Conception et construction de noyaux de systèmes opératoires

Xavier Rousset de Pina

▶ **To cite this version:**

Xavier Rousset de Pina. Conception et construction de noyaux de systèmes opératoires. Réseaux et télécommunications [cs.NI]. Université Joseph-Fourier - Grenoble I, 1993. tel-00068012

HAL Id: tel-00068012

https://theses.hal.science/tel-00068012

Submitted on 10 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DOCUMENT DE PRÉSENTATION de travaux

Pour obtenir

l'Habilitation à diriger des recherches en

en INFORMATIQUE

Par

Xavier Rousset de Pina

Université Joseph Fourier (Grenoble - 1)

Examinateurs :

MM. : R. Balter, Directeur de l'UMR 122 - Institut IMAG, Grenoble

J.-P. Banâtre, Professeur à l'Université Rennes-1

C. Girault, Professeur à l'Université P. & M. Curie (Paris 6)

D. Herman, Professeur à l'Université Rennes-1

S. Krakowiak, Professeur à l'Université J. Fourier (Grenoble-1)

J. Mossière, Professeur à l'I. N. P., Grenoble

J.-P. Verjus, Professeur à l'I. N. P., Grenoble (Président)

Je remercie ici tous ceux qui, par leurs conseils, leurs remarques et leur encouragements, ont contribué à l'aboutissement de ce travail :

Monsieur Jean Pierre Verjus, Professeur à l'INPG et Directeur de l'Institut IMAG, président du jury.

Monsieur Sacha Krakowiak, Professeur à l'Université Joseph Fourier, mon Directeur de recherche.

Monsieur Jean Pierre Banâtre, Professeur à l'Université Rennes 1, Monsieur Claude Girault, Professeur à l'Université P. et M. Curie (Paris 6), et Monsieur Jacques Mossière, Professeur à l'INPG, qui ont accepté de juger ce travail.

Monsieur Roland Balter, Directeur de l'Unité Mixte de recherche Bull-IMAG au sein de laquelle se sont effectués les derniers travaux décrits dans ce mémoire.

Monsieur Daniel Herman, Professeur à l'Université Rennes 1, qui a bien voulu participer au jury.

Monsieur le Professeur Noël Gastinel et Monsieur le Professeur Louis Bolliet qui ont rendu possible et encouragé ma reconversion des mathématiques à l'informatique.

L'ensemble des participants aux projets de recherches et de développements auxquels j'ai participé pendant ma carrière et plus particulièrement : Claude Otrage qui a guidé mes premiers pas dans Gemau, puis m'a accueilli dans son entreprise (APSIS) pour travailler sur le réseau Factor ; Jacques Briat responsable scientifique du projet Ours et grand pourvoyeur d'idées ; Vincent Quint qui m'a accueilli dans le projet Danube puis initié à Edimath ; Irène Vatton avec qui j'ai travaillé dans Ours et dans Danube ; tous ceux qui travaillent ou ont travaillé au projet Guide avec bien sûr une mention spéciale à Dominique Decouchant et Michel Riveill qui ont constitué la cheville ouvrière du premier prototype, à Jacques Cayuela, Pierre-Yves Chevalier, André Freyssinet, Daniel Hagimont et Serge Lacourte qui constituent le noyau dur d'Eliott, à Miguel Santana pour les nombreux conseils et avis qu'il m'a donnés ; enfin le Professeur Santosh Shrivastava qui m'a accueilli à Newcastle upon Tyne dans l'équipe Arjuna ainsi que mes *team mates* Mark Little, Stuart Wheater et Dan MacCue.

J'exprime également ma gratitude à Claude Kaiser pour ses commentaires sur le manuscrit, ainsi qu'à tous les membres de l'Unité Mixte Bull-IMAG, et plus spécialement à Joëlle Macé et Béatrice Claudio qui savent si gentiment et efficacement résoudre nos problèmes administratifs et organiser nos déplacements.

Sommaire du mémoire

1.Introduction 1
2.Évolution des systèmes opératoires depuis 1972 3
2.1.Évolution des outils 3
2.1.1.État des lieux en 70 3
2.1.2.Les facteurs d'évolution 6
2.2.Évolution de la gestion de l'information 9
2.2.1.Désignation et partage 10
2.2.2.Protection et sécurité 12
2.3. Bilan 14
3.Présentation de mes travaux 15
3.1.Outils pour la recherche en systèmes 15
3.1.1.Le projet GEMAU 15
3.1.2.Le projet Ours 18
3.2.Architecture et mise en œuvre de protocoles 22
3.2.1.Travaux autour du réseau local Danube 23
3.2.2.Le réseau local Factor 23
3.3.Étude des systèmes répartis : Le projet Guide 27
3.3.1.Présentation du projet 27
3.3.2.Évaluation critique de la première phase 29
3.3.3.Choix de conception de la seconde maquette 32
3.4.Un travail sur la duplication d'objets dans le cadre d'Arjuna 41
3.4.1.Le projet Arjuna 41
3.4.2.Duplication active d'objets accédés par des transactions 42
3.4.3.Bilan 43
4.Conclusion 45
4.1.Retour sur l'évolution 45
4.2.Situation actuelle 46
4.2.1.Micro-noyaux 46
4.2.2.Langages et systèmes à objets 47
4.2.3.Situation du marché 48
4.3.Perspectives 49
5.Références 51

1. Introduction

Ce mémoire présente une synthèse des travaux de recherche sur les systèmes opératoires auxquels j'ai participé depuis octobre 1970. Le besoin d'acquérir et de consolider des connaissances (connaissances que je dois par ailleurs enseigner) par la participation, même modeste, à la construction de prototypes a été, au cours de ces vingt-deux dernières années, le fil directeur de mes activités de recherche.

Un système opératoire a deux fonctions : d'une part assurer la meilleure répartition possible entre ses utilisateurs des ressources physiques ou logiques de la machine qu'il gère, et d'autre part fournir une interface d'utilisation et des services de plus haut niveau d'abstraction que ceux fournis par la machine hôte. Un système se présente donc pour ses utilisateurs comme une machine abstraite. Tout travail sur les systèmes opératoires doit prendre en compte ces deux aspects même s'il peut choisir d'en privilégier un en fonction de la technologie disponible à une époque déterminée.

Au seuil des années 70, la plus grande partie des concepts qui sont à la base des systèmes opératoires actuels ont déjà vu le jour : processus et synchronisation [Dijkstra 65] [Dijkstra 68], segments [Burroughs 64][Dennis 65], mémoire virtuelle et plus généralement virtualisation de ressources [Sumner 62] [Arden 66][Belady 66], liaison [Presser 72][Daley 68], partage et protection [Lampson 69], [Schroeder 72], [Fabry 74], systèmes distribués [Farber 72] et systèmes répartis de fichiers [Thomas 73]. Cependant la recherche en systèmes opératoires ne peut pas être réduite à l'élaboration de nouveaux concepts, elle porte également en elle une activité d'*ingénierie*, c'est à dire la recherche de la meilleure (au sens qualitatif et quantitatif) mise en œuvre de ces concepts sur une ou des architectures données. Or, la période qui nous concerne a été riche en évolutions : on est passé de gros systèmes partageant leur temps et leurs ressources entre les terminaux des utilisateurs à des stations de travail connectées par des réseaux de plus en plus rapides. L'évolution des technologies se traduit par le fait que des solutions jugées comme non réalistes sur les architectures des années 70 deviennent réalistes sur celles des années 90. Ainsi, par exemple, la réalisation du partage d'entités logiques (segments ou plus généralement objets) entre processus par gestion de copies multiples était-elle jugée impraticable lorsque le partage était envisagé entre processus d'une même machine [Saltzer 78] alors que c'est aujourd'hui, la solution couramment envisagée pour des processus implantés sur des machines différentes [Nitzberg 91] [Scott 92] ou des processeurs différents d'une même machine [Li 89]. Cet aspect d'ingénierie inhérent à toute recherche en systèmes explique pourquoi il est difficile :

- De comprendre les évolutions de la recherche en systèmes opératoires sans les remettre dans la perspective de l'évolution des technologies et par conséquent des architectures disponibles.
- D'avoir une recherche significative sans mise en œuvre d'un prototype. La définition et la réalisation d'un prototype sont souvent des travaux lourds qui ne peuvent pas être menés à bien par une seule personne, c'est nécessairement le résultat d'un travail d'équipe. Aussi aucun des résultats des projets que je présente dans ce mémoire ne sauraient m'être attribués en propre.
- De mener une recherche sur les systèmes opératoires sans liaison avec des industriels et en ignorant les évolutions des matériels (le fait d'avoir travaillé dans un centre qui a accueilli successivement le centre scientifique IBM et le centre de recherche Bull a été sans conteste une grande chance).
- De proposer des solutions qui soient en rupture complète avec les solutions existantes, c'est à dire qui ne puissent pas coexister avec les applications existantes,

remettant par là en question tous les investissements déjà effectués par les clients potentiels.

Le corps de ce mémoire est organisé comme suit : je présente dans une première partie un point de vue sur l'évolution des systèmes et des différents concepts pendant la période 1972-1992, puis je présente par ordre chronologique les différents travaux de recherche auxquels j'ai participé en les replaçant dans le cadre de l'évolution précédemment retracée. Enfin, dans une dernière partie, j'esquisse quelques directions dans lesquelles j'aimerais travailler dans le futur.

2. Évolution des systèmes opératoires depuis 1972

Une étude complète de l'évolution des systèmes opératoires et des concepts qu'ils mettent en œuvre dépasse le cadre de ce mémoire. Cette présentation vise seulement à donner des jalons qui permettent de situer les différents projets qui ont été menés à Grenoble dans le domaine des systèmes opératoires et auxquels j'ai participé.

Comme je l'ai expliqué précédemment, la recherche en systèmes est toujours dépendante des matériels dont on dispose. Les études auxquelles j'ai participé ne couvrent pas de façon exhaustive toutes les architectures qui se sont développées pendant la période qui nous intéresse. En particulier, nous n'avons jamais développé de systèmes pour des architectures massivement parallèles (calculateur vectoriel, hypercube, etc. …).

Je me limite donc, dans une première partie, à rappeler l'évolution des systèmes opératoires qui sont nos outils de travail. Dans une seconde partie, sur l'exemple de la gestion de l'information par le système, j'analyse comment ont évolué les problèmes et les solutions mises en œuvre depuis le début des années 70.

2.1. Évolution des outils

2.1.1. État des lieux en 70

Trois types de systèmes sont disponibles au début des années 70 : les systèmes opératoires dits généraux, les systèmes à partage de temps ou conversationnels, et les systèmes pour machines monolangages.

2.1.1.1. Les systèmes opératoires généraux

Né dans le milieu des années 60, ce type de système, dont l'archétype est le système opératoire OS/360 de la gamme 360 d'IBM, intègre dans une seule machine les fonctions de traitement des applications scientifiques et celles de traitement des applications dites de gestion. Ce système utilisait déjà un certain nombre de techniques nouvelles permettant une meilleure utilisation globale des ressources :

- *La multiprogrammation* :
  La mémoire centrale est partitionnée en zones de taille fixe (OS/ 360 MFT) ou de taille variable (OS/360 MVT) qui sont allouées à des programmes ; quand un programme se termine ou qu'il demande une entrée/sortie, le processeur est alloué à un autre programme.

- *Canaux d'entrées / sorties* :
  Les canaux sont des dispositifs asynchrones assimilables à des ordinateurs (dans le cas de l'IBM 360, ce sont des 360/40) spécialisés dans le traitement des entrées/sorties. Ils sont capables d'exécuter des programmes, dits "programmes canaux", qui sont chargés dans la mémoire principale de l'unité centrale à laquelle ils sont asservis. Le programme canal est lancé par le calculateur maître, puis exécuté par l'esclave (i.e. le canal) qui envoie une interruption à son maître quand le traitement est terminé. Cela permet de réaliser de façon asynchrone les entrées-sorties et le calcul. C'est évidemment l'existence des canaux qui a donné sa raison d'être au spoule d'entrées-sorties.

- *Spoule d'entrées et de sorties* :
  Le spoule permet de lire, dans les partitions libérées, les programmes non encore traités et cela simultanément au traitement d'un programme déjà chargé dans une autre

partition. Il permet également à un programme d'effectuer des entrées-sorties simultanément à son exécution.

- *Dispositif de protection de mémoire* :
  Le dispositif proposé est du type clé/verrou. A chaque partition de mémoire (donc à chaque programme) est associé un verrou (configuration de bits) qui est comparé, lors de chaque accès à la mémoire, à la clé contenue dans le mot d'état du programme courant . S'il n'y a pas correspondance entre la clé et le verrou, un déroutement pour violation de protection de mémoire est déclenché. Ce mécanisme permet de confiner les accès mémoire d'un programme dans la partition qui lui a été allouée.

Cependant la taille et la complexité du système opératoire OS/ 360, qui cherchait à satisfaire des besoins contradictoires (i.e. être efficace en environnement de gestion et en environnement scientifique et cela sur toutes les machines de la gamme 360 !), ont fait que ce système a toujours gardé au cours de ses nombreuses versions un nombre constant de bogues [Tanenbaum 87].

2.1.1.2. Les systèmes conversationnels

Les systèmes opératoires généraux restent essentiellement des systèmes à traitement par lots et ne sauraient satisfaire la demande pour une interaction plus directe entre les utilisateurs et la machine, requise par beaucoup de programmeurs. C'est à ce type de besoin que répondent les systèmes à partage de temps. Le principe du partage de temps, mis en œuvre par ces systèmes, résulte de la constatation qu'entre deux demandes de service effectuées par un utilisateur, il s'écoule un laps de temps important (parfois plusieurs secondes), si on le compare au temps demandé par le service pour être traité. Ce laps de temps peut être utilisé avec profit par l'unité centrale pour traiter les commandes des autres utilisateurs. Ce type de système fait donc non seulement de la multiprogrammation, mais également une allocation partagée du temps de l'unité centrale. Compte tenu des échelles de temps, chaque utilisateur connecté a l'impression qu'il dispose seul d'une machine qu'il utilise de manière interactive. Le premier système en temps partagé réellement opérationnel est le système CTSS développé en 1962 au MIT [Corbato 62], sur un IBM 7044 modifié. Les systèmes les plus représentatifs de cette catégorie sont Multics dont la conception commence en 1964 à la suite de l'expérience acquise avec CTSS, et CP/CMS mis en œuvre sur un modèle particulier de l'architecture 360, le 360/67, élaboré après une étude menée en 1966 sur un modèle d'architecture adaptée au temps partagé.

- *Multics*
  Le système et la machine (i.e. le GE-645) ont été conçus en même temps. Multics [Organick 72] est organisé autour de deux concepts : le segment (unité de désignation et d'adressage) et la procédure (l'appel de procédure est l'instruction de base de la machine). Il a su exploiter ces deux concepts dans tous leurs prolongements possibles et notamment pour ce qui est des segments : les techniques de partage et les mécanismes de liaison dynamique. Multics met en œuvre également :
  - Un système de stockage uniforme de l'information : les segments, qu'ils soient ou non persistants (i.e. ayant une durée de vie indépendante de celle du programme qui les crée) sont désignés et manipulés de la même manière ; la mémoire de stockage est la mémoire virtuelle ;
  - Un système de désignation symbolique structuré : l'ensemble des catalogues et des segments persistants forment une hiérarchie stricte ;
  - Un système de protection basé sur des listes d'accès attachées aux segments et sur le mécanisme d'anneaux.

Les idées développées dans Multics n'ont pas cessé depuis lors de féconder la recherche sur les systèmes. Multics est également à l'origine du système Unix[1] qui, paradoxalement, comme nous le verrons plus loin, apparaît aujourd'hui à nombre de chercheurs en systèmes comme l'un des principaux obstacles à l'innovation dans leur domaine.

• *CP / CMS et le 360/67*
L'originalité du 360/67 par rapport aux autres machines de la gamme 360 est son mécanisme de pagination. Ce mécanisme de pagination à deux niveaux a longtemps été utilisé pour entretenir une confusion fâcheuse entre pagination et segmentation. L'architecture du 360/67 supporte le système CP qui fournit à un utilisateur une machine 360/40 virtuelle (i.e. l'interface fournie par CP à ses utilisateurs est celle de la machine 360/40), sur lequel s'exécute un petit système interactif, CMS. CP/CMS a été un merveilleux outil d'enseignement et de recherche en systèmes opératoires par les manipulations sur "machine nue" qu'il permettait, à une époque où le coût des matériels (en achat et en exploitation) rendait difficile à un chercheur l'expérimentation sur la seule machine dont il disposait : la machine d'exploitation.

2.1.1.3. Les "machines langages"

Conçu pour permettre la mise en œuvre efficace de langages de haut niveau d'abstraction, ce type de machine est représenté par Multics (i.e. l'instruction de base est l'appel de procédure) dont nous avons déjà parlé, mais également par les machines Burroughs. Dans les années 60, Burroughs lance le système B5500, commercialisé en 1962, conçu comme plate-forme d'exécution (*runtime*) du langage Algol 60. Il sera bientôt suivi du système B6500/6700, puis du système B1700 [McKeag 76]. La B5500 a été la première machine à utiliser un adressage segmenté, un mécanisme câblé de gestion de pile et le partage des segments de code. Le but déclaré des architectes des machines langages était de compiler et d'exécuter efficacement des langages de haut niveau, mais on ne peut pas dire que ces machines aient remporté un grand succès commercial. Cependant, pour ce qui est des mécanismes de gestion de pile et de construction d'environnement d'exécution de procédure, ces machines ont influencé l'architecture des microprocesseurs. L'échec commercial des machines langages s'est vérifié, plus récemment, avec les machines Lisp. Les gains en vitesse des microprocesseurs dus à de nouvelles technologies combinées à de nouvelles architectures sont si rapides que les performances des machines spécialisées se trouvent rapidement dépassées par celles des machines à usage général. Les investissements des constructeurs n'ont donc pas le temps d'être rentabilisés ; ce qui semble condamner ce type de système. Les mesures qui ont conduit à l'abandon des architectures de microprocesseur CISC[2] au profit des architectures de microprocesseur RISC[3] fournissent des éléments techniques d'explication. En effet, on a constaté que les instructions complexes d'un processeur sont en fait très peu utilisées dans le code produit par les compilateurs car elles ne rendent que très rarement le service exact demandé. Comme de plus le code produit par les compilateur pour rendre le service demandé sur des processeurs RISC s'exécute plus vite, les architectures CISC semblent bel et bien condamnées [Kane 88].

2.1.1.4. Bilan

De cet état des lieux, nous pouvons tirer deux conséquences pour ce qui est de la recherche en système dans les années 70 :

---

[1] Unix est une marque déposée de Unix System Laboratories
[2] Complex Instruction Set Computer
[3] Reduced Instruction Set Computer

• Importance des problèmes d'allocation de ressources
Le coût du matériel et notamment celui de la mémoire fait que la fonction d'allocation des ressources est au premier rang des préoccupations des chercheurs en systèmes opératoires de l'époque.

• Pauvreté des outils réellement disponibles
Les systèmes qui, comme Multics, fournissent un support de haut niveau d'abstraction au développeur de sous-systèmes ne sont en fait pas disponibles. Les outils les plus diffusés à l'époque sont des machine nues, physiques ou virtuelles, qui présentent l'inconvénient de fournir des interfaces de très bas niveau (le jeu d'instructions des machines). Cet état de fait oblige tout chercheur en système qui veut expérimenter, à développer un grand nombre de lignes de code pour masquer les mécanismes de bas niveau de la machine. On doit remarquer, de plus, que le coût des matériels réserve leur utilisation comme outil d'expérience à un tout petit nombre de privilégiés.

2.1.2. Les facteurs d'évolution

Quatre facteurs ont largement influencé les objectifs poursuivis par la recherche en systèmes au cours de ces vingt dernières années : le développement de la mini-informatique et la diffusion du système Unix, l'apparition sur le marché des micro-noyaux, le développement de la téléinformatique, et enfin la généralisation des terminaux graphiques et des gestionnaires de fenêtres.

2.1.2.1. Développement d'Unix et des minicalculateurs

La mini-informatique est née en 1961 avec le DEC PDP-1. C'était une machine de 4k mots mémoire de 18 bits, qui coûtait à peu près 600 KF (moins de 5% du prix de l'IBM 7044) et qui pouvait traiter certaines instructions non numériques aussi vite que le 7044 [Tanenbaum 87]. Cette machine devait être suivie par beaucoup d'autres, la gamme PDP ayant culminé avec le PDP-11. Depuis, ce type de machine a évolué pour donner ce que nous appelons des stations de travail qui pour moins de la moitié de la valeur du PDP-11 (en francs courants) fournissent à l'utilisateur des puissances de calcul très supérieures à celle du 360 et des tailles de mémoire centrale de l'ordre de 32 Méga octets.

Ce qui a fait le succès du PDP-11 dans le milieu de la recherche, outre son prix très compétitif, a été le système Unix [Ritchie 74] [Bourne 83] dont les Bell Labs fournissaient les sources pour un prix symbolique[4] aux universités. Unix est ainsi devenu l'outil de développement privilégié pour la recherche en systèmes.

La première version de ce système a été développée sur le PDP-7 par K. Thompson qui avait participé à la réalisation de Multics. Il visait à faire un "Multics mono-usager". Évidemment les deux systèmes sont très différents. Unix hérite de Multics son système de désignation symbolique des fichiers et son langage de commande, mais hormis cela ils n'ont rien en commun. Unix devait être porté assez rapidement sur le PDP-11/20, puis sur le PDP 11/45. L'Unix original a rapidement évolué au fur et à mesure que les machines sur lesquelles il était mis en œuvre devenaient plus puissantes et qu'on lui demandait de devenir un système multi-utilisateur. Ainsi, les processus qui initialement ne pouvaient communiquer ou se synchroniser qu'au moyen des fichiers, se voyaient-ils offrir des files de messages, des portes (*socket*), des segments de mémoire partagée et des sémaphores. Cependant,

[4] Les systèmes MS/DOS et maintenant MS/Windows qui font le succès des PC dans le grand public n'ont pas eu le même succès dans le monde de la recherche … Il se peut qu'il en soit autrement de Windows NT développé par Microsoft.

quelques-unes de ces fonctions, notamment celles touchant au partage (mémoire et synchronisation), ne sont pas bien intégrées au système et donc ne sont pas efficaces, ce qui rend difficile leur utilisation intensive.

L'énorme valeur ajoutée par les nombreux logiciels développés sur Unix au cours des années n'a fait qu'asseoir un peu plus son succès. Son adoption comme système sur pratiquement toutes les stations de travail du marché a consacré Unix comme standard de fait, et de la compatibilité ou de l'intégration à Unix un critère d'évaluation de tout nouveau développement. Il faut cependant souligner que sur le plan des concepts et de l'ingénierie Unix est un retour en arrière par rapport à Multics.

2.1.2.2. Les micro-noyaux

L'un des thèmes récurrents de la recherche en systèmes dans les années 70 a été de définir un premier niveau de machine abstraite, souvent appelé noyau, qui soit une base commode et efficace pour le développement de machines abstraites spécialisées, généralement appelées sous-systèmes. Unix, comme nous l'avons déjà mentionné, a longtemps joué ce rôle de noyau de développement de sous-systèmes. Au fur et à mesure des enrichissements qui lui ont été apportés, il est devenu à son tour, sinon un monstre, du moins un système suffisamment volumineux et mal structuré pour que son transport d'une architecture à une autre représente un travail considérable. C'est à ce problème que tentent de répondre les micro-noyaux.

