Accès à l'information répartie : adressage et protection

Daniel Hagimont

Accès à l'information répartie : adressage et protection

Daniel HAGIMONT

Rapport scientifique présenté pour l'obtention de l'
HABILITATION À DIRIGER DES RECHERCHES EN INFORMATIQUE

Institut National Polytechnique de Grenoble

Soutenue de 21 avril 1998 devant le jury composé de :

George Coulouris	(Rapporteur)
Jacques Mossière	(Rapporteur)
Claude Kaiser	(Rapporteur)
Jean-Pierre Verjus	(Président)
Sacha Krakowiak	(Examinateur)
Guy Mazaré	(Examinateur)

Je tiens à remercier les membres du jury :

Monsieur George Coulouris, Professeur au Queen Mary and Westfield College de Londres, et Monsieur Clause Kaiser, Professeur au Conservatoire National des Arts et Métiers, qui ont accepté d'être rapporteurs de mon habilitation,

Monsieur Jacques Mossière, Professeur à l'Institut National Polytechnique de Grenoble, qui a également accepté d'être rapporteur de mon habilitation, mais surtout qui m'a toujours apporté les remarques judicieuses permettant de progresser dans mes travaux de recherche et en particulier dans la rédaction de mon rapport d'habilitation.

Monsieur Jean-Pierre Verjus, Professeur à l'Institut National Polytechnique de Grenoble, pour avoir accepté de présider mon jury.

Monsieur Sacha Krakowiak, Professeur à l'Université Joseph Fourier, et Monsieur Guy Mazaré, qui ont accepté de faire partie de ce jury.

Je tiens également à remercier tous mes compagnons de route, de l'Unité Mixte Bull-IMAG, de l'Université de Colombie Britannique et de l'INRIA Rhône Alpes.

à Eloise ...

Résumé

L'accès et le contrôle d'accès à l'information réparties constituent un domaine fondamental de l'informatique répartie. En effet, la désignation des objets qui encapsulent l'information, la localisation de ces objets, leur partage entre les usagers autorisés, leurs accès et les contrôles de ces accès déterminent les structures de base de toute architecture de système réparti.

Cette Habilitation à Diriger les Recherches résume un ensemble de travaux que j'ai essentiellement réalisés dans le cadre de deux projets de recherche : le projet GUIDE mené à l'Unité Mixte Bull-IMAG de 1990 à 1994, et le projet SIRAC qui a débuté en 1995 et qui se poursuit actuellement. A travers ces projets, j'ai été amené à étudier les problèmes d'accès et de contrôle d'accès dans des environnements très différents, à savoir un système à objets répartis (Guide), une mémoire virtuelle répartie (Arias) et plus récemment un environnement Java réparti sur l'Internet. A partir d'un bilan des travaux effectués et des résultats obtenus, je dresse les perspectives de recherches pour les années à venir.

1. Introduction

1.1. Evolution

Les années 80 ont vu la généralisation de la téléinformatique et des réseaux locaux. Les réseaux locaux ont permis de substituer aux gros serveurs centralisés des configurations matérielles réparties composées de stations de travail interconnectées. Les avantages d'utiliser une architecture répartie sont nombreux. Ils sont principalement économiques. La mise à jour du matériel peut se faire graduellement (par remplacement partiel des machines du réseau) et il est possible d'augmenter à tout moment la puissance du système global par ajout de machines sur le réseau local. Notons enfin qu'un réseau local permet d'interconnecter des machines hétérogènes, permettant d'éviter une dépendance trop forte envers un constructeur.

Cette évolution des environnements matériels s'est accompagnée d'une évolution des systèmes d'exploitation s'exécutant sur les machines. Une première étape dans l'intégration de la répartition a consisté à ajouter aux systèmes existants des primitives d'envoi et de réception de message entre les machines, puis un mécanisme d'appel de procédure à distance. Dans une seconde étape, des environnements répartis plus intégrés ont été proposés parmi lesquels les systèmes de fichiers répartis et les systèmes à objets répartis. La caractéristique principale de ces environnements plus intégrés est de fournir un espace uniforme d'objets ou de fichiers partagés entre les machines d'un réseau local.

Les années 90 ont été marquées par deux évolutions techniques principales :

- Les réseaux rapides. Des réseaux rapides avec des débits de l'ordre du Gigabit par seconde sont apparus. Même s'il ne sont pas encore déployés à grande échelle, ils équipent aujourd'hui des réseaux locaux.
- Les réseaux grandes distances. Les réseaux grandes distances existent déjà depuis longtemps, mais ils se sont développés et surtout sont devenus accessibles au grand public avec l'Internet et le World Wide Web.

Plus particulièrement, ces deux évolutions ont donné naissance à deux thèmes de recherche dans le domaine des systèmes répartis :

- Gérer un réseau local de machines comme un multiprocesseur virtuel. Avec un réseau très efficace, il devient raisonnable de faire une analogie entre un réseau local interconnectant des stations de travail et un bus de communication interconnectant les processeurs d'une machine multiprocesseur. Ceci milite pour une gestion globale des ressources sur le réseau de machines, une machine pouvant notamment utiliser la mémoire ou le temps processeur d'une autre machine lorsqu'elle est surchargée.
- Gérer des applications réparties sur un réseau général. Si l'Internet est devenu un moyen de communication privilégié, les outils pour la construction d'applications réparties sur l'Internet sont mal adaptés. De plus, les contraintes ne sont pas les mêmes que dans un réseau local. Par exemple, on ne peut faire aucune supposition sur les

types des machines ou des systèmes visités (c'est en partie ce qui à fait émerger Java). De même, la connectivité sur l'Internet est difficilement contrôlable.

Pour résumer, nous sommes passés des architectures du type *réseau local* des années 80 à des architectures de type *cluster* (multiprocesseur virtuel) avec une gestion des ressources très intégrée, et à des réseaux grandes distances de type *Internet* interconnectant des machines réparties dans le monde entier.

1.2. Principaux problèmes

Dans le cadre de cette évolution des systèmes répartis, de nombreux problèmes doivent être pris en compte dont voici les principaux.

Un système informatique réparti doit tout d'abord fournir des méthodes d'accès à des informations pouvant être réparties sur l'ensemble des machines gérées. Ces méthodes doivent prendre en compte les schémas de nommage des objets manipulés, être en mesure de localiser les objets à partir de leur nom, puis fournir un moyen d'accès proprement dit permettant de lire ou modifier l'objet. Le nommage des objets et l'accès aux objets sont des problèmes particulièrement délicats dans le contexte d'un système réparti. Par exemple, dans un système réparti de grande taille, la quantité d'information gérée est bien supérieure à celle d'une machine centralisée et il devient difficile de borner le temps de communication entre deux machines. Par contre, sur un réseau local très rapide, les coûts d'accès à l'information répartie sont moins élevés et conduisent à des choix très différents

Un deuxième problème très important de l'informatique répartie est le partage de l'information. L'informatique répartie est devenue un moyen de communication privilégié entre les usagers et le partage de donnée est l'outil de base permettant de construire des applications visant à faire coopérer plusieurs usagers du système. Gérer le partage de données, c'est gérer la concurrence des accès aux données, mais également gérer la cohérence des données lorsque des copies sont réparties sur les sites y accédant. Ici aussi, les solutions mises en œuvre doivent prendre en compte les coûts inhérents à la répartition.

Le partage de données entre plusieurs usagers impose la mise en place de mécanismes de protection permettant le contrôle de l'accès aux données. En général, les usagers du système doivent être authentifiés et le système doit fournir aux applications des mécanismes leur permettant de mettre en œuvre leur propre politique de protection. Dans le contexte de réseaux grandes distances, les messages échangés peuvent être écoutés ou interceptés et il n'est pas possible de faire l'hypothèse que tous les systèmes des machines sont dignes de confiance (comme pour une machine centralisée ou un réseau local). Ces contraintes nécessitent la mise en place de mécanismes de chiffrement et d'authentification des messages échangés.

L'hétérogénéité est également un des problèmes cruciaux de l'informatique répartie. Alors que sur un réseau local, il est possible dans certains cas de faire l'hypothèse que les machines sont toutes de même type, sur des grands réseaux comme l'Internet, cette hypothèse n'est plus réaliste et il est nécessaire de mettre en œuvre des mécanismes traitant les problèmes provenant des différents formats de code et de données rencontrés.

La répartition des architectures matérielles amène également d'autres problèmes très importants comme la tolérance aux pannes des composants de l'architecture (la probabilité de panne augmentant avec la répartition) ainsi que la répartition de charge permettant de mieux utiliser les ressources disponibles.

Dans ce contexte d'évolution des systèmes répartis, ce sont les problèmes liés à l'accès et à la protection (contrôle d'accès) qui ont motivé mes travaux de recherche.

1.3. Accès à l'information

Les méthodes d'accès à des informations réparties doivent permettre d'y accéder à partir d'un nom. Il existe plusieurs types de noms dans les systèmes répartis, les plus connus étant les noms de fichiers et les URLs car ce sont ceux manipulés directement par l'utilisateur. Il en existe d'autres que l'on veut souvent cacher à l'utilisateur comme les adresses disque, mémoire ou de machine. Dans tous les cas, un nom doit permettre de retrouver l'objet qu'il désigne pour y accéder. On appelle localisation la fonction permettant de retrouver un objet à partir de son nom. Cette fonction de localisation doit prendre en compte le fait qu'un objet peut être déplacé.

Lorsqu'un objet a été localisé, une méthode d'accès physique à l'objet est utilisée pour lire ou écrire l'objet. Cette méthode peut être plus ou moins complexe en fonction du système, pouvant prendre la forme d'un appel de procédure à distance ou d'un rapatriement d'une copie de l'objet pour réaliser l'accès localement.

Les mécanismes mis en œuvre pour la localisation et pour l'accès à un objet doivent être efficaces. Cette efficacité se mesure en termes de temps CPU, de temps de communication entre les machines, du taux d'occupation des mémoires des machines et du nombre d'entrées sorties disque.

1.4. Contrôle d'accès

Le contrôle d'accès vise à fournir aux applications le moyen de mettre en œuvre une politique d'autorisation des accès aux objets qu'elles gèrent. En général, l'autorisation d'un accès dépend de l'usager réalisant l'accès ainsi que l'objet auquel il accède.

Une méthode d'authentification des usagers est utilisée afin de s'assurer de l'identité de l'initiateur de l'accès. Afin de s'assurer de la confidentialité des données transférées sur le réseau, des mécanismes de chiffrement sont utilisés.

Pour permettre aux applications de définir leur propre politique de gestion des droits d'accès, le système doit fournir un modèle de protection. Ce modèle doit fournir une interface permettant d'associer des droits d'accès à des usagers, mais également de définir dans quelles conditions ces droits d'accès pourront évoluer dans le temps.

Un modèle de protection s'évalue par sa résistance aux attaques, mais aussi par sa souplesse. La souplesse définit ici la capacité du système à permettre la mise en œuvre de la politique de protection d'une application. En effet, un modèle trop rigide incite les programmeurs d'application à surdimensionner les droits accordés aux usagers des applications, voire même à ne pas protéger l'application. Il est en général préférable de mettre en application le principe du moindre privilège et de n'accorder initialement que les droits

strictement nécessaires, ces droits pouvant évoluer dynamiquement par la suite. Le surcoût lié aux mécanismes de protection rentre également en ligne de compte dans l'évaluation d'un système de protection.

1.5. Travaux de recherche

Dans le cadre de mes travaux de recherche, j'ai principalement étudié les mécanismes mis en œuvre pour accéder à des informations réparties et partagées entre un ensemble d'usagers coopérants. Le partage de données entre plusieurs usagers impose la mise en place d'autres mécanismes de contrôle d'accès visant à protéger ces données contre des tentatives d'accès illégaux.

Mes travaux dans ce domaine se situent dans les trois types d'architecture présentés ci-dessus, à savoir les réseaux locaux, les clusters et l'Internet plus récemment.

1.5.1. Dans les réseaux locaux

J'ai tout d'abord commencé par étudier la conception des systèmes répartis sur des réseaux locaux de stations de travail (projet Guide).

L'objectif était de réaliser un système réparti complet pour le développement d'applications réparties. Afin de faciliter la structuration des applications réparties, une approche à objets a été choisie. Les applications étaient programmées avec un langage orienté-objet et le système gérait implicitement le partage, la permanence, l'accès et le contrôle d'accès aux objets partagés.

Deux prototypes ont été développés respectivement sur les systèmes Unix et Mach. Ces prototypes comprenaient des langages de programmation orientés-objets, des machines à objets comme support de ces langages (gérant les objets partagés répartis) et des outils de mise au point des programmes intégrés dans un environnement de développement. Plusieurs applications réparties de taille importante ont été développées par des utilisateurs externes au projet et ont permis de valider ces prototypes.

Dans ce contexte, je me suis plus particulièrement intéressé à l'adressage et à la protection de données partagées réparties. Les résultats de ces travaux ont été intégrés dans un environnement complet de système réparti à objets. Ils ont montré qu'il était possible de réaliser des mécanismes d'accès intégrant le contrôle d'accès aux objets avec des performances acceptables. Ils ont également montré l'adéquation des micro-noyaux de système pour la construction d'environnements plus complexes, principalement du point de vue de la modularité et de la spécialisation des services du système. Enfin, ces travaux ont montré l'intérêt de fournir un environnement à objets répartis pour le développement d'applications réparties.

1.5.2. Dans les clusters

Plusieurs évolutions techniques m'ont ensuite poussé à m'intéresser aux architectures de type cluster, reposant sur une intégration plus forte des machines dans l'environnement réparti (projets Raven et Sirac). Si les besoins du point de vue applicatif ne changent guère,

les méthodes d'accès et de protection des données sont remises en cause, notamment par l'arrivée des réseaux rapides et des processeurs à capacité d'adressage étendue.

L'objectif de ces travaux était donc d'étudier l'influence de ces évolutions techniques sur la gestion de données réparties partagées. L'utilisation de processeurs à capacité étendue d'adressage rend plausible la gestion d'un espace d'adresses virtuelles unique, permettant une mise en œuvre plus efficace d'une partie des mécanismes d'accès aux données. L'utilisation des réseaux rapides permet une gestion globale des ressources matérielles disponibles sur le cluster, notamment des mémoires et des disques.

Un prototype a été développé sous la forme d'une mémoire virtuelle répartie. Deux applications de taille importante ont été construites sur ce prototype : un système de gestion de fichiers répartis et un système de gestion de base de donnée orienté-objet.

Dans ce contexte, je me suis intéressé à la gestion de la mémoire et à la protection des données dans ce système. Dans un système à espace virtuel unique, la protection revêt un intérêt particulier étant donné que celle-ci ne peut plus reposer sur la gestion d'un espace virtuel privé associé à chaque processus (comme dans les systèmes traditionnels). Un nouveau modèle de protection a été proposé et implanté dans le prototype de mémoire virtuel tournant sur un cluster. Les expérimentations avec ce modèle de protection ont démontré sa souplesse.

1.5.3. Dans l'Internet

Mes travaux de recherche s'orientent actuellement vers les applications réparties sur des réseaux à grande distance et en particulier sur l'Internet. En effet, les évolutions dans le domaine des communications et l'ouverture de l'Internet au grand public vont développer les applications permettant la coopération entre des usagers ou des entreprises répartis dans le monde entier. Des domaines applicatifs privilégiés sont le commerce électronique et l'ingénierie coopérative.

L'objectif de ces travaux autour des réseaux grande distance est de concevoir et de réaliser les mécanismes systèmes qui permettent de faciliter le développement de ces applications.

Dans ce contexte, je me suis initialement intéressé à un nouveau modèle pour le développement d'applications réparties, à savoir la programmation par agents mobiles qui consiste à déplacer des programmes entre les machines afin d'accéder aux données gérées sur ces machines. Dans la programmation par agents, le problème le plus délicat est la sécurité. En effet, les usagers qui rendent disponibles des données ou des services sur leurs machines doivent être en mesure de se protéger des attaques d'agents contre la confidentialité ou l'intégrité des données gérées. Un modèle de protection a été conçu et réalisé pour répondre à ces besoins.

Un projet plus ambitieux est actuellement en cours de définition, visant à fournir sur Java un environnement pour le développement d'applications coopératives réparties.

1.6. Structure du document

Ce rapport vise trois objectifs :

- donner un aperçu de mes travaux de recherche dans le domaine des systèmes répartis,
- en faire ressortir les contributions les plus significatives,
- présenter les thèmes de recherche que je compte développer dans les années à venir.

Volontairement, ce rapport ne rentre pas dans des détails techniques. Les contributions techniques sont rapidement présentées en donnant des références aux publications qui sont données en annexe de ce rapport.

La suite de ce rapport est structuré de la façon suivante. La section 2 contient une analyse des problèmes et des principes de base des mécanismes d'accès et de contrôle d'accès dans les systèmes répartis. Dans la section 3 est proposée une vue globale des projets de recherche auxquels j'ai contribué ; il s'agit d'une présentation chronologique de ces travaux, incluant les références aux publications décrivant les contributions dans le détail. La section 4 présente une évaluation synthétisant les résultats et les comparant aux projets similaires. La section 5 conclut ce rapport avec mon plan de recherche pour les années à venir.

2. Principes généraux

Dans cette section, je présente les principes généraux qui régissent l'accès et le contrôle d'accès dans les systèmes répartis. L'objectif est d'introduire les notions de base qui permettent une présentation plus claire de mes contributions dans les sections suivantes.

2.1. Position du problème

Avant de décrire ces principes, j'introduis les concepts et la terminologie utilisés et je présente les problèmes associés à ces différents concepts. Il s'agit des notions d'objet, de partage, de persistance, de répartition et de protection.

Objets

Le terme d'objet est généralement utilisé en informatique pour désigner toute unité d'encapsulation. Il peut être utilisé pour désigner un module contenant du code, une zone de données, ou même un processus serveur. Les définitions rencontrées dans les différents projets de systèmes ne sont pas toujours en accord. Je précise donc ma propre définition des objets, proche de celle des langages de programmation.

Un objet est une entité contenant des données aussi appelées état de l'objet et exportant des opérations appelées méthodes qui permettent l'utilisation de l'objet. En général, l'état de l'objet est caché aux autres objets et la seule façon d'utiliser un objet est d'appeler une de ses méthodes. Cette définition est souvent associée au concept de Classe. La classe est un moyen donné au concepteur d'application pour définir une structure et un comportement commun à un ensemble d'objets. On dit que les objets qui ont la structure et le comportement décrit par une classe sont des exemplaires ou instances de cette classe. La définition d'une classe revient donc à définir :

- La structure des instances de la classe. Cette structure est commune à toutes les instances de cette classe. Les variables de cette structure sont aussi appelées variables d'état des instances.
- Le comportement des instances de la classe. Ce comportement est décrit dans la classe par la définition (le code) des méthodes appelables sur ses instances.

Notons que le concept d'objet peut être appliqué à toutes les informations gérées dans un système, celles-ci ayant généralement une structure clairement définie et un code permettant de les manipuler. Par exemple, un fichier peut être considéré comme un objet de classe *Fichier* avec des méthodes *ouvrir*, *lire*, *écrire* et *fermer*.

Partage

Pour le développement d'applications réparties, l'intérêt d'un modèle de programmation fondé sur le partage implicite des données manipulées est de fournir aux utilisateurs une abstraction très commode et notamment de masquer les mécanismes de communication sous-jacents. En effet, au niveau des langages, le partage implicite de données permet une programmation plus aisée que l'utilisation explicite de primitives d'envoi de message, ou de

recopie dans une zone partagée. Le développeur d'application peut programmer sans se soucier du fait que les données manipulées doivent être partagées, bien qu'il doive ensuite synchroniser l'accès à ces données.

