SYNTHESE

Problèmes de désignation, de localisation et d'accès dans les systèmes répartis à objets

Daniel Hagimont *
Jacques Mossière **

** INRIA Rhône-Alpes*
*** Institut National Polytechnique de Grenoble*

Laboratoire Bull-IMAG, 2 av. de Vignate, 38610 Gières
Internet : { Daniel.Hagimont, Jacques.Mossiere} @imag.fr

RESUME. *L'introduction de la répartition dans les systèmes d'exploitation a donné lieu à deux types de systèmes, les systèmes répartis à gestion de fichiers et les systèmes répartis à espace d'objets. Dans ces derniers, l'objet est utilisé comme unité de conservation, de désignation, de partage et de protection de l'information. Ces systèmes doivent donc fournir les mécanismes permettant de désigner et de localiser les objets sachant qu'ils peuvent être déplacés ; ils doivent permettre le partage des objets entre des processus s'exécutant sur des machines différentes tout en assurant une certaine isolation entre ces processus et ces objets ; ils doivent enfin fournir des mécanismes d'accès efficaces. Après une présentation de ces problèmes, nous passons en revue et évaluons dans cet article les solutions actuellement adoptées ou explorées.*

ABSTRACT. *Research in distributed operating systems has led to two main types of systems: distributed file systems and object based distributed systems. In the latter, objects are used as unit for data storage, naming, sharing and protection. Those systems must provide mechanisms that allow object naming and location while objects may migrate between nodes ; object sharing between distributed processes should be provided without compromizing isolation between processes and objects; finally, object addressing mechanisms must be efficiently implemented. After the description of the problems, this paper presents and evaluates the different solutions proposed in current research projects.*

MOTS-CLES : *systèmes répartis, objets, adressage, protection, désignation*
KEY WORDS: *distributed systems, objects, addressing, protection, naming*

1. Introduction

Les principaux concepts utilisés aujourd'hui pour la gestion de l'information dans les systèmes d'exploitation ont été définis au seuil des années 70 dans des systèmes comme Multics [ORG 72] ou Hydra [WUL 81]. La conservation de l'information prend la forme de gestion de segments ou d'objets persistants, complétée par une gestion hiérarchisée de noms symboliques. Des mécanismes de partage et de protection élaborés sont définis et réalisés à l'aide de dispositifs matériels spécifiques.

Les années 80 ont vu la généralisation de la téléinformatique et des réseaux locaux: on est passé de gros systèmes partageant leurs ressources entre des processus à des systèmes composés de stations de travail interconnectées. Sur ces nouvelles configurations, les systèmes fournissent un espace de désignation global et masquent la répartition. Parmi les solutions proposées, on peut distinguer celles qui fournissent un espace de fichiers répartis [LEV 90] de celles qui fournissent un espace uniforme d'objets [CHI 91]. Dans les deux cas, la répartition a introduit de nouveaux problèmes, comme le déplacement des objets ou des fichiers d'un site à un autre ou le partage d'objets entre des processus s'exécutant sur des sites différents.

Dans cet article, nous nous intéressons aux systèmes répartis offrant un espace uniforme d'objets, c'est à dire aux systèmes dans lesquels l'objet est la principale abstraction et est utilisé comme unité de conservation, de désignation, de partage et de protection de l'information. Concernant la répartition, notre étude vise donc l'ensemble des systèmes qualifiés de clients-serveurs, les systèmes de gestion de base de données réparties ainsi que les systèmes offrant une mémoire virtuelle répartie permettant le partage et la protection des objets. La propriété essentielle des objets qui nous intéresse est la propriété d'encapsulation, c'est à dire qu'un objet est un ensemble de variables d'état qui ne sont accessibles que par un ensemble de procédures définies dans la classe de l'objet. Nous détaillons plus particulièrement les mécanismes de désignation, de localisation, de partage, de protection et d'accès aux objets dans ces systèmes.

Afin de clarifier la présentation, nous distinguons deux niveaux de mémoire :

- la mémoire de stockage, qui gère la persistance des objets,
- la mémoire d'exécution, niveau de mémoire dans lequel un objet doit être chargé avant exécution.

La suite de l'article présente les techniques utilisées pour gérer les objets à ces deux niveaux de mémoire ; elle est structurée comme suit. La deuxième partie présente les concepts et définit les termes employés. La partie 3 étudie la mise en œuvre de la fonction de stockage (désignation, localisation et migration des objets). La partie 4 est consacrée à la mémoire d'exécution (partage, protection et adressage). Les principaux résultats des parties 3 et 4 sont regroupés en 5 dans des tableaux récapitulatifs. Enfin, nous concluons en deux volets, établissement d'une typologie des systèmes existants et élaboration de perspectives de recherche.

2. Concepts et terminologie

Le but de cette section est de présenter et définir les concepts et la terminologie utilisés dans cette synthèse. Elle propose un modèle qui servira de base à l'analyse des mécanismes mis en œuvre dans les systèmes répartis à objets.

Nous passons tout d'abord en revue les types de noms manipulés dans ces systèmes, puis nous présentons comment ces noms sont utilisés (résolution de noms). Enfin, nous décrivons le modèle sur lequel est fondée l'analyse.

2.1. *Types de noms*

Dans un système, on peut trouver différents types de noms :

• *Noms symboliques :*

Les noms symboliques, le plus souvent des chaînes de caractères, sont employés par l'utilisateur. L'espace de noms est en général hiérarchisé (arbre de catalogues).

Dans notre cas, nous supposons que ces noms sont gérés à un niveau supérieur, comme une application des systèmes que nous étudions. Le service de nommage symbolique permet d'associer un nom symbolique à un nom interne au système.

• *Noms d'objets internes au système :*

La gestion de ce nommage est le sujet principal de cet article. Ces noms permettent d'identifier de façon unique les objets gérés dans le système. Ils sont utilisés par les programmes et peuvent être stockés dans des objets pour construire des structures complexes.

La connaissance du nom interne d'un objet permet de le partager (si les règles de protection du système le permettent). L'accès à un objet se fait par appel d'une des méthodes de son interface d'accès.

• *Adresses virtuelles :*

Ce sont les noms utilisés par les processeurs pour désigner des objets. A l'exécution, il faut donc être capable d'obtenir à partir du nom interne d'un objet l'adresse virtuelle à laquelle l'objet est accessible par le processeur.

• *Adresses disque :*

Ce sont les noms utilisés pour désigner des objets sur les disques. Lorsqu'un objet persistant n'est pas encore accessible en mémoire (physique ou virtuelle), il faut le retrouver sur disque. Il faut alors obtenir à partir du nom interne de l'objet l'adresse disque de l'objet.

2.2. *Résolution des noms*

Dans son étude des systèmes de nommage et de liaison, J. H. Saltzer [SAL 78] modélise un système de nommage d'objets de la façon suivante :

• Un **contexte** est une entité qui enregistre des associations entre des noms et des objets. Un nom n'a donc une signification que par rapport à un contexte.

• Une opération appelée **liaison** permet d'ajouter une association entre un nom et un objet dans un contexte. On dit que le nom est lié à l'objet dans ce contexte.
• Une opération appelée **résolution de nom** permet d'interroger un contexte pour localiser un objet à partir de son nom.

En fait, des contextes de résolution de noms existent à tous les niveaux dans un système et sont souvent utilisés pour enregistrer des associations entre des noms de niveaux différents dans le système. Par exemple, un contexte peut être utilisé pour enregistrer la liaison entre un nom interne d'objet et une adresse virtuelle, ou bien entre un nom interne et une adresse disque.

On peut en général distinguer les contextes globaux des contextes locaux. Les contextes globaux donnent une traduction universelle aux noms qu'ils définissent. On dit que les noms sont globaux ou absolus. Un nom global a donc le même sens dans le système tout entier. Les contextes locaux permettent de ne pas avoir la même traduction d'un nom dans tout le système. Les contextes locaux sont utilisés pour gérer des espaces de noms différents, par exemple des espaces virtuels différents.

2.3. *Modèle de système*

Dans notre modèle, nous distinguons deux niveaux de mémoire dans lesquels les objets peuvent être manipulés. Cette distinction est adoptée pour des raisons de présentation et peut totalement disparaitre dans certains systèmes.

Nous appelons **mémoire d'exécution** le niveau de mémoire dans lequel un objet doit être chargé pour pouvoir être adressé par un processeur au moyen d'une adresse virtuelle. Cette mémoire d'exécution est composée à un instant donné de l'ensemble des espaces de pagination existant sur l'ensemble des stations de travail du réseau.