Les micro-noyaux visent à définir une machine abstraite "minimale", facilement portable[5] d'une machine à une autre et offrant un modèle client/ serveur de structuration d'un système. De plus, les micro-noyaux substituent à la notion "Unixienne[6]" de *processus*, unification d'un flot d'exécution avec un ensemble de ressources, la notion de *tâche*, ensemble de flots d'exécution partageant les mêmes ressources et notamment la même mémoire virtuelle [Freyssinet 91b]. Les flots d'exécution, qu'ils appartiennent ou non à la même tâche, ou à des tâches situées sur des sites différents, communiquent entre eux par messages au moyen d'objets de communication appelés portes. Les micro-noyaux fournissent un mécanisme (appelé paginateur externe) permettant un contrôle souple de la mémoire virtuelle associée à une tâche. Ils gèrent les entrées-sorties de bas niveau. Enfin, un sous-système est vu comme un ensemble de tâches qui coopèrent pour fournir un ensemble cohérent de services et d'interfaces.

Les deux micro-noyaux actuellement disponibles sur le marché, Chorus [Rozier 88] et Mach [Acetta 86] fournissent l'ensemble des services systèmes (entrées-sorties de haut niveau, fichiers) à travers Unix qui est construit comme un serveur unique (Mach) ou un ensemble de serveurs coopérant (Chorus). Les mécanismes fournis par Mach, comme par Chorus, tant pour la gestion du partage de la mémoire entre des tâches s'exécutant sur des sites différents, que pour la communication entre les sous-systèmes qu'ils supportent (et notamment Unix), ouvrent de nouvelles possibilités (notamment l'exploration des possibilités offertes par les processeurs à 64 bits d'adresse [Bryant 91]) aux recherches dans le domaine des systèmes répartis. Ces noyaux pourraient jouer un rôle analogue à celui des machines abstraites des années 70, car ils permettent la coexistence de plusieurs systèmes sur la même machine.

---

[5] Le transport du micro-noyau Mach 3.0, sans les *drivers*, d'une architecture à une autre est estimé à 12 hommes/mois par les ingénieurs de l'Open Software Foundation

[6] Nous devons remarquer que des systèmes comme Esope, développé à l'INRIA au début des années 70, fournissaient déjà ces notions de tâche et de threads. En fait, les micro-noyaux n'apparaissent comme nouveaux que parcequ'ils arrivent après 15 années d'Unix.

2.1.2.3. La téléinformatique et les réseaux d'ordinateurs

La téléinformatique et l'intégration progressive de la fonction de communication dans les systèmes, qui étaient à la fin des années 60 et au début des années 70 un sujet d'étude pour les chercheurs, sont devenues un outil informatique essentiel pour l'industrie. Le modèle d'architecture de système de communication OSI[7] de l'ISO[8] est devenu une norme largement reconnue par l'industrie informatique, si bien qu'il devient de plus en plus facile de faire communiquer des machines provenant de constructeurs différents [Tanenbaum 89]. Si la répartition permise par les réseaux n'a introduit que peu de concepts nouveaux, elle a en revanche rendu beaucoup plus difficile l'ingénierie associée à leur mise en œuvre. On distingue en première approximation deux types de systèmes : les réseaux généraux ou télématiques et les réseaux locaux.

- *Les réseaux généraux ou télématiques*

  Le développement des réseaux généraux a commencé aux États Unis en 65 avec le projet ARPANET [Schultz 88] et en France en 72 avec le projet Cyclades [Pouzin 73 - 82]. Ils ont connu depuis lors un développement considérable. Ils permettent de réaliser l'interconnexion de réseaux ou de machines à l'échelle d'un continent ou entre continents. Ils étendent la portée de certains services de communication disponibles sur les machines. Par exemple, des outils Unix comme les *news*, le *ftp anonymous* ou le *mail*, permettent de disposer d'une bibliothèque quasi mondiale et d'un forum permanent de dialogue avec les autres chercheurs.

- *Les réseaux locaux*

  Les réseaux locaux permettent de réaliser l'interconnexion de machines à l'échelle d'un bâtiment ou d'une entreprise. Par leurs performances (de l'ordre de 10 Megabits/ s pour un Ethernet [Metcalfe 76] à 100 Megabits/ s pour un FDDI[9]) et leur faible coût, ils ont connu un grand développement, qui combiné à celui des stations de travail, a donné naissance à un nouveau type d'architecture de systèmes. Aux systèmes conversationnels classiques se sont substitués progressivement des réseaux de machines où l'utilisateur travaille sur sa station personnelle, peut partager de l'information et communiquer avec d'autres utilisateurs, et accéder à des services généraux trop coûteux pour que sa station en dispose localement. On peut citer, sans vouloir être exhaustif, différents services qui sont venus enrichir Unix : NFS [Sandberg 85] ou AFS [Morris 86] qui permettent d'accéder à des fichiers stockés sur des serveurs spécialisés ou sur d'autres stations de travail ; le service de spoule de gestion d'imprimantes qui permet le partage des imprimantes entre les stations connectées ; l'appel de commande à distance qui permet d'exécuter une commande sur une autre machine ; enfin un service de courrier électronique qui permet l'échange de messages entre utilisateurs du réseau.

Les progrès des outils de communication continuent à être extrêmement rapides et tendent à réduire le fossé existant entre les performances d'un réseau télématique et celle d'un réseau local (les réseaux ATM[10] dont les performances annoncées sont de l'ordre des 600 Méga bits/ s. sont envisagés tant pour des réseaux privés que pour l'architecture B-ISDN[11] du CCITT[12]). Ces progrès permettent d'envisager de gérer à la fois des images, du

---

[7] Open System Interconnection

[8] International Standards Organization

[9] Fiber Distributed Data Interface

[10] Asynchronous Transfer Mode

[11] Broadband Integrated Service Digital Network

[12] Comité Consultatif International pour le Télégraphe et le Téléphone

son, et du texte (information dite multimédia) dans un système opératoire. Ils permettent également d'envisager un système réparti non plus comme gérant une cellule (ensemble de machines connectées par un réseau local), mais comme gérant un ensemble de cellules interconnectées par un réseau B-ISDN [Greaves 92]. Une autre conséquence prévisible de la mise en œuvre de ces réseaux est leur utilisation possible pour implanter non seulement des applications réparties (i.e. à gros grain de parallélisme) mais des applications qui sont actuellement implantées sur des architectures parallèles dédiées. Cela ne va pas être sans conséquences sur l'architecture et les interfaces des systèmes répartis. En effet, les applications parallèles ont des besoins spécifiques, notamment pour ce qui est du contrôle de l'exécution des processus qu'elles utilisent, qui sont très différents de ceux des applications réparties classiques (systèmes de réservation de place, courrier électronique, agenda, etc.).

2.1.2.4. Les terminaux graphiques et les gestionnaires de fenêtres

Les terminaux graphiques et leur complément nécessaire, les gestionnaires de fenêtres, qui permettent le partage de l'écran entre plusieurs sessions de travail, ont permis la réalisation de logiciels utilisables par des non-informaticiens (*user-friendly software*). Ceci a fait beaucoup pour la généralisation de l'utilisation de l'outil informatique.

Le premier système à intégrer des terminaux graphiques comme support des interactions avec ses utilisateurs est le système développé à Xerox Parc sur l'Alto. La motivation première de ce projet, commencé en 1973, était la définition d'un outil efficace capable de satisfaire les besoins d'un utilisateur unique (i.e. une station de travail). Les concepteurs de l'interface homme-machine ont essayé de modéliser les possibilités offertes par les outils manuels de travail [Thacker 83].

Il est clair que les conditions de travail du réalisateur de prototypes sous Unix se sont grandement améliorées grâce à des outils comme *Xterm* qui permet de disposer d'une session de travail dans une fenêtre ou *Xdbx* qui fournit une interface graphique de mise au point. Donc le développeur de prototypes sur Unix dispose de meilleurs outils. En revanche du point de vue du chercheur en systèmes voulant réaliser une maquette pour valider ses résultats, le jugement est différent. En effet, les résultats en systèmes sont extrêmement difficiles à présenter en démonstration et les utilisateurs potentiels ont souvent tendance à juger le système qu'on leur propose à la qualité de son interface graphique. Or, la réalisation d'interfaces graphiques à l'aide du gestionnaire de fenêtre X et de la boîte à outils graphiques OSF/ Motif est un travail excessivement fastidieux et qui peut facilement doubler le volume du code à réaliser. L'étude d'outils de haut niveau permettant de construire automatiquement des interfaces graphiques est un domaine de recherche en soi dont les premiers résultats commencent à arriver sur le marché.

2.2. Évolution de la gestion de l'information

Parmi les quatre points que nous venons de citer, l'importance croissante prise par les réseaux locaux est plus particulièrement à l'origine d'une profonde mutation de la nature même des systèmes opératoires. L'évolution des systèmes conversationnels vers des systèmes distribués gérant des stations de travail puissantes interconnectées par des réseaux locaux ne s'est faite que progressivement. Il a d'abord fallu intégrer au système la fonction de communication, puis des services permettant par des requêtes explicites l'accès à des ressources distantes (par exemple FTP). Puis ont été proposés des systèmes fournissant un espace de désignation et permettant l'intégration des machines individuelles dans cet espace. L'interface commune est généralement celle définie par les primitives d'un système d'exploitation (cas des diverses extensions réparties d'Unix comme la Newcastle Connection [Brownbridge 82]). Enfin, et on arrive à la période actuelle, sont proposés des systèmes à

haut niveau d'intégration, où chaque poste de travail est considéré comme un composant d'un système global réparti. Contrairement aux systèmes précédents, les fonctions liées à la répartition ne sont pas surajoutées à un système déjà existant, mais prises en compte dès l'origine. Les travaux dans ce domaine sont moins avancés que dans les deux précédents, cependant plusieurs systèmes de ce type ont été réalisés : Apollo/ Domain [Leach 86], Locus [Popeck 85] pour ne citer que les premiers en date.

Le but de cette section est d'illustrer, en prenant comme exemple la gestion de l'information, comment ont évolué, du fait de l'intégration progressive de la répartition, les problèmes et les solutions qui leur ont été apportées.

La gestion de l'information recouvre les fonctions assurant la désignation, le partage, la protection et, si l'information est permanente, son stockage. Les techniques utilisées pour le stockage de l'information n'ayant que peu évolué, nous ne nous y intéressons pas[13]. Bien que les trois fonctions qui nous intéressent soient étroitement liées, il nous a semblé plus commode pour la clarté de l'exposé de présenter d'abord les problèmes se rapportant à la désignation et au partage, puis ceux se rapportant à la protection.

2.2.1. Désignation et partage

Comme nous l'avons signalé précédemment, les retombées du système Multics sont d'une grande importance pour l'évolution de la recherche, et notamment en ce qui concerne les problèmes liés à la désignation et au partage de l'information. Les concepts mis en œuvre par Multics devaient, en particulier, influencer les solutions mises en œuvre dans les projets de recherche en systèmes opératoires centralisés menés en France au début des années 70 : le projet Esope [Bétourné 70] développé à l'INRIA, le projet SAR [Bekkers 75] développé à l'Université de Rennes et les projets Gemau [Guiboud-Ribaud 74] et Ours [Briat 76a] développés à l'Université de Grenoble, qui tous les quatre s'inspirent fortement des concepts mis en œuvre dans Multics, et les simulent sur le CII 10070, machine n'offrant pas d'adressage segmenté.

Du point de vue de la désignation et du partage, un certain nombre de problèmes qui ne se posaient pas, ou avec moins d'acuité, sur des architectures centralisées ont émergé avec la répartition [Saltzer 78] :

- Le système de désignation doit permettre de changer un objet de système de stockage (mobilité) sans avoir à changer les références sur cet objet existant dans d'autres objets. Cette propriété recouvre à la fois les propriétés de transparence et d'indépendance d'un nom par rapport à la localisation physique de l'objet qu'il désigne.
- Le partage d'objets peut intervenir entre systèmes, de telle manière qu'un objet d'un système peut désigner des objets et être lui-même désigné par des objets qui sont physiquement stockés sur d'autres systèmes. Le partage implique dans ce cas le maintien de contextes (base d'interprétation d'un nom) qui puissent travailler avec des générateurs indépendants de noms uniques.
- La nécessité de résister à la panne d'un ou plusieurs sites conduit d'une part à gérer plusieurs copies d'un même objet, ce qui pose le problème de leur désignation et du maintien de leur cohérence, et d'autre part à avoir des générateurs de noms indépendants entre les différents systèmes, ce qui pose le problème de la coordination de ces différents générateurs.

---

[13] Les techniques de stockage sur disques des systèmes de fichiers distribués comme NFS ou AFS ne me semblent pas en effet être différentes de celles utilisées par les systèmes centralisés classiques. Il me semble que les évolutions dans ce domaine sont venues plutôt des systèmes de bases de données.

De nombreux systèmes distribués proposent des solutions plus ou moins complètes à ces problèmes ; on doit distinguer les systèmes qui proposent un système de fichiers distribués de ceux qui proposent un système à stockage distribué et uniforme des données :

2.2.1.1. Les systèmes de fichiers distribués

Les systèmes de fichiers distribués doivent mettre en œuvre un espace global de désignation symbolique qui prenne en compte les problèmes de disponibilité et de mobilité des fichiers. En outre, ces systèmes doivent, pour des raisons de compatibilité, définir et mettre en œuvre une sémantique du partage qui ne soit pas trop éloignée de celle d'Unix. Enfin ils doivent garantir des temps d'accès aux fichiers qui soient acceptables. Quelques uns des principaux projets développés au cours de ces dix dernières années :

- La Newcastle connexion, qui interconnecte un ensemble de systèmes Unix sans modification du noyau, et son système de fichiers distribués Unix United [Brownbridge 82],
- Locus, qui est un système distribué développé à UCLA, complètement compatible avec Unix et dont le système de fichiers prend en compte la duplication de fichiers, la mobilité et dans une certaine mesure la résistance aux pannes [Popeck 85] [Weinstein 85],
- NFS, qui est un service distribué de fichiers développé par Sun et qui est maintenant un composant standard d'Unix [Sandberg 85],
- AFS, qui est le système de fichiers de l'environnement de programmation distribué Andrew développé à CMU, et qui est maintenant disponible sur Unix [Howard 88],

sont analysés dans [Levy 90].

2.2.1.2. Les systèmes à stockage distribué et uniforme des données

La plupart des systèmes à stockage uniforme des données sont des systèmes d'exploitation à objets [Hamilton 91]. Le concept d'objet généralise la notion de variable typée issue des langages de programmation [Wegner 90]. Les objets se présentent au niveau du système comme des modules qui contiennent les données et les opérations exportées par leur interface. L'objet est à la fois une unité de désignation et de protection. Il a pour la première fois été mis en œuvre comme constituant de base dans le système Hydra [Wulf 74] ; cependant beaucoup des systèmes existants fournissent des primitives de gestion d'objets sans que le système lui même soit défini en termes d'objets [Shapiro 91].

Un objet est en général désigné au niveau du système par un identificateur unique dans l'espace et dans le temps appelé *uid*. La désignation symbolique des objets est en général un service mis en œuvre au dessus du système et la traduction d'un nom symbolique en *uid* est à la charge de l'utilisateur ou de l'environnement d'exécution du langage qui l'utilise. L'allocation des *uid* est une fonction distribuée, et l'unicité des *uid* est alors garantie par une partition statique de l'espace des *uid* entre les différents sites. L'unicité des *uid* garantit que les objets pourront être mobiles dans l'espace de stockage. Nous allons nous intéresser d'abord à la façon dont un objet est rendu accessible à une tâche (i.e. ensemble de flots d'exécution partageant la même mémoire virtuelle), puis nous verrons comment le partage d'un objet entre tâches est mis en œuvre.

accès à un objet par une tâche

L'objet est rendu accessible à une tâche soit parce qu'il est indissociable de la tâche (i.e. l'objet fait partie statiquement de la définition de la mémoire virtuelle de la tâche, on dit souvent que les objets sont actifs), soit par une opération de couplage de l'objet dans la mémoire virtuelle de la tâche. Dans la première catégorie rentrent, par

définition, les systèmes à objets actifs (citons par exemple : Argus [Liskov 85], Arjuna [Shrivastava 89] et Emerald [Black 86]) ; dans la seconde, on distingue les systèmes où le couplage de l'objet peut être explicite (par exemple Exodus [Schuh 90]), de ceux où le couplage est fait implicitement au premier accès à l'objet. Dans ce dernier cas on peut distinguer deux techniques :

- Les *uid* sont les adresses des objets dans un espace fictif qui est implicitement mis en bijection avec la mémoire virtuelle de chaque tâche. Cette technique cherche à tirer parti du fait que l'on va disposer d'un espace virtuel très grand (machine à 64 bits d'adresse) ; elle est utilisée dans Amber [Chase 89], son successeur Opal [Chase 92], dans EOS [Grubert 92] et partiellement dans Psyche [Scott 89] [Garrett 92].
- Le premier appel à une méthode sur un objet provoque le couplage de l'objet dans une mémoire virtuelle dans laquelle s'exécute le processus appelant. C'est le cas, par exemple, du sytème distribué tolérant les fautes Gothic [Banatre 90], du système distribué Guide-1 [Balter 89] et de son successeur Guide-2 [Freyssinet 91b].

Le système Clouds [Dasgupta 90] utilise une technique qui ne rentre pas dans la classification sommaire que nous venons de faire ; en effet dans Clouds chaque objet est couplé dans une mémoire virtuelle séparée qui devient celle de tout flot d'exécution (unité d'exécution d'un programme dans Clouds) devant exécuter une méthode sur l'objet.

Partage

Dans le cas des systèmes à objets actifs, un objet n'est en général jamais partagé par deux tâches ; tout appel de méthode sur un objet se traduit par un envoi de message à la tâche qui lui est attachée et est donc exécuté par l'un des flots d'exécution de cette tâche.

Dans le cas des systèmes à couplage explicite ou implicite, deux techniques peuvent être utilisées :

- Les objets modifiables ne sont présents que sur un seul site et sont partagés par les tâches présentes sur ce site en utilisant les mécanismes classiques de partage de mémoire sur un système centralisé ; les objets non modifiables sont dupliqués sur tous les sites qui les utilisent. C'est le cas des systèmes Amber et Guide [Balter 91],
- Les objets sont dupliqués sur tous les sites contenant une tâche qui les a couplés. C'est le cas de tous les autres systèmes que nous avons mentionnés plus haut. La gestion de la cohérence entre les copies multiples est faite en général au niveau de la page [Banâtre 91], par le gestionnaire de pagination. Le niveau de cohérence entre les copies peut être strict (comme si tous les sites partageaient une même mémoire contenant l'objet) ou obéir à des critères plus lâches [Ramachandran 89][Boyer 91b] [Carter 91].

2.2.2. Protection et sécurité

Les concepts propres à garantir la confidentialité et l'intégrité des informations d'un système se sont développés dans les années 60 et ont vu leur première mise en œuvre dans les systèmes multi-programmés et les systèmes à partage de temps [Lampson 69]. Les mécanismes fournis par Multics basés sur : les listes d'accès, le contrôle hiérarchique des spécifications d'accès à l'information (mécanisme des anneaux) et sur l'authentification des utilisateurs ne permettent pas l'implantation de politiques non hiérarchisées de protection et notamment l'écriture de sous-systèmes mutuellement méfiants [Saltzer 74]. Ces considérations peuvent expliquer que des mécanismes plus souples aient été développés [Krakowiak 85]. Ces mécanismes sont basés sur les capacités qui regroupent la désignation d'un objet et un ticket d'accès à l'objet désigné [Fabry 74]. Ce concept, introduit par Dennis

[Dennis 66], est d'abord utilisé dans le système Hydra [Wulf 74], puis sera à la base de l'architecture et du système de la Plessey 250 [England 75] et de plusieurs autres machines dont la dernière en date, l'iAPX 432[Kahn 81], n'a jamais été complètement opérationnelle. Les systèmes à capacités sont étudiés dans [Levy 84].

La spécificité des systèmes distribués vis à vis de la sécurité porte sur trois points - on ne peut pas faire confiance aux stations de travail connectées, les systèmes de communication ne sont pas sûrs, on doit répartir les informations et les mécanismes qui concourent à la sécurité - nous allons les analyser dans les sections suivantes :

2.2.2.1. Confiance dans les stations connectées

Dans un système centralisé, la machine est en général dans une salle protégée et le chargement du système est une opération suffisamment contrôlée pour que l'on puisse assurer que le système chargé est bien le système originel. Cela devient totalement impossible dans le cas de systèmes connectant des stations de travail où chaque usager peut facilement passer en mode maître et donc charger un système court-circuitant les mécanismes de protection [Deswarte 81].

On ne peut pas faire face à cette situation sans supposer que, sur un certain nombre de sites au moins, on puisse avoir confiance dans le système chargé. Ainsi dans le système Andrew, le système global est-il vu comme un ensemble de stations non sûres appelées *Virtue* connectées à un ensemble de serveur sûrs appelés *Vice* [Satyanarayanan 89]. Dans Amoeba [Mullender 86] [Tanenbaum 90], certains serveurs sont sûrs et on peut disposer d'un matériel, appelé *F-Box*, qui contrôle l'accès au réseau[14].

Une seconde mesure [Ingels 90] consiste d'une part à limiter ce qu'un utilisateur peut faire à partir d'une station non sûre (ainsi dans Andrew, un fichier global ne sera jamais chargé sur un disque local), et d'autre part à mettre en œuvre des mécanismes d'authentification contrôlant tout accès aux serveurs distants.

2.2.2.2. Sécurité du système de communication

Dans les systèmes centralisés, les communications entre l'usager et le système, ou à l'intérieur du système, utilisent la mémoire. Des mécanismes matériels permettent de contrôler ces échanges : protection des instructions d'entrées/sorties et mécanismes de protection mémoire. De tels mécanismes existent évidemment toujours sur les stations de travail. Cependant, beaucoup de communications ont lieu entre machines et transitent donc par le réseau qui est, en général, particulièrement sensible à l'intrusion. Deux remèdes sont proposés à ce problème : le chiffrement afin de garantir la confidentialité des informations circulant sur le réseau, et l'authentification réciproque des interlocuteurs. Dans le système Andrew, plusieurs modes de protection de la connexion avec un serveur distant sont prévus : authentification simple, authentification et chiffrement des en-têtes des paquets, authentification et chiffrement complet des paquets. Le chiffrement utilise des clés secrètes et l'authentification utilise des serveurs mis en œuvre sur des stations sûres exécutant un protocole dérivé du modèle de Needham et Schroeder [Needham 78]. Le système Athena développé au MIT et son serveur d'authentification Kerberos [Steiner 88] utilisent les mêmes techniques.

Des normes concernant la sécurité des systèmes de communication qui spécifient notamment le chiffrement des données transmises, ont été produites [ISO 88]. Cependant,

---

[14] En l'absence de *F-box*, les serveurs (de fichiers et de calcul) sont protégés en les enfermant dans une salle physiquement protégée.

en l'absence des dispositifs matériels ad hoc, le coût de mise en œuvre et d'exploitation de ces dispositions semble les réserver à des systèmes spécialisés.

2.2.2.3. Gestion des informations de protection

Une fois réglés les problèmes d'authentification, il reste à exprimer la politique d'autorisation, c'est-à-dire de contrôle de l'accès aux objets. Les problèmes nouveaux sont liés à la répartition des structures de données : listes d'accès associées à chaque objet ou capacités associées aux sujets. On peut soit les centraliser comme dans le système Andrew, mais alors, pour des raisons de disponibilité, on doit maintenir de manière cohérente des copies sur un certain nombre d'autres sites ; soit les distribuer sur un, plusieurs ou tous les sites, comme dans Amoeba, mais il faut alors des protocoles adaptés pour les consulter et les mettre à jour.