Nous disons qu'il y a partage d'objets entre des processus différents dès que ces processus ont la possibilité d'appeler les méthodes d'un même objet. Permettre le partage d'objets, c'est tout d'abord fournir un moyen de les désigner par des identificateurs (ou noms) d'objets, puis permettre à tout processus d'utiliser un objet (par appel de méthode) à partir d'un identificateur le désignant.

Le partage nécessite donc la gestion d'un schéma de désignation ainsi que des mécanismes de partage physique de la mémoire entre les applications. Un objet doit pouvoir contenir des noms d'objets, ce qui permet de construire des structures complexes.

La gestion de noms d'objets par le système a comme unique but l'identification et le partage d'objets. Un service de nommage symbolique, permettant d'associer des chaînes de caractères à des noms d'objets, est généralement fourni comme une application du système que nous définissons.

Persistance

Un objet est dit persistant s'il survit à l'application qui l'a créé. On distingue deux niveaux de persistance, en fonction de la survie d'un objet à un arrêt de machine (provoqué ou non). En effet, un objet peut être persistant (survivre à la fin de l'application génitrice) à condition que la machine ne soit jamais arrêtée. En général, on considère qu'un objet est persistant lorsqu'il a été sauvegardé sur un support stable comme un disque.

Dans la plupart des systèmes traditionnels, la gestion de la sauvegarde des données est laissée à la charge du programmeur d'applications, ses outils étant principalement les primitives du système de gestion de fichiers. De plus, cette gestion de la persistance n'est pas uniforme, puisque les données persistantes et temporaires ne sont pas désignées de la même façon.

La volonté d'uniformiser l'accès à ces deux types de données et de décharger le programmeur de la tâche fastidieuse de sauvegarde a donné naissance à des systèmes où la gestion de la persistance des objets est implicite. Ces systèmes permettent alors de définir des objets temporaires ou persistants, tous ces objets étant désignés de façon uniforme.

Répartition

Gérer la répartition, c'est permettre l'utilisation de toutes les ressources disponibles sur le réseau de stations de travail. Le fait que les ressources soient réparties signifie que les objets manipulés dans le système sont disséminés sur tous les disques des machines. Il faut donc être capable de localiser un objet dans le système réparti et permettre son partage entre des applications ne s'exécutant pas forcément sur la même machine.

Pour offrir une gestion d'objets efficace, il faut à la fois permettre le regroupement de certains objets pour augmenter la localité d'accès et permettre la répartition de ces objets afin d'augmenter le parallélisme de l'exécution.

Enfin, intégrer la répartition, c'est également ne plus la montrer, ou plutôt ne la montrer que lorsque c'est nécessaire. Le nommage des objets ne doit donc en aucun cas être lié à des noms de machines. Il doit donner la vision d'un espace uniforme d'objet, cachant le fait que ces objets sont répartis dans le réseau.

Protection

Un des concepts clés associé à la notion d'objet est l'encapsulation des données. Cela signifie que la seule façon d'accéder en lecture ou en écriture à l'état d'un objet est l'appel à une méthode de cet objet.

Cette encapsulation peut être assurée au niveau des langages de programmation utilisés, lorsque ces langages sont fortement typés. Un langage est fortement typé si chaque variable se voit associer un type, qui définit l'ensemble des opérations permettant la manipulation d'une variable de ce type. Le compilateur du langage peut alors assurer l'encapsulation des objets, grâce à des contrôles statiques (réalisés lors de la compilation) et à des contrôles dynamiques (réalisés à l'exécution), en vérifiant que seuls des opérateurs autorisés sont appliqués aux variables. Ces langages ne permettent généralement pas la manipulation directe d'adresses en espace virtuel et n'offrent que la manipulation de types de base et de références à des objets. Les débordements de tableaux sont contrôlés et l'utilisation d'une référence à un objet n'est possible que par appel de méthode.

D'autres langages, comme C++, permettent d'adresser toutes les données de l'espace virtuel courant en construisant des pointeurs et n'assurent donc pas l'encapsulation des données. Ce défaut n'est pas très grave dans un contexte où les applications ne partagent pas d'objets, une erreur dans un programme ne pouvant détruire que des objets du même utilisateur.

Dès l'instant où l'on permet aux applications de partager des objets, le problème de l'encapsulation des objets devient critique, l'appel d'une méthode sur un objet appartenant à une autre application (resp. un autre utilisateur) ne signifiant pas forcément une confiance absolue envers cette application (resp. cet utilisateur). Il faut qu'un appel de méthode ne rende pas l'appelant ou l'appelé sensible aux erreurs ou malveillances de son alter-ego. Nous appelons confinement le fait de garantir un certain degré d'isolation entre des applications et des usagers pouvant partager des objets. Lorsqu'il n'est pas réalisé au niveau des langages de programmation, ce confinement doit être assuré par le système. Il peut alors assurer l'isolation de chaque objet ou l'isolation d'ensembles d'objets se faisant mutuellement confiance.

De plus, même si l'encapsulation des objets est garantie par le système et ne permet que l'appel de méthode sur les objets partagés, il faut permettre au programmeur d'application de contrôler les droits des utilisateurs en termes de méthodes appelables sur les objets partagés. Nous appelons contrôle d'accès le mécanisme permettant de limiter la vue (en terme de méthodes appelables) d'un utilisateur sur un objet.

2.2. Accès à l'information

Je présente maintenant les principes généraux de mise en œuvre de l'accès à l'information. Une version étendue de ces principes généraux se trouvent dans [HAG96a].

2.2.1. Modèle de système

L'accès à une information est déclenché par une application ou un processus qui est une unité d'exécution d'une application. Le code de cette application manipule des noms d'objets, ces objets contenant l'information en question, et l'objectif de l'accès est de réaliser une opération (ou méthode) sur cet objet.

Afin de présenter les principes de base de l'accès à l'information, considérons un modèle de système constitué de deux niveaux de mémoire dans lesquels les objets peuvent être manipulés. Cette distinction est adoptée pour des raisons de présentation et peut totalement disparaître dans certains systèmes.

J'appelle **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.

J'appelle **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 utilise des noms d'objets pour réaliser des appels de méthode. Le système doit alors résoudre ces noms d'objets pour accéder à l'état des objets. On appelle *résolution de nom* [SAL78] l'opération qui consiste à retrouver un objet à partir de son nom dans une des deux mémoires introduites ci-dessus. La résolution de nom en mémoire d'exécution donne une adresse virtuelle. Elle donne une adresse sur un des disques en mémoire de stockage.

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.

2.2.2. Mémoire de stockage

Préambule

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 [BEN92]. L'opération de résolution de nom est aussi appelée "localisation" des objets en mémoire de stockage.

Comme dans un système de gestion de fichiers ou dans le *World Wide Web* (avec les URLs), la gestion des noms et de la localisation peut se faire selon deux modes : avec des noms relatifs ou absolus. 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 [MOS89].

Nous traitons respectivement de l'utilisation de noms relatifs et absolus dans le reste de la section.

Noms relatifs

Dans le cas d'un nommage relatif, un objet ne peut contenir une référence que vers un objet de la même 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 [MAI92].

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 [DAY92, LIS92] : la gestion des homonymes (le même nom pouvant désigner des objets différent dans deux partitions différentes) et la gestion des synonymes (deux noms différents pouvant désigner le même objet). En générale, la solution adoptée consiste à assurer l'unicité des noms dans la mémoire d'exécution.

Noms absolus

On oppose au nommage relatif le nommage absolu, avec lequel un objet peut désigner directement tout objet du système. 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.

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 [LEG88], utilisant notamment la duplication de catalogues (pour la disponibilité) [BIR82] et la diffusion (pour la localisation) [TAN90].

Lorsque des informations de localisation sont incluses dans le nom de chaque objet, on distingue différents cas, en fonction de la nature de cette information. Il peut s'agir de la localisation initiale de l'objet [POW83], le problème de la migration est alors résolu en utilisant des liens de poursuite. Il peut s'agir de la localisation courante de l'objet [FRE91], mais il faut alors traiter le problème des synonymes comme pour le nommage relatif.

2.2.3. Mémoire d'exécution

Préambule

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

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.

Trois éléments motivent l'organisation de la mémoire d'exécution : le confinement de l'exécution, le partage d'objet et l'adressage. Le confinement prend 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. Le partage d'objets consiste à rendre les objets accessible entre des structures d'exécution pouvant être sur des machines différentes. Enfin, lorsqu'un objet doit être adressé à partir de son nom, il faut obtenir une adresse virtuelle à partir d'un nom d'objet, permettant un adressage physique de l'objet. Je traite ces trois aspects dans la suite.

Confinement de l'exécution

L'outil de base pour assurer le confinement est le domaine de protection (définit ci-dessus). La séparation entre domaines peut être utilisée pour créer des cloisonnements entre les activités et les objets :

- Confinement entre les activités. 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 comme une erreur d'adressage. Si 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

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

des 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 (par exemple dans Thor [LIS92]).

- Confinement entre les 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 peut 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 de protection (comme dans les systèmes Argus [LIS87] ou Clouds [DAS90]). Avec le confinement entre objets, le problème est la taille des objets (grain) et le coût de l'appel d'objet. Un objet est associé à un domaine de protection et un appel d'objet nécessite un changement de domaine, qui est une opération très coûteuse. En général, la taille des objets comparée au coût de gestion des domaines de protection incite au regroupement d'objets et à assurer l'isolation pour un ensemble d'objets. Différents critères de regroupement d'objets peuvent être utilisés, notamment le regroupement par propriétaire d'objet [HAG93] et par application.

Enfin, une technique permettant d'assurer le confinement des objets à moindre coût consiste à reposer sur les langages de programmation en supposant que toutes les applications sont écrites avec des langages sûrs[2]. Ces langages sont fortement typés et ne permettent pas la manipulation directe d'adresses virtuelles. Ils implantent des vérifications à la compilation qui garantissent qu'un programme ne peut pas passer outre ce confinement. L'exemple le plus connu à l'heure actuelle est certainement Java [GOS95].

Partage d'objets

Les mécanismes de partage de mémoire entre activités dans le même domaine ou entre domaines sur la même machine sont bien connus. J'insiste plus particulièrement sur le partage d'objets entre activités sur des machines différentes.

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 où l'objet est partagé par des activités sur des périodes ne se recouvrant pas. 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 [MOS90] ou Thor [LIS92].

Dans le partage simultané, deux grandes techniques sont utilisées.

La première est qualifiée de *function-shiping* et consiste à se ramener au cas du partage en local en déplaçant l'activité réalisant l'accès. Une activité voulant accéder à un objet déjà

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

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. L'outil le plus répandu mettant en œuvre cette stratégie est l'appel de procédure à distance (Remote Procedure Call) [BIR84].

La deuxième est qualifiée de *data-shiping* et consiste à gérer des 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 [MOS93]. 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 invalidation ou par diffusion.

Le partage avec cohérence par invalidation 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 (invalidation) 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 peut être la page [LI89] ou l'objet [BER93].

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. 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 [BAL92].

Adressage des objets

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 appelée défaut d'objet 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.

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 possible de gérer un ou plusieurs contextes (ou espace) 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, différents cas sont envisageables, les deux principaux étant :

- 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 généralement employés, afin d'éviter une résolution de nom systématique (aussi appelée interprétation). 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 est de modifier lors de la première liaison cette variable et de l'enrichir d'une information permettant de la résoudre plus rapidement lors des résolutions suivantes.

  La forme la plus répandue de ces raccourcis est le *swizzling* (que je traduis par mutation de pointeur). On remplace 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. On trouve plusieurs formes de mutation de pointeurs [MOS92] qui diffèrent par la technique utilisée pour détecter si un nom a déjà été changé en adresse. Les deux principales formes sont la mutation paresseuse ou *Lazy swizzling* (un nom est transformé à sa première utilisation et une information dans le nom permet de détecter si le nom est déjà transformé) et la mutation anticipée ou *Eager swizzling* (à la liaison d'un objet dans un espace virtuel, tous les noms qu'il contient sont transformés pour pointer sur l'objet désigné s'il est résident, ou sur une objet intermédiaire sinon [LIS92]).

  Il faut noter que la mutation de pointeur se prête bien à des systèmes où le partage est sérialisé, car la liaison simultanée dans plusieurs espaces virtuels entraîne le problème de la cohérence des mutations dans les objets partagés.

- Liaison statique dans un espace virtuel unique.

  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 évite ainsi l'opération coûteuse de résolution de nom qui a lieu lorsque des espaces virtuels différents sont gérés. Cette solution est d'autant plus crédible que des processeurs dont les espaces virtuels sont adressés par des adresses de 64 bits sont disponibles sur le

marché. Cette technique est utilisée dans les projets Opal [CHA94] et Arias [DEC96b].

2.2.4. Résumé

J'ai présenté dans cette section les principes qui régissent l'adressage d'objets partagés persistants dans les systèmes répartis. Le principal problème en mémoire de stockage est la localisation des objets sachant qu'il peuvent être déplacés entre les partitions de stockage. En mémoire d'exécution, les principaux problèmes concernent le confinement de l'exécution, le partage les objets dans un environnement réparti et enfin l'adressage des objets nécessitant une traduction des noms en adresses virtuelles.

Pour synthétiser cette présentation, on peut distinguer trois grandes familles de systèmes utilisant ces techniques:

- 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. 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. Des exemples sont notamment les systèmes Thor [LIS92] et Mneme[MOS90].

- 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 et un domaine de protection propre à chaque objet en mémoire d'exécution. Compte tenu du coût des appels entre objets, 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 le domaine de protection associé à l'objet global. Des exemples sont les systèmes Argus [LIS87] et Clouds [DAS90].

- Les systèmes à partage concurrent d'objets plus fins. 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. D'autres systèmes ont donc visé à 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 [KRA90], Emerald [HUT87]), soit assurée par l'utilisation de domaines de protection (Guide v2 [HAG93], Orbix [ION94], Opal [CHA94]). 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.

2.3. Contrôle d'accès

Le contrôle de l'accès à l'information vise à s'assurer que seuls les usagers autorisés à accéder à l'information peuvent effectivement y accéder. Dans le contrôle d'accès, je distingue les mécanismes matériels et systèmes qui permettent de s'assurer l'inviolabilité du système (sécurité du système), des mécanismes qui permettent à une application d'exprimer une politique de protection. Une politique de protection détermine comment les droits d'accès sont accordés aux usagers du système.

2.3.1. Sécurité du système

De façon générale, la sécurité du système doit répondre aux attaques contre la confidentialité et l'intégrité de l'information. La confidentialité fait référence à la capacité du système d'empêcher la divulgation de l'information. L'intégrité correspond à la capacité du système d'empêcher des modifications illégales de l'information. De façon simplifiée, la confidentialité est une protection en lecture et l'intégrité en écriture.

Parmi les systèmes de sécurité, on distingue généralement les systèmes discrétionnaires des systèmes mandataires [BAL91b]. Dans l'approche discrétionnaire, un usager autorisé à lire un objet peut copier l'état de l'objet dans un autre objet et donner le droit de lire la copie à un usager non autorisé à lire le premier objet. La confidentialité est à la discrétion de l'usager. Dans l'approche mandataire, des règles sont ajoutées visant à contrôler le flux de l'information entre des usagers de classes différentes (souvent une "habilitation" de type militaire). Dans le contexte de mon travail, je ne m'intéresse volontairement qu'aux politiques discrétionnaires.

Dans la suite, je présente respectivement les mécanismes permettant d'assurer la sécurité sur une machine centralisée, puis ceux qui sont rajoutés afin de l'assurer dans un environnement réparti.

En centralisé

Comme cela a été mentionné en section 2.2.3, l'outils de base permettant de confiner l'exécution est le domaine de protection. Un domaine de protection définit un ensemble d'objets accessibles par toute activité s'exécutant dans ce domaine. 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. Cette opération permet à une activité d'étendre ses droits afin d'exécuter une opération protégée. L'utilisation de cette opération de changement de domaine doit être contrôlée ; cet aspect est abordé en section 2.3.2.

On associe généralement des droits initiaux à toute nouvelle activité (ou exécution d'une application) en fonction de l'usager qui démarre cette exécution. L'authentification de cet usager est un mécanisme visant à s'assurer de son identité. La technique la plus couramment utilisée est une authentification par clé secrète (ou mot de passe). Une clé choisie par l'usager est enregistrée dans le système, sa confidentialité est assurée et cette clé est demandée à

l'usager lorsqu'il se connecte au système. La connexion initialise un domaine de protection, incluant les droits de l'usager, dans lequel les applications démarrées par l'usager s'exécutent.

Les notions de domaine de protection et d'authentification des usagers sont les deux mécanismes de base permettant d'assurer la protection d'un système.

En réparti

La répartition rend nécessaire l'intégration d'une technique qui vise à assurer la confidentialité et l'intégrité des données sur les réseaux. Il s'agit du chiffrement des données échangées.

Le chiffrement doit tout d'abord être utilisé pour assurer la confidentialité des données. Les données sont chiffrées avant d'être envoyées sur le réseau, puis déchiffrées avant d'être transmise au destinataire final. A noter que le chiffrement peut être réalisé par un dispositif logiciel comme matériel. Le chiffrement doit également être utilisé à des fins d'authentification des parties qui communiquent par envoi de messages sur le réseau [NEE78]. L'authentification vise à s'assurer de l'identité de l'émetteur du message, afin d'éviter une mascarade, c'est à dire qu'une personne se fasse passer pour une autre.

Il existe deux grandes familles d'algorithmes de chiffrement : les algorithmes de chiffrement avec ou sans apport de connaissance :

- Chiffrement avec apport de connaissance. Chaque partie à une clé privée de chiffrement (qui sert en général à chiffrer et à déchiffrer). Un serveur d'authentification connaît toutes les clés. Ce serveur authentifie les parties communicantes (par leur clé privée) et alloue une clé qui est utilisée pour les communications entre les deux parties. Le problème de cette technique est qu'il faut établir une connexion entre les deux parties avec allocation d'une clé partagée (complexité et coût). Un exemple est le système Kerberos [STE88] du MIT.

- Chiffrement sans apport de connaissance. Chaque partie dispose d'une clé privée et d'une clé publique. La clé publique (resp. privée) permet de déchiffrer ce qui est chiffré avec la clé privée (resp. publique). Un simple serveur de clés (qui peut être dupliqué) permet de distribuer les clés publiques. L'utilisation (en chiffrement) de la clé publique de l'usager U par un usager U' permet d'assurer la confidentialité des données envoyées par U' (car seul U peut déchiffrer avec sa clé privée). L'utilisation (en chiffrement) par U de sa clé privée permet d'authentifier U auprès du receveur U' (car seul U peut chiffrer avec sa clé privée). Le problème du chiffrement par clés publiques est d'être sûr de l'identité de la personne dont on obtient une clé publique, sans quoi une personne peut se faire passer pour une autre et attenter à la confidentialité. Un exemple est PGP [ZIM95].

2.3.2. Contrôle d'accès

Les mécanismes systèmes mentionnés ci-dessus permettent de définir des domaines de protection contenant des droits d'accès et d'authentifier des usagers, ce qui permet de leur associer un domaine de protection pour l'exécution de leurs applications.

Je m'intéresse maintenant à la gestion des droits d'accès dans le système. Il s'agit de la façon de décrire les droits d'accès accordés aux usagers dans le système. Etant donné que ces droits d'accès sont amenés à évoluer lors de l'exécution des applications, il est nécessaire de fournir aux applications un modèle permettant de définir le comportement d'une application vis à vis de la protection, c'est à dire les règles d'évolution des droits d'accès.

Je décris tout d'abord les besoins des applications pour la définition d'une politique de protection, puis je présente les deux principaux modèles existants.

Besoins

L'objectif du contrôle d'accès est de fournir aux applications un moyen de contrôler les droits qu'elles accordent sur les objets qu'elles gèrent. Les objectifs généraux qui doivent être pris en compte sont les suivants :

- Un contrôle de l'accès au niveau de l'objet et de l'opération réalisée sur l'objet. Tout objet étant potentiellement partageable, il doit être possible de définir des règles de contrôle d'accès différentes pour chaque objet du système. Bien que les opérations puissent être de niveaux différents (lecture-écriture, méthode), nous supposons que le contrôle d'accès doit se faire en termes de méthodes appelables sur des objets.

