THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR ingénieur et 3ème cycle

Spécialité : GENIE INFORMATIQUE

par

Bernard MAILLOT
Alain TARABOUT
Irène VATTON

UN OUTIL DE RECHERCHE EN SYSTEMES INFORMATIQUES

Thèse soutenue le 11 décembre 1978 devant la Commission d'Examen

Président S. KRAKOWIAK

Examinateurs R. BALTER
J. BRIAT
C. KAISER
J. MOSSIERE
J.P. VERJUS

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1977-1978

Président : M. Philippe TRAYNARD

Vice-présidents : M. René PAUTHENET
M. Georges LESPINARD

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Electronique - automatique
	BESSON Jean	Chimie minérale
	BLOCH Daniel	Physique du solide - cristallographie
	BONNETAIN Lucien	Génie chimique
	BONNIER Etienne	Métallurgie
*	BOUDOURIS Georges	Electronique - automatique
	BRISSONNEAU Pierre	Physique du solide - cristallographie
	BUYLE-BODIN Maurice	Electronique - automatique
	COUMES André	Electronique - automatique
	DURAND Francis	Métallurgie
	FELICI Noël	Electronique - automatique
	FOULARD Claude	Electronique - automatique
	LANCIA Roland	Electronique - automatique
	LONGEQUEUE Jean-Pierre	Physique nucléaire corpusculaire
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie - physique
	PAUTHENET René	Electronique - automatique
	PERRET René	Electronique - automatique
	POLOUJADOFF Michel	Electronique - automatique
	TRAYNARD Philippe	Chimie - physique
	VEILLON Gérard	Informatique fondamentale et appliquée

* en congé pour études

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuël	Electronique - automatique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electronique - automatique
	GUYOT Pierre	Métallurgie physique
	LACOUME Jean-Louis	Electronique - automatique
	JOUBERT Jean-Claude	Physique du solide - cristallographie

.../...

MM.	ROBERT André	Chimie appliquée et des matériaux
	ROBERT François	Analyse numérique
	ZADWORNY François	Electronique - automatique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	CHARTIER Germain	Electronique - automatique
	CHIAVERINA Jean	Biologie, biochimie, agronomie
	IVANES Marcel	Electronique - automatique
	LESIEUR Marcel	Mécanique
	MORET Roger	Physique nucléaire - corpusculaire
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie Physique

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahim	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	BISCONDI Michel	Ecole des Mines St. Etienne (dépt. Métallurgie)
	BOOS Jean-Yves	Ecole des Mines St. Etienne (Métallurgie)
	DRIVER Julian	Ecole des Mines St. Etienne (Métallurgie)

.../...

MM.	KOBYLANSKI André	Ecole des Mines St. Etienne (Métallurgie)
	LE COZE Jean	Ecole des Mines St. Etienne (Métallurgie)
	LESBATS Pierre	Ecole des Mines St. Etienne (Métallurgie)
	LEVY Jacques	Ecole des Mines St. Etienne (Métallurgie)
	RIEU Jean	Ecole des Mines St. Etienne (Métallurgie)
	SAINFORT	C.E.N. Grenoble (Métallurgie)
	SOUQUET	U.S.M.G.
	CAILLET Marcel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	COULON Michel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	GUILHOT Bernard	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	LALAUZE René	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	LANCELOT Francis	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	SARRAZIN Pierre	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	SOUSTELLE Michel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	THEVENOT François	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	THOMAS Gérard	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	TOUZAIN Philippe	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	TRAN MINH Canh	Ecole des Mines St. Etienne (Chim. Min. Ph.)

E.N.S.E.R.G.

MM.	BOREL	Centre d'études nucléaires de Grenoble
	KAMARINOS	Centre national recherche scientifique

E.N.S.E.G.P.

M.	BORNARD	Centre national recherche scientifique
Mme	CHERUY	Centre national recherche scientifique
MM.	DAVID	Centre national recherche scientifique
	DESCHIZEAUX	Centre national recherche scientifique

Nous remercions S. KRAKOWIAK d'avoir accepté la présidence du jury et C. KAISER de nous avoir aidés lors de la rédaction de ce document par ses remarques judicieuses.

Nous remercions tout particulièrement J.P. VERJUS dont la compétence et le dynamisme ont bien souvent été déterminants au cours de la vie mouvementée du projet OURS, dont il a assuré la responsabilité. Depuis lors, il n'a jamais ménagé son temps ni son soutien pour voir aboutir cette thèse, et nous lui en sommes redevables.

J. MOSSIERE nous a toujours encouragés de manière discrète et amicale. Il s'est chargé de la tâche ingrate qui consistait à relire les premières versions manuscrites de cette thèse ; ses remarques nous ont été précieuses.

Nous remercions également R. BALTER, à la bonne humeur contagieuse, d'avoir accepté de faire partie du jury.

Cette thèse est une thèse de groupe au sens plein du terme : nous en devons la matière à J. BRIAT, responsable scientifique du projet OURS, qui a joué un rôle moteur pendant toute sa durée ; et c'est toute l'équipe qui a réalisé ce que nous allons présenter : V. BEKKERS, J. ESTUBLIER, P. LAFORGUE, B. MAILLOT, G. MENARD, M. PEQUIGNOT, J. RAYMOND, X. ROUSSET de PINA, S. ROUVEYROL, A. TARABOUT, I. VATTON.

F. BLANC et M.J. DOREL ont réalisé avec soin et rapidité la dactylographie de cette thèse. Qu'elles en soient remerciées, ainsi que D. IGLESIAS dont l'équipe a tiré les épreuves avec sa compétence habituelle.

SOMMAIRE

CHAPITRE I

INTRODUCTION

I.1. HISTORIQUE

L'outil que nous allons présenter est l'aboutissement du projet d'OUtil de Recherche en Système (OURS) lancé en Automne 1975 à Grenoble.

A cette date, le projet avait deux objectifs : en premier lieu, il devait constituer un outil pour des utilisateurs de l'ordinateur CII IRIS 80 du Centre Interuniversitaire de Calcul de Grenoble (CICG) ; en second lieu, grâce à l'utilisation espérée, il devait permettre d'évaluer un certain nombre de concepts nouveaux sur l'architecture de systèmes modulaires et extensibles. L'Institut de Recherche d'Informatique et d'Automatique (IRIA), ainsi que le Centre National d'Etude des Télécommunications (CNET) se sont aussi intéressés au projet et ont largement participé à son financement. Le projet a hérité en Automne 1976 d'un troisième objectif : utiliser et donc valider le Langage d'Implémentation de Systèmes, LIS [DEL, BRI2].

I.2. LES OBJECTIFS

Le but du projet est de définir un noyau de système extensible. Ce noyau doit permettre le développement d'une grande variété d'applications : Bases de données, Système conversationnel, transactionnel, ..., en particulier d'un système conversationnel pour les utilisateurs du Centre Interuniversitaire de Calcul de Grenoble.

Ces perspectives de développement du noyau de base ont fait ressortir deux objectifs essentiels pour notre étude :

- offrir un mécanisme d'extension efficace pour construire sur le noyau des niveaux d'interface plus sophistiqués ;
- offrir un niveau d'interface minimal, d'utilisation plus agréable que celui de la machine hôte et adapté à tout type d'applications : "un noyau universel".

I.2.1. Le choix d'une structure

L'équipe de développement de systèmes de Grenoble avait précédemment défini et réalisé un noyau de système : GEMAU [GEM, GUI] ainsi qu'un sous-système conversationnel [LAF]. La méthode de construction de systèmes par abstraction-raffinement [DIJ] a conduit à découper ces systèmes en modules communiquant par appel et retour procédural [MOS, DAR]. L'observation d'un tel système fait apparaître une carence : les processus sont toujours bloqués au niveau le plus élémentaire (Exemple : Entrées/ Sorties), c'est-à-dire avec un maximum d'imbrication de procédures, donc de contextes empilés. Nous avons donc remis en cause le choix d'une structure procédurale. D'une part une étude menée dans le cadre du projet [BRI5, EST1] a montré qu'une structuration du système en réseau de processus communiquant par messages permet dans certains cas un gain de place en mémoire appréciable et parfois un gain de temps à l'exécution.

D'autre part, le développement des réseaux de mini ou micro-processeurs travaillant en parallèle fournit une structure comparable, à savoir un ensemble d'entités communiquant par messages.

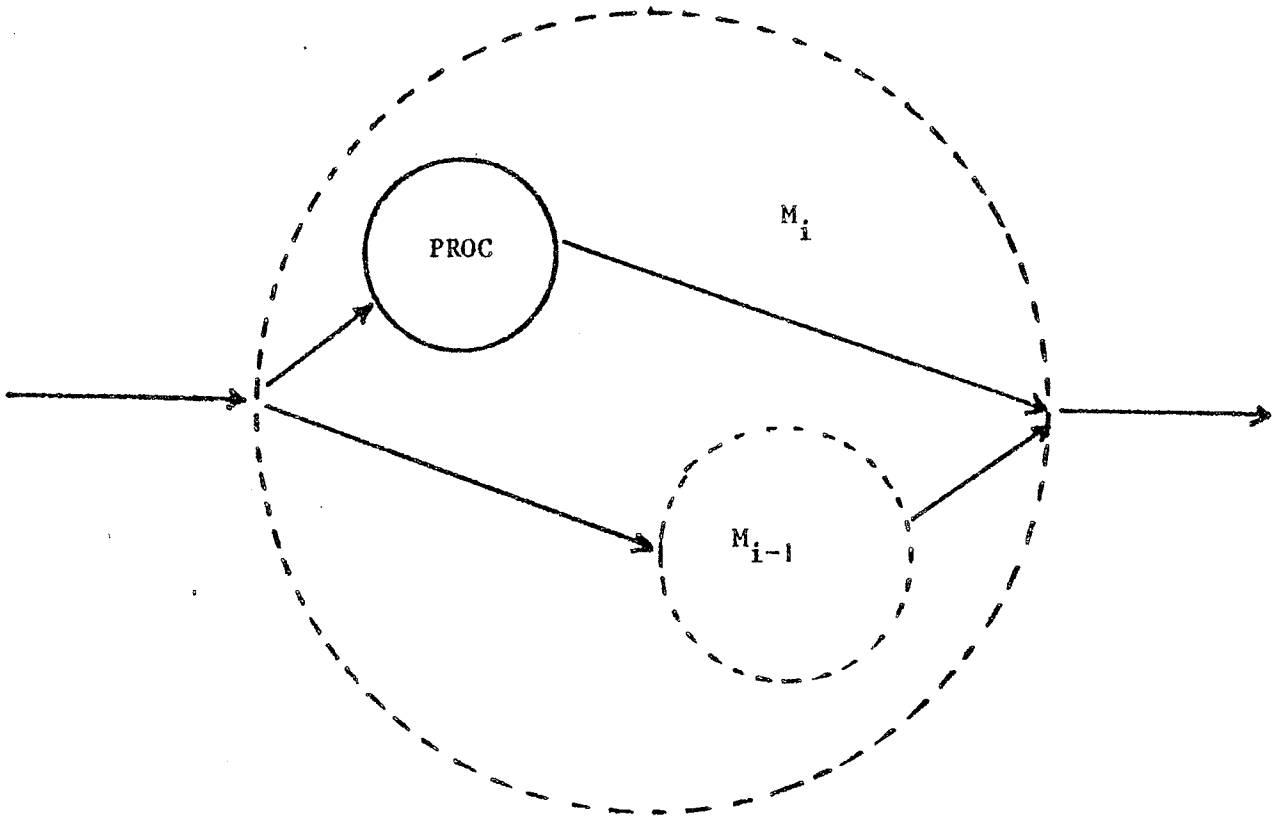
Nous avons alors essayé d'appliquer la méthode de construction par abstraction/raffinement à un système fonctionnellement distribué sur un réseau de mini ou micro-processeurs. Notre démarche se rapproche des études antérieures ou parallèles des réseaux de processus du même type [WUL, BRIN, JAM]. Cependant, les mécanismes de communication inter-processus de ces études diffèrent sensiblement du nôtre : ils ne respectent pas les règles de communication établies par la construction progressive de niveaux d'abstraction.

Comme un calculateur, un système peut être vu comme un ensemble de processus opérationnels (additionneur, multiplieur, ... dans un calculateur) associés à un contrôleur chargé de décoder et d'aiguiller les messages vers les processus.

Vu de l'extérieur, le système et le calculateur deviennent des entités ou machines abstraites capables d'interpréter de nouveaux processus. En regroupant processus et machines abstraites, on peut construire de proche en proche de nouveaux niveaux de machines abstraites i, i+1, ...

Dans [JAM], le mécanisme de communication présenté permet d'obtenir des niveaux d'interprétation semblables, mais il n'offre pas un langage d'expression des connexions : les connexions sont établies dynamiquement. Ce qui implique que deux processus, quel que soit leur niveau dans la structure, doivent se connaître et se mettre d'accord pour ouvrir une voie de communication. Le rôle du contrôleur d'une machine abstraite de niveau i est alors de gérer ces voies de communication et de résoudre les conflits possibles entre les communications inter-processus du même niveau i et les communications avec les processus des niveaux supérieurs i+1, i+2, ...

Dans notre cas, la notion de machine abstraite est reconnue par le mécanisme de communication. La création d'une machine abstraite prédéfinit des voies de communication pour l'échange des messages entre les entités de la machine. Le cheminement de chaque message entrant dans la machine est déductible sans ambiguïté des informations de contrôle qu'il contient et des connexions prédéfinies pour la machine. On peut alors figurer une machine abstraite de niveau i : M_i comme une bulle enfermant un réseau orienté de processus et de machines abstraites de niveaux inférieurs :



I.2.2. Le choix d'un niveau de machine

Si la notion de machine abstraite est généralement admise, la définition du niveau d'interface qu'elle doit offrir reste discutée.

La définition des noyaux de système existants résulte du découpage en niveaux d'abstractions [ZUR]. Notre but est de définir le plus grand niveau d'abstraction commun aux différents systèmes informatiques (base de données [FOR], système transactionnel, ...) et aux noyaux de systèmes existant sur des machines analogues (ESOPE [KAI, KRA], MULTICS [ORG, VYS], GEMAU [GEM, GUI], PLESSEY 250 [ENG, FER], SAR [KER1, KER2] et HYDRA [WUL]).

Les principales caractéristiques que nous avons retenues sont :

- Une grande mémoire uniforme :

Un tel outil général doit obligatoirement fournir de grands espaces virtuels constructibles à partir d'unités d'information. Ces unités d'information doivent pouvoir être conservées pour supporter des fichiers ou bien des bases de données. L'entité manipulée par le système développé sur cet outil pouvant être plus grande ou plus petite que l'unité d'information que nous offrons, leur protection et leur partage ne peuvent être définis qu'au niveau de ce système.

Nous ne développerons donc que des outils permettant la mise en oeuvre du partage et de la protection : nom unique d'objets, protection d'accès.

- Des périphériques standard :

La gestion des périphériques doit aussi être d'un niveau suffisamment élémentaire pour permettre le développement de tout type d'allocation de périphériques et de protocoles d'Entrées/Sorties. On a essayé de définir des objets périphériques malgré tout plus maniables que les organes périphériques utilisés, d'où une définition de périphérique standard.

- Une allocation de ressources paramétrable :

De la même façon, on laissera aux systèmes développés sur notre noyau la possibilité de définir leur propre politique d'allocation des ressources. Nous avons repris là l'idée présentée par HYDRA [WUL] qui consiste à séparer les mécanismes d'allocation fournis par le noyau des politiques d'allocation développés par les systèmes.

I.3. LES RESULTATS ESCOMPTES

Les méthodes et outils de découpage et de construction de systèmes informatiques qui vont être présentés dans cette thèse ont été utilisés pour la construction d'un noyau de système répondant aux caractéristiques définies au paragraphe I.2.2.

L'annonce de l'abandon de l'IRIS 80 par le Centre Interuniversitaire de Calcul de Grenoble a entraîné l'abandon, avant terme, du premier objectif (cf. § I.1) de notre projet : la réalisation d'un système d'exploitation conversationnel. Cette perte d'utilisateurs nous a, du même coup, privés des moyens d'une évaluation quantitative de notre noyau.

Il n'en reste pas moins qu'une appréciation qualitative de la valeur des outils que nous avons réalisés est possible du simple fait qu'ils ont été utilisés. L'expérience que nous avons acquise dans le domaine de la conception et de la réalisation de systèmes informatiques peut être utilisée par d'autrés. L'objet de cette thèse est précisément de faire le point sur notre acquit dans ce domaine, afin qu'il ne soit pas perdu.

Nous pensons que les chercheurs qui travaillent actuellement à la définition d'outils informatiques pour l'élaboration de systèmes répartis sur des architectures multiprocesseurs peuvent trouver une source d'inspiration dans ce qui va être maintenant présenté.

I.4. PLAN DE LA THESE

Cette thèse se divise en deux grandes parties suivies d'une conclusion. Chaque partie est composée de deux chapitres, le premier étant axé sur la présentation de concepts, d'outils et de mécanismes dont l'usage est illustré dans le chapitre qui suit.

La première partie (Chapitres II et III) est consacrée au modèle de décomposition que nous avons adopté.

Le Chapitre II expose les principes du modèle et par quels moyens il deviendra possible de construire des machines abstraites à partir d'éléments de base et de mécanismes d'extension. L'aspect dynamique des communications entre éléments et du nécessaire contrôle des phases d'un traitement à l'intérieur d'une machine abstraite sera exposé. L'évaluation qualitative du modèle sera assez détaillée.

Le Chapitre III donnera, à titre d'exemple, la structure d'une machine à fichiers (c'est-à-dire d'un système élémentaire de gestion de fichiers), montrant ainsi quel parti on peut tirer à la fois de l'existence d'un noyau de système et des outils de construction déjà décrits.

La deuxième partie (Chapitres IV et V) est consacrée au noyau de système MAS.

Le Chapitre IV présente les objets mis en oeuvre par la machine MAS et l'utilisation qui peut en être faite par des utilisateurs programmant sur ce noyau. On ne rentrera pas, à ce niveau, dans des détails de réalisation. Les algorithmes principaux seront évoqués de manière générale, et situés dans le cadre de la littérature technique actuellement disponible sur ce sujet.

Le Chapitre V de la deuxième partie montre comment les mécanismes présentés au Chapitre II ont été utilisés pour construire la machine abstraite que nous avons réalisée : le noyau MAS.

La conclusion finale (Chapitre VI) fait un bref bilan du projet et évoque quelques prolongements possibles de ce travail, à partir de l'apport qu'il représente.

Chapitre II

Le Modèle de Décomposition

Une des tendances actuelles de l'architecture des ordinateurs due à l'apparition et à l'installation des microprocesseurs sur le marché est l'éclatement des Systèmes Informatiques en processeurs spécialisés. Ils se présentent alors comme une fédération de processeurs indépendants qui communiquent entre eux par échange de messages [ANC1].

Partant de cette réalité, nous avons voulu développer un outil qui permette de construire et d'évaluer des architectures de machines et de systèmes aussi bien que les éléments constitutifs de ces architectures :

- processeurs spécialisés, bus de communication, mémoires locales et communes, ... pour ce qui concerne les machines,
- mécanisme de pagination, procédures de transmission, méthodes d'accès fichiers, compilateurs, ... pour ce qui concerne les systèmes.

Dans notre projet, ces éléments sont tous logiciels puisque leur support de réalisation est un IRIS 80 biprocesseur et non pas un atelier de micro-informatique. C'est un état de fait et nous admettrons au chapitre de ses inconvénients que la tentation peut être grande de "tricher", au nom de l'efficacité, en écrivant des gros paquets de code qui n'utilisent pas les mécanismes de décomposition standard qui sont offerts.

Mais une des ambitions du projet est aussi de montrer que l'architecture d'un système d'exploitation peut être définie dans les mêmes termes que celle d'une machine et que, ce faisant, il est plus facile d'insérer des points de mesure et de régulation de débit dans le système.

Dans ce chapitre, nous essaierons donc de faire le lien entre le modèle de décomposition que nous aurons décrit et les études qui ont été faites sur les méthodes d'écriture des systèmes d'exploitation [MOS], notamment celles qui développent la notion de module [LUC].

II.1. PRINCIPES DU MODELE

La résolution d'un problème par la définition d'une machine abstraite est une approche bien connue dans le domaine des langages de programmation [DIJ]. La méthode du raffinement consiste d'autre part à exprimer la réalisation d'une machine abstraite en termes d'autres machines abstraites. Nous avons voulu exploiter la méthode d'abstraction-raffinement et fournir au concepteur de système un moyen commode de la mettre en oeuvre.

Si l'on considère d'un point de vue externe le fonctionnement d'un système d'exploitation, on constate qu'un certain nombre d'activités se déroulent en parallèle et de façon apparemment indépendante :

- lecture de cartes sur un ou plusieurs lecteurs
- impression de listings sur une ou plusieurs imprimantes
- dialogue homme-machine sur chaque terminal en cours d'utilisation.

Dans la plupart des systèmes d'exploitation, un processus est associé à chacune de ces activités externes, et nous pensons que ces processus-là auront toujours leur raison d'être. Si l'on considère plus finement le déroulement de ces processus en regardant comment sont mis en oeuvre les différents programmes qui forment le système d'exploitation, on constate deux types d'organisation :

- une organisation du type libre-service dans lequel le processus exécute lui-même toutes les actions nécessaires, quel que soit leur niveau, y compris les opérations de contrôle et de protection du code et des données qui ont lieu généralement chaque fois qu'ils rentrent dans une nouvelle couche de logiciel. Les processus externes sont alors les seuls processus du système ;

- une organisation du type "hôtel 4 étoiles" dans lequel une armée de processus "serveurs" est à la disposition des processus externes. Chaque processus serveur est en général chargé d'exécuter un programme, c'est-à-dire d'effectuer un service, particulier. Les processus serveurs ne sont en général pas tous au même niveau et l'existence de plusieurs couches de logiciel entraîne parfois une hiérarchie entre certains processus serveurs.

Nous ne ferons pas ici une étude comparative des avantages et des inconvénients respectifs de ces deux types d'organisation, car cette étude a déjà été faite par J. Estublier [EST] dans le cadre de notre projet commun.

Compte tenu du développement des microprocesseurs et du foisonnement des projets de systèmes multi-microprocesseurs [MAZ] et considérant les objectifs énoncés en tête de ce chapitre, notre choix s'est naturellement porté sur un mécanisme permettant de construire un système à partir de processus serveurs spécialisés, suivant donc une organisation du deuxième type.
Mais nous pensons qu'à l'intérieur d'un système, il est souhaitable de pouvoir faire exécuter certains groupes de services par le processus demandeur lui-même : on a alors coexistence entre les deux types d'organisation cités plus haut. Nous verrons que cette coexistence est permise par notre modèle de décomposition.
L'élément de base du modèle de décomposition que nous proposons est donc le processus serveur. La construction d'une machine abstraite s'effectue par création puis assemblage d'un certain nombre de processus serveurs communiquant par messages.

Chaque processus serveur est chargé d'exécuter une ou plusieurs instructions de la machine abstraite, mais certaines instructions peuvent nécessiter la coopération de plusieurs processus serveurs.
D'autre part, une même instruction peut apparaître dans le répertoire de plusieurs machines abstraites, définies indépendamment les unes des autres.

Ces différentes considérations nous ont amenés à définir un mécanisme de connexion entre les processus serveurs qui présente à la fois un aspeçt statique : compilation de la structure de chaque machine abstraite lors de sa création, et un aspect dynamique lors de l'exécution d'un programme par une machine abstraite : c'est l'interprétation des messages par lesquels les processus serveurs communiquent.

Le plan que nous allons suivre pour présenter le modèle de décomposition est le suivant :

- Dans un premier temps, nous décrirons les processus serveurs primitifs à partir desquels toutes les machines abstraites pourront être construites. Nous insisterons en particulier sur la nécessité d'avoir pour tous les processus serveurs un interface externe défini selon une norme commune.
Puis nous introduirons la notion d'unité en remplacement de celle de processus serveur.

- Au paragraphe suivant, nous présenterons le mécanisme de connexion sous ses deux aspects et nous introduirons la notion de machine comme regroupement d'unités. Nous montrerons également de quelle façon le mécanisme de connexion permet d'inclure une machine dans une autre machine.

- Dans un troisième paragraphe, nous étendrons la notion de processus aux exécutions de programmes sur les machines construites et nous en profiterons pour expliquer, sinon justifier, une des limites de notre modèle.

- Puis nous enrichirons le mécanisme de connexion afin que tout processus puisse devenir un processus serveur. Le modèle de construction de machine devient alors itératif et conduit à la définition de niveaux de machines abstraites.

II.2. PRESENTATION DU MODELE

II.2.1. Les processus serveurs primitifs

Comme nous l'avons dit précédemment, le support de réalisation du projet OURS est un IRIS 80 biprocesseur. Nous y avons développé en premier lieu un noyau de multiprogrammation assez élémentaire que nous appelons HARDEXT (HARDware EXTension). Ce noyau assure le multiplexage des deux processeurs IRIS en un certain nombre de processus dits "primitifs". Nous nous intéressons ici à la façon dont ces processus peuvent communiquer. Le partage des deux processeurs IRIS sera examiné en détail au paragraphe IV.4.

Avant de parler de communication, nous allons présenter brièvement les caractéristiques communes de ces processus primitifs.

II.2.11. Contexte IRIS

Les processus primitifs s'exécutent en mode esclave, avec le mode d'adressage C de l'IRIS 80. Précisons tout de suite qu'un processus donné n'est jamais exécuté simultanément par les deux processeurs IRIS.

Pour le noyau HARDEXT, chaque processus est caractérisé par les données d'état qui décrivent l'utilisation qu'il fait d'un processeur IRIS. Ces données sont les suivantes :

- valeurs de 16 registres généraux
- valeurs de 8 registres de base
- valeurs de 4 mots d'état programme.

Le noyau HARDEXT est, comme nous l'avons dit, assez élémentaire et un espace virtuel unique est prédéfini au chargement, pour tous les processus primitifs. En outre, les programmes associés à ces processus et certaines données de ces programmes sont résidents et implantés à des adresses fixes dans cet espace virtuel.

II.2.12. Communication entre les processus primitifs

La possession d'un espace d'adressage commun peut constituer pour les processus primitifs un premier moyen de se communiquer des informations. Nous avons cependant préféré développer un outil de communication plus général, c'est-à-dire ne présupposant pas le partage d'une mémoire commune.

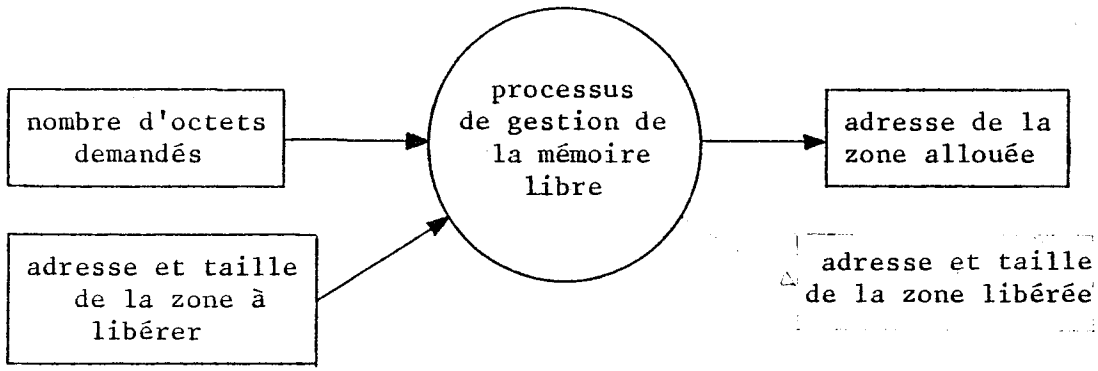
L'outil de communication que nous avons défini se compose de deux primitives ATTEND et RENVOI ; la première signifie : "je n'ai rien à faire, j'attends du travail" ; la seconde veut dire : "j'ai fini un travail que j'avais à faire, voici le résultat". Un mécanisme de transfert de messages est associé à cet outil afin que dans les deux primitives une information puisse être échangée entre le processus et le noyau. L'échange d'information entre le noyau et le processus est effectué par recopie d'une zone de mémoire selon une norme de programmation précise.

La primitive ATTEND est bloquante : le processus qui l'a appelée n'est réactivé que s'il y a un message pour lui.

La primitive RENVOI n'est pas bloquante : le processus est relancé immédiatement ; il peut alors s'exécuter pour son propre compte ou continuer un travail qu'il avait mis de côté, ou encore demander un nouveau message par la primitive ATTEND.

La définition de ces deux primitives appelle un certain nombre de précisions ou de remarques :

a) Chaque processus a la possibilité de multiplexer les travaux qui lui sont fournis. Par exemple, un processus primitif peut être spécialement chargé de gérer une zone de l'espace virtuel constituant une réserve commune de mémoire libre.



Dans cet exemple, le processus de gestion mémoire peut être amené à stocker provisoirement plusieurs demandes d'allocation jusqu'à ce qu'une demande de libération de mémoire lui parvienne. Il déroule donc l'algorithme suivant :

```
Cycle : ATTEND ;
        cas message dans
            demande : si bloc libre alors
                          allouer-bloc ;
                          RENVOI ;
                      sinon
                          stocker la demande ;
                      fin
```

```
        libération ; libérer bloc ;
                     RENVOI ;
                  tantque demande-en-stock et bloc-libre faire
                       allouer-bloc ;
                       RENVOI ;
                  fin
            fin
      fincycle
```

b) Chaque processus est spécialisé dans l'exécution d'une certaine fonction ou d'un certain nombre de fonctions bien déterminées. Il s'attend donc à recevoir certains types de messages bien particuliers. D'autre part ce processus est susceptible de fabriquer et d'émettre, par la primitive RENVOI, d'autres types de messages tout aussi particuliers.

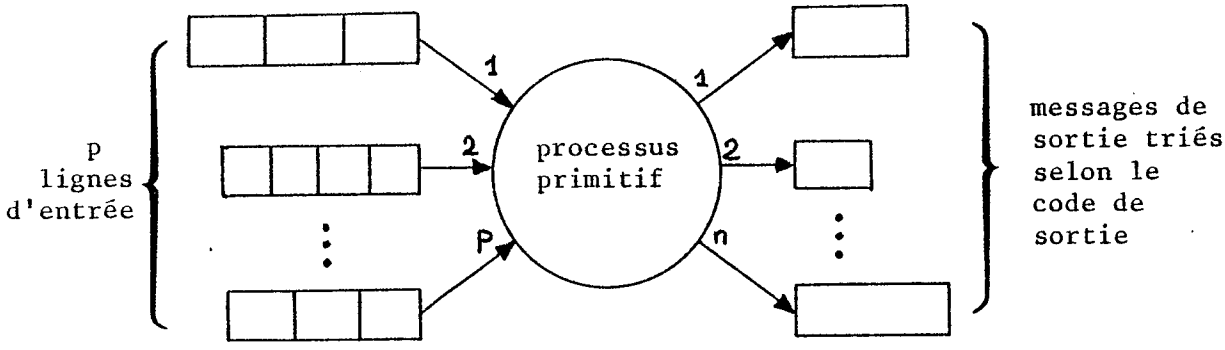
Nous avons adjoint aux primitives ATTEND et RENVOI un mécanisme de contrôle, que nous serions tentés d'appeler contrôle de type mais qui n'en est qu'une ébauche. Ce mécanisme est le suivant :

- A chaque processus sont associées en entrée un certain nombre de files d'attente que nous appelons lignes, et sur lesquelles arrivent les messages. Le processus peut se mettre en attente de façon préférentielle sur une de ces files, mais si elle ne contient pas de message, le noyau l'aiguille sur la file qui contient le message le plus ancien.

- Chaque message émis par un processus est accompagné d'un code de sortie, et à chaque processus sont associés un certain nombre de codes de sortie possibles.

