux nouveaux modules d'autorisation et l'échange des capacités dans le corp des requêtes. Nous avons également modifié le protocole PGP/PEM afin d'éliminer la contrainte qui impose à un serveur de connaître les clés publiques de ses clients potentiels.

L'échange des capacités présente un problème particulier. En effet, le protocole HTTP ne prévoit pas de champ spécifique pour l'échange d'autres informations que celles liées à l'authentification. Nous avons résolu ce problème en concaténant les capacités au corps des messages HTTP avant de les transmettre et en les enlevant avant leur traitement.

Nous allons maintenant décrire les modifications apportées par CAMW3 à la requête et à la réponse dans une transaction. La figure 6.5 illustre le flot de données entre les différents modules du client et du serveur.

La requête. Dans cette opération, le client joint une capacité appropriée à son message et le signe. Comme nous le verrons dans la section 6.2.4, le client stocke les capacités qui lui sont attribués dans un *cache* local. Chaque entrée du cache consiste d'une capacité et de l'époque de la liste de capacités (*cap-liste*) associée au noeud cible. Avant de signer la requête, le client appelle le module d'autorisation. Celui-ci recherche dans le cache s'il a une capacité déléguée donnant accès au noeud. Si c'est le cas, il joint cette capacité et son époque à la requête. Comme preuve de son identité, le client signe le message qui en résulte avec sa clé privée. Cette clé correspond à la clé publique inscrite dans l'autorisateur des capacités.

Notons que le serveur n'a pas besoin de connaitre la clé publique du client *a priori* car celle-ci est inscrite à l'intérieure de la capacité déléguée. Afin de permettre au serveur de vérifier le message signé, nous concaténons une copie de cette clé au message signé.

Le client envoi enfin le message qui y résulte de ces opérations dans le corps d'un nouveau message HTTP (fig. 6.6).

Lors de la réception de la requête, le serveur récupère la clé publique de l'utilisateur et le message signé. Il authentifie ensuite le client en vérifiant cette signature. Le serveur vérifie également que la clé publique du client correspond à celle spécifiée dans l'autorisateur. Enfin, le serveur utilise la capacité pour autoriser l'accès à l'objet.

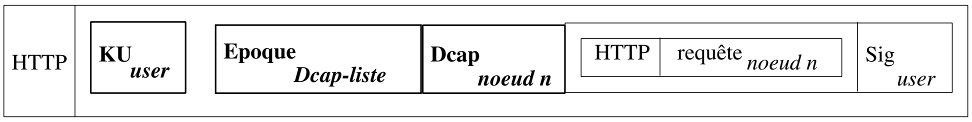


Fig 6.6: Modifications à la requête HTTP PGP/PEM

La réponse. Si l'authentification ou si l'autorisation échouent, le serveur retourne au client un message d'erreur. Dans le cas contraire, le serveur prépare une réponse HTTP contenant le noeud demandé. Si le noeud est associé à une liste de capacités et si l'époque de cette liste est supérieure à celle envoyée par le client, le serveur délègue les capacités et concatène celles-ci et leur époque au message HTTP. Le serveur chiffre ensuite la réponse avec la clé publique du client. Si le noeud a migré vers un autre serveur, la réponse inclut alors la nouvelle URL du noeud et une capacité donnant accès au noeud dans sa nouvelle localité (fig. 6.7).

Enfin, le client déchiffre la réponse avec sa clé privée, Le client traite alors le contenu du noeud et le présente à l'utilisateur.

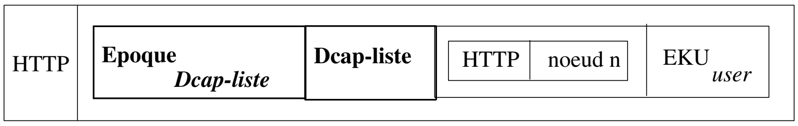


Fig 6.7: Modifications à la réponse HTTP PGP/PEM

En résumé, les clients et les serveurs échangent des capacités en les concatenant aux messages HTTP, avant que ceux-ci soient signés ou chiffrés. Comme le but de l'authentification est de vérifier que le client est le propriétaire de la capacité, il suffit que le client signe sa requête avec la clé privé qui corresponde à la clé publique inscrite dans l'autorisateur. En envoyant cette clé dans la requête, nous évitons au serveur le besoin de la connaître *a priori*.

6.2 Mise en oeuvre de CAMW3

Nous allons maintenant donner les principales caractéristiques pour chacun des composants de CAMW3, à savoir, le script *générateur de listes de capacités*, les *clients*, le *serveur d'autorisation* et les *serveurs de noeuds*. Le lecteur trouvera dans l'annexe E un résumé des principales API que nous utilisons dans CAMW3.

6.2.1 Générateur de listes de capacités

Une des fonctions de l'administrateur de sécurité est la génération et l'installation des listes de capacités dans les différents serveurs du système. On distingue trois types de listes de capacités :

- La liste de capacités du serveur d'autorisation. Elle comprend les capacités pour les noeuds racines et pour les graphes.
- Les listes de capacités associées aux noeuds du domaine. Le contenu de ses listes est établit selon la politique d'assignation des capacités.
- Des listes de capacités de "redirection." Lors de la migration d'un noeud d'un serveur à un autre, le serveur d'origine récupère une capacité donnant accès au noeud dans son nouvel emplacement. Un serveur WWW regroupe toutes ses capacités de "redirection" dans une seule liste.

On génère les listes de capacités au moyen d'un script et d'une métaliste. Chaque liste qui y est produite est stockée dans un fichier à part. Dans CAMW3, Le script ne comprend pas l'installation des listes dans les serveurs. Il s'agit en effet d'une opération que l'administrateur de sécurité doit effectuer manuellement.

6.2.1.1 Format de la méta-liste de capacités

La méta-liste spécifie en ASCII les propriétés des capacités. Chaque entrée de la méta-liste a deux éléments : une URL et une date d'expiration. La figure 6.8.a donne l'exemple d'une méta-liste contenant une capacité pour un noeud et une autre pour un graphe.

On distingue deux types des URL. Pour les noeud, nous utilisons les URL standard du WWW [BLe94]. Pour les graphes, nous avons défini une URL dégénérée formée de trois paramètres :

– le mot réservé `graphe`,

– le nom du domaine d'autorisation,

– le nom unique du graphe dans le domaine.

Dans une URL pour un graphe, le mot réservé `graphe` permet aux clients et aux serveurs de savoir qu'il s'agit d'une capacité pour un graphe. Les deux autres paramètres de cette URL permettent d'identifier uniquement un graphe dans un domaine.

6.2.1.2 Génération de la liste de capacités

Lors de l'appel au script, on lui passe en paramètre la clé publique d'un serveur, le nom d'une méta-liste et l'époque de la liste de capacités. Le script traduira ensuite chaque chaque entrée de la méta-liste en une capacité contenant la clé publique du serveur.

La figure 6.8.b montre la liste de capacités générée à partir de la méta-liste citée précédemment. Afin d'optimiser les recherches dans une liste de capacités, celle-ci inclut un index des capacités contenues dans la liste et leurs dates d'expiration. Après cet index, on trouve les capacités. Celles-ci suivent la structure définie dans la section 5.3.2, sauf pour l'omission du certificat de la clé publique de l'administrateur de sécurité. En effet, dans l'état courant de CAMW3, on supporte un seul administrateur de sécurité. La clé publique de celui-ci est connue par tous les serveurs du système, d'où la décision de l'omettre de la structure des capacités.

6.2.2 Serveur d'autorisation

Le serveur d'autorisation (SA) est responsable de l'autorisation globale des utilisateurs pour la consultation des graphes. L'utilisateur s'adresse à ce serveur pour obtenir une capacité lui permettant de démarrer sa session de consultation. Suite au succès de

BEGIN LIST	
http://groucho.edu/cap-prot/w3paper1-1.html	**URL d'un noeud**
Fri Nov 10 18:16:13 1998	**date d'expiration de la capacité**
/graphe/audomain/paper-icons	**"URL" d'un graphe**
Fri Nov 10 18:17:13 1998	
*END LIST**	

(a) Méta-liste de capacités

BEGIN CAP LIST	**début de la liste de capacités**
005	**époque de la liste de capacités**
*BEGIN CAP INDEX**	**début de l'index de capacités de la liste**
02	**nombre de capacités contenues dans la liste**
http://groucho.edu/cap-prot/text/w3paper1-1.html	
Fri Nov 10 18:16:13 1998	
/graphe/audomain/paper-icons	
Fri Nov 10 18:17:13 1998	
*END CAP INDEX**	
*BEGIN CAP**	**début d'une capacité**
http://groucho.edu/cap-prot/w3paper1-1.html	**URL associée à la capacité**
Fri Nov 10 18:16:13 1998	**date d'expiration de la capacité**
—–BEGIN EXP. SERVER PGP PUBLIC KEY BLOCK—–	**cértificat de clé publique du serveur**
propriétaire de la cap.	
Version: 2.6	
mQBNAzGfh0YAAAECAJZhfMdMPe ...	
—–END EXP. SERVER PGP PUBLIC KEY BLOCK—–	
—–BEGIN SEC-ADMIN SIGNATURE PGP MESSAGE—–	**signature de la capacité**
par l'administrateur de sécurité	
Version: 2.6	
iQBVAwUBMod1JKLdYsv8jyEJAQG ...	
—–END SEC-ADMIN SIGNATURE PGP MESSAGE—–	
*END CAP**	**fin de la capacité**
*BEGIN CAP**	**début d'une capacité**
/graphe/audomain/icons	
Fri Nov 10 18:17:13 1998	
. . .	
. . .	
END CAP	
*END CAP LIST**	**fin de la liste de capacités**

(b) Liste de capacités

Fig 6.8: Génération d'une liste de capacités l'autorisation, le serveur délègue à l'utilisateur une capacité donnant accès à un noeud racine ou à un graphe, d'après l'approche de protection mise en place.

Le serveur est formé des composantes suivantes :

- une *couche d'autorisation*, divisée en un *script d'autorisation global* et un *script de délégation des capacités*,
- une *couche cryptographique*, qui fournit les fonctions de chiffrement et de déchiffrement de messages,
- la *liste de capacités* que le serveur est autorisée à déléguer.

Le script de délégation des capacités est le même que celui utilisé par les serveurs de noeuds. De même, les deux types de serveurs utilisent le même format des listes de capacités. Tous les clients et les serveurs utilisent la même couche cryptographique.

Nous allons maintenant décrire le fonctionnement de la couche cryptographique et des deux scripts de la couche d'autorisation. Le lecteur trouvera une description de la a couche de cryptographique dans la section E.1.

6.2.2.1 Script d'autorisation globale

Ce script est responsable d'autoriser l'accès d'un utilisateur à un graphe. En cas de succès de l'autorisation, le script génère l'autorisateur qui sera associé à la session de consultation de l'utilisateur. Le moteur du serveur appelle ce script après avoir authentifié l'utilisateur.

Autorisation des requêtes

Le script autorise une requête d'après une politique d'autorisation. Comme nous l'avons indiqué précédemment, cette politique dépend du contexte d'utilisation du système. A titre d'expérimentation, nous avons mis en place deux politiques d'autorisation :

Autorisation par authentification et par ACL. Dans cette politique, on suppose que les utilisateurs du système sont connus par le serveur d'autorisation. Le script d'autorisation comprend une ACL dont les entrées spécifient les identifiants des certificats de clé publique des utilisateurs et leurs droits sur les différentes capacités gérées par le serveur. Le serveur authentifie l'utilisateur et autorise sa requête d'après les valeurs contenues dans l'ACL.

Autorisation par paiement. On suppose ici que les utilisateurs doivent payer leur droit de consultation d'un graphe. Un utilisateur prépare un message contenant son numéro de carte banquaire et l'envoie au serveur dans un message signé avec sa clé privée et chiffré avec la clé publique du SA.

En cas de succès de l'autorisation, le script génère un autorisateur. Il stocke ensuite l'autorisateur dans un fichier et copie la capacité demandée vers un autre fichier où le script de délégation peut la trouver. De plus, le script génère un document HTML donnant le temps de vie de l'autorisateur. En cas d'echec, le script génère un document HTML donnant un message d'erreur.

Génération de l'autorisateur

Si le SA autorise la requête de l'utilisateur, il génère un autorisateur. Comme nous l'avons vu précédemment, celui-ci est une capacité qui indique la date d'expiration de la session de l'utilisateur. De plus, l'autorisateur certifie la clé publique utilisée pour authentifier l'utilisateur dans la session.