- Un contrôle d'accès en fonction du domaine de protection courant. Une activité s'exécutant pour le compte d'un usager s'exécute dans un domaine de protection. Les droits d'accès sont accordés aux domaines et une application a les droits d'accès de son domaine d'exécution. Les droits d'un usager du système sont donnés par les droits inclus dans son domaine de protection initial (déterminé lors de sa connexion au système). Dans la suite, nous appelons "droits d'accès d'un usager" les droits d'accès inclus dans son domaine d'exécution.

- Un mécanisme de changement de domaine de protection. Il doit être possible d'étendre dynamiquement les droits d'une activité pour l'exécution d'une opération spécifique correspondant à un service protégé. Cette possibilité est généralement fournie à travers une opération de changement de domaine de protection. Une activité s'exécute à un instant donné dans un seul domaine de protection, mais elle peut changer de domaine de protection par l'intermédiaire d'un point d'entrée du nouveau domaine de protection. Ce mécanisme a généralement pour but de permettre l'écriture de sous-systèmes protégés. Si on prend l'exemple d'un sous-système dont le but est de fournir un service en interdisant aux clients du service d'appeler directement les objets internes, il faut cloisonner le service dans un domaine de protection et imposer le passage par des points d'entrée. Les appels aux objets internes ne sont autorisés qu'après passage par un des points d'entrée. Il doit être également possible de contrôler le passage par les points d'entrée en fonction de l'usager ou du domaine de protection appelant.

  Un exemple est un sous-système gérant une imprimante. Des objets internes comme la queue des impressions en attente ne sont pas directement accessibles. Seuls quelques points d'entrée permettent de changer de domaine de protection pour exécuter le code lançant une impression par exemple.

- L’évolution des droits d’accès. Une application s’exécutant dans un domaine doit être en mesure de transmettre au domaine destinataire, lors d’un changement de domaine, des droits d’accès qu’elle possède. Cette fonction est très importante pour permettre la coopération entre des sous-systèmes protégés et en particulier pour implanter le principe du *moindre privilège* : on accorde toujours des droits minimaux à ses clients et on accorde dynamiquement les droits nécessaires en fonction des besoins.

  Sur l’exemple précédent, lorsqu’une application décide de lancer l’impression d’un objet fichier, elle doit donner au domaine gérant l’imprimante les droits d’accès permettant de lire le fichier.

- La révocation des droits d'accès. Une application ayant accordé des droits d'accès à un domaine de protection doit être en mesure de les révoquer, c'est à dire de retirer les droits donnés de ce domaine de protection.

Après avoir passé en revue les exigences concernant le contrôle de l’accès aux objets, nous allons étudier les techniques utilisées dans des systèmes caractéristiques.

Dans un système, les règles de protection des objets peuvent être représentées par une matrice [LAM71] indicée respectivement par les domaines de protection (généralement associés à des usagers du système) et par les objets du système. La case *M[Domaine2, Obj3]* de la matrice donne alors les droits accordés au domaine *Domaine2* sur l’objet *Obj3*.

	Obj1	Obj2	Obj3	
Domaine1		Meth1 Meth2		
Domaine2			Meth1	
Domaine3		Meth2		

Une approche classique consiste alors à associer à chaque objet une **liste d’accès** qui contient pour chaque domaine de protection du système ses droits d’accès sur cet objet. Cette solution revient à regrouper les informations de protection de la matrice par colonnes. Une autre approche classique consiste à associer à chaque domaine de protection une liste appelée **liste de capacités**, contenant les droits d’accès sur les objets du système accordés à ce domaine. Cette solution revient à regrouper les informations de la matrice par ligne.

Nous présentons maintenant ces approches.

Liste d’accès

Une liste d’accès est associée à chaque objet et définit pour chaque domaine ses droits en termes d’opérations sur cet objet. Le système a alors la charge de vérifier que seuls des appels autorisés par les listes d’accès sont exécutés. Ainsi, les deux premiers objectifs sont

atteints. Il y a contrôle d'accès au niveau de l'objet, puisqu'une liste d'accès est associée à chaque objet et cette liste décrit les droits de chaque domaine sur cet objet.

En général, le propriétaire d'un objet est autorisé à modifier la liste d'accès de l'objet, ce qui lui permet d'accorder des droits supplémentaires à un autre domaine de protection avec lequel il interagit.

Un mécanisme est généralement ajouté pour permettre le changement de domaine de protection. Nous décrivons ici différents exemples :

- Dans Multics [ORG72], la notion d'anneau est comparable à celle de domaine de protection[3]. Une activité peut changer d'anneau de protection en utilisant une opération privilégiée (*CALL*) d'appel de procédure sur un segment procédure se trouvant dans l'anneau cible. A chaque segment procédure est associée une liste de points d'entrée appelés *guichets* et seuls ces points d'entrée sont appelables par l'instruction *CALL* .
- Unix fournit un mécanisme appelé le *setuid* pour changement d'identifiant d'usager. Pour chaque entrée de la liste d'accès d'un fichier exécutable, le droit d'exécution peut s'accompagner du *setuid*, ce qui signifie que l'exécution du binaire s'accompagne d'un changement d'identité de l'activité qui a alors les droits du propriétaire du fichier binaire utilisé. Ce changement d'identité correspond à un changement de domaine de protection, un domaine étant ici associée (un pour un) à un usager.
- Birlix [KOW90] permet de spécifier dans les listes d'accès un identificateur de classe et une méthode, ce qui signifie que l'appel de cette méthode est autorisée depuis toutes les instances de cette classe. Il suffit alors de positionner la protection des objets internes de telle sorte qu'ils ne puissent être appelés que depuis un objet dont la classe est interne au service protégé. On interdit ainsi les appels directs aux objets internes depuis des instances de classes externes au service protégé.

De façon générale, l'avantage reconnu des listes d'accès est de centraliser l'information relative à la protection au niveau de l'objet lui-même, facilitant ainsi la gestion des droits, ce qui n'est pas le cas des capacités (présentées ci-dessous). Un mécanisme de changement de domaine est fourni pour permettre d'étendre les droits d'une activité à des objets internes à un sous-système protégé.

Par contre, dans ces systèmes, aucun mécanisme particulier n'est prévu pour accorder dynamiquement des droits supplémentaires à un domaine de protection. Dans la pratique, seuls des mécanismes d'administration statique sont fournis. Cela implique que le principe du moindre privilège est plus difficile à mettre en œuvre dans ces systèmes.

Capacités

Conceptuellement, une capacité [LEV84] est un ticket permettant à son possesseur d'utiliser un objet avec des droits déterminés. Une capacité contient le nom de l'objet sur lequel des droits sont accordés, ainsi que la définition de ces droits en termes d'opérations sur l'objet. Une capacité est protégée par le système dans le sens où il n'est pas possible de la

[3] Bien que les anneaux de Multics soient concentriques, ce qui implique une hiérarchie entre les anneaux.

forger ou de la modifier. Chaque domaine de protection contient une (ou plusieurs) liste(s) de capacités qui déterminent les droits d'accès des activités dans ce domaine.

Dans les modèles à capacités, un mécanisme de changement de domaine est fourni. Le changement de domaine n'est possible qu'aux activités qui s'exécutent dans un domaine contenant une capacité spéciale appelée *capacité de changement de domaine* ou *capacité de domaine*. Une capacité de changement de domaine est généralement associée à une opération qui peut être une procédure ou une méthode sur un objet. Lorsque la capacité est utilisée, l'activité appelante change de domaine et l'opération associée est appelée dans le nouveau domaine de protection.

Le troisième élément clé des systèmes à capacité est le passage de capacité en paramètre. Une capacité peut être copiée, ses droits restreints et la capacité obtenue passée en paramètre de l'opération de changement de domaine. Ainsi, les systèmes à capacités permettent l'échange dynamique de droits d'accès entre des domaines mutuellement méfiants.

Les systèmes à capacité permettent de répondre à tous les besoins décrits en début de section. Par contre, les systèmes à capacités ont l'inconvénient de répandre les informations de protection dans le système, rendant la gestion des droits et notamment la révocation difficiles à mettre en œuvre. Cependant, des solutions existent.

Deux exemples caractéristiques de systèmes à capacités sont le système Hydra [WUL74], fondé sur des machines spécifiques, et le système Amoeba [MUL86] fondé sur des capacités logicielles. Le problème des systèmes à capacités a longtemps été de n'exister que sur des machines spécifiques, donc chères. Le matériel permettait de contrôler l'allocation et la modification des capacités. Des systèmes à capacités existent aujourd'hui sur des machines banalisées ; Amoeba en est un exemple.

2.3.3. Résumé

J'ai présenté dans cette section les principes qui régissent le contrôle de l'accès à l'information.

Le contrôle d'accès est assuré par deux mécanismes qui visent à assurer le confinement de l'exécution (les domaines de protection) et à authentifier les usagers du système. Dans un environnement réparti, des algorithmes de chiffrement permettre d'assurer la confidentialité des données et d'authentifier les émetteurs des messages échangés entre les machines.

Alors que ces mécanismes permettent d'assurer le respect des règles d'accès, un modèle de protection permet aux applications de définir une politique de protection, déterminant la façon dont les droits sont accordés et échangés entre les application coopérantes.

Le modèle de protection doit permettre d'accorder des droits d'accès à des domaines de protection, ces droits étant décrits en termes de méthodes applicables à des objets. Une fonction de changement de domaine de protection doit être fournie, permettant aux activités d'étendre leurs droits lors de l'exécution d'une opération spécifique, permettant le réalisation de sous-systèmes protégés. On distingue deux modèles de protection, à savoir les modèles à listes d'accès et les modèles à capacités. Les modèles à listes d'accès ont l'avantage de faciliter la révocation des droits et de centraliser l'information de protection, alors que les

modèles à capacités sont attrayants pour leur souplesse, permettant d'exprimer des échanges dynamiques de droits d'accès entre les applications coopérantes.

3. Projets de recherche

Dans cette section, je passe en revue les projets de recherche auxquels j'ai participé. Le but est de donner une vue d'ensemble et chronologique de ces travaux. Les contributions les plus significatives sont brièvement présentées avec des références aux publications les décrivant en détail.

3.1. Historique

Mon travail de recherche s'est inscrit dans le cadre de trois projets :

- Le projet Guide, mené à l'Unité Mixte Bull-IMAG entre 1990 et 1994 sous la direction de Roland Balter et Sacha Krakowiak et dont le but était la conception et la réalisation d'un système réparti à objets. Je me suis intégré à l'équipe Guide en 1990, dans le cadre de mon stage de dernière année de l'ENSIMAG et de mon projet de DEA. J'ai ensuite effectué mon travail de doctorat dans ce même laboratoire sous la direction de Jacques Mossière ; cette thèse a été soutenue en Octobre 1993.
- Le projet Raven, mené à l'Université de Colombie Britannique sous la direction de Gerald Neufeld et Norm Hutchinson. Le but de ce projet était l'étude et la réalisation d'un système adaptable sur une architecture multiprocesseur. J'ai travaillé dans ce projet de Novembre 1993 à Novembre 1994, dans le cadre d'un séjour post-doctoral (bourse INRIA).
- Le projet Sirac, projet commun à l'IMAG et à l'INRIA, sous la direction de Roland Balter et dont le but est de fournir un ensemble d'outils et de services pour la mise en œuvre d'applications coopératives réparties. J'ai rejoint cette équipe en Novembre 1994 en tant que chargé de recherche de l'INRIA. Mes travaux se poursuivent actuellement dans le projet Sirac.

Je rappelle brièvement ci-dessous les principaux thèmes de recherche développés dans ces projets et je donne une présentation générale des systèmes réalisés en indiquant les contributions importantes. J'insiste plus particulièrement sur les projets Guide et Sirac, car ils correspondent à la partie la plus importante de mes travaux.

3.2. Guide

3.2.1. Historique et orientations générales

Le projet Guide [BAL97] a été lancé en fin 1986 comme un projet commun au Laboratoire de Génie Informatique de l'IMAG et au Centre de Recherche de Bull, et a fonctionné dans ce cadre de 1987 à 1989. De 1990 à 1994, le projet a été mené dans le cadre d'un laboratoire mixte Bull-IMAG, sous la tutelle conjointe de Bull, du Centre National de la Recherche Scientifique, de l'Institut National Polytechnique de Grenoble et de l'Université Joseph Fourier. Le projet Guide a bénéficié de son association au projet ESPRIT Comandos [CAH93a], lancé au même moment avec des objectifs analogues.

L'objectif du projet Guide était *la conception et la réalisation d'un prototype préindustriel de système réparti pour le développement d'applications avancées sur un réseau de stations de travail et de serveurs*. Le qualificatif de «préindustriel» s'appliquait à un prototype ayant des qualités suffisantes de robustesse, de fiabilité et de performances pour permettre le transfert total ou partiel des résultats obtenus vers des produits industriels.

Le domaine d'application couvert par le système Guide était celui des *applications coopératives*, caractérisées par une interaction forte entre un ensemble d'utilisateurs, par l'intégration d'applications multiples, et par le partage d'informations complexes persistantes. Des exemples de telles applications sont :

- La gestion de documents, au sens large, y compris des données multimédia (images, etc), ou des ensembles de documents interconnectés (hypertextes).
- Les outils d'aide à la prise de décision dans un groupe (communication par courrier ou par panneaux d'affichage électroniques, gestion d'agendas, etc).
- Le développement de logiciel, notamment la gestion de versions et de configurations complexes.

Un aspect important était l'*intégration* d'un ensemble d'applications (par exemple gestion de logiciel et documentation technique, outils de communication et gestion de données multimédia).

Le choix de base de Guide a été d'organiser les informations comme un ensemble d'objets, unités d'encapsulation de variables associées à une interface d'accès. Cet ensemble d'objets constitue un univers réparti auquel accèdent les processus qui exécutent les applications.

La conception du modèle d'objets de Guide a été orientée par les principes suivants.

- Dissimuler la répartition géographique du système aussi bien pour la localisation des informations que pour l'exécution des programmes.
- Séparer la gestion des objets de celle des processus, en définissant des objets passifs, d'une part, et des activités manipulant ces objets, d'autre part. Ce choix est motivé par la constatation que les applications visées nécessitent, en général, un grand nombre d'objets de petite taille, et qu'il semble peu économique d'associer un ou plusieurs processus à chacun de ces petits objets.
- Fournir un mécanisme uniforme pour le partage d'information et la communication entre activités : ce mécanisme est le partage d'objets.
- Rendre les objets persistants, pour décharger les utilisateurs du souci de la sauvegarde et du chargement explicites des objets.

Le projet a été mené en deux étapes. La première phase du projet Guide (de 1986 à 1990) a abouti à la réalisation d'un prototype appelé Guide-1 [BAL91a]. Deux grands choix ont présidé à cette phase du projet :

- Intégration forte entre un langage de programmation (le langage Guide) et le système d'exploitation (le système sert essentiellement de support d'exécution au langage offrant un modèle de structuration d'applications à base d'objets).
- Réalisation du prototype au-dessus du système Unix (ce choix a permis de bénéficier de l'environnement de développement d'Unix et de réaliser une version du prototype dans des délais acceptables).

Le prototype Guide-1, initialement développé sur Sun 3/60 et Bull DPX1000, a été porté sur Sun 4 (Sparc), DecStation 3100 et 5000, Bull-Zenith/486.

L'objectif de Guide-1 était de montrer la viabilité des principes de conception de l'architecture et leur adéquation aux domaines d'application visés. L'objectif du second prototype Guide-2 (à partir de 1990) a été de montrer que le système peut effectivement fonctionner en vraie grandeur, dans des conditions d'utilisation réalistes, avec des performances acceptables. Les principales orientations de Guide-2 diffèrent de celles de Guide-1 sur les points suivants [HAG94b] :

- Guide-2 a permis d'exécuter des applications écrites dans plusieurs langages (les langages visés sont Guide et une version étendue de C++).
- Guide-2 a été réalisé au-dessus du micro-noyau Mach [ACE86] qui fournit les services de base pour la communication, la gestion des processus et la gestion de la mémoire.
- Guide-2 a une architecture modulaire permettant la réutilisation de composants existants. On visait en particulier la réutilisation de serveurs pour le stockage fiable des objets.

De plus, plusieurs aspects non traités dans Guide-1 ont été pris en compte dans Guide-2 :

- Tolérance aux fautes. Un modèle de transactions permettant une mise à jour atomiques des objets en mémoire permanente a été intégré dans ce second prototype.
- Protection et sécurité notamment. Un modèle de protection permettant de définir des règles de contrôle d'accès aux objets a été intégré dans Guide-2.

Fin 1993, un prototype était disponible sur des PC à base de processeurs Intel 80x86.

La réalisation de plusieurs applications de taille significative a permis de montrer la validité d'un modèle de système réparti fondé sur des objets persistants partagés. Ces réalisations ont mis en évidence la commodité et la sécurité de l'environnement Guide, et la facilité de développement et de modification des applications. Au total quelques 300.000 lignes de code en langage Guide ont été développées dont la moitié à l'extérieur de l'équipe. Les réalisations les plus significatives ont porté sur les applications suivantes : un éditeur coopératif de documents structurés, construit à partir de l'éditeur Grif [DEC93], un tableur partagé réparti, un service de désignation conforme à la norme X-500 (réalisé par l'équipe "administration de système" de Bull), et une application de circulation automatisée de documents (réalisée par le CNET-SEPT). Par ailleurs, le système Guide a également servi de support pour l'enseignement des systèmes à objets dans plusieurs universités.

Enfin, soulignons que les résultats du projet Guide ont été utilisés dans le cadre d'un transfert industriel sous la forme d'une plate-forme pour le support d'applications réparties à objets (projet OODE).

3.2.2. Présentation du système

Cette section contient une présentation générale du système Guide, puis passe en revue les choix essentiels de mise en œuvre du second prototype (Guide-2).

Présentation générale

Deux langages à objets développés au sein du laboratoire, Guide [KRA90] et une extension de C++ intégrant la distribution et la persistance appelée OC++ [SAN93], sont disponibles au dessus du système Guide. Une application se compose d'un ensemble de classes développées soit en Guide soit en OC++, chaque classe définissant de manière classique un ensemble de variables d'état et de méthodes permettant de modifier ou de consulter ces variables. Une variable d'état peut contenir un nom d'objet. On peut à partir d'une classe créer des exemplaires d'objets issus de cette classe ; lors de l'exécution d'une méthode définie sur un objet, il est possible de faire appel à une méthode d'un autre objet.

Le modèle d'exécution de Guide est organisé en **domaines**. Un domaine correspond à l'exécution d'une application répartie. C'est un espace d'adressage multi-sites dans lequel les objets sont rendus accessibles dynamiquement par une opération appelée **couplage**. L'ensemble des sites sur lesquels est représenté un domaine et l'ensemble des objets qu'il contient peuvent évoluer dynamiquement. Le couplage permet l'utilisation d'objets préexistant au domaine ou créés par un autre domaine. Des flots d'exécutions concurrents, que nous appelons **activités**, partagent l'espace d'adressage d'un domaine.

Pour créer un domaine, il faut spécifier un objet-programme. Après création, le domaine est constitué d'un espace d'adressage contenant l'objet-programme et d'une activité, dite principale, qui exécute une méthode de nom prédéfini *main*. Cette activité peut à son tour créer d'autres activités dans le domaine. L'exécution d'une activité est une suite d'appels de méthodes sur des objets.

La communication entre les différents domaines est réalisée par partage d'objets. La communication par partage d'objets remplace l'échange explicite de messages entre les processus communicants. A chaque objet est associé un nom unique dans le système. La connaissance de ce nom par une application lui permet d'appeler les méthodes définies sur cet objet, donc de le partager. Des outils de synchronisation [DEC91] sont fournis par le système afin de contrôler les accès simultanés à un même objet par plusieurs activités.

