La désignation dans les systèmes d'exploitation répartis

José LEGATHEAUX MARTINS et Yolande BERBERS

PRÉSENTATION *Tout système informatique est le siège de relations établies entre ses composants pour pouvoir gérer la coopération entre les activités concurrentes, l'allocation dynamique des ressources communes ou la conservation et la structuration de l'information. Tous ces objets qui interagissent doivent être repérés par le système ou par l'utilisateur pour être individualisés, énumérés, regroupés et manipulés à tous les niveaux que comporte le système. Cela se fait à l'aide de* noms *fournis par les mécanismes de* désignation. *Dans un* système réparti, *la désignation, l'accès et le partage des objets ne doivent pas dépendre de la localisation des objets car il faut pouvoir les déplacer en leur laissant le même nom (car il s'agit du même objet) ; cette attribution définitive d'un nom à un objet doit résister aux pannes, aux redémarrages et aux reconfigurations du système ; enfin l'existence de copies multiples d'un même objet, présent sur plusieurs sites, a conduit à introduire des noms de groupe.*

Les auteurs de cet article se livrent à un examen approfondi des problèmes soulevés par la désignation dans les systèmes répartis. Leur analyse s'appuie sur une hiérarchie de mécanismes de désignation. Les solutions sont illustrées par des exemples représentatifs issus de systèmes répartis construits ces dernières années.

L'expérience des auteurs, acquise en participant à la réalisation des systèmes Chorus et Hermix leur a permis d'aller à l'essentiel et de bien dégager les avantages et les inconvénients pratiques de chaque mécanisme.

Claude Kaiser

Introduction

SYSTÈMES D'EXPLOITATION RÉPARTIS

Un environnement réparti est une collection de processeurs munis de mémoire, reliés par un réseau de communication et coordonnés par des composants logiciels. Avec de tels environnements, on espère améliorer certaines propriétés d'un système informatique comme :

- l'extensibilité, la souplesse et la modularité car ils autorisent une extension souple et progressive du matériel et du logiciel en fonction des besoins ;
- le partage de ressources parce qu'ils gèrent dynamiquement des composants matériels ou logiciels ;
- la fiabilité parce que l'arrêt d'une de ses parties ne conduit pas nécessairement à l'arrêt de la totalité du système.

On relève plusieurs degrés de prise en compte de la répartition dans les systèmes d'exploitation. Par exemple, les systèmes interconnectés par des réseaux généraux (Arpanet, Transpac, etc.) pour réaliser des applications comme le transfert de fichiers ou le courrier électronique sont faiblement coordonnés. Plus fortement coordonnés, on trouve les systèmes de gestion de fichiers répartis (NFS [Sandberg 85], ITC [Satyanaray 85], etc.) qui permettent l'accès aux fichiers indépendamment de leur localisation. La répartition intervient à tous les niveaux des systèmes d'exploitation répartis comme Amoeba [Tanenbaum 81], Chorus [Guillemont 82], Hermix [Berbers 87], Locus [Popek 81] ou le V-Kernel [Cheriton 84a] par exemple.

T.S.I. — Technique et Science Informatiques 0752-4072/88/04/359-14/$ 3.40

C'est ce dernier cas qui fait, pour l'essentiel, l'objet de cet article : un système d'exploitation unique est exécuté sur tous les processeurs du réseau. Ce système est un environnement intégré de telle sorte que tous les services sont indépendants du réseau et de leur localisation. Il est perçu par ses usagers de la même façon qu'un système d'exploitation centralisé, comme *UN* système, mais il s'exécute sur plusieurs processeurs distincts.

Objets, clients et serveurs

Un système d'exploitation peut être décrit comme un ensemble *d'objets* qui ont des interactions entre eux : processus, périphériques, fichiers, boîtes à lettres, catalogues, etc. La nature d'un objet est définie dès que les *opérations* qui permettent de le manipuler ont été précisées (modèle objet ou type de données abstrait [Liskov 74]).

En première approximation, un système est structuré comme un ensemble de *clients* et de *serveurs*. Un serveur gère un ou plusieurs objets : il les met en œuvre et réalise les opérations qui permettent leur manipulation. Un client est un utilisateur d'objets : il invoque des opérations sur les objets (il applique des opérations aux objets). Naturellement, quand un serveur invoque des opérations exécutées par d'autres serveurs, il se comporte également comme un client.

Dans un système d'exploitation réparti, clients et serveurs sont répartis. Même si les présentations externes des divers systèmes sont assez différentes, un système d'exploitation réparti peut toujours être décrit comme un ensemble de clients et de serveurs qui *s'échangent des messages*. C'est ce point de vue que nous adoptons dans cet article.

Objectifs de l'article

Les systèmes d'exploitation répartis ont beaucoup d'aspects communs avec les systèmes centralisés mais ils en diffèrent également de plusieurs manières. Des synthèses ont été publiées sur les systèmes d'exploitation répartis développés ces dernières années ([Guillemont 84], [Stankovic 84], [Tanenbaum 85], etc.). En général, elles concernent un ensemble de questions : la communication, la synchronisation, la protection, la désignation, la résistance aux pannes, etc.

Cet article ne traite qu'une de ces questions. Il présente un examen approfondi des problèmes soulevés par la désignation, ainsi qu'un aperçu des solutions qui ont été mises en œuvre, ces dernières années, pour les résoudre. Ces solutions sont illustrées par des exemples représentatifs des systèmes d'exploitation répartis développés pendant la première moitié des années 80 et actuellement disponibles. Un article plus théorique a été publié en 1981 [Watson 81]. Depuis, beaucoup de systèmes ont été développés, ce qui a permis d'approfondir le problème et de comparer les diverses solutions employées, qui en 1981 n'en étaient qu'au stade conceptuel.

Désigner un objet consiste à lui affecter un nom permettant de lui faire référence *(naming)*. Les mécanismes de désignation disponibles dans un système d'exploitation réparti doivent faire en sorte que les avantages potentiels de l'environnement réparti et le caractère unique du système soient respectés. Un système de désignation y est donc soumis aux contraintes suivantes :

- la désignation, l'accès et le partage d'objets doivent être indépendants de leur localisation ;
- les objets doivent pouvoir changer de localisation (migrer) sans changer de nom ;
- la relation entre le nom et l'objet doit rester constante malgré les pannes éventuelles ;

Liste des systèmes étudiés

Nous nous appuyons sur des exemples pris dans plusieurs systèmes répartis afin d'illustrer les concepts et les méthodes. Cependant, cet article n'est pas une étude approfondie ni un examen comparatif de ces systèmes. Les systèmes cités sont : Accent [Rashid 81], Amoeba [Tanenbaum 81], Apollo/DOMAIN [Leach 83], Chorus [Guillemont 82], Demos/MP [Powell 83], Grapevine et Cedar [Birrel 84], Hermix [Berbers 87] et V-System [Cheriton 84a].

Plan de l'article

Après cette introduction, nous présentons une classification des différents types de noms que l'on trouve dans les systèmes d'exploitation. Cette classification introduit une hiérarchie de couches de noms qui est à la base de notre analyse. Les deuxième, troisième et quatrième parties étudient l'application de cette classification à l'environnement réparti. Chacune de ces parties approfondit une couche différente et décrit, illustre et compare les solutions employées dans plusieurs systèmes. Finalement, nous nous penchons sur une notion qui est spécifique à l'environnement réparti, la notion de groupe d'objets. Ceci est traité dans la cinquième partie. Nous terminons par un paragraphe récapitulatif où nous présentons également certains domaines de recherche à approfondir.

1. Les noms dans les systèmes d'exploitation

1.1. Différence entre nom et adresse d'un objet

Un objet peut être repéré par un *nom* ou par une *adresse*. Un nom repère l'objet lui-même (l'entité logique), l'adresse repère sa représentation (son implantation).

Lorsqu'un objet est une ressource physique d'un calculateur (un périphérique par exemple), son adresse est l'adresse du périphérique dans la machine. Cette adresse est en général constante. Cependant, lorsque

l'objet est une entité créée par le programmeur (un fichier ou un processus par exemple), son adresse change lorsqu'il est déplacé en mémoire, pour des raisons d'allocation d'espace par exemple.

Un environnement réparti, l'adresse mémoire d'implantation d'un objet est complétée par l'adresse réseau de la machine où il réside. Si l'objet change de machine, il doit également changer d'adresse réseau. Par conséquent, les adresses ne sont pas adéquates pour désigner un objet.

En plus, quand un objet change d'adresse, cette adresse peut être réutilisée par un autre objet. Une adresse n'est donc pas une référence indépendante du temps. Par exemple, un numéro de téléphone annulé par un abonné peut être réutilisé pour un nouvel abonné. Quand nous téléphonons à un ami que nous n'avons pas vu depuis quelques années, nous ne sommes pas sûrs d'aboutir au bon interlocuteur. Un numéro de téléphone n'est donc pas approprié pour désigner un abonné [Shoch 78].