On décrit l'autorisateur en ASCII. Deux chaînes de caractères entourent les différents champs de l'autorisateur. La figure 6.9 illustre cette structure.

---BEGIN AUTHORIZATOR ---	**début de l'autorisateur**
Mon Feb 24 00:52:25 1998	**date d'expiration de l'autorisateur**
iQBVAwUBMoe ...	**certificat de clé publique de l'utilisateur**
...	**autres champs de l'autorisateur**
---BEGIN AUTH-SERV. SIGNATURE PGP MESSAGE---	**signature de l'autorisateur par le SA**
Version: 2.6	
iQBVAwUBMoe ...	
---END AUTH-SERV SIGNATURE PGP MESSAGE---	
---END AUTHORIZATOR ---	**fin de l'autorisateur**

Fig 6.9: Exemple d'un autorisateur

Le lecteur peut trouver une description plus détaillée des champs de l'autorisateur dans la section 5.3.4, page 99.

6.2.2.2 Script de délégation des capacités

Ce script est responsable de déléguer la liste de capacités (*cap-liste*) associée à un noeud et de la concaténer au message HTTP qui véhicule le noeud. Le moteur du serveur appelle ce script une fois que la réponse HTTP est formée et juste avant de la chiffrer avec la clé publique de l'utilisateur. Le moteur passe en paramètre au script l'autorisateur et le nom de la liste de capacités à déléguer. En plus, on donne au script la valeur de la dernière époque de la liste de capacités que l'on a déléguée au client. Si le SA est en train de déléguer au client une capacité, l'époque vaut zéro. Autrement, il s'agit d'un paramètre inclu dans la requête du client.

La structure d'une liste de capacités déléguées (*Dcap-liste*) ressemble à celle d'une *cap-liste*. Nous retrouvons les capacités déléguées précédées d'une époque et d'un index donnant leurs URL et leurs dates d'expiration (fig. 6.10).

La délégation de la liste de capacités suit trois étapes :

1. comparaison de l'époque de la *cap-liste*,

2. génération de l'index de la *Dcap-liste*,

BEGIN CAP LIST	
005	**époque de la liste de capacités déléguées**
*BEGIN CAP INDEX**	
01	
http://groucho.edu/cap-prot/text/w3paper1-1.html	
Fri Nov 10 18:16:13 1997	
*END CAP INDEX**	
*BEGIN CAP**	
http://groucho.edu/cap-prot/w3paper1-1.html	
Fri Nov 10 18:16:13 1997	
—–BEGIN SERVER PGP PUBLIC KEY BLOCK—–	**certificat de clé publique du SA**
Version: 2.6	
. . .	
. . .	
—–BEGIN SEC-ADMIN SIGNATURE PGP MESSAGE—–	**signature de la capacité par l'AS**
Version: 2.6	
iQBVAwUBMod1JKLdYsv8jyEJAQG . . .	
—–END SEC-ADMIN SIGNATURE PGP MESSAGE—–	
*END CAP**	
*END CAP LIST**	

(a) Liste de capacités avant délégation

BEGIN DCAP LIST	**début de la liste de capacités déléguées**
005	**époque de la liste de capacités déléguées**
*BEGIN DCAP INDEX**	**début de l'index de capacités de la liste**
01	**nombre de capacités contenues dans la liste**
http://groucho.edu/cap-prot/text/w3paper1-1.html	
Mon Feb 24 00:52:25 1997	**date d'éxpiration**
*END DCAP INDEX**	
*BEGIN DCAP**	**début d'une capacité**
http://groucho.edu/cap-prot/w3paper1-1.html	**URL associée à la capacité**
Fri Nov 10 18:16:13 1997	**date d'expiration de la capacité**
—–BEGIN EXP-SERVER PGP PUBLIC KEY BLOCK—–	**certificat de clé publique du SA**
Version: 2.6	
. . .	
. . .	
—–BEGIN SEC-ADMIN SIGNATURE PGP MESSAGE—–	**signature de la capacité par l'AS**
Version: 2.6	
iQBVAwUBMod1JKLdYsv8jyEJAQG . . .	
—–END SEC-ADMIN SIGNATURE PGP MESSAGE—–	
—–BEGIN AUTHORIZATOR —–	**début de l'autorisateur**
Mon Feb 24 00:52:25 1997	**date d'éxpiration de l'autorisateur**
iQBVAwUBMoe . . .	**certificat de clé publique de l'utilisateur**
. . .	**autres champs de l'autorisateur**
—–BEGIN AUTH-SERV. SIGNATURE PGP MESSAGE—–	**signature de l'autorisateur par le SA**
Version: 2.6	
iQBVAwUBMoe . . .	
—–END AUTH-SERV SIGNATURE PGP MESSAGE—–	
—–END AUTHORIZATOR —–	**fin de l'autorisateur**
—–BEGIN EXP-SERVER PGP MESSAGE—–	**signature de la capacité déléguée par le SA**
Version: 2.6	
iQBVAwUBMoeD+09bt7cKW . . .	
—–END EXP-SERVER SIGNATURE PGP MESSAGE—–	
*END DCAP**	
*END DCAP LIST**	**fin de la liste de capacités déléguées**

(b) Liste de capacités après délégation

Fig 6.10: Délégation d'une liste de capacités

3. délégation des capacités.

Si la liste de capacités avait précédemment été déléguée à l'utilisateur, sa requête inclut l'époque de la liste de capacités que lui a été attribuée. Si cette époque est la même que celle courante, le serveur ne déléguera pas la liste.

Si le serveur doit déléguer la liste de capacités, il prépare tout d'abord l'index de la *Dcap-liste*. Comme pour les *cap-listes*, chaque entrée de l'index d'une *Dcap-liste* comprend une URL et une date d'expiration. La valeur de l'URL est la même que celle de l'index de la *cap-liste*. Pour la date d'expiration, nous prenons la date la plus récente parmi les dates d'expiration de la capacité et de l'autorisateur. Ce choix permettra plus tard au client de ne pas envoyer une capacité déléguée ayant un autorisateur valide mais une capacité invalide.

Après avoir généré l'index, le script délègue les capacités qui y sont listées. Une capacité déléguée est composée d'une capacité concatenée à l'autorisateur de l'utilisateur, le tout signé avec la clé privée du serveur délégateur.

Enfin, le script concatène la liste de capacités déléguées (si elle existe) à la réponse HTTP. Notons que l'époque de la liste est transmise à l'intérieur de celle-ci.

6.2.3 Serveur d'objets

Le serveur d'objets est responsable de l'autorisation locale des requêtes des utilisateurs. De plus, le serveur d'objets peut déléguer les listes de capacités associées à ses noeuds.

Le serveur est formé des éléments suivants :

- des règles de protection spécifiant les droits d'accès aux noeuds dans le contexte de chaque graphe.
- une couche d'autorisation, divisée en un *script d'autorisation locale* et un *script de délégation de capacités*,
- une *couche cryptographique*, qui fournit les fonctions de chiffrement/ déchiffrement de messages,
- des fichiers contenant les listes de capacités associés aux noeuds du serveur,
- un fichier contenant une liste de capacités donnant accès aux noeuds qui ont migré vers d'autres serveurs.

Plusieurs de ces éléments sont communs aux autres acteurs de CAMW3. En particulier, les serveurs d'objets et le serveur d'autorisation utilisent le même script de délégation de capacités et le même format des listes de capacités. En outre, la couche cryptographique est la même pour tous les clients et tous les serveurs.

Nous allons maintenant présenter quelques détails sur la spécification des droits d'accès dans le serveur et ensuite décrire les fonctionnalités du script d'autorisation locale. Le lecteur trouvera une description des éléments en commun du serveur dans la section 6.2.2.

6.2.3.1 Spécification de l'autorisation

Nous avons défini une nouvelle règle de protection pour NCSA/WWW pour spécifier que l'accès à un noeud requiert la présentation d'une capacité pour un noeud ou une capacité pour un graphe. Pour l'approche une capacité par noeud, la règle indique que pour accéder au noeud, il faut présenter une capacité donnant le nom du noeud. Pour l'approche une capacité par graphe, la règle liste le nom des graphes qui doivent être inscrits dans une capacité afin de pouvoir accéder à un noeud.

6.2.3.2 Script d'autorisation locale

Ce script est responsable d'autoriser l'accès d'un client à un noeud hébergé dans le serveur. L'autorisation comprend plusieurs opérations :

- vérification de l'authenticité de la capacité,
- vérification de la date d'expiration de la capacité,
- vérification du propriétaire de la capacité,
- comparaison des attributs de la capacité avec l'ACL associée au noeud
- validation du propriétaire de la capacité.

Nous avons décrit toutes ces opérations précédemment dans la section 5.4.2.3. La vérification du propriétaire de la capacité consiste en la comparaison de la clé publique de l'utilisateur avec celle qui est inscrite dans l'autorisateur. Notons que le client inclut cette première clé dans sa requête.

En cas de succès de l'autorisation et si le noeud demandé est associé à une liste de capacités, le script extrait de la requête du client l'époque de la dernière délégation de la liste de capacités et l'autorisateur et les stocke dans un fichier temporaire. Enfin, le script retourne les noms de ce fichier et celui où se trouve la liste de capacités.

En plus des listes de contrôle d'accès associés aux noeuds, le serveur stocke dans un autre fichier des capacités donnant accès aux noeuds qui ont migré vers un autre serveur. Lorsqu'un client demande un tel noeud, le serveur vérifie s'il possède une capacité donnant accès au noeud. Si c'est le cas, il la copie dans un fichier temporaire et retourne le nom de ce fichier.

6.2.4 Client

Dans CAMW3, le client comporte une couche d'autorisation et un cache de capacités déléguées. La couche d'autorisation est formée de deux scripts qui fournissent les services de concatenation et d'extraction de capacités d'un flot de données HTTP. Les scripts utilisent le cache pour stocker les capacités que le client peut récupérer au cours de la session de consultation.

Nous allons maintenant donner quelques précisions sur le cache de capacités et sur les deux scripts.

6.2.4.1 Cache de capacités déléguées

Le cache est un fichier où le client stocke les capacités qui lui sont octroyées. Le cache est formé d'un *index* et d'un ensemble de capacités.

L'index du cache décrit les capacités qui y sont stockées. Chaque entrée de l'index comporte trois éléments :

URL	Date d'expiration	Epoque$_{cap-liste}$

Fig 6.11: Structure du cache de capacités

L'*URL* et la *date d'expiration* sont une copie des paramètres stockés dans la capacité. Ils permettent de repérer rapidement si l'index contient une capacité valable pour accéder à un objet. Le troisième élément, l'*époque*, permet de minimiser le nombre de délégations de capacités (cf. section 5.4.2.3).

Le client met à jour le cache lors de la réception des nouvelles capacités déléguées. Si lors de la présentation de la capacité, un serveur délègue une liste de capacités, il inclura l'époque de cette liste dans sa réponse. Le client mettra alors à jour la valeur de l'époque dans son cache. Si le client reçoit une capacité qui existe déjà dans le cache, il ne remplacera cette dernière que si la nouvelle capacité expire plus tard.

6.2.4.2 Script de concatenation de capacités déléguées

Lors de l'envoi d'une requête pour un noeud, le moteur du client appelle ce script pour chercher une capacité donnant accès au noeud dans le cache. Si la capacité existe et si elle est encore valable, le script joindra celle-ci et son époque à la requête HTTP.

6.2.4.3 Script d'extraction des capacités déléguées

Ce script agit comme un filtre d'une réponse HTTP. Si une liste de capacités déléguées est joint à la réponse, le script enlèvera la liste et mettra à jour le cache du client.

6.3 Evaluation de CAMW3

Un des objectifs de la réalisation du prototype était de pouvoir avoir une première estimation du coût en performance du traitement des capacités dans le système. Pour ce faire, nous avons effectué deux types de mesures :

- *Le temps de traitement des capacités.* Le but de ces mesures est d'évaluer le temps nécessaire pour la validation et pour la délégation des capacités ;
- *Le temps de traitement d'une requête.* L'objectif de ces mesures est d'évaluer le temps que prend un serveur pour répondre à une requête.

Ces mesures ne concernent que les serveurs d'objets. En effet, les opérations faites coté client pour le traitement des capacités n'impliquent aucune opération cryptographique et sont, donc, en essence négligeables.