Nous appelons **mémoire de stockage** le niveau de mémoire contenant les objets de façon persistante. Elle est constituée par les disques des machines du réseau.

Un objet est toujours présent en mémoire de stockage. Il réside en mémoire d'exécution s'il est utilisé par au moins une **activité** (flot d'exécution séquentiel dont le comportement consiste à exécuter des méthodes sur des objets).

Une activité qui utilise un nom d'objet pour appeler une méthode sur cet objet va tout d'abord essayer de résoudre ce nom dans la mémoire d'exécution. Si cette résolution réussit, alors l'activité peut utiliser l'objet. Dans le cas contraire, une résolution du nom en mémoire de stockage doit avoir lieu pour retrouver l'objet sur disque. Puis cet objet doit être rendu disponible en mémoire d'exécution.

Si l'objet n'est pas en mémoire d'exécution, on dit généralement qu'il y a **défaut d'objet** en mémoire d'exécution.

On peut donc caractériser le processus d'adressage dans un système à base d'objets par :
• la gestion de la mémoire d'exécution,
• la gestion de la mémoire de stockage,
• la résolution des noms dans ces deux espaces.

2.4. *Mémoire de stockage*

La mémoire de stockage permet de gérer la persistance des objets. Elle doit mettre en œuvre la résolution des noms des objets, permettant de retrouver un objet dans l'espace de stockage. Cette résolution de nom consiste à obtenir une adresse disque à partir d'un nom d'objet.

La mémoire de stockage est répartie sur l'ensemble des stations de travail connectées au réseau. Pour des raisons matérielles (les disques) ou de structuration logique, la mémoire de stockage est généralement composée de sous ensembles d'objets appelés partitions.

La mémoire de stockage doit autoriser la migration des objets entre partitions, permettant notamment de regrouper des objets liés dans le cadre d'une même application. L'opération de résolution de nom est aussi appelée "localisation" des objets en mémoire de stockage.

2.5. *Mémoire d'exécution*

En mémoire d'exécution, le système permet aux activités de partager les objets tout en assurant certaines propriétés relatives à la protection.

La mémoire d'exécution doit mettre en œuvre la résolution des noms d'objets, aussi appelée "adressage" des objets en mémoire d'exécution. Cette résolution de nom consiste à obtenir une adresse virtuelle à partir d'un nom d'objet. La liaison est l'opération permettant d'associer une adresse virtuelle à un objet. On appelle donc **liaison dans un espace virtuel** l'opération qui associe un objet à une plage d'adresses dans cet espace virtuel.

A un instant donné, une activité s'exécute dans un seul espace virtuel. Lorsqu'un objet est lié à une adresse donnée dans un espace virtuel, cela signifie que l'utilisation de cette adresse par une activité dans cet espace virtuel permet d'adresser cet objet si l'objet est accessible par cette activité. L'accessibilité d'un objet par une activité dépend de la notion de **domaine de protection** .

Un domaine de protection[1] est un ensemble de droits d'accès à des objets. Une activité s'exécute dans un seul domaine à un instant donné ; elle ne peut accéder qu'aux objets pour lesquels le domaine possède des droits.

Une opération permettant à une activité de changer de domaine est généralement fournie par le système. L'utilisation de cette opération nécessite un droit qui doit être présent dans le domaine de protection dans lequel l'activité s'exécute.

On appelle **couplage dans un domaine de protection** l'opération qui consiste à rendre un objet accessible dans un domaine, en y ajoutant un droit d'accès à cet objet.

---

[1] Nous utiliserons le terme domaine pour domaine de protection.

Une activité s'exécute à un instant donné dans un domaine unique et dans un espace virtuel unique, mais plusieurs activités peuvent en général partager un domaine ou un espace virtuel. Les domaines de protection sont indépendants des espaces virtuels gérés. Une activité peut changer de domaine sans changer d'espace virtuel et vice-versa. Il est possible de gérer un espace virtuel global à tous les domaines, ou de gérer un espace virtuel propre à chaque domaine.

Si les concepts d'espace virtuel et de domaine de protection sont logiquement différents, leurs réalisations sont très liées dans les systèmes existants. Dans la pratique, les machines actuelles n'offrent qu'un mécanisme de traduction dynamique d'adresses avec contrôle de la validité (lecture, écriture, exécution) d'un accès à une page. La notion de domaine de protection est obtenue par restriction des liaisons dans les espaces virtuels, chaque espace virtuel réalisant alors un domaine de protection.

Les sections 3 et 4 détaillent respectivement les principes de mise en œuvre de la mémoire de stockage et de la mémoire d'exécution.

3. Désignation, localisation et migration en mémoire de stockage

Nous présentons maintenant les différentes approches pour la gestion de la mémoire de stockage.

Comme dans un système de gestion de fichiers, le nom que l'on attribue à un objet peut être relatif ou absolu, ce qui revient à gérer des contextes locaux ou globaux (section 2.2). La mémoire de stockage étant composée d'un ensemble de partitions, un nom relatif signifie une désignation à l'intérieur d'une partition. Un nom absolu est un nom pouvant désigner tout objet quelle que soit sa partition de résidence.

Ces deux approches s'expliquent par le fait qu'un nom absolu pouvant désigner tous les objets du système est naturellement plus long (en nombre de bits) qu'un nom relatif. Pour des applications comme les bases de données où les références entre objets sont fréquentes et où le nombre d'objets gérables est un critère important, la réduction de la taille des identificateurs d'objets est appréciable. De plus, réduire la taille des identificateurs d'objets peut être également avantageux pour la vitesse d'exécution, notamment pour les opérations d'affectation ou de comparaison de noms.

3.1. *Noms relatifs*

Dans le cas d'un nommage relatif, un objet ne peut contenir une référence que vers un objet de la même partition, ce qui revient à avoir un contexte de résolution de noms local à la partition.

Pour permettre aux objets de faire référence à des objets appartenant à d'autres partitions, le système doit gérer des objets spéciaux appelés des *liens de poursuite*

vers des objets externes à la partition. Un lien de poursuite contient l'identification de la partition contenant l'objet, ainsi que l'identification de cet objet dans sa partition. La localisation d'un objet est faite tout d'abord dans la partition contenant la référence utilisée. Le système détecte si l'objet référencé dans la partition courante est ou non un lien de poursuite. Si c'est le cas, le lien de poursuite est utilisé pour atteindre l'objet dans sa partition de résidence.

La migration entre partitions est réalisée en recopiant l'objet dans la partition cible et en remplaçant l'ancienne copie par un lien de poursuite. Ainsi, on assure que tous les noms d'objets présents dans le système restent valides. On peut raccourcir lors des accès les chaînes de liens de poursuite lorsqu'elles sont de longueur supérieure à un lien. Lorsqu'un objet arrive dans une partition, on peut également supprimer le lien de poursuite à cet objet s'il y en a un.

L'utilisation de noms relatifs pose toutefois deux problèmes :

• Le même nom peut désigner des objets différents dans deux partitions différentes (problème des homonymes).

Avec le nommage relatif, on ne peut pas rendre accessibles en mémoire d'exécution les noms d'objets de la même manière qu'en mémoire de stockage, car deux objets différents dans deux partitions différentes peuvent avoir le même nom. Ce problème est généralement résolu au niveau de la mémoire d'exécution en remplaçant le nom relatif en mémoire de stockage par une autre désignation qui distingue les objets.

• Deux noms différents (dans la même ou différentes partitions) peuvent désigner le même objet (problème des synonymes).

Il faut notamment s'en rendre compte lorsque l'on charge un objet en mémoire d'exécution pour ne pas le faire plusieurs fois. Lorsqu'un objet doit être localisé, son nom est résolu dans la partition de l'objet contenant ce nom et des liens de poursuite sont suivis si nécessaire. On peut alors identifier un objet de façon unique par le nom de sa partition de résidence et son nom local dans cette partition. Une autre technique consiste à allouer à chaque objet un nom unique dans tout le système et à stocker ce nom dans l'état de l'objet.

On trouve une réalisation du nommage relatif dans le projet Thor [LIS 92]. Les noms relatifs contenus dans un objet sont transformés en adresses virtuelles lorsque l'objet est amené en mémoire d'exécution (cette technique, appelée *mutation de pointeur* ou *swizzling*, est décrite dans la section 4.3.1). C'est à ce moment (chargement) que le système distingue les objets ayant le même nom local en tenant compte de la partition de provenance de l'objet. Ainsi, les noms relatifs ne sont jamais rendus visibles aux programmes clients qui ne voient que des adresses virtuelles, ce qui résout le problème de la localité des noms. Pour déterminer si différents noms font référence au même objet, Thor stocke un identifiant unique (*Uid*) dans chaque objet.