Du point de vue des systèmes d'exploitation, et plus particulièrement des systèmes d'exploitation répartis, tout repère d'un objet qui n'est pas indépendant du temps et de la localisation n'est pas adéquat. Nous utilisons le terme général « adresses » pour les repères qui ne possèdent pas cette indépendance.

1.2. Couches de noms dans les systèmes d'exploitation

La méthode utilisée pour résoudre les problèmes engendrés par la non-unicité des adresses est d'associer à chaque objet un *nom*. Ce nom ne change pas lorsque l'objet est déplacé en mémoire ; on dit qu'il est global. Pour faciliter la mise en œuvre, les systèmes d'exploitation utilisent des noms internes globaux (indépendants de la position en mémoire) et de taille réduite (en général des numéros). Nous les appelons *noms internes globaux*.

Dans quelques systèmes, les noms internes globaux de certains objets sont directement visibles par leurs clients. Dans d'autres, pour des raisons de protection ou autres (voir § 3), les noms directement utilisables par un processus sont, au contraire, des noms locaux ou contextuels. La traduction d'un nom local dans le nom interne global qu'il représente est assurée par les mécanismes d'invocation d'objets. Nous appelons les noms locaux des *noms internes locaux ou contextuels.*

L'acquisition, par un processus, des noms des objets qu'il veut invoquer *(liaison)* peut être réalisée à différents instants : soit avant l'exécution du programme, c'est-à-dire au moment de sa compilation, soit au moment du chargement du programme, soit juste au moment de l'invocation de l'objet.

Cette acquisition de noms peut mettre en jeu un ou plusieurs objets intermédiaires. En général, les objets créés par les programmeurs sont désignés par des *noms symboliques* (une chaîne de caractères). Ces noms symboliques conduisent aux noms internes des objets et sont souvent conservés dans des objets particuliers qui sont les *catalogues* du système. Les catalogues mettent en œuvre la correspondance entre des noms symboliques et des noms internes.

Comme le montre la figure 1, les systèmes d'exploitation utilisent les noms symboliques ou externes *(user-level* ou *user-oriented names)* — manipulés par les usagers du système — et les noms internes *(process-level names)* — manipulés par les processus. Ces derniers noms peuvent être de deux sortes : les noms locaux ou contextuels ou bien les noms globaux directement.

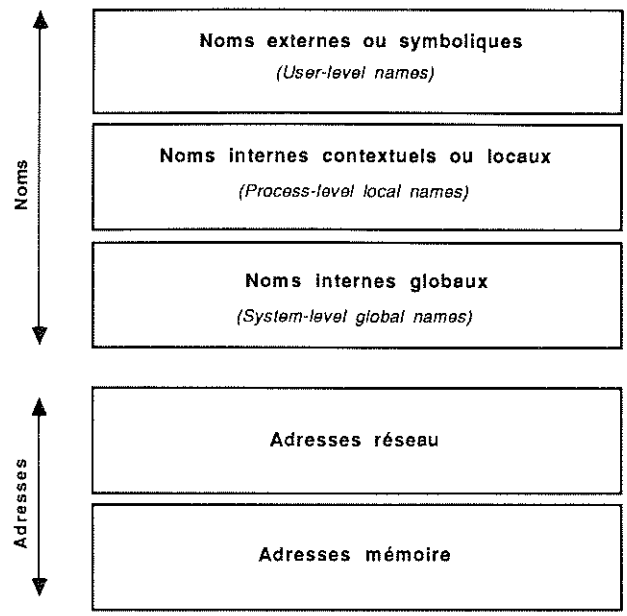

Figure 1. — Couches de désignation dans les systèmes d'exploitation.

Illustration : couches de désignation des fichiers dans Unix

Dans Unix [Ritchie 74], les usagers désignent les fichiers par des noms symboliques, comme par exemple : *« /bin/cat »*. Ces noms sont traduits par l'intermédiaire de catalogues qui associent les noms symboliques à des noms internes globaux (les numéros d'*i-nodes* [Thompson 78]).

Quand un processus ouvre un fichier, les catalogues sont utilisés pour établir une « chaîne d'accès » entre le nom contextuel (le *file-descriptor*) et la structure qui permet de repérer sa représentation (le *i-node* ou bloc de description du fichier) et qui contient les adresses des blocs du disque où il est enregistré. Ce nom interne contextuel du fichier ouvert est le résultat de l'opération d'ouverture. Il n'a de sens que dans le contexte du processus appelant.

Pour garantir la non-ambiguïté des numéros des *i-nodes*, le système Unix associe à chaque fichier (ou

plutôt à chaque *i-node*) un compteur de références. Ce compteur indique combien de fois le fichier est inscrit dans des catalogues. Quand le compteur d'un fichier atteint la valeur 0, le fichier est détruit et son numéro peut être réutilisé car le système est sûr que le fichier n'est plus utilisable.

Remarquons également que dans Unix les noms internes globaux des processus sont directement visibles par les clients du système. Il n'y a pas de noms internes contextuels pour désigner les processus.

1.3. Problèmes spécifiques à l'environnement réparti

Les couches de désignation que nous venons de présenter sont également présentes dans les systèmes d'exploitation répartis. Cependant, ces systèmes soulèvent des problèmes nouveaux pour la génération, la gestion et la traduction de noms.

Dans un système d'exploitation réparti, il n'y a pas de noyau centralisant les tables d'état du système. Dans le système coexistent autant d'exemplaires du noyau que de sites actifs. Ces exemplaires coopèrent par échange de messages, donc de façon asynchrone. Il devient donc impossible de calculer l'état global du système.

D'autre part, les pannes, arrêts et redémarrages des sites se produisent de manière asynchrone. On ne peut pas arrêter tout le système réparti pour éliminer d'éventuelles incohérences et le relancer dans un état cohérent (comme on le fait en environnement centralisé). En environnement réparti, il faut détecter et éliminer les incohérences pendant le fonctionnement normal du système.

Par exemple, si un objet est présent sur un site, il est assez difficile de savoir exactement dans quels sites cet objet est repéré. Par conséquent, la méthode des compteurs de références pour garantir l'unicité des noms internes globaux (voir l'illustration précédente) est très difficile à mettre en œuvre.

Ces contraintes exigent des méthodes nouvelles de génération et d'interprétation de noms. En particulier, dans les systèmes d'exploitation répartis, un nom interne global doit être indépendant de la machine où réside actuellement l'objet qu'il repère. D'autre part, il faut trouver une manière plus simple de résoudre le problème de l'unicité d'un nom. Ces deux problèmes sont résolus par l'usage de *noms internes uniques et globaux*. Nous les appelons dans cet article UID *(Unique-IDentifiers)* globaux ou simplement UID.

2. UID dans les systèmes d'exploitation répartis

Un nom interne unique et global (UID) est un nom qui possède les deux propriétés suivantes :

- *Unicité dans l'espace.* Si un processus a mémorisé un nom global d'objet, il n'a pas à mémoriser sa location, donc l'objet peut être déplacé dans le système en conservant son identité. Ces noms peuvent également être passés d'un site à un autre, sans nécessiter de traduction car leur signification est indépendante de la localisation. Le système est capable de retrouver un objet, à partir de son UID, quelle que soit sa localisation.
- *Unicité dans le temps.* Un UID enregistré, aussi longtemps soit-il, ne génère aucune ambiguïté. Lorsque ce nom est utilisé, ou bien l'objet qu'il désigne existe encore, ou bien l'objet n'existe plus et ce nom ne peut désigner aucun autre objet.

L'utilisation d'UID soulève deux problèmes : leur *génération* et leur *traduction* dans les adresses (réseau et mémoire) d'implantation des objets qu'ils désignent. L'obtention de l'adresse réseau s'appelle la *localisation*.

2.1. Génération d'UID

Les UID sont produits à la création des objets selon une méthode décentralisée. Elle consiste, en général, à juxtaposer une estampille qui détermine, de façon unique, le site de création et une estampille qui est unique dans ce site (*cf.* [Liskov 81] pour une discussion détaillée).

Plusieurs méthodes sont utilisées pour la génération d'estampilles uniques de site : l'administration manuelle (Chorus), le numéro de série du processeur (Apollo/DOMAIN [Leach 82]) ou le tirage aléatoire (V-System [Cheriton 84a]).

Les méthodes de génération d'estampilles uniques sur un site sont également multiples. On utilise une horloge du site de création — *noms historiques* — (Apollo/DOMAIN, Cedar [Birrel 84]), une horloge et des compteurs incrémentés à chaque création (Chorus), ou un générateur de nombres aléatoires (V-System, Amoeba [Tanenbaum 81]).