2.3. Bilan

Le développement des stations de travail et des réseaux de communication a conduit des systèmes conversationnels centralisés aux systèmes répartis. Cette évolution n'a que peu transformé en termes de spécifications les fonctions attendues de ces nouveaux systèmes par rapport aux spécifications qui étaient celles de Multics : garantir de façon fiable, efficace, et permanente un ensemble de services : stockage de l'information avec garantie de fiabilité et de confidentialité, accès à des informations partagées, communication, et utilisation d'outils divers pour le développement et la mise au point d'applications. Cependant, deux aspects ont considérablement changé depuis cette époque :

- Les modalités de prestation de service et de partage des ressources sont différentes. Les fonctions de traitement et de stockage ne sont plus confiées à un organe central unique multiplexé entre un ensemble d'usagers, mais chaque usager dispose d'un poste de travail puissant et accède en outre, via un réseau à grand débit, aux ressources fournies par un ensemble de serveurs coopérants.
- Les interfaces présentées par le système aux utilisateurs ont également changé : les possibilités de visualisation synthétique et d'interaction rapide permises par les écrans à points et les souris ont transformé la notion de langage de commande, et rendu aisée grâce au multi-fenêtrage la gestion simultanée d'activités multiples indépendantes ou corrélées.

Les techniques mises en œuvre pour la construction de tels systèmes, et que nous avons illustrées par la présentation des techniques utilisées pour la gestion de l'information, vont dans le sens d'une intégration de plus en plus poussée en essayant de rendre invisible (mais on dit souvent “transparente” dans le jargon de la communauté système !) la distribution de l'information et plus généralement des ressources.

3. Présentation de mes travaux

A l'exception d'une courte incursion dans le domaine des modèles mathématiques de systèmes [Rousset de Pina 71] et dans celui des interfaces homme-machine [Quint 86], mes travaux ont porté principalement sur trois domaines : le développement d'outils pour la recherche sur les systèmes opératoires, le développement de systèmes de communication et les systèmes répartis.

3.1. Outils pour la recherche en systèmes

Comme nous l'avons rappelé dans la section précédente, le début des années 70 se caractérise pour le chercheur dans le domaine des systèmes opératoires :

• par une grande richesse dans le domaine conceptuel : les principaux concepts permettant d'expliquer les différentes fonctions d'un système opératoire sont élaborés,

• par une grande pauvreté dans le domaine des outils d'expérimentation : les machines sont des ressources coûteuses, peu accessibles aux chercheurs pour réaliser des expérimentations.

Il y a donc un grand besoin d'outils qui permettent aux chercheurs d'expérimenter. Les observations faites sur les systèmes du type de l'OS 360, dits à usage général, montrent à l'évidence qu'il est difficile d'avoir des politiques de gestion des ressources qui soient optimales pour tous les types d'applications. Il faut donc plutôt qu'implanter des politiques au niveau du système, donner des mécanismes qui permettent la mise en œuvre de politiques différentes au niveau de sous-systèmes spécialisés (i.e. traitement par lots conversationnel, base de données, etc. ...).

Les outils proposés sont de deux types : les machines virtuelles et les machines abstraites. La première catégorie est représentée par des systèmes comme CP 67 et VM 70 ou le simulateur de 10070 [Bennett 73], elle se caractérise par le fait qu'elle fournit, grâce à un logiciel appelé hyperviseur, des exemplaires simulés d'une machine réelle. La seconde catégorie offre à ses utilisateurs, grâce à un logiciel appelé noyau, des exemplaires d'une machine qui n'existe pas sur le marché et qu'on appelle machine abstraite. Par opposition à celle d'une machine abstraite, l'utilisation d'une machine virtuelle ne débarrasse pas l'utilisateur (l'écrivain de système) de la charge d'écrire un premier niveau de logiciel prenant en compte les mécanismes directement liés au matériel : adressage, déroutements, interruptions, entrées-sorties. Les machines abstraites, au contraire, permettent le développement d'un système comme une extension du noyau et des systèmes déjà écrits. Elles fournissent en général des fonctions de plus haut niveau que celles des matériels existants tout en permettant aux systèmes développés par les utilisateurs de définir leur propre politique d'utilisation des ressources. C'est pour cela que c'est la définition et le développement de ce dernier type d'outil qui ont principalement motivé les chercheurs, au moins jusqu'à ce qu'ils aient disposé d'Unix.

Les deux projets, Gemau et Ours, que nous présentons maintenant, s'inscrivent dans cette activité de développement de noyaux.

3.1.1. Le projet GEMAU

3.1.1.1. Contexte et motivations

Gemau (GEnérateur de Machine AUtonome) est un projet qui a été mené de 1972 à 1975 par une équipe regroupant des chercheurs de Bull (cinq ingénieurs) et du laboratoire IMAG

(deux enseignants-chercheurs et trois thésards) [Guiboud-Ribaud 74, 75]. Son but était de fournir un noyau pour la construction de sous-systèmes, mais, contrairement à Hydra développé dans le même but à la même période et qui met l'accent sur des mécanismes permettant de faire cohabiter différentes politiques d'allocation des ressources physiques, Gemau s'intéresse plus à la description et au test d'architectures de système. Cependant Hydra et Gemau ont des points communs par l'importance qu'ils donnent au partage et à la protection. Les mécanismes de protection qu'ils offrent sont basés sur les capacités.

3.1.1.2. Description générale du noyau

Désignation, adressage et protection

Le système Gemau est un système à capacités. Les capacités contiennent, outre le nom unique des objets, leur type, et un champ exprimant les droits du détenteur de la capacité sur l'objet. Les capacités sur les objets persistants sont regroupées dans des entités appelées *annuaires* et sont désignées par un nom logique. Un annuaire joue donc à la fois le rôle de catalogue pour les capacités qu'il regroupe (fonction d'association nom logique - capacité) et de domaine de protection (regroupement d'objets avec leurs droits). Les annuaires sont organisés hiérarchiquement à partir d'une racine qui contient les objets du noyau Gemau. Les objets de base gérés sont le segment procédure, le segment de données ou fichier, le périphérique, l'annuaire. A chaque type de base sont associés les opérateurs qui lui sont applicables (par exemple : *lire*, *écrire*, *exécuter* pour un segment procédure). Les champs droit et type d'une capacité permettent ainsi de connaître le sous-ensemble des opérateurs applicables à l'objet désigné. L'environnement d'exécution (i.e. ensemble des objets que l'on peut nommer) d'une procédure est constitué par les capacités qu'elle a reçues en paramètres et par le sous-arbre dont la racine est l'annuaire où elle a été installée et qui sert de base d'interprétation des noms qu'elle utilise. Un segment procédure ne peut être installé que dans un seul annuaire. Le changement de domaine de protection est provoqué par l'exécution d'un appel à une procédure qui peut être désignée depuis l'annuaire courant, mais qui est installée dans un autre annuaire.

Un objet peut être lu, écrit, ou exécuté soit au moyen des primitives systèmes *lire*, *écrire*, ou *exécuter*, soit directement au moyen des instructions de la machine. Dans ce dernier cas, les instructions de la machine n'étant pas interprétées par Gemau, la capacité sur l'objet doit être désignée par un nom local au processus effectuant l'opération. La traduction nom global-nom local est faite soit implicitement lors de l'appel de la primitive système *exécuter* sur un segment procédure, soit explicitement par l'appel de la primitive système de liaison *lier*. L'adresse d'un élément d'un segment est une adresse segmentée du type (*rb*, *dep*) ou *rb* désigne le nom local du segment et *dep* un déplacement dans le segment.

Les objets qui ne sont désignés que par des noms locaux (i.e. il n'existe pas de capacité les désignant dans le graphe orienté des annuaires) sont considérés comme temporaires et sont automatiquement détruits dès lors qu'il n'existe plus aucun nom local les référençant.

Structures d'exécution

Les structures d'exécution implantées par Gemau sont des processus ordonnés par la relation père-fils. L'environnement d'un processus à sa création est celui de son père. Tous les processus de Gemau descendent donc du même ancêtre dont l'environnement d'exécution est la racine du système. Le contrôle élémentaire d'un processus s'effectue au moyen d'appels procéduraux (appel et retour d'un segment procédure). L'état d'un processus à un instant donné est caractérisé par l'ensemble de ses noms locaux, par l'environnement de la procédure qu'il exécute, et par l'état du processeur.

Gemau gère, pour chaque processus, un système d'événements (exceptions) synchrones ou asynchrones qui s'inspire de celui de Multics et qui simule le système d'interruption du processeur. Un événement synchrone peut être soit l'un des événements définis par le système (i.e. déroutement du processeur, exceptions attachées aux primitives du noyau), soit programmé (signalé par une procédure). Les événement asynchrones sont utilisés par un processus père pour contrôler ses fils. Le contrôle permis est du type maître/ esclave.

3.1.1.3. Bilan

Il y a plusieurs façons d'évaluer un projet scientifique de la taille de Gemau : par ses qualités scientifiques (innovations et progrès du savoir-faire), par les qualités techniques de l'outil construit relativement aux objectifs fixés, enfin par le transfert de technologie et de savoir-faire. Nous allons successivement nous intéresser à ces trois aspects.

Qualités scientifiques

Les idées mises en œuvre dans Gemau pour l'espace de noms, l'adressage, les structures de contrôle et d'exécution s'inspirent de celles de Multics. Cependant l'utilisation des capacités, même si le concept date de 1966 [Dennis 66], et les mécanismes de protection fournis par Gemau sont originaux. En outre Gemau, considéré comme un tout, représente une architecture originale et cohérente.

Le travail effectué sur les capacités reste un travail tout à fait actuel dans le domaine de la protection d'accès à l'information. On peut cependant regretter que le concept de module (encapsulation d'un ensemble de procédures avec les données qu'elles manipulent) ne soit pas un concept de base du système, pas plus qu'il ne l'était dans Hydra [Wulf 82] ; on peut certes construire des modules sur Gemau, mais c'est une opération lourde.

Qualités de l'outil

Le système Gemau a été mis en œuvre sur CII 10070, machine sans adressage segmenté et dont la mémoire virtuelle, réalisée par un dispositif de topographie, avait la particularité d'être de même taille que la mémoire physique. La construction du noyau a respecté le modèle de construction en couches proposé pour l'écriture des sous-systèmes hôtes. Le système conversationnel construit au-dessus du noyau Gemau [Laforgue 75] a servi à démontrer la puissance du modèle de désignation, de partage, de protection et d'exécution pour la construction de sous-systèmes spécialisés (moins de trois mois). Comme tous les systèmes de cette période, Gemau a apporté un soin particulier à la gestion du partage des ressources et notamment de la mémoire qui était à cette époque la ressource critique. Cependant, le système n'a pas été exploité dans des conditions réelles suffisamment longtemps pour obtenir des mesures probantes. Toutefois, on doit noter que les temps de réponses du sous-système conversationnel, compte tenu de la puissance de la machine support, étaient comparables à ceux de CMS.

Transfert

Gemau, quoique développé en collaboration avec le centre de recherche CII, n'a pas eu de retombée industrielle évidente. En effet, la CII n'a montré que peu d'intérêt pour le type d'architecture mise en œuvre par Gemau qui était considéré comme trop innovant. On ne peut que le regretter quand on compare le sous-système conversationnel tournant sur Gemau à Unix sur PDP-11.

C'est probablement sur le point du transfert des résultats que l'on peut être le plus critique. L'effort effectué pour communiquer les résultats du projet Gemau n'a pas été suffisant (une seule publication dans un symposium et cinq thèses dont une thèse d'état) vu l'importance du projet et des résultats obtenus. Cependant, c'est la crédibilité acquise par

l'équipe de développement de Gemau dans la communauté grenobloise qui a permis à une partie de l'équipe de pouvoir disposer des machines et du financement lui permettant d'entreprendre le projet Ours que nous présentons maintenant.

3.1.2. Le projet Ours

3.1.2.1. Contexte et motivations

Ours (Outil de Recherche en Système) se proposait, compte tenu de l'expérience acquise dans Gemau, de développer sur un Iris 80 bi-processeur, un noyau de système extensible capable de supporter un grand nombre de sous-systèmes, de fournir un sous-système conversationnel permettant de fournir au développeur d'applications des outils de développement minimaux (édition, compilation, exécution, conservation et mise au point de programmes), et de valider l'adéquation des outils de construction de systèmes proposés en les utilisant pour la définition du noyau lui-même [Briat 76 a], [Briat 76 b], [Estublier 78], [Maillot 78]. Ce projet a été supporté financièrement par l'Institut de Recherche d'Informatique et d'Automatique (IRIA), le Centre de Calcul de Grenoble, et le Centre National d'Études des Télécommunications (CNET).

3.1.2.2. Modèle de construction proposé

Le principe de construction proposé est l'abstraction-raffinement [Dijkstra 71]. L'*abstraction* consiste à définir une machine abstraite propre à résoudre le problème spécifié. Le *raffinement* consiste à définir sa réalisation en termes de programmes et d'objets d'autres machines abstraites. L'itération de ce procédé donne naissance à la notion de niveau : une instruction ou un objet d'une machine de niveau *i* est décrit par un ou plusieurs programmes de machines de niveaux inférieurs.

Pour mettre en œuvre ce principe, Ours offre à ses utilisateurs les concepts de machine abstraite et d'unité de traitement ainsi qu'un mécanisme permettant de construire de nouvelles machines abstraites, que nous présentons avec plus de détails maintenant.

Machine abstraite

Une machine abstraite est une entité capable d'interpréter un programme. Les machines sont identifiées par un nom unique et sont activées pour interpréter un programme.

Une machine est réalisée par l'interconnexion de processus serveurs appelés unités de traitement, s'exécutant sur d'autres machines abstraites.

Unité de traitement

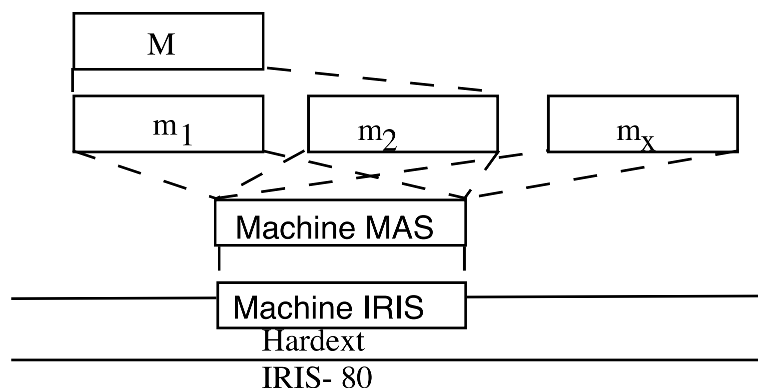
Une unité de traitement est un processus serveur (i.e. processus d'une machine abstraite exécutant un programme cyclique) qui est défini par la liste des portes d'entrée sur lesquelles il peut recevoir des messages et par la liste des portes de sortie sur lesquelles il peut émettre des messages et par le programme qu'il exécute. Un serveur acquiert les messages au moyen de la primitive *Attendre* et émet des messages au moyen de la primitive *Envoyer*. La primitive *Attendre* permet d'attendre l'arrivée d'un message en fixant une priorité aux portes que l'on désire servir ; cependant elle n'est bloquante que si toutes les portes d'entrée sont vides. *Envoyer* permet d'émettre un message sur l'une quelconque des portes de sorties. Le fonctionnement exact de la primitive *Envoyer* est expliqué à la section suivante. Une unité est définie à sa création par la donnée de son contexte initial (machine sur laquelle s'exécute le processus serveur et programme exécuté sur cette machine) et des portes d'entrée et de sortie qu'elle utilise. Une unité ne peut être active que comme composante d'une machine active.

Construction de machines : interconnexion d'unités

La création d'une machine revient dans Ours à interconnecter les portes de sortie et d'entrée des unités constituant la machine. Le résultat de l'opération d'interconnexion est une table de transition d'états qui à chaque état de la machine associe le couple (*unité d'exécution, porte d'entrée*) et une liste des transitions d'état associées aux portes de sortie de l'unité active dans cet état. Le couple (*unité d'exécution, porte d'entrée*) définit, pour l'état auquel il est associé, la porte de l'unité active qui a reçu le message en cours de traitement. La table décrivant l'interconnexion des unités de traitement d'une machine définit le contexte d'exécution, pour les unités de cette machine, de la primitive *Envoyer*. Ainsi *Envoyer* réalise d'abord la transition d'état puis dépose le message sur la porte sélectionnée de la nouvelle unité active. Le premier état de la table représente la transition initiale de la machine et donc la porte d'entrée de l'unité appelée à recevoir le message contenant le contexte initial passé en paramètre de l'instruction d'exécution d'un programme. Certains états sont distingués et correspondent soit aux transitions de sortie de la machine, soit à des transitions d'entrée de la machine autres que la transition initiale. Pour des raisons de simplicité, on interdit la définition récursive de machines (une machine ne peut intervenir ni dans sa propre définition ni dans celle des unités qui la composent). Une machine, comme une unité, a des portes d'entrées et des portes de sorties : elle peut donc être incluse comme constituant d'une autre machine. Dans ce dernier cas, on parlera de machine incluse et la machine qui la possède comme unité s'appellera machine englobante. Une machine est créée par la donnée de sa table de transition.

L'exécution d'un programme sur une machine correspond à la création d'un processus de cette machine et provoque la création d'un message contenant l'identification du programme à exécuter, et celle de la pile associée au processus. Le sommet de la pile associée à un processus contient l'identification de la machine et l'état courant de la machine pour le processus. A chaque transition, *Envoyer* met à jour l'état en sommet de pile. Lorsqu'un message doit être aiguillé sur une des pattes d'entrée d'une machine incluse, après la mise à jour de l'état, *Envoyer* empile l'identité de la machine incluse et l'état d'entrée. La table de transition de la nouvelle machine devient le nouveau contexte de travail de la primitive *Envoyer*. Sur une transition de sortie d'une machine incluse, on dépile l'état et l'identité de la machine courante ; la table transition de la machine englobante redevient la table de transition utilisée par *Envoyer*.

3.1.2.3. Description du système

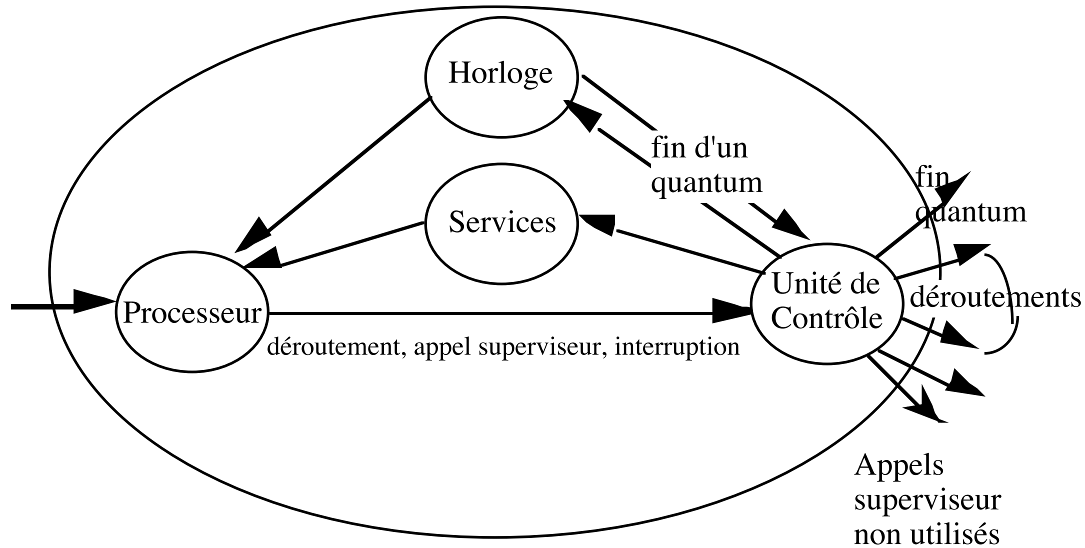


Le système est construit en utilisant les principes proposés (cf. figure ci-dessus) : il est composé d'une couche basse Hardext qui fournit au dessus de l'IRIS 80 physique des machines abstraites primitives : les machines IRIS comportant un processeur IRIS 80 en mode esclave et les mécanismes d'extension que nous avons décrits plus haut. Ces machines

sont utilisées pour construire des machines MAS qui seront utilisées pour construire de nouvelles machines m1, m2, ... etc., par interconnexion d'unités programmées sur MAS. Ces machines deviennent à leur tour disponibles pour construire de nouvelles machines.

La machine IRIS

La machine IRIS peut être représentée par le schéma suivant :



Machine IRIS

• L'unité *Processeur* réalise le multiplexage du bi-processeur IRIS 80. Les messages qu'elle reçoit en entrée sont des contextes de processeur IRIS 80 en mode esclave. Elle alloue les processeurs par tranches de temps élémentaires. Les déroutements, les appels au superviseur, et les interruptions provoquent une transition vers l'unité de contrôle.

• *L'unité de contrôle* analyse les causes d'interruption des processus. Elle joue le rôle d'une opération *Envoyer* primitive pour les processeurs physique IRIS 80. Certains des appels au superviseur correspondent à des appels à des services rendus par la machine IRIS : *Attendre*, *Envoyer*, *Créer_Processus* etc...., et donc à des transitions vers l'unité de service correspondante, et les autres à des conditions de sorties de la machine IRIS. Ces transitions doivent alors être interprétées à l'aide de l'automate de la machine englobant la machine IRIS. De même, l'interruption d'horloge correspond à une transition vers l'unité horloge de la machine IRIS, mais toutes les autres interruptions, ainsi que tous les déroutements, doivent être interprétés à l'aide de la table de transition de la machine englobante. Si on ne trouve pas de transition associée à la cause d'interruption du processus, le processus est détruit.

• *L'unité de service* rend l'ensemble des services offerts par la couche Hardext et qui permettent de créer des unités, des machines et des processus.

• *L'unité horloge* décompte les tranches élémentaires de temps ; si le quantum du processus n'est pas achevé, l'appel à *Envoyer* provoque une transition vers l'unité *Processeur*, sinon il provoque une transition vers l'*Unité de Contrôle*.

La machine MAS

La machine MAS offre en plus des instructions de la machine IRIS, qui est une de ses unités de traitement, des instructions permettant de manipuler des ressources logiques : segments, appareils standards (périphériques), processeurs.

Contrairement à ce qui est fait dans Gemau, les segments ne sont pas adressés au moyen de capacités. L'hypothèse faite est que la protection doit être mise en œuvre par une machine de plus haut niveau. Ainsi Gemau pourrait être un sous-système mis en œuvre sur MAS. Les segments sont désignés par un identificateur unique dans l'espace et dans le temps codé sur 64 bits. Les segments sont rendus adressables par un processus au moyen d'une opération de couplage. De plus, MAS fournit une opération explicite de sauvegarde du segment en espace de stockage qui permet d'avoir des états de reprise en cas de panne. Cette sauvegarde est considérée comme une version en cours du segment tant qu'une opération explicite ne la force pas à devenir la nouvelle version du segment. L'espace de stockage est structuré en volumes logiques.

Pour ce qui est des périphériques d'entrées-sorties, MAS offre un protocole d'accès standard qui est la réplique du protocole d'appareil virtuel développé dans le réseau Cyclades [Pouzin 82], et qui permet d'accéder avec la même interface à tous les périphériques quelle que soit leur nature.