Nous reviendrons en détail au paragraphe II.42 sur l'utilisation que le noyau fait de ces mécanismes pour contrôler le bon fonctionnement des machines.

Munis de ces mécanismes les processus primitifs se comportent en définitive comme autant de boîtes noires définies fonctionnellement par des pattes d'entrée (numéros de lignes) et des pattes de sortie (codes de sortie).



Créer une machine abstraite à partir des processus primitifs revient donc à interconnecter ces processus. Mais avant d'étudier cette connexion, nous allons enrichir la notion de processus serveur en introduisant la notion d'unité de traitement.

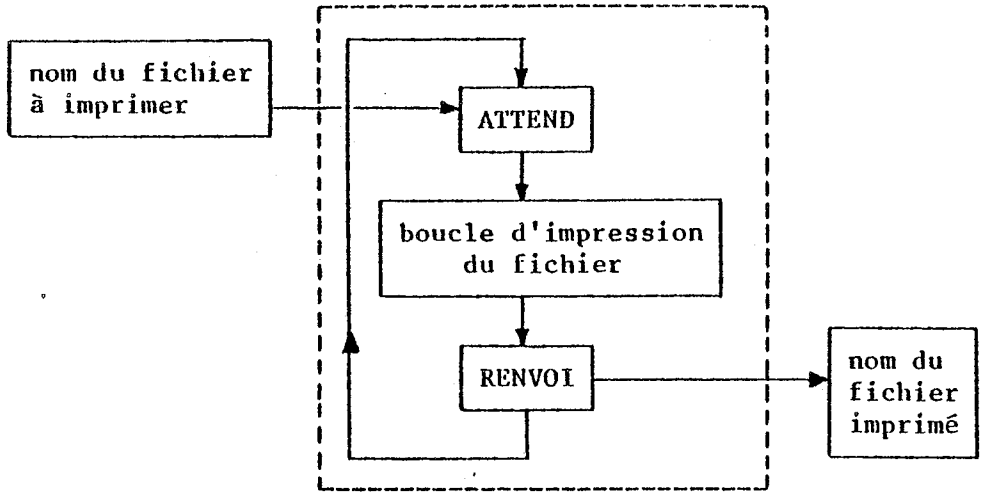
II.2.13. Les unités

Dans l'exemple précédent de la gestion de mémoire, le processus primitif qui réalise cette gestion n'a pas à se préoccuper de conflits d'accès à la réserve de mémoire libre puisqu'il est seul à prendre et à remettre des blocs dans cette réserve. Tel que nous l'avons écrit, le programme qui définit la gestion de cette réserve doit être déroulé en section critique, donc par un seul processus.

Mais il peut être intéressant de faire dérouler par plusieurs processus primitifs le programme associé à une boîte noire ; c'est notamment le cas lorsque ce programme gère un type de ressources dont il existe plusieurs exemplaires et que le traitement est long. Il est alors commode de créer autant de processus qu'il y a d'exemplaires de la ressource, tous ces processus déroulant le même programme.

Exemple

Soit un symbiont de sortie chargé d'éditer des fichiers "listings" sur une imprimante. Ce symbiont peut être réalisé par un processus primitif déroulant le programme suivant :



Le numéro de l'imprimante gérée par ce programme fait partie du contexte du processus primitif qui l'exécute. Dans le cas où il y a plusieurs imprimantes, on crée donc un processus primitif par imprimante de sorte que les imprimantes peuvent débiter simultanément. Mais si les imprimantes sont identiques et banalisées, la meilleure façon de répartir le travail entre elles est que les processus associés puisent dans une file d'attente commune.
Fin de l'exemple.

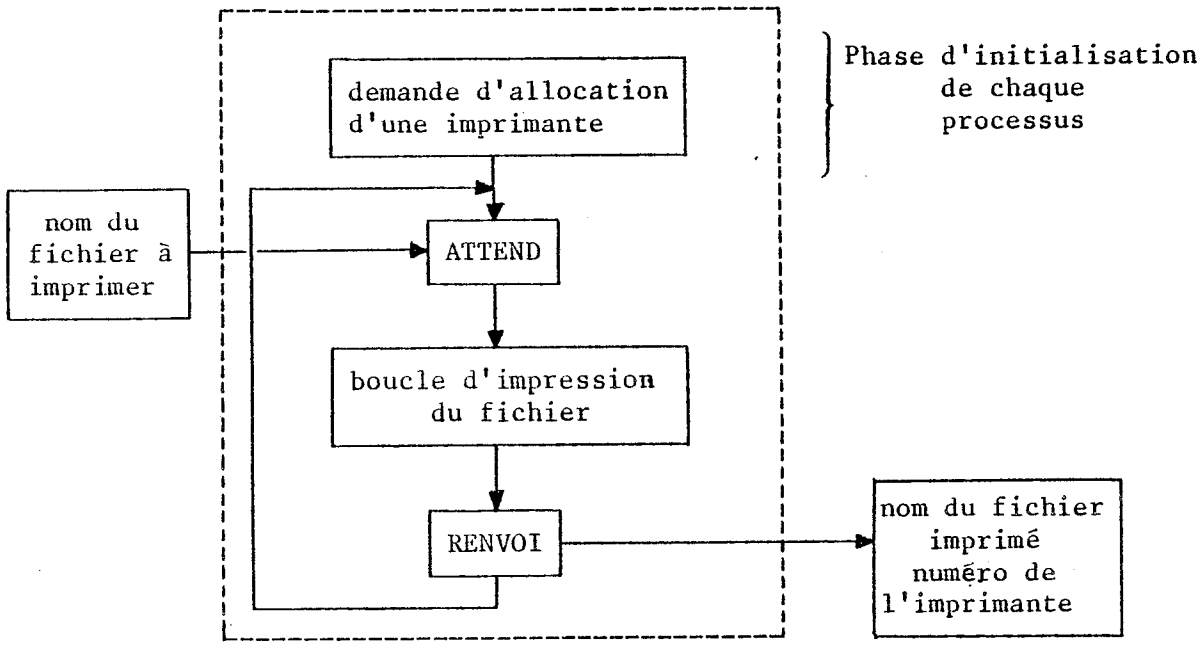
Nous définissons une unité comme étant le regroupement de plusieurs processus primitifs partageant les mêmes pattes d'entrée. Nous appelons degré de multiplication d'une unité le nombre de processus regroupés dans cette unité.

Chaque processus primitif dispose des primitives ATTEND et RENVOI additionnées du mécanisme de choix préférentiel associé à la primitive ATTEND, mais chaque processus doit s'attendre à recevoir tous les types de messages possibles qui arrivent sur l'unité, et doit donc être capable de les traiter : il n'y a pas a priori de processus spécialisé et nous admettrons donc qu'ils exécutent le même programme et qu'ils ont les mêmes pattes de sortie. Les pattes de sortie d'une unité sont définies par celles des processus qui constituent cette unité, plus précisément deux processus émettant le même code de sortie déterminent une seule patte de sortie, qui est définie par ce code.

Bien que tous les processus d'une unité soient définis par le même contexte IRIS initial, chaque processus possède ensuite un contexte IRIS qui lui est propre. Généralement d'ailleurs le contexte IRIS initial définit un programme composé d'une phase d'initialisation de l'unité, puis d'une phase d'initialisation du processus puis de la boucle ATTEND - traiter - RENVOI. La phase d'initialisation de l'unité n'est exécutée qu'une fois, par le premier processus qui déroule le programme.

La création d'une unité s'effectue par la primitive suivante :
CREER UNITE (contexte IRIS initial, degré de multiplication, nombre de pattes d'entrée, nombre de pattes de sortie).
L'identité de l'unité créée est fournie en retour.
C'est le noyau qui se charge de créer les processus primitifs à partir de la description de l'unité.
Ces processus sont donc créés identiques au départ puisque définis de la même façon par le contexte IRIS initial. Ils sont créés en nombre égal au degré de multiplication fourni dans la primitive.

Reprenons alors l'exemple précédent : le symbiont de sortie peut être réalisé par une unité dont le degré de multiplication est égal au nombre d'imprimantes à gérer, et dont le contexte IRIS définit le programme suivant :



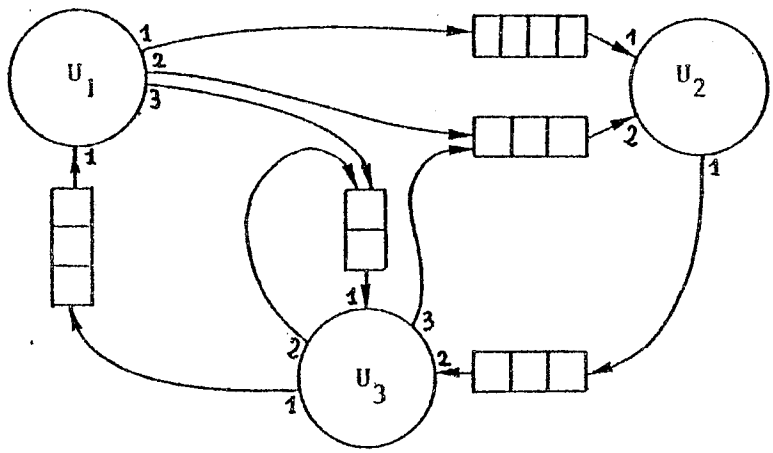
Unité symbiont

Créer une machine abstraite à partir des boîtes noires que sont les unités revient en définitive à interconnecter ces unités. C'est cette connexion que nous allons étudier maintenant.

II.2.2. Le mécanisme de connexion

Le problème paraît simple a priori : il suffit de connecter chaque patte de sortie de chaque unité à une patte d'entrée d'une autre unité (ou éventuellement de la même).

Exemple



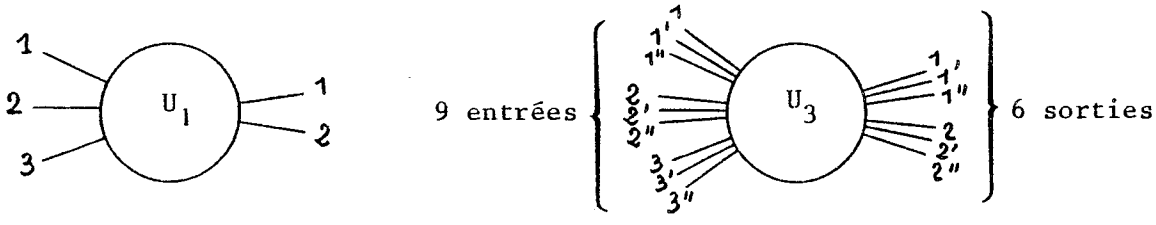
Ce type de connexion peut être défini statiquement et/ou dynamiquement :

a) Connexion statique

Comme cela est illustré sur le schéma précédent, chaque patte de sortie ne peut être connectée qu'à une seule patte d'entrée, donc à une seule unité. On voit tout de suite les conséquences de cette solution en reprenant l'exemple du processus de gestion de la mémoire (nous admettrons que ce processus est une unité) : en effet la patte de sortie, qui véhicule les adresses des zones allouées, doit pouvoir être connectée à n'importe quelle autre unité.

Remarquons que ce problème peut être résolu en multipliant le nombre des pattes de sortie des unités en prévision de connexions multiples. Mais il faut également multiplier le nombre des pattes d'entrées et modifier le programme des unités.

Exemple



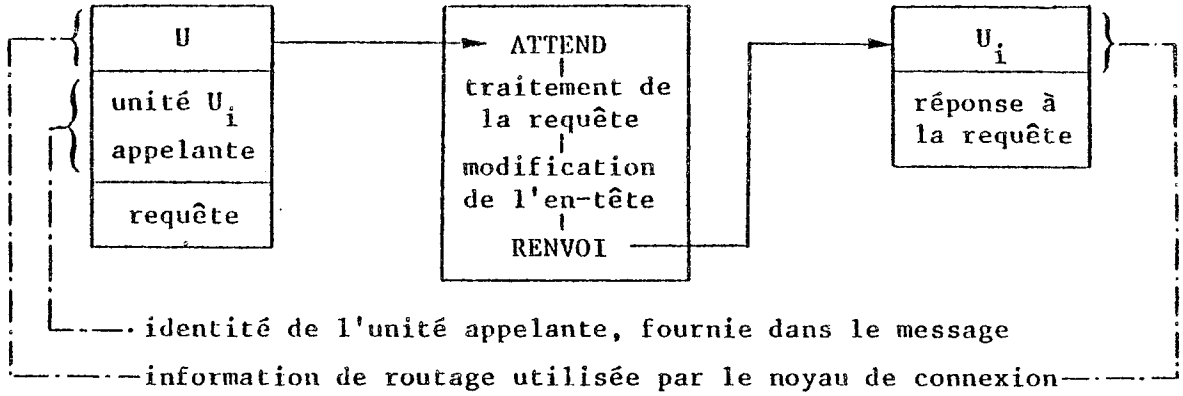
Unité U_1 définie fonctionnellement par 3 entrées et 2 sorties → Unité U_3 = U_1 modifiée pour supporter 3 fois plus de connexions

Cette solution n'est praticable que si le nombre de connexions prévu au départ est très grand. Mais il complique de toute façon la programmation des unités ainsi que la construction des machines puisqu'il faut maintenir en permanence un inventaire des pattes libres pour chaque unité.

b) Connexion dynamique

Le message de sortie doit alors contenir une information permettant au noyau d'acheminer le message vers la patte d'entrée désirée.
Cette information de "routage" peut avoir différentes formes. On peut envisager en premier lieu que le message émis par une unité contienne toujours l'identité du destinataire (unité et numéro de ligne). Ce mode de connexion est relativement plus puissant que le précédent. Il permet par exemple qu'une unité soit utilisée comme un sous-programme.

processus d'une unité
"sous programme" U

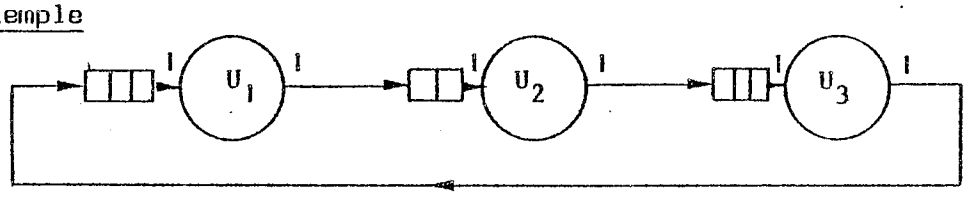


Dans cette solution, les unités doivent se nommer pour pouvoir s'appeler. D'autre part, il faut qu'elles aient un comportement interne qui ne dépende pas uniquement des fonctions qu'elles réalisent mais qui dépende aussi de la façon dont elles sont appelées. Cette solution fait manifestement dépendre chaque unité des machines dans lesquelles on voudra l'intégrer, ce qui est contraire aux objectifs que nous nous sommes fixés.

Une autre solution consiste à définir une partie des connexions de façon statique comme nous l'avons suggéré en a), puis à reporter la connexion effective au moment de l'exécution de la primitive RENVOI. Le système HYDRA [WUL] utilise une telle méthode ; nous allons la présenter brièvement.

- Dans un premier temps, un ensemble de connexions est défini de façon statique entre les pattes d'entrée et de sortie des unités.

Exemple



- Deux primitives, APPEL et RETOUR, sont définies d'autre part en plus de la primitive RENVOI pour l'émission de messages par les unités. Ces trois primitives ont en commun d'être non bloquantes et d'avoir pour paramètres le message à émettre et un code de sortie. La primitive APPEL a un paramètre supplémentaire qui est un numéro de patte d'entrée et qui

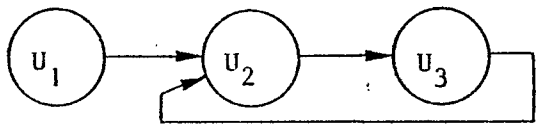
définit un point de retour dans l'unité. Le noyau de connexion gère pour chaque message une pile de points de retour et associe donc aux primitives APPEL et RETOUR des opérations d'empilement et de désempilement.

Pour ces deux primitives, la destination du message émis est donc déterminée par le noyau de connexion à partir du contenu de la pile associée au message. Si la pile est vide, la destination du message est définie comme pour la primitive RENVOI par les connexions statiques.

Le cheminement effectif des messages n'est donc pas entièrement défini par les connexions statiques ; il dépend aussi de la façon dont les unités émettent les messages.

Dans l'exemple précédent, si U_1 fait APPEL, U_2 fait RENVOI et U_3 fait RETOUR, le cheminement sera bien celui indiqué sur le schéma.

Mais si U_1 fait APPEL, U_2 fait APPEL et U_3 fait RETOUR, le cheminement des messages sera le suivant :



Ce mode de connexion permet de traduire des relations de sous-traitance entre les unités tout en évitant que les unités se nomment explicitement. Mais il ne supprime pas l'un des inconvénients de la solution précédente : une partie de la connexion est définie dans l'unité par la primitive qui émet les messages de sortie.

Nous avons donc recherché une méthode de connexion n'ayant aucun des inconvénients rencontrés, c'est-à-dire satisfaisant aux objectifs suivants :

- définition "a posteriori" des connexions
- possibilités de connexions multiples
- définition des unités indépendamment des machines.

c) Mécanisme de connexion proposé

Il est inspiré de la méthode utilisée dans le système HYDRA en ce sens qu'il comporte une définition statique et une définition dynamique.

Chaque machine est d'abord définie de façon statique par une matrice de connexion indiquant pour chaque patte de sortie de chaque unité, à quelle ligne de quelle unité il faut la connecter. Nous verrons au

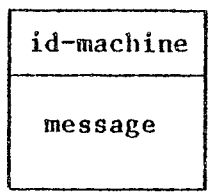
paragraphe suivant qu'il est intéressant de remplacer cette matrice par un automate de transitions.

Par exemple, la machine donnée en exemple au début de ce paragraphe est définie statiquement par la matrice suivante :

unité	code-sortie	unité à connecter	n° de ligne
U_1	1	U_2	1
	2	U_2	2
	3	U_3	1
U_2	1	U_3	2
U_3	1	U_1	1
	2	U_3	1
	3	U_2	2

La connexion est effectuée seulement à l'exécution : chaque message de sortie comporte un en-tête permettant au noyau de connexion d'identifier la machine pour le compte de laquelle le message est émis ; au moyen de la matrice de connexion, le noyau peut alors déterminer quelle unité devra recevoir le message et sur quelle ligne.

L'en-tête d'un message contient l'identité de la machine



Cette identité ne concerne en principe que le noyau de connexion et pas les unités puisque celles-ci sont définies "a priori" indépendamment des machines.
Mais nous avons vu qu'une unité doit pouvoir multiplexer les traitements des messages qui lui parviennent, c'est-à-dire qu'elle peut traiter plusieurs messages en même temps. Les messages doivent donc être

communiqués aux unités munies de leur en-tête pour que le noyau de connexion puisse retrouver les identités de machines dans les messages de sortie.

d) Définition des messages

Les différents aspects du fonctionnement d'une machine abstraite seront examinés en détail au paragraphe qui suit. Nous verrons en particulier que la nécessité de contrôler ce fonctionnement nous a conduits à limiter le nombre des messages émis par les unités.

Si l'interprétation des en-têtes de messages est l'aspect dynamique du mécanisme de connexion, la définition des matrices de connexion en constitue l'aspect statique et nous allons revenir maintenant sur cette définition en la précisant.

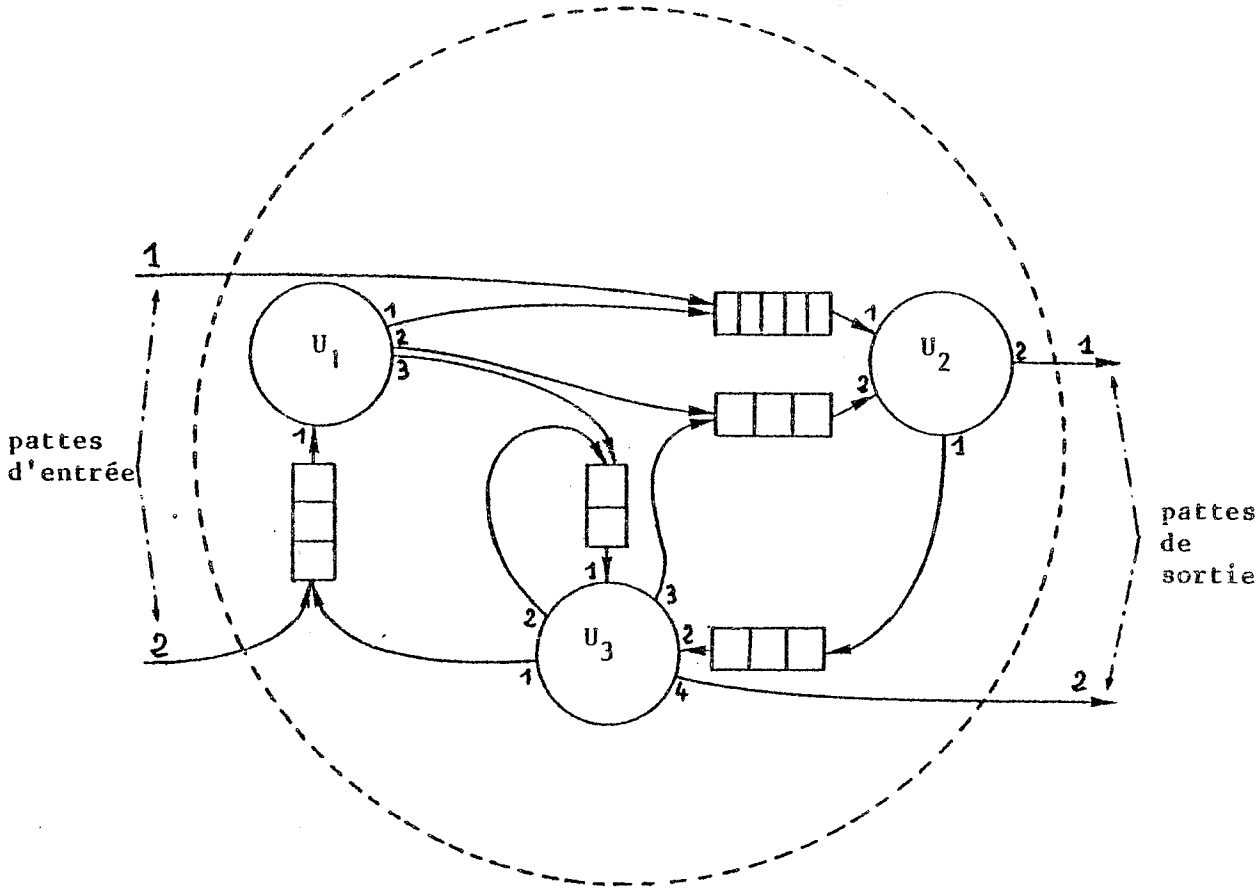
Nous avons supposé implicitement jusqu'ici que la définition d'une machine abstraite impliquait de ne laisser aucune patte d'entrée ou de sortie "dans le vide". Nous pouvons dire au contraire que si l'on veut pouvoir soumettre du travail à une machine et récupérer les résultats en utilisant l'outil de communication que nous avons défini, il faut que cette machine comporte des pattes d'entrée et des pattes de sortie, et qu'elles correspondent à des entrées et sorties d'unités de la machine.

La définition d'une machine abstraite par sa matrice de connexion doit être complétée par les données suivantes :

- des entrées initiales qui sont des entrées d'unités de la machine
- des pattes de sortie correspondant à une ou plusieurs unités de la machine et dont la connexion à d'autres unités n'est pas définie statiquement dans la matrice de connexion de cette machine.

Exemple

Complétons l'exemple donné plus haut :



Machine M

Cette machine a pour patte d'entrée 1 la patte d'entrée 1 de U_2, et pour patte d'entrée 2 la patte d'entrée 1 de U_1.
Elle a deux pattes de sortie correspondant à la sortie 2 de U_2 et à la sortie 4 de U_3.

Pour faire une analogie avec une machine réelle, nous dirons que le "chargement initial IPL" d'une machine se traduit par la soumission d'un message à l'une des pattes d'entrée ; la lecture de données externes sur les organes périphériques correspond alors à la soumission de messages sur les autres pattes d'entrée. Ce genre de considération nous a amené à spécialiser une des pattes d'entrée d'une machine comme "entrée-initiale". Nous verrons plus loin que c'est sur cette "entrée initiale" que sont déposés les messages initiaux lors des créations de processus sur les machines construites.

e) Inclusion de machines

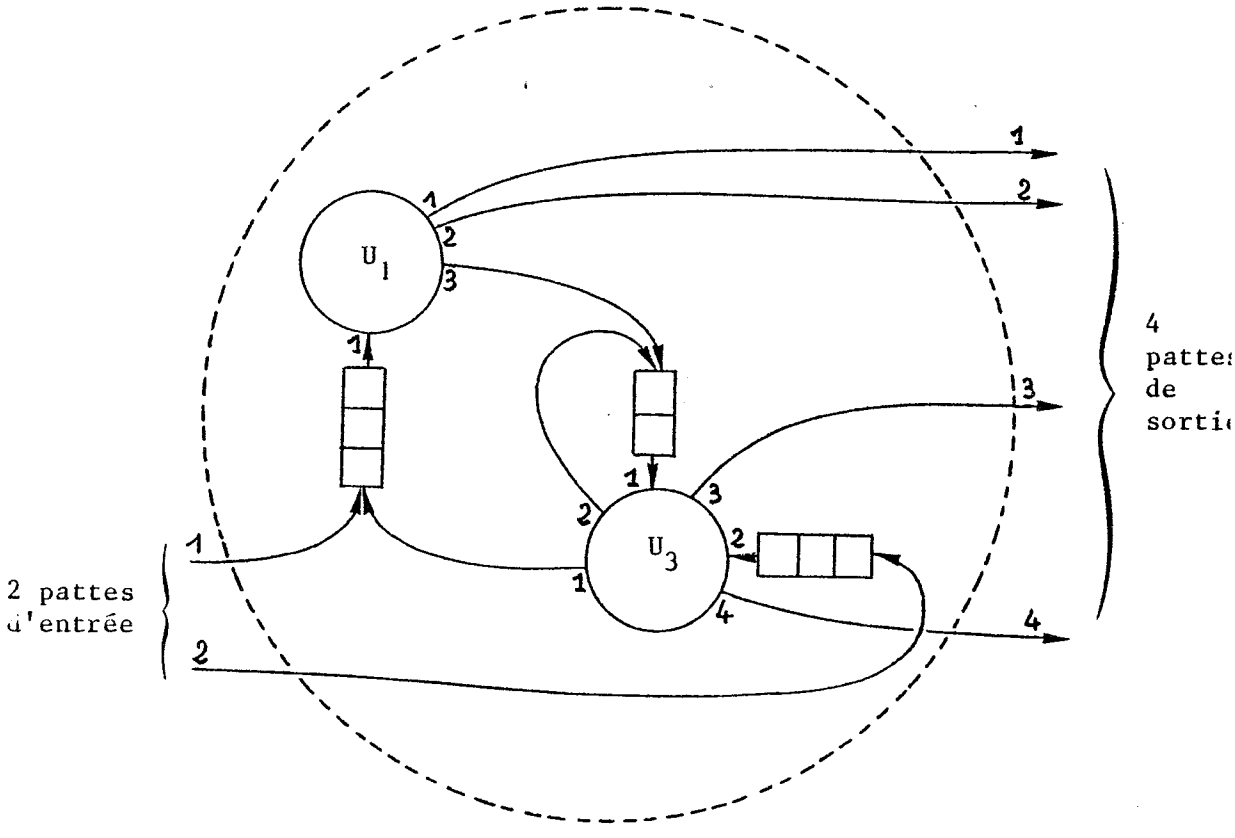
Ainsi définies, les machines abstraites présentent donc les mêmes interfaces externes que les unités. En conséquence, il est possible de construire des machines abstraites composées d'unités, et de machines

abstraites. Nous reviendrons au paragraphe 4.1 sur les possibilités d'enrichissement ou de réduction liées à l'adjonction progressive de nouvelles unités dans une machine. Mettons enfin l'accent sur la possibilité de créer des machines spécialisées dans certains traitements.

Exemple

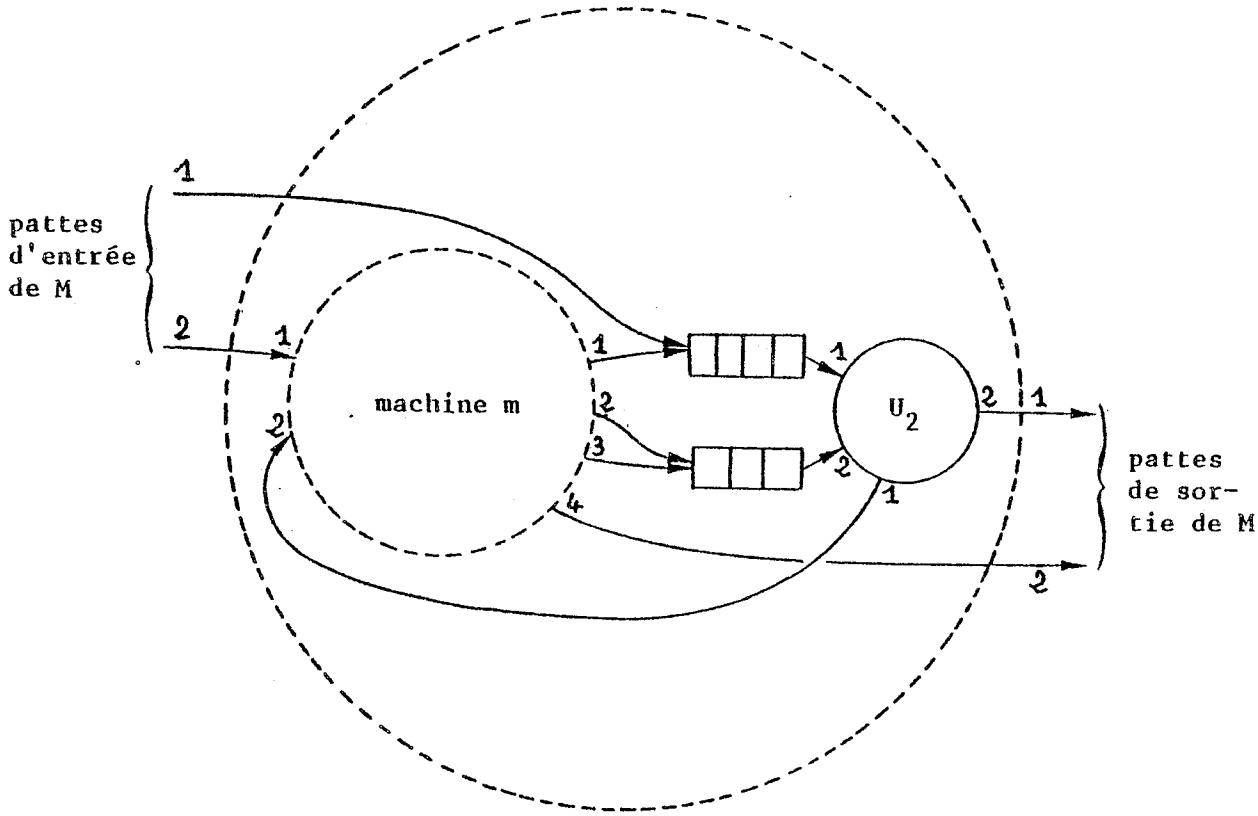
Pour reprendre l'exemple décrit plus haut, nous aurions pu définir la machine en deux temps :

1 - Machine m constituée des unités U_1 et U_3



Machine m

2 - Machine complète M par adjonction de l'unité U_2



Machine M

On verra au chapitre V (MAS) que le noyau de système que nous avons réalisé est lui-même composé de 5 machines.

On notera sur l'exemple précédent qu'une patte d'entrée d'une machine peut être définie par la donnée d'une patte d'entrée d'une machine incluse. Au bout du compte, c'est toujours une patte d'entrée d'une unité.