Pour effectuer les mesures, nous avons utilisé une station Sun/IPC, sous SunOS 4.1.1. et avec une charge de travail réduite. Selon le type de mesure, nous avons utilisé PGP/PEM ou l'extension de CAMW3 à ce protocole pour l'authentification des clients. Afin de réduire le temps nécessaire pour les mesures, nous avons modifié les scripts de la couche cryptographique pour inhiber la signature/vérification des messages HTTP. La couche cryptographique a consisté de deux scripts agissant comme intermédiaires vers PGP 2.6.1. [Zim94a, Zim94b]. PGP utilise notamment l'algorithme RSA pour la signature des messages. Toutes les clés publiques employées avaient une longueur de 512 bits. Chaque mesure a été répétée 1000 fois.

6.3.1 Mesure du temps de traitements des capacités

L'objectif de ces mesures est évaluer le temps nécessaire pour la validation et pour la délégation des capacités dans un serveur de noeuds.

L'approche pour ces mesures a été d'accéder à une page HTML associée à une liste de capacités et d'augmenter le nombre de capacités de cette liste au bout de chaque itération des mesures, allant d'une à dix capacités. Afin de pouvoir calculer une moyenne pour le temps de délégation, toutes les capacités associées à la page étaient identiques. De plus, la capacité déléguée présentée par un utilisateur et la capacité déléguée retournée par le serveur étaient de tailles équivalentes. Les points de mesure ont été établis dans les scripts de la couche d'autorisation, au début et à la fin de chaque opération de traitement des capacités.

Opération	**Temps moyen d'exécution (s)**
Validation d'une capacité déléguée	$3,2$
Délégation d'une capacité	$2,3$

Tab 6.1: Mesures d'efficacité

Le tableau 6.1 présente les mesures effectuées. Le temps de délégation d'une capacité est resté constant, indépendamment du nombre de capacités devant être déléguées. Le tableau n'indique pas la variance, celle-ci étant trop petite.

Du point de vue cryptographique, la différence en temps entre la validation et la délégation d'une capacité semble contraire à ce que l'on pourrait s'attendre. La délégation d'une capacité nécessite la génération d'une signature numérique. En outre, la validation d'une capacité requiert la vérification de deux signatures numériques. Le temps nécessaire pour signer un message est beaucoup plus important que celui nécessaire pour vérifier la signature (cf. section B.4). Donc, la délégation des capacités devrait être plus performante que leur validation.

L'explication pour cette différence dans les temps d'exécution est que le nombre des appels systèmes pour déléguer une capacité est supérieur à celui nécessaire pour la valider. Cette différence est assez grande pour gaspiller le temps gagné lors de la signature de la capacité déléguée.

6.3.2 Mesures du temps de traitement d'une requête

L'objectif de ces mesures est double :

- évaluer le temps nécessaire pour répondre à une requête dans les deux approches **CN** et **CG**
- comparer le temps de réponse de ces approches avec celui de l'approche mise en oeuvre par défaut dans NSCA/WWW.

La technique pour ces mesures a été de protéger un même document avec les approches **CN** et **CG** et avec celle fournit par NCSA/WWW. Comme dans la série de mesures effectuées précédemment, la capacité déléguée présentée par un utilisateur et la capacité délégué retournée par le serveur étaient équivalentes. Le nombre de capacités à déléguer a varié d'un à dix.

Le temps de parcours d'une ACL peut être très court. Afin de pouvoir avoir un seuil de comparaison raisonnable, nous avons opté pour modéliser un cas extrême pour les approches de protection par graphes et par ACL : nous avons utilisé une liste de graphes et une ACL ayant chacune 1000 entrées et plaçant le nom du graphe et de l'utilisateur autorisé dans la dernière entrée de ces listes.

Approche	**Temps moyen de réponse (s)**
ACL (1000 entrées)	$0,72$
CN (avec n délégations)	$3,2{+}(2,3)n$
CG (1000 entrées dans l'ACL)	$3,2+0,48=3,68$

Tab 6.2: Mesures de comparaison

Le tableau 6.2 montre les résultats obtenus. Nous constatons que l'approche **CN** n'est pas compétitive avec les autres deux approches lorsque le serveur doit déléguer une liste de capacités. En effet, le temps de réponse augmente proportionnellement au nombre de délégations. Le temps de délégation est resté constant, indépendamment du nombre de délégations à effectuer.

Le temps de réponse de l'approche **CG** semble, dans un premier temps, être peu efficace. Cependant, environ 87 % du temps dépensé pour autoriser un accès est investi dans la validation de la capacité déléguée. Une fois cette opération réalisée, le temps qui reste est lui moins important que celui employé pour autoriser l'accès dans NCSA/WWW.

En conclusion, les premiers évaluations de performance sont prometteuses : le temps de traitement des capacités est constant. De tels résultats signifient qu'une mise en oeuvre avec un souci d'efficacité permettrai d'atteindre une performance acceptable.

6.4 Bilan de la réalisation

Ce chapitre a présenté CAMW3, un prototype qui met en oeuvre notre modèle d'autorisation sur un système World-Wide Web (WWW). Le but de ce prototype était d'étudier la faisabilité de mise en oeuvre du modèle sur un système existant.

6.4.1 Etat général

Nous avons pris comme point de départ le système NCSA/WWW. L'architecture modulaire de NCSA/WWW s'est avérée très adaptable à l'extension du système. En particulier, nous avons développé des nouveaux modules d'autorisation pour les clients et les serveurs. Ces modules assurent la gestion des capacités : validation et délégation du côté serveur, et stockage et utilisation du côté client.

Le module d'autorisation est composé d'un ensemble de scripts extérieurs aux clients et aux serveurs. Ces derniers contactent les scripts au moyen des API internes à eux. L'utilisation des scripts nous a permis de faire évoluer rapidement le prototype. Cependant, ce choix a nous a obligé à nous servir des fichiers temporaires pour sauvegarder l'état du module lors de la résolution d'une requête. Ceci a pénalisé les performances (augmentation du nombre des appels systèmes) et nous a obligé à dupliquer quelques traitements du serveur dans les scripts (e.g., la prise en compte de l'arrêt d'un appel en cours).

L'échange des capacités a présenté un problème en particulier car le protocole HTTP ne prévoit pas de champ pour l'échange des informations autres que celles d'authentification. Nous avons résolu ce problème en concaténant les capacités au corps des messages HTTP avant de transmettre les messages et en les enlevant avant que les messages ne soient traités.

6.4.2 Etat d'avancement

CAMW3 est actuellement mis en oeuvre sur des ordinateurs Sun/Sparc, sous Solaris 2.5. Les scripts de la couche d'autorisation sont des programmes en langage C et représentent environ 11000 lignes de code. Les scripts de la couche cryptographique sont des scripts shell Unix et représentent moins de 50 lignes de code. Ces derniers scripts font l'interfaçage vers le système PGP [Zim94a, Zim94b]. On utilise PGP pour toutes les opérations de chiffrement/déchiffrement et de signature/vérification des messages.

Le prototype ne met en oeuvre qu'un sous-ensemble des caractéristiques du modèle d'autorisation. En particulier, les extensions permettant d'avoir plusieurs noms d'URL à l'intérieur d'une capacité ne sont pas réalisées. En outre, les listes de capacités sont simples et ne permettent pas encore le regroupement des capacités d'après les graphes où elles sont définies. Ces caractéristiques sont relativement simples à mettre en oeuvre car elles impliquent des améliorations dans le format des capacités et dans leur traitement dans les scripts. Elles ne concernent pas les techniques développées pour l'échange des capacités ou pour l'interfaçage de la couche d'autorisation.

Coté performances, les premières mesures semblent indiquer un temps de traitement constant pour la validation et pour la délégation des capacités. Dans l'état actuel, le temps nécessaire pour effectuer n'importe laquelle de ces deux opérations est encore plus élevée que celui nécessaire par les mécanismes de contrôle d'accès de NCSA/WWW. Néanmoins, nous pensons qu'avec une intégration des différentes scripts à l'intérieur du serveur, nous pourrons ramener ainsi le temps de réponse du serveur à des niveaux plus acceptables.

Troisième partie

Conclusion Générale

Chapitre 7

Bilan et perspectives

7.1 Bilan

Les travaux de recherche qui ont été présentés dans cette étude concernent la conception et l'implémentation d'un modèle de contrôle d'accès non nominatif adapté aux systèmes hypermédia répartis.

Dans la première partie de ce document, nous avons examiné les problèmes de base du contrôle d'accès dans ces systèmes et les principales solutions existantes. Cette démarche nous a permis de mettre en évidence deux limitations importantes présentes dans ces solutions :

- la non prise en compte du partage des noeuds entre graphes lors de l'administration des droits d'accès des utilisateurs,
- le conflit entre l'efficacité de la gestion des utilisateurs et celle de l'autorisation.

Dans la deuxième partie, nous avons proposé un modèle de contrôle d'accès non nominatif qui permet de réduire ces deux limitations. En outre, nous nous sommes fixés, comme objectif secondaire, la prise en compte du principe du moindre privilège. Bien que celui-ci soit un principe de base des systèmes de sécurité, il n'avait pas été traité de manière explicite dans les approches existantes.

Le modèle proposé s'inspire des approches réparties pour la localisation de l'information d'accès dans le système. Ces approches présentent déjà un bon compromis entre les deux types d'efficacité. Nous avons amélioré ce compromis et répondu aux objectifs fixés en utilisant des informations de contrôle d'accès non nominatives inférées à partir de deux propriétés inhérentes aux graphes hypermédia : l'appartenance d'un noeud à un graphe et les liens au départ d'un noeud.

Dans notre modèle, on identifie chaque graphe hypermédia avec un nom unique. On associe ensuite à cette "utilisateur graphe" un ensemble de droits d'accès lui permettant d'accéder à ses composantes. Ces droits sont stockés localement dans les serveurs qui hébergent les composantes.

Un utilisateur acquiert les droits d'accès associés à un graphe en incluant dans sa requête une capacité qui précise le nom du graphe. Au début d'une session de consultation, l'utilisateur récupère une première capacité auprès d'un serveur d'autorisation. Par la suite, l'utilisateur présente cette capacité pour consulter les différents noeuds du graphe.

Nous avons considéré deux approches pour l'assignation des capacités aux utilisateurs : l'approche *une capacité par noeud* (CN) et l'approche *une capacité par graphe* (CG). Dans l'approche **CN**, on associe à chaque noeud d'un graphe, des listes de capacités donnant accès à d'autres noeuds du graphe. L'utilisateur récupère les capacités associées à un noeud lors du parcours de celui-ci. Dans l'approche **CG**, l'utilisateur peut consulter tous les noeuds du graphe avec une seule capacité. Ces deux approches sont indépendantes, mais peuvent être combinées.

Outre une interaction initiale avec le serveur d'autorisation, le modèle n'introduit pas de nouvelle transaction dans une session de consultation. L'échange des capacités s'effectue à l'intérieur des messages échangés dans une transaction normale.

Notre modèle présente la même efficacité d'autorisation que celle des autres approches réparties : un serveur d'objets n'a pas besoin de contacter d'autres serveurs pour autoriser un accès. Les seuls éléments dont le serveur a besoin pour prendre cette décision sont la capacité présentée par le client et les informations de contrôle d'accès associées à l'objet demandé.

L'utilisation des droits d'accès non nominatifs ainsi que celle d'un serveur d'autorisation a permis notamment de simplifier la gestion des clients. Tandis que les serveurs d'objets stockent les droits d'accès non nominatifs, le serveur d'autorisation concentre toute information nominative du système. Toute opération de gestion des utilisateurs ne concerne donc que le serveur d'autorisation. Une exception se présente lors de la révocation explicite des capacités en cours de validité. L'administrateur de sécurité du système doit alors contacter les serveurs du domaine pour leur indiquer quelles sont les capacités invalides.

La spécification des droits d'accès par rapport aux noms des graphes a permis d'atteindre les objectifs de partage de noeuds et de respect du moindre privilège. Le partage de noeuds protégés requiert l'identification du graphe où est valable chaque ensemble de droits d'accès. Dans notre modèle, cette identification est faite explicitement car tous les droits d'accès sont associés aux "utilisateurs graphe." Chaque "utilisateur graphe" possède l'ensemble de droits d'accès nécessaires pour accéder aux composantes du graphe. Si un utilisateur est autorisé à consulter un graphe, il n'obtiendra que les droits d'accès de "l'utilisateur graphe" en question.