Dans le but de permettre la cohabitation de politiques différentes d'allocation des processeurs, MAS offre des classes de machines ; à chaque classe est attribué a priori un taux différent d'utilisation des processeurs physiques. Ainsi une classe de machine MAS apparaît-elle comme une machine plus ou moins performante. Les processus au sein d'une même classe sont ordonnés selon leur priorité. Afin de garantir l'équité entre les classes, une unité de MAS, l'unité de contrôle, met en œuvre une politique de régulation globale [Leroudier 75] [Denning 76], basée sur le taux d'occupation du canal de pagination, et qui détermine le taux de multiprogrammation de la mémoire. Chaque classe se voit attribuer un pourcentage de ce taux.

La définition de la machine abstraite MAS en termes d'unités de traitement mises en œuvre au dessus de la machine IRIS peut être trouvée dans [Maillot 78].

3.1.2.4. Bilan

Le projet Ours n'a pas été mené jusqu'à son terme, le bi-processeur IRIS 80 ayant été remplacé en 1978 par le système Multics. Seule la couche Hardext, la définition de la machine MAS et la spécification de sa réalisation ont été réellement réalisées. Nous choisissons cependant d'évaluer ce projet selon le plan que nous avons adopté pour le projet Gemau.

Qualité scientifique

Soulignons quelques points qui nous semblent positifs et originaux :

- Le modèle de programmation de sous-systèmes offert par Ours est incontestablement original. Il offre, contrairement à Hydra ou Gemau le module (unité de traitement) comme unité de programmation et d'exécution. C'est de plus un modèle constructif : chaque machine abstraite construite existe comme entité identifiable par le système et non comme une collection d'objets primitifs. Cette caractéristique facilite la mise au point et la collecte de données statistiques relatives à un sous-système particulier.
- Le mécanisme de liaison proposé est également original. Il est statique : les unités constituant une machine sont déterminées une fois pour toute. Mais, comme dans les systèmes mettant en œuvre la liaison dynamique, le même module peut être partagé par des processus de machines différentes.
- Le système Ours serait facile à répartir, même si l'aspect répartition n'était pas la préoccupation de ses concepteurs. Si les unités d'une même machine abstraite sont sur des sites différents, le message et la pile du processus vont aller de site en site

selon un routage spécifié par la table de transition. Il faut donc que la table de transition d'une machine soit dupliquée sur l'ensemble des sites où une de ses unités est active, ce qui n'est pas un problème puisque c'est une donnée en lecture seule construite statiquement.

• Sur l'aspect ingénierie générale, MAS utilise pour l'allocation des ressources de l'IRIS 80 bi-processeur des méthodes conformes à l'état de l'art.

Cependant les quelques interrogations et critiques suivantes peuvent être faites au modèle proposé :

• Le fait de ne fournir qu'un modèle de communication asynchrone entre modules d'une même machine abstraite était-il un bon choix ? Je ne le pense pas. D'abord parce que cela rend la programmation délicate, mais on pourra objecter qu'Ours ne vise que les programmeurs de systèmes qui y sont habitués. Ensuite, parce que le plus souvent l'asynchronisme n'est pas utile[15]. En fait, on aurait dû laisser le programmeur choisir, ce qui n'aurait pas accru la difficulté de mise en œuvre.

• Le fait de disposer des outils permettant la mise en œuvre du principe d'abstraction-raffinement aurait-il changé le comportement habituel des concepteurs qui utilisent cette méthode pendant la phase d'analyse, et l'abandonnent au nom de l'efficacité dans la phase de mise en œuvre ? Cela est moins que probable.

• Le fait de ne pas pouvoir coupler des portes d'entrée et de sortie afin d'indiquer que la réponse à une demande émise sur une porte est attendue sur une autre, peut interdire toute vérification par un outil de la correction des connexions qui sont demandées entre unités.

• On note enfin l'absence d'un langage de connexion qui prenne en charge la vérification des associations proposées (le type d'un message reçu par une porte d'entrée doit être compatible avec le type du message reçu sur la porte de sortie qui lui est associée).

Qualité de l'outil

Le projet n'ayant pas été mené à son terme, il est évidemment difficile de juger de la qualité de l'outil, tant sur le plan qualitatif que quantitatif. Cependant la machine IRIS a fonctionné et montré sur des exemples de test que les mécanismes de base proposés fonctionnaient. La décomposition de la machine MAS en termes d'unités de traitement montre le bien fondé des interrogations que nous venons d'exprimer. MAS est réalisée comme un seul niveau de machine au dessus de la machine IRIS alors qu'il semblerait plus logique d'avoir au moins un second niveau de machine offrant la gestion des entrées-sorties physiques. Mais l'argument d'efficacité est mis en avant par les réalisateurs de la machine MAS pour justifier leur architecture [Maillot 78].

3.2. Architecture et mise en œuvre de protocoles

Le développement de la recherche sur les réseaux locaux à Grenoble qui a suivi la participation d'une équipe de Bull et du CICG au projet Cyclades, m'a donné l'occasion d'expérimenter les connaissances que j'avais acquises au sein du groupe Cornafion [Cornafion 81]. Ce groupe a réuni de 1976 à 1981 des enseignants, des chercheurs et des ingénieurs pour étudier les concepts et techniques nouveaux introduits par le développement des systèmes informatiques répartis. J'ai ainsi participé d'une part aux travaux menés à Grenoble autour du réseau local Danube du projet Kayak [Naffah 79], puis, dans le cadre

[15] Par ailleurs, il a été prouvé depuis (études théoriques sur les modèles de communication synchrone/asynchrone) que la sémantique de l'asynchronisme est très difficile, voir impossible, à formaliser.

d'un contrat entre le Laboratoire de Génie Informatique (LGI) et la Société Apsis, à la définition et à la réalisation du réseau local industriel Factor[16].

3.2.1. Travaux autour du réseau local Danube

Le réseau local Danube était un réseau expérimental dérivé du réseau Ethernet. Il était bâti selon une architecture bus avec comme support un câble coaxial sur lequel étaient connectés des communicateurs programmables. Le réseau pouvait s'étendre jusqu'à un kilomètre avec un débit de un méga-bit par seconde. Le mode d'accès au câble se faisait selon la politique CSMA/CD[17]. Le communicateur se composait d'un coupleur et d'une boîte contenant un microprocesseur et de la mémoire. Ce réseau a été construit pour supporter l'architecture de communication du projet de bureautique Kayak financé et dirigé par l'INRIA. J'ai eu deux types d'activité autour du réseau Danube, l'une purement d'ingénierie consistant à écrire une station de transport sur LSI-11 (Micro-1 de Digital Equipment Corporation) pour l'accès au réseau, et l'autre plus théorique, pour définir un protocole de diffusion fiable sur ce réseau.

3.2.1.1. La station de transport du réseau Danube

Le protocole de la station de transport du réseau Danube était complètement défini [Quint 79] et le travail que nous avions à effectuer, I. Vatton et moi, était son implantation sur le système RSX-11M du LSI-11M ainsi que la définition de l'interface d'accès offerte aux utilisateurs. L'architecture et la réalisation ont fait l'objet de l'une des illustrations jointes en annexe de [Cornafion 81]. Nous avons, I. Vatton et moi, défini en commun l'architecture de réalisation, I. Vatton se chargeant de la mise en œuvre de la partie interface utilisateur de cette architecture et moi de la mise en œuvre du protocole. Cette station de transport a été utilisée avec succès pour supporter des protocoles de plus haut niveau (transfert de fichiers) pendant toute la durée du projet.

3.2.1.2. Un protocole de diffusion fiable pour le réseau Danube

En parallèle avec l'activité de réalisation de la station de transport, j'ai étudié au sein d'un groupe de travail, un protocole de diffusion fiable pour le réseau Danube qui devait être utilisé par les applications qui, comme le transactionnel réparti, ont besoin de ce type de communication. Le protocole qui a été défini était basé sur une numérotation des paquets à l'aide d'estampilles allouées par les sites émetteurs et sur un mécanisme d'acquittement négatif : une station détectant qu'elle n'a pas reçu l'un des paquets émis le redemande en point-à-point à la station émettrice. L'originalité du protocole, outre que c'était l'un des premiers protocoles de diffusion fiable proposés, résidait dans le mécanisme de détection de perte d'un paquet par un site. Ce protocole a été publié [Decitre 82], il a été implanté sur Danube et des mesures de performance sont données dans [Khider 80].

3.2.2. Le réseau local Factor

3.2.2.1. Contexte et motivations

Apsis, PME implantée sur la ZIRST de Meylan, profitant du vide existant à l'époque sur le marché des réseaux locaux industriels, voulait développer un réseau local de communication FACTOR [Factor 85] destiné à l'interconnexion d'automates et de calculateurs. Apsis qui envisageait de développer en parallèle logiciel et matériel, m'offrait de diriger l'équipe de spécification et de développement du logiciel. Un contrat de deux ans était

[16] Factor est une marque déposée d'Apsis/ Aptor

[17] Carrier Sense Multiple Access with Collision Detection

signé entre le LGI[18] et la société APSIS prévoyant mon détachement à mi-temps dans cette société. L'intérêt pour moi était double, d'une part assurer le transfert des connaissances et de l'expérience acquises au cours du projet Kayak et d'autre part acquérir une expérience industrielle.

3.2.2.2. L'architecture de Factor

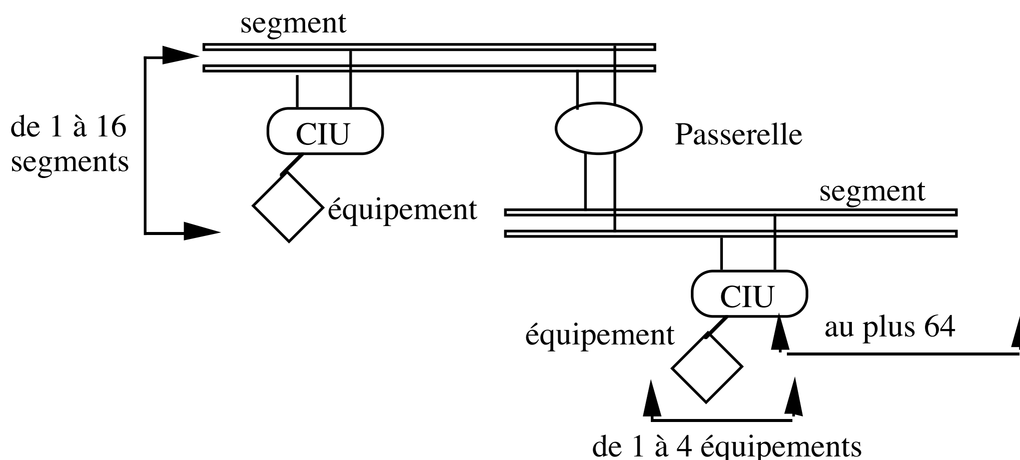
Les contraintes spécifiques

Les contraintes imposées à la réalisation sont de deux ordres :

• Technique : Il s'agit d'assurer la liaison fiable et disponible entre des équipements hétérogènes, programmables ou non, en environnement pouvant être hostile (aspect "industriel"). Le temps entre l'émission de certains messages et leur réception doit pouvoir être garanti (aspect "temps réel"). Les protocoles utilisés doivent tenir compte de la normalisation.

• Économique : Le coût d'une connexion physique d'un équipement au réseau ne doit pas dépasser le coût du marché (à l'époque aux environs de 10 000 F). La durée du développement ne doit pas excéder deux années.

Architecture du matériel



Un réseau Factor est constitué d'au plus 16 segments interconnectés par des passerelles. Chaque segment est constitué d'un ou deux médias sur lesquels peuvent être connectées au plus 64 unités de communication ou CIU (*Communication Interface Unit*) pouvant chacune permettre la connexion de 4 équipements (afin de diminuer le coût de connexion au réseau). Les médias constituant un segment sont des câbles coaxiaux d'au plus 2 Km du type Ethernet gros. Les CIU sont connectées aux câbles au moyen de communicateurs (*transceiver*) Factor piqués sur le câble ou raccordés au moyen d'une embase en T. Une CIU est une machine ouverte (le bus fond de panier peut contenir 1 ou 2 emplacements disponibles pour des extensions mémoires ou pour les besoins des utilisateurs) développée à Apsis, et dans laquelle sont exécutés à la fois les protocoles de communication et les procédures de gestion des quatre lignes de connexion entre les équipements et la CIU. La CIU de base comporte deux cartes ; une carte processeur dessinée autour d'un processeur Intel 380-86 qui comporte :

- Deux contrôleurs DMA[19] pour gérer l'accès des coupleurs réseau à la mémoire,
- Un contrôleur d'attribution de canaux DMA afin d'assurer l'équipriorité d'accès à la mémoire des contrôleurs DMA,

---

[18] Laboratoire de Génie Informatique

[19] Direct Memory Access

- Deux coupleurs réseau, strictement identiques et utilisant chacun un contrôleur de communication multiprotocoles, traitant les niveaux bas du modèle OSI en CSMA/CD,
- Un processeur dédié qui gère d'une part les transferts sur les supports de communication et d'autre part les échanges avec le microcalculateur,
- Un bus périphérique d'entrées-sorties standard permettant de connecter les modules entrées-sorties du communicateur.

La seconde carte est une carte mémoire pouvant comporter jusqu'à 512 K octets de mémoire vive et 256 octets de mémoire morte.

Architecture du logiciel

Pour des raisons économiques (disponibilité de circuits VLSI peu chers et performants et non disponibilité à l'époque de protocole de gestion déterministe d'un bus), le protocole de gestion des médias choisi a été le protocole CSMA/CD. Pour répondre (au moins commercialement) à la grande méfiance des milieux industriels à l'égard du CSMA/ CD qui ne permet pas théoriquement de borner le temps d'accès au médium, nous gérons sur les flots de données des messages prioritaires qui sont acheminés par un des deux médias qui est dédié à ce type de trafic. En cas de panne de l'un des médias (coupure du câble) le trafic, prioritaire ou non, est redirigé sur l'autre média. Dans ce cas, si une collision intervient, le délai de réessai affecté aux trames n'appartenant pas au trafic prioritaire est augmenté de sorte que la compétition pour l'allocation du média n'ait lieu qu'entre les CIU voulant émettre des messages prioritaires.

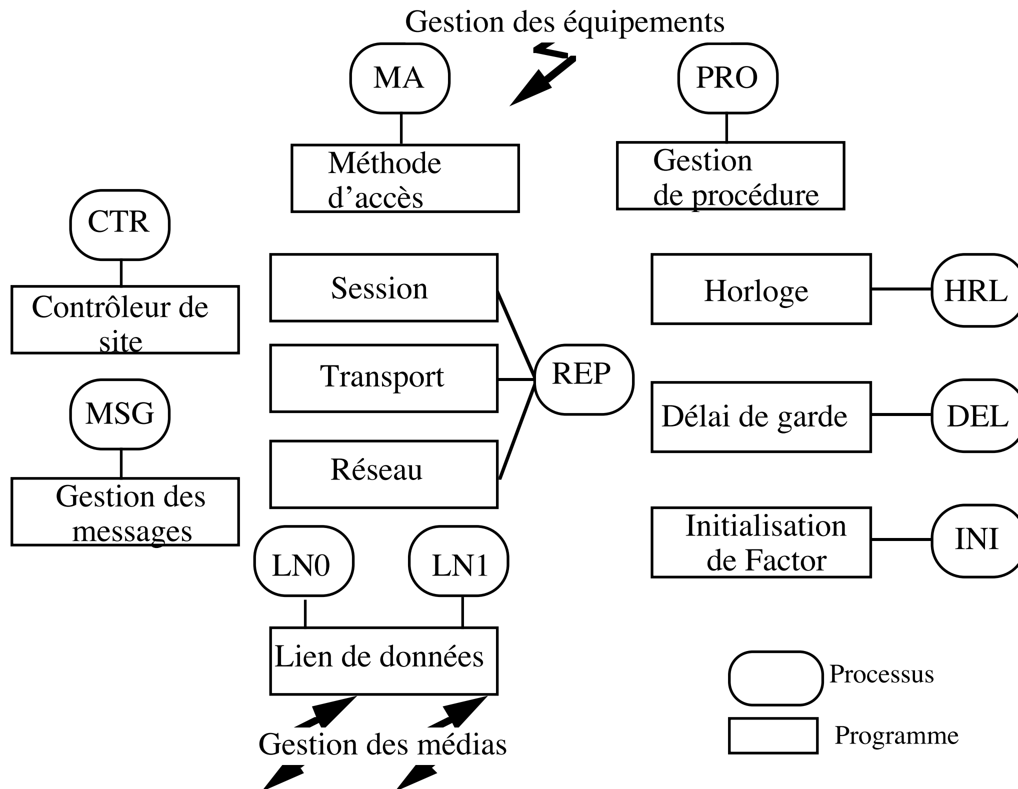
Pour des raisons de compatibilité avec les normes, l'architecture de notre système de communication respecte l'architecture en couches du modèle OSI de l'ISO. Les méthodes d'accès au système de communication Factor appelées Transmate[20] correspondent au niveau présentation (couche 6) du modèle OSI. Chaque couche est mise en œuvre par un module séparé.

L'architecture globale en termes de processus du logiciel Factor (cf. figure ci-dessous) comprend :

- Les processus PRO qui mettent en œuvre les procédures de gestion des lignes connectant les équipements (1 processus par équipement configuré), et les processus MA (1 processus par équipement configuré) qui mettent en œuvre l'un des deux types de méthode d'accès au système de communication : assistée (qui permet la connexion au réseau d'équipement non programmable), ou non assistée (qui permet à un équipement d'attaquer le réseau au niveau session).
- Le processus REP répartiteur qui met en œuvre les protocoles de session, transport et réseau. Dans une première architecture, un processus était associé à chacun des niveaux de protocole. Mais le coût, dû aux changements de contexte pour traiter un message, nous a amené à faire exécuter les modules réalisant les trois niveaux par un seul processus. Cependant, pour permettre un meilleur service des messages urgents, les trois modules réalisant les différentes couches sont désynchronisés et communiquent par message via la boîtes aux lettres du processus REP.
- Les processus LN0 et LN1 qui mettent en œuvre le niveau lien de données pour les deux médias de communication.
- Un processus d'initialisation INI qui crée l'ensemble des processus et des objets utilisés par Factor (boîtes aux lettres, descripteurs de session, files d'attente, etc.). Cette initialisation est faite en fonction de la table décrivant la configuration de la

[20] Transmate est une marque déposée d'Apsis/ Aptor

station qui a été préalablement téléchargée via le réseau. Ce processus disparaît à la fin de l'initialisation.
• Les processus CTR et MSG sont des processus qui gèrent l'administration du réseau en coopération avec le poste de pilotage central si le réseau en possède un.
• Les taches HRL et DEL gèrent respectivement le temps et les délais de garde sur des échelles de temps paramétrables.



Architecture du logiciel Factor

Les processus communiquent via des boîtes aux lettres et se synchronisent au moyen d'événements. Les évènenements sont, en effet, les outils offerts en standard par le noyau AMSTER 86[21] développé à Apsis. Ce type d'outil, que l'on ne trouve, malheureusement, comme outil de synchronisation primitif que dans les exécutifs temps réel, est très bien adapté dès lors qu'il s'agit pour un processus de se synchroniser sur des événements asynchrones de nature différente.

3.2.2.3. Bilan

A la fin du contrat en Octobre 1983, nous disposions de deux réseaux Factor expérimentaux en fonctionnement l'un dans une entreprise de sidérurgie pour le pilotage d'un laminoir et l'autre dans la Centrale Solaire THEMIS pour le pilotage des miroirs. Les cartes des communicateurs n'utilisaient pas encore la technologie multicouche mais des composants reliés par des fils (*wrapped technology*). Actuellement Apsis a commercialisé via la société Aptor quelques quatre cents réseaux Factor qui fonctionnent dans une grande variété d'environnements.

---

[21] AMSTER 86 est un exécutif temps réel offrant une interface compatible avec celle de l'exécutif RMX86-Intel ; c'est une marque déposée d'Apsis/ Aptor

Les différents modules du système de communication ont été développés en PLM que nous avions, grâce à son macrogénérateur, habillé en Pascal. L'ensemble des développements du système de communication lui même représente environ 40 000 lignes de code commentées.

La modularité de l'architecture et de la réalisation proposée a permis de remplacer sans mal le protocole de gestion du câble par un protocole CSMA/ DCR[22] développé à l'INRIA [Boudenant 87] qui permet sur le plan technique de borner les temps d'accès aux médias et sur le plan du marché de mieux affronter la concurrence de la technologie des bus à jetons (i.e. le réseau MAP[23])

Les deux méthodes d'accès au réseau de communication, Transmate, sont originales et se sont révélées satisfaisantes pour les nombreuses applications développées (plus de 300).

La méthodologie de développement obligeant à chaque étape de la vie d'un module à produire un document le décrivant, ainsi que la pratique de la relecture croisée des modules de code source s'est révélée très efficace dans la phase de mise au point du logiciel.

Les difficultés rencontrées dans la définition et la réalisation d'un système de communication, s'il repose sur des protocoles corrects, sont celles que l'on rencontre quand on définit et qu'on réalise un système de gestion d'entrées-sorties pour un système centralisé. Il s'agit de minimiser le nombre de changements de contextes entre tâches et de minimiser le nombre de recopies des messages d'un tampon dans un autre. La solution adoptée [Factor 85], qui permet de déporter un certain nombre des fonctions sur un processeur dédié et de n'avoir qu'une recopie par message s'est révélée tout fait efficace.

3.3. Étude des systèmes répartis : Le projet Guide

La possibilité de disposer de stations de travail dans le milieu des années 80, ainsi que le grand nombre de projets menés aux États Unis ou en Europe dans le domaine des systèmes répartis, devaient entraîner une petite équipe regroupant des personnes du Centre de Recherche Bull et du Laboratoire de Génie Informatique à réactiver à l'IMAG la recherche sur les systèmes opératoires qui était en sommeil depuis la fin des années 70. La conséquence de cette démarche a été la définition du projet Guide, commencé en 1986, dont la première phase a abouti à la réalisation d'une maquette de système réparti Guide-1 et dont la seconde phase est en cours de réalisation. C'est ce projet que nous présentons dans cette section.

3.3.1. Présentation du projet

3.3.1.1. Objectifs généraux

Le thème général du projet Guide est le développement d'un système d'exploitation pour des postes de travail individuels et des serveurs interconnectés par un réseau local, et utilisés pour des applications générales. Il ne s'agit pas, dans un premier temps, de construire un système pleinement opérationnel destiné à remplacer les systèmes existants, mais de réaliser un système expérimental qui doit servir de banc d'essai pour des recherches sur les systèmes répartis : validation de nouvelles méthodes de conception, structure des systèmes d'exploitation répartis, outils pour le développement d'applications réparties. Les classes d'application prévues sont le génie logiciel (production, mise au point et maintenance de

[22] Carrier Sense Multiple Access with Deterministic Collision Resolution

[23] Manufacturing Automation Protocol, MAP est un réseau développé par Général Motors et des entreprises cherchant à automatiser le fonctionnement des usines.

programmes), la communication (courrier électronique, agenda) et la gestion de documents (édition, archivage, etc...).

Comme nous l'avons signalé en conclusion du chapitre 2, les propriétés demandées à un tel système sont voisines de celles qui ont servi de base au système Multics d'où sont issus les systèmes en temps partagé. Seules les modalités de prestation de services et de partage ainsi que les interfaces présentées aux usagers ont considérablement changé. On doit également ajouter que, depuis Multics, des concepts nouveaux pour la structuration des logiciels (modules, types abstraits, objets, généricité) ont été élaborés. Ils permettent notamment de dissocier la description des interfaces de celle des réalisations et de définir des classes d'objets ayant un même comportement ; ils facilitent ainsi la réutilisation du logiciel et permettent de dissimuler aux usagers l'hétérogénéité du matériel ou des systèmes.