Le paragraphe suivant décrit l'aspect dynamique de la connexion. En préambule à cette description, signalons une des conséquences qu'a l'inclusion d'une machine dans une autre machine.

Chaque message est porteur d'un en-tête qui contient l'identité de la machine dans lequel ce message chemine. Lorsqu'un message arrive sur une patte d'entrée d'une machine incluse, le noyau doit mettre dans l'en-tête

du message l'identité de la machine incluse. Pour ne pas effacer l'identité de la machine englobante, il effectue donc un empilement. Lorsqu'un message sort d'une machine incluse, le noyau effectue l'opération inverse, c'est-à-dire un désempilement, et retrouve donc dans l'en-tête du message l'identité de la machine englobante. Cette gestion de pile n'est pas très coûteuse si le niveau d'inclusion des machines n'est pas très élevé. En tout état de cause, elle peut être remplacée par une compilation faite à la création de chaque machine et destinée à ramener toutes les unités au même niveau. Par ailleurs, la taille de la pile est connue à la compilation car nous interdisons la définition récursive des machines.

Par exemple la machine décrite dans ce paragraphe a été définie de deux façons différentes.
La première description pourrait être déduite de la seconde de façon automatique.

II.2.3. Fonctionnement des machines abstraites

L'activité d'une machine abstraite comporte deux aspects :

- le traitement des messages par les unités
- l'échange de messages entre les unités.

Cette activité correspond à un certain nombre de travaux qui sont exécutés simultanément par la machine abstraite ; par exemple n utilisateurs actifs, sur une "machine conversationnelle". Notre modèle de décomposition est basé sur le contrôle précis de cette activité, le principe directeur étant que les travaux sont des entités connues du noyau et obligatoirement déclarées. Nous donnons à ces entités le nom de processus.

A ce stade de l'exposé, le lecteur a le droit d'être dérouté par le choix de ce terme. Aussi allons-nous essayer de le justifier en décrivant les caractéristiques de nos travaux afin de montrer qu'ils correspondent à la notion habituelle de processus.

Le noyau ne peut contrôler l'activité d'une machine abstraite qu'aux moments où les messages sont échangés entre les unités. Nous avons donc défini des règles de fonctionnement agissant au niveau de la communication par messages. Ces règles sont les suivantes :

- Chaque message est associé à un travail et réciproquement. En plus de l'en-tête décrit au paragraphe précédent, chaque message est donc porteur de l'identification du travail auquel il est associé. Notons que cette identification est nécessaire de toute façon dans la mesure où une machine abstraite, au même titre qu'une unité, doit pouvoir traiter simultanément des messages associés à plusieurs travaux.

- La soumission d'un travail à une machine ou à une unité s'effectue par l'envoi d'un message sur une patte d'entrée de cette machine ou de cette unité. Inversement, lorsqu'un message est émis sur une patte de sortie, cela signifie que la machine ou l'unité a fini le travail associé au message.

- A tout instant, il peut exister au plus un message portant l'identification d'un travail donné.

Il découle de ces règles qu'une machine ou une unité ne peut pas émettre plus de messages qu'elle n'en reçoit. Notons que cela protège le noyau de communication contre tout risque de saturation en messages.

Mais il y a là une des limites de notre modèle puisque l'exécution d'un travail par une machine abstraite doit être séquentielle : elle se traduit par le cheminement d'un message dans la machine abstraite.

En fait le choix de ne pas autoriser l'émission par une unité de plusieurs messages associés au même travail provient de la difficulté de définir le point de jonction des activités parallèles engendrées. Pour contrôler le déroulement d'un travail, il est nécessaire en effet de pouvoir d'abord démarrer ce travail, puis de pouvoir définir à tout instant un état d'exécution de ce travail et enfin de pouvoir déterminer à quel moment ce travail se termine.

Ce contrôle est bien sûr grandement facilité par notre choix d'exécution séquentielle, notamment dans le cas de la détection et prise en compte d'erreurs (cf. § 4.2).

La notion de travail telle que nous venons de la définir correspond bien à la notion habituelle de processus et c'est ce terme que nous utiliserons à partir de maintenant.

II.2.31. Les processus

Les processus sont donc des entités actives gérées par le noyau. Nous avons vu que l'exécution d'un processus par une machine abstraite se traduit par le cheminement d'un message dans cette machine.

Nous allons maintenant donner une définition plus formelle des processus :

- primitives de création et destruction
- états d'un processus.

Nous avons vu que les unités n'ont pas la possibilité de fabriquer des messages spontanément. Cependant, il faut bien que les messages qu'elles traitent aient été fabriqués à un moment ou à un autre.

La primitive CREER PROCESSUS permet aux unités d'engendrer de nouveaux processus

CREER PROCESSUS (identité d'une machine, message).

Le message fourni en paramètre est chaîné par le noyau sur la file d'attente associée à l'entrée initiale de la machine spécifiée. Ce message constitue le contexte initial du processus.

L'identité du processus créé est fournie en retour à l'unité qui a émis la primitive.

Notons que cela constitue un moyen de faire exécuter en parallèle plusieurs parties d'un traitement donné : chaque partie sera exécutée par un processus particulier créé pour l'occasion. Lorsque la partie de traitement dévolue à chacun de ces processus sera terminée, les messages associés pourront être conservés ou détruits (primitive DETRUIRE PROCESSUS) par l'unité-mère ou une autre unité.

Nous reviendrons plus en détail au paragraphe 4.2 sur les outils proposés dans notre modèle pour le contrôle des processus.

Etats d'un processus

A tout instant un processus est dans l'un des 4 états suivants, ces états étant définis par la situation du message associé au processus :

- état initial : le message est dans la file d'attente associée à la patte d'entrée initiale
- état d'interprétation : le message a été lu par une unité et il est en cours de traitement par cette unité ; on dit alors que le contexte du processus est indéfini.
- état de transition : le message est sur une file d'attente d'une unité de la machine abstraite. Le contexte du processus est défini par le contenu du message
- état final : le message a été émis sur une des pattes de sortie de la machine abstraite. Il définit le contexte final du processus.

Lorsqu'un processus passe dans l'état final, le message de sortie associé est communiqué à une unité qu'on appelle "gérant" de ce processus ; nous reviendrons sur ce point au § 4.2 . Pour le noyau, ce processus continue à exister : il est simplement dans l'état final.

Un processus n'est détruit que par une demande explicite faite par un autre processus ou par lui-même (suicide) au moyen de la primitive :

DETRUIRE PROCESSUS (identité de processus).

Si le processus désigné n'est pas dans l'état d'interprétation, la destruction est immédiate : le message et les informations de routage gérées par le noyau sont détruits.

Si le processus désigné est par contre dans l'état d'interprétation, la destruction doit être différée jusqu'au moment où le message associé au processus sera émis par l'unité qui est en train de le traiter.

II.2.32. L'automate de transition

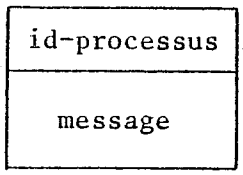
L'exécution d'un processus par une machine abstraite se traduit donc par le cheminement d'un message dans cette machine.

Chaque message comporte un en-tête standard que le noyau interprète pour déterminer sur quelle unité ou quelle machine le message doit être aiguillé. Cet aiguillage fait intervenir, on l'a vu, une matrice de connexion définissant la machine abstraite.

Avant de revenir sur la définition de cette matrice, nous allons préciser la structure des messages.

Dans la mesure où l'en-tête des messages ne concerne pas directement les unités, nous avons cherché à réduire le plus possible sa taille.

L'en-tête d'un message contient l'identification du processus auquel il est associé.

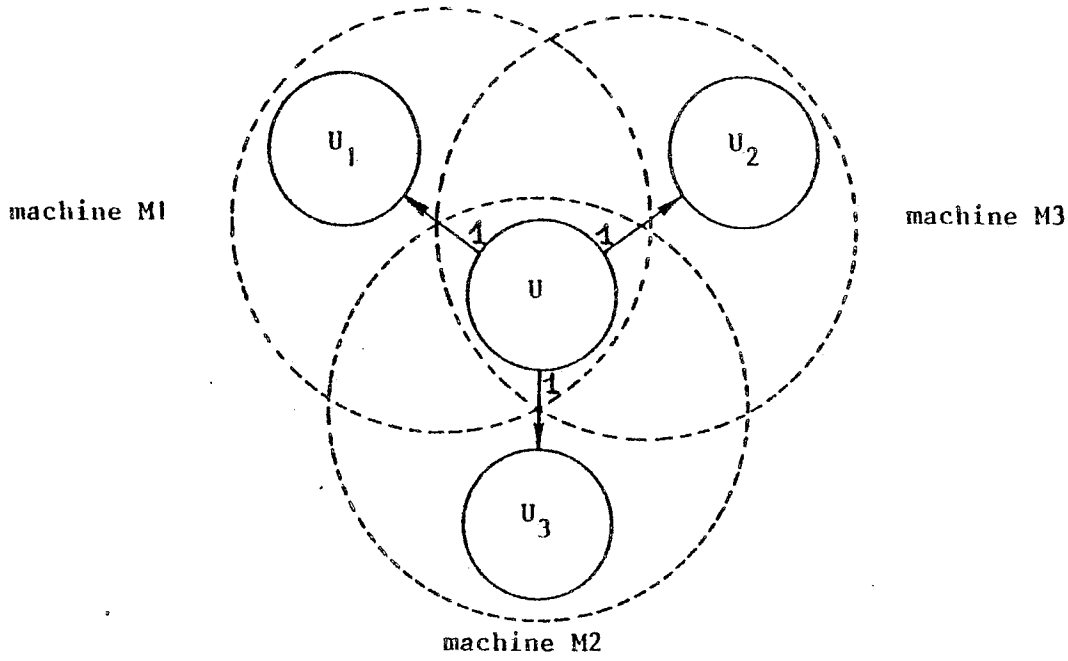


L'id-processus permet au noyau de retrouver les informations de routage qui sont associées au processus et que nous avons mentionnées jusqu'ici.

Plus précisément ces informations se résument en une pile d'exécution ainsi constituée :
chaque niveau de la pile contient l'identité d'une machine abstraite ainsi que l'état courant d'exécution du processus sur cette machine et la position du message dans cette machine. Au bas de la pile, on trouve la machine abstraite sur laquelle a été créé le processus. Au sommet de la pile, on trouve la machine sur laquelle s'exécute le processus.

Sur tous les niveaux autres que le sommet, le processus est dans l'état d'interprétation ; sur le sommet il peut être dans un état quelconque.

Revenons maintenant sur la définition statique des machines abstraites au moyen de matrices de connexion. Une patte de sortie d'une unité peut être connectée à plusieurs autres unités, mais à la condition que ces unités soient dans autant de machines différentes puisque chaque matrice ne définit qu'une connexion possible pour chaque patte de sortie.



Une unité peut par exemple être connectée comme unité de service à plusieurs autres unités pour effectuer une sous-traitance donnée. Le mécanisme de connexion permet à chacune de ces unités de récupérer le résultat de cette sous-traitance.

Mais il serait coûteux d'être obligé de définir une machine abstraite pour chaque couple d'unités devant collaborer de cette façon.

La relation de sous-traitance doit pouvoir être prise en compte de façon plus simple entre les unités d'une même machine.

C'est dans ce but que nous avons modifié légèrement la définition d'une machine abstraite, en remplaçant la matrice de connexion par une matrice de transition représentant un automate d'états finis.

Chaque ligne de la matrice est ainsi constituée :

état d'entrée	machine ou unité et numéro de patte d'entrée	code de sortie	nouvel état		code de sortie	nouvel état

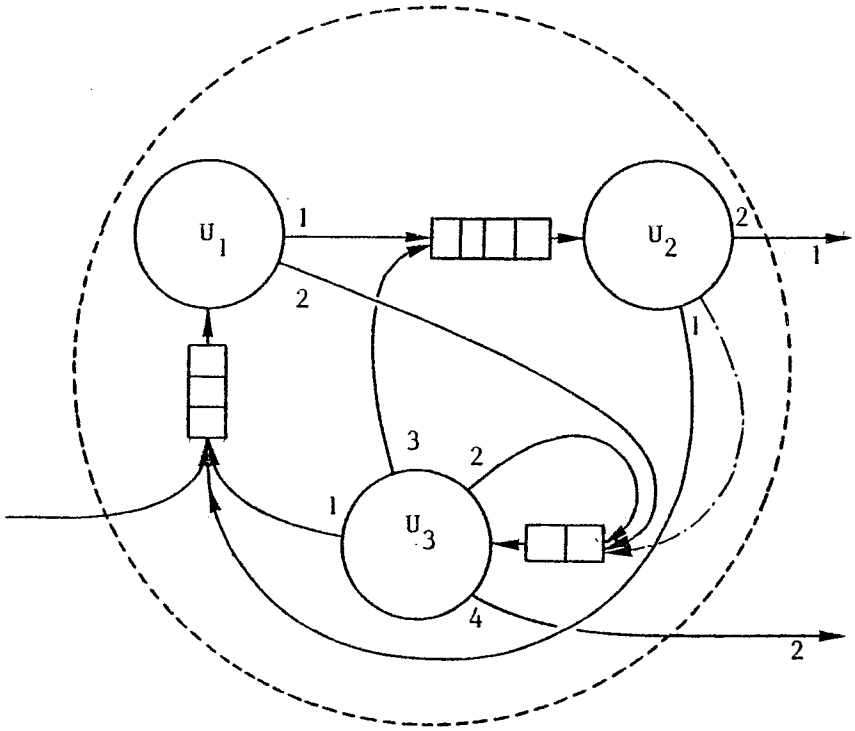
Dans la pratique, les états sont des numéros de ligne de la matrice de transition. On voit que cette définition diffère de celle donnée pour les matrices de connexion dans la mesure où un même couple (identification d'unité ou de machine, code de sortie) peut correspondre à plusieurs états différents et donc apparaître plusieurs fois dans une matrice de transition.

La première ligne de la matrice correspond à l'état initial, on y trouve évidemment l'identification de l'unité initiale de la machine décrite.

Les états de sortie sont marqués d'un signe spécial.

Exemple

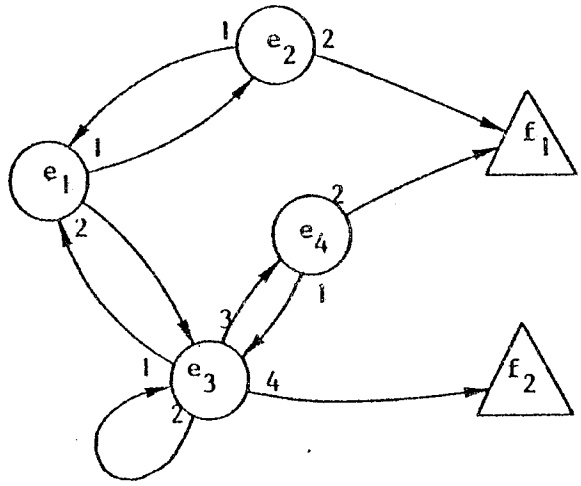
Modifions l'exemple de la machine M déjà présentée dans ce chapitre.



Supposons que U_2 réalise une sous-traitance :

- pour le compte de U_1 suivant le circuit $U_1.1 \rightarrow U_2.1$
- mais aussi pour le compte de U_3 suivant le circuit $U_3.3 \rightarrow U_2.1$.

Dans ce dernier cas, le retour à U_3, représenté sur le schéma par la flèche en pointillé, peut être exprimé par l'automate de transition suivant :



- état initial e_1
- états finaux f_1, f_2

Dans cet automate, les deux états e_2 et e_4 correspondent à l'unité U_2, cela signifie que lorsqu'un processus exécuté par la machine correspondante se trouve dans l'un de ces deux états, le message associé est en cours de traitement par l'unité U_2.

La pile d'exécution située dans le bloc de contrôle de chaque processus contient pour chaque niveau la position du message associé au processus dans une machine abstraite. Cette position est donc en fait l'état du processus dans l'automate définissant la machine.

Lorsqu'un message doit être aiguillé sur une patte d'entrée d'une machine incluse, le noyau inscrit sur le sommet courant de la pile d'exécution un numéro d'état de l'automate associé à la machine englobante. Puis il incrémente le pointeur de pile et inscrit en sommet de pile l'identité de la machine incluse et l'état initial. L'automate de la machine incluse prend alors le relais de celui de la machine englobante.

La création d'une machine abstraite s'effectue en définitive par la primitive suivante :

CREER MACHINE (e_1 : (unité,ligne), (s_1:e_1^1, s_2:e_1^2,..., s_{n_1}:$e_1^{n_1}$), [E],

e_2 : (unité,ligne), (s_1:e_2^1, ..., s_{n_2}:$e_2^{n_2}$), [E].

⋮

e_p : (unité,ligne), (s_1:e_p^1, ..., s_{n_p}:$e_p^{n_p}$), [E],

e_{p+1} : s_{p+1}

...

e_n : s_n) identité de la machine créée.

e_1, ..., e_n : états de l'automate

e_{p+1}, ..., e_n : états finaux de l'automate.

s_i : code de sortie de l'unité

e_k : nouvel état d'automate, appartenant à e_1, ..., e_n

s_{p+1}, ..., s_n : codes de sortie de la machine correspondant aux états finaux $e_{p+1}, \ldots, e_n$

(unité,ligne) : référence d'une patte d'entrée d'un composant

E : indique que la patte d'entrée référencée est une patte d'entrée de la machine.

La première patte d'entrée de la machine est prise comme entrée initiale. En retour, le noyau fournit l'identité de la machine créée.

II.2.4. Le mécanisme d'extension

Le modèle que nous avons décrit jusqu'à présent repose sur l'existence de processus primitifs définis de façon relativement figée et réalisés par un multiplexage des deux processeurs IRIS 80. Au moyen des primitives que nous avons présentées, ces processus peuvent être organisés suivant une architecture composée d'unités et de machines abstraites. On peut également créer sur ces machines des activités autonomes, que nous appelons également processus et dont ils deviennent alors les serveurs. Nous allons voir que ces processus peuvent être munis des mêmes outils de connexion que les processus primitifs. Le développement que nous avons fait à partir de ces outils peut alors s'appliquer à ces processus qui peuvent donc à leur tour être serveurs. Puis ces serveurs peuvent être groupés en unités, à partir desquelles on peut construire de nouvelles machines abstraites, ...

Cette méthode de construction conduit à la définition de niveaux de machines abstraites : chaque machine est composée d'unités réalisées sur des machines de niveau inférieur.

Nous dirons qu'il y a là extension verticale, l'extension horizontale correspondant à la création de machines par composition d'unités de base et de machines déjà existantes.

C'est le mécanisme d'extension verticale que nous développons dans ce paragraphe.

II.2.41. Les unités ATTEND et RENVOI

Le problème que nous avons voulu résoudre est donc de fournir le même outil de connexion aux processus créés sur les machines abstraites qu'aux processus primitifs.

Afin de bien situer la façon dont nous avons procédé, nous allons revenir un instant sur la notion de machine considérée du point de vue fonctionnel.

Chaque machine abstraite définit un répertoire d'instructions et d'objets qui constitue le langage de programmation "machine" de cette machine abstraite. Si la machine abstraite contient une machine incluse, cela signifie que le langage de la machine englobante se déduit, par enrichissement et/ou par réduction, de celui de la machine abstraite incluse (cf. § III.21). En tout cas, ce sont les unités qui interprètent les instructions du langage. Si l'on veut modifier le langage d'une machine abstraite, il faut donc soit modifier certaines unités, soit ajouter de nouvelles unités.

Cette dernière solution est la plus commode à mettre en oeuvre puisqu' elle permet de prédéfinir un certain nombre d'instructions en prédéfinissant des unités. Le créateur d'une machine abstraite peut ensuite intégrer ou non chacune de ces unités dans sa machine selon qu'il veut intégrer ou non les instructions correspondantes dans le langage de cette machine.

Revenons maintenant au problème posé qui est donc de munir les processus exécutés par les machines abstraites des outils de connexion décrits précédemment.

Le mécanisme de connexion comprend d'une part la définition des automates de transition et d'autre part l'interprétation des primitives ATTEND et RENVOI. Plus précisément cette interprétation se compose de deux parties :

- une partie "communication" qui consiste en une recopie de message. Cette recopie nécessite une norme de programmation respectée par tous les processus primitifs ;
- une partie "connexion" qui fait intervenir, en plus de l'automate de transition, l'identité du processus (donc de l'unité) qui a appelé la primitive et l'en-tête du message référencé dans la primitive.

Cette partie "connexion" est indépendante des normes de programmation imposées aux processus primitifs, ainsi même que du langage machine IRIS 80.

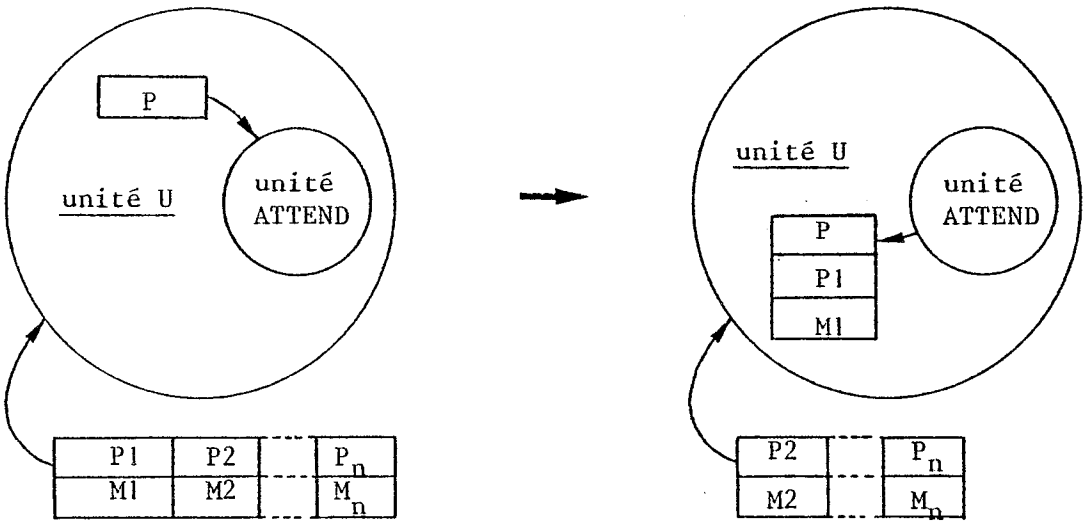
C'est cette partie qui contient l'aspect dynamique du mécanisme de connexion. Nous l'avons donc isolée et nous en avons fait deux unités que nous avons appelées ATTEND et RENVOI. Ces deux unités sont donc prédéfinies dans notre modèle, c'est-à-dire que le créateur d'une machine abstraite peut les utiliser comme n'importe quelle autre unité en les désignant simplement par leur nom.

En réalité ces deux unités correspondent à deux programmes appelés de façon procédurale par le noyau, mais nous les appelons cependant unités puisqu'elles peuvent être intégrées comme telles dans une machine abstraite.

Lorsque le noyau de connexion rencontre leur identité dans un automate de transition, il effectue les traitements suivants :

Rencontre de l'unité ATTEND :

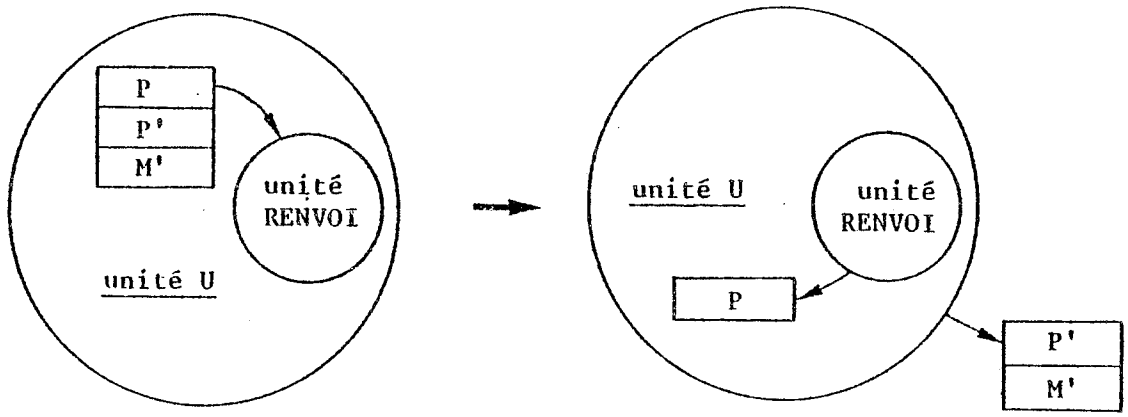
Le processus P qui a fait la transition vers cette unité est supposé être serveur dans une unité U ; si ce n'est pas le cas, il y a erreur, le processus P n'a pas le droit de faire ATTEND. Sinon le processus P repart vers une nouvelle transition avec le premier message qui était en attente sur U.



file d'attente associée à U AVANT

file d'attente associée à U APRES

Rencontre de l'unité RENVOI :



Le processus P qui a fait la transition vers l'unité RENVOI est supposé être serveur d'une unité U. Si ce n'est pas le cas, il y a erreur puisque le processus P n'a pas le droit de faire RENVOI. Sinon le processus P repart vers une nouvelle transition.
D'autre part, le contenu du message "apporté" par P est interprété par le noyau de connexion. Il doit normalement contenir un en-tête, soit P' ; si ce n'est pas le cas, c'est que le processus P, donc l'unité U, a renvoyé un message incorrect : il y a erreur (voir le paragraphe intitulé "défauts de fonctionnement"). Sinon le processus P' effectue une transition de sortie de l'unité U ; c'est l'automate de transition de P' qui est alors utilisé.

D'autre part, lorsque nous avons présenté les primitives ATTEND et RENVOI au paragraphe 2.12, nous leur avons associé deux paramètres qui sont partie intégrante du mécanisme de connexion :

- primitive ATTEND : choix préférentiel sur une ligne dont le numéro est fourni en paramètre de la primitive
- primitive RENVOI : donnée d'un code de sortie définissant une patte de sortie de l'unité.

Dans la mesure où ces primitives sont des "unités", les paramètres doivent figurer dans l'en-tête des messages puisque ces messages sont la seule information communiquée par les processus aux unités ATTEND et RENVOI.

II.2.42. Extension

L'inclusion des unités ATTEND et RENVOI dans une machine abstraite permet aux processus exécutés par cette machine de recevoir et d'émettre des messages. Une unité peut alors être créée sur une machine abstraite pourvu que cette machine contienne les unités ATTEND et RENVOI, quel que soit d'ailleurs le niveau où elles se trouvent (unités contenues éventuellement dans une machine incluse).

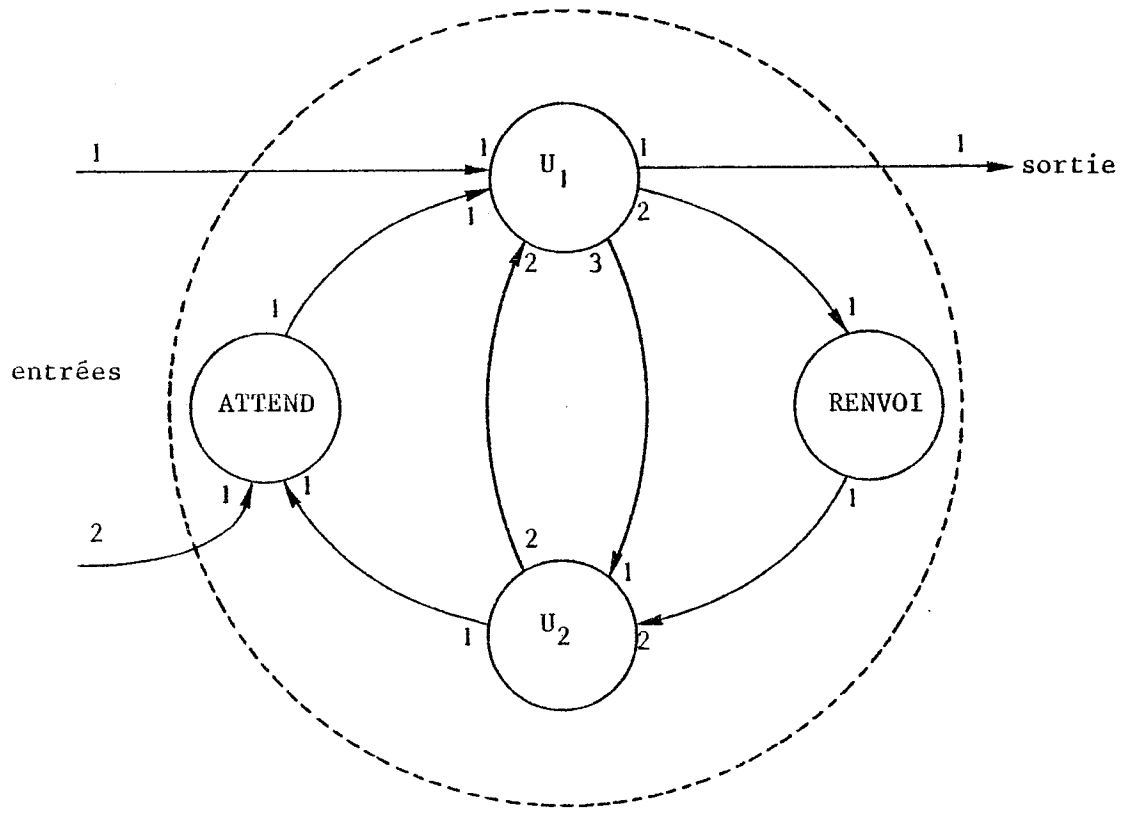
La création d'une unité s'effectue par la primitive CREER-UNITE, que nous avons déjà présentée (§ 2.13). La forme générale de cette primitive est la suivante :

CREER UNITE (id-machine, message initial, degré de multiplication,
nombre de pattes d'entrée, nombre de pattes de sortie).

L'identité de l'unité créée est fournie en retour.

Remarque

Les pattes d'entrée et de sortie définies par la primitive CREER UNITE n'ont rien à voir avec celles de la machine supportant l'unité. Soit par exemple la machine M ainsi définie.



Les pattes d'entrée de cette machine correspondent à un sous-ensemble des pattes d'entrée des unités qu'elle contient :

- entrée n° 1 de U_1
- entrée de ATTEND.

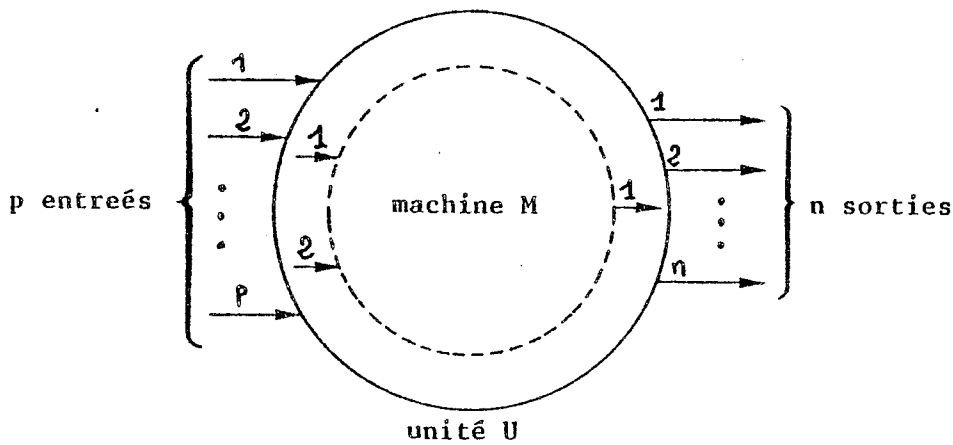
Les pattes de sortie de cette machine correspondent à un sous-ensemble des pattes de sortie des unités qu'elle contient :

- sortie n° 1 de U_1.