En rétrospective, nous estimons que l'approche **CG** est celle qui convient de retenir. Par rapport à l'approche **CN**, l'approche **CG** présente plusieurs avantages :

- elle ne requiert qu'une opération de délégation,

- la mise à jour de l'information de contrôle d'accès associée aux objets est relativement simple,

- elle ne requiert pas la protection des serveurs de liens ni de serveurs de noms,
- un serveur ne dépend pas de la sécurité d'un autre,
- elle permet l'annotation locale[1] des noeuds.

L'approche **CN** présente plusieurs propriétés intéressantes, tels que l'imposition du parcours d'un graphe, l'association d'une liste de capacités à un noeud et sa délégation lors du parcours du noeud. En particulier, il est possible d'utiliser l'approche **CN** pour lier rapidement un sous-graphe à un autre graphe. Il suffit alors d'inclure une capacité pour un noeud du sous graphe dans la liste de capacités d'un noeud. Par comparaison, lors de l'inclusion d'un sous-graphe dans l'approche **CG**, il est nécessaire de donner des droits d'accès à "l'utilisateur graphe" sur tous les noeuds du sous-graphe.

Afin de montrer la faisabilité du modèle, nous avons développé CAMW3, un prototype mettant en oeuvre notre modèle d'autorisation sur un système World-Wide Web. Le prototype a montré qu'il est possible d'intégrer le modèle d'une manière quasi-transparente dans un système hypermédia existant :

- les protocoles d'autorisation peuvent s'imbriquer facilement dans les protocoles de consultation hypermédia,
- on peut contrôler l'accès aux noeuds d'un système hypermédia sans besoin d'inscrire des informations de contrôle d'accès dans les attributs de ceux-ci,
- cependant, le traitement des capacités requiert la modification du client et du serveur.

Les premières mesures de performance indiquent qu'avec une optimisation de la mise en oeuvre du modèle, le traitement des capacités dans le serveur serait effectué dans un délai raisonnable.

7.2 Résumé des contributions

Nous estimons que ce travail de thèse apporte les éléments de réponse suivants aux problèmes d'organisation des droits d'accès des utilisateurs, de localisation de l'information de contrôle d'accès et de gestion des utilisateurs :

L'introduction du concept de graphes dans la structure des listes de contrôle d'accès (ACL). Lorsque deux ou plusieurs graphes incluent un même noeud, il est nécessaire alors de distinguer les droits d'accès des utilisateurs par rapport au graphe où ils sont définis. Notre proposition consiste dans le nommage des graphes et dans l'organisation des droits d'accès autour de ces noms.

1. Le terme en anglais est *bookmarking*.

L'utilisation des propriétés inhérentes aux graphes hypermédia pour un contrôle d'accès non nominatif. Aujourd'hui, les modèles de contrôle d'accès hypermédia associent les droits d'accès à un noeud soit par rapport à l'identité d'un utilisateur ou d'un groupe d'utilisateurs, soit par rapport à une fonction au sein d'une organisation. Notre proposition est d'associer un nom unique à chaque graphe dans le système et d'accorder à ce graphe les droits d'accès sur ces composantes. Il est ainsi possible de prendre en compte explicitement le partage de noeuds protégés et le principe du moindre privilège.

Les critères d'efficacité d'autorisation et d'efficacité de gestion. Nous avons défini deux critères d'efficacité pour pouvoir comparer les différentes approches pour la localisation de l'information de contrôle d'accès dans un système hypermédia. On mesure l'efficacité par le nombre de serveurs qui doit être contacté pour effectuer une opération. Plus petit est ce nombre, plus efficace sera l'opération. Nous distinguons *l'efficacite d'autorisation d'une requête* et *l'efficacité dans la gestion des clients*. Notamment, ces critères nous ont permis de mettre en évidence le conflit présent entre les deux types d'efficacité dans les modèles d'autorisation hypermédia existants.

7.3 Perspectives

Les travaux de recherche présentés dans cette thèse ouvrent plusieurs axes de recherche, notamment en ce qui concerne l'assignation des droits d'accès des utilisateurs et la mise en oeuvre de protocoles d'autorisation. Ces travaux ont été développés d'une façon générale dans la perspective d'évolutions futures. Cette section cite quelques axes de recherche importants qui peuvent être développés à partir de ces travaux.

Evaluation du modèle. Nous pensons que les choix de conception du modèle proposé doivent être évalués dans le cadre des applications réelles. Une application potentielle est la protection d'un intranet où les noeuds du réseau se trouvent dans différents emplacements géographiques. Aujourd'hui, c'est le cas de plusieurs centres de recherche, tels que le CNET et l'INRIA. Une autre application est la protection d'une bibliothèque électronique, où les livres sont représentés par des graphes hypermédia. Un utilisateur pourrait alors acheter un abonnement lui permettant de consulter un ou plusieurs livres.

Extension du modèle à plusieurs domaines d'autorisation. Dans notre modèle, nous avons supposé que les noeuds qui forment un graphe appartiennent tous au même domaine d'autorisation. Une extension intéressante du modèle est de considérer le cas où les noeuds du graphe se trouvent sur des domaines différents. Plusieurs problèmes se présentent alors, tels que la validité de l'autorisation globale et le changement de domaine. Notre modèle apporte quelques premiers éléments de solution à ce problème par l'inclusion dans la structure des capacités des champs identifiant le serveur d'autorisation et l'administrateur de sécurité.

Enrichissement de l'organisation des droits d'accès. Le modèle propose une infrastructure permettant d'associer des droits d'accès aux objets et de mettre en relation un utilisateur avec ces droits au moyen d'une capacité. Dans l'état actuel, les droits d'accès sont organisés autour des noms des graphes. Un utilisateur acquiert les droits d'un graphe lors de la présentation d'une capacité contenant le nom du graphe. Un problème de cette approche est qu'elle est trop rigide par rapport à la spécification des droits d'accès. Si l'on veut définir deux ensembles de droits pour une composante d'un graphe, on est obligé de faire un nouveau graphe pour prendre en compte cette exception. On peut s'affranchir de cette limitation en utilisant un niveau supplémentaire pour l'organisation des droits. Par exemple, on peut associer des catégories aux droits et utiliser l'autorisateur pour indiquer les catégories accordés à l'utilisateur. Cette extension permettra également de mettre en place plus facilement une politique d'accès multi-niveaux.

Amélioration du protocole d'échange de capacités. Dans notre modèle, nous préconisons l'échange des capacités à l'intérieur des messages des protocoles de consultation. En outre, le prototype met en oeuvre cet échange suivant un paradigme de filtre où les capacités sont concaténées aux messages et puis filtrées avant que ceux-ci soient traités. D'une manière plus générale, il est souhaitable d'effectuer cet échange d'une manière plus portable, en utilisant les en-têtes des messages. PEP [CKF97], une nouvelle norme pour l'extension du protocole HTTP, est l'un des choix possibles à considérer pour effectuer cette tâche. PEP permet à deux entités homologues de choisir pendant une transaction l'utilisation d'une extension du protocole HTTP, sans besoin d'accord préalable entre les participants.

Critères de comparaison des modèles de contrôle d'accès. Dans la pratique, le choix d'un modèle de contrôle d'accès dépend fortement de l'environnement d'utilisation du système hypermédia. Dû au manque de critères de comparaison, nous pensons que ce choix est fait d'une manière trop intuitive. Les critères introduits dans cette recherche nous permettent d'effectuer une première comparaison *qualitative* des modèles de contrôle d'accès hypermédia par rapport à la localisation de l'information de contrôle d'accès dans le système. Il est important de compléter ces critères et d'introduire également des critères *quantitatifs* pour pouvoir simplifier le choix d'un modèle d'autorisation.

Outils d'administration adaptés aux graphes. Dans un système hypermédia réparti, on associe aux composantes d'un graphe différentes informations de contrôle d'accès. Un des problèmes que nous avons soulevé est la mise en place d'outils permettant de parcourir un graphe et d'administrer l'information associée à chaque composante. En particulier, l'outil doit garantir l'atomicité des opérations concernant l'ajout ou la révocation des droits d'un utilisateur sur un graphe. Notons que ce problème est indépendant de la localisation des composantes dans le système : dans un système centralisé, les composantes du graphe peuvent être stockées sur différents répertoires. Ce problème est semblable à celui que l'on trouve lors de la replication des droits d'accès dans les systèmes de base de données répartis [SAJ96].

Annexe A

Acronymes et notations

Cette annexe est un guide aux acronymes et à la notation employés dans ce document.

Acronyme	**Signification**
ACI	Information de contrôle d'accès (*Access Control Information*)
ACL	Liste de contrôle d'accès (*Access Control List*)
ACLE	Liste de contrôle d'accès etendue
AS	Administrateur de sécurité
CN	Approche une capacité par noeud
CG	Approche une capacité par graphe
CA	Autorité de certification (*Certification Authority*)
CRL	Liste de certificats révoqués (*Certificate Revocation List*)
DES	Data Encryption Standard
KDC	Centre de distribution de clés (*Key Distribution Center*)
NCSA	National Center for Supercomputing Applications
OSF/DCE	*Distributed Computing Environment.* Un système réparti developpé par l'Open Software Foundation
PAS	Serveur de privilèges (*Privilege Attributes Server*)
PEM	Privacy Enhanced Mail
RBAC	Contrôle d'accès fondé sur des rôles. Role Based Access Control.
SA	Serveur d'autorisation
SESAME	Un système réparti mettant en oeuvre l'architecture de sécurité ECMA (*A Secure European System for Applications in a Multi-vendor Environment*)
TGS	Serveur expéditeur de tickets (*Ticket Granting Server*)
TGT	Ticket expéditeur de tickets (*Ticket Granting Ticket*)
URL	Un identifiant absolu (*Uniform Resource Locator*)
URN	Un identifiant symbolique (*Uniform Resource Name*)
URI	Un raccourci pour se référer aux termes URL et URI (*Uniform Resource Identifier*)
WWW	World-Wide Web

Notation	Signification
$\text{cap}_{objet\ o}$	une capacité donnant accès à l'objet o
cap-liste	Liste de capacités
$\text{Dcap}_{objet\ o}$	une capacité déléguée donnant accès à l'objet o
Dcap-liste	Liste de capacités déléguées
$M1$, $M2$	Concaténation de deux messages.
AR_A	L'adresse réseau du principal A.
UID_A	L'identificateur unique du principal A.
PV	Une période de validité.
$A \rightarrow B : M$	Le principal A envoie le message M au principal B.
$K_{A,B}$	Une clé secrète partagée par les principaux A et B.
KU_A	La clé publique du principal A.
KR_A	La clé privée du principal A.
$\{M\}_K$	Une transformation cryptographique du message M utilisant une clé K.

Annexe B

Algorithmes cryptographiques

Cette annexe présente les principales familles d'algorithmes cryptographiques, leurs propriétés et leurs caractéristiques de performance. La cryptographie est une des couches de base sur laquelle sont construits la plupart des mécanismes d'authentification. [ISO89]. Elle se caractérise par un ensemble de fonctions mathématiques qui permettent de transformer un message (une suite d'octets) en un autre type de message [Den82]. Dans le cadre du contexte du contrôle d'accès, les principales applications de la cryptographie sont :

- l'authentification de l'origine et du contenu de l'information de contrôle d'accès présentée par un principal à un moniteur de référence ;
- l'authentification de l'identité d'un principal ainsi que des messages échangés au cours d'une session.

La section 2.2 décrit plus en détail les techniques d'authentification. [Den82] et [Sch94] donnent une étude plus approfondie des techniques cryptographiques et de leurs applications.

B.1 Algorithmes à chiffrement réversible

Cette famille d'algorithmes comporte deux fonctions de transformation de messages. L'une transforme un message en clair dans un message chiffré et l'autre réalise la transformation inverse. Chacune de ces transformations est effectuée sous le contrôle d'une clé cryptographique. Les équations suivantes représentent ces deux opérations :

$$E(M, \text{clé}_1) = C$$

$$D(C, \text{clé}_2) = M,$$

où E,D, M et C représentent respectivement les algorithmes de chiffrement, de déchiffrement, le message en clair et le message chiffré.

Il existe deux types d'algorithmes à chiffrement réversible : les algorithmes symétriques et les algorithmes asymétriques.