L'un des objectifs du projet est d'atteindre un degré d'intégration total des composants du système réparti en "virtualisant" les ressources. Autrement dit, on souhaite décharger l'utilisateur de la nécessité de connaître la localisation des ressources qu'il demande ou le détail des protocoles de communication nécessaires pour y accéder. De façon concrète, on peut imaginer qu'un usager du système ait la possibilité d'accéder à tout ou partie de l'ensemble des ressources d'un réseau, et de construire une application composée utilisant des ressources locales et des ressources distantes. Inversement un usager a la possibilité de déclarer comme accessibles des ressources qu'il possède.

3.3.1.2. Orientations de conception

Les objectifs du projet, qui viennent d'être rappelés, tendent à privilégier une démarche de conception descendante et une vue intégrée des divers composants du système. Nous visons donc :

- à transposer, dans l'organisation d'un système d'exploitation réparti, les concepts introduits dans les langages de programmation pour faciliter l'abstraction et la réutilisation de composants,
- à intégrer en un ensemble cohérent les aspects relatifs à l'exécution et ceux touchant à la conservation permanente de l'information.

Ces considérations nous ont amené à nous intéresser aux modèles issus des travaux sur la programmation par objets, en y intégrant les aspects liés à la conservation permanente des objets qui sont traditionnellement traités séparément sous la rubrique des systèmes de gestion de fichiers ou des bases de données. Par ailleurs, nous avons spécifié un modèle abstrait d'exécution et de conservation aussi indépendant que possible de la réalisation physique et notamment de la répartition, afin de pouvoir faciliter l'expérimentation et la comparaison de divers mécanismes de mise en œuvre du modèle sur un système réparti, et la prise en compte de l'hétérogénéité des composants physiques du système.

3.3.1.3. Les résultats de la première phase : Guide-1

La première phase du projet a été menée de 1987 à 1990. Le choix de base a été celui d'une intégration forte entre langage de programmation et système d'exploitation, le système servant essentiellement de support d'exécution pour un langage, lui même issu d'un modèle de structuration d'applications à base d'objets.

Les résultats suivants ont été obtenus :

- Élaboration d'un modèle d'objets, avec définition séparée de types et de classes, objets persistants passifs, désignation au moyen de références, héritage simple restreint par une relation de conformité.
- Définition d'un langage mettant en œuvre le modèle d'objets.

- Spécification d'un modèle d'exécution : structuration et gestion réparties des activités, communication entre les activités par partage d'objets, synchronisation, exceptions.
- Réalisation d'un prototype du compilateur et de son support d'exécution au-dessus d'Unix.
- Démonstration du prototype sur des applications expérimentales développées soit par notre équipe, soit par des partenaires extérieurs au projet (entre 90 000 et 100 000 lignes de langage Guide écrites pour la plupart par des équipes extérieures à la notre).

Plutôt que de décrire le modèle d'objets et le langage ainsi que le modèle d'exécution et sa mise en œuvre au-dessus d'Unix qui ont fait l'objet de publications dans des revues [Krakowiak 90] [Balter 91], et dont un tiré à part est joint en annexe de ce document, nous nous intéressons maintenant à tirer un bilan critique de cette première phase.

3.3.2. Évaluation critique de la première phase

3.3.2.1. Couverture du domaine

Un certain nombre d'aspects n'ont pas été couverts dans Guide-1. Ont délibérément été laissés de coté les aspects touchant à la tolérance aux fautes et à la sécurité. L'administration du système a été réduite au strict minimum nécessaire à l'expérimentation. Enfin, aucun support n'a été prévu pour d'autres langages que le langage Guide. En conséquence, Guide-1 est une machine abstraite que l'on peut mettre en œuvre sur un ensemble de machines interconnectées, plutôt qu'un système. En outre, l'expérience effective sur Guide-1, tant dans notre laboratoire qu'à l'extérieur, est limitée à un très petit nombre de machines[24] (3 ou 4).

3.3.2.2. Modèle d'objets et Langage

Le modèle d'objet et le langage qui le met en œuvre ont dans l'ensemble été jugés satisfaisants, en tant qu'outils pour la conception et le développement d'applications réparties. La communication par objets partagés et la construction d'objets complexes fournissent un mode d'expression bien adapté aux applications coopératives. Le système de types apporte une sécurité appréciable. Cependant, trois aspects principaux peuvent être critiqués. D'une part le modèle d'objets et le langage souffrent d'un manque de définition rigoureuse. En particulier, les aspects liés à la spécification et à l'organisation du système de types (uniformités des types, conformité, héritage, etc.) doivent être approfondis. D'autre part le traitement de la persistance est primitif (tous les objets sont automatiquement persistants). Cet aspect doit être repris. Enfin le mécanisme de gestion de la synchronisation, qui est un des aspects les plus originaux du langage [Decouchant 91], s'est révélé très commode sous sa forme actuelle, mais il nécessite des travaux supplémentaires : relation entre synchronisation et types, relation entre synchronisation et exceptions, relation entre synchronisation et transactions, qui sont déjà en cours [Riveill 92] [Lacourte 92]. On trouvera une étude critique complète du modèle d'objets et du langage dans [Riveill 93].

3.3.2.3. Schéma d'exécution

Le modèle d'exécution de Guide spécifie un schéma à deux niveaux (domaines et activités). La justification d'un tel modèle était de permettre l'expression de deux granularités de coopération entre activités selon qu'elles appartiennent au même domaine ou à des domaines différents. Tout objet couplé dans un domaine devient immédiatement adressable

[24] Guide-1 a fonctionné, ou fonctionne sur des SUN 360 et 380, des SPARC, des Zenith (PC 386 et 486) et des Bull DPX-1000.

par toutes les activités du même domaine. Un objet partagé par deux domaines différents, pour être adressable, devra être couplé par l'une quelconque des activités de chacun de ces domaines.

En fait, la réalisation de Guide-1 sur le système Unix n'établit que peu de différence entre les domaines et les activités, en raison de la représentation des activités par des processus Unix et de la réalisation de la mémoire partagée d'objets (i.e. tous les représentants d'activité présents sur un sites partagent la même mémoire virtuelle d'objets qu'ils appartiennent ou non à des domaines différents [Decouchant 88]). La principale différence entre les domaines et les activités reste le mode d'activation, synchrone (i.e. l'activité qui crée d'autres activités est bloquée jusqu'à la terminaison de ses filles) pour les activités et asynchrone pour les domaines. La validité du modèle d'exécution n'a donc pas pu être complètement évaluée.

La gestion du parallélisme (création synchrone d'activités dans un domaine avec conditions de terminaison) n'est pas entièrement satisfaisante car le modèle de terminaison proposé (destruction des activités non terminées lors du COEND) présente des risques d'incohérence ; ce problème devrait être pris en compte au moyen du mécanisme d'exceptions [Lacourte 91]. L'absence de mode asynchrone de création d'activité dans un domaine ne semble pas avoir posé de problème aux écrivains d'applications. L'utilisation d'activités parallèles au sein d'un même domaine, que ce soit dans les applications agenda et Griffon développées au sein de notre unité ou dans l'application CIDRE développée au SEPT, sert au traitement des événements asynchrones (affichage, mise à jour des fenêtres, etc.) pris en compte par ces applications.

La réalisation répartie des domaines et des activités a conduit à développer deux mécanismes : la diffusion et l'appel de méthode à distance. Les modèles de base sont bien compris, mais plusieurs aspects restent à examiner. En premier lieu, les mécanismes de tolérance aux défaillances, s'ils ont fait l'objet d'une étude dans le cadre d'un DEA [Laribi 91], n'ont pas été implantés dans la maquette de Guide-1. D'autre part, les problèmes de répartition de charge, de migration des objets en cours d'exécution et de localisation explicite d'objets n'ont pas été traités en dehors d'un projet de DEA [Christaler 91]. Le système de gestion des transactions n'a pas été mené à son terme (i.e. les transactions ne fonctionnent pas en réparti). Enfin les mécanismes de synchronisation devraient être rendus plus efficaces (pour un appel de méthode synchronisé, 25% du temps est passé dans l'exécution des mécanismes de synchronisation [Freyssinet 91b]).

3.3.2.4. Mémoire d'objets

La mémoire d'objets de Guide a été organisée en deux niveaux distincts : la partie du système chargée de la conservation permanente des objets, appelée Mémoire Permanente d'Objets (MPO), le support des objets qui sont couplés dans au moins un domaine appelé Mémoire Virtuelle d'Objets (MVO). Ce choix avait pour but d'expérimenter divers modes de réalisation tant de la MVO que de la MPO. Il aurait également dû permettre, si nous n'avions pas fait le choix malencontreux d'utiliser la MPO comme mémoire d'échange, d'avoir une sauvegarde en cas de panne d'un ou plusieurs sites, ou de mort soit d'un domaine, soit d'une activité non terminée. En effet, en cas de saturation de la MVO, un objet en cours d'utilisation (et donc dans un état incertain) peut être temporairement vidé de la MVO et recopié sur son image en MPO. L'ensemble MVO-MPO fonctionne donc comme une mémoire virtuelle globale et dans les situations que nous avons évoquées, le système ne peut pas garantir la cohérence de l'image en MPO des objets qui étaient en cours d'utilisation par le site, le domaine ou l'activité. Cette confusion entre les fonctions d'espace d'exécution et

d'espace de stockage à long terme, du fait de la fragilité du système qu'elle implique, est certainement le plus grand reproche que nous puissions faire à Guide-1.

Les mécanismes de gestion sont mal adaptés à la gestion des petits objets (près de 90% des objets ont une taille inférieure à 512 octets [Freyssinet 91b]). L'unité de couplage dans la MVO ne devrait donc pas être l'objet mais un regroupement d'objets, ceci afin d'améliorer et la gestion de l'allocation de mémoire en MVO et les transferts entre la MVO et la MPO.

3.3.2.5. Environnement de développement

Seuls le compilateur du langage Guide [Nguyen Van 91], l'observateur d'applications [Roisin 91] et le metteur au point [Jamrozik 91] (en cours de développement) ont été développés. Il manque en particulier des outils pour la gestion de l'évolution des types et des classes (gestion de versions, etc.). Le gestionnaire de types a été réduit à sa plus simple expression et est intégré au compilateur. Enfin le système manque de possibilité de liaisons statiques partielles, qui permettraient une exécution plus efficace notamment dans le cas d'applications répétitives (cet aspect est également lié au modèle d'objet et au langage)

3.3.2.6. Services

Deux services ont été réalisés sous Guide-1. Le service de désignation a été réalisé dans une version minimale, suffisante cependant pour permettre d'exécuter des applications. Une première version d'un langage de commande (shell) graphique a été réalisée mais n'a pas été intégrée aux dernières versions de Guide-1. Deux voies ont été suivies pour la construction d'interfaces interactives utilisant OSF/ Motif : d'une part un générateur automatique d'interfaces en Guide pour les primitives de Motif (c'est la méthode utilisée dans l'application CIDRE, dont on trouvera une description dans [Cahill 92]), d'autre part une boîte à outils fournissant un ensemble de fonctions usuelles (c'est la méthode utilisée par Griffon). La première méthode, la plus complète, s'est révélée lourde à mettre en œuvre au niveau du système, mais s'est révélée très commode à utiliser aux dires de l'équipe qui a construit l'application CIDRE. La seconde méthode est limitée aux fonctions fournies par la boîte à outils (actuellement essentiellement des fonctions de dialogue).

3.3.2.7. Coopération avec Unix

Deux voies ont été suivies en Guide-1 pour réutiliser des applications existantes sur Unix. La première consiste à "lancer" l'application depuis un objet Guide, en fournissant une interface pour les points d'accès à cette application. Cette méthode permet de récupérer rapidement une application en bloc mais ne s'applique pas à la récupération de fonctions ou de bibliothèques. La seconde consiste à inclure, dans le code C produit par le compilateur du langage Guide, des séquences d'appel à un programme existant qui sera exécuté dans la mémoire du domaine appelant. Dans les deux cas, il faut regretter une absence totale de sécurité à l'exécution puisque les programmes ainsi récupérés s'exécutent sans contrôle, et pour la seconde méthode dans le même espace virtuel que le programme Guide appelant. Par ailleurs, les versions du prototype, jusqu'à la version 1.6 non comprise, ne disposaient pas de chargeur dynamique pour le code Unix ; les programmes récupérés, au moyen de la seconde méthode, devaient donc être liés statiquement avec le noyau. Cette solution était peu souple et pénalisait les performances du noyau en raison de la taille élevée des images mémoires des activités.

3.3.2.8. Réalisation

Le prototype Guide-1 a été réalisé sur Unix. Les principaux avantages sont la rapidité de réalisation, l'utilisation de l'environnement de développement, et la portabilité (le transport

de Guide sur une nouvelle machine Unix nécessite environ 1 homme x mois). Les inconvénients sont l'inadéquation des primitives de base d'Unix pour la mise en œuvre des mécanismes de Guide :

- Les processus sont trop coûteux pour la réalisation des activités.
- La mémoire partagée se prête mal à la gestion des objets, les opérations de liaison/ déliaison des segments de mémoire partagée sont coûteuses. On doit cependant noter que nous n'avons pas utilisé rationnellement les segments de mémoire partagée Unix. En effet, la solution que nous avons choisie [Decouchant 89 a] consiste à implanter la MVO d'un site comme un seul segment de mémoire partagée, couplé une fois pour toute dans la mémoire virtuelle des représentants d'activités sur ce site à leur création. Cette solution nous interdit donc de tirer parti des protections offertes par Unix sur les segments de mémoire partagée, et nous oblige à gérer l'allocation de mémoire aux objets dans le segment de mémoire partagée. Une association regroupement d'objets (*cluster*) avec un segment de mémoire partagé aurait permis d'une part d'optimiser les transferts MPO-MVO et d'autre part d'allouer un regroupement d'objets dans un segment de mémoire partagée.
- Les sémaphores Unix sont également des outils très coûteux, ce qui nous a conduit à développer nos propres outils de synchronisation [Decouchant 89].

Le prototype Guide-1 reste, malgré ces critiques, un bon outil pour l'expérimentation d'applications et d'outils de développement. Ses temps de réponses (de l'ordre de 31 ms pour un appel distant de méthode avec paramètres, et de 8 ms pour créer un objet à distance) ne sont pas de mauvais résultats.

3.3.3. Choix de conception de la seconde maquette

L'objectif de Guide-1 était, comme nous l'avons rappelé, de montrer la viabilité des principes de conception de l'architecture et leur adéquation aux domaines d'application visés. De ce point de vue, cette première phase du projet a été un succès. Cependant, le désir d'avoir une version améliorée du système afin de mener des expérimentations systématiques, et de valider des hypothèses sur son architecture, nous ont conduit à concevoir la plate-forme Guide-2.

3.3.3.1. Orientations Générales

Les principales orientations de Guide-2 diffèrent de celles de Guide-1 sur les points suivants :

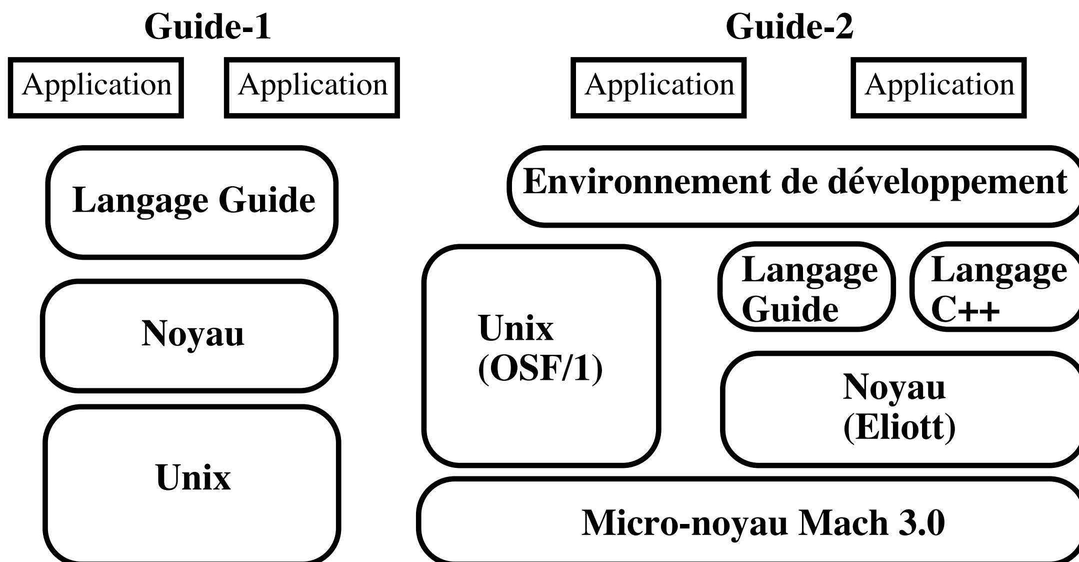
- Guide-2 doit permettre d'exécuter des applications écrites dans plusieurs langages à objets et de faire communiquer de telles applications entre elles. Dans un premier temps les langages visés sont le langage Guide et une version étendue du langage C++.
- Plusieurs aspects non traités dans Guide-1 seront pris en compte dans Guide-2 : sécurité et protection, administration du système, tolérance aux fautes.
- Guide-2 doit avoir une architecture modulaire permettant la réutilisation de composants existants (en particulier des serveurs de stockage et des applications s'exécutant sur Unix), et l'expérimentation de politiques différentes pour la gestion du partage de la mémoire.

Une première expérience, consistant à construire une maquette sommaire de Guide-1 sur micro-noyau [Boyer 91a] a montré que les micro-noyaux permettaient de satisfaire les contraintes de modularité et donnaient un contrôle plus fin des mécanismes du système

notamment pour ce qui concerne la gestion de la mémoire virtuelle. La maquette Guide-2 est donc construite sur Mach 3.0.

Les stations utilisant Guide-2 sont d'abord des machines à base de processeurs Intel 80386/ 80486 (stations Bull-Zenith).

Le schéma ci-après montre l'architecture d'ensemble des systèmes Guide-1 et Guide-2. Il fait apparaître un noyau, appelé Eliott, permettant de supporter plusieurs langages à objets, les environnements de développement de ces langages et un ensemble d'outils et de services constituant un environnement de développement d'applications.



Nous nous intéressons maintenant au noyau Eliott qui fournit les mécanismes et les fonctions nécessaires au support générique de différents langages à objets. La définition d'Eliott tient compte de l'expérience que nous avons acquise avec Guide-1 et des orientations générales que nous venons de donner.

3.3.3.2. Les choix de bases pour la conception d'Eliott

Mécanisme pour le support des langages

Nous présentons dans cette section les mécanismes nécessaires pour prendre en compte au niveau d'Eliott les caractéristiques des langages utilisés. Nous nous limitons donc aux caractéristiques des langages qui sont visibles du système.

- Eliott gère des entités appelées objets, qui servent à représenter des données complexes. Tout objet est un exemplaire (instance) d'une classe. Une classe définit la structure interne de ses instances et le programme des opérations qui leur sont applicables. Les classes sont elles-mêmes représentées par des objets particuliers appelés objets-classes. Le système maintient un lien explicite entre chaque objet et sa classe.
- Les classes sont organisées selon une relation "est sous-classe de". Le détail de cette organisation et son implication sur les opérations applicables aux objets dépendent du langage utilisé ; cependant Eliott fournit des mécanismes, les segments de code, qui permettent de mettre en œuvre plusieurs modèles, en particulier pour l'héritage multiple.
- Pour le support des programmes (*main* C++), Eliott fournit des objets particuliers appelés objets-programmes, qui exportent au moins le point d'entrée *main* et qui sont gérés différemment des autres objets du système comme nous le verrons par la suite.

• Tout objet est désigné par une référence, c'est à dire une information qui permet de l'identifier et de le retrouver. Le système fournit plusieurs sortes de références qui diffèrent par leur portée et leur durée de vie afin de gérer des objets connus dans des environnements différents et ayant différentes durées de vie.

Modèle d'exécution

Le modèle d'exécution de Guide définit deux entités : les domaines et les activités. Un *domaine* est un espace d'adressage multi-site, dans lequel des objets peuvent être rendus accessibles. L'ensemble des sites sur lesquels est représenté un domaine et l'ensemble des objets qu'il contient peuvent évoluer dynamiquement, dans des conditions que nous précisons plus loin. Dans un domaine s'exécutent des *activités*, flots séquentiels d'exécution. Concrètement, un programme (au sens Unix) est représenté par un domaine. Une application peut éventuellement être représentée par plusieurs domaines.

L'espace d'adressage d'un domaine est constitué d'un ensemble de *contextes*. Un contexte est un espace d'adressage homogène (mémoire virtuelle) local à un site. Cette décomposition est motivée par la répartition (comme dans Guide-1, un domaine a un contexte par site visité) et par des considérations de protection (contrairement à Guide-1, un domaine peut avoir plus d'un contexte par site).

Pour créer un domaine, il faut spécifier un objet-programme. Après création, le domaine est constitué d'un contexte unique contenant l'objet-programme et d'une activité, dite principale, qui exécute la méthode *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-instances. Un appel de méthode peut provoquer un changement de contexte (ou la création d'un nouveau contexte) s'il y a changement de site ou si la protection le requiert.

La communication entre activités, à l'intérieur d'un même domaine ou entre deux domaines différents, se fait par partage d'objets. Le système fournit des mécanismes permettant diverses politiques pour la mise en œuvre du partage (exemplaire unique pour les objets-instances, copies multiples pour les objets-classes, synchronisation des accès).

Eliott, fournit des mécanismes pour contrôler la localisation des objets. Par défaut, la localisation n'est pas visible des utilisateurs.

Espace d'exécution et espace de stockage

Dans le modèle d'exécution que nous venons de définir, les objets utilisés par les activités sont considérés comme faisant partie d'un espace unique que nous appelons l'*espace d'exécution*. Cet espace d'exécution est vu globalement comme une mémoire commune. Le support de cette mémoire est l'ensemble des mémoires de pagination des stations de travail (mémoire centrale et espace disque réservé à la pagination). Cet espace est essentiellement un espace de travail. Il ne fournit aucune garantie de conservation de longue durée ; en particulier en cas d'arrêt d'une station, volontaire ou résultant d'une panne, les informations appartenant à la portion d'espace concernée sont perdues. Cet espace doit donc être complété par un espace de conservation plus fiable que nous appelons *espace de stockage*. L'interface entre l'espace de stockage et l'espace d'exécution est du type chargement implicite à la demande, recopie explicite.

Une telle solution garde les avantages d'uniformité d'un espace d'adressage à un niveau à la Multics (on accède aux données de la même manière qu'elles soient en mémoire d'exécution ou de stockage), tout en permettant de garder les avantages de modularité (possibilité de réutiliser des serveurs existants pour le stockage), de sécurité (possibilité de

contrôler l'accès à l'espace de stockage) et de fiabilité (les données ne sont recopiées que sur demande explicite, ce qui garantit, en cas d'arrêt, que l'image sur disque est cohérente).

L'espace de stockage doit comprendre trois services : un service de stockage rapide (la MPO de Guide-1), l'Andrew File System qui est mis en œuvre sur des serveurs ne supportant pas Guide[25], et un service de stockage assurant la disponibilité des objets par duplication d'objets [Chevalier 92].

Partage

Le partage des objets entre les applications ou les programmes (donc entre les domaines) peut être séquentiel ou simultané. Dans le partage séquentiel, les périodes d'utilisation de l'objet par les applications sont disjointes. Dans le partage simultané, les périodes d'utilisation se recouvrent, l'objet pouvant être accessible dans deux (ou plusieurs) domaines simultanément. Le partage séquentiel est résolu par l'existence d'objets persistants. Nous examinons les contraintes induites par le partage simultané.