Lorsqu'on crée une unité U sur cette machine, on définit un message que le noyau dépose sur l'entrée initiale de la machine autant de fois qu'indiqué par le degré de multiplication. Pour chaque processus ainsi créé, un message chemine donc dans la machine M, et le noyau sait reconnaître que ces processus, et seulement ceux-là, sont associés à l'unité U.

Si un nouveau message arrive alors sur une des entrées de M, cela ne signifie pas qu'il arrive sur l'unité U. De même si un des processus de U emprunte la sortie 1 de M, cela ne signifie pas que le message associé est émis par l'unité U, mais simplement que l'unité U a un processus serveur de moins.

Les unités ATTEND et RENVOI permettent de définir entièrement de nouvelles entrées et de nouvelles sorties pour chaque unité que l'on crée.



Les unités ainsi créées sont définies fonctionnellement de la même façon que les unités construites à partir des processus primitifs. Comme la constitution des automates de transition ne fait intervenir que la définition fonctionnelle des unités, la primitive CREER MACHINE présentée

précédemment permet de construire des machines abstraites à partir d'unités elles-mêmes construites sur des machines abstraites.

Le modèle que nous avons développé permet bien en définitive de mettre en oeuvre la méthode d'abstraction raffinement grâce à laquelle une application peut être décomposée en niveaux de machines abstraites.

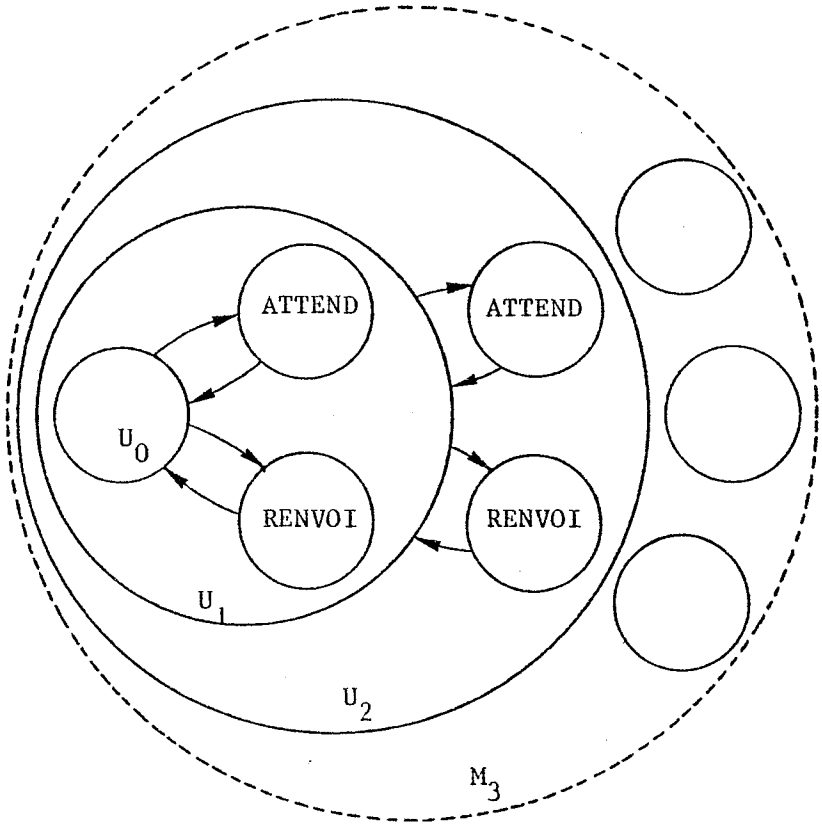
Nous reviendrons au chapitre III-2 sur un certain nombre de relations que ces niveaux de machines induisent entre les langages qu'ils définissent. Nous développerons en particulier la relation d'interprétation et les relations de représentation et de désignation des objets manipulés à chaque niveau et nous illustrerons chacune d'elles par un exemple.

II.2.43. Limites du mécanisme d'extension

Nous terminons la description du mécanisme d'extension en mettant en évidence une de ses limites due à la communication par messages.

Nous avons vu comment fonctionnent les unités ATTEND et RENVOI. Nous allons revenir un instant sur ce fonctionnement en l'illustrant sur un exemple plus complet que celui donné précédemment.

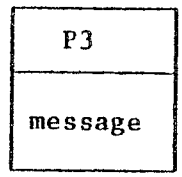
Supposons donc que nous ayons construit une application mettant en jeu trois niveaux de machines.



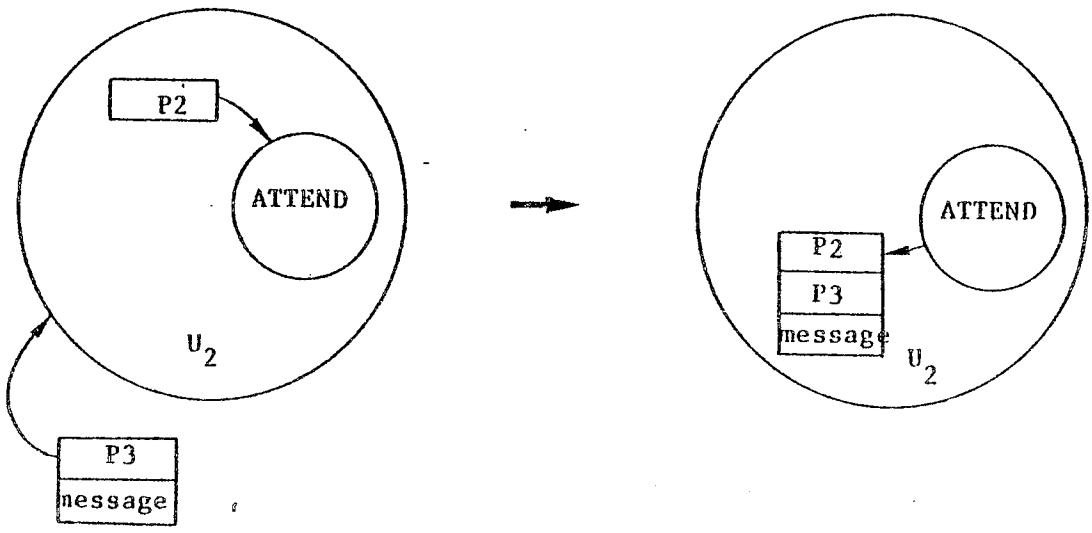
L'unité U_0 est constituée de processus primitifs.
L'unité U_1 est construite sur une machine M_1 constituée des unités U_0, ATTEND et RENVOI.
L'unité U_2 est construite sur une machine M_2 constituée des unités U_1, ATTEND et RENVOI.
Nous supposerons que cette unité U_2 est contenue dans une machine M_3 figurée en pointillé sur le schéma.
Pour simplifier, on suppose également que chaque unité est exécutée par un seul processus serveur.

Soit U_0 par P_0, U_1 par P_1, U_2 par P_2.

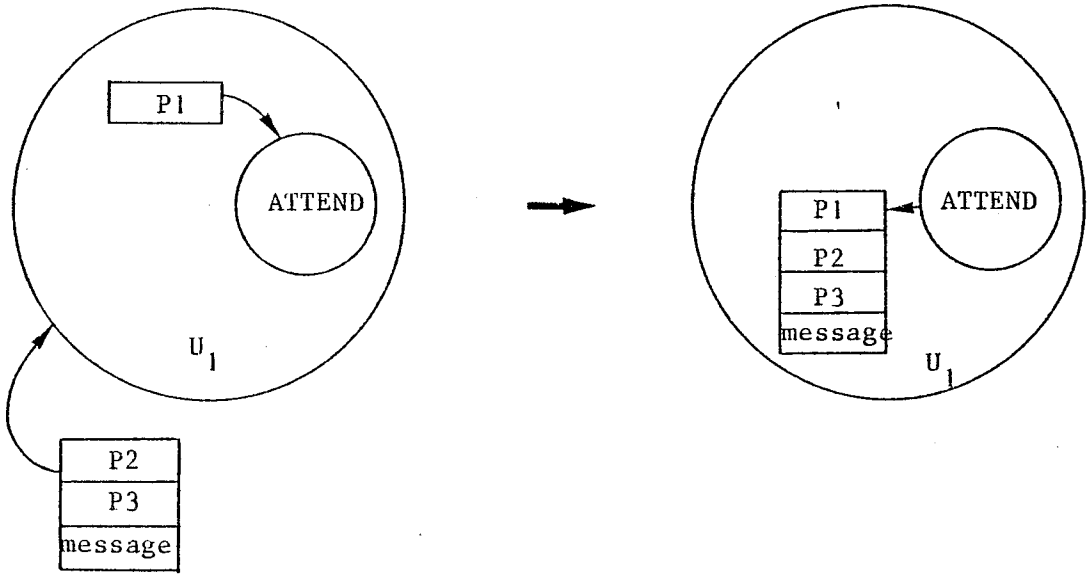
Sur M_3 tourne le processus P_3.
Le message associé à ce processus est ainsi constitué :



Lorsque ce message arrive sur l'unité U_2, il est mis en file d'attente. Il en est retiré par le processus P2 lorsque le message associé à P2 arrive sur l'unité ATTEND.



Le nouveau message arrive alors sur l'unité U_1, il est mis en file d'attente. Il en est retiré par le processus P1 lorsque le message associé à P1 arrive sur l'unité ATTEND.



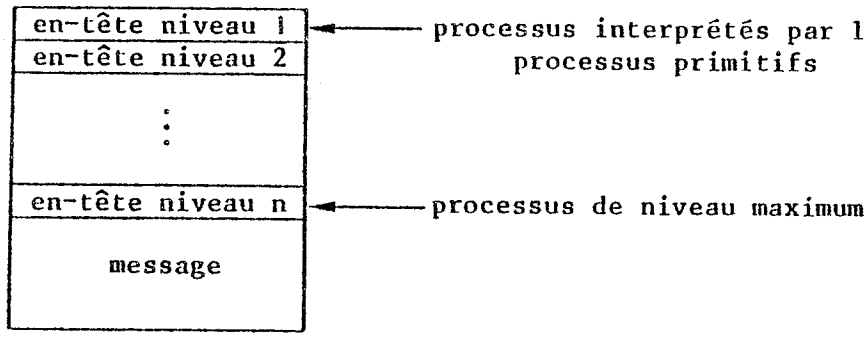
Le nouveau message arrive en définitive sur l'unité U_0, il est mis en file d'attente. Il en est retiré par le processus P_0 lorsque celui-ci appelle la primitive ATTEND.

Le processus P_0 traite en définitive un message contenant une pile de trois identités de processus.
Ce point constitue une limite quantitative du modèle, comme nous allons le voir.

Chaque unité traite le processus qui se trouve au sommet de la pile et peut avoir besoin d'accéder à l'identité de ce processus. Mais elle a également besoin d'accéder au message qui se trouve au bas de la pile. Le problème vient du fait qu'il lui est impossible de connaître la taille de cette pile.

En effet, cette taille est variable puisqu'elle est égale au nombre de niveaux que l'unité est en train de traiter. Ainsi dans notre exemple, P_0 traite P_1 qui traite P_2 qui traite P_3. Mais P_0 aurait pu tout aussi bien traiter directement P_3 si l'on avait mis U_0 dans la machine M_3.

Pour résoudre ce problème, il faut donc prévoir dans la mise en oeuvre du modèle un nombre maximum de niveaux autorisés au dessus du niveau IRIS 80 pour la création d'unités. Soit n ce nombre.
Chaque unité doit alors prévoir un certain nombre de niveaux pour la pile des en-têtes dans les messages qu'elle traite ; ce nombre étant d'autant plus grand que l'unité est plus près du niveau IRIS. Ainsi les processus primitifs doivent-ils réserver n niveaux dans les messages qu'ils traitent.



Une unité de niveau i doit prévoir une pile de n-i en-têtes dans les messages qu'elle traite.

II.3. LA MACHINE IRIS

Nous avons présenté jusqu'ici un certain nombre d'outils permettant de mettre en oeuvre la méthode d'abstraction raffinement par la construction de niveaux de machines abstraites.

Le développement que nous avons fait repose sur l'existence de processus primitifs définis de façon relativement figée et réalisés par un multiplexage des deux processeurs IRIS 80.

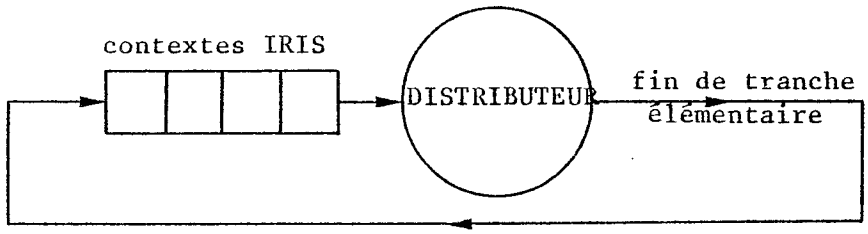
Dans ce paragraphe, nous allons montrer que le modèle de décomposition permet de prendre en compte la structure même du noyau HARDEXT. En d'autres termes, le noyau HARDEXT peut être défini fonctionnellement par une machine abstraite que nous appelons machine IRIS. Les processus primitifs sont alors des processus exécutés par cette machine IRIS.

La définition de la machine IRIS ne présente pas seulement un intérêt académique. Cette machine est en effet prédéfinie et répertoriée dans

notre modèle afin d'être mise à la disposition des utilisateurs. Tout processus créé sur une machine qui inclut la machine IRIS a alors la possibilité d'utiliser les services offerts par cette machine,notamment les instructions IRIS 80 en mode C esclave.

II.3.1. Le noyau de multiplexage

Les contextes IRIS associés aux processus primitifs "activables" sont placés dans une file d'attente commune aux deux processeurs IRIS 80. Un programme du noyau de multiplexage est chargé de distribuer la ressource unité centrale aux processus ainsi en attente. Il le fait sur la base d'un partage de temps : lorsqu'un processeur est disponible, le "distributeur" charge sur ce processeur un contexte IRIS pris dans la file d'attente en lui allouant une tranche de temps élémentaire grâce au mécanisme d'horloge virtuelle. Au bout de cette tranche de temps élémentaire, le contexte IRIS est récupéré par le distributeur et remis en file d'attente. Le processeur se trouve donc libre et un nouveau contexte peut être chargé ...

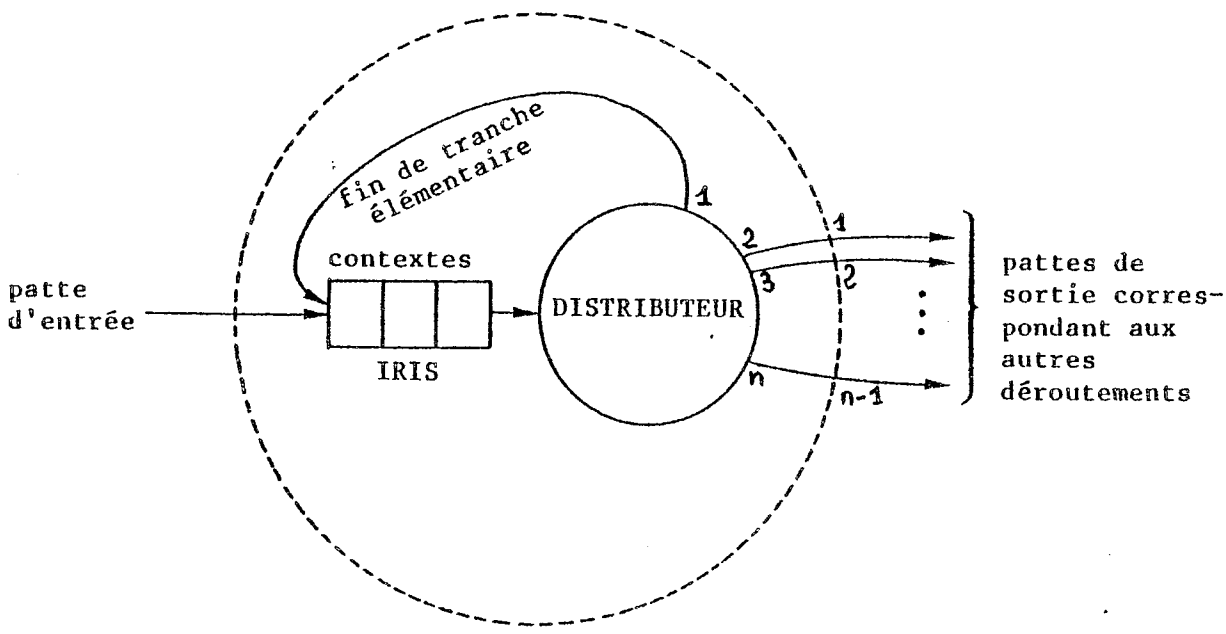


Le distributeur fonctionne donc exactement comme une unité qui serait bouclée sur elle-même, et pour illustrer cette similitude de fonctionnement nous ferons le parallèle suivant :

Unité	Distributeur
File d'attente de messages	File d'attente de contextes IRIS
ATTEND	Chargement d'un contexte IRIS sur un processeur
Traitement d'un message	Exécution par le processeur du programme défini par le contexte IRIS
RENVOI	Récupération du contexte IRIS
Code de sortie	Fin de tranche de temps élémentaire

La fin d'une tranche de temps élémentaire n'est cependant pas le seul cas où le distributeur retire la ressource unité centrale à un processus. Il le fait également pour tous les appels superviseurs et déroutements.

Poursuivant l'analogie avec le modèle de décomposition que nous avons présenté, nous dirons que le noyau de multiplexage de l'IRIS 80 se comporte comme une machine abstraite ainsi définie :



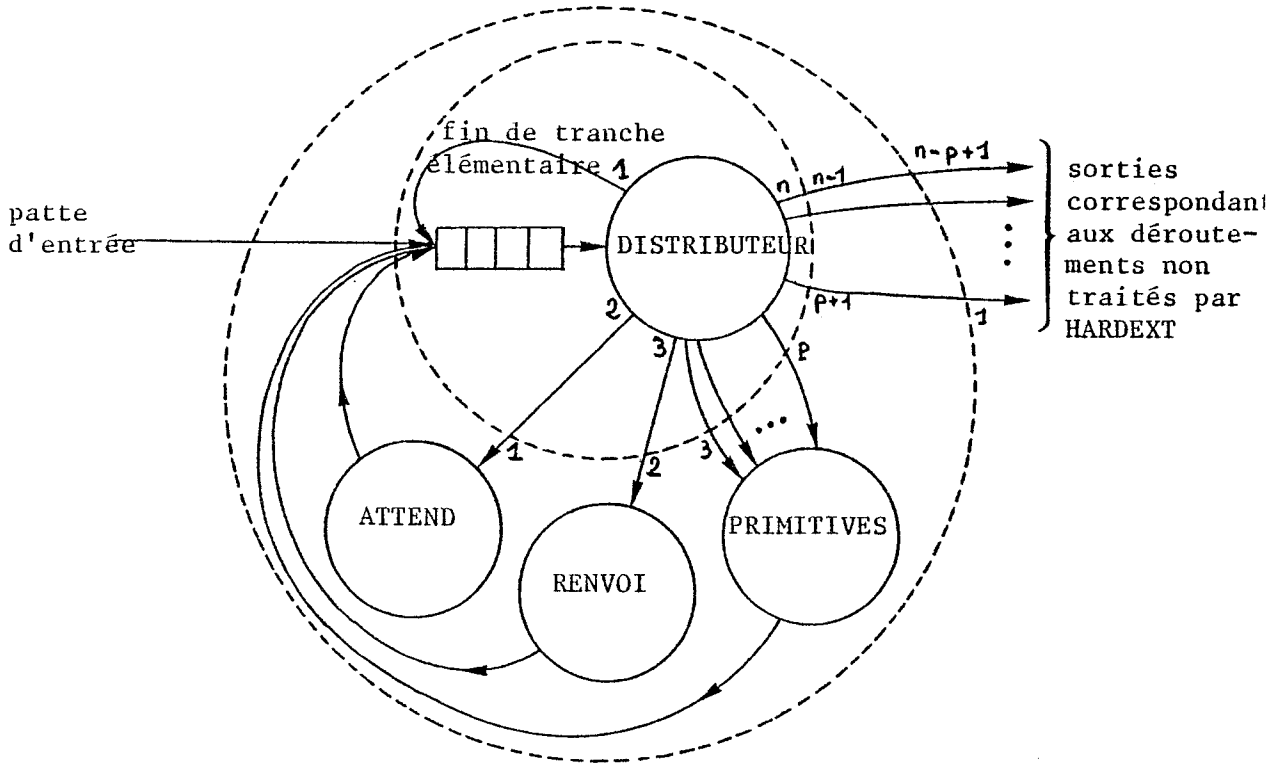
Seul le déroutement "fin de tranche" élémentaire est donc rebouclé sur le distributeur.

II.3.2. La Machine IRIS

Toutes les primitives que nous avons présentées au paragraphe précédent sont exécutées par le noyau HARDEXT de façon synchrone, c'est-à-dire que le contrôle n'est rendu au processus appelant qu'après exécution de la primitive.

Chacune de ces primitives peut alors être modélisée par une unité particulière comme nous l'avons fait d'ailleurs pour les primitives ATTEND et RENVOI. Pour simplifier nous n'ajouterons aux unités ATTEND et RENVOI déjà définies qu'une seule unité regroupant les traitements correspondant à toutes les autres primitives. Nous appellerons cette unité "PRIMITIVES". On trouvera au Chapitre V une modélisation plus détaillée du noyau HARDEXT.

L'appel des différentes primitives par les processus primitifs se fait par un certain nombre d'appels superviseurs : $CAL_{i,j}$. Tout se passe en définitive comme si les processus primitifs étaient exécutés par une machine abstraite ainsi constituée :



Nous appelons cette machine "Machine IRIS".

Les sorties numéro 1 à n-p+1 correspondent aux déroutements ou appels superviseurs non traités par HARDEXT. Dans la pratique, ces sorties ne sont jamais empruntées par les processus primitifs. Si elles le sont, il s'agit d'une erreur de programmation : par exemple les processus primitifs ne sont pas supposés faire des fautes de pages, ni exécuter des instructions inexistantes.

La patte d'entrée de la machine IRIS correspond à l'arrivée d'un nouveau processus dans l'état activable. Cette entrée n'est empruntée par les processus primitifs que lors de leur création. Par la suite, ils restent toujours activables.

La machine IRIS ainsi définie est répertoriée dans notre modèle comme le sont les unités ATTEND et RENVOI. Il est donc possible a priori de créer sur cette machine d'autres processus que les processus primitifs et de l'inclure dans d'autres machines abstraites. Mais la machine IRIS ne peut pas être mise ainsi à la disposition des utilisateurs sans que certaines précautions soient prises. Ces précautions concernent notamment la définition des contextes IRIS soumis en entrée et c'est ce que nous allons développer maintenant.

II.3.3. Utilisation de la machine IRIS

Les contextes IRIS définissent les programmes qui sont exécutés par la machine IRIS. Rappelons brièvement leur constitution :

- valeurs de 16 registres généraux
- valeurs de 8 registres de base
- valeur d'un double-mot d'état-programme
- description d'un espace virtuel mode C.

Le double mot d'état programme détermine les modes d'exécution et d'adressage des instructions IRIS 80. Sa valeur doit être en partie figée dans notre modèle (mode C esclave), ce qui ne peut être garanti que par un contrôle fait par HARDEXT sur la valeur des messages soumis en entrée de la machine IRIS.

D'autre part, la gestion des espaces virtuels est réservée dans notre projet à des unités de la machine MAS. Ces unités doivent donc pouvoir contrôler systématiquement les descriptions d'espace virtuel situées dans les messages soumis en entrée de la machine IRIS. Cela peut être réalisé grâce à l'inclusion de la machine IRIS dans la machine MAS. Mais cette solution est coûteuse puisqu'alors toute transition vers la machine IRIS doit être préalablement interceptée par une unité de MAS.

Les deux problèmes que nous venons d'évoquer peuvent être résolus de façon simple et efficace si l'on considère que les contextes IRIS sont des données locales de la machine IRIS, bien que rémanentes aux processus. Le noyau HARDEXT doit donc protéger ces données en même temps qu'il permet le multiplexage de la machine IRIS. Nous allons décrire maintenant les mécanismes liés à cette protection.

Les contextes IRIS sont conservés par HARDEXT lorsque les processus correspondants ne sont pas actifs. Lorsqu'un processus est créé sur une machine incluant la machine IRIS, le créateur peut spécifier un contexte IRIS initial pour ce processus :

```
CREER PROCESSUS (identité d'une machine
                 , message initial
                [, contexte IRIS]).
```

Notons qu'un contexte IRIS peut être spécifié de la même manière dans la primitive CREER UNITE.
L'unité "PRIMITIVES" est donc chargée d'emmagasiner les contextes IRIS nouvellement créés.
Le distributeur pour sa part est chargé du chargement, puis de la sauvegarde de ces contextes lorsqu'il active puis désactive les processus.
Les messages reçus en entrée de la machine IRIS et ceux qui en sortent, ainsi que les messages échangés entre les unités de la machine IRIS, ne sont donc pas des contextes IRIS.
Ces derniers sont donc bien protégés par le noyau HARDEXT. Mais "protection" ne signifie pas "tout accès interdit".
Si l'on permet d'inclure la machine IRIS dans une machine abstraite, il faut en même temps permettre que les unités de cette machine abstraite puissent lire et modifier certaines parties des contextes IRIS.

Nous avons défini deux mécanismes permettant cet accès :

a) Le contexte IRIS d'un processus non actif est conservé par le noyau ; il peut être lu ou partiellement modifié par les deux primitives suivantes :

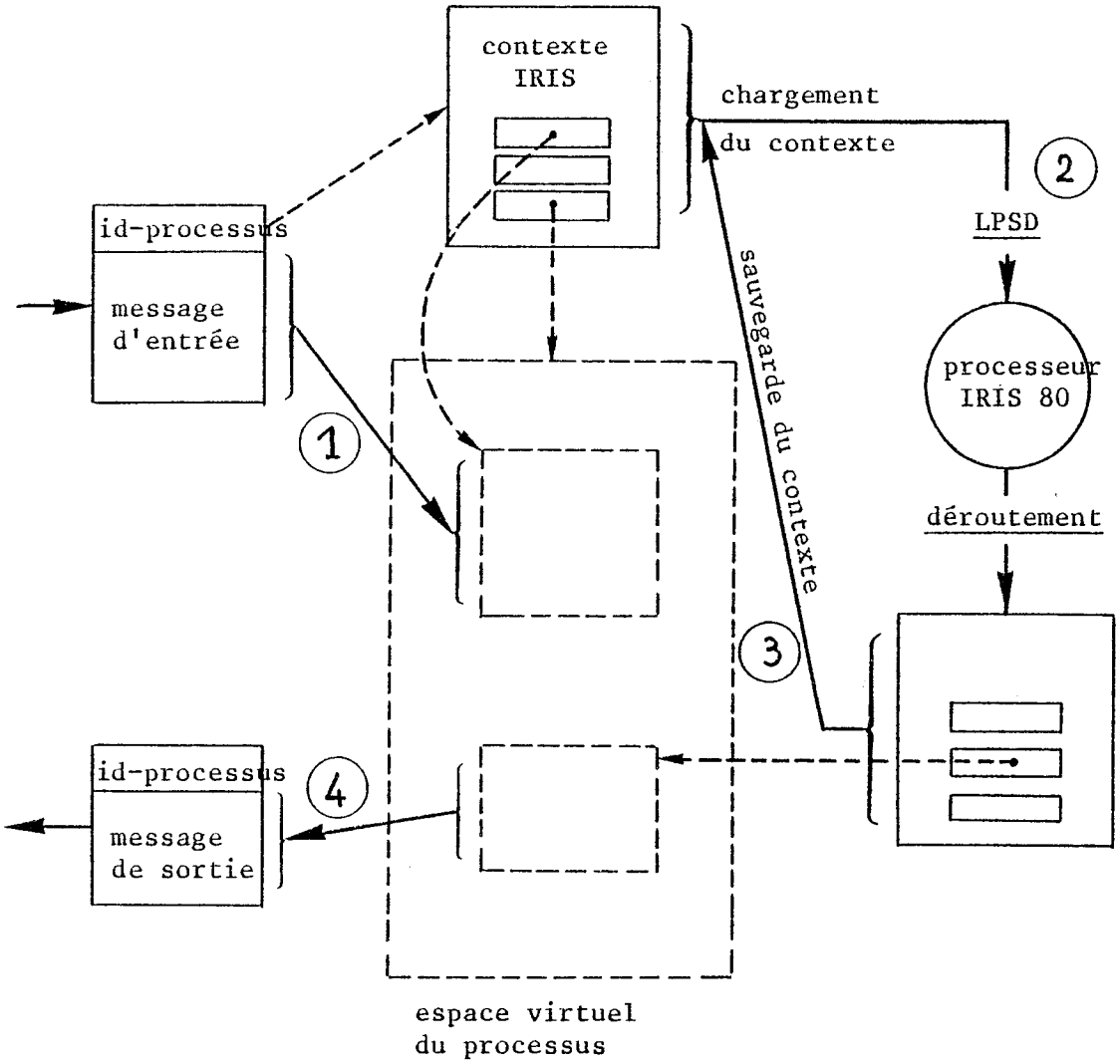
LIRE-IRIS (identité d'un processus).

Le contexte IRIS du processus indiqué est fourni en retour

MODIF-IRIS (identité d'un processus; contexte IRIS).

Après contrôle, les valeurs fournies en paramètre remplacent le contexte IRIS du processus désigné.
Ces primitives sont exécutées par l'unité "PRIMITIVES".

b) Lorsqu'un processus est pris en compte par le distributeur, le message associé est recopié dans l'espace virtuel du processus. Inversement lorsque le processus sort du distributeur par certains déroutements, notamment les appels superviseurs, un message est fabriqué à partir d'une zone de l'espace virtuel du processus. L'adresse et la longueur de cette zone sont déterminées au moyen d'une norme de programmation très précise que nous avons déjà évoquée lors de la présentation des primitives ATTEND et RENVOI (§ 2.12). Ce mécanisme définit comment est réalisée la communication par messages dans la machine IRIS. On peut le schématiser de la façon suivante :



(1) Recopie du message d'entrée, sans l'en-tête, dans l'espace virtuel du processus

(2) Chargement du contexte IRIS sur un processeur IRIS 80

(3) Sauvegarde du contexte IRIS du processus

(4) Fabrication du message de sortie du distributeur

Les flèches en pointillé représentent les pointeurs utilisés par le distributeur pour effectuer ces opérations.

Remarque : Comme nous l'avons signalé, les opérations (1) et (4) ne sont pas effectuées pour tous les déroutements.

A toutes les primitives présentées au paragraphe 2, nous avons associé des appels superviseurs. Par ce moyen, un programme exécuté par la machine IRIS peut donc appeler ces primitives. Les paramètres sont communiqués au moyen des mécanismes que nous venons de décrire.

II.3.4. Résumé

Le modèle de décomposition permet de prendre en compte la structure même du noyau HARDEXT par la définition d'un certain nombre d'unités correspondant aux fonctions réalisées par HARDEXT :

- le multiplexage des processeurs IRIS 80 est assuré par l'unité DISTRIBUTEUR
- les mécanismes d'extension sont réalisés par les unités PRIMITIVES ATTEND et RENVOI.

Le noyau HARDEXT peut alors être modélisé par une machine abstraite constituée de ces unités et que nous appelons machine IRIS. Cette machine est la machine de base de notre modèle. Elle définit le langage de niveau 0 qui est le langage IRIS 80 mode C esclave, ainsi que les possibilités de sa propre extension et les mécanismes permettant de la mettre en oeuvre.

II.4. QUELQUES ASPECTS PRATIQUES

Nous avons présenté au paragraphe 2 un certain nombre d'outils permettant d'une part de construire et d'assembler les éléments d'une architecture de système ou de machine, et d'autre part de créer à un quelconque niveau d'une telle architecture des activités parallèles qu'on appelle des processus.