Les *algorithmes symétriques*, ou *algorithmes à clé secrète*, utilisent la même clé cryptographique pour chiffrer et déchiffrer un message. Comme la clé cryptographique n'est connue que par les entités autorisées à chiffrer et à déchiffrer un message, elle est usuellement appelée *clé secrète*. L'algorithme DES [NBS77] est l'exemple le plus représentatif de cette famille d'algorithmes.

Les *algorithmes asymétriques*, ou *algorithmes à clé publique* utilisent une clé différente pour chaque transformation. Ces deux clés vont par paires. Elle sont établies au moyen d'une procédure mathématique. On appelle *clé publique* une de ces clés car elle peut être connue par tous sans compromettre la transformation d'un message ou la confidentialité de l'autre clé. On appelle *clé privée* l'autre clé car elle est (idéalement) associée à une seule personne.

Certains algorithmes asymétriques, tels que RSA [RSA78], ont une propriété associative qui permet d'inverser les deux fonctions de transformation. Cette propriété est surtout utilisée dans les techniques de signature numérique, que nous décrivons dans la section 2.2.3.

On peut résumer les propriétés des algorithmes à chiffrement réversible comme suit [Den82] :

- Dans les algorithmes symétriques, $\text{clé}_1 = \text{clé}_2$.
- Dans les algorithmes asymétriques, $\text{clé}_1 \neq \text{clé}_2$.
- Certains algorithmes ont des fonctions de transformation qui sont associatives :
$$D(E(M, \text{clé}_1), \text{clé}_2) = E(D(M, \text{clé}_2), \text{clé}_1) = M.$$
- Sans la connaissance de la clé de chiffrement, il ne doit pas être possible d'inférer C à partir de M et de E. En d'autres termes, personne d'autre que ceux connaissant la clé de chiffrement ne peut générer C à partir de M (authenticité d'un message chiffré).
- Sans la connaissance de la clé de déchiffrement, il ne doit pas être possible d'inférer M à partir de C et D. C'est à dire, seuls ceux qui connaissent la clé de déchiffrement peuvent générer M à partir de C (confidentialité d'un message chiffré).

B.2 Algorithmes à chiffrement irréversible

Cette famille d'algorithmes à chiffrement irréversible, ou *algorithmes à une seule voie*, se distingue des précédents par le fait qu'ils ne comportent pas de fonction de déchiffrement. Parmi les algorithmes appartenant à cette famille, on trouve les fonctions de hachage.

Une *fonction de hachage* transforme un message de longueur arbitraire en un message de longueur fixe moins importante, appelé *hash*. Typiquement, un hash a une taille inférieure à 512 bits.

La famille d'algorithmes MD [Riv92] est un des exemples les plus répandus des fonctions de hachage.

L'application la plus répandue des fonctions de hachage est la réduction du temps de signature d'un message. Au lieu de signer un message, on signe le hash du message (cf. section 2.2.3). Autres applications sont l'authentification des messages [Tsu92] et l'authentification des principaux [FHBH+96]. Pour pouvoir utiliser un hash comme preuve d'authenticité d'un message, il faut :

- Diffuser le hash du message d'une manière fiable. Un principal vérifie l'authenticité du message en calculant son hash et en le comparant avec la valeur publiée. Le CERT emploie cette méthode pour diffuser ses rapports de sécurité concernant l'Internet [GS96].
- Concaténer une valeur secrète au message avant de calculer le hash [Tsu92]. Ainsi, on peut assurer que seuls les personnes connaissant la valeur secrète ont pu générer le hash.
- Chiffrer le hash avec une clé secrète ou une clé privé [Sch94].

On peut résumer les propriétés des algorithmes à une voie comme suit [Sch94] :

- Il ne doit pas être possible de régénérer un message à partir de son hash (irréversibilité).
- Une fonction de hachage génère toujours la même valeur de hash à des exécutions répétées sur un seul message (déterminisme). Il ne doit pas être possible d'obtenir la même valeur de hash lors de l'application d'une fonction de hachage sur deux messages différents.
- Le moindre des changements sur un message doit engendrer une valeur de hash différente (résistance aux collisions).
- Le hash d'un message peut être utilisé comme preuve d'authenticité du message si le hash est publié de manière fiable, si le hash est chiffré ou si une valeur secrète est concatenée au message avant de calculer le hash.

B.3 Résistance aux attaques

La *résistance aux attaques* d'un algorithme cryptographique est mesurée par l'habilité de l'algorithme à protéger un texte codé contre son déchiffrement non autorisé. La résistance dépend de plusieurs facteurs parmi lesquels nous pouvons citer [GS96] :

- Le secret de la clé de chiffrement.

- La difficulté que représente le fait de trouver une clé ou d'effectuer une recherche exhaustive des clés.
- La complexité que représente l'inversion de l'algorithme de chiffrement si l'intrus ignore la clé de chiffrement.
- La robustesse mathématique des algorithmes.
- La connaissance d'une partie du texte en clair par l'intrus.
- Le temps de vie des clés. Les algorithmes de chiffrement laissent dans un message chiffré quelques traces sur le message en clair. Plus il y a des messages cryptés avec une même clé, plus un éventuel intrus dispose d'échantillons de textes chiffrés. L'intrus peut alors plus facilement effectuer une cryptanalise du texte. Une exception à cette règle est le cas où la longueur d'une clé est identique à celle du message en clair [Den82].

B.4 Performance des algorithmes

Le tableau B.1 donne des exemples du débit de quelques algorithmes représentatifs (source: [LABW92]).[1] Nous pouvons noter que le débit de DES est entre 1000 et 5000 fois plus important que celui de RSA. En général les algorithmes à clé publique sont peu performants car ils effectuent d'importantes opérations de calcul, surtout lors du chiffrement des données. Ils ne sont donc pas appropriés au chiffrement de grandes quantités de données. Une solution consiste à chiffrer les données avec un algorithme à clé symétrique et ensuite à chiffrer la clé secrète, qui est généralement de petite taille, avec l'algorithme à clé publique. La norme PEM[2] pour la protection du courrier électronique sur l'Internet suit cette approche [Ken93].

Algorithme	**Implantation matérielle**	**Implantation logicielle**
Chiffrement RSA	220 Kbits/sec	0,5 Kbits/sec
Déchiffrement RSA	–	32 Kbits/sec
DES	1,2 Gbits/sec	400 Kbits/sec
MD4	–	1300 Kbits/sec

Tab B.1: Débit des opérations cryptographiques

1. Les mesures logicielles ont été effectuées sur un processeur Intel 286 travaillant à 8 MHz et à une performance de 0,5 MIPS et sur un processeur Sparc travaillant à 20 MHz et à 9 MIPS. L'algorithme RSA a été utilisé avec un modulus de 500 bits et un exposant de 3. L'algorithme DES a été utilisé avec une table de 64 KB/clé. Nous renvoyons le lecteur à la référence pour avoir plus de détails.

2. *Privacy Enhanced Mail*

Annexe C

Introduction à la protection dans OSF/DCE

Le projet OSF/DCE[1] a comme objectif la proposition d'un environment complet pour le développement, l'utilisation et la gestion d'applications réparties [RKF92, HU95].

Le principe de base du contrôle d'accès dans OSF/DCE est le suivant [Bai91] : un système réparti est divisé en domaines. Un domaine comprend un serveur d'authentification (AS), un serveur de privilèges (PAS) et un serveur expéditeur de tickets (TGS).[2]

L'AS et le TGS ont des fonctions équivalentes à celles de leurs entités homologues dans Kerberos, que nous avons décrit dans la section 2.2.4.3. Le PAS stocke l'information de contrôle d'accès commune aux serveurs du domaine. La motivation pour ce faire est d'éviter la duplication inutile des informations de contrôle d'accès dans les serveurs du système.

Au début d'une session, un principal doit d'abord s'authentifier auprès d'un AS afin d'obtenir un ticket Kerberos qui prouve son identité, Le principal présente ensuite le jeton au PAS du système pour obtenir une capacité contenant un ensemble de privilèges de sécurité. Une capacité est protégée par des moyens cryptographiques dès qu'elle quitte le PAS pour empêcher son utilisation non autorisée par un intrus.

Lorsqu'un principal veut accéder à un objet, il joint une capacité à sa requête. Un moniteur de référence prend une décision de contrôle d'accès en fonction des privilèges de l'utilisateur et d'autres informations de contrôle d'accès stockées dans une base de données locale.

Dans OSF/DCE, les tickets Kerberos sont utilisés comme capacités. En conséquence, l'ACI d'un utilisateur est échangée par le protocole d'authentification.

Dans la suite de cette section, nous détaillons les étapes qu'un principal doit suivre pour récupérer une capacité et pour la présenter à un serveur d'objets.

1. *Open Software Foundation's Distributed Computing Environment.*
2. Ces termes ont été définis dans les sections 2.1.4 et 2.2.

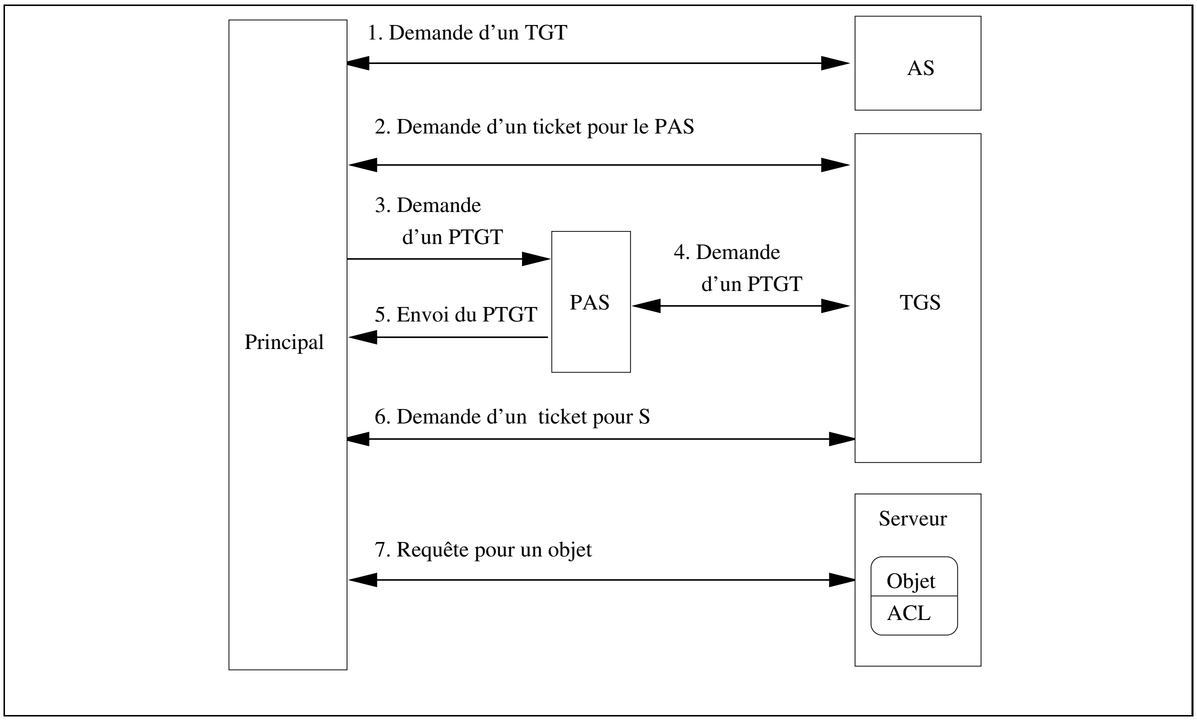


Fig C.1: Aperçu du contrôle d'accès dans OSF/DCE

C.1 Contrôle d'accès dans OSF/DCE

Dans l'architecture de sécurité d'OSF/DCE, tout objet d'un domaine est associé à une ACL. Une entrée d'une ACL contient l'identifiant d'un principal ou d'un groupe de principaux, et leurs droits d'accès sur l'objet. Afin de s'affranchir de longueurs variables des identifiants, OSF/DCE préconise une taille unique de 128 bits. Le PAS est responsable du "mappage" des noms de login des principaux avec leurs identifiants. Le PAS stocke également les listes des groupes de principaux du domaine.

Le processus d'acquisition et d'utilisation d'une capacité est le suivant (fig. C.1) :

1. Un principal obtient un TGT auprès de l'AS. Ce ticket contient le nom de login du principal.

2. Le principal sollicite un ticket pour le PAS auprès du TGS.

3. Le principal demande au PAS un TGT contenant ses privilèges de sécurité : l'identifiant unique du principal et ceux des groupes auxquels il appartient. On appelle ce ticket $PTGT$ car il s'agit d'un TGT qui contient des privilèges de sécurité.