Dans Guide, la persistance des objets est implicite. Chaque objet créé est **persistant**, ce qui signifie qu'il survit à la mort du processus l'ayant créé. Un composant du système, le ramasse-miettes, est chargé de détruire les objets qui ne sont plus utilisés (les objets ou les groupes d'objets qui ne sont plus accessibles depuis les racines de persistance).

Afin de résister à la panne d'un ou de plusieurs sites, les objets potentiellement persistants peuvent être recopiés sur disque. On dit alors que ces objets sont **permanents**. Les images sur disques sont utilisées après une panne de site pour reconstruire l'image des objets en mémoire. La cohérence entre l'image d'un objet en mémoire et son image sur disque n'est

pas assurée par le système, mais par les applications au moyen d'une demande de recopie explicite sur les disques de l'image en mémoire. L'opération de recopie fournie permet la mise à jour atomique des images d'un groupe d'objets sur des disques pouvant être différents.

La figure 3.1 donne une vue globale incluant les abstractions définies par le système, à savoir les domaines, les activités, les objets partagés et le stockage.

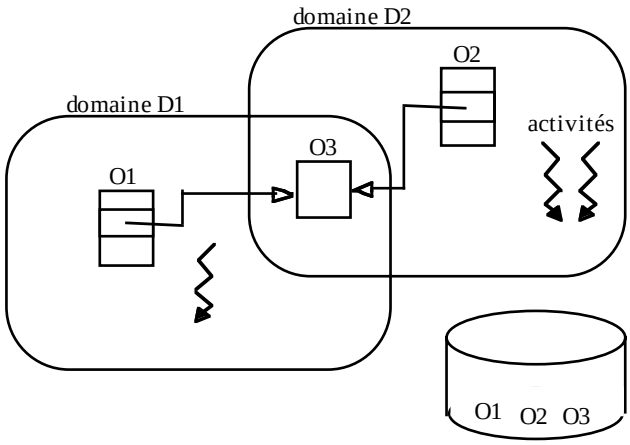

Figure 3.1. Les abstractions du système Guide

Trois objets O1, O2 et 03 sont définis et ont une copie permanente sur disque. O1 et O3 sont couplés dans le domaine D1, O2 et O3 dans le domaine D2. O3 est donc partagé entre les deux domaines. O1 et O2 contiennent tous deux une référence à O3.

Organisation de la mémoire de stockage

Pour faciliter la gestion de la persistance et le traitement des défaillances, la conservation des objets fait intervenir les deux niveaux de mémoire définis en 2.2.1 : mémoire de stockage et mémoire d'exécution.

Avant toute utilisation, un objet doit être transféré de la mémoire de stockage à la mémoire d'exécution ; il est recopié en mémoire permanente soit lorsqu'il n'est plus utilisé, soit sur demande explicite effectuée par une application. Plus précisément, le transfert porte non pas sur un objet unique, mais sur un regroupement d'objets appelé **grappe**.

Nos expériences de programmation d'applications ont en effet montré que la plupart des objets gérés sont de petite taille (moins de 300 octets) ; en conséquence, choisir l'objet comme unité d'accès en mémoire de stockage implique le coût d'un couplage lors de chaque liaison d'objet. Nous regroupons donc les objets dans des grappes. La grappe est l'unité de partage entre les structures d'exécution. C'est aussi l'unité de couplage dans les domaines. La grappe est une notion qui peut être cachée au programmeur, le système gère alors le regroupement des objets dans les grappes. Elle peut également être exploitée par les langages de programmation pour permettre au programmeur de définir sa propre politique de regroupement.

Le mécanisme de désignation du système permet de localiser les objets dans les grappes, tout en autorisant la migration d'objet, c'est à dire le déplacement d'un objet d'une grappe à une autre. La localisation des objets en mémoire de stockage est basée sur un nommage absolu. Des liens de poursuite permettent de localiser les objets déplacés. Un mécanisme supplémentaire permet d'éviter le couplage inutile de la grappe de création d'un objet déplacé lorsque la grappe de résidence de l'objet est déjà couplée [CHE96].

Organisation de la mémoire d'exécution

Pour l'organisation de la mémoire d'exécution, la motivation principale est la protection prenant ici le sens de confinement de l'exécution. Comme nous ne souhaitons pas faire d'hypothèse sur les langages d'écriture des applications, nous ne pouvons pas faire reposer la protection dans Guide sur un contrôle statique effectué à la compilation. En conséquence, nos structures d'exécution et de partage doivent fournir un contrôle dynamique des accès à l'exécution.

Notre objectif a donc été d'assurer l'isolation entre les programmes et les objets, et plus précisément :

- l'isolation entre domaines : une erreur d'adressage dans un domaine ne doit pas provoquer d'erreur dans un domaine avec lequel il ne partage aucun objet.
- l'isolation entre objets : dans un domaine, une erreur dans un objet ne doit pas atteindre un autre objet. À défaut de pouvoir mettre en œuvre cette règle de façon stricte, nous nous proposons d'assurer au moins l'isolation entre les objets de propriétaires différents.

Dans ce qui suit, nous utilisons le terme de *tâche* pour désigner des espaces d'adressage gérés à partir des mécanismes de mémoire virtuelle des machines actuelles[4].

Afin d'assurer l'isolation entre les domaines, les domaines ne partagent pas de tâches. Chaque domaine est implanté par une tâche sur chacun des sites où il est représenté. Le partage d'objets entre les domaines est mis en œuvre en couplant dynamiquement les objets dans l'espace virtuel de la tâche associée au domaine qui désire y accéder. Chaque activité de Guide est également réalisée par un ensemble de processus[5], un par machine. Chacun de ces processus s'exécute dans la tâche appartenant au domaine sur cette machine. Le processus représentant d'une activité est créé lors du premier appel, et réutilisé lors des appels ultérieurs. L'interaction entre les deux représentants de l'activité prend la forme d'un appel de procédure à distance (RPC) réalisé par échange de messages.

Pour assurer l'isolation entre les objets, nous interdisons (à l'intérieur d'un domaine) le couplage dans la même tâche d'objets appartenant à des propriétaires différents. Ainsi, une erreur d'adressage malencontreuse dans une méthode ne peut affecter que des objets appartenant au même propriétaire. Un domaine est réalisé sur chaque machine par un

---

[4] Nous choisissons ce terme parce qu'il correspond à l'abstraction effectivement utilisée sur le micro-noyau Mach, qui a servi de base à la réalisation du système Guide.

[5] un processus léger ou *thread*.

ensemble de tâche, chacune d'elles correspondant à un propriétaire dont il utilise au moins un objet. Comme l'unité de couplage n'est pas l'objet mais la grappe, ceci implique que tous les objets d'une même grappe appartiennent au même propriétaire. Un appel de méthode entre des objets appartenant à des propriétaires différents est réalisé par un mécanisme d'appel à distance entre les deux tâches, qu'ils appartiennent au même site ou à des sites différents. Bien que l'isolation au niveau de l'objet ne soit pas strictement appliquée, nous avons constaté que l'isolation par propriétaire constitue un bon compromis entre fonctions et efficacité.

Adressage

Pour être utilisé, un objet doit être adressable par une adresse virtuelle. A l'exécution, il faut donc être en mesure de traduire les noms d'objets en adresses virtuelles.

Nous avons rejeté l'approche qui consiste à associer statiquement (à la création) une adresse différente à chaque grappe. Cette approche nécessite de disposer de grands espaces virtuels. Au moment de la conception de Guide, les processeurs munis d'espaces d'adressage à 64 bits n'étaient pas encore généralisés.

Une grappe est donc couplée à des adresses virtuelles différentes dans les différentes tâches et la traduction des noms d'objets est locale à chaque tâche. Pour n'effectuer cette traduction que lors du premier appel à un objet dans une tâche, nous avons utilisé un schéma de liaison analogue à celui de Multics [ORG72].

Dans chaque tâche dans laquelle un objet *O1* est couplé, une zone de mémoire est associée à *O1*. Cette zone, appelée **segment de liaison**, est construite à la première utilisation de l'objet dans la tâche. Elle est gérée comme une table qui contient une entrée par référence externe (nom d'objet) contenue dans *O1*. Cette entrée est destinée à recevoir l'adresse dans la tâche courante de l'objet référencé.

Chaque accès à *O2* depuis *O1* (*O1* contenant une référence à *O2*) est réalisé par indirection à travers l'entrée associée à cette référence externe dans le segment de liaison de *O1*. L'entrée en question est mise à jour (on dit que la référence est **liée**) lors de la première utilisation de la référence externe et elle reste valide pour les futurs accès. La liaison d'une référence est réalisée par le système. Lorsqu'on utilise une référence déjà liée, l'appel de la méthode s'effectue sans intervention du système. Cette technique peut être vue comme une mutation de pointeur (*swizzling*) paresseuse sur une zone mémoire locale à chaque tâche.

Une présentation plus complète de ces mécanismes d'adressage ainsi qu'une évaluation basée sur des applications significatives est disponible dans [CHE93].

Protection

Les objets étant tous potentiellement partageables, des mécanismes de protection sont nécessaires afin de contrôler les droits des usagers sur les objets. Ces mécanismes doivent répondre aux besoins identifiés en 2.3.2. Nous avons en particulier mis l'accent sur :

- l'expression des droits d'accès en termes de méthodes,
- la délégation de droits.

Pour le contrôle des droits d'accès, Guide définit la notion de **vue**. Une vue, définie dans une classe, est un ensemble de méthodes autorisées. Une liste d'accès associée à chaque objet donne pour chaque usager la vue que cet usager a sur cet objet, ce qui définit un ensemble de méthodes autorisées sur l'objet pour cet usager. L'implantation des vues ne modifie pas le schéma d'adressage de Guide ; la liaison d'une référence à un objet dépend (est fonction) de la vue que l'usager courant a sur cet objet [HAG94a]. Cette liaison étant effectuée dans une tâche associée au propriétaire de l'objet, un usager ne peut contourner la protection que pour ses propres objets. Le contrôle des accès effectués par d'autres usagers est garanti par la séparation entre les tâches.

La fonction de délégation de droits est généralement utilisée pour fournir des sous-systèmes protégés de leurs clients et devant résister aux tentatives de fraude. Dans Guide, la seule protection sûre est la séparation des espaces d'adressage des tâches, utilisée pour protéger des objets de propriétaires différents. Le principe de notre mécanisme est de prendre en compte dans l'expression des droits le propriétaire de l'objet appelant. A chaque objet est attaché un **attribut de visibilité**, qui selon sa valeur (visible, invisible) indique si l'objet appartenant à un propriétaire *P* peut ou non être appelé depuis un objet appartenant à un autre propriétaire que *P*. Par défaut, cet attribut est positionné à la valeur "invisible". Un objet *O* (appartenant à *X*) joue le rôle de guichet pour un ensemble d'objets appartenant à *X* lorsque l'ensemble des objets est invisible et que *O* est visible. Un sous-système protégé peut être géré en lui associant un nom d'usager (propriétaire) et en définissant les objets guichets (visibles) du sous-système. Les listes d'accès des objets déterminent les usagers autorisés à utiliser le service. Ce mécanisme est comparable à celui des guichets de Multics, mis à part que les domaines n'imposent aucune hiérarchie entre les services protégés (les anneaux de Multics le font) et sont associés à des propriétaires d'objets.

Ces deux mécanismes de protection permettant le contrôle d'accès par usager ainsi que la délégation de droits sont détaillés dans [HAG94a].

3.2.3. Pour plus de détails

Pour plus de détails sur la conception du prototype Guide-2, le lecteur est invité à consulter les articles suivants, dont les plus significatifs sont donnés en annexes de ce rapport :

- [CHE96] pour la conception globale
- [HAG94b] pour une comparaison aux systèmes contemporains
- [CHE93] pour les mécanismes d'adressage
- [HAG94a] pour les mécanismes de protection
- [BAL93] pour la conception sur le micro-noyau Mach

- ainsi que les rapports de thèse de
    - D. Hagimont [HAG93]
      Adressage et protection dans un système réparti
    - P-.Y. Chevalier [CHE94]
      Persistance et Disponibilité dans les Systèmes Répartis

3.3. Raven

J'ai rejoint le projet Raven de l'Université de Colombie Britannique en Novembre 1993 au lendemain de ma thèse, dans le cadre d'un séjour post-doctoral.

Deux thèmes étaient particulièrement étudiés dans cette équipe.

Le premier concerne la programmation d'applications réparties et plus spécifiquement l'expression de la concurrence dans les langages à objets. Un langage avait été défini et son compilateur réalisé. Ce langage offre de nombreuses caractéristiques comparables à celle du langage Guide.

Je détaille un peu plus le second thème sur lequel j'ai travaillé durant ce séjour.

Le deuxième thème étudié était la conception et la réalisation d'un micro-noyau de système pour une machine multiprocesseur appelée *Hypermodule*. L'Hypermodule était une machine à base de quatre processeurs 88100 partageant une mémoire commune. Ce noyau [RIT93] devait fournir un support adapté à l'exécution d'applications parallèles développées à l'aide du langage Raven.

L'objectif principal qui a orienté la conception de ce micro-noyau a été de sortir aux maximum les services systèmes du noyau et de les réaliser au niveau utilisateur sous forme de librairies liées avec l'application. Réaliser un service système au niveau de l'application permet à l'application de spécialiser ce service en fonction de ses besoins et ainsi de le gérer plus efficacement. Par exemple, une application a la possibilité de gérer un protocole réseau personnalisé.

Plusieurs services systèmes ont été réalisés de cette façon :

- Des traitants d'interruption chargés de piloter les périphériques. Notamment, les pilotes de l'interface SCSI (pour l'accès aux disques) et de la carte Ethernet ont été réalisés ainsi.
- L'ordonnancement de processus légers. La gestion de processus a également été réalisée au niveau des applications, chaque application pouvant gérer ses processus légers à sa façon.
- L'échange de messages entre processus. Un mécanisme d'échange de message spécialisé entre processus locaux à une machine a été réalisé en utilisant la mémoire partagée pour réduire les copies de données et en limitant les interactions avec le système.

Les résultats de ces travaux ont montré qu'il était possible de gérer la plupart des services systèmes en dehors du noyau et ainsi de rendre ces services adaptables aux besoins des

applications. Récemment, cette idée a été développée plus profondément dans le projet Exokernel [ENG95] au MIT.

3.4. Sirac

3.4.1. Historique et orientations générales

J'ai rejoint le projet Sirac à la fin de mon séjour post-doctoral en Novembre 1994.

Le projet Sirac est un projet commun à l'INRIA, à l'Institut National Polytechnique de Grenoble, à l'Université Joseph Fourier et à l'Université de Savoie. Sirac a été créé à la fin du projet Guide et s'inscrit dans la continuité de Guide pour ce qui concerne l'objectif global, à savoir la conception et la réalisation d'un environnement de développement et d'exécution d'applications réparties. Les thèmes de recherche du projet Sirac sont motivés, d'une part par l'expérience et le bilan du projet Guide, et d'autre part par la prise en compte de certaines évolutions technologiques marquantes. Trois axes de recherche principaux sont développés dans le projet Sirac :

- Construction d'applications réparties [BEL96]. L'objectif est de fournir des outils répondant à deux besoins : a) construire des applications réparties en combinant des techniques de programmation à base d'objets et des techniques d'intégration de composants ; b) faciliter l'administration, la configuration et l'évolution de ces applications. Les applications coopératives constituent un domaine d'application privilégié pour l'expérimentation et la validation des outils.
- Gestion de la mobilité dans l'Internet. L'objectif est de fournir des protocoles et services systèmes pour faciliter l'accès à l'Internet à partir de postes mobiles. Cette activité vise à fournir des mécanismes permettant de supporter simultanément plusieurs interfaces de réseaux sur une station mobile, dans le but d'utiliser le réseau le plus adapté aux caractéristiques du flot de données de chaque application. Cette souplesse d'utilisation est rendue possible par extension des fonctions du protocole IP Mobile.
- Support système pour serveurs d'information [DEC96b]. L'objectif est de fournir un support générique et efficace utilisable pour la construction de plates-formes à objets répartis et de serveurs d'objets, en utilisant une mémoire virtuelle partagée répartie. Sa conception tient compte des nouvelles infrastructures matérielles (grandes mémoires virtuelles et réseaux rapides) et de l'évolution des concepts (nouvelles formes de gestion mémoire).

Etant donné que ma contribution dans le projet Sirac se situe dans le dernier axe (appelé Arias), je détaille maintenant son contenu.

L'objectif d'Arias est de réaliser une mémoire virtuelle partagée répartie dans laquelle les processus utilisateurs peuvent partager des objets comme s'ils disposaient d'une mémoire physique commune. Les applications visées par ces services sont les systèmes à objets tels de Guide, les systèmes de gestion de base de données ou les serveurs d'information répartis.

Une attention particulière est portée sur les serveurs d'information exécutés sur une architecture répartie de type *cluster* (grappe de machines). Le serveur peut ainsi bénéficier des

ressources de toutes les machines incluses dans le cluster. De plus, la configuration du cluster peut évoluer dynamiquement par ajout, retrait ou remplacement des machines le composant. Enfin, la mémoire virtuelle répartie permet au serveur de s'exécuter comme sur une seule machine. Des exemples concrets de serveurs sont les serveurs WEB ou les gestionnaires de bases de données.

Le premier prototype a été réalisé sur une grappe de machines homogènes reliées par un réseau local. Deux principes directeurs ont guidé la définition de ce prototype :

- Intégration forte dans l'existant. Nous avons décidé de fournir un service système pour le partage d'information intégré au système Unix, minimisant ainsi l'effort de répartition des applications (serveurs) existant sur Unix.
- Système "à la carte". Nous avons mis l'accent sur la possibilité pour les applications de spécialiser les services que nous proposons en fonction de leurs besoins. Nous permettons aux applications de mettre en œuvre leur propre politique de gestion mémoire, de synchronisation, de journalisation et de protection des données partagées.

Nous avons plus précisément dirigé nos recherches suivant 4 axes :

- Grands espaces d'adressage. L'arrivée des processeurs à adressage 64 bits rend possible la gestion d'un espace virtuel unique. Cela signifie que les objets manipulés sont couplés dans la mémoire à des adresses virtuelles fixes (quels que soient les processus). On peut ainsi utiliser les adresses virtuelles comme identifiants uniques de ces objets et améliorer les performances, car on évite toute traduction des identifiants en adresses virtuelles lors de l'exécution.
- Gestion mémoire. La plupart des systèmes à mémoire virtuelle répartie fournissent un seul protocole de gestion mémoire (cohérence, chargement, pagination). Cependant, il a été montré que l'on peut améliorer les performances de façon significative en fournissant aux applications la possibilité de spécifier leurs propres protocoles de gestion de la mémoire partagée. Ainsi, nous permettons aux applications d'adapter le comportement du système en fonction du type des objets qu'elle manipule (table dans une base de données, document multimédia, hypertexte).
- Tolérance aux pannes. De façon similaire, la gestion de données stables sur disque passant par la gestion de journaux (pour l'atomicité des mises à jour), nous offrons aux applications la possibilité de spécifier leurs propres protocoles de journalisation, c'est à dire les méthodes d'enregistrement dans les journaux et de recouvrement en cas de panne.
- Protection. Les données étant partagées entre tous les usagers, elles sont protégées par des mécanismes de protection. La protection utilise un schéma classique à base de domaines et de capacités logicielles. Cependant, nous permettons de spécifier la politique de protection d'une application à l'extérieur de l'application, le code de l'application restant ainsi indépendant de la protection.

Un des points forts de ce travail est l'intégration de plusieurs technologies dans un même système. Le prototype Arias prend à la fois en compte les problèmes d'efficacité du partage, de tolérance aux pannes et de protection des données. De plus, Arias a été intégré au système

Unix AIX sur des machines Bull de type Escala et Estrella sans modification du système (les extensions sont installées dynamiquement dans AIX).