Nous allons nous intéresser maintenant à quelques-uns des aspects opérationnels du modèle de décomposition.

En premier lieu nous décrirons les moyens mis en place au niveau de la réalisation pour protéger les architectures contre la malveillance ou l'erreur. Puis nous aborderons les problèmes liés au mauvais fonctionnement éventuel d'un élément d'une architecture : processus unité ou machine. Nous verrons quels types d'erreurs peuvent être détectés et nous proposerons un mécanisme visant à limiter leur propagation sinon à les corriger.

Pour terminer, nous examinerons le modèle de décomposition sous l'angle des ressources logiques définies par les machines abstraites.

Nous décrirons notamment le service de calcul automatique des consommations effectué par le noyau HARDEXT pour chaque élément d'une architecture. Nous insisterons en particulier sur une caractéristique essentielle de la méthode de calcul utilisée qui est de prendre en compte la relation d'interprétation induite par les niveaux de machines abstraites.

II.4.1. Protection de la structure

Cette protection s'effectue de deux façons différentes :

II.4.11. Désignation des éléments

Chaque élément (processus, unité, machine) est accessible par l'utilisateur au moyen de son nom unique. Cette désignation ne reflète pas le niveau d'abstraction de l'élément et cela correspond au choix que nous avons fait que tous les éléments définis à un instant donné doivent pouvoir être accessibles à tous les niveaux.

Ce nom-unique est suffisamment volumineux pour que la probabilité de le fabriquer par erreur ou malveillance soit très faible (elle est égale à $1/2^{64}$).

II.4.12. Pouvoirs

L'utilisation de chaque primitive HARDEXT est contrôlée par un pouvoir. A chaque processus est associé un ensemble de pouvoirs qui définissent donc les primitives HARDEXT que ce processus peut appeler.

L'attribution de pouvoirs à un processus s'effectue à la création de ce processus, mais ils peuvent être modifiés par une primitive spéciale : MODIF-POUVOIR. Il existe bien sûr un pouvoir associé à cette primitive.

Ce mécanisme est une généralisation du mode maître-esclave.

II.4.2. Défauts de fonctionnement

De même que les constituants d'une architecture matérielle sont sujets à des pannes, de même les éléments que nous avons présentés peuvent avoir des défauts de fonctionnement. Ces éléments sont les processus, les unités, les machines et le noyau de communication.

Nous supposerons cependant que le noyau HARDEXT est exempt de ces défauts car il faut bien qu'il y ait au moins quelques éléments sûrs, et précisément dans notre modèle, c'est lui qui est chargé de détecter les erreurs et qui permet de les prendre en compte.

II.4.21. Détection des erreurs

Cette détection est faite à deux niveaux.

a) A la création de chaque élément (processus, unité, machine), l'unité PRIMITIVES effectue un certain nombre de vérifications dont voici quelques exemples :

- l'automate de transition d'une machine ne doit pas comporter d'état indéfini
- les connexions déclarées dans un automate de transition doivent être correctes : les éléments référencés doivent avoir être créés et leurs pattes d'entrée et de sortie qui sont utilisées doivent avoir été déclarées

. un processus ou une unité ne peuvent être créés que sur une machine déjà créée.

On notera que bon nombre de contrôles qui sont effectués à la création des unités ou machines pourraient être différés jusqu'au moment où ces éléments sont effectivement utilisés, c'est-à-dire au cours de l'exécution des processus.

Nous n'avons pas développé ces possibilités dans le projet puisque nous avons choisi d'imposer que tout élément référencé ait été préalablement défini. Ce choix est relativement contraignant pour l'utilisateur puisqu'il l'oblige à respecter un certain ordre dans la construction des éléments d'une architecture.

Mais ce choix a l'avantage de faciliter la mise en oeuvre des contrôles et nous pensons qu'il est réaliste. En outre, il ne met pas en cause la modularité du modèle : tout élément peut être redéfini dynamiquement pourvu que le nombre des pattes d'entrée et de sortie reste le même.

b) Une architecture "correcte" ayant été définie, il est nécessaire de vérifier que chaque élément se comporte conformément à la définition qui en a été faite.

Ce contrôle est effectué chaque fois qu'un message est interprété par le noyau de connexion (les unités ATTEND et RENVOI). L'en-tête du message est analysé et doit permettre de déterminer sur quelle file d'attente le message doit être acheminé. Au cours de cette analyse, les erreurs suivantes peuvent être détectées :

. en-tête fantaisiste : le noyau ne peut pas déterminer à quel processus le message est associé : on diagnostique un mauvais fonctionnement de l'unité ayant émis le message
. code de sortie non prévu dans la déclaration de l'unité : on diagnostique un mauvais fonctionnement de l'unité émettrice
. élément non défini : l'unité ou la machine référencée dans l'automate de transition sont absentes (parce qu'il a été détruit)
. connexion non prévue : la matrice de transition ne prévoit pas de transition pour le code de sortie fourni par l'unité

le processus se retrouve dans un état indéfini. On diagnostique un mauvais fonctionnement de l'automate, donc de la machine associée

- le processus sort d'une unité dans laquelle il n'était pas sensé se trouver. On diagnostique un mauvais fonctionnement de l'unité ayant émis le message.

II.4.22. Prise en compte des erreurs

Il est difficile pour le noyau de savoir précisément comment doit être traitée chaque erreur. Il appartient en fait au concepteur de l'application de définir ce traitement car lui seul connaît a priori les fonctions associées à chaque élément de l'architecture. Nous avons donc défini un mécanisme de prise en compte des erreurs décrites plus haut qui est le suivant :

a) A chaque machine est associée une unité "opérateur". Quand le noyau de connexion détecte un mauvais fonctionnement d'une machine, il fabrique un message conventionnel décrivant la panne et le dépose en entrée de l'unité opérateur de cette machine.
L'unité "opérateur" peut être définie par l'utilisateur à la création de la machine :

CREER MACHINE (matrice de transition
[,unité-opérateur]).

Par défaut, une unité "opérateur" prédéfinie est associée à la machine.

b) A chaque processus est associée une unité "gérant". Cette unité récupère d'une part les messages de fin d'exécution du processus (destruction ou transition finale de la machine sur laquelle le processus a été créé) et d'autre part des messages conventionnels fabriqués par le noyau de connexion lorsque le processus est en difficulté (par exemple se retrouve dans un état indéfini) ou en faute (par exemple c'est le processus d'une unité et il a émis un message incorrect).
L'unité gérant peut être définie par l'utilisateur à la création du processus.

CREER PROCESSUS (machine
,message initial
[,unité-gérant])

Par défaut, l'unité "gérant" d'un processus est l'unité "opérateur" de la machine support.
Signalons d'autre part qu'un processus en faute ou en difficulté est placé automatiquement par le noyau de connexion dans un mode spécifique qu'on appelle "mode panne". Dans ce mode, la progression du message associé au processus est stoppée. Il peut tout au plus passer dans un état "transition" s'il était dans l'état "interprétation" au moment de la panne.

c) A chaque unité est associé un "gérant" qui est donc le "gérant" commun de tous les processus de cette unité. Il peut être défini par l'utilisateur à la création de l'unité

CREER UNITE (machine
,message initial
,degré de multiplication
[,unité gérant])

Par défaut, le gérant d'une unité est l'unité opérateur de la machine support.

Les conditions de mauvais fonctionnement d'une unité sont alors traitées de la façon suivante :

- le processus qui a émis le message "erroné" est considéré comme en faute : il est mis dans le mode "panne" et un message conventionnel est envoyé par le noyau au "gérant" de l'unité associée
- un autre message conventionnel est fabriqué par le noyau de connexion et envoyé à l'unité opérateur de la machine supportant l'unité.

Notons qu'il peut sembler plus logique a priori que ce message d'erreur soit envoyé à l'opérateur de la machine contenant l'unité en faute plutôt qu'à l'opérateur de la machine supportant l'unité en faute. Mais une unité peut appartenir à plusieurs machines et il n'est pas toujours possible de déterminer à laquelle il faut se référer. C'est notamment le cas lorsque l'en-tête du message émis par une unité est totalement fantaisiste. Aussi se réfère-t-on en définitive à la machine supportant l'unité

II.4.23. Traitement des erreurs

La prise en compte des erreurs par le noyau se traduit par l'envoi de messages conventionnels "d'erreur" aux unités "opérateur" et "gérant", et dans certains cas par la mise en "mode panne" d'un processus. Il appartient à ces unités de décider quelle suite elles doivent donner aux erreurs signalées. Pour effectuer leur travail, elles disposent de primitives spécifiques de consultation et de modification de l'état d'un processus, de l'activité d'une unité ou d'une machine, et de la composition du système. Citons à titre indicatif quelques-unes de ces primitives :

LIRE-MESSAGE (id-processus) → message associé au processus

MODIF-MESSAGE (id-processus, nouveau message)

LIRE-ETAT (id-processus) → pile d'exécution du processus

MODIF-ETAT (id-processus, nouvel état d'automate)

LIRE-PROCUNIT (id-unité) → identité des processus de l'unité

LIRE-DESCENDANCE (id-machine) → processus et unités supportés par la machine

LIRE-DEFINITION (id-machine) → structure de la machine :

- matrice de transition
- unité opérateur.

Le traitement d'erreur effectué par les unités "opérateur" et "gérant" peut alors se solder de deux façons différentes :

- soit l'erreur peut être corrigée. S'il y avait un processus en "mode panne", il peut être relancé au moyen de la primitive RETRO-PANNE (id-processus)
- soit l'erreur est grave et l'élément incriminé (processus, unité ou machine) doit être détruit.

La destruction d'un élément est une opération à utiliser avec précaution. Outre les problèmes que cela peut poser au niveau de la gestion de ressources, la destruction d'un élément peut entraîner dans certains cas une cascade de destructions à plus ou moins brève échéance.

Par exemple la destruction d'une machine entraîne la destruction de tous les processus et unités supportés par cette machine. Rappelons au passage que les "gérants" concernés en sont avertis par message. Les machines

qui contiennent les unités ainsi détruites risquent alors de se trouver à leur tour en panne ...

Nous dirons pour résumer qu'un élément est d'autant plus crucial dans une architecture qu'il est situé plus bas dans la hiérarchie des niveaux de machines : il est clair que si la machine IRIS est en panne tout l'édifice s'en ressent.

II.4.24. Autres erreurs

Les conditions d'erreur que nous avons citées plus haut déterminent un mauvais fonctionnement structurel d'une machine ou d'une unité et il appartient logiquement au noyau de connexion de les détecter.

Mais il existe bien sûr d'autres causes qui peuvent entraîner un mauvais fonctionnement d'une unité ou d'une machine. Sans en faire ici un inventaire exhaustif, qui serait trop long, nous en citerons quelques-unes. Nous indiquerons le cas échéant à quels domaines elles se rattachent.

- Mauvaise spécification fonctionnelle d'un élément. En d'autres termes, une machine ou une unité n'exécutent pas la fonction que l'on attend d'elle. Ce type d'erreur ne peut être détecté par le noyau ; il résulte manifestement d'une erreur de conception ou de réalisation.

- Mauvaise spécification des interfaces. Ce genre d'erreur peut être détecté par un mécanisme de contrôle automatique des types à la sortie et à l'entrée des unités. Une partie de ce contrôle peut être fait statiquement lors de la constitution d'une machine par la vérification de concordance des types sur les connexions indiquées. Un contrôle dynamique par le noyau de connexion nécessite de conserver à l'exécution une description des types de messages émis et reçus par les unités.
Ces mécanismes n'ont pas été développés dans notre projet.

- Blocage d'un élément. Au niveau externe, cela se traduit par le fait qu'une unité ne rend pas les messages qui lui sont transmis. Au niveau interne du comportement d'une telle unité, il peut y avoir, mis à part un "bug" qui fait boucler un programme, un interblocage entre les éléments de la machine support. En tout état de cause, ce genre de problèmes relève d'une mauvaise politique de répartition de ressources ou d'un mauvais contrôle de l'application de cette politique.

Ce problème peut être résolu en fixant pour chaque unité une borne maximum de la durée de rétention d'un message. Cette solution n'a cependant été mise en oeuvre dans notre projet que pour les unités construites sur la machine IRIS.

II.4.3. Comptabilisation des ressources

Un service complet de calcul des consommations de ressources est assuré pour chaque élément d'une architecture et chaque processus durant toute son existence.

Ce service comporte donc deux aspects correspondant à deux besoins essentiels que l'on rencontre dans l'exploitation d'un système :

- mesure de la consommation faite par chaque composant
- facturation.

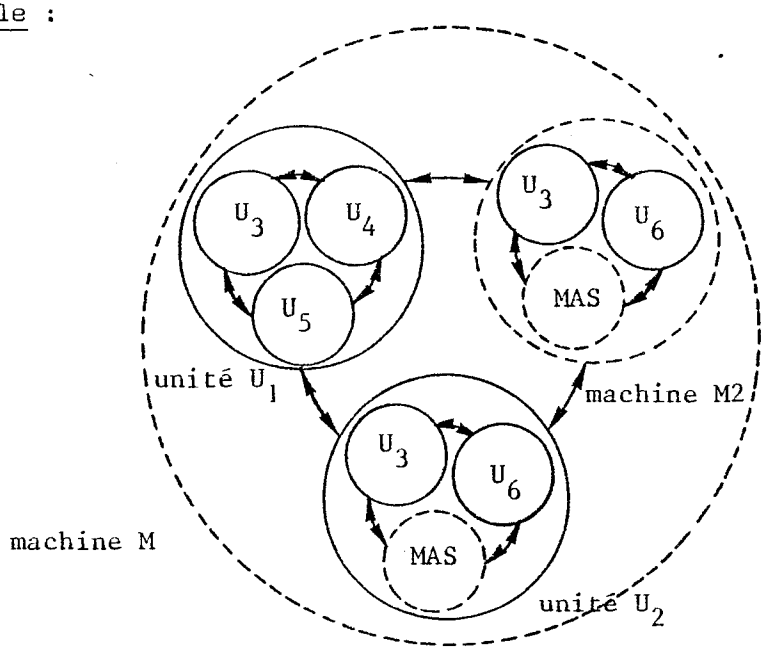
II.4.31. Mesures

Une architecture étant constituée de niveaux de machines abstraites, il paraît intéressant de considérer que les ressources consommables par un élément quelconque sont les constituants de la machine sur laquelle cet élément est construit. La consommation de chaque ressource peut alors être mesurée en nombre d'appels au constituant correspondant.

On notera par exemple que c'est souvent de cette façon que sont mesurées les consommations d'entrées-sorties.

Le mécanisme de connexion dynamique fournit un moyen commode de mettre en oeuvre cette méthode de calcul des consommations : chaque fois qu'un processus effectue une transition, le noyau met à jour les blocs de consommation du processus ainsi que des éléments impliqués dans cette transition : unités et machines.

Exemple :



La machine M est constituée des unités U1 et U2 et de la machine M2.
La machine M2 est constituée des unités U3 et U6 et de la machine MAS.
L'unité U2 est réalisée sur la machine M2.
L'unité U1 est réalisée sur une machine M1 constituée des unités U3, U4 et U5.
Les unités U3, U4, U5 et U6 sont réalisées sur la machine MAS.

Si l'on utilise la méthode suggérée plus haut, la consommation d'un processus P exécuté par la machine M s'exprime alors par la mesure suivante :

- nombre d'appels à l'unité U1 effectués par P
- nombre d'appels à l'unité U2 effectués par P
- nombre d'appels à l'unité U3 effectués par P
- nombre d'appels à l'unité U6 effectués par P
- nombre d'appels à chacune des unités de la machine MAS effectués par P.

La consommation d'une unité est la somme des consommations des processus associés à cette unité. Par exemple, la consommation de U1 s'exprime par la mesure suivante :

- nombre d'appels à l'unité U3 effectués par les processus de U1
- nombre d'appels à l'unité U4 effectués par les processus de U1
- nombre d'appels à l'unité U5 effectués par les processus de U1.

Enfin, la consommation d'une machine est la somme des consommations des processus exécutés par cette machine, y compris ceux des unités créées sur cette machine.

II.4.32. Facturation

Mais le service de calcul des consommations que nous voulons offrir doit également faciliter la mise en place d'un système de facturation.

On connaît la difficulté qu'il y a à exprimer dans une même unité, qu'on appelle "point comptable" ou "unité de facturation", des mesures de consommation relatives à des ressources différentes. Cette difficulté croît rapidement avec le nombre de ressources, c'est-à-dire dans notre cas avec le nombre d'unités définies dans une architecture.

Si l'on s'en tient uniquement à la méthode de calcul décrite ci-dessus, on est alors contraint de fournir une estimation du coût moyen d'appel de chaque unité, ce coût étant exprimé en utilisant une unité de mesure commune que l'on aura préalablement définie. On voit que cette façon de facturer est très difficile à mettre en oeuvre et qu'en particulier elle s'accomode très mal de l'addition de nouveaux éléments dans une architecture.

Nous proposons donc de compléter la méthode de calcul suggérée plus haut en y intégrant un cumul automatique des consommations qui peut être résumé de la façon suivante :
lorsqu'une unité consomme des ressources pour le compte d'un processus qu'elle traite, la consommation correspondante est cumulée sur le compte du processus en même temps que sur le compte de l'unité et sur celui de la machine supportant l'unité. Ce cumul est effectué automatiquement à chaque transition.

L'utilisation itérative de ce cumul conduit à maintenir des informations comptables volumineuses pour chaque élément d'une architecture. Si l'on reprend l'exemple précédent, on aura pour le processus P :

- nombre d'appels à U1 effectués par P
- nombre d'appels à U2 effectués par P
- nombre d'appels à U3 effectués par P
- nombre d'appels à U6 effectués par P
- nombre d'appels à chaque unité de la machine IRIS effectués par P
- nombre d'appels indirects à U3
- nombre d'appels indirects à U6

- nombre d'appels indirects à U4
- nombre d'appels indirects à U5
- nombre d'appels indirects à chaque unité de la machine MAS.

Mais l'exemple précédent montre également que l'appel à n'importe quel élément d'une architecture se traduit en définitive par un certain nombre d'appels aux unités de la machine MAS. Les unités de la machine MAS vont donc pouvoir constituer les ressources de base à partir desquelles une facturation va pouvoir être établie. Comme la machine MAS contient la machine IRIS, c'est en fait en termes d'appels aux unités de ces deux machines que les consommations peuvent être exprimées.

Le problème qui reste à résoudre est alors de fixer un coût moyen de traitement pour chaque type d'appel, ou bien alors de fournir aux unités de MAS un moyen de modifier explicitement les informations comptables lors de chaque appel.

C'est cette solution que nous choisissons. Les unités de MAS disposent donc de moyens spécifiques qui leur permettent de porter explicitement des mesures de consommation sur le compte des processus qu'elles traitent. Ces moyens sont en fait un accès direct aux blocs décrivant les consommations de chaque processus, ces blocs étant situés dans l'espace virtuel commun des processus primitifs. (N'oublions pas, en effet, que les unités de la machine MAS correspondent à des regroupements de processus primitifs).

Cette méthode permet en outre de comptabiliser des ressources de façon plus réaliste que par un nombre d'appels aux unités de MAS.

En effet, il n'y a pas correspondance biunivoque entre les unités de MAS et les ressources gérées par cette machine.
Par exemple, pour les entrées-sorties, chaque unité n'est pas associée à un périphérique particulier, ni même à un type de périphérique particulier mais plutôt à une certaine phase de déroulement d'une entrée-sortie.

II.4.33. Méthode de calcul des consommations

Afin de remplir le double objectif que nous nous sommes fixé, nous avons finalement combiné les différentes techniques de calcul et d'expression des consommations décrites jusqu'ici.

a) Ressources comptabilisées

- temps d'unité centrale
- fautes de page
- pages en mémoire
- entrées-sorties sur chaque type de périphérique géré par MAS
- appels superviseurs
- déroutements
- ATTEND
- RENVOI.

b) Méthode de calcul

A chaque élément d'une architecture, unité, machine, et à chaque processus on associe deux blocs contenant chacun des mesures de consommation exprimées dans les termes que nous avons cités, c'est-à-dire relatives à la machine MAS.
Le premier bloc décrit les consommations dites directes.
Le second bloc décrit les consommations dites indirectes.
Ces blocs sont remplis comme suit :

- Au niveau bas, le distributeur met à jour le compte du temps d'unité centrale de chaque processus qu'il traite. Cette mise à jour est faite chaque fois que le processus fait un déroutement, et elle a lieu dans le bloc des consommations directes du processus.
Notons que les processus primitifs sont également traités de cette façon.

- Au niveau de la machine MAS, comme on l'a vu, chaque unité met à jour, pour les ressources qu'elle gère, le bloc de consommation directe du processus qu'elle traite. Il appartient à chaque unité de déterminer à quelles phases du traitement elle doit effectuer cette mise à jour.

- Lorsqu'un processus effectue un RENVOI, ses deux blocs de consommations, directes et indirectes, sont cumulées dans le bloc de consommations indirectes du processus renvoyé (c'est-à-dire celui qui va faire une transition de sortie). Chaque bloc est de plus cumulé dans le bloc corres-

pondant de l'unité associée au processus qui a fait RENVOI, et aussi dans celui de la machine qui supporte l'unité. Puis les deux blocs de consommation du processus qui a fait RENVOI sont remis à zéro.

Notons que ce traitement est appliqué également aux processus primitifs, de sorte que l'unité centrale qu'ils consomment se trouve automatiquement facturée aux processus qu'ils traitent.

Cette méthode de calcul permet de combiner la facturation des processus "externes" et la mesure du "coût" de chaque élément d'une architecture ainsi que son degré de sollicitation (nombre de ATTEND).

On notera cependant que les mesures de facturation effectuées par cette méthode peuvent être faussées lorsqu'une unité multiplexe les traitements ou bien lorsqu'elle travaille pour son propre compte. Pour remédier à cet inconvénient, nous donnons aux unités la possibilité d'anticiper les opérations de cumul des consommations faites automatiquement par le noyau. C'est le but de la primitive FACTURER, qui recouvre exactement les opérations de cumuls et remise à zéro décrites plus haut pour la primitive RENVOI :

FACTURER (id-processus).

Le processus désigné en paramètre joue le rôle du processus client dans la primitive RENVOI ; c'est sur lui que sont cumulées les consommations du processus qui exécute cette primitive. Il appartient au programmeur d'une unité d'utiliser cette primitive à bon escient afin que le remède ne soit pas pire que le mal.

Citons enfin la primitive CONSOMMATION qui permet de connaître les consommations directes et indirectes d'un élément à un instant donné :

$$\text{CONSOMMATION} \left(\left[\begin{array}{l} \text{id-processus} \\ \text{id-unité} \\ \text{id-machine} \end{array} \right] \right)$$

Par défaut, le processus demandeur obtient ses propres consommations.

Chapitre III

Evaluation Critique du Modèle de Décomposition

Nous avons présenté au chapitre II des outils qui permettent de construire des applications informatiques à partir d'une décomposition basée sur la méthode d'abstraction-raffinement.

Dans ce paragraphe, nous allons essayer d'illustrer l'utilisation de ces outils sur quelques exemples. Ce faisant nous tenterons une évaluation qualitative du modèle sous chacun de ses deux aspects principaux qui sont l'abstraction et le raffinement.

Pour ce qui concerne la méthode d'abstraction, nous dégagerons un certain nombre de problèmes qui peuvent se poser dans la définition d'une machine abstraite. Nous proposerons le cas échéant plusieurs solutions et nous en profiterons pour établir certaines analogies avec les projets d'architectures multimicroprocesseurs proposés dans la littérature.

Nous ferons également le lien avec la décomposition en modules et nous montrerons qu'en fait nos "unités" et nos "machines" peuvent très bien correspondre à des modules d'accès aux données. Les deux méthodes de décomposition sont alors complémentaires et permettent d'apporter une solution commune et globale aux problèmes de contrôle et d'accès aux données.

Pour ce qui concerne la méthode de raffinement, nous essaierons de montrer que cette méthode induit certaines relations entre les langages définis à chaque niveau de machine abstraite. Le cas échéant nous suggèrerons un certain nombre de règles d'utilisation des outils qui sont proposés.

Quelques uns parmi les exemples que nous allons donner font référence à la machine MAS qui est la machine de base offerte aux concepteurs de système. Les ressources logiques, objets et opérations primitives, définies par cette machine seront présentées en détail au chapitre suivant, mais pour faciliter la compréhension des exemples donnés ici, nous allons présenter brièvement le langage de la machine abstraite MAS.

- Conservation de l'information :

L'unité de base de conservation de l'information est le segment : ensemble de pages de 512 mots de 32 bits ; le nombre de ces pages est limité à 256 par segment. Les segments sont implantés sur des disques qu'on appelle volumes MAS et qui sont des entités connues de l'utilisateur. L'ensemble

des volumes MAS constitue ce qu'on appelle la banque de segments. Elle est partagée par tous les utilisateurs, chaque segment étant accessible par tous les utilisateurs qui connaissent le nom unique qui le désigne, (cf. paragraphe IV.1.2). Des primitives MAS permettent de créer et détruire les segments.

- Manipulation de l'information :

La machine MAS contient la machine IRIS. Par rapport à la machine réelle IRIS 80 celle-ci est réduite à l'exécution en mode esclave C. L'espace d'adressage des instructions est constitué de 128 blocs de 128 Kmots chacun. MAS prend en charge la pagination de ces espaces virtuels. Des primitives MAS permettent de charger des segments dans les blocs d'un espace virtuel. Pour chaque segment ainsi "lié" une copie temporaire est maintenue par MAS sur les disques de pagination. Des primitives MAS permettent de recopier un segment de la banque de segments où sa copie temporaire dans un bloc d'espace virtuel et vice-versa.

- Communication avec l'extérieur :

Tous les périphériques sont gérés par MAS suivant le même schéma, exception faite des unités de disques supportant la banque de segments :

- périphérique initialement dans l'état non alloué,
- demande d'allocation d'un périphérique ou d'un type de périphérique particulier au moyen d'une primitive MAS ; le seul paramètre de cette primitive est un nom symbolique pris dans un catalogue qui doit être bien sûr connu du demandeur. Un nom unique est fabriqué par MAS et retourné au demandeur en même temps qu'il est conservé par MAS. Le périphérique passe dans l'état alloué,
- demande d'entrée-sortie sur le périphérique selon un protocole dit d'appareil standard. Le périphérique est désigné uniquement par son nom unique dans la primitive associée à cette demande,
- libération du périphérique au moyen d'une primitive spécifiant le nom unique : le périphérique est de nouveau non alloué.

III.1. LES MACHINES ABSTRAITES

Rappelons ici un des objectifs du projet OURS : fournir des outils permettant de construire des architectures de machines et de systèmes aussi bien que les éléments constitutifs de ces architectures. Etant donné le développement rapide des microprocesseurs et son incidence prévisible sur l'architecture des machines, nous avons cherché à développer des outils qui puissent s'apparenter à ceux qui existent dans la réalité. Nous avons donc choisi de définir une machine abstraite comme un réseau d'unités communiquant par messages.

Ce choix laisse cependant une assez grande liberté de manoeuvre au concepteur d'une application lorsqu'il définit une machine abstraite.

Dans sa thèse [MAZ], G. MAZARE propose de classifier les projets d'architectures multi-microprocesseurs suivant trois critères qui sont :

- le contrôle
- les communications entre processeurs,
- spécialisation ou banalisation des processeurs.

Nous allons montrer dans ce paragraphe qu'un grand nombre d'architectures différentes peuvent être définies avec les outils que nous avons présentés. Pour ce faire nous utiliserons les critères de classification proposés par G. MAZARE comme critères de choix d'une architecture.

III.1.1. Le contrôle

Ce terme recouvre deux notions : d'une part la façon dont les différents éléments coopèrent et d'autre part la façon dont le travail est réparti entre ces éléments [MAZ]. Notons que ces deux notions se recouvrent dans le cas où une machine n'exécute qu'un seul travail à la fois, mais c'est loin d'être le cas général et de toute façon ce n'est pas le cas de nos machines abstraites ni même des unités.

III.1.11. Coopération entre les unités

Nous allons étudier pour commencer la façon dont les unités d'une machine abstraite peuvent coopérer à l'interprétation d'un programme. Puis nous étendrons cette étude au cas où une machine contient non seulement des

unités mais aussi des machines incluses.
Pour terminer nous aborderons le problème de la mise en oeuvre du parallélisme dans l'interprétation d'un programme.

III.1.111. Interprétation d'un programme par les unités d'une machine

Chaque machine abstraite définit un répertoire d'instructions et d'objets qui constitue le langage de programmation "machine" de cette machine abstraite.

Une machine abstraite apparaît donc comme l'interpréteur de langage qu'elle définit. C'est le fonctionnement d'un tel interpréteur que nous étudions ici.

Un message qui arrive sur une machine abstraite définit un programme. Ce programme est interprété par les différentes unités de la machine au fur et à mesure que le message chemine entre ces unités.

Un programme se compose d'instructions élémentaires et d'informations de contrôle décrivant l'ordre dans lequel ces instructions doivent être exécutées. Dans les informations de contrôle, nous classerons notamment la définition des points d'entrée du programme, la définition implicite ou explicite de séquences d'instructions, l'expression du parallélisme ou encore les instructions du type boucle ou branchement.

L'interprétation d'un programme par une machine comporte donc du point de vue du contrôle, les trois aspects suivants :

- le décodage des instructions,
- l'interprétation des instructions élémentaires,
- l'interprétation des informations de contrôle, que nous appellerons pour simplifier "séquencement".

Il ne fait pas de doute que les instructions élémentaires doivent être interprétées par les unités de la machine abstraite. Mais en ce qui concerne le décodage et le séquencement, le travail peut être plus ou moins réparti entre les unités. A notre avis, plusieurs solutions sont envisageables et correspondent à des choix d'architectures de machines faits à partir des éléments suivants :

a) la fonction de séquencement est concentrée dans une unité spécifique ou bien alors elle est répétée dans chaque unité,

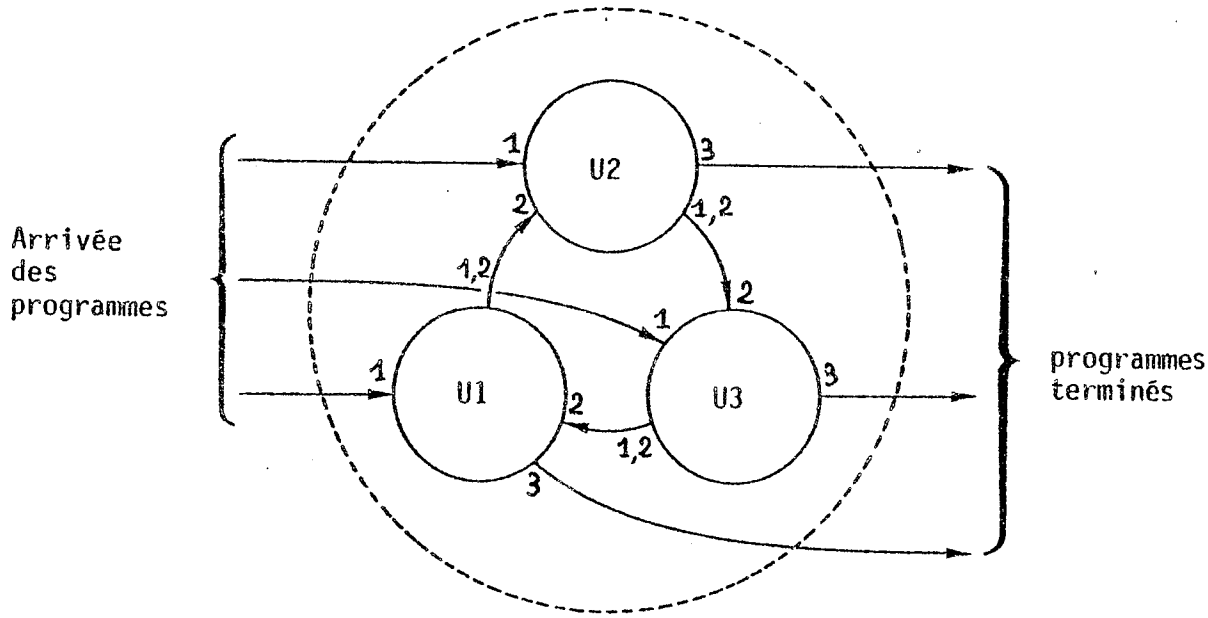
b) chaque message transporte la description complète d'un programme ou bien alors cette description est située dans une mémoire partagée par les unités.

c) la fonction de décodage est concentrée dans une unité ou bien elle est répétée dans chaque unité.

d) la distribution des messages est effectuée par une unité spécifique, ou bien c'est l'automate de transition qui assure cette distribution.