3.2. *Noms absolus*

On oppose au nommage relatif le nommage absolu, avec lequel un objet peut désigner directement tout objet du système. Ceci correspond à avoir un contexte global de résolution de noms. Dans ce cas, la localisation d'un objet peut être plus ou moins aisée selon que le nom contienne ou non des informations sur la localisation de l'objet.

Noms absolus indépendants de la localisation

Lorsque les noms des objets sont indépendants de leur localisation, alors la localisation est généralement assurée par un ensemble de serveurs coopérants répartis. Chaque fois qu'un objet doit être localisé en mémoire de stockage, cette organisation de serveurs doit être interrogée afin de déterminer dans quelle partition se trouve l'objet, ce qui est peu efficace. Par contre, cette solution simplifie la gestion de la migration des objets : une migration n'est enregistrée que par les serveurs de localisation et le principe de la localisation n'est pas modifié.

Différentes stratégies de gestion de ces serveurs coopérants sont présentées dans [LEG 88], dont notamment :

• *La gestion par diffusion :*

Il y a en général un serveur par site. Un message de localisation est envoyé à tous les sites et seul le site où l'objet réside répond. Des caches de localisation sont utilisés pour éviter un recours systématique à la diffusion. Cette solution, difficilement extensible à de grands réseaux, est utilisée dans le système Amoeba [TAN 90].

• *La gestion de catalogues dupliqués :*

Des catalogues de localisation sont dupliqués et répartis sur les machines. Une localisation est effectuée en consultant le premier catalogue accessible. Dans la plupart des cas, les serveurs ne garantissent pas une cohérence stricte entre les copies, afin de garantir une haute disponibilité. Un tel serveur donne une indication de localisation ([BIR 82]). Si cette indication est fausse, il faut contacter d'autres serveurs, par diffusion par exemple.

Noms absolus dépendants de la localisation

Supposons maintenant que des informations de localisation soient incluses dans le nom de chaque objet afin d'accélérer leur localisation. Dans ce cas, l'attribution du nom de l'objet lors de sa création est fonction de l'endroit où l'objet est créé et la localisation de l'objet est directe, car l'identification de la partition contenant l'objet est insérée dans le nom de l'objet. Le problème d'une telle solution est la réalisation de la migration d'objet. Nous passons en revue les différentes améliorations en partant de la solution fondée sur les liens de poursuite :

• *Les liens de poursuite :*

Cette méthode a été introduite par DEMOS/MP [POW 83].

Comme dans le cas des noms relatifs, on laisse des liens de poursuite dans les partitions pour chaque objet déplacé. Ainsi, la demande d'accès à un objet est

systématiquement envoyée à la partition de création, puis éventuellement transmise de lien de poursuite en lien de poursuite jusqu'à ce que l'objet soit trouvé.
Cette solution a l'inconvénient de pénaliser toutes les migrations sans ne jamais y remédier. On peut raccourcir les chaînes de liens de poursuite lorsqu'elles sont de longueur supérieure à un lien, mais on ne peut pas revenir au coût de localisation initial.

• *Le changement de nom :*

Cette solution consiste à ajouter à la précédente un changement de nom pour les objets qui sont déplacés. Lorsqu'un nom d'objet est stocké dans un objet et lorsque l'utilisation de ce nom montre que l'objet désigné a été déplacé, le nouveau nom de l'objet, incluant les nouvelles informations sur sa localisation, est placé dans l'objet contenant ce nom. Ainsi, les futures utilisations de ce nom bénéficieront d'une localisation directe.
Le problème posé par une telle solution est qu'un même objet peut être désigné par plusieurs noms (comme dans le cas des noms relatifs). Il faut alors être capable de comparer deux noms afin de savoir s'ils désignent le même objet. Ceci signifie que la comparaison de deux noms d'objet implique une résolution de ces noms, pour arriver aux localisations finales des objets et déterminer s'il s'agit du même objet.

• *Ajout d'informations sur la localisation courante dans les identificateurs d'objets :*

Le nom d'un objet peut être composé de deux champs : *identité* et *localisation.* Cette technique, qui est une variante de la précédente, consiste à augmenter la taille du nom d'un objet pour qu'il puisse contenir le nom qui lui est attribué à la création de l'objet et qui sert d'identificateur unique, ainsi que des informations sur la localisation courante. La première partie sert à identifier l'objet sans qu'un accès à cet objet soit nécessaire. La seconde partie du nom sert à localiser l'objet et peut être modifiée à chaque action de localisation, ce qui lui permet d'indiquer la dernière localisation connue. Des liens de poursuite sont toujours gérés pour les localisations dont la deuxième partie n'est pas à jour.
On évite ainsi de localiser des objets pour comparer leur identité, mais la taille du nom des objets peut devenir une contrainte très gênante.
Cette solution est utilisée dans la première version du système Guide [FRE 91], dans lequel un appel de méthode peut modifier une partie de la référence utilisée si l'objet a été déplacé.

• *Un ensemble de serveurs de migration :*

Cette solution revient à utiliser un ensemble de serveurs répartis coopérants qui enregistrent les déplacements des objets.
La localisation d'un objet commence par utiliser les informations présentes dans son nom ; si l'objet n'est pas trouvé dans la partition indiquée (la partition de création), un serveur de migration est appelé pour déterminer la partition contenant l'objet. L'avantage de cette solution, par rapport à la solution fondée sur des serveurs et où le nom est indépendant de la localisation, est d'optimiser l'accès aux objets qui n'ont pas été déplacés : le serveur de migration n'est contacté que pour les objets déplacés, le nombre d'objets concernés devant rester faible. Son avantage par rapport aux solutions fondées sur des liens de poursuite est d'éviter le parcours

d'une chaîne de liens de poursuite et de faire payer le même coût quel que soit le nombre de migrations d'un objet.
Une technique comparable est utilisée dans le système Gothic v2 [PUA 93]. Dans Gothic, lorsqu'une localisation est nécessaire, un message de localisation est envoyé au site de création de l'objet (donné par son nom). Si l'objet a été déplacé, une diffusion de message à tous les sites est utilisée.

3.3. *Résumé*

La gestion de la mémoire de stockage permet la désignation, la localisation et la migration des objets dans l'espace de stockage. Nous avons présenté les différentes approches qui se regroupent en deux classes :
• Celles qui gèrent des noms relatifs : dans ce cas, la localisation et la migration des objets est simplifiée, mais il faut alors résoudre les problèmes d'homonymes et de synonymes.
• Celles qui gèrent des noms absolus : dans ce cas, il est possible d'utiliser des noms indépendants de la localisation ; la localisation est alors assurée par un ensemble de serveurs coopérants. Il est également possible d'utiliser des noms dépendants de la localisation ; la localisation est plus directe, mais nécessite l'ajout d'un mécanisme pour localiser les objets déplacés.

4. Partage, protection et adressage en mémoire d'exécution

Une des motivations pour l'organisation de la mémoire d'exécution est la protection (prenant ici le sens d'isolation). Il est en effet nécessaire de confiner un objet ou une activité dans un espace afin, d'une part, de l'isoler des perturbations possibles provenant d'une erreur de comportement d'un autre (objet ou activité) et, d'autre part, de l'empêcher de nuire aux autres. L'outil de base pour garantir cette isolation est le domaine de protection (défini en section 2.4). Nous étudions donc dans la section 4.1 comment les domaines de protection peuvent être utilisés pour l'organisation de la mémoire d'exécution.

Une fois cette organisation décrite, nous étudions comment le partage d'objets peut être réalisé. Ce partage peut avoir lieu entre des activités pouvant s'exécuter dans des domaines de protection différents et pouvant être sur des machines différentes. La section 4.2 présente les différentes techniques.

Enfin, lorsqu'un objet doit être adressé à partir de son nom, deux phases doivent être mises en œuvre, la première permettant de rallier un domaine de protection dans lequel on peut l'adresser, en fonction des règles choisies pour le partage et la protection, et la deuxième permettant de récupérer l'adresse de l'objet dans l'espace virtuel dans lequel il est couplé. L'adressage des objets est abordé dans la section 4.3.

4.1. *Organisation de la mémoire d'exécution*

Pour étudier les différentes possibilités, nous partons d'une version sans protection, puis nous étudions les conséquences de l'introduction de règles de protection de plus en plus fines à l'aide de domaines de protection.