Remarquons, au passage, que des noms internes non réutilisés ont été introduits dans des systèmes centralisés (des noms internes *historiques* notamment) mais que les problèmes de performance qu'ils présentent (par rapport à des noms dont l'unicité est garantie par l'usage de compteurs de références) ont été un frein à leur généralisation.

Illustration : la génération d'UID dans le système Apollo/DOMAIN

Le système de gestion d'objets d'Apollo/DOMAIN (fichiers) possède un espace de désignation interne d'objets au travers d'UID. Ces noms sont indépendants de la localisation des objets qu'ils désignent.

Ils sont construits par la juxtaposition d'un champ de 36 bits qui reflète l'heure de création de l'objet, d'un champ de 20 bits qui indique le site de création et d'un champ réservé de 8 bits (voir *fig.* 2). L'unicité de ces noms est garantie par le matériel : le nom du site de création est alloué à chaque machine en usine ; l'heure de création provient d'une horloge spéciale strictement croissante.

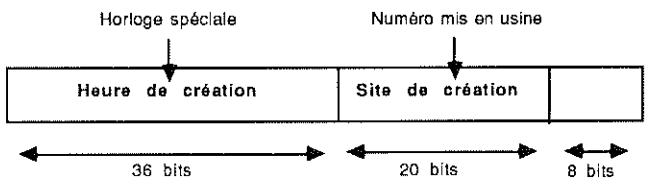

Figure 2. — Anatomie d'un UID dans le système Apollo/DOMAIN.

2.2. Localisation d'objets nomades

Le second problème qui concerne l'utilisation d'UID est la localisation d'objets nomades. L'intérêt de pouvoir changer le site de résidence des objets est multiple. D'une part, cela permet de répartir la charge : les objets sont déplacés pour les rapprocher de leurs clients ou pour optimiser l'utilisation des ressources disponibles. D'autre part, c'est un facteur de résistance aux pannes : si un site tombe en panne, on peut transférer sur un autre site les objets qu'il gérait.

La migration d'objets est mise en œuvre par plusieurs systèmes d'exploitation répartis. Par exemple, le système Apollo/DOMAIN permet la migration des fichiers, DEMOS/MP, V-System et Amoeba permettent la migration de processus, Accent, Chorus et Hermix permettent la migration de portes de communication.

La migration comporte deux aspects : le transfert de l'état des objets nomades et leur localisation. Nous développons ici seulement quelques-unes des stratégies de localisation d'objets nomades.

2.2.1. *Stratégie de diffusion*

Cette méthode est la plus simple. Elle consiste à diffuser sur tous les sites une requête de localisation contenant l'UID de l'objet à localiser. Seul le site où l'objet réside (s'il y en a un) répond.

- *Avantages.* Simplicité et compatibilité avec les pannes car seul les objets inaccessibles ne sont pas localisables.
- *Désavantages.* L'utilisation de la diffusion soulève quelques problèmes (performance, échelle d'application, portabilité, sécurité, ...).
- *Améliorations.* Dans chaque site, des caches de localisation sont utilisés pour éviter un recours systématique à la diffusion.
- *Exemples.* Chorus dans ses premières versions [Zimmermann 84], Amoeba, V-System [Theimer 85], [Welch 86], ...

2.2.2. *Stratégie heuristique*

Cette méthode consiste à évaluer la localisation d'un objet en utilisant un ensemble d'hypothèses heuristiques sur sa localisation. Elle est utilisée par Apollo/DOMAIN [Leach 82] pour localiser les fichiers. Voici, à titre d'illustration, une hiérarchie d'hypothèses heuristiques pour localiser un fichier :

1. Chercher le fichier sur le site où il a été créé (son UID contient son site de création) ;
2. sinon, le chercher sur le même site que le catalogue où il est inscrit ;
3. sinon, chercher le fichier localement ;
4. sinon, diffuser une requête de localisation du fichier.

- *Avantages.* En connaissant bien la manière dont un système est sollicité, ainsi que son comportement, il est possible de définir une hiérarchie d'hypothèses heuristiques performante.
- *Désavantages.* Cette méthode doit être affinée cas après cas et en fonction de l'expérience acquise et des statistiques connues sur le fonctionnement et l'utilisation du système. Cet affinage exige des modifications de l'algorithme et peut devenir incompatible avec la croissance du système.

2.2.3. *Stratégie reposant sur l'évaluation itérative de la localisation*

Cette méthode, introduite par DEMOS/MP [Powell 83], consiste à associer à chaque UID connu sur chaque site un champ (variable) contenant une suggestion de localisation de l'objet qu'il désigne. Simultanément, le système enregistre dans tous les sites où un objet nomade a résidé (sites relais), une suggestion sur sa localisation actuelle. Localiser un objet revient alors à évaluer itérativement sa localisation, à partir des différentes suggestions, jusqu'à aboutir au site de résidence effectif. La demande de localisation est redirigée successivement, depuis le site initial, jusqu'au site de résidence, en traversant les éventuels sites relais.

- *Avantages.* Le prix de cette stratégie est directement proportionnel au taux de migration des objets.
- *Désavantages.* A la destruction d'un objet, la mise à jour des différents relais de localisation s'avère très coûteuse. D'autre part, un objet existant peut devenir inaccessible pour certains clients, si l'évaluation de sa localisation traverse des relais en panne.
- *Optimisations.* A chaque itération, si la suggestion de départ s'avère fausse, elle est mise à jour [Powell 83]. Cela revient à enlever un relais de la boucle d'itération.

2.2.4. *Stratégie reposant sur des serveurs de localisation*

Cette méthode utilise des catalogues qui traduisent l'UID d'un objet en l'adresse de son site de résidence. Ces catalogues peuvent exister en plusieurs copies et sont gérés par des serveurs spécifiques, de même nature que les serveurs de noms, voir § 4.

- *Avantages.* Elle isole fonctionnellement le service de localisation.
- *Désavantages.* A la création, destruction et migration d'un objet, il faut mettre à jour les catalogues de localisation. Comme ces catalogues résident éventuellement sur d'autres sites que l'objet nomade, leur cohérence avec la réalité sera difficile à maintenir. Par exemple, si des pannes ont introduit des incohérences, le service de localisation peut donner une

indication de localisation d'un objet qui n'existe plus. Si les catalogues de localisation n'existent pas en plusieurs copies, la panne d'un site peut rendre inaccessible un objet existant sur un site actif.

- *Les solutions adoptées.* Dans la pratique, les catalogues de localisation sont disponibles dans plusieurs serveurs. Cependant, ces serveurs ne garantissent pas une cohérence stricte entre les différentes copies. Une telle exigence réduirait leur taux de disponibilité [Birrell 82].
- *Exemples.* Grapevine [Birrell 82], Clearinghouse [Oppen 81], ...

Illustration : la localisation des portes dans Chorus

Le système Chorus autorise la migration des portes de communication. Nous présentons ci-dessous l'algorithme qui est utilisé pour localiser les portes non locales [Rozier 87].

Chaque site maintient une table qui indique le site de résidence actuel des portes *créées sur ce site*. En cas de migration ou de destruction d'une de ces portes, cette table est mise à jour. Quand un message est envoyé à partir d'un site A vers une porte P non locale, une requête de localisation de P est dirigée vers le site où P a été créée (l'UID de P contient son site de création). Le message est alors acheminé vers le site contenu dans la réponse.

En même temps, l'information de localisation est introduite dans un cache du site A de telle sorte que la demande de localisation ne soit ré-exécutée que si la porte migre à nouveau.

Si le site de création d'une porte nomade est en panne, la localisation de cette porte devient impossible. En face de cette situation, le site de l'émetteur diffuse une demande de localisation de la porte, et s'il obtient une réponse, il introduit cette information dans son cache de localisation comme dans le cas normal.

Cet algorithme contient des éléments empruntés à plusieurs des algorithmes présentés ci-dessus. D'une part, l'information de localisation d'une porte est recherchée d'abord dans le site qui l'a créée. En effet, la pratique montre que la plupart des portes ne migrent pas. D'autre part, les catalogues de localisation sont considérés comme une sorte de cache partagé. S'ils s'avèrent incohérents ou inaccessibles, la diffusion est alors utilisée.

2.2.5. *Conclusion et comparaison à propos de la localisation*

Pour obtenir une bonne performance, l'idéal serait que chaque site connaisse l'adresse du site de résidence actuel de n'importe quel objet distant. Cependant, maintenir ces tables cohérentes et constamment à jour serait trop coûteux. Par conséquent, la stratégie consiste à évaluer la localisation le plus tardivement possible, de préférence à la demande *(lazy evaluation)*, et à conserver dans un cache la localisation des objets invoqués précédemment.