Nous allons illustrer par un exemple la façon dont ces différents éléments de choix peuvent être combinés.

Soit donc une machine M dont les unités sont organisées en anneau.



Nous supposons en outre que chaque unité exécute un algorithme incluant le séquencement et le décodage ; le séquencement est ici décentralisé.

```
Cycle:ATTEND;
si entrée = 1 alors   ⟨commentaire : ligne d'entrée de la machine⟩
   pointeur courant = début du programme;
sinon fin             ⟨commentaire : ligne d'entré n° 2 = programme en cours⟩
si code-instruction (pointeur-courant)≠code connu alors
      RENVOI (2);     ⟨commentaire : aucune exécution effectuée par l'unité⟩
fin
tant que code-instruction (pointeur courant) = code connu alors
         exécuter instruction (pointeur-courant);
         mise à jour (pointeur courant);
fin
```

```
si programme-terminé alors
   RENVOI (3);              ⟨commentaire : sortie de la machine⟩
sinon
   RENVOI (1);              ⟨commentaire : vers l'unité suivante⟩
fin
fin;
```

Une telle organisation peut se concevoir pour des microprocesseurs branchés sur un bus unique. Par ailleurs, dans un ordinateur "classique" les unités de contrôle des périphériques sont souvent organisées d'une façon analogue à celle-ci, mais les commandes d'entrée-sortie sont envoyées une par une sur le bus : la fonction de séquencement est concentrée dans le canal, mais le décodage est réparti dans les unités.

Dans le cas de notre modèle, plusieurs arguments peuvent être avancés pour un séquencement centralisé dans une seule unité. Nous en citerons trois :

- En premier lieu, il n'est pas souhaitable pour des raisons de protection que les données de séquencement et l'ensemble du programme soient accessibles par toutes les unités du programme : mettre le programme dans une mémoire commune à toutes les unités n'est possible que s'il y a une mémoire commune, et transmettre le programme en entrée dans les messages est une solution coûteuse et impraticable si ce programme est très volumineux.

- En second lieu, si l'on veut rajouter une unité dans une machine, il est toujours délicat d'être obligé de modifier préalablement le programme de cette unité pour y intégrer une nouvelle fonction ; ici le séquencement.

- Enfin, tant qu'on ne dispose pas de langage permettant de programmer de façon "répartie", il est plus facile de concentrer le traitement de chaque fonction dans un seul programme en utilisant les langages de programmation existants.

Nous allons illustrer ce dernier argument en reprenant l'exemple précédent. Dans l'algorithme que nous avons écrit, le problème du séquencement n'est pas entièrement résolu.

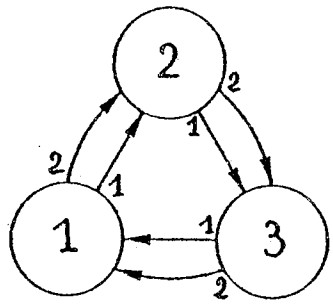
En effet, si dans un programme interprété par la machine M, il y a une instruction qui n'est reconnue par aucune des unités, alors le message associé tourne indéfiniment dans la machine sans espoir de sortie.

Pour remédier à cela, il y a deux solutions :

- la première consiste à modifier l'algorithme décrit de façon à ajouter aux données de séquencement (réduites ici au pointeur courant) une variable d'état que l'on gèrera de façon à éviter le bouclage infini.

- la seconde solution peut être apportée en définissant de façon ad hoc l'automate de transition. La sortie n° 2 des unités est prévue à cet effet : trois sorties n° 1 consécutives entraînent une sortie de la machine.
L'automate de transition qui était initialement simple devient alors compliqué.

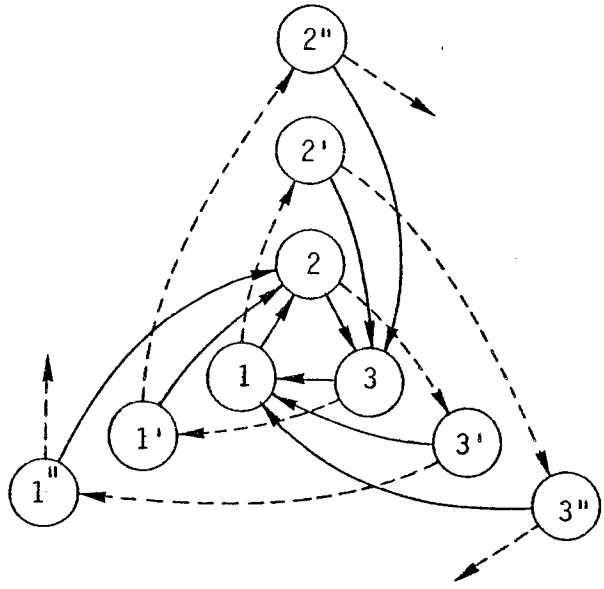
Automate initial



Pour simplifier nous n'avons pas représenté les sorties 3 correspondant à la fin d'exécution du programme, ni l'état final associé. Les états 1, 2 et 3 sont des états initiaux, ils correspondent respectivement aux lignes d'entrée des unités U1, U2 et U3.

Pour résoudre le problème du bouclage infini, nous introduisons pour chaque unité deux états supplémentaires.

L'automate de transition obtenu est le suivant :



Nous avons représenté en trait continu les transitions correspondant aux sorties n° 1 des unités (cas où des instructions ont été exécutées).

Les sorties n° 2 des unités 1", 2" et 3" doivent provoquer normalement une sortie de la machine du type " instruction inexistante.

Nous pensons avoir montré sur cet exemple que la prise en compte d'une partie du séquencement par l'automate de transition n'est pas chose facile à définir. En conséquence, nous conseillons de réaliser cette fonction par programme et de la concentrer dans une seule unité.

Par contre, la fonction de décodage peut être assez facilement répartie; en général chaque unité peut exécuter plusieurs instructions et cela nécessite donc un décodage systématique des commandes qu'elle reçoit. Le séquencement, s'il existe, n'échappe d'ailleurs pas à la règle puisqu'il doit interpréter les instructions de séquencement et envoyer les autres instructions se faire interpréter ailleurs.

C'est en définitive dans la distribution des messages aux unités que l'automate de transition permet le plus de souplesse.

Nous classerons les types de distribution possibles en deux grandes catégories :

- distribution en étoile
- distribution en anneau.

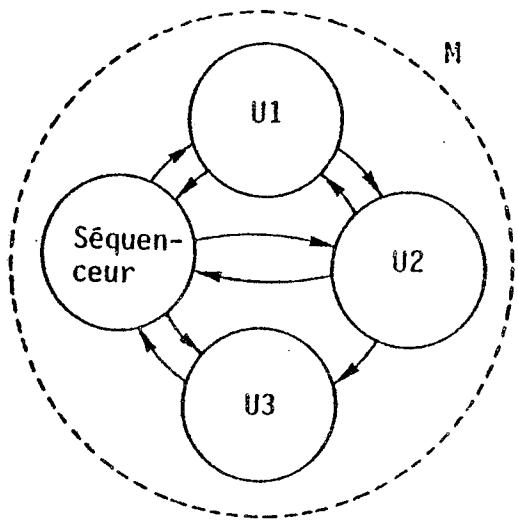
Toute distribution de messages est obtenue par combinaison de ces deux méthodes.

III.1.112. Cas des machines incluses

Lorsqu'on emploie les termes "programme" et "séquencement", il faut bien préciser à quel niveau d'observation on se place. Ce qu'on considère comme un programme à un certain niveau peut n'être plus qu'une instruction au niveau immédiatement supérieur.

Par exemple une opération d'entrée-sortie sur l'IRIS 80 se traduit par l'envoi d'un programme canal sur une unité d'échange. Au niveau de cette unité d'échange, il y a un programme, mais au niveau du programme IRIS 80 il y a seulement une instruction SIØ.

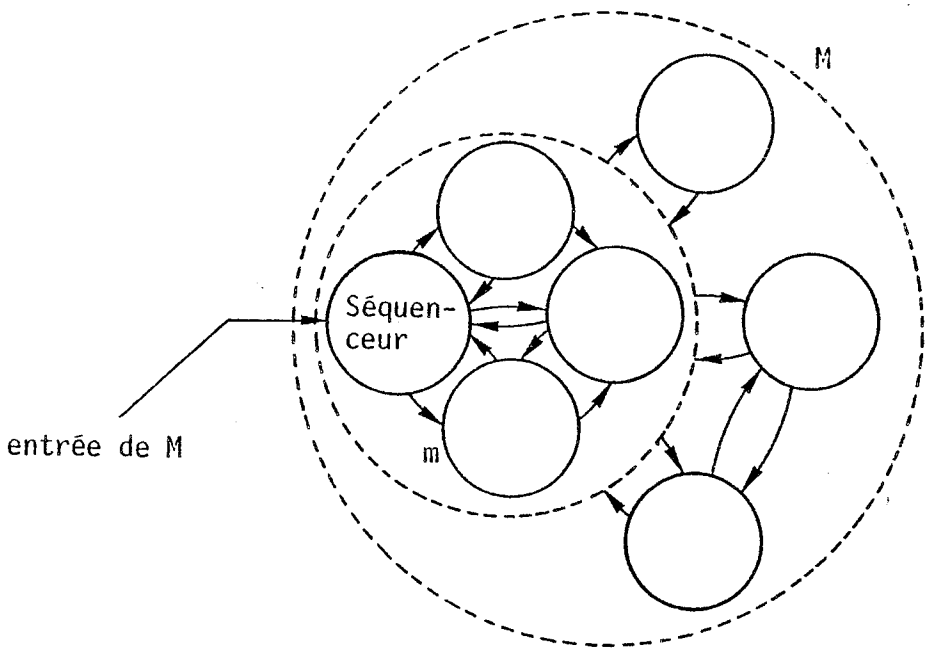
Chaque élément d'une machine semble donc contenir son propre séquencement. Dans le cas où le séquencement est centralisé, l'un de ces éléments doit effectuer en plus le séquencement du niveau supérieur. Plaçons nous dans ce cas et appelons cet élément le séquenceur de la machine.



Le séquenceur d'une machine M peut être une unité, mais ce peut être aussi une machine incluse.

Dans ce cas, il est clair que le séquenceur de la machine englobante est le même que celui de la machine incluse.

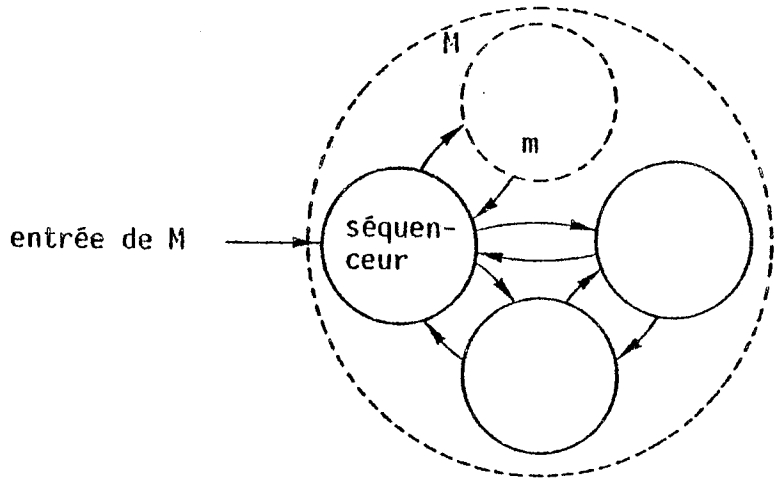
Dans ce type d'organisation, les unités de la machine englobante ne sont appelées que depuis la machine incluse et jamais depuis l'extérieur.



On verra que la machine MAS (ch. IV) est construite de cette façon à partir de la machine IRIS. Les unités de la machine englobante (MAS) ne sont appelées que depuis la machine incluse (IRIS), laquelle effectue le séquencement.

On verra également que les unités de la machine englobante servent aussi à enrichir le langage de la machine incluse (§III.2.1.).

Dans le cas où le séquenceur est une unité, c'est la machine incluse qui en définitive n'est jamais appelée directement de l'extérieur.



On verra dans l'exemple de la machine fichier (§ III.3.2) que les unités de gestion des "contextes d'accès fichier" assurent de cette façon le séquencement des programmes d'accès aux fichiers et filtrent les accès à la machine à enregistrements qui est incluse dans la machine à fichiers.

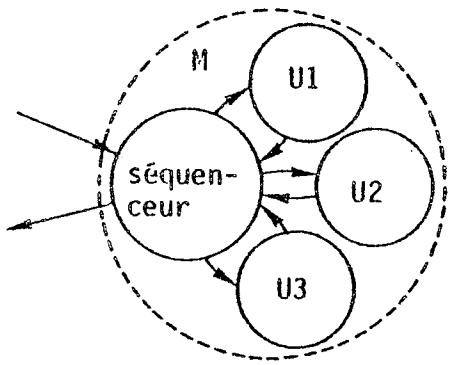
Nous reviendrons au paragraphe III.2.1. sur ces notions d'enrichissement et de filtrage.

III.1.113. Mise en oeuvre du parallèlisme

Nous avons dit que l'exécution d'un processus par une machine abstraite était purement séquentielle. Nous allons voir ici que cela n'empêche pas qu'une machine abstraite puisse exécuter en parallèle différentes parties d'un même programme.

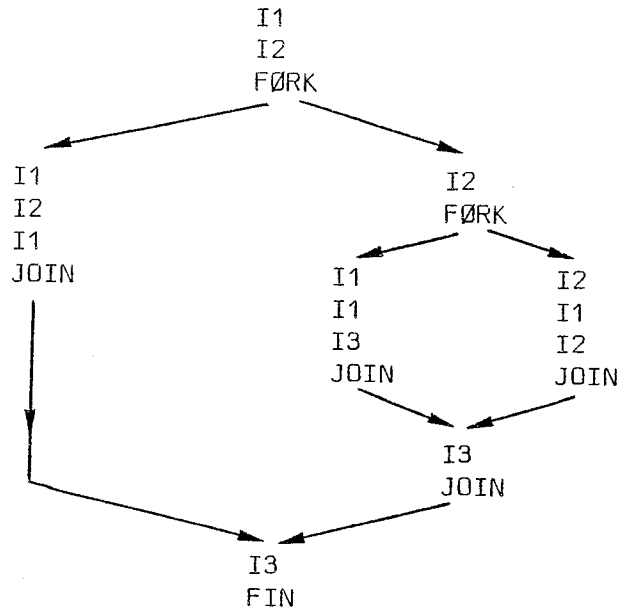
La solution est en fait assez simple : si des instructions d'exécution parallèle (ex FØRK, JOIN) figurent dans un programme, il appartient aux unités interprétant ces instructions de rendre ce parallèlisme effectif en créant plusieurs processus.

Exemple :
Soit une machine M composée de la façon suivante :



U1 : exécute l'instruction I1
U2 : exécute l'instruction I2
U3 : exécute l'instruction I3

Le séquenceur exécute en outre les instructions FORK et JOIN qui comme nous l'avons dit (§II.5.1111) sont des instructions de contrôle. Sur cette machine arrive un message décrivant le programme suivant :



Soit P le processus porteur de ce message. Nous supposerons que le séquencement est centralisé et que le programme reste dans la mémoire locale du séquenceur. Dans cette mémoire figurent également les données de séquencement associées à chaque processus en cours d'interprétation.

Les messages qui transitent vers les unités U1, U2 et U3 ne contiennent donc qu'une instruction et l'identité du processus intéressé.

Lorsque le séquenceur rencontre la première instruction FORK, il associe une des branches parallèles du programme du processus P, puis il crée un processus P' (processus fils) pour l'autre branche. Soit par exemple P pour la branche de gauche et P' pour celle de droite.

La création de P' s'effectue par l'appel CREER PROCESSUS ; la machine référencée est M, le message initial contient un pointeur vers la branche de droite du programme. Une entrée spéciale est donc prévue dans le séquenceur (et donc aussi dans la machine M) pour les processus qu'il créé lui-même.
Par ailleurs les données de séquencement associées à chaque processus contiennent les identités des processus fils (dans notre exemple, il n'y a jamais plus d'un processus fils).

Les processus P et P' associés aux deux branches principales du programme sont alors interprêtés en parallèle.

Lorsque le séquenceur rencontre la deuxième instruction FORK, il opère de façon analogue : il associe par exemple la sous-branche de gauche à P' et il crée sur M un processus P" pour exécuter la sous-branche de droite.

Voyons maintenant comment est exécutée l'instruction JOIN. Les données de séquencement de chaque processus contiennent l'identité des processus fils et du processus père. Dans notre exemple, P a un fils P', lequel a un fils P" ; P' et P" ont chacun un seul père.

L'exécution de l'instruction JOIN se fait alors selon les principes suivants :

- un processus père attend que tous ses fils aient fait JOIN avant de continuer son travail,
- un processus qui est uniquement fils est tué par le séquenceur lorsqu'il fait JOIN. Si ce fils n'avait pas de frère (ou n'en avait plus), alors le processus père est réactivé.

fin de l'exemple.

Peu de langages permettent d'exprimer du parallèlisme, sauf en ce qui concerne les entrées-sorties qui sont alors dites "asynchrones". Dans la mesure où les futures machines seront des réseaux de processeurs (spécialisés ou non), on peut penser que les futurs langages de programmation permettront de plus en plus d'exprimer le parallèlisme dans les programmes.

III.1.12 Multiprogrammation

Nous disons qu'il y a multiprogrammation lorsqu'il y a interprétation simultanée de plusieurs programmes (plusieurs processus) par une machine abstraite.

Cette simultanéité peut être mise en oeuvre de trois façons différentes dans une machine abstraite :

a) Par le simple fait qu'une machine se compose de plusieurs unités qui travaillent en parallèle. Pour exécuter un programme donné, une seule unité est active à la fois. Cependant, à un instant donné, toutes les

unités peuvent être actives simultanément parce qu'elles traitent des programmes différents. Il y a alors simultanéité au niveau de la machine abstraite.

b) Par le fait qu'une unité peut être multipliée, c'est-à-dire regrouper plusieurs processus serveurs. Les processus serveurs traitent alors des messages associés à des processus "clients" différents et il y a simultanéité de traitement au niveau de l'unité.

c) Par le fait qu'une unité peut multiplexer les traitements qu'elle effectue. Nous avons donné au début du chapitre II un exemple de multiplexage : la gestion d'une réserve de mémoire. Dans le cas du multiplexage, la simultanéité résulte d'une décision prise par l'unité de suspendre momentanément un processus client au profit d'un autre. Il y a alors simultanéité au niveau du processus serveur.

Chaque niveau de simultanéité peut être contrôlé individuellement :

- Au niveau de la machine abstraite, le contrôle s'effectue par l'insertion d'unités de contrôle dans l'automate de transition.
- Au niveau du processus serveur, la décision de suspendre un processus client plutôt qu'un autre peut relever de la répartition de ressources ou bien de la mise en oeuvre de priorités relatives entre les clients.
- La simultanéité au niveau de l'unité, évoquée en b), doit être contrôlée au niveau de la machine sur laquelle cette unité est réalisée. Les processus serveurs sont en effet des clients sur cette machine et c'est à ce niveau que doit être fait le contrôle.

Une étude plus poussée des outils de contrôle que nous avons mentionnés est faite dans un ouvrage annexe intitulé : "Outils de Répartition des Ressources". Ne refaisons donc pas cette étude ici et contentons nous d'en fournir la conclusion :
"Le modèle de décomposition fournit potentiellement les moyens de décentraliser la fonction de répartition des ressources. Pour chaque machine abstraite, le concepteur de système peut alors définir et mettre en oeuvre une politique locale de répartition en termes des ressources logiques fournies par cette machine".

III.1.2. Moyens de communication

L'outil de communication de base que nous avons défini est le message. Les informations échangées entre les unités transitent donc dans les messages. Mais la taille d'un message est forcément bornée supérieurement. Dans notre projet cette borne supérieure est de 1024 octets, en-tête compris.

Lorsque les informations à transmettre sont volumineuses (par exemple des fichiers) on a parfois recours à la technique de fragmentation, comme dans les réseaux d'ordinateurs, mais l'échange des informations reste une opération coûteuse.

Aussi préfère-t-on limiter autant que possible les transferts d'informations. La communication s'effectue alors conjointement par accès à des mémoires partagées et par messages.

Dans ce paragraphe, nous allons nous intéresser au rôle particulier joué par chacune de ces deux méthodes dans la communication. En particulier nous distinguerons le partage de données considéré comme moyen de communication du partage de données qui résulte d'activités intrinsèquement parallèles (ex : base de données partagée par n utilisateurs).

Nous aborderons ensuite les problèmes de synchronisation résultant du partage de données et nous donnerons un exemple d'outil général de synchronisation réalisé avec une unité.

Pour terminer nous ferons le lien entre la notion d'unité et la notion de module telle qu'elle est définie dans [LUC].

III.1.21. Le partage de données

Comme nous l'avons dit, il y a une forme de partage qui correspond à une nécessité logique. La machine à fichiers présentée au paragraphe III.3 en fournit plusieurs exemples.

Ainsi, un même fichier peut être utilisé simultanément par plusieurs processus via plusieurs contextes d'accès fichiers.

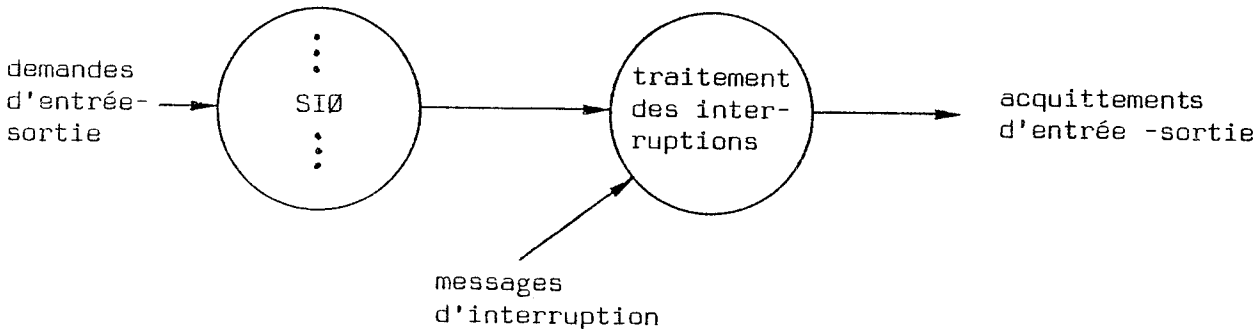
Ainsi également les contextes d'accès fichiers peuvent être des enregistrements d'une copie de fichier, auquel cas cette copie de fichier est utilisée simultanément par tous les processus qui accèdent à au moins un fichier.

Cette nécessité de pouvoir partager des informations nous a d'ailleurs conduits à définir sur la machine MAS une banque de segments partagée par tous les processus.

Un même segment peut alors être lié dans l'espace virtuel de plusieurs processus pour être partagé par ces processus.

Il existe une autre forme du partage qui, comme nous l'avons dit, permet d'éviter des transferts d'information entre unités. Reprenons par exemple le cas de la méthode d'accès fichiers. Dans la machine à enregistrements MAE, les unités U1, U2 et U3 ont toutes accès aux segments contenant les sous-catalogues associés aux fichiers.

Un autre exemple est celui des unités qui réalisent les entrées-sorties dans la machine MAS. Ces unités ne sont pas toutes détaillées dans la décomposition décrite au chapitre V, mais il existe une unité chargée de lancer les entrées-sorties par l'instruction SIØ et une autre unité qui reçoit les messages associés aux interruptions d'entrée-sortie.



Ces deux unités ont en commun les tables décrivant l'état de chaque périphérique. Ces tables ne sont pas transmises dans les messages et ces derniers ne transportent que des adresses dans ces tables.

De façon générale, lorsque plusieurs unités accèdent à des données communes, les messages qu'elles échangent contiennent des références plutôt que des valeurs.

Les deux schémas suivants se substituent alors l'un à l'autre.

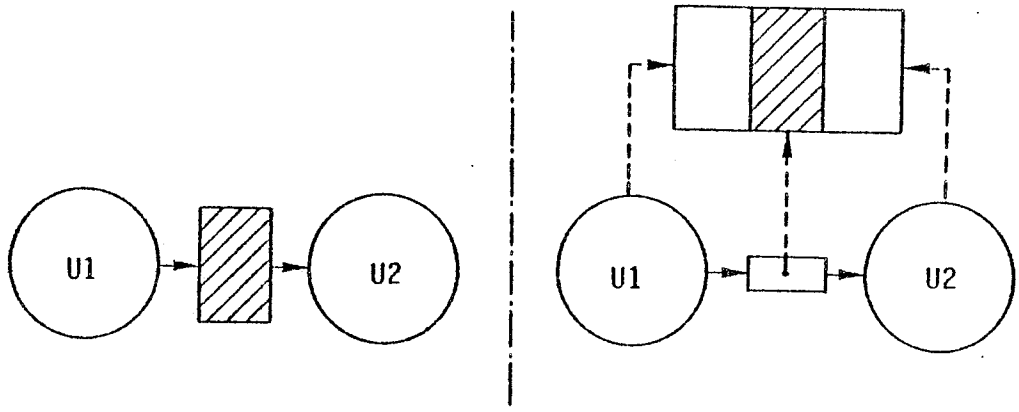


Schéma 1 → Schéma 2

Les flèches en pointillé représentent l'accès à une mémoire commune

III.1.22. Synchronisation

Les deux formes de partage que nous venons d'illustrer ont cependant un point commun : elles sont source de conflits au niveau de l'accès aux données et ces conflits ne peuvent être résolus que s'il existe une section critique commune à tous les chemins d'accès à chaque donnée partagée.

Dans le cas de la machine à fichiers par exemple, la section critique commune à tous les chemins d'accès aux enregistrements est située dans l'unité UAF2. Dans le cas des entrées-sorties effectuées dans MAS, cette section critique commune est la fonction d'accès aux tables décrivant les périphériques.

Nous retrouvons ici la notion de module qui est un ensemble de données munies des fonctions d'accès à ces données. Un module peut correspondre à une unité ou à un groupe d'unités.

Ainsi, dans la machine MAS le module de segmentation-pagination a été réalisé avec une seule unité, par contre le module d'entrées-sorties physiques regroupe une dizaine d'unités. Faisons ici une petite parenthèse pour remarquer que ces deux façons de faire correspondent à deux écoles :

- la première est représentée par le module de segmentation-pagination : le parallèlisme est obtenu en multipliant le nombre de processus serveurs dans l'unité.

- la seconde est représentée par le module d'entrées-sorties : le parallèlisme est obtenu en multipliant le nombre des unités, donc en affinant la décomposition.

Sans doute le choix d'une méthode plutôt que de l'autre doit-il être guidé en définitive par les possibilités réelles de parallèlisme offertes par le matériel dont on dispose.

Fermons maintenant la parenthèse et revenons au problème de la synchronisation.

Pour résoudre les conflits d'accès à une donnée partagée, il faut pouvoir définir, nous l'avons dit, une section critique commune à tous ses chemins d'accès.

Mais cette section critique commune ne doit pas obligatoirement apparaître au niveau même où l'on programme l'accès. Par exemple, les instructions IRIS 80 de lecture et d'écriture en mémoire centrale peuvent apparemment être exécutées en parallèle sur les deux processeurs IRIS. Il y a cependant une section critique au niveau de l'accès à la mémoire de sorte que des instructions d'écriture dans un même mot ne peuvent pas être exécutées en même temps et ne risquent donc pas d'interférer.

Notre point de vue est que la section critique est l'outil de base à partir duquel des instructions de synchronisation plus évoluées peuvent être définies (sémaphores, moniteurs,...).

Le modèle de décomposition permet de réaliser cet outil de base et, qui plus est, cet outil peut être réalisé à n'importe quel niveau de machine abstraite.

En effet, à n'importe quel niveau de machine abstraite, il est possible de créer une unité n'ayant qu'un seul processus serveur. Cette unité représente alors automatiquement une section critique pour tout accès aux données qu'elle manipule.

La communication par messages permet donc de synchroniser des processus sur des données partagées à la condition que ces données soient "enfermées" dans autant d'unités particulières.

Cette solution n'est guère utilisable de façon systématique. Aussi allons nous proposer, à titre d'exemple, un outil de synchronisation plus commode, mais réalisé cependant grâce à la communication par messages.

Exemple :

Convenons d'abord d'appeler "ressource" toute donnée sur laquelle des conflits d'accès peuvent se produire.

Le nombre et la nature des ressources dans un système sont très variables au cours du temps. Pour cette raison la plupart des outils généraux de synchronisation comprennent d'une part quelques instructions et d'autre part un mécanisme de désignation des ressources.

Ce mécanisme de désignation a deux objectifs :

- permettre aux programmes de nommer sans ambiguité et sans interférences possibles les ressources qu'ils gèrent,
- ramener toutes les ressources au même niveau.

Dans le cas de l'instruction LOAD-AND-SET par exemple, la désignation utilisée est celle des mots mémoire IRIS 80. On peut donc avoir autant de ressources gérées par LOAD-AND-SET à un instant donné, qu'il y a de mots disponibles dans la mémoire.

De façon générale, un outil de synchronisation apparaît comme un ensemble d'instructions et d'objets "ressources" manipulés par ces instructions. En tout état de cause, il appartient à chaque programme de gérer la correspondance entre les ressources réelles auxquelles il accède et les objets "ressources" qui sont manipulés par les instructions de synchronisation.

Nous allons maintenant présenter un outil général pour la synchronisation, cet outil étant construit selon le schéma que nous venons de décrire.

a) Principe général d'utilisation :

Cet outil est constitué d'une unité gérant des objets "ressources" et interprétant un certain nombre d'instructions de synchronisation sur ces objets.

En intégrant cette unité dans une machine abstraite, on fournit alors ces instructions de synchronisation aux processus exécutés par cette machine abstraite.

b) Désignation des ressources :

Les ressources sont des objets créés sur demande par l'unité de synchronisation. Nous définissons donc les deux opérations suivantes :

CREER-RESSOURCE

L'unité crée un nouvel identificateur (magnum) et le fournit en retour

DETRUIRE-RESSOURCE (magnum)

L'unité détruit l'objet ressource et l'identificateur ne peut plus être réutilisé.

c) Opérations sur les ressources :

Une ressource ayant été créée, trois opérations sont possibles sur cette ressource :

PARTAGER-RESSOURCE (magnum)

Le processus effectuant la demande a besoin d'utiliser la ressource, mais il accepte de la partager avec d'autres processus

RESERVER-RESSOURCE (magnum).

Le processus effectuant la demande a besoin d'utiliser la ressource, mais il veut être seul à l'utiliser :

LIBERER-RESSOURCE (magnum).

Le processus effectuant la demande n'a plus besoin de la ressource, il la libère.

En outre, un processus ayant demandé une ressource peut la demander une seconde fois en changeant éventuellement les conditions d'accès (partage ou exclusion mutuelle).