Version sans protection

La version minimale est obtenue en supprimant les problèmes liés à la protection. Dans ce cas, il suffit d'utiliser un seul domaine de protection. Les objets sont alors couplés à la demande dans ce domaine et toutes les activités de toutes les applications s'exécutent dans ce domaine.

Dans la réalité, étant donné qu'un domaine de protection est une structure locale à un site, il y a plutôt un domaine de protection par site dans le système et toutes les activités et tous les objets sur le même site s'exécutent dans le même domaine.

C'est le cas notamment dans le projet Emerald [BLA 86] réalisé sur le système Unix. Sur chaque site, un processus Unix contient le noyau d'Emerald ainsi que tous les objets en cours d'exécution sur ce site. L'ordonnancement des activités dans ce processus Unix est réalisé par le noyau d'Emerald et les activités sur un site partagent toutes le domaine de protection de ce processus. Dans ce cas, aucune protection physique entre les objets n'est fournie. La protection des objets ne peut être assurée que par les langages de programmation en supposant que toutes les applications sont écrites avec des langages sûrs[2].

Si la protection ne repose pas sur des contrôles effectués par les langages de programmation, elle ne peut être fournie que par la séparation entre les différents domaines de protection. Cette séparation des domaines peut être utilisée pour créer des cloisonnements entre les activités et entre les objets.

Séparation des activités

Une erreur dans l'exécution d'une application s'exécutant pour le compte d'un utilisateur ne doit pas perturber l'exécution d'autres applications. Cette propriété peut être plus ou moins respectée en fonction de l'organisation des structures d'exécution.

Ainsi, il ne faut pas que deux activités qui s'exécutent pour le compte d'applications différentes puissent se perturber. Une telle perturbation, si elle provient du partage d'objets entre les applications, ne peut pas être évitée. Mais il ne faut pas que cette perturbation vienne d'une source différente. Lorsque tous les objets et toutes les activités en mémoire d'exécution sur un site sont dans le même domaine de protection, une activité peut adresser par erreur un objet qu'elle ne partage pas (qu'elle n'utilise pas dans le cadre de son application), donc perturber une activité avec laquelle elle n'a rien en commun. L'objectif de la séparation des

---

[2] La sureté d'un langage se traduit par une confiance dans le code généré par le compilateur.

activités est donc de limiter les facultés d'adressage d'une activité au strict nécessaire. Cette propriété est généralement obtenue en isolant chaque activité dans un domaine de protection et en n'y couplant que les objets qu'elle utilise.

C'est le cas dans le projet Thor [LIS 92]. Dans Thor, un processus voulant utiliser un objet le charge dans le domaine de protection qui lui est associé. Ainsi, ce processus ne peut adresser que les objets effectivement chargés dans ce domaine.

L'isolation entre les activités procure un certain degré de protection, mais il est possible d'assurer un meilleur cloisonnement en isolant les objets entre eux.

Séparation des objets

Traditionnellement, à la notion d'objet est associée la notion d'encapsulation. Un objet est souvent considéré comme une boite noire dont seules les méthodes en permettent l'utilisation. En conséquence, la règle d'encapsulation des objets voudrait qu'un seul objet soit adressable à un instant donné par une activité exécutant une méthode. Une erreur d'adressage dans l'exécution d'une méthode ne pourrait atteindre que les données de cet objet.

Cette propriété peut être obtenue en faisant en sorte qu'il n'y ait qu'un seul objet par domaine, soit statiquement, soit dynamiquement :

• *Statiquement :*

A sa création, un domaine se voit attribuer un seul et unique objet. Ainsi, chaque objet se trouve isolé des autres objets et la seule façon d'adresser un objet est alors d'appeler une méthode de cet objet, en changeant par conséquent de domaine.

Cette technique est celle choisie dans le projet Argus [LIS 87] où les objets sont réalisés en mémoire d'exécution par des processus sur le système Unix. Chaque objet est dans un domaine privé et l'appel de méthode entre objets est réalisé par envoi de messages entre les processus contenant l'objet appelant et l'objet appelé. Un processus Unix est alors un serveur pour un objet unique.

• *Dynamiquement :*

Un seul objet est présent dans un domaine à un instant donné et il est possible de changer l'objet accessible dans un domaine. Cette technique est utilisée dans le projet Clouds [DAS 90]. Dans Clouds, un objet n'est couplé dans un domaine que pendant le temps d'exécution d'une méthode de cet objet. Chaque appel de méthode provoque une annulation du couplage de l'objet appelant et le couplage de l'objet appelé. Les objets sont alors isolés, puisque seul l'objet appelé est adressable pendant un appel de méthode.

Le problème de ces approches est la taille des objets : on ne peut réaliser de la sorte un petit objet sans gaspiller les ressources du système. De plus, chaque appel à un objet entraîne soit un changement de domaine, soit un couplage, c'est à dire une opération coûteuse. Pour éviter ce coût inacceptable à chaque appel, ces systèmes gèrent au niveau langage des objets de plus petite taille. Ces objets appelés objets locaux ne sont pas visibles de l'extérieur de l'objet global les contenant. Un appel entre objets locaux est réalisé par appel de procédure.

Dans les deux cas, la taille des objets comparée au coût de gestion des domaines de protection incite donc au regroupement d'objets et à assurer l'isolation pour un

ensemble d'objets. Différents critères de regroupement d'objets peuvent être utilisés et offrent différents types de protection. Nous en citons deux.

• Un premier exemple est le regroupement par propriétaire d'objet. On n'autorise le couplage de plusieurs objets dans le même domaine que s'ils appartiennent au même propriétaire. Ainsi, la protection ne peut être violée que par un objet du même utilisateur. C'est ce type d'isolation qui est assurée dans la seconde version du système Guide [HAG 93].

• Un autre exemple est le regroupement des objets utilisés dans le cadre de la même application. Ainsi, seuls les objets explicitement utilisés par l'application peuvent être adressés.

4.2. *Partage*

Nous avons présenté les différentes possibilités d'organisation de la mémoire d'exécution à partir de la notion de domaine de protection. On se pose maintenant le problème du partage d'objets entre activités en mémoire d'exécution.

Dans le partage d'objets entre activités, il faut distinguer le partage simultané où des activités partagent l'objet au même instant, du partage sérialisé ou l'objet est partagé par des activités sur des périodes ne se recouvrant pas.

Lorsque le partage est simultané entre différentes activités, il faut distinguer les cas où :

• Les activités sont dans le même domaine.

• Les activités sont dans des domaines différents sur la même machine.

• Les activités sont dans des domaines différents sur des machines différentes.

4.2.1. *Partage sérialisé*

Dans le cas du partage sérialisé d'un objet, une seule activité est autorisée à utiliser cet objet à un instant donné. Ainsi, il est possible de délivrer une copie de l'objet à l'activité le demandant.

Cette technique est notamment utilisée dans des systèmes transactionnels de gestion de base de données comme dans les projets Mneme [MSS 90] ou Thor [LIS 92]. Dans Thor, le verrouillage des objets par les transactions implique qu'un objet ne peut être chargé que dans un processus à la fois. Ce verrouillage est d'autant plus nécessaire que, chaque processus ayant un espace virtuel privé et les noms d'objets étant remplacés par des adresses virtuelles en mémoire d'exécution (*swizzling*), il serait difficile d'assurer la cohérence des adresses dans tous les processus partageant les objets (ce problème est traité plus en détail en section 4.3.1).

4.2.2. *Partage dans le même domaine*

Le partage d'objets entre activités dans le même domaine est automatiquement réalisé.

4.2.3. *Partage entre domaines différents sur la même machine*

Lorsque deux activités dans deux domaines différents doivent partager un objet, il est toujours possible de se ramener au cas précédent du partage dans le même domaine en utilisant l'opération de changement de domaine.

Pour des raisons d'isolation des applications entre elles, le modèle d'exécution du système peut cependant imposer le partage d'objets entre des activités dans des domaines différents. Le mécanisme de couplage permettant le partage simultané de mémoire entre domaines différents existe dans la plupart des systèmes. Il se distingue de l'opération de chargement par le fait que, si plusieurs domaines couplent le même objet pour le partager, ils travailleront sur le même objet et non sur des copies différentes.

Sur une même machine, le partage de mémoire entre domaines n'est pas un concept nouveau et il est fourni en mettant en correspondance des morceaux de tables des pages dans le noyau. Un exemple de partage de mémoire entre domaines de protection est le partage de mémoire entre processus sur le système Unix.