Une première application a permis d'expérimenter le prototype Arias. Il s'agit d'un système de gestion de fichiers répartis (*Cluster File System*, ou CFS) [FAS96]. CFS utilise un cache de données réparti sur l'ensemble des serveurs d'une grappe. Les premières mesures montrent des performances supérieures, dans beaucoup des cas courants, à celles du système NFS, avec une souplesse d'utilisation et une facilité d'administration supérieures à celles de NFS. Une deuxième application en cours de développement est le support d'une base de donnée orientée-objet sur la mémoire répartie Arias. Une collaboration avec O2 [DEU91] a débuté.

Notons enfin que ces travaux font l'objet d'un transfert industriel dans le cadre de l'action Mescaline du GIE Dyade.

3.4.2. Présentation du système

Mémoire virtuelle répartie partagée

La mémoire partagée gérée par Arias [DEC96a, DEC96b] est accessible depuis tous les processus Unix du système. L'espace virtuel de chaque processus est composé de deux parties : la première est gérée par Unix et pour les fonctions d'Unix ; la seconde est réservée par Arias et contient la mémoire virtuelle partagée d'Arias. Les adresses dans cette seconde partie sont allouées par Arias et sont uniques dans le système. Notre service gère donc un espace virtuel unique dans une partie de l'espace d'adressage des machines.

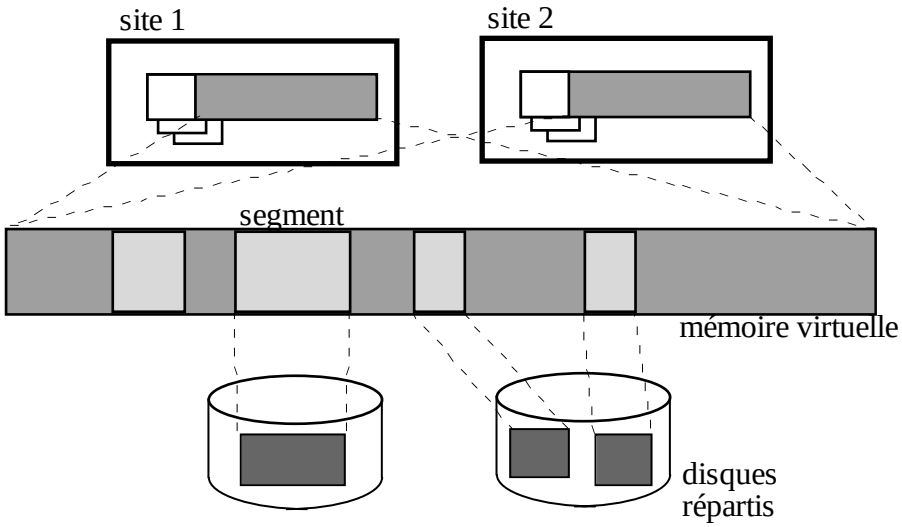

Figure 3.2. La mémoire virtuelle partagée persistante répartie Arias

L'unité d'allocation de cette mémoire est le *segment* qui est une suite de pages contiguës [HAN96]. La taille d'un segment est fixée à sa création. Les segments sont gérés de manière persistante (leur durée de vie n'est pas liée à celle du processus qui les a créés). Un segment est désigné par son adresse qui reste inchangée pendant toute sa durée de vie. Les segments doivent être explicitement détruits. Le système garantit que la portion de

mémoire occupée par un segment détruit ne sera jamais réallouée et que toute tentative d'accès à un segment détruit donnera lieu à une erreur. La mise en œuvre d'un "ramasse-miettes" est laissée à la charge des applications qui seules peuvent interpréter le contenu des segments.

À l'exécution, la gestion de la mémoire partagée repose sur les mécanismes de pagination des machines. En raison du partage des segments, des accès concurrents à une même page peuvent être effectués sur plusieurs sites simultanément, cette page étant dupliquée dans la mémoire de chacun des sites. Le système ne gère pas la cohérence entre les différentes copies d'une même page mais il fournit des mécanismes permettant aux applications de mettre en œuvre leur propre politique de cohérence.

Gestion de la mémoire

La motivation principale qui a orienté la conception d'Arias est de ne pas figer les politiques de gestion des ressources, en particulier pour la gestion de la mémoire et plus précisément pour la cohérence [PER95, PER96]. Ainsi, Arias permet aux applications de définir et installer leur propre protocole de gestion de la cohérence associé aux données qu'elles gèrent.

Les mécanismes offerts reposent sur les constatations suivantes :

- Le segment et même la page sont des unités de grain trop gros pour être choisies comme unité de contrôle d'accès (en terme de synchronisation) et de cohérence. Il est donc nécessaire de fournir une unité de grain plus fin si on veut éviter les problèmes dus au faux partage (processus accédant à des zones disjointes de la même page).
- Les applications partageant des données potentiellement accessibles simultanément par plusieurs processus synchronisent leurs accès à ces données. Il est donc possible de lier la synchronisation et la mise en cohérence des données et notamment de retarder l'application du protocole de mise en cohérence d'une zone de données lors de la demande de verrou sur cette zone.

Les mécanismes de mise en cohérence gèrent des *zones* qui sont des suites d'octets contigus. Toute zone a une copie particulière dite copie *maîtresse*. À tout instant le système sait localiser la copie maîtresse de la zone et fournit des primitives permettant de mettre à jour une copie de zone à partir de sa copie maîtresse, ou de changer le site de résidence de la copie maîtresse d'une zone.

Ce jeu de primitives permet aux applications de mettre en œuvre leur propre politiques de gestion de cohérence. Par ailleurs, nous fournissons un certain nombre de protocoles "standards" : c'est le cas pour le protocole de cohérence faible "entry consistency" [BER93] (la zone est mise en cohérence forte lors de la prise d'un verrou d'accès) marié d'une part à une politique de synchronisation de type lecteurs/rédacteurs et d'autre part à une politique de verrouillage transactionnelle à deux phases.

Permanence

Un segment de cette mémoire peut être rendu *permanent*, c'est à dire avoir une image sur un support permanent, disque ou autre, ce qui lui permet de résister aux pannes du système. À l'opposé, les segments non permanents sont dits *volatils*. Deux mécanismes sont fournis

pour la gestion de la permanence [KNA96, KNA97]. Le premier permet de rendre permanente l'image d'un segment (le segment passe alors de l'état volatil à l'état permanent). L'atomicité est offerte par le deuxième mécanisme, en l'occurrence la journalisation. Le système permet de stocker des informations dans des journaux gérés sur disque. L'utilisateur peut enregistrer une liste d'enregistrements dans un journal, puis après validation du journal utiliser ces enregistrements pour modifier l'image permanente de la mémoire. En cas de panne, le journal permet de reconstruire une image cohérente des modifications qui y ont été validées. Les enregistrements non validés dans le journal sont perdus. Remarquons que le format des enregistrements de modification et les opérations de répercussion de ces modifications sont effectuées sous le contrôle des applications. En effet, les applications contrôlent le format et la sémantique des enregistrements écrits dans le journal, ainsi que l'opération d'exploitation du journal lors de la reprise d'un site après une panne. Une application peut ainsi gérer un journal "avant" (sauvegardant les anciennes valeurs validées) ou "après" (mémorisant les modifications validées). L'application contrôle également la mise à jour des images permanentes des segments qui peut être effectuée de manière asynchrone implicitement ou synchrone explicitement.

Dans ce système, nous distinguons comme dans Guide deux niveaux de mémoire : la mémoire d'exécution (dans laquelle les segments sont manipulés par les applications) et la mémoire permanente. Cette distinction permet de toujours maintenir, même pendant l'exécution, une version cohérente des segments en mémoire permanente.

Pour faciliter leur administration, les images permanentes des segments sont regroupées dans des unités logiques de stockage appelées volumes. Un volume est géré par un seul serveur à un instant donné. Il peut néanmoins être déplacé vers un autre serveur au cours de son existence. Pour assurer une plus grande disponibilité des segments, les volumes gérés par un serveur pourront être dupliqués sur d'autres machines.

Protection

Les concepts sur lesquels reposent le service de protection sont ceux de domaine de protection et de capacité. Un domaine de protection définit un ensemble de segments accessibles et pour chacun de ces segments les opérations permises. Un mécanisme du système assure que les processus s'exécutant dans un domaine accèdent aux segments du domaine en respectant les droits définis. Une capacité est une structure de données qui intègre le nom d'un segment (une adresse virtuelle) et un ensemble de droits d'accès à ce segment [SAU96]. Les droits sur un segment sont exprimés en termes des opérations lire, écrire et exécuter.

La mémoire virtuelle répartie est partagée par l'ensemble des processus Unix utilisant le service de données partagées. À un instant donné, un processus Unix s'exécute dans un domaine de protection qui définit son contexte courant d'adressage de la mémoire virtuelle partagée répartie. Deux domaines de protection différents peuvent partager des segments avec les mêmes droits ou des droits différents.

Afin de permettre l'extension des droits d'un processus, le système permet de définir des capacités de changement de domaine de protection. Un domaine peut exporter vers les autres domaines des capacités de changement de domaine permettant d'appeler un sous ensemble

des procédures qu'il contient. Les procédures exportées sont appelées points d'entrée d'un domaine. La tentative d'appel dans un domaine *D* d'un des points d'entrée du domaine *D'* provoque le changement de domaine du processus qui a effectué l'appel si *D* contient une capacité de changement de domaine autorisant l'opération. Un changement de domaine s'accompagne en général d'un transfert de capacité du domaine appelant vers le domaine appelé pour les paramètres d'appel et du domaine appelé vers le domaine appelant pour les résultats.

Lors de la définition de ce modèle de protection, notre motivation principale fut de séparer la définition de la politique de protection de l'application du code de celle-ci [HAG96b], pour des raisons de modularité, de simplicité et de réutilisabilité. Dans Arias, le code des applications est totalement indépendant des capacités. La validité des accès aux segments est vérifiée automatiquement par le système lorsque l'application accède aux segments. De même, un changement de domaine n'est pas explicitement appelé par une application, mais a lieu implicitement lorsque le segment contenant la procédure ne peut pas être couplé localement.

Le service de protection fournit un langage d'interface (IDL) qui permet de spécifier pour un point d'entrée la nature des capacités à transférer de l'appelant vers l'appelé et réciproquement. Ainsi, la politique de protection est entièrement définie par des domaines de protection et des capacités, l'application étant seulement composée de procédures sans programmation de capacités.

3.4.3. Pour plus de détails

Pour plus de détails sur la conception d'Arias, le lecteur est invité à consulter les articles suivants, dont les plus significatifs sont donnés en annexes de ce rapport :

- [DEC96b] pour une présentation générale
- [PER95] pour la gestion de la mémoire
- [HAG96b] pour le modèle de protection
- [KNA97] pour la gestion du stockage
- ainsi que les rapports de thèse de
    - A. Knaff [KNA96]
      Conception et réalisation d'un service de stockage fiable et extensible pour un système réparti à objets persistants
    - F. Saunier [SAU96]
      Protection d'une mémoire virtuelle répartie par capacités implicites
    - E. Pérèz [PER96]
      La cohérence sur mesure dans une mémoire virtuelle répartie partagée
    - T. Han [HAN96]
      Motivation, conception et réalisation d'une mémoire partagée répartie

3.5. Travaux de recherche plus récents

Dans le cadre du projet Sirac, les résultats des travaux autour de la protection dans Arias ont été appliqués à d'autres environnements de systèmes répartis. Il s'agit des environnements CORBA et Java :

- Nous avons réalisé une version du modèle de protection à base de capacités logicielles cachées sur le système Orbix [ION94] qui est une implantation de la spécification CORBA. Les résultats [HAG97b] ont montré l'adéquation du modèle de protection à ce type d'environnement.
- Nous avons également réalisé une version du modèle de protection à base de capacités logicielles cachées pour des applications à base d'agent dans l'environnement Java [GOS95]. Dans ce contexte, un agent mobile est un processus se déplaçant entre des machines avec son contexte d'exécution afin de réaliser la tâche spécifiée par son programme. Les applications visées incluent la recherche d'information sur le Web et les outils de coopération entre usagers de l'Internet. Un des problèmes fondamentaux avec les agents mobiles est de protéger les sites visités contre des agents malveillants. En particulier, il est important d'être capable de protéger les agents les uns des autres. Les capacités logicielles se prêtent également bien à ce type d'applications [HAG97a].

Ces dernier travaux autour de Java et de l'Internet m'ont permis de m'ouvrir les perspectives de recherche qui sont décrites dans la section 5.

3.6. Conclusion

Les travaux de recherche que j'ai effectué dans les différents projets présentés ci-dessus m'ont permis d'étudier les principes de mise en œuvre de l'accès et du contrôle d'accès à des informations réparties. Ces travaux ont essentiellement pris place dans les deux projets successifs Guide et Sirac.

Dans la section suivante, je présente une évaluation de ces travaux de recherche.

4. Evaluation

Dans cette section, je propose une évaluation des travaux de recherche qui ont été présentés au chapitre précédent. Respectivement pour les systèmes Guide et Arias, je dresse un bilan des résultats obtenus et je compare ces systèmes à ceux développés à la même époque.

4.1. Guide

4.1.1. Bilan

Les expérimentations conduites dans le cadre du projet Guide ont prouvé la validité de nos convictions initiales, tant sur le plan de la commodité d'écriture des applications que sur celui de l'efficacité d'exécution. Les résultats ont montré qu'il était possible de gérer un espace d'objets répartis avec des performances acceptables sur des noyaux de systèmes standards largement utilisés.

Commodité

Sur le plan de la commodité d'écriture des applications, le développement d'applications de tailles significatives a montré l'intérêt d'offrir :

- un langage à objets de haut niveau. Les caractéristiques principales du langage Guide sont l'absence de pointeurs, la séparation stricte entre types et classes et la synchronisation par variables internes partagées.
- le partage comme moyen de communication. Le partage d'objet est un mécanisme plus naturel que l'envoi explicite de message pour établir une coopération entre des applications.
- une gestion implicite de la répartition. Dans Guide, tout objet peut être utilisé depuis n'importe quel site sans que le programme prenne en charge une opération d'adressage ou de localisation (un programme ne manipule que des variables contenant des noms uniques d'objets).
- une gestion implicite de la persistance. Les objets dans Guide sont implicitement persistants. Un ramasse-miettes est chargé de récupérer les ressources utilisées par les objets qui ne sont plus accessibles.
- un modèle de contrôle d'accès intégré au modèle à objets. Les vues dans Guide permettent de spécifier les droits d'accès en termes des méthodes que l'on peut appeler sur des objets.

Efficacité

Le prototype réalisé (Guide-2) a montré qu'il était possible de réaliser un environnement de programmation d'applications réparties orienté-objet avec des performances

acceptables [HAG93, CHE94a]. Cette efficacité a principalement été obtenue en utilisant deux techniques :

- la segmentation avec liaison au premier accès. Lorsqu'une référence à un objet est liée dans un espace virtuel, l'adresse virtuelle obtenue est conservée dans un segment de liaison, évitant ainsi les liaisons successives de la même référence. Des mesures ont montrées que l'on économise ainsi de nombreuses opérations de liaison.
- le regroupement d'objets dans des grappes. Alors que l'unité de désignation est l'objet, l'unité de couplage dans les domaines est la grappe, ce qui factorise les opérations de couplage (qui sont chères) pour les objets inclus dans ces grappes.

Points faibles

Plusieurs points faibles ont été relevés dans le travail réalisé dans le cadre du projet Guide :

- un nouveau langage. L'introduction d'un nouveau langage (le langage Guide) a probablement limité (au moins dans la première phase du projet) l'utilisation de l'environnement par des partenaires externes au projet. Il eut été plus réaliste de fournir un langage plus proche de C++, ce qui fut le cas pour le deuxième prototype de Guide qui supportait à la fois les langages Guide et une extension de C++.
- récupération d'applications existantes. En développant des applications réparties sur le système Guide, nous avons souvent été confrontés à la difficulté de récupérer des composants logiciels complexes sans les modifier ou avoir à les reprogrammer. Des outils pour intégrer des composants existants sont nécessaires et ont fait l'objet de travaux dans le projet Sirac, succédant au projet Guide.
- programmation de la protection. Dans Guide, les mécanismes de protection sont basés sur des listes d'accès associées aux objets. La définition d'une politique de gestion des droits d'accès, intégrant des règles d'échange de droits entre des usagers, doit se faire explicitement dans le code de l'application gérant ces objets. L'idée de séparer l'expression d'une politique de contrôle d'accès du code de l'application protégée nous a conduit à définir le modèle de protection à capacités cachées intégré dans le système Arias.
- gestion des regroupements d'objets. Le système Guide fournit des fonctions pour gérer des regroupements d'objets appelés grappes, mais cette gestion est laissée à la charge des applications. Ici aussi, la définition d'une politique de regroupement est un problème complexe et une gestion par les applications débouche dans la plupart des cas sur des politiques simplistes et inefficaces. Sur ce sujet, nous manquons d'expérience permettant de définir des politiques génériques ou de concevoir des outils aidant les applications à définir des politiques adaptées aux objets qu'elles gèrent.

Comme on peut le voir ci-dessus, le projet Guide a ouvert de nombreuse perspectives dont certaines sont traitées dans le cadre du projet Sirac.

4.1.2. Comparaison à l'existant de l'époque

Dans le milieu des années 80, au moment du démarrage du projet Guide, la conviction que le modèle de structuration à base d'objets serait seul capable de répondre aux exigences du développement des grosses applications logicielles était déjà largement répandue dans la communauté. Aussi un nombre important de recherches ont été menées pour étudier, du point de vue tant langage que système, les mécanismes à fournir pour la mise en œuvre efficace de telles applications. Les solutions proposées peuvent être classées en deux catégories :

- Les machines réparties à un seul langage. Le but recherché est d'intégrer dans un langage à objets existant le parallélisme, la synchronisation des accès aux objets, la distribution sur plusieurs sites des objets et même pour certains la persistance. Les mécanismes systèmes nécessaires à la mise en œuvre de ces nouveaux traits du langage sont implantés par son environnement d'exécution et ne sont pas utilisables par un autre langage. Les précurseurs de cette démarche ont été les projets Eden [ALM85] dont le langage de programmation résultait d'une extension de Concurent Euclid et Argus [LIS85] qui étendait le langage CLU pour permettre l'exécution de transactions réparties. Ces projets ont été suivis de nombreux autres : Emerald [BLA87] (successeur direct d'Eden), Orca [BAL92], Gothic-2 [PUA93]. La première phase du projet Guide, qui a consisté à définir et à mettre en œuvre le langage de programmation à objets persistants partagés et distribués Guide [BAL91a, KRA90] et son environnement d'exécution sur un réseau de stations de travail Unix, s'inscrit dans cette lignée.

- Les plates-formes distribuées à objets. Le but ici est d'offrir des mécanismes de plus haut niveau sémantique que ceux disponibles sur Unix pour la mise en œuvre de la répartition, du parallélisme et de la persistance tout en restant suffisamment générique pour permettre leur utilisation efficace par les environnements d'exécution de plusieurs langages à objets. Le précurseur des systèmes de la seconde catégorie est le système Clouds [DAS90] qui fournit une base de mise en œuvre pour les environnements d'exécution de DC++ (une version distribuée de C++) et d'une version distribuée d'Eiffel. Les systèmes SOS [SHA89], Opal [CHA94], Cool [LEA93], Amadeus [CAH93b], Corba [SOL93], Spring [MIT94] et le système Guide-2 [HAG94b] que nous avons décrit visent également le même objectif.