Néanmoins, ces informations ne peuvent être considérées que comme des suggestions *(hints)* de localisation. En effet, l'objet peut toujours migrer entre deux invocations, ou peut devenir inaccessible à cause d'une panne. Un serveur de localisation n'est en fin de compte qu'un recueil partagé et très disponible de suggestions de localisation. Une stratégie heuristique d'évaluation de la localisation revient à coder des suggestions dans l'algorithme.

En environnement réparti, l'évaluation tardive et l'usage de caches de suggestions deviennent une méthode de plus en plus répandue [Mullender 87]. En effet, vouloir maintenir une cohérence stricte entre toutes les tables du système reviendrait à réduire sa disponibilité car une mise à jour ne serait possible que si toutes les tables étaient accessibles.

L'usage de caches et de suggestions exige, d'une part, une méthode sûre pour détecter les incohérences et, d'autre part, une stratégie pour trouver l'information au cas où la suggestion est inexistante ou qu'elle s'avère fausse.

L'usage d'UID répond très convenablement à cette première exigence. L'utilisation de la diffusion est la stratégie de secours la plus simple, mais elle souffre de problèmes de portabilité et d'échelle d'application. Ces deux problèmes mis à part, elle continue à être une méthode très intéressante surtout dans le cadre des réseaux locaux.

La méthode la plus générale pour localiser des objets utilise des serveurs de localisation. Cependant, elle est également la plus lourde et ne peut se justifier totalement que par un souci de portabilité et d'indépendance face à la taille et de l'extension géographique du système.

3. Noms internes et protection

L'utilisation d'UID qui, rappelons-le, permet l'accès à un objet indépendamment de sa localisation et de son éventuelle migration est très puissante. Néanmoins, elle pose un problème de protection. En effet, si dans un système les UID sont directement visibles aux clients et que leur connaissance permet d'invoquer les objets, le système n'est pas protégé car, dans ce cas de figure, rien n'empêche un processus de générer tous les noms possibles.

Un UID présente les propriétés d'adressage d'une *capacité* mais pas ses propriétés de protection. En effet, une capacité (*capability* [Fabry 74], [Ferrié 75]) vers un objet est un repère de l'objet, qui n'est pas falsifiable, et qui regroupe :

- le nom unique (UID) de l'objet,
- et les « droits » qui indiquent les opérations permises sur l'objet, sous contrôle de cette capacité.

Par la suite nous n'avons pas l'intention de discuter les problèmes soulevés par la répartition en matière de protection. Nous discuterons seulement quelques-unes de leurs conséquences sur les couches de désignation

présentes dans les systèmes d'exploitation répartis actuellement opérationnels.

Certains de ces systèmes utilisent des capacités pour protéger l'adressage de messages aux points d'accès aux objets (portes, processus, etc.). L'invocation des objets est ainsi implicitement protégée. D'autres mettent en œuvre des capacités qui contrôlent toutes les opérations sur n'importe quel objet. Tous ces systèmes utilisent une des deux méthodes qui sont présentées ci-après.

3.1. Noms contextuels ou locaux

Cette méthode de protection consiste à introduire un contexte de désignation interne par processus, contrôlé par le système, et qui cache les UID, ainsi que les droits qui leur sont associés, derrière des noms internes contextuels ou locaux.

Ces contextes sont une nouvelle couche de désignation interne qui permet la traduction des noms locaux en UID et vice-versa. Les systèmes Accent [Rashid 81], Chorus, DEMOS/MP et Hermix [Berbers 87] par exemple, mettent en œuvre une couche de noms internes locaux pour protéger les droits d'émission et de réception de messages aux points d'accès aux objets.

Illustration : la désignation des portes de communication dans Accent

Une porte Accent [Rashid 81] est un objet protégé, dans lequel des messages peuvent être déposés par un processus, et duquel des messages peuvent être retirés par un autre processus. Toutes les opérations disponibles pour invoquer un objet ne sont accessibles qu'au travers de portes.

A chaque porte sont associés trois droits :

- le droit 'possession' (de la porte),
- le droit 'réception' (de messages),
- et le droit 'émission' (de messages).

Pour pouvoir manipuler une porte, un processus doit posséder une capacité sur cette porte avec les droits adéquats. Cette capacité est un nom contextuel de la porte. Ce nom n'a de sens que dans le contexte du processus qui le connaît.

Un processus crée une porte par l'opération *Allocate Port*, laquelle retourne le nom contextuel de l'objet créé comme conséquence de l'appel. Initialement, le créateur possède tous les droits sur la porte, mais ces droits peuvent être transmis aux autres processus dans des messages (les messages Accent sont typés).

Dans le cas de destruction ou d'inaccessibilité soudaine d'une porte, tous les processus qui possèdent le droit d'émission sur cette porte sont avertis par un signal d'exception et perdent cette capacité. Ceci est possible parce que le système contrôle toutes les portes connues par un processus. Cette méthode de désignation et de protection a été reprise dans le système Mach [Accetta 86].

3.2. Avantages et désavantages des noms contextuels ou locaux

Avantages

L'emploi de noms internes locaux associés à des droits est de même nature que la protection par des listes de capacités *(capability lists)*. En effet, un processus ne peut acquérir un nom interne que par le biais du système. Il est donc impossible qu'un processus génère tous les noms, ou qu'un processus, par erreur logicielle, emploie un UID erroné et envoie des messages à des processus quelconques.

D'autre part, comme le système contrôle tous les repères connus, il est (en principe) plus simple d'introduire des fonctionnalités de notification d'exceptions (signalant aux processus qu'un objet qu'ils connaissent vient d'être détruit) ou de ramasse-miettes (détruisant les objets qui ne sont plus accessibles).

Désavantages

D'abord, il faut considérer la lourdeur introduite par cette méthode de désignation. L'utilisation d'un nom contextuel impose sa traduction par le système. Si un processus communique avec beaucoup d'entités, sa table d'adressage peut devenir très grande (d'où une occupation plus importante en mémoire).

Ensuite, l'échange de noms et de droits entre processus et leur passage comme paramètre d'opérations nécessitent des mécanismes spéciaux. En effet, cet échange exige la traduction des noms entre contextes et l'analyse des droits. Les solutions sont multiples : utilisation de messages typés (comme dans Accent, voir [Rashid 81] ou introduction d'appels systèmes spécifiques (comme dans Chorus, voir [Rozier 87]).

Finalement, cette méthode soulève un autre problème (rarement analysé dans la littérature) : si les capacités sont gardées par le noyau et que les processus ne les connaissent que par des noms locaux, comment les restaurer après une panne ? Ceci peut être particulièrement critique lorsqu'au moment de la panne s'exécute un serveur de noms (un dépositaire de capacités par excellence).

3.3. Espaces de noms clairsemés

Certains systèmes utilisent une autre méthode pour la mise en œuvre de capacités. Elle consiste en l'utilisation d'UID engendrés dans un *espace de noms clairsemés (sparse address spaces)*.

L'idée est la suivante : si les UID des objets sont choisis aléatoirement dans un espace de désignation comportant N bits, on peut choisir pour N une valeur suffisamment grande pour que la probabilité de trouver illégitimement l'UID d'un objet, dans un espace de temps t, soit arbitrairement petite ([1]).

([1]) Cette probabilité peut être au moins du même ordre de grandeur que la probabilité de trouver le mot de passe de l'administrateur du système en réalisant plusieurs essais [Tanenbaum 81].

Certains systèmes, faiblement protégés, font reposer la protection de l'espace d'adressage des objets sur cette méthode. Tel est par exemple le cas du *V*-System qui utilise des UID de processus, codés sur 32 bits et engendrés aléatoirement (voir § 2).

Cette démarche est incomplète du point de vue de la protection dans la mesure où les UID ainsi engendrés ne partagent pas toutes les propriétés des capacités. En particulier, des droits ne leur sont pas associés.

Le système Amoeba, qui utilise également cette méthode de protection, associe, en plus, des droits aux UID. Ces droits sont codés dans des champs supplémentaires de ces noms par des procédés de cryptographie. Ces procédés empêchent la modification des droits par les processus clients. On trouvera dans [Tanenbaum 81] et [Mullender 84] une description détaillée de la méthode utilisée.

3.4. Avantages et désavantages des espaces de noms clairsemés

Avantages

Le principal avantage des espaces de noms clairsemés est que les noms internes sont directement manipulés par les processus. En particulier, ils peuvent être échangés entre processus clients directement dans des messages normaux, sans intervention du système. L'échange de noms entre les objets est ainsi facilité.