4.2.4. *Partage entre domaines différents sur des machines différentes*

Partage par migration

Ici aussi, il est possible de se ramener au cas précédent, en utilisant un mécanisme permettant de réaliser le partage effectif sur la même machine.

Le principe du partage par migration est de n'autoriser le partage que sur un seul site à la fois. Cette exclusion mutuelle entre les sites candidats à l'accès à l'objet est compensée par des mécanismes garantissant la disponibilité des objets :

• *La migration des activités :*

Une activité voulant accéder à un objet déjà couplé sur un autre site change de site d'exécution pour aller sur le site où est l'objet, le partage se faisant donc de façon locale. Une fois les deux activités sur le même site, le partage en local peut être réalisé avec les deux techniques données précédemment, c'est à dire en partageant dans le même domaine ou par couplage dans des domaines différents sur la même machine.

Les systèmes Guide v1 [BLT 91] et Amber [CHA 89] utilisent cette technique. Dans le système Guide v1 réalisé sur Unix, un objet en mémoire d'exécution ne peut être couplé que sur un site. Si une activité sur un autre site veut appeler cet objet, on utilise un mécanisme d'appel à distance qui transfère l'exécution de l'activité sur le site où l'objet est couplé.

• *La migration des objets :*

Une activité voulant accéder à un objet déjà couplé sur un autre site demande le découplage de l'objet sur ce site et le couplage sur son site. Le partage se fera toujours de façon locale mais, cette fois-ci, c'est l'objet qui va à l'activité. Les activités en cours d'exécution sur cet objet sont généralement déplacées avec l'objet. Le système Amber [CHA 89] permet également la migration des objets. Par défaut, une activité ne peut accéder à un objet couplé sur une autre machine qu'après

migration de l'activité. Cependant, il est possible de changer le site de couplage sur requête explicite de l'utilisateur.

C'est, en pratique, un compromis entre ces deux possibilités qui est employé en fonction du taux d'utilisation des objets sur les sites : si l'objet est très utilisé sur un site, la migration d'activité sera préférée à la migration d'objet. Les techniques de migration d'objets et d'activités permettent également de réguler la charge sur les stations de travail du réseau.

Partage par couplage et copies multiples

Pour le partage entre machines différentes, une autre technique consiste à étendre le mécanisme de couplage pour permettre la gestion de copies multiples sur les sites partageant l'objet. Le principal problème est alors de garantir la cohérence des données partagées, un objet pouvant être adressé simultanément sur des machines différentes. Différents niveaux de cohérence peuvent être offerts [MSB 93]. La cohérence la plus couramment réalisée est la cohérence du type lecteur/rédacteur strict : des lectures peuvent être effectuées simultanément sur plusieurs machines, mais toutes les écritures doivent être vues dans le même ordre sur toutes les machines. Il est toutefois possible d'offrir des cohérences moins restrictives.

La cohérence peut être assurée par va-et-vient ou par diffusion :

• *Cohérence par va-et-vient :*

Le partage avec cohérence par va-et-vient est caractérisé par un protocole qui détermine les conditions dans lesquelles des copies peuvent être données aux différentes machines et une opération de réquisition de copie qui permet de reprendre une copie donnée à un site, afin de rendre à nouveau cette copie disponible pour un autre site candidat au partage.

L'unité de partage, de copie et de réquisition peut être l'objet ou la page, ce qui signifie que la détection du type de l'accès (lecture ou écriture) peut être faite au niveau de l'objet ou de la page. Une tentative d'accès à un objet (resp. page) non présent sur le site provoque un défaut d'objet (resp. page). Un défaut est traité en donnant une copie de l'objet (resp. page) au site demandeur. Ce don peut nécessiter auparavant une réquisition sur un ou plusieurs autres sites.

Si l'unité de partage est l'objet, il faut détecter à la compilation pour chaque méthode si on fera référence à l'objet (potentiellement) en lecture ou en écriture. L'exécution d'une méthode vérifie alors la présence de l'objet dans le bon mode et provoque un défaut d'objet si ce n'est pas le cas.

Le maintien de la cohérence au niveau de la page lui est généralement préféré [LI 89], car il repose sur une détection matérielle du type de l'accès. Une page peut être délivrée à un site en mode écriture ou à plusieurs sites en mode lecture. Une tentative d'accès à une page dans un mode non autorisé provoque un défaut de page, pouvant provoquer la réquisition de cette page sur un autre site, pour traiter ce défaut. Cette technique ne peut être réalisée qu'à un niveau très proche de la machine (dans le gérant de pagination). L'arrivée des micro-noyaux permettant de réaliser le contrôle de la pagination dans un serveur externe au noyau a généralisé ce type d'approche.

On retrouve cette approche notamment dans les projets Clouds [DAS 90] et Opal [CHA 92].

• *Cohérence par diffusion :*

En cohérence par diffusion, des copies des données partagées résident sur toutes les machines les partageant et les modifications de ces données sont diffusées à tous les sites propriétaires d'une copie.

La diffusion doit garantir que les modifications sont faites dans le même ordre sur toutes les machines. Pour ce faire, un séquenceur doit être utilisé pour ordonner les messages. On distingue ici deux techniques. La première consiste à répéter l'exécution des opérations qui modifient l'objet sur les copies. La seconde consiste à diffuser les nouvelles valeurs après modification pour mettre à jour les copies.

La cohérence par diffusion est avantageuse lorsqu'il y a un faible taux d'écriture, les lectures pouvant se faire en parallèle sur les sites partageant les objets.

Elle est utilisée dans le projet Orca [BAL 92]. Les opérations se voient attribuer lors de la compilation un type qui est "lecture seule" ou "modification possible". Une opération de type "lecture seule" utilise directement sa copie locale, alors qu'une opération de type "modification possible" diffuse les paramètres de l'opération aux sites ayant une copie pour réexécution sur ce site. L'inconvénient de cette solution est qu'une analyse statique des programmes risque d'attribuer le type "modification possible" à la plupart des opérations.

4.3. *Adressage des objets*

Nous avons présenté ci-dessus les possibilités d'organisation de la mémoire d'exécution, puis les techniques de mise en œuvre du partage dans cette mémoire. Nous allons maintenant étudier les principes de réalisation de la résolution des noms d'objets en mémoire d'exécution.

A chaque appel de méthode sur un objet, il faut être en mesure d'adresser les données de l'objet. Il faut donc que l'activité courante s'exécute dans un domaine de protection autorisé à coupler l'objet et que l'objet appelé y soit effectivement couplé. Il faut également que l'objet soit lié dans un espace virtuel pour être adressable par une adresse virtuelle et avoir déterminé à partir du nom de l'objet l'adresse virtuelle qui lui est associée.

Une première phase consiste donc à faire en sorte que l'objet soit accessible, c'est à dire à réunir l'objet et l'activité dans le même domaine de protection. Cette phase peut se traduire :

• par le couplage de l'objet appelé dans le domaine d'exécution de l'activité,

• ou par une migration de l'activité vers un autre domaine, pour permettre le partage, pour des raisons de protection, ou pour les deux raisons.

Pour cette première phase appelée défaut d'objet, il est nécessaire de gérer un contexte global de résolution de noms indiquant soit le couplage nécessaire, soit le domaine de protection dans lequel s'exécuter pour que l'objet soit accessible. Cette résolution de nom est généralement interprétée (il s'agit d'une gestion de table de

traduction entre le nom de l'objet et l'information recherchée). Le coût de cette première phase est lié à la technique utilisée pour réaliser le partage et au degré de protection fourni. Elle aura lieu à chaque appel d'objet si chaque objet est confiné dans un domaine privé.

Dans une seconde phase, l'objet étant accessible dans le domaine courant, le problème est d'obtenir une adresse virtuelle à partir du nom de l'objet. Il est alors nécessaire de gérer un ou plusieurs contextes de résolution de noms associant une adresse virtuelle à un nom d'objet, ce qui revient à gérer un ou plusieurs espaces virtuels. Selon le nombre des espaces virtuels et la nature statique ou dynamique de la liaison dans ces espaces virtuels, les cas envisageables sont les suivants :

- liaison dynamique dans plusieurs espaces virtuels,
- liaison dynamique dans un espace virtuel unique,
- liaison statique dans plusieurs espaces virtuels,
- liaison statique dans un espace virtuel unique.