Pour être accessible dans un domaine, un objet doit être couplé dans l'un des contextes du domaine. L'expérience de Guide-1 montre que l'objet ne peut pas être l'unité de partage si on veut optimiser les transferts entre l'espace d'exécution et l'espace de stockage. A cette raison, il convient d'ajouter le fait que sur Mach une opération de couplage est coûteuse en temps d'exécution et en ressources du système consommées. Le partage des objets peut donc être envisagé de deux manières : l'unité de partage est le contexte entier (la mémoire virtuelle) ou l'unité de partage est un regroupement d'objets appelé *grappe*. Dans le premier cas l'objet n'est couplé que dans un contexte à la fois, puisque tout domaine partageant l'objet partage le contexte. Dans le second cas une grappe peut être couplée simultanément dans plusieurs contextes.

Le partage de contexte a deux avantages : permettre de partager des objets dont la représentation contient des adresses, ce qui est le cas des objets produits par un compilateur C++ pour Unix ; permettre une exécution efficace à l'intérieur d'un contexte, les objets étant directement désignés par des pointeurs. Son inconvénient est de ne pas permettre à des domaines qui partagent un objet d'avoir des droits différents sur cet objet et sur ceux qui sont couplés dans le même contexte. En outre, tout domaine qui accède à un objet a également accès à l'ensemble des objets du même contexte. La protection nécessite alors de séparer les objets en familles de même protection. Si on désire une protection sélective, on est amené à multiplier le nombre de contextes, et donc d'appels inter-contexte, qui sont plus coûteux que les appels internes à un contexte.

Pour Eliott nous avons choisi la solution du partage de grappes. Ce choix est justifié par le souci d'assurer la protection sélective des objets. Il en résulte les contraintes suivantes : un objet partagé (soit *O*) ne doit pas contenir d'informations spécifiques à un contexte. De telle informations sont essentiellement des adresses virtuelles qui peuvent désigner soit des informations internes à l'objet *O*, soit des objets distincts de *O* (objets externes).

Pour les adresses virtuelles référençant des informations internes, le problème se pose uniquement pour les objets-classes et les objets-programmes. On le résout en imposant pour les objets-classes une représentation indépendante de leur adresse de couplage. En pratique le code produit par les compilateurs doit donc être au format PIC (Position Independant Code) et ne doit contenir aucune adresse absolue de donnée. Pour les objets-programmes, dans lesquels nous voulons permettre l'utilisation de pointeurs, Eliott assure qu'ils sont

---

[25] Cela pour deux raisons : d'une part le portage d'un serveur de fichier sur Guide serait une charge incompatible avec les forces dont nous disposons et d'autre part nous augmentons ainsi la sécurité globale du système.

couplés à la même adresse dans tous les contextes qui les partagent (contextes créés au démarrage des domaines).

Pour les informations externes, le problème peut être résolu de deux manières, soit garantir que toute adresse est valide dans tout contexte, soit éliminer les adresses de la représentation des objets partagés. La première solution implique que chaque objet soit implanté à la même adresse dans tous les contextes, dans la seconde un objet peut être couplé à des adresses différentes.

La première solution demande que l'on soit capable d'avoir une gestion commune des adresses de couplage dans tous les contextes d'un même site (cas du partage d'une copie unique) ou dans tous les contextes existant sur tous les sites (cas du partage par copies multiples). Si l'adresse de chargement est calculée dynamiquement lors du couplage, cette solution, dans le cas où le partage est fait par copie multiple, n'est pas réaliste car elle est trop chère à mettre en œuvre. Dans le cas du partage d'une copie unique, elle reste coûteuse, mais peut être envisagée[26]. En revanche si l'adresse est calculée statiquement et qu'elle est allouée une fois pour toute à un objet pendant toute sa vie (mémoire d'objet à un niveau), cette solution devient réaliste et peut présenter des avantages si on opère avec des contextes suffisamment grands, ce qui est le cas si on dispose de mémoires virtuelles à 64 bits d'adresses[27].

Pour Eliott, nous avons choisi la seconde solution, les objets ne contiennent pas d'informations relatives au contexte. Cela impose des conditions sur le code généré par les compilateurs des langages supportés.

Persistance

Un objet est dit persistant si sa durée de vie n'est pas liée à celle de l'unité d'exécution où il a été créé. Un objet persistant ne peut être détruit que par commande explicite, ou par ramassage de miettes s'il devient inaccessible.

Dans Eliott, un objet est créé dans un domaine, et plus précisément dans un contexte d'un domaine. Pour être persistant, un objet doit survivre au domaine dans lequel il a été créé, donc avoir un mode de désignation et un support indépendants d'un domaine particulier. Notons que ces conditions sont également celles nécessaires au partage d'objets entre domaines. Nous confondons donc les notions d'objets partageables et d'objets persistants.

Les objets persistants qui ont une image dans l'espace de stockage (à la suite d'une demande de recopie explicite) sont dit *permanents*. Un objet persistant qui n'est pas permanent est perdu en cas d'arrêt de la machine qui supporte le morceau d'espace d'exécution qui le contient.

Accès aux objets : désignation

Un objet peut être désigné de deux manières différentes :

- Un objet persistant (permanent ou non) est désigné par un identificateur unique et invariant (référence système, en abrégé *sysref*) qui permet de le retrouver. Cette désignation est universelle : sa portée est l'ensemble des machines gérées par Eliott et sa durée de vie n'est pas limitée.
- Un objet qui est couplé dans un contexte (conséquence du couplage dans ce contexte de la grappe à laquelle il appartient) peut être indifféremment désigné par sa *sysref* ou par l'adresse virtuelle de son origine. L'adresse virtuelle est une

[26] C'est ce que nous avons fait dans Guide-1 pour les raisons développées en 3.3.2.3

[27] Nous citons dans le § 2, un certain nombre de projets de recherche qui étudient cette solution

désignation locale dont la portée est le contexte et la durée de vie celle du couplage de la grappe qui le contient.

La référence système est attribuée à la création de l'objet. Elle comporte l'identification unique de la grappe de création (*gid*) et une estampille propre à cette grappe. Une fonction de la machine qui gère les grappes est de retrouver un objet à partir de sa *sysref*. On fait l'hypothèse que les migrations d'objets d'une grappe dans une autre sont peu fréquentes ; la grappe de création est donc souvent la grappe de résidence. En cas de migration on utilise un catalogue des objets déplacés qui doit contenir le *gid* de la nouvelle grappe de résidence.

Accès aux objets : liaison

Un objet ou une classe peut contenir des noms (noms symboliques ou références systèmes) désignant d'autres objets ou classes. Pour réaliser l'accès effectif à une information celle-ci doit être couplée dans un contexte et être désignée par une adresse virtuelle. On appelle *liaison* l'opération qui permet de passer d'un nom à une adresse.

Rappelons que la liaison peut-être réalisée de différentes façons :

- Statiquement. Les noms sont alors transformés en adresses, avant toute exécution.
- Dynamiquement, lors de l'exécution. Deux schémas sont alors possibles : soit les noms sont résolus lors de chaque accès (le système interprète tout nom, à chaque accès, pour déterminer l'adresse correspondante), soit les noms sont résolus au premier accès. Dans ce dernier cas, au premier accès, le système établit une correspondance directe ou chemin d'accès entre le nom et l'adresse de l'objet désigné. Lors des accès suivants, le chemin d'accès est utilisé pour accéder directement à l'objet sans interprétation.

L'avantage de la liaison statique est la rapidité d'accès. Les avantages de la liaison dynamique sont la facilité d'évolution (la modification d'un objet ne nécessite pas d'édition de liens de tout l'ensemble du programme) et la souplesse d'exécution. On peut déterminer au dernier moment l'identité d'un objet (ce qui est agréable, et peut être nécessaire, dans un schéma de programmation par objets) ou son site d'exécution (ce qui facilite la migration ou le placement dynamique dans un système réparti) [Ho 91]. Un exemple du schéma statique est l'édition de lien classique, qui construit un module-objet exécutable où toutes les références sont résolues en adresses. Un exemple du premier schéma de liaison dynamique est celui des systèmes à capacités (dans lequel les références sont interprétées par la machine à capacité) ; un exemple du second schéma de liaison dynamique est celui de Multics.

Le système Guide-1 utilise un schéma de liaison dynamique interprétatif qui malgré les améliorations apportées par la gestion d'un cache d'adresses reste un schéma assez pénalisant du point de vue des performances. Nous avons donc choisi de mettre en œuvre dans Eliott un schéma à la Multics de liaison dynamique au premier accès. Comme les besoins que nous venons d'exprimer pour le partage interdisent de mettre des adresses dans les objets, nous avons décidé d'utiliser la technique classique du segment de liaison à la Multics, adaptée à la gestion des objets. Pour simplifier l'architecture, et avoir une définition propre et modulaire des mécanismes, nous avons défini une machine à segments offrant un adressage segmenté et un mécanisme de liaison dynamique au premier accès. Puis nous avons défini comment étaient mises en œuvre dans des segments les différentes catégories d'objets gérés par Eliott [Freyssinet 91a]. Les grappes que nous avons introduites plus haut ne regroupent donc pas des objets mais des segments qui contiennent les objets.

Protection

Le système de protection doit résoudre les problèmes suivants :

- Garantir l'isolation mutuelle des usagers (une erreur dans un objet d'un usager ne doit pas pouvoir avoir d'effets sur les objets d'un autre usager) et des exécutions de programmes (l'exécution d'un programme par un usager ne doit pas avoir d'effet imprévu sur un programme d'un autre usager, ou sur un autre programme du même usager).
- Permettre à un usager de spécifier les droits d'accès accordés aux autres usagers, ou à des groupes d'usagers, sur les objets dont il est propriétaire.
- Être cohérent avec le modèle d'objets, c'est à dire que les règles d'accès doivent être définies en termes des méthodes applicables sur les objets plutôt qu'en terme des actions élémentaires lire, écrire, exécuter.
- Permettre l'extension contrôlée et temporaire des droits d'un usager (ce problème classique en protection porte le nom de délégation des droits).

Illustrons le problème de la délégation sur un exemple tiré de [Kowalski 90]. Une classe *jeu* exporte une méthode *jouer*. Chaque joueur qui veut jouer appelle la méthode *jouer* sur une instance de jeu. Un objet *palmarès* est utilisé pour conserver le meilleur résultat de chacun des joueurs. L'objet palmarès est mis à jour au moyen de la méthode *éditer_palmarès* (*usager*, *nouveau_total*). Tout joueur doit pouvoir mettre à jour *palmarès* depuis son exemplaire de *jeu*, mais ne doit pas pouvoir le faire depuis un autre objet.

Le modèle de protection que nous avons mis en œuvre pour résoudre ces problèmes, s'appuie sur les concepts d'usager, de propriétaire, de groupe et de vue. Avant de présenter le modèle donnons quelques définitions.

- *Usager*. Un usager est une entité administrative qui est désigné par Eliott au moyen d'un identificateur (*uid*) qui est unique dans l'espace et dans le temps.
- *Propriétaire*. Chaque objet a un propriétaire défini par un attribut dont la valeur est l'*uid* de son propriétaire. L'attribut propriétaire d'un objet est égal à celui de l'objet qui l'a créé. Eliott fournit une primitive qui permet aux administrateurs du système (et à eux seulement) de changer le propriétaire d'un objet.
- *Groupe*. Un groupe est un ensemble d'utilisateurs qui est identifié par un *uid*. Chaque usager peut être vu comme un groupe réduit à lui même. Un usager peut être membre de plusieurs groupes.
- *Vue*. Une vue est un sous-ensemble des méthodes qui peuvent être appelées sur une instance d'une classe. La description d'une vue est attachée à l'objet-classe. Un objet-classe peut exporter un nombre quelconque de vues sur ses instances.

Chaque domaine s'exécute pour le compte d'un utilisateur appelé son propriétaire et pour le compte d'un groupe auquel il appartient. Par défaut, le groupe associé à un domaine à sa création est celui du domaine qui l'a créé. Un service du système, le *gestionnaire de groupes*, contrôle l'allocation des identificateurs de groupe aux domaines. Le gestionnaire de groupe doit authentifier l'*uid* du propriétaire et vérifier que l'usager est bien membre du groupe pour le compte duquel il veut s'exécuter. Le gestionnaire de groupe occupe une position clé dans notre mécanisme de protection, car les droits d'accès aux objets d'un domaine dépendent du groupe qui lui est associé.

Comme nous l'avons vu, les domaines ne partagent pas de contexte ce qui assure l'isolation mutuelle des domaines qui s'exécutent sur le même site en même temps. Pour assurer le respect des droits d'accès, les objets de propriétaires différents utilisés par un domaine ne seront jamais couplés dans le même contexte. Donc, si une activité exécutant une méthode sur un objet appartenant à X effectue un appel de méthode sur un objet appartenant à Y, elle doit exécuter un appel inter-contexte, qui est interprété par Eliott. Ainsi une erreur

ou une malveillance dans une méthode d'un objet ne peut affecter que les objets appartenant au même propriétaire.

Les droits d'accès associés à un objet sont mis en œuvre par des listes d'accès (*Acl*) qui définissent pour chaque objet la vue de lui-même qu'il fournit aux différents groupes. L'*Acl* ne peut être modifiée qu'au moyen d'appels à la primitive de modification des droits effectués depuis un objet ayant le même propriétaire que l'objet auquel elle est associée. Les modifications des droits accordés à un groupe sur un objet sont sans effet sur les domaines de ce groupe ayant déjà couplé l'objet. Si le propriétaire d'un objet n'a pas spécifié de liste d'accès, l'objet n'est accessible qu'à lui même et sans contrôle de vue (le propriétaire peut appeler toutes les méthodes de la classe de l'objet). Un objet peut être déclaré non protégé, ce qui signifie que tous les groupes ont la même vue de cet objet, celle de son propriétaire.

Cette définition des entrées d'*Acl* en terme de vues d'un objet accordées à un usager ne permet pas de résoudre le problème de la délégation des droits. Pour résoudre ce problème, on attache à chaque objet un attribut appelé attribut de visibilité qui, selon qu'il est positionné ou non, autorise l'appel de l'objet depuis un objet appartenant à un autre utilisateur. Combiné au mécanisme d'isolation que nous avons présenté plus haut, l'attribut de visibilité fournit une solution. Reprenons l'exemple du jeu donné plus haut. Lors de l'installation d'un nouveau joueur, l'administrateur du jeu crée un exemplaire de *jeu* dont il reste propriétaire avec l'attribut de visibilité positionné (peut être appelé depuis un objet d'un autre propriétaire) et des droits d'accès permettant au nouveau joueur d'appeler *jouer*. Lors de l'installation du jeu dans le système, l'administrateur avait créé un objet *palmarès* avec l'attribut de visibilité non positionné (objet invisible par les autres usagers) dont il est propriétaire avec tous les droits d'accès donnés à tout le monde. Une activité appelant la méthode *jouer* sur l'objet *jeu* passera dans un contexte de son domaine réservé aux objets appartenant à l'administrateur du jeu et pourra donc mettre à jour *palmarès*. En revanche elle ne pourrait pas atteindre *palmarès* depuis un de ses propres objets même si elle en connaissait la référence système. On garantit ainsi que tout domaine d'un joueur pourra mettre à jour son résultat, mais qu'il ne pourra pas le faire depuis n'importe quel objet.

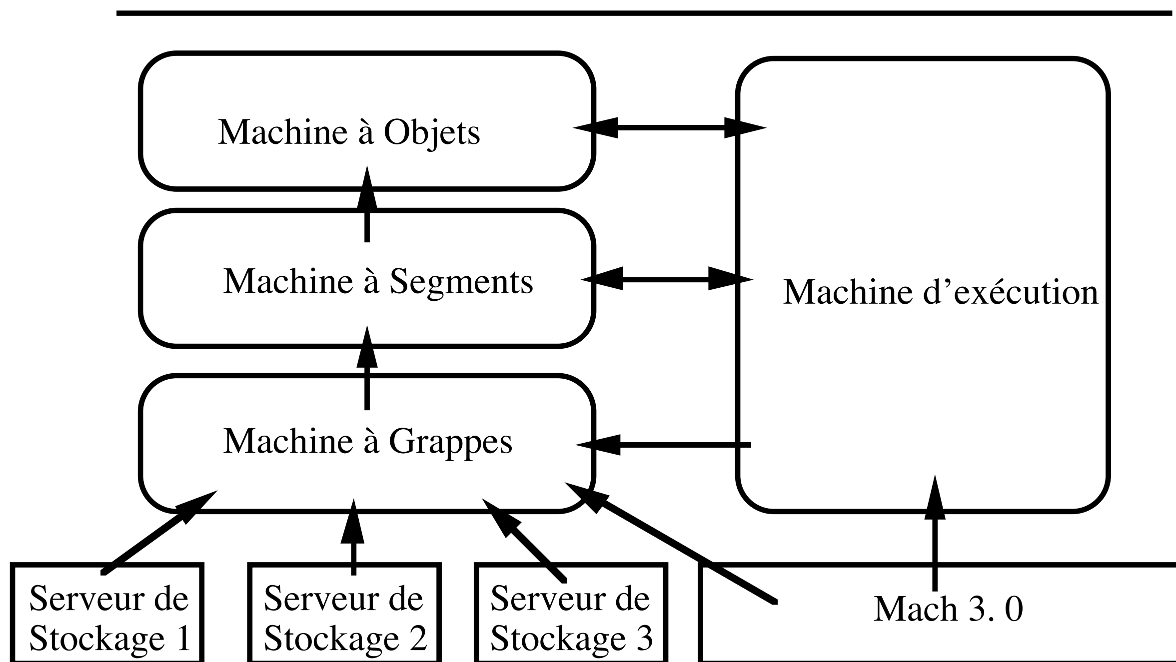
Pour résoudre le problème de la délégation des droits, la voie la plus naturelle pour un modèle à objets est de faire dépendre les droits sur un objet non seulement de l'usager qui veut y accéder, mais également de la classe depuis laquelle il effectue l'accès à cet objet. C'est d'ailleurs la solution choisie dans le système à objets Birlix [Kowalski 90]. Dans ce système, quand une méthode est appelée sur un objet, on teste non seulement l'*uid* du groupe associé au domaine mais également la référence système de l'objet appelant avant d'autoriser l'accès. Notre mise en œuvre du mécanisme d'appel de méthode ne nous permet pas de vérifier de manière sûre la référence système de l'objet appelant [Hagimont 92], aussi nous ne pouvons pas appliquer ce modèle. Cependant le modèle que nous avons adopté, s'il est moins élégant, permet une mise en œuvre qui se marie parfaitement avec notre schéma d'adressage. En effet le chemin d'accès à l'objet est construit à la liaison de l'objet de telle sorte que seules les méthodes autorisées par la liste d'accès seront adressables par le domaine appelant. Ces mécanismes sont décrits dans [Hagimont 92]

3.3.3.3. Architecture générale d'Eliott

L'architecture d'Eliott (voir la figure ci-après) est définie par :

- La machine à objets. La machine à objets réalise au moyen des segments les objets instances, classes et programmes. Elle met en œuvre les mécanismes de protection des objets que nous venons de décrire. Elle fournit l'appel de méthode sur un objet.

• La machine à segments. La machine à segments fournit les mécanismes d'adressage segmenté et de liaison dynamique.
• La machine à grappes. La machine à grappes gère des grappes (regroupement de segments), à partir d'un contexte (couplage/découplage de grappes, recopie de grappe en mémoire de stockage). Elle interface d'autre part le noyau Eliott avec les serveurs de stockage qu'il utilise. Les grappes sont regroupées dans des volumes qui sont les unités de structuration choisies au niveau du système de stockage.
• La machine d'exécution. La machine d'exécution gère les domaines et les activités qui sont les composants du modèle d'exécution. Elle gère également l'identification des entités.



3.3.3.4. Réalisation

La réalisation du système Eliott a commencé en fin 1991, au dessus du noyau Mach 3.0 sur des machines Bull-Zenith 386 et 486. Le développement utilise le système de gestion de composants logiciels Adèle, réalisé au Laboratoire de Génie Informatique de l'IMAG.

Une version multi-site du système a été réalisée, elle comporte les composant suivants :

• la machine à objets et la machine à segments ont été réalisées de manière regroupées afin d'avoir une réalisation plus efficace. L'ensemble des mécanismes de protection ont été implantés.

• La machine à grappes est réalisée selon un modèle client-serveur. La partie client s'exécute dans les contextes des domaines et la partie serveur dans une tâche paginateur Mach 3.0. Le service de gestion de la MPO ne gère actuellement qu'un seul volume mis en œuvre dans des fichiers Unix.

• La machine d'exécution gère les domaines et les activités, ainsi que l'appel inter-contexte.

Quelques mesures préliminaires ont été réalisées sur ce prototype et ont permis une première comparaison avec le prototype Guide-1 sur la même machine (Bull-Zenith 486),

pour le mécanisme de base qu'est l'appel d'une méthode sur un objet exécutable dans le même contexte que l'appelant (objet appartenant à un même propriétaire).

	Guide-1	Eliott
Premier appel	59 µs	64 µs
Appels suivants	49 µs	3,4µs

Ces résultats montrent que le mécanisme de liaison dynamique d'Eliott (à la Multics), moyennant un surcoût de 10% pour le premier appel de méthode par rapport à Guide-1, permet un gain considérable sur les appels ultérieurs.

Une version répartie d'Eliott, est prévue pour Juin 1993, et nous pourrons alors commencer une campagne de mesure systématique sur les différents mécanismes que nous avons introduits dans Eliott.

3.4. Un travail sur la duplication d'objets dans le cadre d'Arjuna

A la fin de la première phase du projet Guide, j'ai eu l'opportunité de travailler un an à l'Université de Newcastle upon Tyne dans l'équipe qui développait le projet Arjuna [Shrivastava 89]. Quoique mon travail n'ait pas porté sur le système Arjuna lui-même, je présente d'abord un bref rappel des traits principaux du système Arjuna, puis je donne les motivations qui m'ont amené à définir le protocole TSync [McCue 90], un protocole de gestion d'objets dupliqués manipulés uniquement par des transactions, sur lequel j'ai travaillé avec D. McCue.

3.4.1. Le projet Arjuna

Arjuna est un système réparti à objets résistant aux pannes qui a été mis en œuvre sur un ensemble de stations de travail Unix connectées par un réseau Ethernet. Arjuna utilise des actions atomiques emboîtées (transactions) comme outil de structuration et d'exécution des programmes. Les programmes opèrent sur des objets qui sont des instances de types abstraits (en fait des objets C++). Les objets gérés par Arjuna sont persistants et sont utilisés pour conserver l'état du système. En garantissant qu'on ne peut accéder aux objets que par des actions atomiques, l'intégrité des objets peut être garantie et donc l'intégrité du système est maintenue en présence de pannes telles que la mort d'un site ou la perte de messages. Arjuna fournit les mécanismes qui sont nécessaires pour la distribution, la persistance et la résistance aux pannes. Une des caractéristiques principale d'Arjuna est que ces mécanismes ont été intégrés au modèle d'objets de C++ qui constituent l'interface d'accès au système.

Arjuna, comme Guide-1, a fait le choix d'offrir la distribution et la persistance à travers un langage à objets. Cependant Arjuna diffère de Guide sur les points suivants :

- Plutôt que construire un nouveau langage, Arjuna a choisi d'enrichir le langage C++ en lui ajoutant la persistance et la distribution. Cette extension est faite en utilisant les outils du langage C++ (i.e. une hiérarchie de classes prédéfinies).
- L'unique mode d'exécution dans Arjuna est la transaction.
- Sur le plan de la mise en œuvre, Arjuna s'appuie sur un modèle client-serveur et sur un système d'appel de procédure à distance, Rajdoot [Panzieri 88]. Le générateur de talons utilisé [Parrington 90] apporte un certain nombre de limitations aux traits du langage C++ ; cependant, aucun contrôle statique du respect par le programmeur de ces limitations n'est effectué.