Désavantages

La contrepartie est que le système ne peut plus contrôler « qui connaît qui ». Pratiquement, cela rend plus difficile l'introduction de mécanismes automatiques de traitement d'exceptions ou de ramasse-miettes.

3.5. Conclusion

Le choix des méthodes de désignation interne d'un système est un aspect fondamental de sa conception. Les conséquences de ce choix sont multiples : protection, acquisition de noms, traitement d'exceptions, résistance aux pannes, ramasse-miettes d'objets inaccessibles, ... Malheureusement, le plus souvent, à cause de sa complexité intrinsèque, ce problème est traité d'une manière plus ou moins *ad hoc* [Watson 81].

4. Noms externes ou symboliques

Dans les paragraphes précédents, nous avons examiné les noms internes *(process-level names)* utilisés par les systèmes d'exploitation répartis. Nous allons maintenant analyser les noms directement manipulés par les usagers.

Conceptuellement, le sous-système de désignation symbolique *(directory service)* est cette partie d'un système d'exploitation, formée d'un ensemble de catalogues, qui réalise la traduction d'identificateurs symboliques (des chaînes de caractères) en des UID d'objets. Naturellement, dans un système réparti cet ensemble de catalogues est également réparti.

Comme les catalogues sont eux-mêmes des objets qui possèdent des UID, leurs références peuvent apparaître dans d'autres catalogues. L'espace des noms symboliques devient ainsi un graphe (souvent un arbre). L'analyse du nom symbolique d'un objet est alors assimilé à un parcours dans ce graphe (également appelé « chemin d'accès à l'objet »).

La couche de désignation symbolique joue un rôle central pour la commodité d'emploi d'un système. Elle permet :

- la *lisibilité,* car les usagers préfèrent utiliser des chaînes de caractères, avec une signification, à des numéros dont la structure leur est cachée,
- *l'acquisition de noms internes uniques ou contextuels,* par consultation de catalogues, et
- *l'établissement de relations* entre objets en structurant l'espace des noms du système.

Ce sous-système dépend beaucoup du cahier des charges du système réparti (l'analyse de toutes les variantes possibles sort du cadre de cet article) mais, du point de vue de son organisation interne, deux approches fondamentales sont utilisées.

La première consiste à confier la gestion des catalogues à des *serveurs de noms (name servers).* La deuxième consiste à confier la gestion d'un catalogue au serveur qui gère les objets que ce catalogue désigne. C'est le cas de beaucoup de serveurs de fichiers. Dans la pratique, certains systèmes mélangent les deux approches.

Les paragraphes suivants présentent deux sous-systèmes de désignation symbolique différents dans leur philosophie. Le premier, celui du système Grapevine, est construit selon l'approche du serveur de noms. Le deuxième, celui du V-System, intègre la gestion des noms symboliques aux gestionnaires des objets. La description de chaque système sera suivie par la liste de leurs avantages et inconvénients.

4.1. Illustration : le système Grapevine

Grapevine [Birrell 82] est un système réparti qui propose simultanément un sous-système de messagerie électronique et un sous-système de désignation symbolique. Tel qu'il est présenté dans [Birrell 82], l'environnement de Grapevine s'étend de la Californie (U.S.A.) jusqu'en Angleterre (G.B.) et comporte plus de 1 000 stations de travail et quelques milliers d'usagers.

L'espace des noms symboliques de Grapevine est structuré comme un arbre à deux niveaux. Logiquement, un catalogue-racine contient des identificateurs de sous-catalogues ou partitions *(registries).* Chacune de ces partitions contient des identificateurs d'objets. Un nom est de la forme 'O.P.', où 'O' est l'identificateur de l'objet et 'P' l'identificateur de la partition. (Exemples : 'jose.inria', 'yolande.kuleuven'.) La notion de partition (sous-catalogue) ne correspond ni à celle de site ni à celle de serveur. Elle est entièrement logique et peut être établie en fonction des besoins administratifs. Une partition peut regrouper des objets

d'une même localisation géographique, d'une organisation présente dans plusieurs régions géographiques, ou d'un service spécifique (toutes les imprimantes du système par exemple).

Un descripteur est associé à chaque nom. Grapevine distingue deux types de descripteurs.

1. *Les individus*. Leurs descripteurs enregistrent des informations de protection, leurs adresses réseau ([2]), etc. Un nom du type individu sert à désigner des usagers, des serveurs, etc.
2. *Les groupes*. Leurs descripteurs enregistrent des informations de protection et la liste de leurs membres (des individus ou des groupes). Les noms du type groupe peuvent regrouper un ensemble de serveurs qui rendent le même service (et donc servent à désigner le service), peuvent représenter une liste de diffusion de courrier, etc.

Les utilisations de ce sous-système de désignation sont multiples. Par exemple, le client d'un serveur de fichiers consulte son descripteur pour obtenir son adresse réseau, un utilitaire d'impression consulte le groupe des serveurs d'impression pour en choisir un, etc. Le caractère global et unique de l'espace de désignation de Grapevine en fait un référentiel *unique* de désignation symbolique.

Pour des raisons de performance et de résistance aux pannes, les catalogues existent en plusieurs copies. La mise en œuvre fait une utilisation récursive de ses fonctionnalités :

- Le service est disponible dans plusieurs serveurs. Le client peut s'adresser à n'importe quel serveur à propos de n'importe quel nom. Si le serveur contacté ne connaît pas le nom, sa requête est acheminée vers un autre serveur.
- Le catalogue global est connu de tous les serveurs. Chacune des partitions est connue de N serveurs mais pas nécessairement de tous les serveurs.
- Le service Grapevine est lui-même décrit par une partition, connue de tous les serveurs. Dans cette partition, chaque individu représente un serveur Grapevine et chaque groupe représente une partition. Ce groupe contient la liste des serveurs qui connaissent cette partition.
- Grapevine ne garantit pas une cohérence stricte entre les différentes copies d'une partition. Il garantit seulement qu'elles convergent vers le même état après un laps de temps qui dépend de l'état des serveurs et du réseau.

4.2. Évaluation de Grapevine

Le système Grapevine s'appuie sur des serveurs de noms. Il est écrit spécialement pour la désignation : cette fonction est bien isolée. Les objets dont il permet la désignation sont surtout des usagers, des boîtes à lettres, des serveurs, etc. Il ne s'occupe pas de la désignation d'objets du genre fichiers (bien que naturellement les serveurs de fichiers soient désignés par Grapevine).

Avantages

- Les noms symboliques de Grapevine sont structurés indépendamment de la localisation. Les seules contraintes sur la structuration de l'espace des noms sont de caractère administratif.
- La désignation est une fonction bien isolée. Ceci garantit l'uniformité de la sémantique des opérations sur les noms symboliques et permet la duplication aisée des informations.

Désavantages

- Comme la gestion des noms est faite indépendamment de la gestion des objets qu'ils désignent, un problème de cohérence se pose. Par exemple, la suppression d'un objet exige la suppression de son nom du service de désignation. Si l'un des serveurs concernés tombe en panne pendant l'opération, une incohérence peut subsister quelque part dans le système et le service de désignation peut considérer que l'objet existe toujours.
- L'introduction d'un service spécialisé dans la désignation exige une interaction supplémentaire entre un client et un serveur de noms avant d'accéder à un objet. Cette pénalisation serait trop importante si, par exemple, on utilisait Grapevine pour désigner des fichiers (et non leurs serveurs).

4.3. Illustration : la désignation symbolique dans le V-System

Description

Le V-System [Cheriton 84a] est un système d'exploitation réparti qui comprend un petit noyau réparti, le V-Kernel, et des serveurs système, également répartis. Chaque serveur gère un ou plusieurs objets. Le noyau met en œuvre des primitives d'échange de messages pour permettre l'invocation des objets.

Chaque processus reçoit à sa naissance un UID de 32 bits. Le nom interne global et unique d'un objet (que nous appellerons dans ce paragraphe OID) est formé de deux champs (*fig.* 3) : l'UID du serveur qui le gère et un nom spécifique au serveur (qu'il ne réutilise

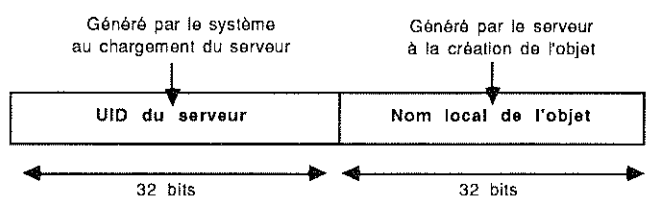

Figure 3. — Structure d'un nom interne d'objet (OID)

([2]) Grapevine associe une adresse réseau à un nom du type individu (et non un UID) pour des raisons liées à son environnement d'exécution. Néanmoins, du point de vue de la présente discussion sur la désignation symbolique cet aspect n'est pas significatif.

que le plus tard possible). Les UID et les OID sont directement visibles aux clients ([3]).