4.3.1. *Liaison dynamique dans plusieurs espaces virtuels*

Les objets sont liés dynamiquement (à l'exécution) dans différents espaces virtuels et peuvent donc l'être à des adresses virtuelles différentes (sinon, on a un espace virtuel unique). Un contexte de résolution de noms est associé à chaque espace virtuel et donne pour chaque nom d'objet son adresse de liaison dans cet espace virtuel. La gestion du contexte pour la résolution des noms prend généralement la forme d'une table dont la fonction d'accès est fonction du nom de l'objet (généralement un adressage dispersé ou *hash-code*).

Des raccourcis d'adressage sont en général employés, afin d'éviter une résolution de nom systématique (aussi appelée *interprétation*). Ces raccourcis sont différents dans les cas où un objet n'est jamais lié simultanément dans plusieurs espaces virtuels (liaison sérialisée) et ceux où un objet peut être lié dans plusieurs espaces virtuels au même instant (liaison simultanée) :

- *Liaison sérialisée :*

Lorsqu'un nom d'objet doit être résolu, ce nom d'objet est contenu dans une variable qui peut être une variable temporaire sur la pile ou une variable d'état d'un objet. Le principe du raccourci consiste à modifier cette variable et à l'enrichir d'une information permettant de résoudre plus rapidement le nom qu'elle contient lors des prochaines résolutions. Un raccourci très répandu dans le domaine, appelé *mutation de pointeur* (*swizzling*), consiste à remplacer le nom de l'objet par son adresse virtuelle de liaison. Ainsi, les utilisations suivantes de cette variable trouveront l'adresse de liaison de l'objet à la place du nom.

Pour utiliser la mutation de pointeur, deux problèmes doivent être résolus : il faut détecter lors de l'utilisation d'un nom s'il a déjà été changé en adresse et il faut lors de la déliaison d'un objet remplacer les adresses qu'il contient par les noms des objets qu'elles désignent.

On distingue les réalisations suivantes de la mutation [MOS 92] :

1) La mutation paresseuse (Lazy swizzling). Un nom d'objet est transformé à sa première utilisation. Une information dans ce nom permet de détecter si le nom est déjà transformé. A l'exécution, un test doit être fait avant chaque utilisation de nom.
2) La mutation anticipée (*Eager swizzling*). Lorsqu'un objet est lié dans l'espace virtuel (lors du premier accès), chaque nom dans cet objet est transformé en adresse. Cette adresse est l'adresse de l'objet désigné s'il est résident ; autrement, c'est l'adresse d'un objet intermédiaire. L'appel à un objet intermédiaire provoque un défaut d'objet qui lie l'objet et le substitue à l'objet intermédiaire. Cette solution est utilisée dans Thor [LIS 92].
La mutation de pointeur est une technique très utilisée dans le domaine des bases de données. Elle repose en général sur une sérialisation des couplages dans les domaines de protection et des liaisons dans les espaces virtuels, la sérialisation étant obtenue par les verrouillages effectués par des transactions.
• *Liaison simultanée :*
La mutation de pointeur ne peut pas être utilisée en cas de liaison simultanée, puisqu'un objet peut être lié à des adresses virtuelles différentes. De même, il est nécessaire que le code exécuté soit indépendant de sa localisation dans l'espace virtuel où il est lié.
Il est tout de même possible de mettre en place un raccourci en associant une zone locale de l'espace virtuel courant à la variable contenant le nom de l'objet. L'association de cette zone et de la variable est gérée statiquement par le compilateur qui réalise toute utilisation de la variable par indirection à travers cette zone locale. La mutation de pointeur est réalisée sur cette zone locale et on accélère ainsi fortement la résolution de nom si cette zone est à jour.
Cette technique a été introduite dans le système Multics [DAL 68], afin de permettre le partage de segments à des adresses virtuelles différentes. A chaque segment de Multics est associé, dans tout espace virtuel le partageant, une table appelée segment de liaison, et à chaque référence externe dans ce segment est associée une entrée de cette table. Cette entrée, qui est mise à jour au premier accès, permet de mémoriser dans l'espace virtuel courant l'adresse de résolution du nom du segment référencé. Cette méthode est aussi utilisée dans le système Guide v2 [CHE 93] qui simule une machine segmentée à la Multics.
Un autre exemple nous est donné par le projet E [SCH 90], dans lequel l'utilisation d'un nom d'objet ne peut se faire que par l'utilisation d'une variable locale.Une telle variable étant créée sur la pile, elle est par conséquent locale à l'espace virtuel courant et peut être remplacée (mutation de pointeur) par l'adresse de liaison de l'objet dans l'espace virtuel courant.

4.3.2. *Liaison dynamique dans un espace virtuel unique*

Chaque objet est lié dans un espace virtuel unique (indépendamment de la gestion des domaines de protection). Un seul contexte de résolution de noms d'objets est géré pour tout le système. La technique de raccourci appelée mutation de pointeur décrite en 4.3.1 est également applicable dans ce cas, puisque chaque objet se voit associer une seule adresse virtuelle.

Une variante de cette solution est mise en œuvre dans le système COOL [LEA 93] réalisé sur le micro-noyau Chorus [ROZ 92]. COOL gère la liaison des objets à la même adresse dans tous les acteurs (processus de Chorus) les partageant. Cette adresse est allouée dynamiquement et la technique de mutation anticipée de pointeur est utilisée.

4.3.3. *Liaison statique dans plusieurs espaces virtuels*

Les objets sont liés statiquement (lors de leur création) dans plusieurs espaces virtuels. En général, la liaison statique implique qu'un objet ne sera lié que dans un seul espace, ce qui est indépendant du fait que plusieurs espaces virtuels soient gérés. On peut alors distinguer deux cas :

• *Un seul objet par espace virtuel :*

Un espace virtuel différent est associé à chaque objet. L'adresse de liaison d'un objet peut être la même quels que soient les objets. Le contexte de résolution de noms est implicite, puisque tout objet est accessible à la même adresse.

Cette technique est utilisée dans le système Argus [LIS 87] dans lequel chaque objet en mémoire d'exécution est couplé dans le domaine de protection et lié dans l'espace virtuel privés d'un processus Unix ; chaque appel à un objet nécessite une migration de l'activité vers le domaine et l'espace virtuel contenant l'objet appelé.

Elle est également utilisée dans le projet Clouds [DAS 90].

Dans les deux cas, les objets sont de grosse taille ; l'appel de méthode sur un objet est une opération lourde et coûteuse. C'est pourquoi ces systèmes introduisent en plus des objets de plus petite taille, ce qui nous amène à l'autre solution.

• *Plusieurs objets par espace virtuel :*

Dans ce cas, les objets sont regroupés dans des ensembles logiques. Un espace virtuel est associé à chacun de ces ensembles et à chaque objet de l'ensemble est attribuée statiquement une adresse virtuelle. Un domaine utilisé pour adresser un tel regroupement d'objets est généralement appelé un *serveur d'objets* . Un contexte est associé à ce regroupement et donne pour chaque objet l'adresse qui lui est associée de façon statique.

4.3.4. *Liaison statique dans un espace virtuel unique*

Dans ce cas, on gère un espace virtuel unique dans le système tout entier. L'adresse associée à l'objet lors de sa création est généralement utilisée comme nom de l'objet. Aucun contexte de résolution de nom n'est à gérer, le nom d'un objet donnant directement l'adresse de liaison de l'objet dans cet espace virtuel unique.

On suppose alors que tous les objets du système peuvent être contenus dans l'espace virtuel.

Cette technique est utilisée dans les projets Amber [CHA 89] et Opal [CHA 92]. Dans le système Opal, un objet est désigné quel que soit le domaine d'exécution par l'adresse virtuelle que constitue son nom. Un objet peut être couplé dans un domaine de protection sur demande explicite, ou implicitement par le système à la suite d'un adressage invalide.

Cette technique évite ainsi l'opération coûteuse de résolution de nom qui a lieu lorsque des espaces virtuels différents sont gérés. Elle évite également des indirections d'adressage par rapport à des variables locales.

Cette solution est d'autant plus crédible que des processeurs dont les espaces virtuels sont adressés par des adresses de 64 bits arrivent sur le marché.

4.4. *Résumé*

La gestion de la mémoire d'exécution met en œuvre la protection, le partage et l'adressage des objets à l'exécution :

• La protection est assurée à l'aide de domaines de protection qui peuvent être utilisés pour assurer la séparation entre les activités ou entre les objets.

• Le partage des objets, que ce soit entre des domaines sur la même machine ou sur des machines différentes, peut être mis en œuvre soit en réunissant les activités (dans le même domaine ou sur la même machine), soit par couplage (local à une machine ou par copies multiples en réparti).