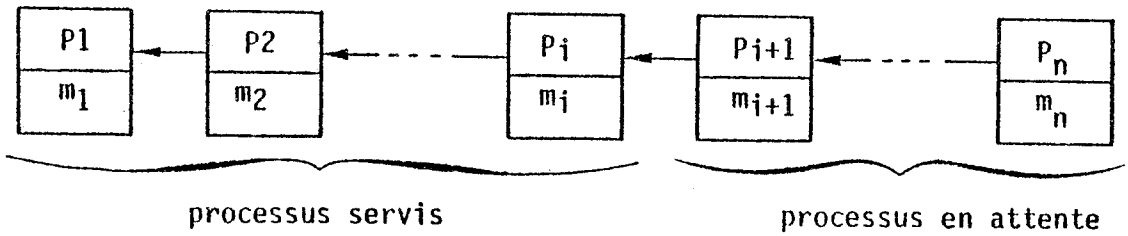
Des interblocages peuvent intervenir entre processus, mais nous pensons qu'ils ne sont pas le fait des mécanismes définis ici. En outre, si toutes les ressources du système sont gérées dans la même unité, la détection des interblocages en est facilitée.

d) Mise en oeuvre :

L'unité de synchronisation comporte un seul processus serveur. Elle gère un ensemble de files d'attente où sont chaînés les messages reçus.

Chaque file correspond à une ressource, il y a donc un nombre variable de files.

Pour chaque ressource les demandes sont servies dans l'ordre chronologique.
Chaque file est organisée de la façon suivante :



P_i : identité de processus (en tête des messages reçus)
m_i : mode d'utilisation de la ressource (partage ou réservation).

L'allocation d'une ressource à un processus se traduit par le renvoi du message associé par l'unité de synchronisation ; ce message est en outre conservé par l'unité dans la partie "processus servis" de la file. Il n'en est enlevé que lorsque le processus libère la ressource.

Si la ressource n'est pas libre, le message est mis en queue de file et n'est pas renvoyé.

Nous ne détaillerons pas davantage le fonctionnement de l'unité de synchronisation, ni les nombreuses variantes qui peuvent y être apportées (demandes non bloquantes, demandes multiples...).

III.1.23. Modules, unités et machines :

Le module regroupe un ensemble de données et les fonctions d'accès à ces données. Il peut correspondre dans notre modèle à une unité où à un groupe d'unités.

La définition de modules dans un système correspond grossièrement à la nécessité de protéger les données et de garantir la cohérence de leur représentation [LUC]. Nous allons montrer sur quelques exemples que la coïncidence entre les modules et nos unités a des conséquences importantes sur le contenu des messages échangés entre les unités.

Exemple 1 :

Le premier exemple que nous prendrons est celui de la machine IRIS.

Nous avons vu que l'unité DISTRIBUTEUR effectue le multiplexage des deux processeurs IRIS 80. A ce titre, elle est chargée des sauvegardes et restaurations de contextes.

Les contextes IRIS sont d'autre part accessibles via des opérations LIRE-IRIS et MODIF-IRIS qui sont exécutées par l'unité PRIMITIVES.

Les messages qui rentrent et qui sortent de la machine IRIS ne contiennent pas les contextes IRIS, mais seulement l'identité des processus auxquels ils sont associés. Les valeurs et pointeurs situés dans le reste du message ne sont que des paramètres pour les fonctions d'accès appelées.

Les contextes IRIS sont protégés par un module qui regroupe ici les unités DISTRIBUTEUR et PRIMITIVES.

Exemple 2 :

Il est également tiré de la machine IRIS.

Le mécanisme de connexion présenté au chapitre II comporte un aspect statique et un aspect dynamique.

Les machines et unités sont créés par l'unité PRIMITIVES. La connexion dynamique est effectuée par les unités ATTEND et RENVOI.

La description des unités et des machines est située dans un module de connexion qui regroupe les unités PRIMITIVES, ATTEND et RENVOI (en toute logique, il aurait fallu définir une unité PRIMITIVES pour le module de connexion et une autre pour la gestion des contextes IRIS : espérons que le lecteur nous excusera pour cette "bavure").

Les messages qui rentrent et qui sortent des unités ATTEND et RENVOI ne contiennent pas la pile d'exécution que nous avons décrite au paragraphe II.2.32, mais seulement l'identité des processus auxquels elle est associée.

Là encore les piles d'exécution sont protégées par le module de connexion et elles n'en sortent pas.

Exemple 3 :

Prenons pour terminer un exemple tiré de la machine MAS. Une unité de segmentation-pagination (SEGPAGE) gère toutes les tables de segments du système. Une table de segments est associée à chaque processus, mais celui-ci peut modifier tout ou partie de son contenu par les opérations CREER-EV, LIER et DELIER.

Les messages qui entrent et qui sortent de cette unité ne contiennent pas les tables de segments, mais seulement l'identité des processus auxquels elles sont associées. Les valeurs ou pointeurs contenus en outre dans les messages ne sont que des paramètres pour les fonctions d'accès à ces tables de segments.

Les tables de segments sont ici protégées par le module de segmentation pagination.

Conclusion :

Les unités d'une machine peuvent être regroupées en un certain nombre de modules. Chaque module contient non seulement les données associées aux ressources dont il a la charge, mais aussi des données décrivant l'utilisation que chaque processus fait de ces ressources.

Le contexte d'un processus dans notre modèle est donc réparti dans un certain nombre de modules et ce sont les messages qui permettent de faire le lien entre ces différentes parties de contexte. Ils transportent en outre des paramètres pour les appels aux fonctions d'accès aux données protégées par les modules.

III.1.3. Spécialisation ou banalisation

Nous avons besoin ici de faire un parallèle entre d'une part une machine abstraite telle qu'elle peut être définie dans notre modèle et d'autre part un réseau de microprocesseurs communiquant par messages et possédant éventuellement des mémoires communes et des mémoires locales.

Si l'on considère chaque constituant d'une machine comme un processeur particulier, on peut alors dégager facilement deux catégories de processeurs :

- les processeurs spécialisés, qui correspondent aux unités,
- les processeurs banalisés, qui correspondent aux machines incluses.

Certains processeurs partagent une mémoire commune et définissent alors une grappe. C'est le cas des unités regroupées dans un module. La mémoire correspond aux données du module. Elle n'est accessible que par les processeurs appartenant à la grappe. Ouvrons ici une parenthèse : une grappe peut regrouper des processeurs spécialisés et des processeurs banalisés, de la même façon que des machines et des unités peuvent accéder à une mémoire commune. C'est un aspect que nous n'avons pas développé plus tôt car il fait appel d'une part à la notion de contrôle et d'autre part à la notion de mémoire. Disons pour simplifier que si une machine regroupe une unité et une machine, alors cette unité et le séquenceur (cf. § III.1.112) de la machine accèdent au même niveau de représentation des données (cf. § III.3) dans la mémoire qu'ils partagent.

L'unité va y chercher des données, le séquenceur de la machine va y chercher des programmes.

Nous reviendrons au paragraphe III.3 sur la notion de niveau de représentation des données. Refermons donc ici la parenthèse.

Si l'on considère maintenant qu'une unité peut être multipliée, les processeurs de base d'une machine ne sont plus les unités et les machines incluses. Une unité apparaît alors comme un "pool" de processeurs spécialisés partageant une mémoire commune.

La distinction entre processeur spécialisé et processeur banalisé est importante dans le cas des architectures multi microprocesseurs.
G. MAZARE suggère [MAZ] que l'utilisation de processeurs spécialisés permet d'obtenir une bonne efficacité globale car elle minimise les échanges entre processeurs. Cette efficacité est cependant conditionnée par un degré de multiprogrammation suffisant qui puisse garantir une occupation simultanée de tous les processeurs ou presque.
Dans le cas où les processeurs sont banalisés, ces taux d'occupation peuvent être optimisés plus facilement quelle que soit la charge globale. En effet, le travail est alloué dynamiquement aux processeurs par chargement des programmes dans les mémoires locales des processeurs. Mais ce

chargement peut être coûteux et il constitue la contrepartie du fait que les processeurs sont banalisés.

Le choix d'une architecture dans notre modèle de décomposition échappe heureusement à ce dilemne.
D'une part les échanges entre les unités (modules) sont peu coûteux puisque les messages échangés ne contiennent que les appels à des fonctions et leurs paramètres ou bien leurs résultats.
D'autre part la banque de segments MAS peut être assimilée à une mémoire commune à tous les processeurs. En effet les segments MAS sont le dénominateur commun à tous les niveaux de mémoires définis par les machines abstraites (cf. § III.2.3.). Les recopies sont donc limitées au maximum car elles peuvent être remplacées par la communication d'un nom de segment dans un message.

Enfin le taux d'occupation des unités ou des machines est peu important puisqu'en définitive toute l'activité du système est supportée par les deux processeurs IRIS 80 et les unités périphériques.

On objectera bien sûr que la pirouette est un peu facile. En fait, les problèmes de gestion de la charge dans notre modèle ne se posent pas en termes de taux d'occupation de chaque constituant (unité ou machine); mais plutôt en termes de temps de réponse de chacun de ces constituants. Ce problème est étudié en détail dans un ouvrage annexe de celui-ci.

Dans l'hypothèse où notre modèle de décomposition doit servir à évaluer des architectures multimicroprocesseurs, les simplifications que nous avons mentionnées ne peuvent plus être faites. Une telle évaluation déborde cependant le cadre de cet ouvrage et nous n'aborderons qu'au chapitre des perspectives (chapitre VI) les problèmes que cette évaluation peut poser.

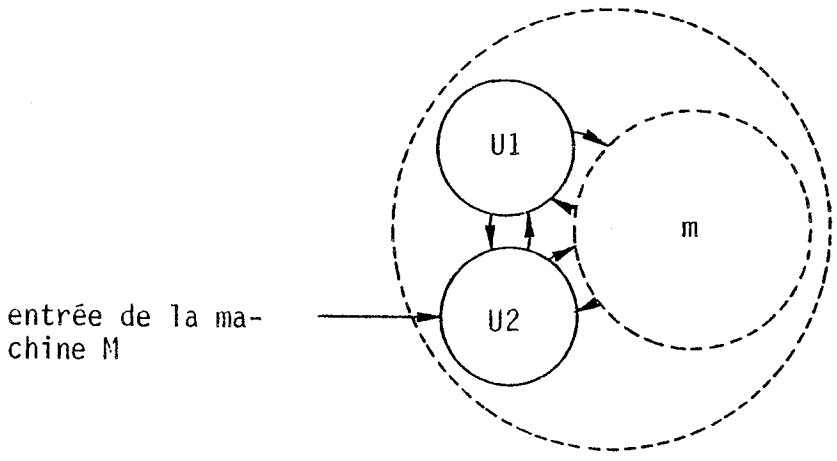
III.2. LES NIVEAUX DE MACHINES ABSTRAITES

Chaque machine abstraite définit un répertoire d'instructions et d'objets qui constitue le langage de programmation "machine" de cette machine abstraite.

Dans ce paragraphe, nous essaierons de montrer que la méthode d'abstraction-raffinement induit certaines relations entre les langages définis à chaque niveau de machine. Le cas échéant, nous suggèrerons un certain nombre de règles d'utilisation des outils qui sont proposés dans notre modèle.

III.2.1. Enrichissement et Réduction

Nous avons déjà évoqué ces deux notions lorsque nous avons décrit les différentes façons de réaliser le contrôle dans une machine (cf. III.1.112.). Rappelons en ici les principes.
Ce sont les unités d'une machine abstraite qui interprètent les instructions du langage défini par cette machine. Donc si l'on veut modifier le langage d'une machine abstraite, il faut soit modifier certaines unités, soit ajouter de nouvelles unités.
Cette dernière solution peut être mise en oeuvre facilement sans que l'on ait besoin de modifier l'automate définissant la machine initiale. Il suffit en effet de construire une nouvelle machine constituée des nouvelles unités et de la machine initiale.



m : machine initiale définissant un langage l
U1 : unité interprétant une nouvelle instruction i
U2 : unité filtrant les appels à m

La machine M définit alors un langage L qui, par rapport à l, est enrichi de l'instruction i. Mais à l'inverse, le filtre effectué par U2 a pour effet que la puissance du langage L est réduite par rapport à celle du langage l.

On verra que la construction de la machine MAS à partir de la machine IRIS fournit un exemple caractéristique d'enrichissement-réduction. En effet, les déroutements des processeurs IRIS 80 qui ne sont pas traités par la machine IRIS provoquent autant de sorties de cette machine IRIS. Les unités de MAS récupèrent et traitent les messages associés à un certain nombre de déroutements.

Ainsi les déroutements relatifs à l'adressage virtuel sont récupérés par MAS et traités de façon à paginer les espaces virtuels.

Ainsi certains appels superviseurs sont utilisés pour enrichir le jeu d'instructions correspondant au mode C esclave de l'IRIS 80 : par exemple, une méthode d'accès standard à tous les périphériques est définie dans MAS et les primitives correspondant à cette méthode d'accès sont associées dans la machine IRIS à des appels superviseurs.

Disons pour terminer que bon nombre de déroutements des processeurs IRIS ne sont pas traités par MAS et provoquent donc autant de sorties de la machine MAS. Bien que la machine IRIS ne soit pas offerte directement aux utilisateurs mais seulement à travers la machine MAS, il est donc possible de récupérer ces déroutements en créant une machine abstraite incluant la machine MAS. On peut de cette façon enrichir encore le langage MAS + IRIS.

III.2.2. Niveaux d'interprétation

Lorsqu'on crée un processus sur une machine abstraite, on fournit un message initial qui définit un programme. Ce programme est ensuite interprété par la machine abstraite au fur et à mesure que le message chemine entre les unités.

Lorsqu'on crée une unité sur une machine abstraite, on fournit un message initial qui définit le programme que tous les processus de l'unité exécuteront. Au moyen des unités ATTEND et RENVOI, ce programme, qui

est interprété, devient lui-même un interpréteur, ou plus exactement une partie d'interpréteur si l'on considère que c'est chaque machine abstraite qui est l'interpréteur du langage qu'elle définit.

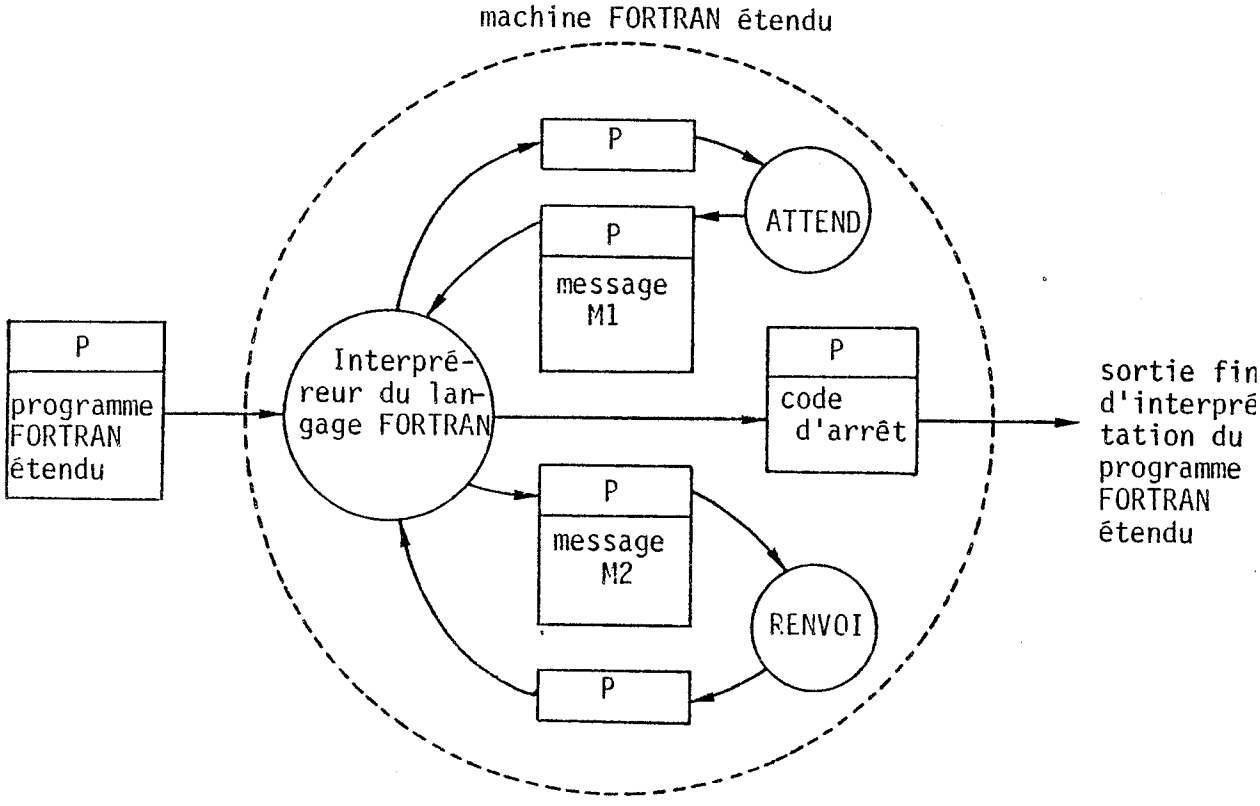
Les unités ATTEND et RENVOI jouent un rôle essentiel dans la programmation d'un interpréteur car elles constituent le moyen par lequel cet interpréteur peut recevoir des commandes et envoyer des résultats.

Nous allons illustrer par un exemple l'utilisation de ces deux unités dans la mise en oeuvre d'une hiérarchie d'interpréteurs.

Exemple :

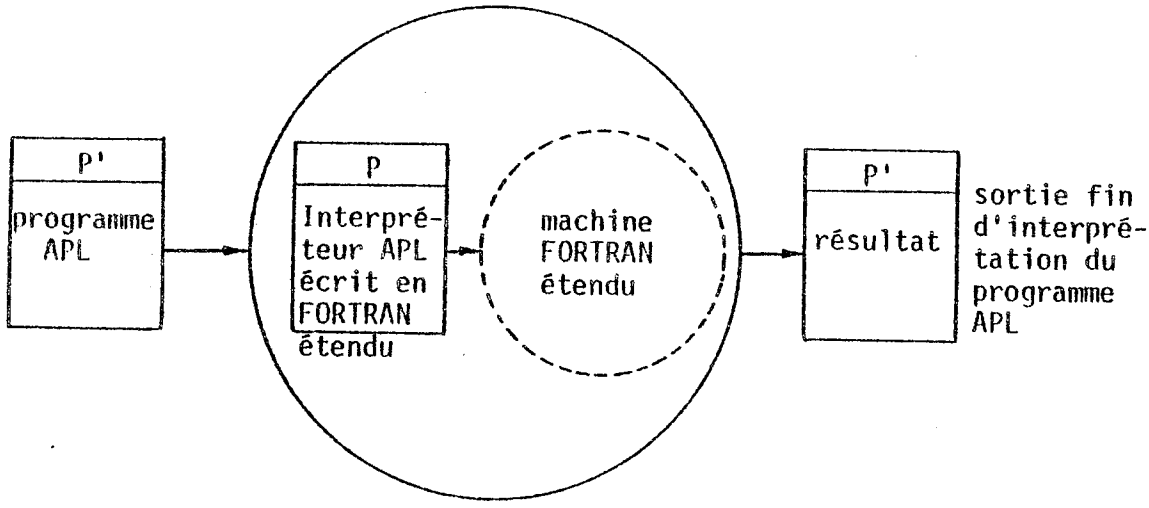
Supposons qu'on dispose d'une machine "FORTRAN étendu", c'est-à-dire qui définisse un langage comprenant d'une part le langage FORTRAN et d'autre part les instructions ATTEND et RENVOI. Ces deux instructions peuvent correspondre par exemple à deux fonctions FORTRAN supposées interprétables par les unités que nous avons décrites.

Pour fixer les idées, nous supposerons qu'un interpréteur FORTRAN a été écrit en langage IRIS 80 et s'exécute donc sur la machine IRIS. Nous pouvons alors donner la structure simple suivante à la machine FORTRAN étendue.



La définition de cette machine par un automate de transition permet de rendre compte des appels procéduraux effectués par l'unité Interpréteur du langage FORTRAN lorsqu'elle rencontre les appels des fonctions ATTEND et RENVOI dans le texte du programme FORTRAN étendu qu'elle interprète.

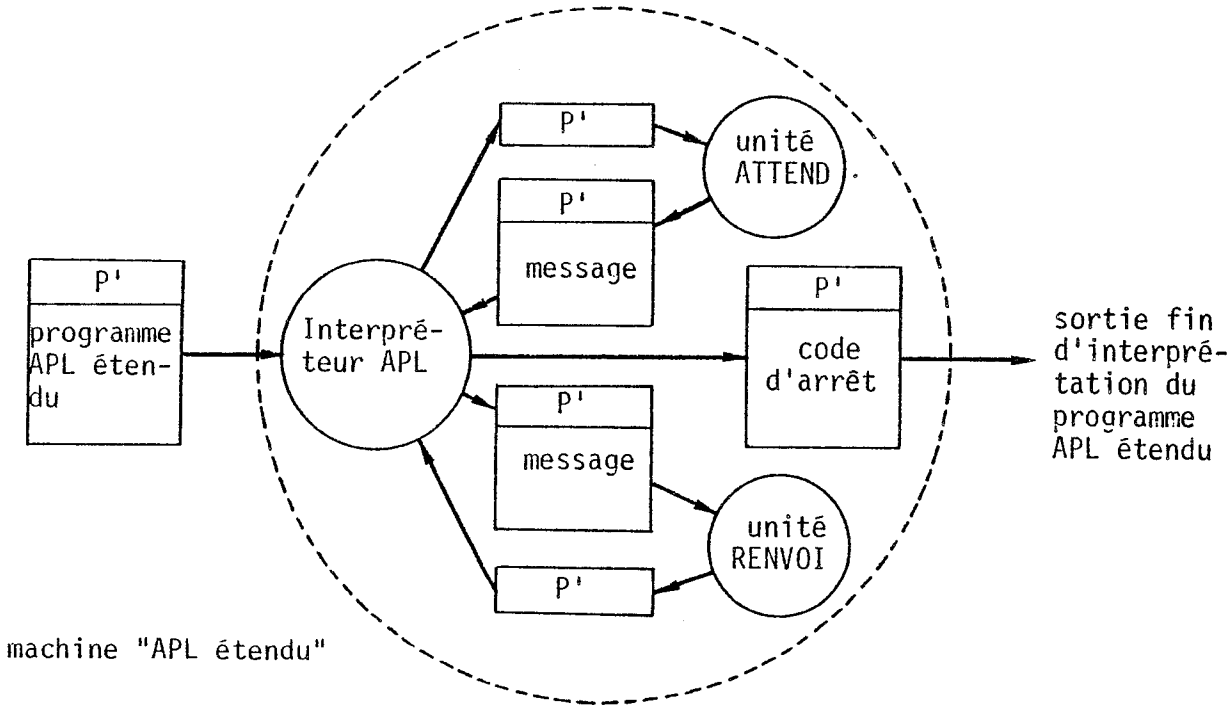
Sur cette machine, on peut créer des unités. Supposons par exemple que l'on créée une unité APL en fournissant comme message initial à cette machine le texte d'un interpréteur APL écrit en FORTRAN étendu.



Interpréteur APL

Les messages M1 et M2 représentés dans la machine FORTRAN sont alors respectivement le programme APL et son résultat figurant sur le schéma ci-dessus, et accompagnés de l'en-tête P'.

On peut évidemment itérer le mécanisme en créant une machine APL étendue constituée de l'unité Interpréteur APL et des unités ATTEND et RENVOI suivant une structure semblable à celle de la machine FORTRAN étendu.



Cette machine est capable d'interpréter des programmes constitués d'instructions APL et des instructions ATTEND et RENVOI. Sur cette machine, on peut donc créer des unités en lui soumettant des programmes qui soient des interpréteurs écrits en langage APL étendu...

La hiérarchie d'interprétation développée sur cet exemple peut être énoncée de la façon suivante (en assimilant les programmes des unités aux processus qui les déroulent) :

Le programme P' est interprété par la programme P, qui est lui-même interprété par l'interpréteur FORTRAN, qui est lui-même exécuté par l'IRIS 80.

Remarque :

Lorsque l'interpréteur APL, exécuté par le processus P, décode une instruction d'attente de commande dans un programme APL étendu, il envoie le processus P' associé à ce programme sur l'unité ATTEND et continue son travail. L'interprétation d'une instruction d'attente de commande ne bloque donc pas cet interpréteur.

Cette remarque met en évidence une propriété importante de la hiérarchie d'interpréteurs mise en oeuvre par les niveaux de machine abstraite.

Les unités ATTEND et RENVOI sont en effet disponibles à n'importe quel niveau de machine abstraite. Elles définissent respectivement une instruction d'attente de commande et une instruction d'envoi de résultats qui sont essentielles lorsqu'on écrit un interpréteur. En conséquence, ces instructions peuvent être intégrées dans n'importe quel langage sans qu'on doive se soucier de les interpréter. Il en résulte en particulier qu'un interpréteur ne se met jamais en attente d'une commande pour le compte des programmes qu'il interprète, ce qui lui permet d'être disponible au mieux pour ces programmes.

Notons pour terminer que si l'on n'a pas besoin d'interpréter les instructions d'attente de commande et d'envoi de résultats, il faut quand même définir pour ces instructions une syntaxe qui s'accorde avec le langage dans lequel elles sont intégrées. Il faut, en outre, prévoir leur décodage dans une des unités de la machine qui interprète ce langage.

III.2.3. Niveaux de représentation des données

Le schéma fondamental du traitement de l'information est le suivant [HOA] :

- lecture d'une information
- traitement de l'information
- écriture de l'information traitée.

Ce schéma est manifestement récursif, et chaque étape de la récursion est généralement associée dans une application à une couche (ou niveau) de logiciel ou de matériel. Dans le modèle de décomposition que nous avons présenté, chacune de ces couches correspond à un niveau de machine abstraite et nous avons développé dans ce chapitre la relation d'interprétation qui relie les programmes déroulés sur chaque niveau de machine. Mais à chaque niveau de machine correspond également un niveau de représentation des données manipulées par ces programmes, que ces données soient dans les mémoires des machines ou bien qu'elles soient échangées avec le monde extérieur.

Dans cette relation, un objet défini sur un niveau de machine i sera représenté par une structure d'objets définis au niveau i-1.

Une instruction de traitement de l'objet de niveau i sera alors interprétée sur des machines i-1 au moyen d'instructions de lecture, d'écriture et aussi de traitement d'objets de niveau i-1, conformément au schéma indiqué plus haut.

Si l'on reprend par exemple le cas de la machine APL donné précédemment, une inversion de matrice est une opération unaire qui nécessite l'accès à un seul objet APL ; mais son interprétation sur la machine FORTRAN étendue nécessitera l'accès à tous les objets FORTRAN que sont les éléments de la matrice.

Nous verrons au paragraphe suivant un exemple de mise en oeuvre de cette relation dans la définition d'une méthode d'accès fichier, c'est-à-dire d'un niveau de représentation et d'accès aux données.

On notera cependant que, dans le cas des fichiers, il est d'usage courant d'offrir simultanément à l'utilisateur trois niveaux de représentation et d'accès aux objets.

- le niveau fichier : les primitives d'accès sont OUVRIR et FERMER
- le niveau enregistrement : les primitives d'accès sont LIRE et ECRIRE,
- le niveau caractère : les primitives d'accès sont les instructionsde manipulation de chaînes de caractères.

Cela correspond au fait qu'il est pratiquement impossible de définir à chaque niveau un ensemble de primitives suffisamment puissant qui permette de faire tous les traitements possibles sur les objets du niveau. On préfère donc laisser ce travail à l'utilisateur en lui fournissant pour chaque objet de niveau i, sa représentation en termes d'objets de niveau i-1, ainsi que les primitives d'accès à ces objets.

Dans notre modèle, cette solution se traduit par l'inclusion de la machine de niveau i-1 dans celle de niveau i. Prenons par exemple le cas de la machine MAS. Les segments sont les objets offerts au niveau MAS ; les primitives de transfert entre un segment et un bloc d'espace virtuel constituent la méthode d'accès à ces objets au niveau MAS. Mais MAS n'offre pas directement toutes les primitives permettant de modifier ces objets. Pour ce faire, elle communique à la machine IRIS incluse dans MAS la représentation de ces objets. Cette représentation est constituée d'objets accessibles sur la machine IRIS : les mots mémoire.

L'utilisateur peut alors modifier lui-même la valeur d'un segment au moyen des instructions d'accès à sa représentation, c'est-à-dire les instructions d'accès mémoire.

Un certain nombre de précautions doivent cependant être prises dans l'utilisation de cette possibilité d'inclusion et plus généralement dans toute mise en oeuvre, avec notre modèle, de niveaux d'accès aux données.

a) En premier lieu, nous dirons qu'une relation d'inclusion entre des machines ne peut remplacer une relation d'interprétation que dans la mesure où un assemblage quelconque d'objets d'un certain niveau constitue toujours une représentation correcte d'un objet du niveau supérieur. Dans la pratique, ces deux relations coexistent souvent, c'est-à-dire que, si une machine A contient une machine B, elle contiendra également un certain nombre d'unités construites sur B. De cette façon, les programmes déroulés dans ces unités pourront travailler sur le niveau B de représentation des objets de A ; ils pourront en particulier vérifier que les assemblages d'objets de B fabriqués par l'utilisateur sur la machine B forment des représentations correctes d'objets de A. C'est ce que fait par exemple une méthode d'accès direct pour un fichier en format fixe :

- au niveau de l'enregistrement, elle vérifie que le nombre de caractères fournis est correct,
- au niveau du fichier, elle vérifie que l'index fourni avec l'enregistrement se situe bien dans les limites définies par la taille du fichier.

b) La deuxième remarque concerne le lien qui existe entre l'état d'un processus tournant sur une machine, et l'état des objets manipulés par ce processus.
Un processus est représenté dans notre modèle par un message cheminant dans une machine. On a vu au paragraphe II.2.31 qu'un processus passe ainsi alternativement d'un état d'intreprétation à un état de transition. Si l'on considère maintenant que ce processus est exécuté par la machine A citée plus haut, on conçoit aisément que des objets de A manipulés par ce processus risquent d'avoir une représentation incorrecte (incohérente) en termes d'objets de B lorsque ce processus est dans l'état d'interprétation. En effet un programme tournant sur la machine

B est peut-être précisémment en train de modifier cette représentation.

On devine les problèmes de synchronisation que cela peut amener lorsque des objets de A sont partagés par plusieurs processus. Il appartient au constructeur d'une machine ou d'une unité de gérer correctement l'accès aux objets qu'elle permet de manipuler, puisqu'en effet l'existence même de ces objets n'est pas connue du noyau. Suggérons cependant deux principes qui devraient permettre de faciliter cette gestion :

- Premièrement faire en sorte que dans l'automate associé à chaque machine, un état de transition corresponde toujours à un état de cohérence des objets manipulables sur cette machine. Concrètement, lorsque plusieurs unités doivent collaborer à la construction d'un objet, nous suggérons donc d'isoler le sous-automate associé à cette collaboration en définissant une machine incluse.

En respectant ce principe, l'expression de la synchronisation peut alors être faite au niveau des constituants d'une machine. Remarquons d'ailleurs que sur une machine réelle, les opérations de base sont en général ininterruptibles, ce qui garantit la cohérence des objets manipulés.

- Deuxièmement, lorsque les utilisateurs ont la possibilité d'accéder directement à plusieurs niveaux de représentation des objets d'une machine, il est prudent de ne leur fournir à chaque niveau que des copies d'objets du niveau supérieur. L'utilisateur ne risque alors pas de détériorer les objets puisqu'il ne travaille que sur des copies ; mais copies et originaux se situent bien sûr au même niveau de représentation.

Généralement, une copie j d'un objet A_i constitué des objets B_i^1, B_i^2...B_i^n sera constituée elle-même d'objets $B_{i,j}^1$, $B_{i,j}^2$ ... $B_{i,j}^n$ ayant la même valeur initialement que les objets B_i^k.