Par rapport à ces derniers systèmes, le système Guide-2 se démarque sur différents aspects :

- Grain des objets. Des systèmes comme Clouds, Corba ou Spring fournissent des mécanismes qui s'adressent à des objets à très gros grain et supposent que ce seront les compilateurs qui devront prendre en charge complètement la gestion des accès (synchronisation et adressage notamment) aux objets de petite taille. L'approche de Guide a été de fournir une machine à objets beaucoup plus proche de celle requise par les compilateurs des langages.

- Intégration des mécanismes. Une autre approche consiste à fournir un ensemble de mécanismes de base permettant aux applications de mettre en œuvre l'accès à des objets répartis. C'est le cas du projet SOS, fondé sur le principe du mandataire (ou

proxy [SHA86]) et qui fournit par ce biais un outil d'appel à distance. Guide intègre plus fortement la répartition dans les langages de programmation supportés, un appel d'objet restant le même, que l'objet soit local ou distant.

- Adressage. D'autres techniques d'adressage peuvent être utilisées. Par exemple le système Cool utilise la technique de mutation de pointeur (*swizzling*). Cependant, cette solution ne permet pas le partage concurrent, ou elle nécessite alors de coupler les objets à la même adresse virtuelle dans tous les processus. Une autre technique consiste à gérer un espace virtuel unique dans le système comme dans Opal, mais cette solution nécessite de grands espaces d'adressage. Au moment du développement de Guide, les processeurs 64 bits n'étaient pas encore disponibles. Ils ne se sont pas encore généralisés d'ailleurs.

4.2. Arias

4.2.1. Bilan

Dans le projet Sirac, les travaux de recherche autour du système Arias ont permis d'explorer les principes de mise en œuvre d'une mémoire virtuelle répartie pour des architectures de type *cluster*.

L'objectif était de fournir aux applications l'abstraction d'une seule mémoire commune entre des machines reliées par un réseau local. Les applications visées par ces services sont les systèmes à objets tels de Guide, les systèmes de gestion de base de données ou les serveurs d'information répartis. Ces travaux se sont déroulés dans un contexte semi-industriel, donc avec de fortes contraintes quand aux objectifs et à l'environnement d'expérimentation.

Ces travaux ont permis des avancées significatives sur plusieurs axes, même si l'évaluation du système réalisé n'a été que partielle.

Aspects innovants

Dans le cadre d'Arias, nous avons plus particulièrement mis l'accent sur des aspects innovants qui sont l'adaptabilité, la gestion d'un espace virtuel unique et les problèmes de protection dans ce type de système.

- Adaptabilité de la gestion mémoire. Arias permet aux applications d'adapter les protocoles de gestion de la mémoire répartie à des besoins spécifiques en vue d'une amélioration de l'efficacité. Ces protocoles sont intégrés dynamiquement dans le système et gèrent la cohérence et la permanence des données de l'application.
- Gestion d'un espace virtuel unique. La gestion d'un espace virtuel unique permet de désigner les données par des adresses virtuelles qui restent valides dans tous le système et peuvent donc être échangées entre les processus des différentes machines.
- Protection par capacités cachées. Arias implante un modèle de protection à capacité logicielle, permettant aux applications d'échanger des droits d'accès dynamiquement. Ces capacités sont cachées dans le sens où les programmes sont indépendants de toute

politique de contrôle d'accès (les programmes ne manipulent que des pointeurs). La politique de contrôle d'accès est définie au niveau de l'interface de l'application à l'aide d'un langage de description d'interface (IDL).

Efficacité

Les mesures effectuées sur le prototype Arias ont montré de bonnes performances, même si l'outil de développement utilisé (basé sur les Streams) a été très pénalisant. Le prototype a été évalué avec l'application CFS. Des mesures comparatives entre CFS et NFS ont montré un gain significatif de performance en faveur de CFS.

L'adaptabilité n'a été que partiellement démontrée par les applications CFS et O2 qui implantent des protocoles différents : cohérence à l'entrée sans verrouillage pour CFS et cohérence à l'entrée avec verrouillage transactionnel pour O2.

Points faibles

Le principal point faible que je retiens de ces travaux est la difficulté d'évaluation du prototype réalisé. Cette faiblesse a différentes origines :

- Complexité de la tâche. L'intégration d'extensions dans un système Unix est une opération périlleuse et délicate, ce qui rend plus difficiles à la fois la réalisation et l'évaluation. De plus, Arias s'adresse à des applications qui sont également des environnements complexes, comme le système Guide, un système de fichier ou un système de gestion de base de données. En général, nous n'avons prototypé qu'une partie de ces environnements, le reste reposant sur des services systèmes préexistants (par exemple le stockage dans CFS repose sur la gestion de fichier d'AIX). Empiler des services biaise énormément les mesures.
- Contraintes industrielles. Arias a été intégré dans le système AIX (pour des raisons de transfert industriel), mais nous n'avons pas réussi à obtenir le code source de ce système. En conséquence, il nous a été très difficile de faire une évaluation fine du système réalisé sans avoir une vision claire des mécanismes d'AIX que nous utilisions. L'évaluation aurait été facilitée si nous avions prototypé par modification d'un système existant dont nous aurions disposé du code source (comme Linux).

Le second point faible que je soulignerai concerne le modèle de protection à capacités cachées. Dans ce modèle, les changements de domaines de protection sont déclenchés sur des appels de procédure. Il ne nous pas été possible de construire sur le modèle de protection d'Arias un modèle de protection par capacités pour des objets à grain fin (comme dans Guide). Le modèle de protection Arias se prête bien à la gestion d'objets à gros grain comme des programmes, des services systèmes ou des composants logiciels. Cependant, d'un point de vue conceptuel, le modèle s'applique élégamment à des systèmes à objets, ce qui a été démontré par des réalisations : une version des capacités cachées sur Orbix et une sur Java.

4.2.2. Comparaison à l'existant de l'époque

Je compare le projet Arias à des projets contemporains suivant deux critères : l'adaptabilité de la gestion mémoire et la protection.

Gestion mémoire adaptable

A partir de 1993, un grand nombre de projets en système se sont intéressés à l'adaptabilité des systèmes d'exploitation. L'objectif est d'adapter la gestion des ressources aux besoins des applications. Parmi les projets de recherche traitant ce sujet, on peut comparer Arias à deux classes de projet :

- Les projets de noyaux adaptables. Le but est de réaliser un noyau dont l'architecture permet à l'application de réaliser une gestion optimisée des ressources en fonction de ses besoins. Deux grandes approches ont été explorées. La première consiste à permettre d'intégrer du code des applications dans le noyau afin de spécialiser le système. Des exemples sont les systèmes Spin [BER95] et Vino [SEL96]. La deuxième approche consiste à réaliser un noyau minimal et à faire s'exécuter la plus grande partie du système comme une librairie au niveau utilisateur. Ainsi une application peut spécialiser le système en adaptant/remplaçant cette librairie. L'exemple le plus connu est Exokernel [ENG95]. A noter que le projet Raven se situait dans cette approche. Dans les deux approches, le problème principal est la sécurité du système, car il faut s'assurer que le code intégré dans le système ne peut pas perturber le bon fonctionnement du système.
- Les projets de mémoires virtuelles réparties à cohérence adaptable. Il s'agit ici d'appliquer l'adaptabilité des systèmes pour spécialiser la gestion de la cohérence dans une mémoire virtuelle répartie. Plusieurs études ont montré qu'aucun protocole de cohérence n'était adapté à toutes les applications. Dans [ANA92], on propose de permettre à l'application de piloter la cohérence à l'aide d'un ensemble de primitives. Par contre, Munin [CAR91, CAR93] propose un ensemble de protocoles parmi lesquels l'application peut choisir.

Arias se situe dans ce contexte général. L'objectif est de permettre aux applications de spécialiser la gestion de la cohérence par intégration de code dans le noyau du système. Les aspects de sécurité du noyau face aux extensions n'ont pas été explorés. Arias fournit une couche générique qui implante les mécanismes de base nécessaires à la réalisation des protocoles de cohérence. Ces protocoles sont installés dynamiquement dans le système à l'installation de l'application. Notons que la réalisation d'Arias s'est faite sans modification du système AIX, ce qui était un impératif pour le transfert industriel. Arias se distingue des autres projets par les applications visées. Il s'agit de serveurs d'information s'exécutant sur une grappe de machines dont le système de gestion de fichiers réparti CFS est un exemple.

Le travail autour d'Arias peut également être comparé à des travaux autour des serveurs répartis de gestion de fichiers :

- Le projet XFS [AND95] propose de réaliser un système de gestion de fichiers bénéficiant d'une gestion globale de la mémoire. CFS partage les mêmes objectifs d'extensibilité qu'XFS. La différence principale est que CFS sur Arias permet de spécialiser la gestion mémoire en fonction des besoins des applications. De plus, Arias fournit des outils génériques qui sont susceptibles d'être réutilisés pour d'autres applications. C'est le cas actuellement avec le système de gestion de base de donnée O2.

- Le projet de systèmes de fichiers extensible sur Spring [KHA93] propose d'empiler des systèmes de gestion de fichiers afin de les spécialiser. On peut par exemple gérer un système de fichiers avec compression des données stockées sur le système de fichiers standard. CFS s'inscrit dans la même lignée.

Protection par capacités

Avec l'arrivée des processeurs à capacité étendue d'adressage, de nombreux projets se sont intéressés à la gestion d'un espace d'adressage unique. Les projets les plus connus sont Opal [CHA94] qui est le précurseur, Angel [MUR93] et Mungi [HEI93]. Lorsqu'un espace d'adressage unique est géré, la protection des objets ne peut plus reposer sur des espaces d'adressage privés comme dans Unix. Tous ces systèmes ont adopté le modèle à capacité pour le contrôle d'accès aux objets partagés.

Le modèle de protection d'Arias se distingue par le fait que les capacités dans Arias sont cachées [HAG96b]. Dans les systèmes mentionnés ci-dessus, les capacités sont rendues visibles au programmeur d'application. Celui ci doit manipuler les capacités et les présenter pour avoir accès à un objet. Dans certains systèmes comme Opal ou Mungi [VOC93], la vérification qu'une capacité autorise un accès est réalisée de façon transparente, mais l'utilisation des capacités de changement de domaine reste explicite dans les programmes ainsi que le passage de capacités en paramètre. A cet égard, Arias offre une transparence totale : l'utilisation des capacités de domaine est implicite (lorsqu'un service ne peut être appelé localement) et le passage de capacité en paramètre est défini au niveau de la capacité de changement de domaine à l'aide d'un IDL.

5. Conclusions et perspectives

Pour conclure mon rapport d'habilitation, je présente tout d'abord quelques réflexions personnelles issues de mes sept années de recherche passées. Je présente ensuite les évolutions du domaine qui sont, de mon point de vue, les plus marquantes et qui orientent mes recherches actuelles. Enfin, je présente mon programme de recherche en détaillant quelques problèmes déjà identifiés et pour lesquels j'évalue actuellement les solutions potentielles.

5.1. Réflexions

Maîtrise des systèmes utilisés

Un premier enseignement concerne les systèmes et logiciels utilisés pour effectuer nos recherches. Ces environnements sont en général très complexes et il est difficile d'en avoir la maîtrise. Le terme "maîtrise" désigne le fait de connaître parfaitement l'environnement système utilisé pour mener les expérimentations, et en particulier les spécifications des matériels et les sources des logiciels composant cet environnement. Il s'agit d'un investissement très lourd et il est important dans un projet de développer une expertise durable autour d'un système d'exploitation. Dans certains cas, cette expertise peut nécessiter des relations privilégiées avec un constructeur de machine assurant également les évolutions du système pour ses machines.

A travers les projets Guide et Sirac, nous avons respectivement travaillé avec des systèmes Unix Bull et Sun, puis sur des micro-noyaux Chorus et Mach, et enfin dans le système AIX. S'il est intéressant d'expérimenter avec des environnements différents afin de les comparer et de développer une expertise critique de ces environnements, je me suis à plusieurs reprises posé la question du temps passé à reconstruire une expertise sur de nouveaux environnements. Il s'agit à la fois d'une richesse et d'une perte de temps. De plus, dans le cadre des expérimentations autour d'Arias, il est étonnant que nous ayons pu travailler en étendant le système AIX sans en avoir la maîtrise. Nos liens avec Bull ne nous ont pas permis d'obtenir les sources de ce système, ce qui a été un élément fort préjudiciable.

Evaluation

L'évaluation d'un système d'exploitation est un problème très complexe qui est même un domaine de recherche en soi. En général, l'évaluation comporte une phase de comparaison à des expérimentations ou des produits similaires, pouvant prendre la forme d'une comparaison de mesures d'efficacité.

De façon schématique, on trouve deux types de projet de recherche en système. Le premier consiste à ajouter de nouveaux mécanismes à un système et à les comparer avec des équivalents dans d'autres systèmes. La seconde consiste à prendre des mécanismes existants dans un système et à les rendre plus efficace en changeant leur implantation. Dans ce second type de projet, l'évaluation est plus facile, puisqu'on compare deux réalisations dans le même environnement de système.

Les travaux de recherche auxquels j'ai participé ont souvent visé à fournir de nouveaux mécanismes, ceci expliquant en partie (seulement) le manque d'évaluation quantitative de ces travaux, en particulier pour le système Arias.

Industrialisation

Un des objectifs d'un travail de recherche peut être de réaliser un transfert industriel des résultats. C'est un des objectifs du projet Sirac qui est un projet de l'INRIA. Le transfert industriel est une opération très délicate et difficile à réussir, demandant des ressources humaines en rapport avec cette difficulté.

J'ai pu participer à deux tentatives de transfert industriel, dans le cadre des systèmes Guide et Arias. Deux approches ont été explorées. La première (Guide) a vu le développement d'un prototype de recherche par des chercheurs dont une version industrielle a ensuite été développée par une équipe d'industriels. Dans la second approche (Arias), une équipe composée de chercheurs et d'industriels a développé un prototype devant avoir des qualités proches de celles d'un produit.

La première approche sépare clairement les objectifs de recherche de ceux de transfert, mais nécessite probablement plus de ressources humaines. La seconde rend implicite le transfert, mais entraîne une confusion dans les objectifs des travaux ; on peut notamment choisir de limiter l'ambition de la recherche afin d'arriver à une proposition réaliste.

Objectifs de la recherche

Il est important d'identifier clairement les objectifs d'un travail de recherche au début même de ces travaux. Il s'agit là d'une des parties les plus délicates et difficiles du travail.

Lors de la première phase du projet Guide (Guide-1), l'objectif était de fournir un environnement pour le développement et l'exécution d'applications réparties. L'accent était porté sur les fonctions fournies, permettant de décharger le programmeur de nombreuses tâches fastidieuses, plutôt que sur les performances du système réalisé.

Pour la seconde phase du projet Guide, l'objectif a été de réaliser un support système plus efficace, en particulier en ce qui concerne l'adressage des objets. Si les objectifs ont été atteints, on peut se poser la question de la validité de ces objectifs. Il aurait probablement été plus judicieux de développer de nouvelles fonctions pour enrichir l'environnement Guide et adresser de nouveaux problèmes.

Le succès du langage et de la machine virtuelle Java a montré que si l'efficacité de l'adressage des objets est important, les problèmes liés à l'hétérogénéité et à la mobilité du code et des objets sont déterminants. L'évolution entre les prototype Guide-1 et Guide-2 devient alors discutable. L'adressage dans le prototype Guide-1 était beaucoup plus proche de celui de Java (avec une interprétation des noms d'objets) que celui de Guide-2.

5.2. Evolutions du domaine

Je présente maintenant les évolutions de domaines qui orientent mes recherches actuelles.

5.2.1. L'Internet

Je pense que l'évolution la plus marquante du domaine est le développement de l'Internet. De façon générale, lorsque l'on mentionne *l'Internet* dans un article ou une discussion, c'est le plus souvent pour évoquer l'évolution vers des réseaux de grandes tailles. Le développement de ce réseau a différentes conséquences dont les plus importantes sont :

- les distances entre les machines. L'Internet permet d'interconnecter des machines très éloignées géographiquement, potentiellement à des extrémités du globe. La connexion entre deux machines traverse en général par un nombre variable de machines intermédiaires (ou routeurs), ce qui implique que les débits et latences de ces connexions sont également très variables. Le caractère variable de la qualité des communications dans l'Internet le distingue des réseaux locaux actuels, pour lesquels la qualité du service rendu ne dépend pas (ou de façon négligeable) des positions relatives des machines sur le réseau.

- le nombre et la nature de ses usagers. L'Internet se développe actuellement à une vitesse telle qu'il sera d'ici quelques années présent dans la plupart des foyers, au même titre que l'électricité, la télévision ou le téléphone (pour ces deux derniers, l'Internet pourra même servir de support à la transmission de l'image et du son). Cela signifie que les applications sur l'Internet devront être utilisables facilement par des usagers non initiés, utilisant des périphériques ne nécessitant qu'un minimum de connaissance de l'informatique. De plus, les usagers devront être en mesure de se connecter à l'Internet et de retrouver le même environnement quel que soit le terminal (et le point d'accès à l'Internet) utilisé.

- la diversité des matériels et des logiciels utilisés. L'Internet relie des matériels de tout type et il est inconcevable d'envisager un réseau de machines homogènes à l'échelle mondiale, même si les PC semblent aujourd'hui écraser le marché. De même, le nombre des logiciels utilisés sur l'Internet pour visiter des sites, éditer des documents, communiquer l'information, en résumé coopérer, grandit avec le nombre des usagers et l'hétérogénéité des matériels utilisés.

- la sécurité des systèmes connectés à l'Internet. Alors qu'un réseau local est généralement associé à une unité administrative à l'intérieur de laquelle règne une certaine confiance entre les entités coopérantes, l'ouverture sur l'Internet nécessite la mise en place de mécanismes de sécurité ne faisant aucune hypothèse concernant les machines, systèmes ou applications impliqués dans une coopération. Par exemple, contrairement à un réseau local, on ne peut plus faire la supposition que tous les systèmes d'exploitation installés sur les machines connectées au réseau sont fiables et implantent une même politique de contrôle des droits d'accès.

En résumé, les dimensions de l'Internet en termes de distances, d'usagers, de matériels, de logiciels et de sécurité posent de nombreux problèmes. Si dans la plupart des cas, ces

problèmes ont été résolus dans le contexte des réseaux locaux, les caractéristiques de l'Internet modifient les données de ces problèmes et requièrent une évaluation fine de ces solutions, voire de nouvelles solutions.

5.2.2. Le Web

L'autre évolution très marquante de ces dernières années est le développement du World Wide Web à l'échelle mondiale. Le Web, structure hypermédia répartie, peut être vu d'un point de vue système comme un système de fichier réparti à grande échelle. Le développement du Web sur l'Internet a nécessité une prise en compte des caractéristiques citées ci-dessus :

- distance entre les machines et nombre des usagers. Les distances entre les machines nécessitent la mise en place de caches conservant l'information à laquelle les usagers accèdent le plus souvent. Etant donné le nombre des usagers utilisant le Web, la gestion d'une politique de cache en cohérence forte n'est plus adaptée. Il n'est pas possible de conserver dans un serveur Web la liste des clients possédant une copie d'une page. Les clients implantent généralement une politique de cache, mais avec une cohérence faible des données.
- nature des usagers. Le Web s'adresse à des usagers non initiés. L'accès au Web se fait au travers de navigateurs Web très conviviaux, permettant de visiter des sites dans le monde entier en suivant simplement des liens entre des documents répartis.
- diversité des matériels et des logiciels. Des normes telles que HTTP, HTML ou MIME définissent les formats des informations échangées et permettent au Web d'être indépendant des machines et des systèmes utilisés. Cependant, étant donné la diversité des types de documents servis par les serveurs Web, les navigateurs offrent des possibilités d'extension pour ajouter des programmes spécifiques de traitement de ces documents (appelés aussi "plug-in").

Le Web, qui à la base offre des fonctions comparables à celle d'un système de fichiers répartis, est construit de façon à être adapté aux caractéristiques de l'Internet, notamment sa taille et sa diversité.