• L'adressage des objets peut être géré en associant aux objets une adresse virtuelle de façon statique (à la création) ou dynamique (à l'exécution). L'allocation des adresses virtuelles peut se faire soit dans un espace virtuel unique, ce qui permet de partager les adresses entre tous les domaines de protection, soit dans différents espaces virtuels, ce qui évite de coordonner l'allocation de ces adresses.

5. Récapitulatif

Nous regroupons ci-après les principales caractéristiques des systèmes que nous avons retenus pour illustrer notre étude.

5.1. ***Mémoire de stockage***

	Désignation	**Localisation**
Thor/Mneme	Noms relatifs	Locale partition et liens de poursuite
Amoeba	Noms globaux (indépendant localisation)	Diffusion + caches
Grapevine	Noms globaux (indépendant localisation)	Catalogues dupliqués
Gothic v2	Noms globaux (dépendant localisation)	Serveurs de migration
Guide v1	Noms globaux (dépendant localisation)	Nom contenant localisation courante

5.2. ***Mémoire d'exécution***

	Taille objets	**Isolation**	**Partage**	**Défaut d'objet**	**Adressage**
Argus	Gros	Objet	Migration	A chaque appel	Adresse constante
Clouds	Gros	Objet	Couplage (va et vient et migration)	A chaque appel	Adresse constante
Emerald	Petits	Non	Migration	Au premier appel et au changement de site	Adresse statiquement associée à l'objet
Opal	Petits	Groupement d'objet	Couplage (va et vient et migration)	Au premier appel et au changement de site	Adresse statiquement associée à l'objet
Gothic v2	Gros	Non	Couplage (va et vient et migration)	Au premier appel et au changement de site	Adresse statiquement associée à l'objet
Guide v1	Petits	Non	Couplage sur le même site, Migration entre sites différents	Au premier appel et au changement de site	Adresse allouée dynamiquement, Interprété

Guide v2	Petits	Groupement d'objet	Couplage (va et vient et migration)	Au premier appel et au changement de domaine	Adresse allouée dynamiquement, raccourci à la Multics

6. Conclusion

Nous avons passé en revue dans cet article l'ensemble des techniques utilisées pour gérer la mémoire de stockage (problèmes de localisation et de migration) et la mémoire d'exécution (problèmes de protection, de partage et d'adressage) dans les systèmes répartis à objets. En pratique, les choix de ces techniques ne sont pas indépendants et des compromis doivent être réalisés. En guise de conclusion, nous allons donc synthétiser les choix les plus fréquents pour trois classes de systèmes avant de présenter quelques perspectives de recherches.

6.1. *Synthèse*

Les systèmes répartis à objets disponibles à l'heure actuelle peuvent être regroupés en trois classes : les systèmes de gestion de bases de données, les systèmes clients-serveurs et les systèmes à partage concurrent de petits objets. Nous allons passer en revue les principales caractéristiques de ces trois classes et donner quelques exemples représentatifs de chacune.

Les systèmes de gestion de bases de données

Lorsqu'un système est amené à gérer la persistance d'un très grand nombre d'objets, ces objets contenant de nombreuses références des uns aux autres, la réduction de la place occupée par ces objets sur les disques est un objectif très important. On utilise alors la gestion de noms relatifs dans les partitions de la mémoire de stockage. Cette solution est généralement adoptée pour des systèmes de gestion de base de données, car elle permet un gain appréciable de place. En général, les bases de données permettent l'accès aux objets par l'intermédiaire de transactions qui sérialisent le partage. La sérialisation du partage permet d'utiliser la mutation de pointeur (*swizzling*), qui serait difficilement applicable avec du partage concurrent. La mutation de pointeur permet alors de distinguer des objets différents ayant le même nom relatif, mais également d'accélérer l'adressage des objets.

Des exemples sont notamment les systèmes Thor [LIS 92] et Mneme [MOS 90].

Les systèmes clients-serveurs

Les premières tentatives de réalisation d'un système supportant des objets partagés persistants (avec partage concurrent) ont directement associé un espace virtuel propre à chaque objet en mémoire d'exécution. Compte tenu du coût de résolution des noms, les usagers de ces systèmes ont été amenés à distinguer deux

types d'objets : les objets globaux fournis par le système et les objets locaux gérés par les langages de programmation. Les objets globaux sont alors des serveurs pour les objets locaux qu'ils contiennent et des clients potentiels pour les autres objets globaux. L'isolation est assurée entre les objets globaux et le partage est réalisé par migration des activités clientes vers l'espace virtuel associé à l'objet global.

Des exemples sont les systèmes Argus [LIS 85] et Clouds [DAS 90].

Les systèmes à partage concurrent de petits objets

Le principal inconvénient des systèmes clients-serveurs est de ne pas offrir un espace de désignation uniforme pour de petits objets, les objets locaux n'étant pas visibles de l'extérieur de l'objet global les contenant. Les systèmes actuellement développés tentent donc de fournir un support efficace pour de petits objets. Ces objets sont tous désignés de façon uniforme, même s'ils sont parfois regroupés pour en améliorer la gestion. L'isolation est soit assurée au niveau du langage (Guide v1 [KRA 90], Emerald [HUT 87]), soit assurée par l'utilisation de domaines de protection (Guide v2 [HAG93], Opal [CHA 92]). Les systèmes les plus récents proposent à la fois le partage par migration et le partage par couplage en réparti. Le problème est alors d'utiliser lors de l'exécution la méthode la plus efficace.

6.2. *Perspectives*

Malgré l'évolution technologique rapide et continue des matériels informatiques, la technologie des systèmes d'exploitation n'a pas connu de rupture depuis une vingtaine d'années. Les mutations actuelles concernant les processeurs à large capacité d'adressage ainsi que l'évolution des réseaux devraient amener des changements plus profonds.

Le premier facteur de changement est la disponibilité des processeurs 64 bits. Les solutions fondées sur la gestion d'un espace virtuel unique, qui n'étaient pas réalistes sur un processeur 32 bits lorsque l'on veut gérer un grand nombre d'objets, deviennent réellement utilisables lorsque les machines disposent d'un adressage étendu. On peut en particulier utiliser l'adresse virtuelle comme référence universelle à un objet (suppression des mutations de pointeurs ou des indirections) et éviter la réutilisation des adresses. Ces questions sont explorées dans de nombreux projets, mais plusieurs problèmes restent à résoudre, notamment la protection (l'isolation et le contrôle d'accès) dans ce type de système. Les questions d'efficacité et de facteur d'échelle sont complètement ouvertes.

Le deuxième facteur d'évolution est l'arrivée des réseaux très rapides : d'une part les vitesses de transmission seront du même ordre que les échanges processeurs mémoire (lorsque le temps de latence peut être négligé, c'est à dire pour des volumes échangés importants, l'ensemble des informations devient "immédiatement" disponible) ; d'autre part, des cadences de transfert élevées seront disponibles à grande distance. L'impact sera sensible notamment sur les points suivants :

• La nature des données partagées risque d'être affectée. Le partage d'information sous forme d'images et de son, autrefois limité par la lenteur et le faible débit des réseaux, va se répandre. Les systèmes futurs devront être capables de gérer des objets de tailles très diverses, et également permettre d'adapter la gestion des objets en fonction du type de ces objets et du comportement des applications.
• Les différences entre les systèmes centralisés, répartis ou parallèles vont s'estomper et une synthèse devient envisageable.
• La taille des réseaux va également être affectée. Il faut envisager des applications impliquant des objets répartis dans le monde entier (avec de gros problèmes de désignation en perspective).

Certains de ces impacts sont déjà visibles sous la forme d'applications réparties sur le réseau Internet. Des applications comme le World-Wide-Web offrent aujourd'hui accès à une quantité énorme d'information de type texte, image, video et son, répartie à travers le monde.

7. Bibliographie

[ATK 83] M.P. ATKINSON, P.J. BAILEY, K.J. CHISHOLM, P.W. COCKSHOTT et R. MORRISON, "An approach to persistent programming", The Computer Journal, 26(4), pp. 360-365, Novembre 1983.

[BAL 92] H.E. BAL, M.F. KAASHOEK et A.S. TANENBAUM, "Orca: A Language for Parallel Programming of Distributed Systems", *IEEE Transactions on Software Engineering*, 18(3), pp. 190-205, Mars 1992.