La démarche pour la désignation symbolique est la suivante [Cheriton 84b] : chaque serveur qui gère des objets désignés symboliquement doit également implanter les catalogues qui les désignent. Chaque serveur possède donc son propre sous-système de désignation symbolique, et les différents sous-systèmes sont indépendants les uns des autres.

Un serveur peut également enregistrer dans un catalogue l'OID d'un objet géré par un autre serveur. L'ensemble des graphes de désignation devient alors une forêt entrelacée.

Tout message concernant une opération sur un objet désigné symboliquement possède un en-tête standard. Cet en-tête comporte un code d'opération, l'OID du catalogue où l'interprétation du nom doit commencer et un chemin d'accès. Le message qui sert à l'invocation de l'objet doit être envoyé au serveur dont l'UID est contenu dans l'OID du catalogue où commence l'interprétation.

L'interprétation d'un nom est un calcul explicitement réparti. Le serveur qui reçoit le message commence la traduction du chemin d'accès à partir du catalogue dont l'OID est spécifié dans l'en-tête. A ce niveau le code opération est ignoré. Si la traduction aboutit, le code opération est considéré et l'opération est appliquée à l'objet.

Toutefois, si pendant la traduction le serveur rencontre un OID qui désigne un objet d'un autre serveur, il remplace l'OID qui est dans l'en-tête du message par celui qu'il a rencontré, le chemin d'accès par sa partie non encore analysée, et il achemine la requête vers le serveur associé au nouvel OID.

Un serveur peut ainsi participer à l'interprétation de chemins d'accès désignant des objets qu'il ne connaît pas, concernant des opérations qu'il ne peut pas exécuter et comportant des syntaxes qu'il ne sait pas interpréter.

Pour faciliter l'utilisation, chaque usager possède un serveur de désignation privé, le 'serveur de préfixes'. Ce serveur maintient un catalogue qui associe des noms symboliques aux OID de différents catalogues de la forêt des noms. En particulier, des identificateurs prédéfinis sont associés aux OID des catalogues-racine des différents serveurs du réseau. Une syntaxe d'exception, de la forme '[préfixe] chemin d'accès', spécifie que l'interprétation de 'chemin d'accès' doit commencer dans le serveur de préfixes et non dans le catalogue courant de l'usager.

Les serveurs de préfixes acquièrent les OID, à la demande de l'usager, en utilisant la diffusion de requêtes du type *'quel serveur gère l'objet de nom symbolique x et quel est son OID ?'*, (cela est également proposé dans [Welch 86]).

Évaluation du sous-système de désignation symbolique du V-System

Comme beaucoup de systèmes, le V-System incorpore la désignation des objets comme les fichiers dans les serveurs qui les gèrent mais, pour une question de cohérence, il généralise cette philosophie à tous les gestionnaires d'objets qui sont désignés symboliquement (fenêtres, connexions aux réseaux publics, etc.).

Avantages

- Cette approche est plus adaptée à l'interconnexion de systèmes préexistants (ayant déjà un sous-système de désignation) dans la mesure où elle est plus flexible et n'impose pas une base commune. A la limite, un chemin d'accès peut contenir différentes parties, interprétables selon différentes syntaxes. L'interprétation des noms n'est pas faite par un service unique mais déléguée aux serveurs concernés.
- La cohérence entre les catalogues et les objets gérés par le même serveur est facile à maintenir. De plus, l'accès aux objets est généralement plus efficace car il ne faut pas consulter systématiquement un serveur de noms.

Désavantages

- Puisque tous les serveurs des différents types d'objets sont concernés par la désignation, une duplication de fonctions est introduite. Cette duplication soulève un problème d'uniformité. L'uniformité de la sémantique des opérations concernant la gestion des catalogues n'est mise en œuvre que par un ensemble de conventions. On suppose donc l'existence d'une sorte de « gentlemen's agreement » pour rendre viable ce schéma.
- Cette approche suggère également une structuration de l'espace des noms (internes et symboliques) calquée sur la répartition des objets dans les différents serveurs. L'administration du système est donc subordonnée à des contraintes liées à la localisation. Ces contraintes empêchent, par exemple, la migration d'un objet à l'extérieur du serveur qui l'a créé. Seuls les serveurs peuvent migrer.

4.4. Conclusion sur la comparaison des deux approches

Si un ensemble d'objets est géré par un serveur et que celui-ci gère également la plupart des catalogues où ils sont inscrits (localité des références), la mise en œuvre du sous-système de désignation symbolique est plus simple :

- en face de pannes, la cohérence entre le contenu des catalogues et l'état des objets est plus simple à maintenir.
- l'interprétation d'un chemin d'accès est éventuellement plus performante parce qu'elle est localisée,

([3]) Remarquons que les OID ainsi formés ont des propriétés qui font qu'un objet ainsi désigné ne peut pas migrer à l'extérieur du serveur qui le gère. En effet, l'UID du serveur donne implicitement une information logique de localisation de l'objet.

- en cas de panne d'un site, seuls les noms des objets non accessibles ne sont plus interprétables.

Une telle organisation du sous-système de désignation symbolique privilégie la simplicité et la performance. Cependant, elle est quand même assez restrictive. En effet, elle empêche une redistribution libre des objets sur les sites ou sur les serveurs.

Au contraire, la séparation de la gestion des noms et de la gestion des objets qu'ils désignent est une méthode plus générale mais moins performante. De plus, elle comporte des problèmes supplémentaires de cohérence.

Finalement, l'utilisation de noms symboliques globaux, et sa mise en œuvre par des catalogues privilégiés (« catalogues-racines du réseau »), sont très importantes dans un système d'exploitation réparti. Néanmoins, comme ces catalogues sont vitaux pour le fonctionnement du système, ils doivent exister en plusieurs copies. La manière la plus simple de réaliser cette duplication consiste à utiliser un serveur de noms.

Pour l'interprétation des noms symboliques, les deux approches que nous venons de présenter ont simultanément des avantages et des inconvénients. Elles sont donc complémentaires l'une de l'autre et beaucoup de systèmes les mélangent [Rozier 87]. Par exemple :

- Les systèmes de gestion de fichiers répartis dans lesquels les graphes de désignation propres à chaque serveur sont interconnectés arbitrairement *(« montage distant »)* ont besoin de serveurs de noms « logiquement centralisés » pour désigner les usagers, les serveurs et les sites. Tel est, par exemple, le cas du service 'yellow-pages' introduit avec NFS [Sandberg 85] ou du 'Berkeley Name-server' [Terry 84]. Ces services de désignation constituent une base uniforme et cohérente pour résoudre les problèmes de désignation d'usagers, de sites, de serveurs, etc.
- Souvent, les systèmes de gestion de fichiers répartis qui contiennent la notion de « racine du réseau » utilisent un serveur de noms « logiquement centralisé » pour gérer les catalogues de plus haut niveau de l'arbre du système et ils décentralisent ensuite la gestion de la désignation des différents sous-arbres au soin des différents serveurs de fichiers.

5. La notion de groupe

Comme les systèmes répartis disposent de plusieurs sites, ils permettent la mise en œuvre de fonctionnalités ou d'objets qui existent en plusieurs copies. Cet aspect demande des mécanismes spécifiques de désignation que nous présentons brièvement ci-dessous.

Du point de vue des clients, un tel objet est unique, mais, en réalité, dans le système coexistent plusieurs copies de cet objet. Son unicité n'est que logique. Pour les clients, un tel objet doit avoir un nom unique (interne ou symbolique) qui le désigne. Néanmoins, du point de vue du gestionnaire de l'objet, il est indispensable de pouvoir désigner chacune des copies. Ainsi, du point de vue de la désignation, cet objet peut être vu comme une paire qui regroupe son nom et la liste des noms de ses copies. Une telle paire s'appelle souvent un *groupe*.

La duplication et le regroupement (au sens large du terme) peuvent se présenter de plusieurs manières. Par exemple :

- Un fichier peut être reproduit de telle sorte que ses différentes copies soient toujours identiques.
- Un ensemble de serveurs fonctionnellement équivalents sont regroupés dans un groupe de serveurs ou *service*, comme par exemple le service de désignation. (Pour un client, le serveur contacté lui est indifférent.)
- Un ensemble de processus appartenant à la même application sont regroupés. (Pour permettre un contrôle collectif d'exécution.)
- Un ensemble de portes de communication sont regroupées pour permettre la diffusion de messages.