De façon similaire, je me pose le problème du développement et de l'exécution d'applications réparties, en prenant en compte les caractéristiques de l'Internet. Je précise cette direction de recherche dans la section suivante.

5.3. Direction de recherche

L'objectif général de mes recherches (actuelles et dans un futur immédiat) est de fournir un environnement pour le développement et l'exécution d'applications réparties sur l'Internet. Les services systèmes fournis doivent répondre aux besoins de ces applications et s'intégrer dans les environnements matériels et logiciels utilisés. Je décris tout d'abord ce que devraient être ces environnements dans les années à venir, posant ainsi mes hypothèses de travail, puis je développe les axes de travail que je me propose d'explorer.

5.3.1. Les machines-réseaux

Je fais l'hypothèse que l'architecture de l'Internet de demain sera composée de serveurs et de terminaux appelés *machines-réseaux* [SUN97]. Les serveurs gèrent des données, des applications et des services systèmes disponibles pour un ensemble d'usagers, alors que les machines-réseaux sont des terminaux graphiques utilisés par les usagers pour se connecter à l'Internet et accéder à distance aux objets gérés sur les serveurs.

On peut noter que cette architecture est semblable à celle que l'on a vu apparaître dans les années 80 avec des serveurs et des Terminaux X. Les Terminaux X sont des terminaux graphiques qui incluent un microprocesseur, de la mémoire et une carte d'accès au réseau. Sur ce terminal s'exécute en général un serveur X-Windows qui gère le terminal graphique. Un Terminal X n'est pas fait pour accueillir les applications de l'usager (bien que des commandes existent pour y installer certaines applications comme un gestionnaire de fenêtres). Les applications et les données utilisées sont stockées dans des serveurs de fichiers et un utilisateur a généralement un serveur de connexion sur lequel il exécute ses programmes, l'affichage étant systématiquement redirigé vers le Terminal X. L'avantage est une banalisation des postes de travail, un usager pouvant se connecter depuis tout poste à toute machine du réseau. De plus, l'architecture à base de Terminaux X simplifie l'administration des postes des usagers.

L'organisation à base de Terminaux X se prête bien aux conditions des réseaux locaux d'aujourd'hui, où l'on peut faire la supposition d'un réseau parfait. En effet, il est difficile avec les stations de travail actuelles de saturer les capacités de ces réseaux locaux qui sont également devenus très fiables. Ceci est principalement dû au fait que, sur un réseau local, il est possible de contrôler le nombre de machines et d'usagers gérés et d'anticiper l'évolution des infrastructures.

L'organisation à base de machines-réseaux est une généralisation à l'Internet de celle à base de Terminaux X, mais dans un environnement où le réseau est loin d'être parfait. Les performances de l'Internet sont en effet très variables et les partitions du réseau sont même courantes. Il est donc difficile d'exécuter toutes les applications à distances et de reposer sur des envois de message pour réaliser une mise à jour de l'affichage sur le terminal graphique. C'est pourquoi la machine-réseau permet l'exécution locale d'applications téléchargées à partir de serveurs. Les serveurs stockent à la fois des données et les applications servant à les manipuler.

5.3.2. Un environnement universel

Pour adopter une architecture à base de machines-réseaux, l'objectif à atteindre est d'offrir un environnement universel sur lequel les applications peuvent être téléchargées depuis des serveurs à travers l'Internet. Le téléchargement des applications signifie le chargement à distance du code et du contexte d'exécution de ces applications. Ce contexte peut comprendre des informations de configuration de l'application (propre à un usager particulier), mais également des données persistantes sur lesquelles opère l'application pour le compte de l'usager. Un exemple est une application *éditeur* composée du code de l'éditeur, d'un fichier contenant les options utilisées par l'usager courant et du document édité par cet usager.

Il y a différentes façons d'implanter cet environnement universel. Il peut être conçu comme une application d'un système standard s'exécutant sur la machine-réseau, ou être directement implanté par le matériel (la machine-réseau elle même). Il est cependant nécessaire d'en avoir une version logicielle portée sur la plupart des systèmes si l'on veut être en mesure d'exécuter du code téléchargé sur ces machines. Afin d'être indépendant des machines sous-jacentes, le code téléchargé est généralement interprété par l'environnement universel (comme par exemple dans les environnements Java ou Tcl).

L'avantage du téléchargement des applications est le même que pour un Terminal X. Il est possible depuis toute machine-réseau donnant accès à l'Internet (et offrant l'environnement universel) d'accéder à son environnement de travail sans nécessiter une installation préalable des programmes ou un chargement explicite des données utilisées.

5.3.3. Partage de l'information

Dès l'instant où l'Internet est utilisé pour permettre la coopération entre un ensemble d'usagers, il devient nécessaire de fournir les mécanismes systèmes nécessaires à la mise en œuvre du partage de l'information.

Les mécanismes de mise en œuvre du partage sont bien connus : il s'agit essentiellement du partage par appel de procédure à distance et du partage par gestion de copies multiples. Etant donné que l'Internet se caractérise par des capacités variables de communication, il convient d'adopter une solution qui sache s'adapter aux conditions du réseau pour une gestion optimale du partage. Selon les cas (capacité du réseau, volume d'information utilisée), il peut être plus intéressant d'utiliser l'une ou l'autre de ces techniques.

Il peut même arriver que l'Internet fasse l'objet de partitions assez longues. Dans ce cas, il peut être souhaitable de permettre aux usagers de continuer à utiliser leurs applications en mode déconnecté. Ceci implique de mettre en œuvre une politique de partage par duplication de l'information, permettant aux copies de diverger sur les différentes machines. Un protocole de réconciliation permet ensuite de remettre les copies en cohérence lorsque la connexion entre les machines est rétablie.

Un exemple d'application est un éditeur coopératif permettant l'édition de documents partagés. Un document peut être partagé par gestion de copies multiples. Si certains usagers deviennent inaccessibles, les copies de ce document peuvent diverger et seront remises en cohérence à l'aide du protocole de réconciliation.

5.3.4. Migration d'application

L'objectif général de ces recherches est de permettre aux usagers de retrouver leur environnement de travail depuis toute machine-réseau utilisée pour se connecter à l'Internet.

Jusqu'ici, j'ai considéré qu'un accès à une application se traduisait par le téléchargement de l'application et d'un ensemble d'objets contextuels, puis du démarrage de l'application (démarrage à froid). Si on considère un usager mobile se connectant depuis plusieurs machines-réseaux (domicile, bureau), il est fastidieux de terminer sa session de travail en quittant un endroit pour reconstruire cette session de travail depuis un autre endroit. Il est plus confortable de permettre la migration des applications composant la session de travail, de

façon similaire au redémarrage à chaud de certains systèmes. De même, cette fonction permet de sauvegarder l'état de sa session de travail sur un serveur lorsque l'on décide d'éteindre sa machine-réseau, puis de la retrouver ultérieurement, soit sur la même machine, soit sur une autre, en la téléchargeant à partir de ce serveur de sauvegarde.

Une telle fonction peut toujours être réalisée explicitement par chaque application, mais je pense qu'il est préférable d'offrir un mécanisme générique au niveau système pour que toute application puisse en bénéficier implicitement.

5.3.5. Protection

Si l'on permet aux usagers d'accéder à leur environnement de travail depuis toute machine-réseau connectée à l'Internet, il est nécessaire de mettre en place des mécanismes visant à protéger les usagers contre d'éventuelles attaques. La protection recouvre deux aspects : le premier, appelé *sécurité*, vise à se protéger contre des attaques venues du réseau par émission ou interception de messages ; le second, appelé *contrôle d'accès*, vise à permettre aux usagers de mettre en œuvre une politique de contrôle d'accès aux objets qu'ils partagent avec les autres usagers.

Dans une organisation à base de machines-réseaux sur l'Internet, la protection est un problème majeur, puisque contrairement à un réseau local, on ne peut plus faire l'hypothèse que toutes les personnes ayant physiquement accès au réseau sont dignes de confiance (l'écoute et l'interception de message sont donc possibles). De même, on ne peut plus faire l'hypothèse que tous les systèmes du réseau sont dignes de confiance (ce qui rend plus délicat le contrôle d'accès). Il faut donc mettre en place une politique de protection identifiant clairement des niveaux de confiance envers les intervenants. Un usager du système n'aura pas la même confiance envers une autre machine-réseau qu'envers un serveur authentifié, ce qui ne veut pas dire qu'il aura une confiance absolue envers ce serveur authentifié.

5.3.6. Exemple d'application

Un premier exemple d'application est une application de lecture et d'édition de courrier électronique. Si l'on veut permettre aux usagers d'utiliser cette application depuis n'importe quelle machine-réseau, différentes solutions existent déjà à l'heure actuelle :

- se servir de sa machine locale comme d'un Terminal X. Cette solution est utilisable avec toutes les applications de courrier électronique existantes, mais comme cela a été mentionné auparavant, elle engendre de nombreuse communications, chaque mise à jour de l'affichage se traduisant par un envoi de message au serveur X local. Cette solution n'est donc pas réaliste sur l'Internet.
- utiliser une application locale qui coopère avec un serveur qui gère le courrier de façon persistante. L'application locale contient toute la partie graphique de l'application et les messages et les dossiers de messages sont gérés sur le serveur. Les solutions à base de serveurs POP sont proches de cette solution, mais les serveurs POP ne permettent pas de gérer des dossiers de messages à distance ; l'archivage des messages dans des dossiers est réalisé sur le poste client, ce qui rend difficile la gestion des messages depuis plusieurs machines. De plus, cette solution nécessite une installation préalable

de l'application locale sur toutes les machines potentiellement utilisables pour se connecter, ce qui n'est pas notre hypothèse de travail. Enfin, l'application locale, pour fonctionner efficacement, devra certainement gérer un cache des objets les plus utilisés et gérer les mises à jour entre ce cache et le serveur. Je pense qu'il s'agit là d'un service de partage d'objets répartis qui doit être fourni par le système et utilisable par toutes les applications ayant ce même problème.

Si cette application est réalisée sur l'environnement universel proposé, elle sera téléchargeable depuis tout poste de travail ayant une version de cet environnement. Le service de partage d'objets permettra d'accéder aux objets de l'application gérés sur un serveur, soit par appel à distance, soit par copie de l'objet localement. En cas de déconnexion, la copie locale pourra être exploitée jusqu'à ce que la reconnexion soit possible, un algorithme de réconciliation permettant d'assurer la cohérence des données partagées. Le service de migration permettra de laisser une session d'édition de courrier en cours à son bureau, puis de la reprendre à son domicile, ceci sans nécessiter que cette option soit explicitement implantée dans l'application. Enfin, le service de protection assurera que seul un usager est autorisé à consulter/composer des messages appartenant/provenant de cet usager.

Un autre exemple très proche de ce dernier est un éditeur coopératif de documents partagés. Il ajoute à l'exemple précédent la possibilité de partager la même application (ainsi que les objets qu'elle gèrent) entre un ensemble d'usagers. La gestion des droits d'accès peut également être raffinée.

5.3.7. Résumé

En résumé, ces recherches visent à fournir un environnement universel sur lequel il est possible de télécharger des applications et leur contexte, à partir de serveurs et à travers l'Internet. Ce mode de fonctionnement permet de n'administrer les applications que sur les serveurs, indépendamment des machines réseaux utilisées pour se connecter à l'Internet.

Cet environnement universel doit permettre le partage d'information avec les serveurs qu'il utilise ainsi qu'avec les autres environnements avec lesquels il interagit. Le partage d'information doit s'adapter aux conditions du réseau et prendre en compte d'éventuelles partitions du réseau, fréquentes sur l'Internet.

L'environnement universel doit prendre en compte la mobilité des usagers en leurs permettant d'accéder à une même session de travail depuis plusieurs machines-réseaux. Ceci nécessite d'être en mesure de déplacer les applications en cours d'exécution entre des machines différentes.

Enfin, l'environnement universel doit assurer la sécurité des usagers l'utilisant. L'ouverture sur l'Internet ne doit pas mettre en péril les applications que l'on rend accessibles par l'Internet. L'environnement universel doit fournir un moyen de définir des règles de contrôle d'accès et assurer le respect de ces règles.

5.4. Programme de recherche

Des expérimentations concernant cette direction de recherche ont déjà commencé. Je présente dans cette section le cadre de travail que je me suis fixé ainsi que les deux expérimentations qui sont en cours dans ce cadre.

5.4.1. Java

Une évolution marquante de ces dernières années est l'arrivée de l'environnement Java [GOS95]. Java s'est imposé comme langage et environnement d'exécution d'applications réparties sur l'Internet, principalement grâce aux propriétés suivantes :

- traitement de l'hétérogénéité. Java est un langage interprété. Le code généré par le compilateur Java est indépendant de la machine sur laquelle il est exécuté ; il peut donc être téléchargé sur des machines de types différents.
- sécurité du langage. Le langage Java est fortement typé et proscrit l'utilisation d'adresses virtuelles, ce qui garantit l'isolation entre les objets Java. L'environnement Java peut également être configuré pour contrôler l'accès aux ressources de la machine où l'application est téléchargée.
- très large diffusion. Java, dès son apparition, a été intégré dans le Web. Il a été intégré dans plusieurs navigateurs (initialement HotJava, puis Netscape et InternetExplorer), et de nombreuses applications, développées sous forme d'*Applets* téléchargeables avec ces navigateurs, ont été rendues accessibles sur le Web. Java est donc aujourd'hui installé sur la plupart des machines dans le monde.

Ces caractéristiques de Java en font une base très intéressante pour réaliser l'environnement universel décrit ci-dessus.

5.4.2. Problèmes et solutions explorés

Je présente maintenant deux expérimentations qui ont déjà commencé sur des sujets évoqués précédemment. Il s'agit de la gestion d'objets partagés répartis et de la migration d'applications. Ces deux expérimentations sont effectuées en utilisant la machine virtuelle Java.

Gestion d'objets partagés répartis

La première de ces expérimentations concerne la gestion d'objets partagés entre des machines virtuelles Java différentes [HAG98].

Dans ce cadre, Java propose déjà un mécanisme réalisant cette fonction qui est Java-RMI (Java Remote Method Invocation) [WOL96]. Java-RMI est un outil permettant de réaliser des appels d'objets à distance entre des machines virtuelles Java différentes. En utilisant Java-RMI, des références à des objets Java peuvent être exportées vers d'autres machines virtuelles Java à travers un serveur de nom. Ce serveur de nom permet de récupérer cette référence ainsi que les classes (des talons comme dans un RPC conventionnel) permettant de mettre en œuvre un appel de méthode à distance. Le client récupérant cette référence peut ensuite appeler des méthodes sur l'objet dont la référence a été exportée. Cet appel se traduit

par des envois de message à l'aller et au retour, avec emballage et déballage des paramètres. Des références d'objets peuvent être également échangées en paramètres de ces appels de méthodes. Les talons nécessaires aux appels de méthodes sur ces références sont en général téléchargés depuis le site stockant l'objet.

Si RMI est un outil très pratique (similaire à ce qu'offre CORBA), il reste fondé sur un schéma client-serveur et il ne permet pas de gérer des copies d'objets sur un site client (dans un cache) afin de réaliser des accès locaux une fois qu'une copie a été rapatriée. Un mécanisme appelé sérialisation permet de copier des objets d'une machine virtuelle Java à une autre, mais la gestion de la cohérence entre ces copies est laissée à la charge des applications utilisant la sérialisation.

La gestion de copies multiples d'objets afin de permettre des accès locaux (et de profiter de la localité des accès) est indispensable, en particulier dans le cadre d'applications réparties sur l'Internet où les performances du réseau sont très variables. Afin de fournir un tel service, nous avons réalisé un service système qui fournit l'abstraction d'un espace d'objets Java réparti. Les objets Java sont copiés à la demande sur les machines clientes et conservés jusqu'à ce qu'ils soient invalidés par le protocole de gestion de la cohérence.

Les caractéristiques essentielles de ce services sont les suivantes :

- Les applications sont développées comme des applications centralisées. La répartition est gérée implicitement par le service.
- Une application est installée sur un seul site et téléchargée sur les sites clients l'utilisant.
- Les objets sont téléchargés à la demande sur les sites qui y ont accès. Ils peuvent ensuite être appelés localement jusqu'à ce qu'ils soient invalidés par le protocole de gestion de la cohérence.
- Pour limiter le coût des mécanismes mettant en œuvre le partage, les objets sont regroupés en *clusters* (grappes d'objets) qui constituent l'unité de désignation, de partage et de cohérence.

Un prototype de ce service a été réalisé sur la machine virtuelle Java et des applications sont en cours de développement afin d'évaluer ce service.

Migration d'applications

La seconde de ces expérimentations concerne la migration d'applications. L'objectif est d'être en mesure de déplacer dynamiquement des applications entre des machines virtuelles Java.

Pour réaliser un tel service, deux problèmes doivent être résolus :

- déplacer le contexte d'un processus élémentaire (ou *thread*). Il faut être capable de récupérer l'état de ce processus, composé d'un ensemble de registres et de la pile du processus. Cet état, s'il ne contient que des valeurs, ne pose pas de problème particulier. Toutefois, Java de fournit pas à l'heure actuelle de fonction permettant de capturer l'état d'un processus.

- déplacer le contexte de l'application. En général, le contexte d'une application n'est pas seulement composé de l'état d'un processus, mais aussi d'un ensemble d'objets Java gérés par l'application. De plus, l'état du processus peut inclure (sur la pile par exemple) des références à des objets Java faisant partie du contexte de l'application. Globalement, le problème est d'identifier l'ensemble des objets faisant partie du contexte de l'application et de déplacer ce contexte vers une autre machine virtuelle Java (tout en conservant sa structure).

Un des problèmes difficiles est alors de déplacer des objets gérés par l'application, pouvant être liés au site courant où l'application s'exécute. Par exemple, un objet Java qui implante un *socket* peut difficilement être déplacé.

Ce travail est moins avancé que le précédent, mais nous étudions déjà quelques pistes intéressantes. Nous pensons que la migration d'application nécessite la coopération entre l'application et le service système que nous réalisons afin de déterminer les objets qui doivent être déplacés et, le cas échéant, de réaliser un traitement spécifique sur ces objets lorsqu'il sont déplacés. Sur l'exemple du canal de communication, l'application peut décider de fermer le canal de communication lorsque l'objet est déplacé et de le rouvrir sur le site de destination.

Sur ce sujet, nos travaux actuels visent à étendre la machine virtuelle Java afin de permettre la migration de processus, en offrant aux applications une interface permettant de spécifier les objets devant être déplacés ainsi qu'un traitement spécifique à appliquer lorsque ces objets sont déplacés.

5.5. Conclusion

Une des évolutions marquantes du domaine est le développement de l'Internet. Le réseau Internet est aujourd'hui accessible à tous et il va se généraliser comme support de communication et de coopération, pour les professionnels comme pour les particuliers.

Dans ce contexte, le problème que j'étudie est de permettre aux usagers d'accéder au même environnement depuis leurs différents points d'accès à l'Internet, les usagers se connectant à partir de terminaux appelés *machines-réseaux*. Cet objectif nécessite de disposer d'un environnement universel sur les différentes machines-réseaux utilisées pour se connecter. Les applications utilisées ainsi que les données qu'elles manipulent peuvent être téléchargées sur l'environnement universel et utilisées localement sur ces machines.

A partir de ce premier objectif, d'autres deviennent immédiatement souhaitables. Il faut permettre aux usagers de coopérer en partageant une partie de leur environnement. Un usager doit pouvoir rendre accessible une application à un autre usager, et même partager une application coopérative entre un groupe d'usagers travaillant à une tâche commune. Le partage nécessite alors la mise en place de mécanismes permettant de contrôler l'accès aux environnements des usagers. Enfin, un usager pouvant se connecter depuis plusieurs points d'accès à l'Internet, il doit pouvoir retrouver sa session de travail au stade où il l'avait laissée lors de sa dernière connexion, ce qui nécessite un mécanisme de migration d'applications.