4. Le PAS extrait le nom de login contenu dans le ticket du principal et l'utilise pour chercher les privilèges du principal dans une base de données. Enfin, le PAS envoie ces privilèges au TGS en lui demandant de générer le PTGT.

5. Le PAS renvoie au principal le PTGT et la clé de session associés à ce ticket.

6. Plus tard, lorsque le principal veut effectuer une opération sur un objet stocké dans un serveur S du domaine, il utilise le PTGT pour demander un ticket pour S au TGS. Le TGS authentifie le principal, génère le ticket et y copie les privilèges contenus à l'intérieur du PTGT.

7. Le principal utilise le ticket obtenu dans (6) pour s'authentifier auprès de S. Enfin, S utilise les privilèges contenus dans le ticket pour chercher l'ACL de l'objet et ainsi autoriser ou refuser la requête du principal.

C.2 Résumé

Le contrôle d'accès dans l'architectures OSF/DCE est essentiellement construit autour d'une organisation des serveurs dans des domaines. Un domaine comprend un serveur d'authentification (AS), un serveur expéditeur de tickets (TGS) et un serveur de privilèges (PAS).

L'AS et le TGS ont des fonctions équivalentes à celles de leurs entités homologues dans Kerberos. Le PAS regroupe toute l'information commune aux serveurs d'un domaine, ce qui permet de réduire les problèmes inhérents à la duplication de cette information. En plus de servir à l'authentification des principaux, OSF/DCE utilise les tickets Kerberos comme des capacités Pour récupérer ses privilèges de sécurité, un principal doit s'authentifier auprès du PAS au moyen du protocole Kerberos. Le PAS renvoie alors un nouveau TGT, le PTGT, contenant une copie de ses privilèges. Lors de la présentation du PTGT au TGS, ce dernier dupliquera les privilèges du PTGT dans tout nouveau ticket qu'il delivrera.

Annexe D

Protocole d'authentification PGP/PEM

Cette annexe décrit brièvement ce protocole d'authentification, les modification que nous avons faites et son interface dans le prototype CAMW3. Le lecteur trouvera dans [McC] la spécification originale du protocole.

Les techniques PGP et PEM ont été développées pour la protection du courrier électronique [KPS95, Chaps. 13-14]. Ces techniques permettent d'authentifier l'identité de l'émetteur d'un message et de protéger la confidentialité et l'intégrité du message. Le protocole d'authentification PGP/PEM de NCSA/WWW est une application des messages PGP/PEM à l'authentification mutuelle des clients et des serveurs WWW. Dans la suite, nous donnons un bref résumé du principe de base de PGP/PEM avant d'expliquer son utilisation dans NCSA/WWW.

PGP/PEM est un protocole d'authentification à clé publique. Chaque principal dans le système possède une paire de clés publiques. Le principe de base du protocole d'authentification PGP/PEM de NCSA/WWW est la signature et chiffrement des requêtes et des réponses du protocole HTTP [McC]. Lorsqu'une principal veut envoyer un message à une autre, il signe le message avec sa clé publique et le chiffre ensuite avec la clé publique de son correspondant. Notons que ce protocole n'offre aucune garantie sur la vivacité des échanges. Il est donc proie à des attaques par rejoue des messages.

La mise en oeuvre du protocole d'authentification PGP/PEM dans le système NCSA/WWW suit un modèle de filtres. Les requêtes et les réponses HTTP sont transformées en messages PGP/PEM comme décrit ci-dessus. Afin de permettre aux clients et serveurs d'identifier et de traiter un message PGP/PEM, celui-ci est transmis dans le corps d'un autre message HTTP. Des en-têtes dans ce message identifient le corps comme étant un message PGP/PEM et donnent des indications sur les clés à utilisés pour le créer. Lors de la réception d'un tel message, celui-ci est d'abord filtré par la couche cryptographique. Le message HTTP qui en résulte est alors traité de la manière usuelle.

Dans cette mise en oeuvre, on suppose qu'un serveur connaît *a priori* les clés publiques de ses utilisateurs. De plus, on suppose que le client connaît à l'avance les clés publiques des serveurs où qu'il peut les récupérer au cours de la session au moyen d'un service réseau, tel que `finger`.

Nous présentons maintenant d'une manière abrégée l'échange des messages HTTP nécessaires pour récupérer un document HTML protégé par PGP/PEM (adapté de [Cai96]) :

1. Le client envoie d'abord une requête HTTP standard.
   `GET /chemin/document.html HTTP/1.0`

2. Le serveur renvoie un message d'erreur indiquer au client qu'il doit s'authentifier au moyen d'un message PGP/PEM. Ce message inclut l'identité du serveur pour aider au client à trouver la clé publique du serveur.
   ```
   HTTP/1.0 401 Unauthorized
   WWW-Authenticate: PGP entity=''www-server@ip.address''
   ```

3. Le client prépare le message PGP/PEM. Ce message est ensuite encapsulé dans un autre message HTTP. Une des en-têtes de ce nouveau message inclut l'identité de l'utilisateur pour aider au serveur à trouver la clé publique de celui-ci.
   ```
   GET / HTTP/1.0
   Authorization: PGP entity=''user@ip.address''
   Content-type: application/x-www-pgp-request

   --- BEGIN PGP MESSAGE ---
   Lfelk292k la vraie requête, signée et chiffrée j42kj9j4924+ ...
   --- END PGP MESSAGE ---
   ```

4. Après avoir déchiffré et vérifié la signature du message PGP/PEM, le serveur connaît l'identité de l'émetteur du message. Ensuite, le serveur vérifie si l'utilisateur a le droit d'accéder ou non au document. Enfin, le serveur envoie au client le document ou un message d'erreur dans un autre message PGP/PEM.
   ```
   HTTP/1.0 200 Document follows
   Content-type: application/x-www-pgp-reply
   Content-length: 280

   --- BEGIN PGP MESSAGE ---
   Lfelj42k la vraie réponse, signée et chiffrée j42429j4924+ ...
   --- END PGP MESSAGE ---
   ```

5. Le client déchiffre et vérifie le message PGP/PEM et puis traite la réponse du serveur.

Extensions

Nous avons apporté deux extensions au protocole d'authentification PGP/PEM. La première concerne l'utilisation de ce protocole pour l'échange des capacités. La deuxième extension permet d'utiliser PGP/PEM pour authentifier les clients sans que les serveurs connaissent *a priori* les clés publiques de ceux-ci.

Pour l'échange des capacités, nous profitons du modèle de filtres de la mise en oeuvre de PGP/PEM. Lors de la préparation d'un message PGP/PEM, la couche d'autorisation intercepte le message HTTP. Si une ou plusieurs capacités doivent être échangées, elles sont concaténées au message HTTP. Celui-ci est ensuite transformé en un message PGP/PEM. Lors de la réception d'un message PGP/PEM, la couche d'autorisation intercepte le résultat du traitement PGP/PEM et extrait les capacités.

Dans CAMW3, l'authentification des clients s'effectue également par la signature d'un message PGP/PEM. L'autorisateur, qui est inclus dans toute capacité déléguée, spécifie la clé publique du client. Si la clé privée utilisée pour signer un message correspond à la clé publique inscrite dans l'autorisateur, le serveur considère que la capacité a été effectivement déléguée au client. Notons que le serveur n'a pas besoin de connaître *a priori* les clés publiques de ses clients car celles-ci sont échangées à l'intérieur des capacités.

Pour faciliter au serveur la recherche de la clé publique de l'utilisateur dans les requêtes, et également pour prévoir un éventuel chiffrement de celle-ci, on concatène cette clé au message PGP/PEM. Le message 3 du protocole précèdent prend alors le format suivant :

$3'$ `GET / HTTP/1.0`
`Authorization: PGP entity=''user@ip.address''`
`Content-type: application/x-www-pgp-request`

`--- BEGIN PGP MESSAGE ---`
`oi3=43kl` **la clé publique de l'utilisateur**
`--- END PGP MESSAGE ---`
`--- BEGIN PGP MESSAGE ---`
`Lfelk294092j42k` **la capacité déléguée et la vraie requête, signées et chiffrées**
`j42kj9j429j4924+ ...`
`--- END PGP MESSAGE ---`

Annexe E

Résumé des API de CAMW3

Cette annexe donne à titre indicatif un résumé des principales API que ont été défini pour CAMW3. Nous presentons tout d'abord l'API de la couche cryptographique, qui est utilisée par tous les acteurs de CAMW3. Nous décrivons ensuite les API specifiques à chaque acteur.

E.1 Couche cryptographique

Les clients et les serveurs de CAMW3 s'appuient sur cette couche pour le traitement cryptographique des messages. Cette couche est formé d'une API et de plusieurs scripts qui agissent comme interface vers l'outil cryptographique PGP [Zim94a, Zim94b].

PGP permet de chiffrer, déchiffrer, signer et vérifier la signature d'un message. Il est possible de combiner ces opérations. Par exemple, on peut chiffrer et signer un message dans un seul pas. Le format des messages PGP inclut des indications qui spécifient l'opération associée à un message et qui identifient le principal qui a généré le message.

NCSA/WWW définit une première couche cryptographique permettant de chiffrer et de déchiffrer un message. Chacune de ces fonctions est mise en place par un script.

Dans CAMW3, nous avons introduit un troisième script dans cette couche pour effectuer une opération de signature, sans chiffrement des messages. Il n'a pas été nécessaire de définir un script complémentaire à celui-ci. En effet, PGP utilise une seule fonction pour le déchiffrement et pour la vérification des signatures. Le format des messages indique à PGP quelle opération il faut effectuer. Nous utilisons donc le même script pour le déchiffrement de messages et pour la vérification des signatures.

L'interface de la couche cryptographique est composée des fonctions suivantes :

Chiffrer message {
`entrées` : *texte en clair*, *clé publique destinataire*;
`sorties` : *texte chiffré*;
`code de retour` : *néant*;
};

Déchiffrer/vérifier signature message {
`entrées`: *texte signé/chiffré, clé publique destinataire*;
`sorties`: *texte en clair*;
`code de rétour`: *Succés ou Erreur*;
};

Signer message {
`entrées`: *texte en clair*;
`sorties`: *texte signé*;
`code de retour`: *néant*;
};

Remarque. Les scripts ont des informations codées en dur qui leur permettent d'accéder aux clés publiques et privées du principal qui exécute les scripts.

E.2 Générateur des capacités

L'interface du générateur de capacités est composée de la fonction suivante :

Générer liste de capacités {
`entrées`: *Méta-liste de capacités,*
Certificat de clé publique du serveur destination,
Epoque de la liste de capacités;
`sorties`: *liste de capacités*;
`code de retour`: *Néant*;
`Commentaires`: L'administrateur de sécurité a une base de données
contenant les clés publiques de tous les serveurs de
son domaine.
};

E.3 Serveur d'autorisation

L'interface de la couche d'autorisation de ce module est composée des fonction suivantes :

Autoriser requête pour une capacité déléguée {
`entrées`: *requête HTTP*,
clé publique du client;
`sorties`: *réponse HTTP*,
nom du fichier contenant la liste de capacités à déléguer,
nom du fichier contenant l'autorisateur;
`code de retour`: *Néant*;
`Commentaires`: Si l'accès n'a pas été autorisé ou
s'il y a eu une erreur dans la requête de
l'utilisateur, aucun fichier contenant des
capacités ne sera créé.
};

Déléguer liste de capacités {
`entrées`: *réponse HTTP*,
nom de la liste des capacités à déléguer,
époque de cette liste de capacités datant de sa
dernière délégation au client,
l'autorisateur,
clé publique du client;
`sorties`: *liste de capacités concatenée à la réponse HTTP*;
`code de retour`: *Néant*;
`Commentaires`: Si le fichier contenant les capacités
n'existe pas ou s'il n'existe pas d'autorisateur
la réponse HTTP ne sera pas modifiée.
};

E.4 Server de noeuds

L'interface de la couche d'autorisation de ce module est composée de la fonction **Déléguer liste de capacités**, que nous avons décrit dans la section E.3 et de la fonction suivante :

Autoriser requête pour un noeud {
`entrées`:*capacité déléguée*,
$époque_{Dcap-liste}$,
requête HTTP,
certificat de clé publique du client;
`sorties`:*requête HTTP*;
nom du fichier contenant l'autorisateur et
l'époque;
`code de retour`: *Autorisation ou Erreur*;
};

E.5 Client

Les interfaces des deux scripts de la couche d'autorisation du client sont les suivantes :

Concatener une capacité déléguée {
`entrées` : *requête HTTP*,
URL,
domaine d'autorisation;
`sorties` : *requête HTTP concatenée à une capacité déléguée*,
$Epoque_{cap-liste}$;
`code de retour` : *Néant*;
`Commentaires` : Si l'on inclut une capacité et si c'est la première fois qu'on demande cette URL, l'époque aura une valeur nulle.
};

Extraire liste de capacités déléguées {
`entrées` : *réponse HTTP*,
liste de capacités déléguées;
`sorties` : *réponse HTTP*;
`code de retour` : *Néant*;
`Commentaires` : Si l'on trouve une liste de capacités déléguées, le script mettra à jour le cache de capacités du client.
};

Bibliographie

[ABKL90] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. Technical Report 67, DIGITAL, Systems Research Center, October 1990.