[BLT 91] R. BALTER, J. BERNADAT, D. DECOUCHANT, A. DUDA, A. FREYSSINET, S. KRAKOWIAK, M. MEYSEMBOURG, P. LE DOT, H. NGUYEN VAN, E. PAIRE, M. RIVEILL, C. ROISIN, X. ROUSSET DE PINA, R. SCIOVILLE et G. VANDÔME, "Architecture and implementation of Guide, an object-oriented distributed system", *Computing Systems*, 4(1), pp. 31-67, Hiver 1991.

[BIR 82] A.D. BIRRELL, R. LEVIN, R.M. NEEDHAM et M.D. SCHROEDER, "Grapevine: An Exercise in Distributed Computing", *Communications of the ACM*, 25(4), pp. 260-274, Avril 1982.

[BLA 86] A.P. BLACK, N. HUTCHINSON, E. JUL et H. LEVY, "Object structure in the Emerald system", *Proceedings of the 1st ACM Conference on Object-Oriented Systems, Languages and Applications (OOPSLA)*, pp. 78-86, Septembre 1986.

[CHA 89] J.S. CHASE, F.G. AMADOR, E.D. LAZOWSKA, H.M. LEVY et R.J. LITTLEFIELD, The Amber System: Parallel Programming on a Network of Multiprocessors, (TR 89-04-01), Department of Computer Science and Engineering, University of Washington, Seattles, Avril 1989.

[CHA 92] J.S. CHASE, H.M. LEVY, E.D. LAZOWSKA et M. BAKER-HARVEY, "Lightweight Shared Objects in a 64-Bit Operating System", *Proceedings of the 7th ACM Conference on Object-Oriented Systems, Languages and Applications (OOPSLA)*, 27(10), pp. 397-413, Octobre 1992.

[CHE 93] P.Y. CHEVALIER, A. FREYSSINET, D. HAGIMONT, S. KRAKOWIAK, S. LACOURTE et X. ROUSSET DE PINA, "Experience with Shared Object Support in the Guide System", *Proceedings of the 4th Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS)* , pp. 157-173, Septembre 1993.

[CHI 91] R. CHIN, S. CHANSON, "Distributed Object-Based Programming Systems", *ACM Computing Surveys*, 23(1), pp. 91-124, Mars 1991.

[DAL 68] R.C. DALEY et J.B. DENNIS, "Virtual Memory, Processes, and Sharing in MULTICS", *Communications of the ACM*, 11(4), pp. 308-318, Mai 1968.

[DAS 90] P. DASGUPTA, R.C. CHEN, S. MENON, M.P. PEARSON, R. ANANTHANARAYANAN, U. RAMACHANDRAN, M. AHAMAD, R.J. LEBLANC, W.F. APPELBE, J.M. BERNABEU-AUBAN, P.W. HUTTO, M.Y.A. KHALIDI et C.J. WILKENLOH, "The Design and Implementation of the Clouds Distributed Operating System", *Computing Systems*, 3(1), pp. 11-45, Hiver 1990.

[FRE 91] A. FREYSSINET, *Architecture et Réalisation d'un Système Réparti à Objets*, Thèse de doctorat, Université Joseph Fourier, Juillet 1991.

[HAG 93] D. HAGIMONT, *Adressage et protection dans un système réparti*, Thèse de Doctorat, Institut National Polytechnique de Grenoble, Octobre 1993.

[HUT 87] N.C. HUTCHINSON, *Emerald: An Object-Based Language for Distributed Programming*, PhD thesis, University of Washington, Janvier 1987.

[KRA 90] S. KRAKOWIAK, M. MEYSEMBOURG, H. NGUYEN VAN, M. RIVEILL, C. ROISIN et X. ROUSSET DE PINA, "Design and implementation of an object-oriented strongly typed language for distributed applications", *Journal of Object-Oriented Programming (JOOP)*, 3(3), pp. 11-22, Octobre 1990.

[LEA 93] R. LEA, C. JACQUEMOT et E. PILLEVESSE, "COOL: system support for distributed object-oriented programming", *Communications of the ACM, Special issue on Concurrent Object Oriented Programming*, 36(9), pp. 37-46, September 1993.

[LEG 88] J. LEGATHEAUX MARTINS et Y. BERBERS, "La désignation dans les systèmes d'exploitation répartis", *Technique et Science Informatique*, 7(4), pp. 359-372, 1988.

[LEV 90] E. LEVY et A. SILBERSCHATZ, "Distributed File Systems: Concepts and examples", *ACM Computing Surveys*, 22(4), pp. 321-374, Decembre1990.

[LI 89] K. LI et P. HUDAK, "Memory Coherence in Shared Virtual Memory Systems", *ACM Transactions on Computer Systems*, 7(4), pp. 321-359, Novembre 1989.

[LIS 85] B.H. LISKOV, *The Argus language and system, Distributed systems: methods and tools for specification*, Lecture Notes in Computer Science, Vol. 190, Springer-Verlag, pp. 343-430, 1985.

[LIS 87] B.H. LISKOV, D. CURTIS, P. JOHNSON et R. SCHEIFLER, "Implementation of Argus", *Proceedings of the 11th ACM Symposium on Operating System Principles, SIGOPS Operating System Review*, 21(5), pp. 111-122, Novembre 1987.

[LIS 92] B. LISKOV, *Preliminary design of the Thor object-oriented database system*, (Programming Methodology Group Memo 74), Laboratory of Computer Science, MIT, 1992.

[MSB 93] MOSBERGER, "Memory Consistency Models", *Operating Systems Review*, 27(1), pp. 18-26, Janvier 1993.

[MOS 90] J.E.B. MOSS, "Design of the Mneme Persistent Object Store", *ACM Transactions on Information Systems*, 8(2), pp. 103-139, Avril 1990.

[MOS 92] J.E.B. MOSS, "Working with Persistent Objects: To Swizzle or Not to Swizzle", *IEEE Transactions on Software Engineering*, 18(8), pp. 657-673, Août 1992.

[ORG 72] E.I. ORGANICK, *The Multics system: an examination of its structure*, MIT Press, 1972.

[POW 83] M. POWELL et B. MILLER, "Process Migration in DEMOS/MP", *Proceedings of the 8th ACM Symposium on Operating System Principles, SIGOPS Operating System Review*, 17(5), pp. 110-119, Octobre 1983.

[PUA 93] I. PUAUT, *Gestion d'objets actifs dans les systèmes distribués : problématique et mise en œuvre*, Thèse de doctorat, Université de Rennes I, Janvier 1993.

[ROZ 88] M. ROZIER, V. ABRASSIMOV, F. ARMAND, J. BOULE, M. GIEN, M. GUILLEMONT, F. HERRMANN, C. KAISER, P. LEONARD, S. LANGLOIS et V. NEUHAUSER, "Chorus Distributed Operating System", *Computing Systems*, 1(4), pp. 305-370, 1988.

[SAL 78] J.H. SALTZER, *Naming and Binding of objects, in Operating Systems - an advanced course*, Lecture Notes in Computer Science, Vol. 60, Springer-Verlag, pp. 99-208, 1978.

[SCH 90] D. SCHUH, M. CAREY et D. DEWITT, "Persistence in E Revisited - Implementation Experiences", *Proceedings of the 4th International Workshop on Persistent Objects Systems*, pp. 345-359, Septembre 1990.

[TAN 90] A.S. TANENBAUM, R. VAN RENESSE, H. VAN STAVEREN, G.J. SHARP, S.J. MULLENDER, J. JANSEN et G. VAN ROSSUM, "Experiences with the Amoeba distributed operating system", *Communications of the ACM*, 33(12), pp. 46-64, Decembre 1990.

[WUL 81] W.A. WULF, R. LEVIN et S.P. HARBISON, *Hydra/C.mmp: An Experimental Computer System*, Mc Graw-Hill, New York, 1981.

Daniel Hagimont à soutenu en 1993 une thèse de doctorat à l'Institut National Polytechnique de Grenoble. Après un séjour d'une année à l'Université de Colombie Britannique (Vancouver), il occupe actuellement les fonctions de chargé de recherches à l'unité Rhône-Alpes de l'INRIA.

Jacques Mossière est Professeur à l'Institut National Polytechnique de Grenoble (ENSIMAG) où il enseigne l'algorithmique et les systèmes d'exploitation. Après une phase consacrée essentiellement à l'organisation de la recherche et de l'enseignement, une année sabbatique lui a permis de reprendre une activité de recherche en systèmes. Il met ensuite en place le laboratoire Logiciel, Systèmes et Réseaux de l'IMAG.