3.4.2. Duplication active d'objets accédés par des transactions

La disponibilité des objets dans un système distribué peut être accrue en dupliquant les objets sur un certain nombre de sites. Le groupe des objets dupliqués se comporte comme un objet unique du point de vue de l'usager de l'objet. Le but d'un protocole de gestion d'objets dupliqués est de maintenir fortement cohérent les différents exemplaires (duplicata) d'un même objet.

Dans un système à objets comme Arjuna l'unité naturelle de duplication est l'objet. Dans un schéma de duplication active, tous les objets dupliqués implantés sur des sites actifs (i.e. non en panne) exécutent chacun des appels de méthodes. Quand un usager appelle une méthode sur un objet, chacun de ses duplicata exécute l'appel indépendamment des autres et retourne le résultat à l'usager. Une fonction de filtrage des réponses doit être exécutée (i.e. comparaison des résultats pour s'assurer qu'ils sont identiques, vote sur le résultat pour s'assurer qu'une majorité des résultats sont identiques) avant de passer le résultat à l'usager appelant.

Une manière d'assurer que l'état des duplicata reste identique est d'assurer que tous les duplicata reçoivent dans le même ordre tous les appels même en présence de pannes soit du système de communication soit d'un ou de plusieurs sites (i.e. en ordonnant tous les messages contenant des appels de méthode). Si les opérations sont déterministes, chaque duplicata, recevant le même flot d'appels, devra produire le même flot de résultats. Le problème de la cohérence des duplicata se transforme en un problème de communications fiables et atomiques entre les différents duplicata d'un même objet.

Cependant, si un protocole de communication assurant l'ordre et la délivrance atomique des messages est suffisant pour assurer la cohérence des duplicata, il n'est pas nécessaire dans le cas où, comme dans Arjuna, les objets ne sont accédés que par des transactions. Par exemple, deux opérations de lecture consécutives peuvent être exécutées dans un ordre différents par les différents duplicata d'un même objet sans affecter la cohérence des duplicata, ni des réponses. Il n'est donc pas nécessaire que tous les duplicata voient tous les appels dans le même ordre. Ce qui détermine les appels pour lesquels l'ordre est important est la contrainte de "sérialisabilité" qui est assurée par le contrôle de concurrence du mécanisme de transaction. Il n'est donc pas nécessaire que le sous-système de communication assure l'ordonnancement de tous les appels de méthode. Si un certain nombre d'opérations de lecture consécutives sont effectuées sur un objet dupliqué, un duplicata particulier peut n'avoir pas effectué tout ou partie de ces opérations tout en restant un membre à jour du groupe de duplicata. Cette propriété reste vraie aussi longtemps que le duplicata ne perd pas d'opération d'écriture. L'utilisation de duplicata actifs permet à l'usager d'un objet dupliqué de recevoir le résultat d'un appel de méthode sur cet objet même dans le cas où un ou plusieurs duplicata n'ont jamais reçu l'appel. Si les conséquences de la politique de contrôle de concurrence (i.e. acquisition de verrou) et la nature des opérations (i.e. lecture ou écriture) peut être reflétée à l'interface de communication, le coût des communications peut être diminué en appliquant des politiques différentes aux différents appels de méthode transférés.

[Mishra 89] décrit Psync, un protocole de diffusion qui emploie un graphe de contexte, maintenu sur chaque site et mis à jour au moyen d'informations transmises dans les messages, pour obtenir un ordre partiel distribué entre les différents sites. Le protocole Tsync que nous avons conçu reprend l'idée d'utiliser les messages pour transporter les informations nécessaires à chaque site pour reconstruire l'état de la transaction (i.e. la liste des identifications des dernières opérations d'écriture effectuées sur chacun des objets écrits

par la transaction). A la réception d'une demande d'exécution d'opération, un duplicata pourra, grâce aux informations contenues dans le message, savoir s'il est à jour (il a bien effectué la dernière opération d'écriture effectuée par la transaction) ou non. S'il n'est pas à jour il abandonne la transaction. Dans le contexte d'objets manipulés uniquement par des transactions un ordre total n'est en général pas nécessaire puisque le contrôle de concurrence associé à la transaction assure un ordre d'accès aux objets à un plus haut niveau. De même on n'est pas obligé d'assurer que les messages soient délivrés de manière atomique puisque certains messages peuvent être perdus sans affecter l'état des objets et la poursuite des calculs.

3.4.3. Bilan

Mon séjour à Newcastle avait essentiellement pour but de me familliariser avec les techniques utilisées pour rendre robuste un système réparti à objets. Il m'a permis de définir les grandes lignes de ce qui devrait permettre au système Eliott d'assurer un meilleur service en présence de pannes [Krakowiak 91]. Deux types de travaux sont actuellement en cours : d'une part des travaux de P. Y. Chevalier [Chevalier 92] sur la définition et la réalisation d'un système de stockage d'objets résistant aux pannes, et d'autre part les travaux, moins avancés, de Y. Laribi sur la duplication de domaines et d'activités dans Eliott. Le travail effectué sur le protocole Tsync n'a pas été terminé, dans la mesure où son évaluation reste à faire.

4. Conclusion

Après avoir décrit l'évolution des systèmes opératoires au cours de ces vingt dernières années (première partie), puis les projets auxquels j'ai participé (deuxième partie), je conclus en trois points.

Dans un bref retour sur l'évolution, je situe mes travaux personnels par rapport aux grandes phases de la recherche. Je commente ensuite la situation actuelle des systèmes, en développant plus particulièrement les aspects liés à mon travail (micro-noyaux et langages à objets). J'essaie enfin de dégager quelques tendances pour les systèmes opératoires du futur proche.

4.1. Retour sur l'évolution

La recherche en systèmes s'est toujours intéressée à deux types de problèmes :

- Assurer la meilleure répartition possible des ressources logiques et physiques entre les utilisateurs ;
- Fournir une interface d'utilisation et des services de plus haut niveau d'abstraction que ceux de la machine hôte.

Selon la technologie disponible, l'accent est mis sur l'un ou l'autre de ces deux aspects qui coexistent au sein d'un système d'exploitation.

Dans une première phase, caractérisée par des ressources limitées et coûteuses, l'accent est mis sur une gestion efficace des ressources physiques et sur les problèmes posés par leur partage entre des applications concurrentes. Cette phase connaît son apogée avec Multics et les systèmes à capacités. L'interface offerte aux utilisateurs est très proche de la machine physique support. Dans les projets Gemau et Ours, nous avons cherché à fournir une interface de plus haut niveau, permettant en particulier la cohabitation d'utilisateurs finaux et de constructeurs de sous-systèmes. Nous avons du cependant consacrer beaucoup d'efforts à la gestion des ressources physiques.

La seconde phase que nous pouvons distinguer correspond à l'interconnexion de systèmes existants par des réseaux généraux ou locaux. Les systèmes opératoires sont donc étendus par des dispositifs permettant l'échange d'informations entre machines. La nécessité de connecter des machines hétérogènes a conduit à introduire des normes tant pour les niveaux physiques que logiques, de façon probablement hâtive par rapport à l'état des connaissances. La participation aux projets Danube puis Factor, la participation à la rédaction du livre Cornafion m'ont donné une bonne connaissance de ces problèmes de connexion.

Enfin, la troisième phase, qui se poursuit actuellement, vise à construire des systèmes intégrés sur un ensemble de machines interconnectées. L'interface utilisateur doit fournir des outils de développement d'applications réparties offrant des fonctions analogues à celle fournies par les outils disponibles dans les systèmes centralisés. La gestion des ressources physiques est simplifiée et les problèmes clés sont ceux de la désignation et du partage des informations, de la tolérance aux défaillances et de l'administration d'un système composé d'un grand nombre de machines.

Guide offre un langage d'écriture d'applications réparties et son environnement d'exécution sur Unix et sur micro-noyaux, ainsi qu'un modèle pour l'exécution de ces applications. Il reste à prendre en compte les problèmes d'administration et de résistance aux défaillances, et surtout à passer à des expérimentations faisant intervenir de quelques dizaines à quelques centaines de machines.

4.2. Situation actuelle

Dans cette section, j'essaie de présenter la situation dans deux domaines qui ont connu un certain effet de mode aux cours des dernières années et ont donc fait l'objet de nombreux travaux dans la communauté de recherche : les micro-noyaux et les langages et les systèmes à objets. Si la plupart des projets récents en systèmes répartis, et notamment Guide, ont touché à l'un ou à l'autre de ces aspects, leur influence sur les systèmes effectivement utilisés et diffusés sur le marché est encore marginale.

4.2.1. Micro-noyaux

Comme nous l'avons montré au §2, les machines dédiées à un langage ont été condamnées par le fait que les constructeurs ont constaté que les instructions très spécialisées qu'elles fournissaient coûtaient cher et qu'elles n'étaient en fait que très peu utilisées par les compilateurs des langages de haut niveau auxquels elles étaient destinées. Cette constatation est le début d'une évolution qui a abouti aux architectures RISC. Ces machines permettent aux compilateurs de programmer exactement ce dont ils ont besoin et d'obtenir un code plus efficace. Une observation analogue peut être faite pour les systèmes ; il est en effet très difficile, sinon impossible, de définir une interface et une politique de gestion des ressources capables de satisfaire efficacement des sous-systèmes très différents. Il est en revanche plus facile de définir un ensemble, de préférence petit, de fonctions élémentaires qui permettent l'écriture de sous-systèmes mettant en œuvre diverses politiques de gestion des ressources, et leur cohabitation sur une même plate-forme. Les micro-noyaux visent à définir une telle plate-forme capable

- de servir de support efficace à des sous-systèmes dédiés,
- d'être facilement portable d'une architecture de machine à une autre.

Nous allons analyser comment les micro-noyaux actuels répondent à ces deux exigences.

Les micro-noyaux comme support de sous-systèmes

Les micro-noyaux actuels fournissent : le modèle client-serveur comme modèle de structuration et de construction de sous-systèmes ; la tâche que l'on peut voir comme la virtualisation d'un multiprocesseur à mémoire commune ; le mécanisme des paginateurs externes qui permet aux sous-systèmes de contrôler la mémoire virtuelle associée à une tâche ; un système de communication entre les flots d'exécution ou *thread* appartenant à une même tâche ou à des tâches différentes ; enfin des entrées-sorties au niveau coupleur.

L'ensemble des fonctions proposées représente bien le minimum que l'on retrouve dans tout système. Et, de ce point de vue, les micro-noyaux sont bien les supports génériques qu'ils prétendent être, ayant permis la mise en œuvre aussi bien d'Unix que de MS-DOS ou le développement du système du NEXT. Cependant, on doit remarquer que les micro-noyaux ne sont pas toujours prévus pour supporter des systèmes répartis. Ainsi, dans Mach 3.0, la communication entre tâches n'appartenant pas au même site passe par un serveur (le *net message server*) et n'est donc pas intégrée au noyau ce qui est source d'inefficacité. L'OSF propose une version de Mach 3.0 intégrant les communications sur réseau qui pourrait être la base de développement attendue pour les sous-systèmes répartis.

On doit également s'interroger, sur le bien fondé du compromis que constitue le mécanisme des paginateurs externes. Ce mécanisme, permet à l'usager d'associer un contenu logique (i.e. un fichier) à des régions de mémoire et d'effectuer, contrairement au mécanisme de mémoire partagée d'Unix, le chargement à la demande des régions de

mémoire. En revanche il ne permet pas à l'usager de définir sa propre politique de pagination (choix de la page à remplacer). Si, comme le prétend Wilkes [Wilkes 92], les changements qui interviennent dans les architectures (i.e. temps d'accès aux mémoire centrales de plus en plus petit alors que les temps d'accès aux mémoires secondaires restent stables) remettaient les recherches sur les algorithmes de pagination à l'ordre du jour de notre communauté, les micro-noyaux dans leur état actuel ne pourraient pas constituer l'outil d'expérimentation nécessaire. On doit cependant signaler qu'il n'est pas certain qu'avec les progrès faits par les outils de modélisation et de simulation [Heidelberger 84] il soit réellement indispensable d'avoir une plate-forme d'expérimentation pour étudier de nouveaux algorithmes de pagination. En revanche, ce qui pourrait se révéler nécessaire est de pouvoir faire varier la politique de pagination avec l'application.

Les micro-noyaux comme base de portage

Les temps de portage d'un micro-noyau Mach 3.0 sans ses pilotes (*drivers*) d'une architecture à une autre, donnés par l'OSF (12 hommes*mois) suffiraient à eux seuls à justifier d'un point de vue économique les micro-noyaux. Il faut remarquer que beaucoup de constructeurs avaient déjà architecturé leur système en tenant compte du fait que leur durée de vie serait beaucoup plus grande que celle des architectures matérielles supports. Aussi, parmi les nombreux constructeurs participant à l'OSF, peu ont remplacé leur propre noyau par Mach 3.0. Les raisons commerciales s'opposent probablement ici aux raisons techniques. C'est ce qui permet peut-être d'expliquer pourquoi "l'effet RISC" (un programme sur une nouvelle architecture s'exécute plus vite que l'instruction spécialisée, qu'il émule, d'un modèle précédent) n'est pas encore visible pour les systèmes.

4.2.2. Langages et systèmes à objets

De la même façon que les micro-noyaux fournissent une solution aux problèmes que pose l'évolution très rapide des architectures matérielles, les langages à objets sont un élément de réponse aux besoins exprimés par l'ingénierie du logiciel. Dès le début des années 70, un certain nombre de principes de conception et de construction de logiciels étaient énoncés [Dijkstra 71]. Selon ces principes un programme doit présenter les caractéristiques suivantes :

- *Abstraction*. Un programme doit être défini indépendamment des structures de données qu'il utilise et de la façon dont il est implanté sur une architecture donnée (i.e. indépendance entre interface et réalisation)
- *Structuration*. Un grand programme doit être décomposé en composantes de taille raisonnable ; les relations entre les différentes composantes doivent être clairement définies.
- *Modularité*. La mise en œuvre d'une composante d'un programme ne doit pas dépendre de la façon dont sont implantés les autres composantes du programme.
- *Concision*. Le code doit être clair et compréhensible
- *Vérifiabilité*. Le programme doit être facile à tester et à mettre au point.

Ces principes sont ceux qui ont donné naissance d'abord au modèle basé sur la procédure, puis au modèle basé sur le module et enfin au modèle basé sur l'objet. Par rapport au modèle modulaire, le modèle objet permet la spécialisation grâce au mécanisme de sous-typage, la réutilisation de code grâce au mécanisme d'héritage et la prise en compte de l'hétérogénéité grâce aux implantations multiples d'une même interface d'accès à un objet. De plus, la possibilité de remplacer dynamiquement une mise en œuvre par une autre (ayant une interface compatible) est une fonction importante pour la conception de systèmes et

d'applications évolutives. La complexité croissante des logiciels et la nécessité de pouvoir les maintenir et les faire évoluer expliquent le succès grandissant du modèle objet.

La conviction, largement partagée par la communauté système, qu'un système est un environnement d'exécution pour un langage de programmation, ainsi que les possibilités offertes par le modèle objet, expliquent que la presque totalité des études qui sont menées par les équipes de recherches offrent un support à ce modèle (i.e. l'objet existe comme entité au niveau du système). Ce support est fourni soit comme un ensemble de primitives pouvant être appelées depuis un langage, soit fortement couplé à un langage et dans ce dernier cas le système n'apparaît à l'usager que comme l'exécutif du langage. Cette deuxième approche est celle que nous avons choisie dans Guide, mais c'est également celle qui a été choisie par la plupart des projets de systèmes répartis dont nous avons parlé dans la section 2. Le couplage fort du système et du langage de programmation, en même temps qu'il permet une meilleure efficacité à l'exécution rend plus sûre l'utilisation par l'usager des fonctions offertes par le système. Elle permet, en effet d'effectuer à la compilation la détection d'un certain nombre d'erreurs. Enfin elle est cohérente avec l'architecture de construction de sous-systèmes offerte par les micro-noyaux : un environnement d'exécution réparti à objets constituant naturellement un sous-système parmi d'autre. Il reste cependant que pour sortir du domaine de la recherche, les systèmes répartis à objets doivent permettre la coopération d'applications écrites dans plusieurs langages (et donc pouvant être mise en œuvre sur des sous-systèmes différents) sans pour autant ouvrir de brèches dans les contrôles propres à chacun des langages. Des premiers travaux ont été entrepris par les constructeurs visant à définir une architecture générique permettant la coopération entre des environnements à objets hétérogènes. Mais la définition de l'architecture proposée par l'Object Management Group (OMG), appelée CORBA[28], reste encore trop imprécise pour qu'elle puisse être la base définitive de coopération entre les environnement à objets annoncés par les constructeurs.

4.2.3. Situation du marché

A l'heure actuelle sont utilisés des systèmes pour micro-ordinateurs (MS-DOS et Mac-OS) ; Unix s'est imposé en tant que système d'exploitation des stations de travail ; enfin il subsiste un certain nombre de systèmes dits propriétaires supportant principalement des applications de gestion ou des applications scientifiques.

Cette stabilisation provient de la nécessité d'assurer le fonctionnement d'un très grand nombre d'applications à longue durée de vie, qu'il n'est pas question de réécrire à chaque changement de machine (l'évolution technologique amène à remplacer les machines tous les trois ou quatre ans alors que les applications ont facilement une durée de vie supérieure à dix ans). Unix apparaît dès lors comme une norme de fait qui permet en outre de faire communiquer sans trop de problèmes des matériels de divers constructeurs ainsi que le partage de logiciel d'une machine à une autre.

Enfin, la réalisation de versions d'Unix conçues comme des extensions à des micro-noyaux (Mach et Chorus) doit permettre de simplifier la réalisation d'Unix pour les machines futures. Des environnements d'exécution répartie, construits au dessus d'Unix, sont ou vont être offerts par les constructeurs. Ils permettront d'interconnecter de nombreuses machines hétérogènes. L'accès à un système informatique sera presque aussi banal qu'une connexion électrique ou téléphonique. C'est pourquoi de nombreux informaticiens considèrent que la recherche en systèmes opératoires se trouve dans une phase terminale et qu'il faut maintenant

---

[28] The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Revision 1.1, Draft 10 December 1991.

se concentrer sur les extensions aux outils existants, en particulier en ce qui concerne les environnements de développement et de mise au point de programmes.

4.3. Perspectives

Je suis pour ma part persuadé que la recherche en système a encore de belles perspectives, pour deux raisons principales : d'une part, dans toute l'histoire de l'informatique, c'est la technologie matérielle qui s'est trouvée en avance et qui a littéralement "tiré" la recherche en logiciel, et en particulier en systèmes opératoires ; d'autre part une synthèse reste aujourd'hui à faire puisque plusieurs classes de systèmes opératoires coexistent et que certaines machines d'avant-garde, machines massivement parallèles surtout, fonctionnent pratiquement sans système d'exploitation.

Deux apports technologiques majeurs sont sur le point d'être disponibles : les réseaux de communication à très haut débit et des ordinateurs à grande capacité d'adressage. Ces apports sont suffisamment novateurs pour amener les chercheurs à remettre en cause certains choix faits dans les systèmes actuels.

Communication à haut débit

Les systèmes répartis actuels sont structurés pour tirer au mieux parti des performances différentes entre le bus d'un système centralisé et celles des réseaux. Or les réseaux du futur auront des vitesses de transmission du même ordre que les échanges processeur mémoire. Dès lors, c'est l'ensemble des informations d'un système d'exploitation qui devient "immédiatement " disponible et des solutions écartées pour des raisons de performances deviennent viables.

Avec ces nouveaux réseaux, les différences entre les systèmes centralisés, répartis, voire massivement parallèles vont s'estomper et une synthèse devrait être possible. Enfin, les capacités de ces réseaux permettent d'envisager la gestion d'information multimédia (i.e. image, son et texte) dans un système d'exploitation.

Processeurs à grande capacité d'adressage

Ce facteur technologique peut changer de manière révolutionnaire la façon dont les systèmes opératoires et les applications utilisent la mémoire virtuelle. Un champ d'adresse de 64 bits peut éliminer la nécessité d'avoir à réutiliser des adresses dans le but de fournir à chaque processus un espace d'adressage suffisamment grand, supprimant ainsi une des hypothèses de base sur laquelle sont construits les systèmes depuis les années 60 [Chase 92].

Avec de telles capacités d'adressage il devient possible (et tentant) de considérer que l'ensemble des données gérées par un système réparti appartiennent à un même espace d'adressage virtuel global. On retombe alors sur des problèmes d'allocation de mémoire entre les différents sites du système réparti, de récupération des références sur un objet qui a été déplacé, enfin de gestion des espaces disques de pagination et de sauvegarde si on veut tolérer les défaillances.

Ces évolutions technologiques vont donc conduire les chercheurs à faire évoluer les systèmes opératoires. Il est sans doute trop tôt pour annoncer si ces extensions se produiront en continuité avec les outils actuels, ou si elles seront assez fondamentales pour entraîner la production de nouveaux concepts de base.

De même, il est difficile de prévoir si les industriels de l'informatique pourront se contenter d'une politique conservatrice autour d'Unix et de Windows/3 ou dans quel délai des questions économiques les amèneront à promouvoir de nouvelles solutions.

Quoiqu'il en soit, de nouvelles architectures ne cesseront pas d'apparaître. Leur exploitation repose sur l'existence de systèmes opératoires efficaces qu'il importe d'étudier dès maintenant.