Mon programme de recherche est de réaliser un environnement de ce type sur la machine virtuelle Java. Deux expérimentations ont déjà commencé.

La première consiste à réaliser une couche logicielle sur Java réalisant un espace d'objets Java partagés répartis. Les objets sont partagés par copies multiples. Cette couche logicielle permet en premier lieu de partager les objets composant l'environnement d'un usager entre la machine serveur à laquelle l'usager se connecte et la machine-réseau utilisée pour se connecter ; les objets sont ainsi copiés à la demande sur la machine-réseau. Cette couche logicielle permet ensuite de partager un ensemble d'objets entre différentes machines-réseaux dans le cadre d'une application coopérative.

La seconde expérimentation consiste à réaliser un mécanisme de migration d'application entre des machines virtuelles Java différentes. La migration consiste à déplacer une application ainsi que son contexte (un ensemble d'objets Java utilisés par cette application) entre deux machines. Cette fonction peut être utilisée pour sauvegarder une session de travail et la reprendre sur une autre machine-réseau dans l'état où elle avait été arrêtée.

Ces expérimentations sont en cours, des résultats ont d'ores et déjà été obtenus pour la première [HAG98] et le travail continue.

[ALM85] G. T. Almes, A. P. Black, E. D. Lazowska, and J. D. Noe, "The Eden System: A Technical Review", *IEEE Transactions on Software Engineering*, 11 (1), pp. 43-59, Janvier 1985.

[ANA92] R. Ananthanarayanan, M. Ahamad, R.J. Leblanc, "Application Specific Coherence Control for High Perfomance Distributed Shared Memory", *3rd Symposium on Experiences with Distributed and Multiprocessor Systems*, Mars 1992.

[AND95] T.E. Anderson, M.D. Dahlin, J.M. Neefe, D.A. Patterson, D.S. Roselli, R.Y. Wang, "Serverless Network File System", *15th Symposium on Operating Systems Principles (SOSP)*, Copper Mountain Resort, Colorado, Décembre 1995.

[ACE86] M. J. Acetta, R. Baron, W. Bolowsky, D. Golub, R. Rashid, A. Tevanian, M. Young, "Mach: A New Kernel Foundation for Unix Development", *Proc. of the USENIX 1986 Summer Conference*, Juillet 1986, pp. 93-112.

[BAL91a] 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, G. Vandôme, "Architecture and Implementation of Guide, an Object-Oriented Distributed System", *Computing Systems*, vol. 4, 1, pp. 31-67, 1991.

[BAL91b] R. Balter, J.-P. Banâtre, S. Krakowiak Editeurs, *Construction des systèmes d'exploitation répartis*, Collection didactique de l'INRIA, 1991.

[BAL92] H.E. Bal, M.F. Kaashoek, A.S. Tanenbaum, "Orca: A Language for Parallel Programming of Distributed Systems", *IEEE Transactions on Software Engineering*, 18(3), pp. 190-205, Mars 1992.

[BAL93] R. Balter, P.Y. Chevalier, A. Freyssinet, D. Hagimont, S. Lacourte, X. Rousset de Pina, "Is the Micro-Kernel Technology well suited for the support of Object-Oriented Operating Systems: the Guide Experience", *2nd Symposium on Microkernels and Other Kernel Architectures (MOKA), San Diego*, pp. 1-11, Septembre 1993.

[BAL97] R. Balter, S. Krakowiak, "Rétrospective sur le projet Guide : un environnment à base d'objets pour applications réparties", *L'Objet*, Juin 1997.

[BEL96] L. Bellissard, S. Ben Atallah, F. Boyer, M. Riveill, "Distributed Application Configuration", *Proc. 16th International Conference on Distributed Computing Systems (ICDCS)*, Hong-Kong, Mai 1996.

[BEN92] V. Benzaken, C. Delobel, G. Harrus, "Clustering Strategies in O2: an Overview", *Building an Object-Oriented Database System: the Story of O2*, Morgan Kaufman, 1992.

[BER93] B. N. Bershad, M. J. Zekauskas, W. A. Sawdon, "The Midway Distributed Shared Memory System, *Proc. 38th IEEE Computer Society International Conference (COMPCON'93)*, Février 1993, pp. 528-537.

[BER95] B. N. Bershad, S. Savage, P. Pardyak, E.G. Sirer, M.E. Fiuczynski, D. Becker, C. Chambers, S. Eggers, "Extensibility, Safety and Performance in the SPIN Operating System",*15th Symposium on Operating Systems Principles (SOSP)*, Copper Mountain Resort, Colorado, Décembre 1995.

[BIR82] 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.

[BIR84] A. D. Birrell, B. J. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Programming Languages and Systems*, 2(1), Février 1984.

[BLA87] A. Black, N. Hutchinson, E. Jul, H. M. Levy, and L. Carter, "Distribution and Abstract Types in Emerald", *IEEE Transactions on Software Engineering*, vol. 13 (1), pp. 65-76, Janvier 1987.

[CAH93a] V. Cahill, R. Balter, X. Rousset de Pina, N, Harris, *The Comandos Distributed Application Platform,* Chapitre 8, Springer-Verlag, 1993.

[CAH93b] V. Cahill, S. Baker, C. Horn, G. Starovic, "The Amadeus GRT - Generic Runtime Support for Distributed Persistent Programming", *8st ACM Conference on Object-Oriented Systems, Languages and Applications (OOPSLA),* pp. 144-161, Octobre 1993.

[CAR91] J. Carter, J. Bennett, W. Zwaenepoel, "Implementation and performance of Munin", *13th ACM Symposium on Operating Systems Principles (SOSP'91)* , Octobre 1991.

[CAR93] J.B. Carter, *Efficient distributed shared memory based on multi-protocols release consistency,* Phd Thesis, Rice University, Septembre 1993.

[CHA94] J. S. Chase, H. M. Levy, E. J. Feeley, E. D. Lazowska, "Sharing and Protection in a Single-Address-Space Operating System", *ACM Transactions on Computer Systems*, 12 (4), pp. 271-307, Novembre 1994.

[CHE93] P.Y. Chevalier, A. Freyssinet, D. Hagimont, S. Krakowiak, S. Lacourte, 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.

[CHE94] P-.Y. Chevalier, *Persistance et Disponibilité dans les Systèmes Répartis,* Thèse de Doctorat en Informatique, Université Joseph Fourier (Grenoble I), Octobre 1994.

[CHE96] P. Y. Chevalier, D. Hagimont, J. Mossière, X. Rousset de Pina, "Le système réparti à objets Guide", *Technique et Science Informatiques (TSI),* 15(6), 1996.

[DAS90] 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, C.J. Wilkenloh, "The Design and Implementation of the Clouds Distributed Operating System", *Computing Systems,* 3(1), pp. 11-45, Hiver 1990.

[DAY92] M. Day, B. Liskov, U. Maheshwari, A.C. Myers, *Naming and Locating Objects in Thor*, Laboratory of Computer Science, MIT, 1992.

[DEC91] D. Decouchant, P. Le Dot, M. Riveill, C. Roisin, X. Rousset de Pina, "A Synchronization Mechanism for Typed Objects in a Distributed System", *11th International Conference on Distributed Systems (ICDCS),* pp. 152-161, Septembre 1991.

[DEC93] D. Decouchant, V. Quint, M. Riveill, I. Vatton, *Griffon: A Cooperative, Structured, Distributed Document Editor,* (93-01), Bull-IMAG, Mai 1993.

[DEC96a] P. Dechamboux, D. Hagimont, J. Mossière, X. Rousset de Pina, "Un service de gestion de données persistantes partagées", Rencontres Francophones du Parallèlisme(*RenPar'8*), Bordeaux, Mai 1996.

[DEC96b] P. Dechamboux, D. Hagimont, J. Mossière, X. Rousset de Pina, "The Arias Distributed Shared Memory: an Overview", Twenty third International Winter School on Current Trends in Theory and Practice of Informatics (invited talk), Milovy, République Tchèque, Novembre 1996.

[DEU91] O. Deux et al. The O2 System, *Communications of the ACM*, 34 (10), , pp. 34-38, Octobre 1991.

[ENG95] D.R. Engler, M.F. Kaashoek, J. O'Toole, "Exokernel: An Operating System Architecture for Application-Level Resource Management", *15th Symposium on Operating Systems Principles (SOSP),* Copper Mountain Resort, Colorado, Décembre 1995.

[FAS96] J. Fassino, *Utilisation d'une Mémoire Virtuelle Répartie pour le support d'un Système de Gestion de Fichiers Réparti,* Rapport de DEA Informatique, ENSIMAG, Juin 1996.

[FRE91] A. Freyssinet, *Architecture et Réalisation d'un Système Réparti à Objets*, Thèse de doctorat, Université Joseph Fourier, Juillet 1991.

[GOS95] J. Gosling, H. McGilton, "The Java Language Environment", White Paper, Sun Microsystems Inc., 1995.

http://www.javasoft.com/docs/white/index.html

[HAG93] D. Hagimont, *Adressage et protection dans un système réparti*, Thèse de Doctorat, Institut National Polytechnique de Grenoble, Octobre 1993.

[HAG94a] D. Hagimont, "Protection in the Guide Object-Oriented Distributed System", *8th European Conference on Object-Oriented Programming (ECOOP'94)*, Bologna, Juillet 1994.

[HAG94b] D. Hagimont, P. Y. Chevalier, A. Freyssinet, S. Krakowiak, S. Lacourte, J. Mossière, X. Rousset de Pina, "Persistent Shared Object Support in the Guide System: Evaluation and Related Work", *9th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Portland, Octobre 1994.

[HAG96a] D. Hagimont, J. Mossière, "Problèmes de désignation, de localisation et d'accès dans les systèmes répartis à objets", *Technique et Science Informatiques (TSI)*, 15(1), 1996.

[HAG96b] D. Hagimont, J. Mossière, X. Rousset de Pina, F. Saunier, "Hidden Software Capabilities", *16th International Conference on Distributed Computing Systems (ICDCS)*, Hong-Kong, Mai 1996.

[HAG97a] D. Hagimont, L. Ismail, "A Protection Scheme for Mobile Agents on Java", *3rd ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Budapest, Septembre 1997.

[HAG97b] D. Hagimont, O. Huet, J. Mossière, "A Protection Scheme for a CORBA Environment", *ECOOP Workshop on CORBA*, Juin 1997.

[HAG98] D. Hagimont, D. Louvegnies, "Javanaise: distributed shared objects for Internet cooperative applications", *Rapport Technique*, Janvier 1998.

[HAN96] J. Han, *Motivation, conception et réalisation d'une mémoire partagée répartie*, Thèse de Doctorat, Institut National Polytechnique de Grenoble, Novembre 1996.

[HEI93] G. Heiser, K. Elphinstone, S. Russell, J. Vochteloo, *Mungi: a distributed single address-space operating system*, (TR 9314), School of Computer Science and Engineering, University of New South Wales, Novembre 1993.

[HUT87] N.C. Hutchinson, *Emerald: An Object-Based Language for Distributed Programming*, PhD thesis, University of Washington, Janvier 1987.

[ION94] Iona Technologies, *Orbix distributed object technology - programmer's guide*, Version 1.2, Février 1994.

[KHA93] Y.A. Khalidi, M.N. Nelson, "Extensible File Systems in Spring", *14th Symposium on Operating Systems Principles (SOSP)*, Asheville, Décembre 1993.

[KNA96] A. Knaff, *Conception et réalisation d'un service de stockage fiable et extensible pour un système réparti à objets persistants*, Thèse de Doctorat, Université Joseph Fourier, Octobre 1996.

[KNA97] A. Knaff, P. Dechamboux , "Reliable Support for a Persistent Distributed Shared Memory", *17th International Conference on Distributed Computing Systems (ICDCS)*, Baltimore, 1997.

[KOW90] Oliver C. Kowalski, Hermann Härtig, "Protection in the BirliX Operating System", *10th International Conference on Distributed Computing Systems (ICDCS)*, pp. 160-166, Mai 1990.

[KRA90] S. Krakowiak, M. Meysembourg, H. Nguyen Van, M. Riveill, C. Roisin, 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.

[LAM71] B.W. Lampson, “Protection ”, *Proceedings of the fifth Annual Princeton Conference on Information Sciences and Systems*, pp. 437-443, Mars 1971.

[LEA93] R. Lea, C. Jacquemot, 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, Septembre 1993.

[LEG88] J. Legatheaux Martins et Y. Berbers, “La désignation dans les systèmes d’exploitation répartis”, *Technique et Science Informatique,* 7(4), pp. 359-372, 1988.

[LEV84] H.M. Levy, *Capability-Based Computer Systems*, Digital Press, Bedford, 1984.

[LI89] K. Li et P. Hudak, “Memory Coherence in Shared Virtual Memory Systems”, *ACM Transactions on Computer Systems,* 7(4), pp. 321-359, Novembre 1989.

[LIS85] B.H. Liskov. The Argus Language and System, in *Distributed Systems: Methods and Tools for Specification*, M. Paul and H. J. Siegert, editors, Lecture Notes in Computer Science, n° 190, pp. 343-430.

[LIS87] B.H. Liskov, D. Curtis, P. Johnson et R. Scheifler, “Implementation of Argus”, *11th ACM Symposium on Operating System Principles, SIGOPS Operating System Review,* 21(5), pp. 111-122, Novembre 1987.

[LIS92] B. Liskov, *Preliminary design of the Thor object-oriented database system,* (Programming Methodology Group Memo 74), Laboratory of Computer Science, MIT, 1992.

[MAI92] J. Maisonneuve, M. Shapiro, P. Collet, ”Implementing References as Chains of Links”, 3rd International Workshop on Object Orientation in Operating Systems (IWOOOS), Dourdan, Octobre 1992.

[MIT94] J. G. Mitchell, J. J. Gibbons, G. Hamilton, P. B. Kessler, Y. A. Khalidi, P. Kougiouris, P. W. Madany, M. N. Nelson, M. L. Powell, and S.R. Radia, "An Overview of the Spring System", *Proceedings of CompCon Spring 1994*, Février 1994.

[MOS89] J.E.B. Moss, “Addressing Large Distributed Collections of Persistent Objects: The Mneme Project’s Approach”, *2nd International Workshop on Database Programming Languages*, Gleneden Beach, Oregon, Juin 1989.

[MOS90] J.E.B. Moss, “Design of the Mneme Persistent Object Store”, *ACM Transactions on Information Systems*, 8(2), pp. 103-139, Avril 1990.

[MOS92] 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.

[MOS93] D. Mosberger, “Memory Consistency Models”, *Operating Systems Review,* 27(1), pp. 18-26, Janvier 1993.

[MUL86] S.J. Mullender, A.S. Tanenbaum, ”The design of a capability-based distributed operating system”, *Computer Journal,* 29(8), pp. 289-299, Août 1986.

[MUR93] K. Murray, T. Stiemerling, T. Wilkinson, P. Kelly, "Angel: Resource Unification in a 64-bit Micro-Kernel", *Proceedings of the 27th Hawaii International Conference on Systems Science*, Juin 1993.

[NEE78] R.M. Needham, M.D. Schroeder, “Using encryption for authentication in large network of computers”, *Communications of the ACM,* 21(12), pp. 993-999, Décembre 1978.

[ORG72] E.I. Organick, *The Multics system: an examination of its structure*, MIT Press, 1972.

[PER95] E. Pérez Cortés, P. Dechamboux, J. Han, "Generic Support for Synchronisation and Consistency in Arias", *5th Workshop on Hot Topics in Operating Systems (HotOS V)*, Mai 1995.

[PER96] E. Pérez Cortés, *La cohérence sur mesure dans une mémoire virtuelle répartie partagée*, Thèse de Doctorat, Institut National Polytechnique de Grenoble, Novembre 1996.

[POW83] M. Powell, 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.

[PUA93] I. Puaut, *Gestion d'objets actifs dans les systèmes distribués : problématique et mise en œuvre*, Thèse de doctorat, Université de Rennes I, Janvier 1993.

[RIT93] S. Ritchie, *The Raven Kernel: a Microkernel for Shared Memory Multiprocessors*, (TR 93-36), Department of Computer Science, University of British Columbia, Avril 1993.

[SAL78] J.H. Saltzer, *Naming and Binding of objects, in Operating Systems - an advanced course*, Lecture Notes in Computer Science, Springer-Verlag, Vol. 60, pp. 99-208, 1978.

[SAN93] M. Santana, *Manuel de Référence du Langage OC++*, Centre de Recherche Bull 2 rue de vignate Gières 38, Décembre 1993.

[SAU96] F. Saunier, *Protection d'une mémoire virtuelle répartie par capacités implicites*, Thèse de Doctorat, Institut National Polytechnique de Grenoble, Octobre 1996.

[SEL96] M.I. Seltzer, Y. Endo, C. Small, K.A. Smith, "Dealing With Disaster: Surviving Misbehaved Kernel Extensions", *2nd Symposium on Operating Systems Design and Implementation (OSDI)*, Octobre 1996.

[SHA86] M. Shapiro, "Structure and Encapsulation in Distributed Systems: The Proxy Principle", *6th International Conference on Distributed Computing Systems*, pp. 198-204, 1986.

[SHA89] M. Shapiro, Y. Gourhant, S. Habert, L. Mosseri, M. Ruffin, C. Valot, "SOS: An Object-Oriented Operating System - Assessment and Perspectives", *Computing Systems*, 2(4), pp. 287-337, 1989.

[SOL93] R.M. Soley, *Object Management Architecture Guide*, Rev. 2.0, 2nd edition, Whiley-QED, 1993.

[STE88] JG. Steiner, C. Neumann, J.I. Schiller, "Kerberos: an authentication service for open network systems", *Proceedings of the USENIX Winter Conference*, Dallas, Février 1988.

[SUN97] Sun, "Remote Windowing on a JavaStation Network Computer", White Paper, Sun Microsystems Inc., 1997.

http://www.sun.com/javastation/whitepapers/remote.html

[TAN90] A.S. Tanenbaum, R. Van Renesse, H. Van Staveren, G.J. Sharp, S.J. Mullender, J. Jansen, G. Van Rossum, "Experiences with the Amoeba distributed operating system", *Communications of the ACM*, 33(12), pp. 46-64, Décembre 1990.

[VOC93] J. Vochteloo, S. Russel, and G. Heiser, *Capability-Based Protection in a Persistent Global Virtual Memory System*, SCSE Report 9303, University of New South Wales, Mars 1993.

[WOL96] A. Wollrath, R. Riggs, J. Waldo, "A Distributed Object Model for the Java System", Computing Systems, 9(4), pp. 291-312, Fin 1996.

[WUL74] W.A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, F. Pollack, "Hydra: The Kernel of a Multiprocessor Operating System", *Communications of the ACM*, 17(6), Juin 1974.

[ZIM95] P. Zimmermann, *The Official PGP User's Guide*, MIT Press, 1995.

Liste des articles fournis en annexe du rapport d'habilitation

[CHE96] P. Y. Chevalier, D. Hagimont, J. Mossière, X. Rousset de Pina, ”Le système réparti à objets Guide”, *Technique et Science Informatiques (TSI)*, 15(6), 1994.

[DEC96b] P. Dechamboux, D. Hagimont, J. Mossière, X. Rousset de Pina, "The Arias Distributed Shared Memory: an Overview", Twenty third International Winter School on Current Trends in Theory and Practice of Informatics (invited talk), Milovy, République Tchèque, Novembre 1996.

[HAG96b] D. Hagimont, J. Mossière, X. Rousset de Pina, F. Saunier, ”Hidden Software Capabilities”, *16th International Conference on Distributed Computing Systems (ICDCS)*, Hong-Kong, Mai 1996.

[HAG98] D. Hagimont, D. Louvegnies, ”A Java-based system support for distributed applications on the Internet”, *Rapport Technique*, Avril 1998.