La notion de groupe prend toute sa valeur lorsque l'on considère les opérations qui lui sont associées : d'une part pour la gestion du groupe (construction, évolution dynamique, liste des membres, test d'appartenance, etc.) et d'autre part pour les communications avec un groupe (diffusion à tous les membres d'un groupe, envoi à N membres parmi P, diffusion atomique, « appel de procédure distant » avec un membre du groupe, etc.). C'est la richesse, la protection et l'efficacité de ces opérations qui donnent toute sa puissance à ce concept. Enfin, pour que sa mise en œuvre soit performante et contrôlée, il est très courant que les groupes soient intégrés comme une fonction de base du système et non sous forme de bibliothèques ou d'utilitaires.

Les mises en œuvre des groupes par les systèmes répartis sont diverses. Dans certains systèmes, l'invocation d'un groupe est automatiquement répercutée à l'ensemble de ses membres. Dans d'autres systèmes, ne sont disponibles que des fonctionnalités élémentaires pour la diffusion de messages à tous les membres. Enfin, quelques systèmes ne gèrent que la relation entre le nom d'un groupe et le nom de ses membres. C'est au client de se servir de cette liste de noms selon ses besoins. Dans tous les cas, nous trouvons des listes d'appartenance réparties. Pour les détails sur la gestion de ces listes, le lecteur doit consulter la bibliographie citée ci-dessous.

Voici quelques exemples concrets :

- La notion de groupe symbolique qui est présente dans Grapevine (voir § 4) s'intègre dans la couche de désignation symbolique. Un groupe symbolique est un ensemble de noms symboliques. Ceux-ci sont utilisés pour la diffusion de courrier électronique, pour la constitution de listes d'entités qui partagent un droit, pour la désignation symbolique de services (en regroupant le nom des serveurs qui le rendent), pour la mise en œuvre de la duplication des catalogues de désignation, voir [Birrell 82], etc.

- La notion de groupe de processus (comme le fournissent le V-System [Cheriton 85] et le système Chorus [Rozier 87] par exemple) s'intègre dans la couche de noms internes. Ces groupes désignent un ensemble de processus par un UID unique. Ils servent pour l'application d'opérations sur l'ensemble de leurs membres (la destruction en une seule opération de tous les membres par exemple).
- La notion de groupe de portes (Chorus), permet la diffusion de messages [Legatheaux 86]. Cette fonctionnalité s'intègre également dans la couche de désignation interne. Un groupe de portes possède un UID unique (qui peut être également enregistré dans un catalogue).

La notion de groupe a aussi des implications particulières sur l'acquisition de noms. En effet, les groupes permettent une indirection supplémentaire dans ce processus. Le client peut choisir explicitement, ou le système implicitement, un membre arbitraire du groupe.

Supposons, par exemple, qu'un groupe comprenne l'ensemble des serveurs d'imprimantes du système. Quand un client désire imprimer un document, au lieu d'invoquer un serveur particulier, il invoque le groupe des imprimantes. C'est le système qui implicitement choisit une des imprimantes parmi celles qui sont disponibles à ce moment-là. Une telle facilité est mise en œuvre par le sous-système d'acquisition de noms *(binding)* de Cedar (voir [Birrell 84]).

Cette indirection est particulièrement intéressante si le groupe est formé d'entités qui réalisent la même fonction ou qui gèrent le même objet, et qui sont interchangeables. Si le client n'utilise que le nom du groupe, les éventuelles reconfigurations du service lui sont complètement cachées.

Le système Chorus présente une fonctionnalité de ce genre qui est intégrée à l'adressage de messages. Quand un message est adressé à un groupe de portes, le client peut spécifier si ce message doit être diffusé à tous les membres du groupe (adressage 1 à *N*) ou simplement adressé à un seul membre du groupe, n'importe lequel — adressage 1 à (1 parmi *N*) (voir [Rozier 87] et [Legatheaux 86] pour la mise en œuvre).

En guise de conclusion, nous pouvons résumer l'intérêt des groupes de noms par sa capacité à représenter une multiplicité de points d'accès.

6. Récapitulatif

Afin de faire face aux nouveaux problèmes introduits par la répartition, les systèmes d'exploitation répartis doivent être structurés à l'aide de mécanismes simples mais puissants et généraux.

Ainsi, comme nous l'avons vu dans le § 2, les mécanismes de désignation interne de ces systèmes sont bâtis sur une couche de noms uniques et globaux (UID). Ces noms partagent les propriétés des capacités du point de vue de l'adressage, mais non du point de vue de la protection. Leurs avantages sont évidents : simplicité, non-ambiguïté, indépendance de la localisation, indépendance du contexte d'utilisation, etc.

Que ce soit en environnement réparti ou centralisé, la désignation des objets peut être assimilée à un problème de traduction de noms entre couches de désignation. Cette traduction est mise en œuvre à l'aide de tables de correspondance et de catalogues. Dans un système réparti, l'objet « client », l'objet « catalogue » et l'objet « cité par le catalogue » peuvent être répartis. L'asynchronisme des communications et des pannes introduit, dans ce contexte, des problèmes aigus de performance et de cohérence.

Ces dernières années, les recherches sur la désignation ont été dominées par l'invention de méthodes permettant de « vivre avec des incohérences momentanées », de les détecter et de les corriger au besoin. En particulier, l'évaluation tardive et l'usage de caches et de suggestions *(hints)* (voir le § 2) sont devenus des procédés très employés [Terry 87], [Cabrera 87] et [Nelson 88].

La désignation est un domaine de recherches très actives. Parmi les problèmes à approfondir, nous pouvons citer :

- Le traitement formel des couches de désignation, de traduction et d'interprétation de noms (voir par exemple [Comer 86]).
- Les moyens de désignation interne et externe adaptés à des systèmes à très grande échelle ou hétérogènes [Terry 86], [Schwartz 87], [Cabrera 87], [Howard 88].
- La localisation d'objets par classe ou attributs et non pas seulement par nom [Cabrera 87].
- Une plus forte intégration des méthodes de désignation des systèmes et des environnements de développement. Par exemple, l'usage d'UID pour dénoter des types (caractérisant des interfaces) permettrait la détection automatique, à l'exécution, des incohérences (de version ou de conception) entre les différents processus d'une application.

Remerciements

Les auteurs remercient B. Deslandes, M. Guillemont, L. Mosseri, M. Shapiro et C. Kaiser ainsi que les membres du Comité de rédaction pour leurs commentaires des versions précédentes de cet article.

BIBLIOGRAPHIE

[Accetta 86] M. ACCETTA, *et al.* : *Mach : a New Kernel Foundation for UNIX Development,* Carnegie-Mellon University Report, mai 1986.

[Armand 86] F. ARMAND, M. GIEN, M. GUILLEMONT, P. LÉONARD : *Towards a Distributed UNIX System. The Chorus Approach,* EUUG, Autumn'86, Manchester, septembre 1986, pp. 413-431.

[Berbers 87] Y. BERBERS, B. DE DECKER, H. MOONS and P. VARBAETEN : *The Design of the Hermix Distributed System,* 34th ISMM International Conference, Lugano, Switzerland, juin 1987.

[Birrell 82] A. D. BIRRELL, R. LEVIN, R. M. NEEDHAM, M. D. SCHROEDER : *Grapevine : An Exercise in Distributed Computing,* Communications of the ACM, vol. 25, avril 1982, pp. 260-274.

[Birrell 84] A. D. BIRREL, B. J. NELSON : *Implementing Remote Procedure Calls,* ACM Transactions on Computer Systems, vol. 2, février 1984, pp. 39-59.

[Cabrera 87] L. F. CABRERA and J. WYLLIE : *Quick Silver Distributed File Services : An Architecture for Horizontal Growth,* Research Report, RJ 5578(56697), Computer Science Department, IBM Almaden Research Center, 1987.

[Cheriton 84a] D. R. CHERITON : *The V-Kernel : a Software Base for Distributed Systems,* IEEE Software, avril 1984, pp. 19-42.

[Cheriton 84b] D. R. CHERITON, T. P. MANN : *Uniform Access to Distributed Name Interpretation in the V-System,* Research Report, Computer Science Department, Standford University, décembre 1984.

[Cheriton 85] D. R. CHERITON, W. ZWAENEPOEL : *Distributed Process Groups in the V-Kernel,* ACM Transactions on Computer Systems, vol. 3, 2 mai 1985, pp. 77-107.

[Comer 86] D. E. COMER, L. L. PETERSON : *A Model of Name Resolution in Distributed Systems,* The 6th International Conference in Distributed Computing Systems Cambridge, Massachusetts, mai 1986, pp. 523-530.

[Fabry 74] R. S. FABRY : *Capability-based Addressing,* Communications of the ACM, vol. 17, 7 juillet 1974, pp. 403-412.

[Ferrié 75] J. FERRIÉ : *Contrôle de l'accès aux objets dans les systèmes informatiques,* Thèse d'État ès-Sciences Mathématiques, Paris, septembre 1975.