5. Références

[Acetta 86]
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*, Jul. 1986, pp. 93-112

[Arden 66]
B. W. Arden, B. A. Galler, T. C. O'Brien, F. H. Westervelt, Program and Addressing Structure in a Time-Sharing Environment, *J. ACM*, vol. 13, n° 1, Jan. 1966, pp. 1-16

[Balter 89]
R. Balter, D. Decouchant, A. Duda, A. Freyssinet, S. Krakowiak, M. Meysembourg, M. Riveill, C. Roisin, X. Rousset de Pina, R. Scioville, G. Vandôme, Experience with object-based distributed computation in the Guide operating system, *Proc. 2 nd Int. Workshop on Object-Orientation in Operating Systems*, Asilomar, Sep. 1989, pp. 16-19

[Balter 91]
R. Balter, J. Bernadat, D. Decouchant, A. Duda, A. Freyssinet, S. Krakowiak, M. Meysembourg, P. Le Dot, H. Nguyen Van, E. Paire, M. Riveill, C. Roisin, X. Rousset de Pina, R. Scioville, G. Vandôme, Architecture and implementation of Guide, an object-oriented distributed system, *Computing Systems*, vol. 4, n° 1, 1991, pp. 31-68

[Banâtre 90]
J.P. Banâtre, M. Banâtre, *Les systèmes distribuées, Concepts et expérience de Gothic*, InterEditions, 1991

[Banâtre 91]
M. Banâtre, Gestion répartie de la mémoire, *Construction des systèmes d'exploitation répartis* (R. Balter, J. P. Banâtre et S. Krakowiak ed.), INRIA collection didactique, Chap. 5, 1991

[Belady 66]
L. A. Belady, A study of replacement algorithms for virtual storage computers, *IBM Syst. J.*, vol. 12, n° 6, Jun. 1969, pp 349-353

[Bekkers 75]
Y. Bekkers, D. Herman, M. Raynal, *Conception et réalisation d'une machine langage de haut niveau adaptée à l'écriture de systèmes*, Thèse de troisième cycle, Université de Rennes, 1975

[Bennett 73]
M. J. Bennett, A Virtual Evaluation Tool for a System having a Virtualizable, *1st Annual SIGME Symposium on Measurement and Evaluation*, Palo Alto, Feb. 1973, pp. 63-68

[Bétourné 70]
C. Bétourné, J. Ferrié, C. Kaiser, S. Krakowiak, J. Mossière, Process management and resource sharing in the multiaccess system ESOPE, *Comm. ACM*, vol 13, n° 12, Dec. 1970

[Black 86]
A. Black, N. Hutchinson, E. Jul, H. Levy, L. Carter, Distribution and abstract types in Emerald System, IEEE Trans. on Software Engineering, vol. SE-12, Dec. 1986

[Boudenant 87]
J. Boudenant, B. Feydel, P. Rolin, An IEEE 802.3 compatible deterministic protocol, *INFOCOM'87*, San Francisco, Apr. 1987, pp. 573-579

[Bourne 83]
S. R. Bourne, *The Unix System*, Addison-Wesley, 1983

[Boyer 91a]
F. Boyer, J. Cayuela, P. Y. Chevalier, A. Freyssinet, D. Hagimont, Supporting an Object Oriented Distributing System : Experience with Unix, Mach and Chorus, *Proc. Symposium on Experience with Distributed and Multiprocessor Systems*, Atlanta, Mar. 1991

[Boyer 91b]
F. Boyer, A Causal Distributed Shared Memory Based on External Pagers, *Proc. of the 2nd Usenix Mach Symposium*, Monterey, Nov. 1991, pp 41-59

[Briat 76 a]
J. Briat, J. P. Verjus, Implication de certaines propriétés d'un noyau de système (MAS) sur son langage d'écriture, *BIGRE*, n° 4, Oct. 1976

[Briat 76 b]
J. Briat, X. Rousset de Pina, J. P. Verjus, Le projet OURS : ses principes, *Actes du Congrès de l'AFCET, Gif sur Yvette*, Nov. 1976, pp 745-55

[Brownbridge 82]
D. R. Brownbridge, L. F. Marshall, B. Randell, The Newcastle connection or Unixes of the world unite!, *Sofware-Practise and Experience*, vol. 12, n° 12, Dec.1982, pp. 1147-1162

[Bryant 91]
R. Bryant, P. Carani, H. Chang, B. Rosenburg, Supporting Structured Shared Virtual Memory under Mach, *Proc. of the 2nd Usenix Mach Symposium*, Monterey, Nov. 1991, pp. 59-76

[Burroughs 64]
Burroughs B5500 information Processing System Reference Manual, Burroughs Corp., 1964

[Cahill 92]
V. Cahill, R. Balter, X. Rousset de Pina, N. Harris, *The Comandos Distributed Application Platform*, Springer Verlag, Esprit Series, à paraître 1992

[Carter 91]
J. B. Carter, J. K. Bennett, W. Zwaenepoel, Implementation and Performance of Munin, *Proceedings of the thirteenth Symposium on Operating Systems Principles*, Oct. 1991, pp. 152-164

[Chase 89]
J. S. Chase, F. G. Amador, E. D. Lazowska, H. M. Levy, R. J. Littlefield, The Amber System: Parallel Programming on a Network of Multiprocessors, *Computer Science and Engineering Tech. Rep.*, Univ. of Washington, n° 89-04-01, Apr. 1989, pp. 1-20

[Chase 92]
J. S. Chase, H. Levy, M. Baker-Harvey, E. D. Lazowska, Opal: A Single Address Space System for 64-bit Architectures, *Proc. of The Third Workshop on Workstation Operating Systems (WWOS III)*, Key Biscayne, Apr. 1992

[Chevalier 92]
P. Y. Chevalier, Replicated Object Server for a Distributed Object-Oriented System, *11th Symp. on Reliable distributed Systems*, Houston, Oct. 92, pp. 4-11

[Corbato 62]
F.J. Corbato, An experimental time-sharing system, *Proc. AFIPS SJCC*, 1962

[Cornafion 81]
Cornafion, *Systèmes informatiques répartis : concepts et techniques* , Dunod, 1981

[Daley 68]
R. C. Daley, J. B. Dennis, Virtual memory, processes and sharing in Multics, *CACM*, vol.11, n° 4, May 1968, pp. 308-318

[Decitre 82]
P. Decitre, J. F. Estublier, A. Khider, X. Rousset de Pina, I. Vatton, An efficient error detection mechanism for a multicast transport service on the Danube network, *IFIP TC-6 Symposium on Local Computer Network*, Firenze, Apr. 1982

[Decouchant 88]
D. Decouchant, A. Duda, A. Freyssinet, E. Paire, M. Riveill, X. Rousset de Pina, G. Vandome, Guide: an implementation of the Comandos object-oriented distributed system on Unix, *Proc. European Unix Users Group (EUUG),* Lisbonn, Oct. 1988, pp. 181-193

[Decouchant 89 a]
D. Decouchant, A. Duda, A. Freyssinet, H. N. Van, M. Riveill, X. Rousset de Pina, Guide un Système Réparti à Objets, *Congrès de l'AFUU*, Paris, Mar. 1989, pp. 297-315

[Decouchant 89 b]
D. Decouchant, E. Paire, M. Riveill, Efficient implementation of low level synchronization primitives in the Unix based Guide kernel, *Proc. European Unix Users Group (EUUG)*, Vienna, Oct. 89, pp.

[Decouchant 91]
D. Decouchant, P. Le Dot, M. Riveill, C. Roisin, X. Rousset de Pina, A synchronization mechanism for an object-oriented distributed system, *Proc. 11th Int. Conf. on Distributed Computing Systems (ICDCS),* Arlington, Sep. 1991, pp. 152-161

[Denning 76]
P. J. Denning, K. C. Kahn, J. Leroudier, D. Potier, R. Suri, Optimal multiprogramming, *Acta informatica*, vol 7, n° 2, Feb. 1976, pp. 197-216

[Dennis 65]
J. B. Dennis, Segmentation and the design of multiprogrammed systems, *J. ACM,* vol.12, n° 4, Oct. 1965, pp 589-602

[Dennis 66]
J. B. Dennis, E. C. Van Horn Programming semantics for multiprogrammed computations, *Comm. ACM*, vol. 9, n° 3, Mar. 1966, pp. 143-155

[Deswarte 81]
Y. Deswarte, Tolérance aux fautes - sécurité et protection, *Construction des systèmes d'exploitation répartis* (R. Balter, J. P. Banâtre et S. Krakowiak ed.), INRIA collection didactique, Chapt. 9, 1991

[Dijkstra 65]
E. W. Dijkstra, Cooperating sequential processes, *Tech. report,* EWD-123, 1965

[Dijkstra 68]
E. W. Dijkstra, The structure of the THE multiprogramming system, *Comm. ACM,* vol.11, n° 5, May 1968, pp 341-346

[Dijkstra 71]
E. W. Dijkstra, Hierarchical ordering sequential process, *Acta Informatica,* vol.1, n° 1, Jan. 1971, pp 115-139

[England 75]
D. M. England, Capability concept, mechanism and structure in System 250, *RAIRO-informatique* (AFCET), vol 9, Sep. 1975, pp. 47-62

[Estublier 78]
J. Estublier, *Processus cyclique et appel procédural*, thèse de troisième cycle, Université J. Fourier (Grenoble-1), Apr. 78

[Fabry 74]
R. S. Fabry, Capability-based addressing, *Comm. ACM*, vol. 17, n° 7, Jul. 1974

[Factor 85]
Introcductory manual of Factor : an Open Local Area Network, Aptor Publication, Meylan, 1985

[Farber 72]
D. S. Farber, K. C. Larson, The system architecture of distributed computer system : The communication system, *Proc. of the Int. Conf. on Computer Communication*, Networks and Teletraffic, Brooklyn, Apr. 1972

[Freyssinet 91a]
A. Freyssinet, S. Krakowiak, S. Lacourte, A generic object-oriented virtual machine, *Proc. 3rd Int. Workshop on Object-Orientation in Operating Systems*, Palo Alto, Oct. 1991, pp. 73-77

[Freyssinet 91b]
A. Freyssinet, Architecture et réalisation d'un système réparti à objets, *Thèse de Doctorat de l'Université J. Fourier,* Grenoble, Jul. 1991

[Garrett 92]
W. E. Garrett, R. Bianchini, L. Kontothanassis, R. A. McCallum, J. Thomas, R. Wisniewski, M. L. Scott, Dynamic Sharing and Backward Compatibility on 64-bit Machines, *Computer Sciences Tech. Rep.*, Univ. of Rochester, n° 418, Apr. 1992, pp. 1- 34

[Greaves 92]
D. J. Greaves, D. McAuley, L. J. French, Protocols and interface for ATM LANs, *Proc. of the 5th IEEE Workshop on Metropolitan Area Networks*, Taormina (Italy), May 92

[Grubert 92]
O. Grubert, L. Amsaleg, L. Daynès, P. Valduriez, Eos An Environment for Object-Based Systems, *Proc of the 25th. Hawaii Int. Conf. on System Sciences*, Jan.1992, vol 1, pp 757-768

[Guiboud-Ribaud 74]
S. Guiboud-Ribaud, J. Briat, Address space and execution space in the GEMAU system, *Proc. Int. Symposium on Operating Systems*, Rocquencourt, Apr. 1974, pp 126-62.

[Guiboud-Ribaud 75]
S. Guiboud-Ribaud, *Mécanisme d'adressage et de protection dans les sytèmes informatiques. Application au noyau GEMAU*, Thèse d'état, Juin 75, Grenoble

[Hagimont 92]
D. Hagimont, S. Krakowiak, X. Rousset de Pina, Protection in an Object-Oriented Distributed System, *Proc. of the 4th Int. Workshop on Object Orientation in Operating System*, Dourdan, Sep. 1992, pp. 273-279

[Hamilton 91]
G. Hamilton, Y. A. Khalidi, M. AN. Nelson, Why Object Oriented Operating Systems are Boring, *Proc. 3rd Int. Workshop on Object-Orientation in Operating Systems*, Palo Alto, Oct. 1991, pp. 118-119

[Heidelberger 84]
P. Heidelberger, S. S. Lavenberg, Computer Peformance Evaluation Methodology, *IEEE Trans. on Computers*, vol C-33, n° 12, Dec. 1984, pp. 1195-1220

[Ho 91]
W. W. Ho, R. A. Olsson, An Approach to Genuine Dynamic Linking, *Software-Practice and Experience*, vol. 21, n° 4, Apr 1990, pp. 375-390

[Howard 88]
J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, M. J. West, Scale and performance in a distributed file system, *ACM Trans. on Comp. Systems*, vol 6, n° 1, Feb 1988, pp. 51-81

[Ingels 90]
P. Ingels, Sécurité et protection dans le système Guide, *note technique interne Bull-IMAG*, Dec. 1990, pp 1-14

[ISO 88]
ISO, International Standard 7498-2: Information processing systems-OSI Reference model - Part 2: Security Architecture, n° 2890, ISO/IEC JTC1/SC21, Jul. 1988

[Jamrozik 91]
H. Jamrozik, C. Roisin, M. Santana, A graphical debugger for object oriented distributed programs, *Proc. TOOLS USA*, Jul 91.

[Kane 88]
G. Kane, *mips RISC Architecture*, Prentice-Hall Inc, 1988

[Kahn 81]
K. C. Kahn, W. M. Corwin, T. D. Dennis, H. d'Hooge, D. E.Hubka, L. A. Hutchins, J. T. Montague, F. J. Pollak, M. P. Gifkins, iMAX: a multiprocessor operating system for an object-based computer, *Proc. of the height ACM Symp. on Operating Systems Principles*, Dec. 1981, pp. 127-136

[Khider 83]
A. Khider, *Protocoles de diffusion fiable pour réseaux locaux à diffusion*, thèse de l'Institut National Polytechnique de Grenoble, May. 1983

[Kowalski 90]
O. C. Kowalski, H. Härtig, Protection in the Birlix operating system, *Proc. Int. Conf. on Distributed Computing Systems*, May 1990, pp. 160-166

[Krakowiak 85]
S. Krakowiak, *principes des systèmes d'exploitation des ordinateurs*, Col. Dunod Informatique, Bordas, Paris 1985

[Krakowiak 90]
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*, vol. 3, n° 3, Sept.-Oct. 1990, pp

[Krakowiak 91]
S. Krakowiak, X. Rousset de Pina, Fault tolerant support in Guide, *Proc. Fourth ACM SIGOPS European Workshop*, Bologne, Sep. 1991

[Lacourte 91]
S. Lacourte, Exceptions in Guide, an objeect-oriented language for distributed applications, *European Conference on Object Oriented Programing*, Genève, Jul. 91, Lectue Notes in Computer Science, n° 512 Springer-Verlag, pp. 268-287

[Lacourte 92]
S. Lacourte, M. Riveill, Generic System Support for shared Objects Synchronization, *Proc. of 2nd International Workshop on Object Orientation in Operating Systems*, Dourdan, Sep. 92, pp. 153-157

[Laforgue 75]
P. Laforgue, *Construction de sous-systèmes utilisant une machine abstraite*, Thèse de docteur-ingénieur de l'Institut National Polytechnique de Grenoble, INPG, Feb. 1975

[Lampson 69]
B. W. Lampson, Dynamic protection structures, *Proc. AFIPS Fall Joint Computer Conf.*, 1969

[Laribi 91]
Y. Laribi, *Réalisation d'Application tolérante aux défaillance dans un système réparti orienté objets* , DEA d'informatique, INPG, Jun. 91

[Leach 83]
P. J. Leach, P. H. Levine, B. P. Douros, J. A. Hamilton, D. L. Nelson, B. L. Stumpf, The architecture of an integrated local network, *IEEE Journal on Selected Areas in Communication*, Nov. 83, pp. 842-856

[Leroudier 75]
J. Leroudier, Systèmes adaptatifs à mémoire virtuelle, Thèse d'état, Université J. Fourier (Grenoble-1), 1977

[Levy 90]
E. Levy, A. Silberschatz, Distributed File Systems: Concepts and examples, *ACM Comp. Survey*, vol. 22, N° 4, Dec. 1990, pp. 321-374

[Levy 84]
H. L. Levy, *Capabilities based Computer System*, Burlington: Digital Press, 1984

[Li 89]
K. Li, P. Hudak, Memory Coherence in Shared Virtual Memory Systems, *ACM Transactions on Computer Systems*, vol. 7, n° 4, Nov. 1989, pp. 321-359

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

[Maillot 78]
B. Maillot, A. Tarabout, I. Vatton, *Un outil pour la recherche en informatique*, thèse de Institut National Polytechnique de Grenoble, Dec 78

[McCue 90]
D. McCue, X. Rousset de Pina, *Tsync: Transaction-level Synchonisation Of Active Replicated Objects*, Internal notes of Arjuna Project, University of Newcastle upon Tyne, Sep. 1990

[McKeag 76]
R. M. McKeag, R. Wilson, *Studies in Operating Systems*, D. H. R. Huxtable ed., Academic Press, 1976, pp. 1-66

[Metcalfe 76]
R. M. Metcalfe, D. R. Boggs, Ethernet : Distributed packet switching for local computer network, *CACM*, vol 19, 7, Jul. 76, pp. 395 - 404

[Mishra 89]
S. Mishra, L. L. Peterson, R. D. Schichting, Implementating Fault-Tolerant replicated Objects Using Psynch, *Proc. 8th. Reliable Distributed Systems*, Seatle, Oct. 1989, pp. 42-52

[Morris 86]
J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, F. D. Smith, Andrew: a distributed personal computing environment, *Comm. of the ACM*, vol. 29, n° 3, Mar. 1986, pp 184- 201

[Mullender 86]
S. J. Mullender, A. S. Tananbaum, The design of a capability-based distributed operating system, *Computer Journal*, vol. 29, n° 8, Aug. 86, pp. 289-299

[Naffah 79]
N. Naffah, Projet Pilote Kayak - Description du réseau expérimental Danube, Rapport INRIA, Nov. 1979

[Needham 78]
R. Needham, M. Schroeder, Using encryption for authentication in large networks of computers, *Comm. ACM*, vol. 21, n° 12, Dec. 78, pp. 993-999

[Nguyen Van 91]
H. Nguyen Van, *Compilation et environnement d'exécution d'un langage à base d'objets*, Thèse de doctorat de l'Institut National Polytechnique de Grenoble, Feb. 91

[Nitzberg 91]
B. Nitzberg, V. Lo, Distributed Shared Memory: A survey of Issues and Algorithms, *Computer*, vol. 24, n° 8, Aug. 90, pp. 5-60

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

[Panzieri 88]
F. Panzieri, S. Shrivastava, Rajdoot: A Remote Procedure Call Mechanism Supporting Orphan Detection and Killing, *IEEE Transaction on Sofware Engineering*, vol. SE-14, n° 1, Jan 1988, pp. 30-37

[Parrington 90]
G. D. Parrington, Reliable Distributed Programming in C++: The Arjuna Approach, *Proc.of USENIX Conf.on C++*, San Francisco, April 90

[Popek 85]
G. Popek, B. J. Walter, *The Locus distributed system architecture*, MIT Press, 1985

[Pouzin 73]
L. Pouzin, Network architectures and components, *Proc. 1st. European Workshop on Comp. Networks*, Arles, IRIA ed., May 1973, pp 227-265

[Pouzin 82]
L. Pouzin (editor), *The cyclades computer network, Monograph of the Inter. Council for Computer Comm.*, Noth-Holland, 1982

[Quint 79]
V. Quint, N. Naffah, Protocole de transport pour réseaux locaux, *IRIA Projet-Pilote Kayak,* Rel.2.204.1, Feb 1979, 1-21

[Quint 86]
V. Quint, X. Rousset de Pina, *Edimath Version 3.0 - manuel d'utilisation*, Microsphère edit., Lyon 1986

[Presser 72]
L. Presser, J. R. White, Linkers and loaders, *ACM Computing Survey*, vol. 4, n° 3, 1972, pp 149-167

[Ramachandran 89]
U.Ramachandran, M. Ahamad, Y. A. Khalidi 89, Coherence of Distributed Shared Memory: Unifying Synchronization and data Transfer, *International Conference on Parallel Processing*, 1989

[Ritchie 74]
D. M. Ritchie, K. Thomson, The Unix time-sharing system, *Comm. ACM*, vol. 56, n° 6, Jul.1974, pp 365-375

[Riveill 92]
M. Riveill, X. Rousset de Pina, Reusable Synchronized Objects, *ECOOP Workshop on Object based Concurency and Reuse*, Ultrech, Jun. 92

[Riveill 93]
M. Riveill, *Langages et Systèmes pour des Applications Réparties*, mémoire d'habilitation de l'INPG, Grenoble, Jan. 1993

[Roisin 91]
C. Roisin, M. Santana, The observer: a tool for observing distributed applications, Rapport Technique 8-91, Bull- IMAG, 2 rue de Vignate - ZI de Mayencin - 38610 Gières - France, Jan. 91

[Rousset de Pina 71]
X. Rousset de Pina, Modèle Déterministe pour l'Etude de l'Efficacité d'un Système en Temps Partagé, *Revue Française d'Informatique et de Recherche Opérationnelle*, n° B-3, 1971, pp 109-121

[Rozier 88]
M. Rozier, V. Abrassimov, F. Armand, J. Boule, M. Gien, M. Guillemont, F. Herrmann, P. Leonard, S. Langlois, V. Neuhauser, Chorus Distributed Operating Systems, *Computing Systems*, vol. 1, n° 4, 1988, pp. 305-370

[Saltzer 74]
J. H. Saltzer, Protection and the Control of Information Sharing in Multics, *Comm. ACM*, vol. 17, n° 7, Jul. 1974, pp. 388-402

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

[Sandberg 85]
R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, design and implementation of the Sun Network Filesystem, *Proc. Summer 85 USENIX Conf.*, Portland OR, Jun. 1985, pp. 119-130

[Satyanarayanan 89]
M. Satyanarayanan, Integrating Security in a Large distributed System, *ACM Trans. on Computing Systems*, vol 7, n° 3, Aug. 1989, pp. 247-249

[Schroeder 72]
M. D. Schroeder, J. H. Saltzer, A hardware architecture for implementing protection rings, *Comm. ACM*, vol 15, n° 3, Mar. 1972, pp. 157-170

[Schuh 90]
D. Schuh, M. Carey, D. DeWitt, Persistence in E revisited-Implementation experiences, *Proc. of the 4th Int.Workshop on Persistent Objects Systems*, Martha's Vineyard, Sep. 1990, pp. 345-359

[Schultz 88]
B. Schultz, The evalution of ARPANET, *Datamation*, vol. 34, n° 15, Aug. 1988, pp. 61-67

[Scott 89]
M. L. Scott, T. J. Leblanc, B. D. Marsh, A Multi-User, Multi-Language Open Operating Sytem, *Proc. of the 2nd Workshop on Workstation Operating Systems (WWOS II)*, Pacific Grove, Sep. 1989, pp 15-129

[Scott 92]
M. L. Scott, W. Garrett, Shared Memory Ought to be Commonplace, *Proc. of the 3rd Workshop on Workstation Operating Systems (WWOS III)*, Key Biscayne, Apr. 1992

[Shapiro 91]
M. Shapiro, Gestions réparties d'objets, *Construction des systèmes d'exploitation répartis* (R. Balter, J. P. Banâtre et S. Krakowiak ed.), INRIA collection didactique, 1991

[Shrivastava 89]
S. Shrivastava, G. N. Dixon, G. D. Parrington, An overview of Arjuna: A Programming System for Reliable Distributed Computing, *Tech. Report Series University of Newcastle upon Tyne*, n° 298, Nov. 1989

[Siewiorek 83]
D. Siewiorek, C. Gordon Bell, A. Newell, *Computer Structures: Principles and Examples*, McGraw-Hill Advanced Computer Science Series, 1983

[Steiner 88]
J. G. Steiner, C. Neumann, J. I. Schiller, Kerberos: an authentification service for open network systems, *Proceedings of the USENIX Winter Conf.*, Dallas, Feb. 1988, pp. 9-12

[Sumner 62]
F. H. Sumner, G. Haley, E. C. Y. Chen, The Central Control Unit of the Atlas Computer, *Proc. IFIP Cong.*, 1962, pp. 657-662

[Tanenbaum 87]
A. S. Tanenbaum, *Operating Systems design and implementation*, Prentice-Hall, 1987

[Tanenbaum 89]
A. S. Tanenbaum, *Computer Networks*, 2nd ed, Prentice-Hall, 1989

[Tanenbaum 90]
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, *Comm. of the ACM*, vol. 33, 12, Dec. 1990, pp. 46-64

[Thomas 73]
R. H. Thomas, A ressource sharing executive for the ARPANET, *AFIPS Conf. Proc. 42*, 1973, pp. 159-163

[Thacker 83]
C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, D. R. Boggs, Alto: A Personal Computer, *in [Siewiorek 83]*, Chap. 33, pp 549-572

[Wegner 90]
P. Wegner, Concepts and paradigms of Object-Oriented Programming, *OOPS Messenger (ACM Press)*, vol. 1, n° 1, Aug. 1990, pp.8-87

[Weinstein 85]
M. J. Weinstein, T. W. Jnr. Paje, B. K. Livezey, G. J. Popeck, Transaction and synchronization in a distributed operating system, *Proc. of the 10th Symposium on Operating Systems Principles*, Orcas Island, Dec. 1985

[Wilkes 92]
M. V. Wilkes, The long-term future of Operating Systems, *Comm. of the ACM*, vol.35, n° 11, Sep. 92

[Wulf 74]
W. A. Wulf & al, Hydra: the kernel of a multiprocessor operating system, *Comm. ACM*, vol.17, n° 6, Jun. 1974