[ABLP92] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. In *Proc. Advances in Cryptology – Crypto '91*, LNCS 576, pages 1–23. Springer, 1992.

[Adi93] C. Adie. Network access to multimedia information. Technical Report RARE Project OBR(93)015, Réseaux Associés pour la Recherche Européene, August 1993. 2nd. Edition.

[AG95] S. Anderson and R. Garvin. Sessionneer: Flexible session level authentication with off the shelf servers and clients. In *Third World-Wide Web Conference*, pages 1047–1053, April 1995.

[AK94] K. Andrews and F. Kappe. Soaring through hyperspace, a snapshot of Hyper-G and its Harmony client. In *Eurographics Symposium on Multimedia/Hypermedia in Open Distributed Environments*, pages 181–191, June 1994.

[Akl83] S. G. Akl. Digital signatures: A tutorial survey. *Computer*, 16(7), February 1983.

[AKM95] K. Andrews, F. Kappe, and H. Maurer. Serving information on the web with Hyper-G. In *Third World-Wide Web Conference*, pages 919–926, April 1995.

[And73] J.P. Anderson. Information security in a multi-user computer environment. In *Advances in Computers*. Academic Press, 1973.

[Bai91] E. Baize. Mise en oeuvre des normes de sécurité ECMA dans les systèmes distribués. In *De nouvelles architectures pour les communications*, pages 82–94, 1991.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Research Report 39, DIGITAL Systems Research Center, February 1989. Revised February 22, 1990.

[BCF+97] J. F. Barkley, A. V. Cincotta, D. F. Ferraiolo, S. Gavrilla, and D. R. Kuhn. Role Based Access Control for the World Wide Web. Technical report, NIST, April 1997. http://hissa.ncsl.nist.gov/rbac/rbacweb/paper.ps.

[BD91] E. Berk and J. Devlin, editors. *Hypertext/Hypermedia Handbook*, chapter 20, pages 299–325. Intertext Publications, McGraw-Hill Pub. Co, Inc., 1991.

[BD92] F. Bauspiess and F. Damm. Requirements for cryptographic hash functions. *Computers and Security*, 11(5):427–437, 1992.

[BGS92] J. A. Bull, L. Gong, and K. R. Sollins. Towards security in an open systems federation. In *Proc. European Symposium on Research in Computer Security (ESORICS)*, pages 3–20, 1992.

[BL94] T. Berners-Lee. Universal Resource Identifiers in WWW: A unifying syntax for the expression of names and addresses of objects on the network as used in the World-Wide Web. Internet RFC 1630, June 1994.

[BL96] T. Berners-Lee. WWW: Past, Present, and Future. *Computer*, 37(8):69–77, October 1996.

[BLCL+94] T. Berners-Lee, R Cailliau, A. Luotonen, H. Frystik Nielsen, and A. Secret. The World-Wide Web. *Comm. of the ACM*, 37(8):76–82, August 1994.

[BLe94] T. Berners-Lee and L. Masinter (ed.). Uniform Resource Locators (URL). Internet RFC 1738, December 1994.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. Internet RFC 1945, May 1996.

[BLPS96] J-P. Balpe, A. Lelu, F. Papy, and I. Saleh. *Techniques avancées pour l'hypertexte*. Hermès, Paris, 1996.

[Boo92] J. Boone. A survey of data models for hypermedia. Technical report, Department of Computer Science, University of North Carolina, April 1992.

[Cai96] A. Cain. Web security – Technologies for security, authentication, and privacy on the World-Wide Web. In *Fifth World-Wide Web Conference - Tutorial Notes*, pages 1–31. May 1996.

[Car94] U. Carlsen. *Formal Specification and Analysis of Cryptographic Protocols*. PhD thesis, Université Paris XI - Orsay, October 1994.

[CCI87] CCITT. Recommendation X.509 and ISO 9594-8, Information Processing Systems - Open Systems Interconnection - The Directory - Authentication Framework. CCITT Blue Book, CCITT, November 1987.

[CKF97] D. Connolly, R. Khare, and H. Frystyk. PEP - an extension mechanism for HTTP. Working Draft draft-ietf-http-pep-04, W3C-IETF, July 1997. http://www.w3.org/TR/WD-http-pep.

[Con87] J. Conklin. Hypertext: An introduction and survey. *Computer*, 20(9):17–41, September 1987.

[Dav95] H. Davis. To Embed or Not to Embed? *Comm. of the ACM*, 38(8):48–60, August 1995.

[Den82] D. E. Denning. *Cryptography and Data Security.* Addison-Wesley Publishing Company, 1982.

[DeR89] S. J. DeRose. Expanding the notion of links. In *Proc. Hypertext 89*, pages 249–257, 1989.

[Des91] Y. Deswarte. Tolérance aux fautes, sécurité et protection. In R. Balter, J.-P. Banâtre, and S. Krakowiak, editors, *Construction des Systèmes d'Exploitation Répartis*, Collection Didactique. INRIA, 1991.

[DS81] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Comm. ACM*, 24(8):533–536, August 1981.

[ECM88] ECMA. Security In Open Systems : A Security Framework. Technical Report ECMA-TR/46, ECMA, 1988.

[ECM89] ECMA. Security In Open Systems : Data Elements and Service Definitions. Standard ECMA-138, ECMA, December 1989.

[ECM96] ECMA. Authentication and Privilege Attribute Security Application with related key distribution functions. Standard ECMA-219, ECMA, March 1996.

[FCK95] D. F. Ferraiolo, J. A. Cugini, and D. R. Kuhn. Role-Based Access Control (RBAC): Features and Motivations. In IEEE Computer Society Press, editor, *Annual Computer Security Applications Conference*, 1995. http://hissa.ncsl.nist.gov/rbac/newpaper/rbac.ps.

[FGM+96] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Internet Draft draft-ietf-http-v11-spec-07, August 1996.

[FHBH+96] J. Franks, P. Hallam-Baker, J. L. Hostetler, P. Leach, A. Luotonen, E. W. Sink, and L. W. Stewart. An Extension to HTTP : Digest Access Authentication. Internet Draft draft-ietf-http-digest-aa-05.txt, September 1996.

[FKK96] A. O. Freier, P. Karlton, and P. C. Kocher. The Secure Sockets Layer (SSL) protocol Version 3.0. Internet Draft draft-freier-ssl-version3-01.tx, March 1996.

[FNS91] C. Fritzner, L. Nilsen, and A. Skomedal. Protecting security information in distributed systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 245–254, 1991.

[Gas88] M. Gasser. *Building a secure computer system.* Van Hoostrand and Reinhold, 1988.

[GBD93] D. Gollman, T. Beth, and F. Damm. Authentication services in distributed systems. *Computers and Security*, 12(8):753–764, August 1993.

[GD72] G. S. Grahman and P. J. Denning. Protection– principles and practice. In *Proc. Spring Jt. Computer Conference*, volume 40, pages 417–429. AFIPS Press, Montvale, N. J., 1972.

[GNE+97] N. Grady, N. Nachtigal, M. Elmore, J. Kohl, J. Rome, J. Reed, and P. Papadapoulos. On-the-Fly Translation and Role-Based Authorization For Intranet Document Systems. Poster presented at the WWW7 conference, April 1997.

[Gon89a] L. Gong. On security in capability-based systems. *ACM Operating Systems Review*, 23(2):56–60, April 1989.

[Gon89b] L. Gong. A secure identity-based capability system. In *IEEE Symposium on Security and Privacy*, pages 56–63, 1989.

[GS96] S. Garfinkel and G. Spafford. *Practical Unix and Internet Security.* O'Reilly and Associates, Inc., 2ème edition, 1996.

[HA94] N. Haller and R. Atkinson. Internet authentication guidelines. Internet Draft draft-haller-auth-requirements-03.txt, ISOC, February 1994.

[HBvR92] L. Hardman, D. C. A. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model: Extending hypertext to support real multimedia. Technical Report RARE Project OBR(93)015, CWI, July 1992. 2nd. Edition.

[HKR+92] B. J. Hann, P. Kahn, V. A. Riley, J. H. Coombs, and N. K. Meyrowitz. IRIS Hypermedia Services. *Comm. of the ACM*, 35(1):36–51, January 1992.

[HRU76] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Comm. ACM*, 19(8):461–471, August 1976.

[HS94] F. Halasz and M. Schwartz. The Dexter hypertext reference model. *Comm. of the ACM*, 37(2):30–39, February 1994.

[HU95] W. HU. *DCE Security Programming.* O'Reilly and Associates, Inc., 1995.

[ICL96] D. Ingham, S. Caughey, and M. Little. Fixing the "Broken-Link" Problem: the W3Objects Approach". In *Fifth World-Wide Web Conference*, pages 1255–1268, May 1996.

[Int94] Interactive Multimedia Assocation. *IMA Intellectual Property Project Proceedings*, volume 1, January 1994.

[ISO89] ISO. Information Processing Systems – OSI Reference Model – Part 2: Security Architecture, International Standards Organization. Standard ISO 7498-2, ISO, 1989.

[ISO92a] ISO. Information Technology - Open Systems Interconnection - Security Frameworks in Open Systems - Part 3: Access Control. Committee Draft ISO/IEC JTC1/SC 21 N6947, International Standards Organization, October 1992.

[ISO92b] ISO/IEC. Information Technology - Hypermedia/Time-based Structuring Language (HyTime). Draft International Standard ISO/IEC JTC1/SC18/WG8 DIS 10744, ISO/IEC, 1992.

[ISO94] ISO/IEC. Information Technology - Coding of Multimedia and Hypermedia Information - Part 1: MHEG Object Representation, Base Notation (ASN.1). Draft International Standard ISO/IEC DIS 13522-1, ISO/IEC, October 1994.

[JM91] P. Janson and R. Molva. Security in open networks and distributed systems. Elsevier Science Publishers, 1991.

[Kah94] J. Kahan. Un nouveau modèle d'autorisation pour les systèmes de consultation d'information multimédia répartie. In *Où, quand, comment protéger vos logiciels et documents électroniques.* AFCET, December 1994. In French.

[Kah95a] J. Kahan. A capability-based authorization model for the World-Wide Web. In *Third World-Wide Web Conference*, pages 1055–1064, April 1995.

[Kah95b] J. Kahan. A distributed authorization model for WWW. In *INET'95*, June 1995. http://www.isoc.org/HMP/PAPER/107/.

[Kap91] F. M. Kappe. *Aspects of a Modern Multi-Media Information System.* PhD thesis, TU Graz, June 1991.

[Kar89] P.A. Karger. New methods for immediate revocation. In *IEEE Symposium on Security and Privacy*, 1989.

[Ken93] S. T. Kent. Internet Privacy Enhanced Mail. *Comm. of the ACM*, 36:48–60, August 1993.

[KN93] J. Kohl and B. C. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, ISOC, September 1993.

[Koe92] J. F. Koegel. On the design of multimedia interchange formats. In *Network and Operating System Support for Digital Audio and Video*, pages 262–271, November 1992.

[KP93] F. Kappe and G. Pani. Hyper-G Client/Server Protocol - Version 5. Technical report, IICM,, Graz University of Technology, A-8010 Graz, Austria, December 1993. ftp://iicm.tu-graz.ac.at/pub/Hyper-G/doc/protocol.ps.

[KPS93] F. Kappe, G. Panni, and F. Schnabel. The Architecture of a Massively Distributed Hypermedia System. *Internet Research: Electronic Networking Applications and Policy*, 3(1):10–24, 1993.