[Guillemont 82] M. GUILLEMONT : *The CHORUS Distributed Operating System : Design and Implementation,* International Symposium on Local Computer Networks, Florence, Italy, avril 1982, pp. 207-223.

[Guillemont 84] M. GUILLEMONT : *Étude comparative de quelques systèmes répartis,* TSI, vol. 3, 1, janvier 1984, pp. 5-21.

[Howard 88] J. HOWARD *et al.* : *Scale and Performance in a Distributed File System,* To Appear in ACM Transactions on Computer Systems — 1988. Presented at the 11th ACM Symposium on Operating Systems Principles, Austin, Texas, USA, novembre 1987.

[Lampson 83] B. W. Lampson : *Hints for Computer System Design,* Ninth ACM Symposium on Operating Systems Principles, Bretton Woods, New Hampshire, OSR vol. 17, 5, october 1983, pp. 33-48.

[Leach 82] P. J. LEACH *et al.* : *UIDS as Internal Names in Distributed Systems,* ACM Symposium on Principles of Distributed Computing, Ottawa, Canada, août 1982.

[Leach 83] P. J. LEACH *et al.* : *The Architecture of an Integrated Local Network,* IEEE Journal on Selected Areas in Communications, vol. 1, 5, novembre 1983, pp. 842-857.

[Legatheaux 86] J. LEGATHEAUX MARTINS : *La désignation et l'édition de liens dans les systèmes d'exploitation répartis,* Thèse de l'Université de Rennes I, novembre 1986.

[Levine 87] P. H. LEVINE : *The Apollo Domain Distributed File System,* Distributed Operating Systems : Theory and Practice, Springer Verlag, NATO ASI Series, 1987.

[Liskov 74] B. H. LISKOV, S. ZILLES : *Programming with Abstract Data Types,* ACM SIGPLAN Notices, vol. 9, 4, avril 1974, pp. 50-59.

[Liskov 81] B. H. LISKOV : *Report on the Workshop on Fundamental Issues in Distributed Computing,* Fallbrook, California, December 1980, ACM OSR, vol. 15, 3, juillet 1981, pp. 9-38.

[Mullender 84] S. J. MULLENDER, A. S. TANENBAUM : *Protection and Resource Control in Distributed Operating Systems,* Computer Networks, vol. 8, 1984, pp. 421-432.

[Nelson 88] M. NELSON, B. WELCH and J. OUSTERHOUT : *Caching in the Sprite Network File System,* To Appear in ACM Transactions on Computer Systems — 1988. Presented at the 11th ACM Symposium on Operating Systems Principles, Austin, Texas, USA, novembre 1987.

[Oppen 81] D. C. OPPEN and Y. K. DALAL : *The Clearinghouse : A Decentralized Agent for Locating Named Objects in a Distributed Environment,* ACM Transactions on Office Information Systems, vol. 1, 3, juillet 1983.

[Powell 83] M. L. POWELL, B. P. MILLER : *Process Migration in DEMOS/MP,* 9th ACM Symposium on Operating Systems Principles ACM OSR, vol. 17, 5, octobre 1983, pp. 110-119.

[Rashid 81] R. F. RASHID, G. G. ROBERTSON : *Accent : A Communication Oriented Network Operating System Kernel,* 8th ACM Symposium on Operating System Principles, Pacific Grove, California, décembre 1981.

[Ritchie 74] D. M. RITCHIE, K. THOMPSON : *The UNIX Time-Sharing System,* CACM, vol. 17, 7, juillet 1974, pp. 365-375.

[Rozier 87] M. ROZIER, J. LEGATHEAUX MARTINS : *The CHORUS Distributed Operating System : Some Design Issues,* Distributed Operating Systems : Theory and Practice, Springer Verlag, NATO ASI Series, 1987.

[Sandberg 85] R. SANDBERG, D. GOLDBERG, S. KLEIMAN, D. WALSH, B. LYON : *Design and Implementation of the Sun Network File System,* Usenix, Portland, juin 1985, pp. 119-130.

[Satyanaray 85] M. SATYANARAYANAN *et al.* : *The ITC Distributed File System : Principles and Design,* 10th ACM Symposium on Operating Systems Principles, Orcas Island, Washington, décembre 1985, pp. 35-50.

[Schwartz 87] M. SCHWARTZ, J. ZAHORJAN and D. NOTKIN : *A Name Service for Evolving Heterogeneous Systems,* Presented at the 11th ACM Symposium on Operating Systems Principles, Austin, Texas, USA, novembre 1987.

[Schoch 78] J. F. SHOCH : *Inter-Network Naming, Addressing, and Routing,* 17th IEEE Computer Society International Conference — Compcon, septembre 1978, pp. 72-79.

[Stankovic 84] J. H. STANKOVIC : *A Perspective On Distributed Computer Systems,* IEEE Transactions on Computers, vol. C-33, 12, décembre 1984.

[Tanenbaum 81] A. S. TANENBAUM, S. J. MULLENDER : *An Overview of the AMOEBA Distributed Operating System,* ACM OSR, vol. 15, 3, juillet 1981, pp. 51-64.

[Tanenbaum 85] A. S. TANENBAUM, R. VAN RENESSE : *Distributed Operating Systems,* ACM Computing Surveys, vol. 17, 4, décembre 1985, pp. 419-470.

[Terry 84] D. B. TERRY, M. PAINTER, D. W. RIGGLE, S. ZHOU : *The Berkeley Internet Name Domain Server,* USENIX Summer'84, Salt Lake City, Utah, juin 1984, pp. 23-31.

[Terry 86] D. B. TERRY : *Structure-free Name Management for Evolving Distributed Environments,* The 6th International Conference in Distributed Computing Systems Cambridge, Massachusetts, mai 1986, pp. 502-508.

[Terry 87] D. B. TERRY : *Caching Hints in Distributed Computer Systems,* IEEE Transactions on Software Engineering, vol. SE-13, 1, janvier 1987, pp. 48-54.

[Theimer 85] M. M. THEIMER, A. LANTZ, D. CHERITON : *Preemptable Remote Execution Facilities for the V-System,* The Tenth ACM Symposium on Operating Systems Principles, décembre 1985, pp. 2-12.

[Thompson 78] K. THOMPSON : *UNIX Implementation,* The Bell Systems Technical Journal, vol. 5, 6, Part 2, juillet 1978, pp. 1931-1946.

[Welch 86] B. WELCH, J. OUSTERHOUT : *Prefix Tables : A Simple Mechanism for Locating Files in a Distributed System,* The 6th International Conference in Distributed Systems Computing Systems, Cambridge, Massachusetts, mai 1986, pp. 523-530.

[Watson 81] R. W. WATSON : *Identifiers (Naming) in Distributed Systems,* Distributed Systems, Architecture and Implementation, Lecture Notes in Computer Science number 105, Springer Verlag, 1981.

[Wupit 83] A. WUPIT : *Comparison of UNIX Network Systems,* 1983 ACM Conference on Personal and Small Computers, San Diego, ACM SIGPC Notes, vol. 6, 3, décembre 1983, pp. 99-108.

[Zimmermann 84] H. ZIMMERMANN, M. GUILLEMONT, G. MORISSET, J. S. BANINO : *CHORUS : A Communication and Processing Architecture for Distributed Systems,* Rapport de recherche INRIA 328, septembre 1984.

Article reçu le 2 juillet 1987, version révisée le 22 janvier 1988.

José Legatheaux Martins est né au Portugal. Il a été à l'Universidade Nova de Lisboa depuis 1977 jusqu'en 1983, d'abord comme étudiant en Informatique, puis comme enseignant de programmation réalisant de la recherche dans le domaine du graphique. Son travail de thèse a été développé à l'INRIA (France), depuis 1983 jusqu'en 1986, où il a été intégré à l'équipe CHORUS et a participé à la conception et réalisation du système réparti de même nom. Depuis la fin de l'année de 1987, J. Legatheaux Martins est retourné au Portugal où il est maintenant enseignant en Informatique à la Faculdade de Ciências de Lisboa.

Departamento de Informática Universidade Nova de Lisboa
Quinta da Torre - 2825 Monte da Caparica (Portugal)

Yolande Berbers est actuellement chercheur au département d'informatique de la K.U. Leuven, Belgique, au sein du projet HERMIX qui est dédié aux systèmes distribuées. Elle obtint de la même université le diplôme d'ingénieur civil en informatique, et y défendit sa thèse de doctorat en 1987. En 1985 et 1986 elle participa à temps particl au projet CHORUS à l'INRIA (Rocquencourt, France).

K.U. Leuven, département d'informatique
Celestijnenlaan 200A, B-3030 Leuven (Belgique).