[KPS95] C. Kaufman, R. Perlman, and M. Speciner. *Network Security - Private Communication in a Public World.* Prentice Hall, 1995.

[Kra91] S. Krakowiak. Principes et mécanismes de base de la répartition. In R. Balter, J.-P. Banâtre, and S. Krakowiak, editors, *Construction des Systèmes d'Exploitation Répartis*, Collection Didactique. INRIA, 1991.

[LABW92] B.W. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Operating Systems Review*, 10(4):265–310, November 1992.

[Lam74] B.W. Lampson. Protection. *Comm. ACM*, 8(1):18–24, January 1974.

[Lan81] C.E. Landwerh. Formal models for computer security. *Computer Surveys*, 13:247–275, 1981.

[LB92] K.-Y. Lam and T. Beth. Timely authentication in distributed systems. In *European Symposium on Research in Computer Security (ESORICS)*, pages 293–306, 1992.

[LC94] M. R. Low and B. Christianson. Technique for authentication, access control and resource management in open distributed systems. *Electronics Letters*, 30(2):124–125, January 1994.

[Lev84] H. M. Levy. *Capability-based Computer Systems*. Digital Press, Bedford, Massachusetts, 1984.

[Lew95] S. Lewontin. The DCE Web Toolkit: Enhancing WWW Protocols with Lower-Layer Services. In *Third World-Wide Web Conference*, pages 765–771, April 1995.

[LG92] K.-Y. Lam and D. Gollmann. Freshness assurance of authentication protocols. In *Proc. European Symposium on Research in Computer Security (ESORICS)*, pages 261–272, 1992.

[Lie93] A. Liebl. Authentication in distributed systems: A bibliography. *ACM Operating Systems Review*, 27(4):31–41, October 1993.

[LK94] M. G. Lavenant and J. A. Kruper. The Phoenix project: Distributed hypermedia authoring. In *First World-Wide Web Conference*, May 1994. http://www.cern.ch/PapersWWW94/j-kruper.ps.

[LS96] D. La Liberte and M. Shapiro. The Path URN Specification. Internet Draft, 15 Sep. 1995 (Expires 17 Jan. 1996). http://union.ncsa.uiuc.edu/~liberte/www/path.html.

[LZ94] S. Lewontin and M. E. Zurko. The DCE Web Project: Providing Authorization and other Distributed Services to the World-Wide Web. In *Second World-Wide Web Conference*, October 1994. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Security.

[Mau96] H. Maurer. *Hyper-Wave: The Next Generation Web Solution*. Addison-Wesley, 1996.

[McC] R. McCool. Using PGP/PEM Encryption. Technical report, NCSA. http://hoohoo.ncsa.uiuc.edu/docs/PEMPGP.html.

[MNSS87] S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos authentication and authorization system. Technical report, MIT Proj. ATHENA, Technical Plan, Section E.2.1, December 1987.

[NBS77] National Bureau of Standards. Data Encryption Standard. FIPS PUB 46, NBS, 1977.

[Neu92] B. C. Neuman. *The Virtual System Model: A Scalable Approach to Organizing Large Systems*. PhD thesis, University of Washington, 1992.

[Neu93] B. C. Neumann. Proxy-based authorization and accounting for distributed systems. In *Proc. 13th International Conference on Distributed Computing Systems*, pages 283–291, May 1993.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Comm. ACM*, 21(12):993–999, 1978.

[NS91] J. Noll and W. Scacchi. Integrating Diverse Information Repositories: A Distributed Hypertext Approach. *Computer*, 20(12):38–45, December 1991.

[NT94] C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[Pea89] A. Pearl. Sun's Link Service: A Protocol for Open Linking. In *Proc. Hypertext 89*, pages 137–146, 1989.

[PJ96] J. E. Pitkow and R. K. Jones. Supporting the Web: A distributed hyperlink database system. In *Fifth World-Wide Web Conference*, pages 981–991, May 1996.

[Pla91] C. Plaisant. An Overview of Hyperties, its user interface and data model. In *Hypermedia/Hypertext and Object-oriented Databases*, pages 17–31. 1991.

[PP95] T. Parker and D. Pinkas. SESAME V4 - Overview. Technical report, SESAME, December 1995. http://www.esat.kuleuven.ac.be/cosic/sesame.html.

[QV93] V. Quint and I. Vatton. Combining Hypertext and Structured Documents in Grif. In *Proc. Hypertext 93*, pages 23–32, 1993.

[Rag97] D. Ragget. HTML 3.2 Reference Specification. W3C Recommendation 1866, World-Wide Web Consortium, January 1997. http://www.w3.org/TR/REC-html32.html.

[Riv92] R.L. Rivest. The MD5 Message Digest Algorithm. RFC 1321, ISOC, April 1992.

[RKF92] W. Rosenberry, D. Kenney, and G. Fisher. *Understanding DCE*. O'Reilly and Associates, Inc., 1992.

[RMV94] P. Rolin, L. Me, and J. Vazquez. Sécurité des systèmes informatiques. *Réseaux et Informatique Repartie*, 4(1), 1994.

[RN95] K. E. Rowe and C. K. Nicholas. Reliability of www name servers. In *Third World-Wide Web Conference*, pages 773–783, April 1995.

[RS96a] E. Rescorla and A. Schiffman. Security Extensions For HTML. Internet Draft draft-ietf-wts-shtml-02.txt, July 1996.

[RS96b] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. Internet Draft draft-ietf-wts-shttp-02.txt, May 1996.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, February 1978.

[SAJ96] P. A. Samarti, P. Amman, and S. Jajodia. Maintaing replicated authorizations in distributed database systems. *Data & Knowledge Engineering*, 18:55–84, 1996.

[SCFY96] R.S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *Computer*, pages 38–47, February 1996.

[Sch94] B. Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, Inc, 1994.

[Sha77] K. S. Shankar. The total computer security problem. *Computer*, 10(6):50–73, 1977.

[SM94] K. Sollins and L. Masinter. Functional Requirements for Uniform Resource Names. Internet RFC, December 1994. 1737.

[SNS88] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. USENIX Winter Conference*, pages 191–202, February 1988.

[SS75] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[SS94] R.S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, pages 40–48, September 1994.

[SSS93] D. E. Shackelford, J. B. Smith, and F. Donelson Smith. The Architecture and Implementation of a Distributed Hypermedia Storage System. In *Hypertext '93*, pages 1–13, November 1993.

[Sta95] W. Stallings. *Network and Internetwork Security - Principles and Practice.* Prentice Hall, 1995.

[Ste95] L. D. Stein. *How to Set Up and Maintain a World Wide Web Site*, pages 55–144. Addison-Wesley, 1995.

[SWJF96] K. Shafer, S. Weibel, E. Jul, and J. Fausey. Inroduction to persistent uniform resource locators. Technical report, Online Computing Library Center, Inc., 1996. http://purl.oclc.org/OCLC/PURL/INET96.

[Tan92] A. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 1992.

[TKM+91] I. Tomek, S. Khan, T. Müldner, M. Nassar, G. Novak, and P. Proswynski. Hypermedia – introduction and survey. *Journal of Microcomputer Applications*, 14(2):63–103, 1991.

[TMvR86] A. S. Tanenbaum, S. J. Mullender, and R. van Renesse. Using sparse capabilities in a distributed operating system. In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 558–563, May 1986.

[Tra] Transarc. *AFS User's Guide.*

[Tsu92] G. Tsudik. Message authentication with one-way hash functions. In *INFOCOM 92*, pages 8D.2.1–8D.D2.5, 1992.

[VAB91] V. Varadharajan, P. Allen, and S. Black. An analysis of the proxy problem in distributed systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 255–275, 1991.

[VK85] V.L. Voydock and S.T. Kent. Security in high level network protocols. *IEEE Comm.*, pages 12–24, July 1985.

[VSW94] R. J. Vetter, C. Spell, and C. Ward. Mosaic and the World-Wide Web. *Computer*, 23(12):49–57, October 1994.

[WL92] T. Woo and S. Lam. Authentication for distributed systems. *Computer*, pages 39–52, January 1992.

[Won89] R. M. Wong. Issues in secure distributed operating system design. In *Proc. IEEE Compcon*, pages 338–341, 1989.

[YM96] N. Yeager and R. E. McGrath. *Web Server Technology.* Morgan Kaufman, 1996.

[ZHB95] M. E. Zurko and P. M. Hallam-Baker. Secure issues on the web. In *Third World-Wide Web Conference - Tutorial Notes*, pages 143–186. April 1995.

[Zim94a] P. Zimmerman. PGP User's Guide - Volume I: Essential Topics, May 1994. Version 2.6.

[Zim94b] P. Zimmerman. PGP User's Guide - Volume II: Special Topics, May 1994. Version 2.6.

VU : **VU :**

Le Directeur de Thèse **Le Responsable de l'École Doctorale**

Kadi BOUATOUCH **Jean-Pierre BANÂTRE**

VU pour autorisation de soutenance

Rennes, le

Le Président de l'Université de Rennes 1

Jacques LENFANT

VU après soutenance pour autorisation de publication :

Le Président de Jury,

Conception and experimentation of a non nominative access-control model for distributed hypermedia systems

Abstract

Distributed hypermedia systems require specification and evaluation of users' access rights to the information available in the system. The structuring of information into graphs as well as the distribution of a graph's nodes over the network present new access control problems. In particular, this research work concerns the following two problems: user administration and sharing of a protected node among different graphs.

An access control model is proposed that combines the association of a name with each graph and a distribution of the system's access control information. Instead of granting each user rights over each node, we give each graph a unique name. We then use this name to specify the access rights over the nodes of the graph. These rights, which are not user nominative, are stored locally on each node server. An authorization server grants a user a global right to consult a graph by giving him a capability which contains the name of the graph. The presentation of this capability allows the user to acquire the set of access rights associated with the graph.

We consider two approaches for using capabilities. In the first approach, the user must present a different capability to access each node. Moreover, each node is associated with a capability list according to the outgoing links from the node. A user acquires this list when browsing the node. In the second approach, the user only needs one capability to access all the nodes belonging to a graph.

The access control model that is proposed conforms to the least privilege principle, while taking into account the sharing of nodes between graphs and simplifying user administration.

Finally, an implementation of the model on a World-Wide Web system is described.

Keywords: distributed hypermedia systems, access control, authorization, security, protection, capabilities, World-Wide Web, Internet.

Conception et expérimentation d'un modèle de contrôle d'accès non nominatif pour les systèmes hypermédia répartis

Résumé

Le contrôle d'accès dans un système hypermédia réparti implique la déclaration et l'évaluation des droits d'accès des utilisateurs sur l'information disponible dans le système. La structuration en graphes de l'information et la répartition des noeuds des graphes sur plusieurs serveurs posent de nouveaux problèmes de contrôle d'accès. En particulier, nous nous intéressons aux problèmes de gestion des utilisateurs et d'inclusion d'un noeud protégé dans plusieurs graphes.

Le schéma de protection adopté dans cette thèse consiste à combiner un nommage des graphes avec une localisation répartie de l'information de contrôle d'accès du système. Au lieu d'associer à chaque noeud des paires identifiant un utilisateur ou un groupe d'utilisateurs et ses droits d'accès sur le noeud, nous assignons un nom unique aux graphes et utilisons ce nom pour spécifier les droits. Ces droits, dits non nominatifs, sont stockés localement dans chaque serveur de noeuds. Un serveur d'autorisation est responsable d'autoriser globalement l'accès d'un utilisateur à un graphe et de lui fournir une capacité contenant le nom du graphe. La présentation de cette capacité permettra à l'utilisateur d'acquérir les droits d'accès attribués au graphe.

Nous considérons deux approches pour l'utilisation des capacités. Dans la première approche, un utilisateur doit présenter une capacité différente pour accéder à chaque noeud. De plus, chaque noeud est associé avec une liste de capacités en fonction des liens au départ du noeud. Un utilisateur obtient cette liste lors du parcours du noeud. Dans la deuxième approche, l'utilisateur n'a besoin que d'une seule capacité pour accéder à tous les noeuds du graphe.

Le modèle de contrôle d'accès qui est proposé permet de respecter au mieux le principe du moindre privilège, prend en compte le partage des noeuds entre graphes et présente une administration simplifiée des utilisateurs.

Enfin, une mise en oeuvre du modèle sur le système World-Wide Web est détaillée.

Mots-clés : systèmes hypermédia répartis, contrôle d'accès, autorisation, sécurité, protection, capacités, World-Wide Web, Internet.