Sur certaines machines ces objets $B_{i,j}^k$ sont créés dynamiquement par la machine A chaque fois qu'une copie est nécessaire.

Sur d'autres machines, un certain nombre d'objets B_p sont prédéfinis et réservés pour les copies. Seule leur valeur est affectée dynamiquement lorsqu'une copie est nécessaire.

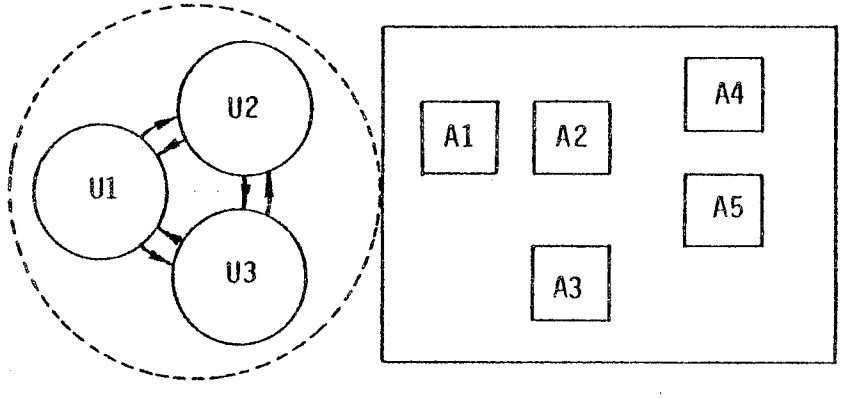
Les chaînes d'octets qu'un utilisateur échange avec une méthode d'accès fichier sont un exemple caractéristique de copies d'objets, les objets en question étant des enregistrements de fichier.

L'utilisation des copies n'est cependant pas obligatoire comme nous l'avons dit, ce qui permet d'éviter une multiplication coûteuse de ces copies lorsqu'on déroule une cascade d'interprétations.

Par exemple, dans le cas cité plus haut d'une méthode d'accès direct,des copies ne sont créés qu'au niveau de l'enregistrement. Mais on notera que la cohérence du fichier ne peut alors être garantie qu'en bloquant l'accès au fichier pendant tout le temps qu'un processus y accède en écriture. Disons pour résumer qu'on a le choix entre deux méthodes : d'une part créer des copies de façon à limiter les conflits d'accès à la mémoire de la machine abstraite, ce qui permet de gérer ces conflits assez simplement ; d'autre part ne pas créer de copies, mais gérer assez finement (par exemple sur chaque objet) les accès à la mémoire de la machine abstraite si l'on ne veut pas que celle-ci constitue un goulot d'étranglement.

III.2.4. Niveaux de désignation des objets

Si, comme nous l'avons dit, la mémoire d'une machine abstraite contient les objets accessibles au niveau de cette machine, l'adresse de chaque objet dans la mémoire constitue sa désignation.



machine A mémoire de A

Soit donc la machine A dont la mémoire est constituée des objets créés ou préexistants sur cette machine, soient A1, A2, A3, A4 et A5.

Chaque objet est créé par l'une des unités U1, U2 ou U3 (ou par plusieurs) sur demande d'un processus tournant sur la machine A. Cet objet reçoit en même temps de la part de ces unités une adresse qui doit permettre de le retrouver dans la mémoire de A.

Cette adresse est fournie en réponse à la demande de création. Chaque objet doit évidemment posséder une adresse différente. De plus cette adresse ne doit jamais être réaffectée par la machine A, même après destruction de l'objet. Nous appellerons nom unique de l'objet cette adresse.

Supposons maintenant que les unités U1, U2 et U3 sont construites sur une machine B. Les programmes déroulés par ces unités manipulent donc exclusivement des objets de B. Ce sont ces programmes qui déterminent la représentation des objets de A en termes d'objets de B. Le mécanisme de désignation des objets décrits pour A s'applique à toutes les machines, de sorte que la représentation de chaque objet de A se compose des noms uniques des objets de B qui le constituent, et d'un schéma de montage interprétable par la machine A.

Il se pose alors le problème de savoir où doivent être conservées ces représentations. Il n'est guère possible de les conserver systématiquement dans la désignation : la désignation de chaque objet contiendrait alors les désignations des objets qui le composent, et ainsi de suite jusqu'aux désignations des objets de base, qui, dans notre cas, sont les octets.

Une autre solution consiste alors à gérer dans chaque machine abstraite, ou du moins dans chacune de celles qui définissent des objets, un catalogue des objets créés par cette machine. Ce catalogue permet d'accéder à la représentation de chaque objet au moyen de son nom unique. Il est géré par les unités de cette machine et constitué lui-même d'objets de niveau inférieur dont les noms uniques doivent être des constantes dans les programmes des unités.
Généralement d'ailleurs la machine abstraite donnera au catalogue qu'elle gère la même représentation qu'aux objets qu'elle définit.

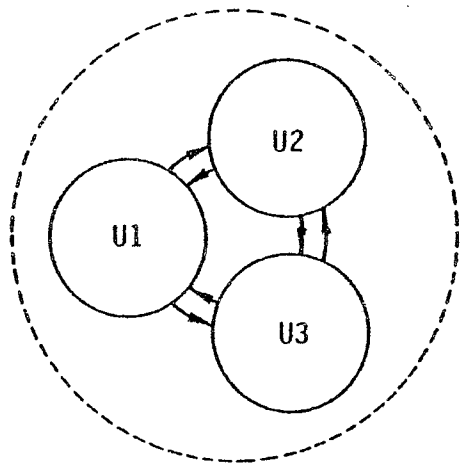
Considérons par exemple un système de gestion de fichiers. Il définit une machine abstraite dont les objets sont les fichiers. Chaque fichier possède un nom unique et est constitué d'un certain nombre de pistes.

Le cataloque des fichiers est lui-même un fichier qui contient donc son propre nom.

Mais le système de gestion de fichiers doit évidemment pouvoir accéder au catalogue sans passer par son nom. Les numéros de pistes qui constituent le fichier catalogue doivent donc figurer en constantes dans les programmes du système de gestion de fichiers.

Ajoutons d'autre part que le schéma de désignation que nous avons suggéré s'applique uniquement aux originaux des objets et non aux copies. En effet celles-ci ne sont jamais accessibles au niveau supérieur même si elles ont la même représentation que les originaux. Elles n'ont pas de nom unique et ne sont donc pas cataloguées. Dans le cas des enregistrements d'un fichier par exemple, un ordre d'écriture devra avoir pour paramètre la copie d'un enregistrement, mais aussi le nom unique de l'enregistrement dont cette copie doit remplacer la valeur.

Nous dirons pour terminer que les deux méthodes de désignation suggérées coexistent le plus souvent dans un système. C'est notamment le cas lorsqu'un niveau de machine abstraite B n'est utilisé que comme support d'un niveau de machine abstraite A.



Machine A

Exemple :

Les unités U1, U2 et U3 sont construites sur la machine B, qui ne supporte pas d'autres unités ou processus. Si l'on suppose que les unités de la machine B sont construites sur une machine C, alors il n'est pas besoin de gérer un catalogue dans la machine B si le catalogue de A contient pour chaque objet de A les noms uniques des objets de C qui le constituent. En d'autres termes, le catalogue de la machine B est remonté dans celui de A.

Cela entraîne les deux conséquences suivantes sur la désignation des objets de B :

- les unités U1, U2 et U3 désignent les objets de B par leur représentation complète en termes d'objets de C,
- tout objet de B est constituant d'un objet de A. Si la machine B est incluse dans A, les programmes déroulés sur la machine A ne peuvent donc accéder qu'à des objets de A ou à leurs constituants de niveau B. Ils désignent alors ces derniers par une notation pointée :

  nom unique d'un objet de A . nom local d'un constituant de niveau B.

L'organisation d'une méthode d'accès fichier est représentative de ce type de schéma. Nous en verrons un exemple au paragraphe suivant.

Dans le cas où une seule copie est maintenue pour chaque objet, le partage d'un objet peut entraîner le partage de la copie. Il est alors nécessaire de mémoriser dans le catalogue de la machine abstraite la constitution des copies en même temps que celle des objets originaux : à chaque constituant d'un objet on associe le constituant correspondant dans la copie. La notation pointée indiquée ci-dessus désigne alors les constituants de la copie et non pas ceux de l'original puisque ces derniers ne sont pas accessibles.

On verra au chapitre IV que les segments sont gérés de cette façon sur la machine MAS.

III.3. EXEMPLE : UN SYSTEME DE GESTION DE FICHIERS

Les supports d'information offerts par MAS sont les segments ; ils sont limités en taille à 128 K mots. Les enregistrements d'un segment sont les mots, et les opérations primitives d'accès à ces mots sont les instructions IRIS 80 d'accès mémoire.

Sur cette machine, nous nous sommes proposés de construire une méthode d'accès fichiers, c'est-à-dire une machine abstraite définissant des notions de fichiers et d'enregistrements plus classiques.

Nous nous sommes inspirés plus précisemment de la méthode d'accès VSAM définie sur les systèmes IBM OS 370. On trouvera dans la thèse de J.M. FORESTIER [FOR] une spécification détaillée des services définis par la méthode d'accès proposée. Nous allons présenter dans ce paragraphe la réalisation de cette méthode d'accès sur la machine MAS.

III.3.1. La machine à enregistrements

Il faut noter en premier lieu que les seuls mots IRIS 80 accessibles par les programmes tournant sur MAS appartiennent à la version temporaire (c'est-à-dire la copie) des segments. Conformément au schéma que nous avons indiqué au paragraphe III.2.4., chaque mot de la copie d'un segment doit être désigné par une notation à deux niveaux :

nom unique de segment . adresse du mot dans le segment

c'est-à-dire que l'accès à chaque mot doit passer par le catalogue décrivant les segments et leurs copies. On peut considérer que les copies de segments sont décrites par les tables de pages gérées par les unités de MAS. Comme une table de pages est associée biunivoquement à un segment, la désignation d'une table de pages peut remplacer le nom unique dans la notation indiquée plus haut. Cette désignation est interprétée comme on le verra au chapitre IV par le mécanisme de translation automatique d'adresses :

adresse table de segments
+ 7 bits de poids fort de l'adresse du mot
→ adresse de la table de pages

La correspondance représentée par la flèche est établie par les unités de MAS sur demande de l'utilisateur (primitives LIER et DELIER).

Nous pouvons écrire sur MAS un programme ou un ensemble de programmes qui créent de nouveaux objets par assemblage de mots ; nous appellerons ces objets des enregistrements. Ces programmes peuvent communiquer soit parce qu'ils accèdent aux mêmes mots, soit parce qu'ils utilisent le mécanisme de communication par messages disponible sur la machine IRIS. Dans ce cas, ces programmes doivent être associés à des unités regroupées dans une machine.

Nous définissons alors une machine à enregistrements MAE de la façon suivante :

- objets : Ce sont les enregistrements ; chaque enregistrement est identifié par un nom unique que nous détaillerons plus loin et sa valeur est constituée de mots appartenant à des copies de segments MAS.

Toute copie d'un enregistrement sera également constituée de mots appartenant à des copies de segments MAS.

Le mécanisme de communication par messages permet aux unités de la machine MAE d'échanger des copies d'enregistrements entre elles et aussi avec l'extérieur.

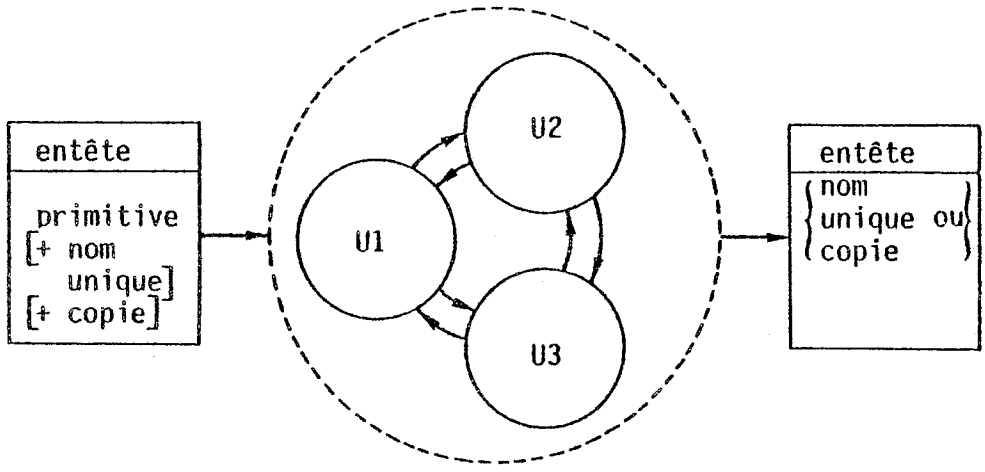
- Opérations primitives :

CREER (copie) → nom unique de l'enregistrement créé

ECRIRE (nom unique, copie) : la valeur de chaque mot de la copie remplace la valeur de chaque mot de l'original

LIRE (nom unique) → copie de l'enregistrement désigné

DETRUIRE (nom unique) : l'enregistrement désigné est détruit, c'est-à-dire retiré du catalogue.



machine MAE

Les unités U1, U2 et U3 sont construites sur MAS. Comme nous l'avons développé au paragraphe 2.4., le nom unique d'un enregistrement peut être constitué de deux façons différentes :

- soit il contient les noms uniques des mots qui constituent l'enregistrement,
- soit la machine MAE gère un catalogue des enregistrements, auquel cas le nom unique est un pointeur dans ce catalogue.

Dans la méthode d'accès que nous voulons réaliser, comme d'ailleurs dans toutes les méthodes d'accès fichier classiques, les objets "enregistrements" sont toujours des constituants d'objets plus évolués qu'on appelle fichiers. Dans notre modèle, cela signifie que la machine MAE ne sera jamais fournie à l'utilisateur directement, mais elle servira de support à la réalisation d'une machine à fichiers MAF.

Il est alors possible de remonter les catalogues d'enregistrements dans le catalogue des fichiers géré par MAF. C'est cette solution qu'illustre le schéma ci-dessous :

Noms uniques des fichiers	Noms locaux des enregistrements constitutifs	noms uniques des mots constituant chaque enregistrement				
f_1	e_{11}	$(s.ad)^1_{11}$	$(s.ad)^2_{11}$	...	$(s.ad)^{n_{11}}_{11}$	
	e_{12}	$(s.ad)^1_{12}$	$(s.ad)^2_{12}$	...	$(s.ad)^{n_{12}}_{12}$	
	⋮					
	e_{1n_1}	$(s.ad)^1_{1n_1}$	$(s.ad)^2_{1n_1}$	...	$(s.ad)^{n_{1n_1}}_{1n_1}$	
⋮	⋮	⋮				
f_p	e_{p1}	$(s.ad)^1_{p1}$	...	$(s.ad)^{n_{p1}}_{p1}$		
	⋮					
	e_{pn_p}	$(s.ad)^1_{pn}$	...	$(s.ad)^{n_{pn_p}}_{pn_p}$		

$(s.ad)^k_{ij}$ = notation pointée désignant le mot numéro k de l'enregistrement numéro j du fichier i.

s est le nom unique du segment contenant ce mot, ad est l'adresse relative de ce mot dans le segment.

Si on adopte cette solution, l'utilité d'avoir créé une machine à enregistrements entre MAS et la machine à fichiers devient pratiquement nulle car les unités U1, U2 et U3 n'ont pas grand chose à faire. Dans la méthode d'accès que nous voulons réaliser, le plus gros travail consiste en effet à gérer le catalogue des enregistrements d'un fichier donné.

La solution que nous préconisons permet de laisser faire cette gestion à la machine à enregistrements tout en imposant aux utilisateurs de désigner les enregistrements par la notation pointée suivante :

nom unique de fichier . nom local d'enregistrement

Cette solution consiste à englober la machine à enregistrements dans la machine à fichiers et à communiquer à cette dernière le sous-catalogue associé à chaque fichier en même temps que le nom local de l'enregistrement désigné.

Pour pouvoir être exploité par la machine à enregistrements qui est construite sur MAS, chaque sous-catalogue doit être constitué d'objets MAS, c'est-à-dire de mots appartenant à des segments.

Par ailleurs un sous catalogue peut être volumineux,de sorte qu'il est exclu d'en transmettre une copie dans les messages qui sont communiqués à la machine MAE. D'ailleurs cette machine doit travailler directement sur les sous-catalogues si on veut permettre le partage des fichiers au niveau de l'enregistrement. C'est donc directement la désignation des mots contenant les sous-catalogues qui est communiquée à MAE. Enfin, lors de la mise en marche de chaque machine abstraite, celle-ci déroule une séquence d'initialisation du catalogue des objets qu'elle définit ; mais dans le cas de la machine MAE, cette phase doit pouvoir être déroulée dynamiquement chaque fois qu'un nouveau sous-catalogue doit être crée. Le jeu d'instructions de la machine MAE doit donc être enrichi d'une primitive CREER-CATALOGUE.

CREER-CATALOGUE → noms uniques des mots constituant ce catalogue.

Le catalogue des fichiers gérés par la machine à fichiers MAF est alors constitué de la façon suivante :

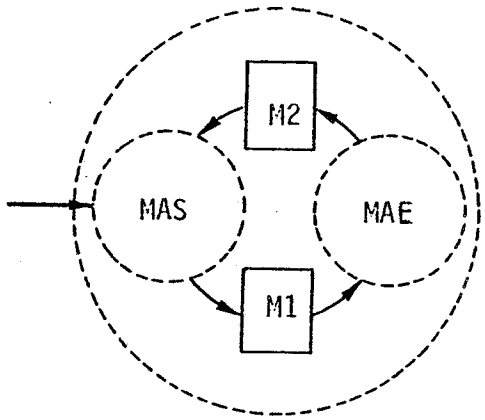
f_1	$(s.ad)_1^1$	$(s.ad)_1^2$	...	$(s.ad)_1^{n_1}$	
f_2	$(s.ad)_2^1$	$(s.ad)_2^2$	...	$(s.ad)_2^{n_2}$	
⋮	⋮				
f_p	$(s.ad)_p^1$	$(s.ad)_p^2$	...	$(s.ad)_p^{n_p}$	

$(s.ad)_i^j$: notation pointée désignant le mot j du sous-catalogue associé au fichier i.

La structure du catalogue s'apparente manifestement à celle des sous-catalogues ; il semble donc intéressant de le faire gérer par la machine MAE en construisant les unités de MAF sur MAE.

La désignation des mots contenant ce catalogue doit alors être passée en paramètre dans les requêtes adressées à la machine MAE. Cette désignation est obtenue par la primitive CREER-CATALOQUE dans la phase d'initialisation de la machine à fichiers, puis elle figure en constante dans les programmes réalisant cette machine (cf. § III.2.4.).

Avant de présenter la structure que nous donnons à la machine MAF, nous apportons une légère modification à la machine à enregistrements telle que nous l'avons présentée.
Cette modification vise à permettre aux programmes manipulant des enregistrements de fabriquer et recevoir des copies de ces enregistrements. Nous construisons donc une machine à enregistrements ME constituée de MAE et de MAS.



machine à enregistrements
ME

Les messages échangés entre MAS et MAE ont le format que nous avons décrit précédemment.

M1 : en tête,
code opération
[catalogue de référence (noms uniques des mots qui contiennent ce catalogue)]
[nom local d'enregistrement]
[copie d'enregistrement fabriquée sur MAS]

M2 : entête

[catalogue créé ou modifié]

[copie d'enregistrement]

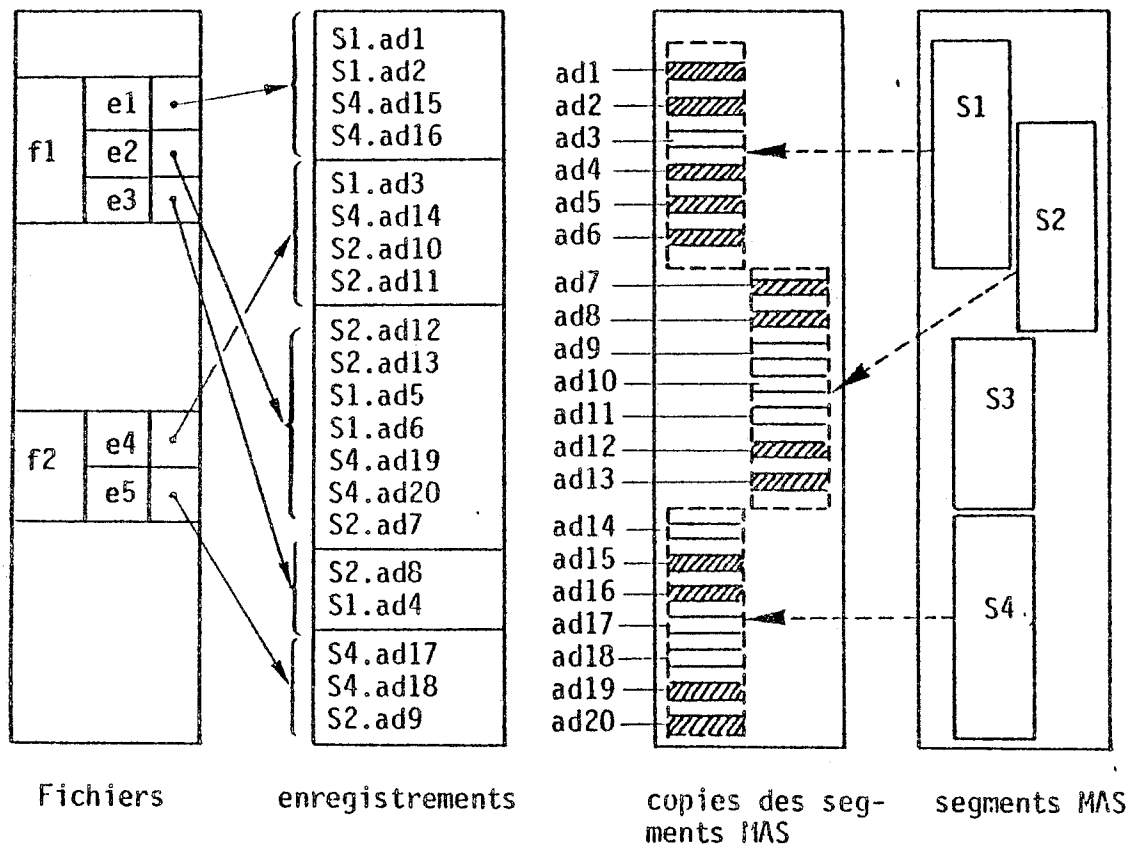
[nom local d'enregistrement]

C'est en définitive sur la machine ME que nous programmons les unités de la machine à fichiers, et nous réservons ME à cet usage. Les enregistrements référencés dans les messages M1 et M2 sont donc ceux du catalogue des fichiers.

III.3.2. La machine à fichiers

III.3.21. Définition fonctionnelle

La constitution des objets fichiers telle que nous l'avons décrite jusqu'à présent ne permet pas de faire de ces fichiers un véritable support de conservation de l'information. En effet, leur représentation se compose d'enregistrements dont les originaux sont eux-mêmes constitués de mots appartenant à des copies de segments MAS. C'est ce qu'illustre le schéma ci-dessous :



Sur ce schéma par exemple la représentation du fichier f_1 se compose des enregistrements e1, e2 et e3 dont les originaux sont constitués des mots figurés en foncé.

Cette remarque nous amène à considérer que la représentation que nous avons définie n'est pas celle d'un fichier, mais plutôt celle de sa copie. Plus précisemment, c'est le sous-catalogue associé à chaque fichier qui matérialise cette copie.

A ce stade de l'exposé, il convient de dire en effet que la méthode d'accès fichier que nous développons diffère sensiblement des méthodes d'accès fichier classiques pour se rapprocher davantage de la façon dont MAS gère les segments.

Dans une méthode d'accès fichier classique, ancune copie du fichier n'est vraiment fabriquée puisque toute addition, suppression ou remplacement d'enregistrement est implicitement répercuté sur disque. L'utilisateur accède donc directement (mais sous contrôle) à la représentation de l'objet fichier.

Dans la méthode d'accès que nous développons, nous choisissons au contraire de ne fournir à l'utilisateur qu'une copie du fichier, ainsi qu'une primitive lui permettant d'affecter à l'original la valeur de la copie. Ajoutons qu'une seule copie est créée pour chaque fichier de façon à permettre le partage au niveau enregistrement. En définitive la machine à fichiers doit conserver les objets fichiers sur plusieurs sessions.Elle doit pouvoir construire une copie pour chaque fichier et maintenir dans un catalogue la correspondance avec l'original. Enfin elle doit fournir aux utilisateurs des primitives de création (CREER), destruction (DETRUIRE), lecture (primitives d'accès aux enregistrements de la copie), et écriture (SAUVER). Nous allons examiner chacun de ces différents points.

A - Conservation des fichiers :

Les unités de la machine à fichiers sont construites sur la machine ME qui contient MAS. Nous utilisons donc les supports de conservation de l'information offerts par cette machine pour conserver les fichiers. Ces supports sont de deux types : les disques dits "privés" et les segments MAS.

Les disques privés sont utilisables via le protocole "appareil standard" d'entrée-sortie. C'est toujours ce type de support qu'utilisent les méthodes d'accès classiques (y compris VSAM - d'OS 370).
Les segments MAS sont utilisables via les primitives SAUVER et METTRE-A-JOUR. C'est cette solution que nous choisissons et nous développerons au paragraphe suivant les raisons de ce choix.

L'original d'un fichier est donc constitué de mots appartenant à des segments MAS. Compte-tenu de la façon dont fonctionnent les primitives SAUVER et METTRE-A-JOUR, un ensemble de segments MAS doit être réservé en particulier pour chaque fichier, c'est-à-dire qu'un segment MAS ne doit pas contenir des mots appartenant à plusieurs fichiers.

B - Construction des copies de fichiers :

Nous abordons ici le problème du choix de la représentation à utiliser pour les originaux des fichiers. Cette représentation doit être aussi compacte que possible pour des raisons simples d'économie de place sur disque. Mais il n'en est pas de même pour la représentation des copies de fichiers : la rapidité d'accès aux enregistrements est dans ce cas un objectif parfois plus important que la taille de la copie.

En tout état de cause une correspondance biunivoque doit être établie entre les deux représentations et il appartient à la machine à fichiers de gérer cette correspondance.

Nous avons envisagé deux façons de résoudre ce problème.
La première consiste à faire reconstruire un nouveau sous-catalogue par la machine à enregistrements chaque fois qu'on a besoin de la copie d'un fichier. L'original d'un fichier et sa copie peuvent dans ce cas avoir des structures différentes, correspondant à des objectifs différents. Mais la construction de l'un à partir de l'autre par la machine à fichiers est une opération coûteuse car elle doit se faire enregistrement par enregistrement.
Nous n'avons donc pas retenu cette solution.

La solution que nous avons choisie est basée sur le fait que les segments MAS, dont les copies contiennent la copie d'un fichier, peuvent très bien constituer le support de l'original de ce fichier.
L'original d'un fichier et sa copie ont de ce fait la même structure et la construction de l'un à partir de l'autre se réduit à l'appel

des primitives MAS, LIER, SAUVER et METTRE-A-JOUR.

Compte-tenu de la remarque faite plus haut, la mise en oeuvre de cette solution nécessite cependant que les copies de fichiers occupent des ensembles disjoints de copies de segments. En d'autres termes, la machine à enregistrements ne doit pas mettre dans la copie d'un segment MAS des données (enregistrements ou partie du sous-catalogue) relatives à des copies de fichiers différents.

D'autre part, l'organisation des données dans chaque copie de fichier doit viser l'économie de place autant que la rapidité d'accès aux enregistrements. Cela n'est d'ailleurs pas une nouvelle contrainte imposée à la machine à enregistrements puisqu'elle travaille de toute façon sur un espace virtuel paginé.

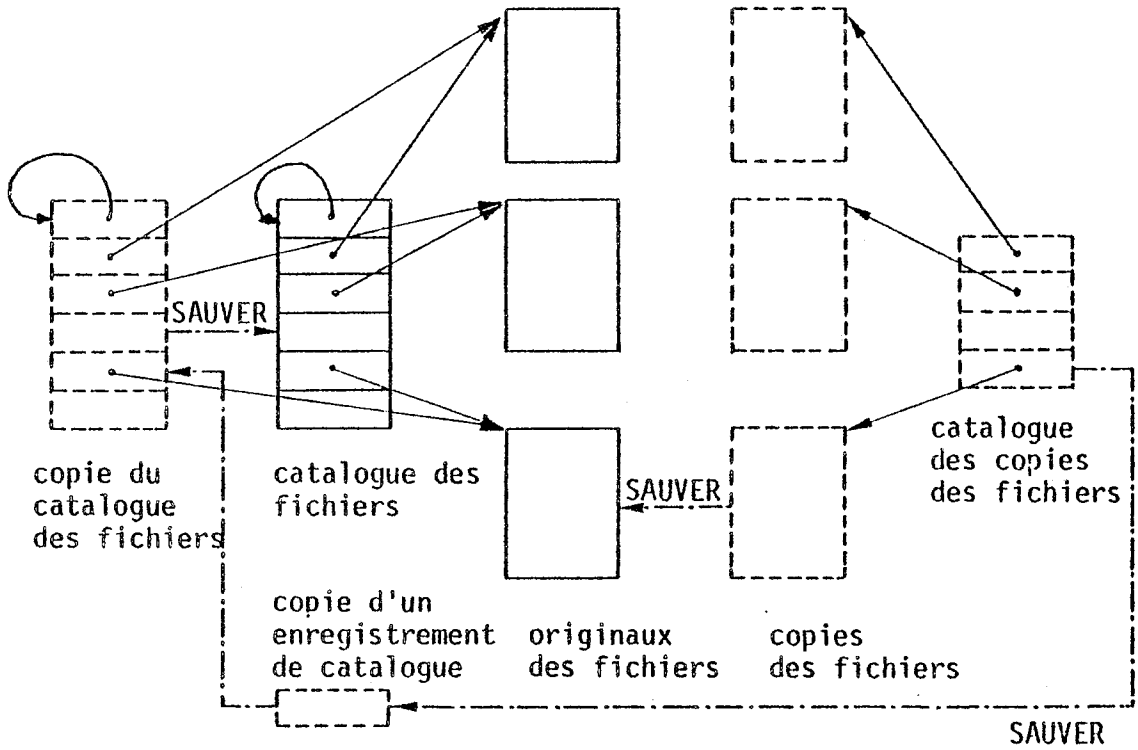
C - Catalogue des fichiers :

Le mode de désignation que nous choisissons pour les fichiers est celui des noms uniques. Le même nom unique désigne à la fois l'original d'un fichier et sa copie. Nous avons déjà présenté (§ III.2.3 et § III.2.4.) les mécanismes liés à ce mode de désignation ; nous ajouterons ici que ces mécanismes n'excluent pas que le nom unique puisse être choisi par le créateur du fichier. La machine à fichiers doit en tout cas refuser la création de plusieurs fichiers avec le même nom.

La correspondance que nous avons définie entre l'original d'un fichier et sa copie peut être utilisée pour le catalogue des fichiers puisque ce dernier est également un fichier. Cependant la machine à fichiers doit gérer un catalogue des copies de fichiers qui est différent de la copie du catalogue des fichiers. Ce catalogue des copies contient un enregistrement pour chacun des fichiers qui possèdent une copie. Cet enregistrement fait référence au sous-catalogue associé à cette copie ; mais il est recopié dans l'original du catalogue des fichiers (via la copie du catalogue) que lorsque l'utilisateur décide de SAUVER le fichier.

Le catalogue des copies a la même structure que la copie du catalogue des originaux. C'est donc logiquement la machine à enregistrements qui doit manipuler cette structure, comme elle manipule celle du catalogue des originaux.

Schématisons comme suit l'organisation des catalogues :



La copie d'enregistrement que nous avons figurée sur le schéma est manipulée par la machine à fichiers. Les flèches en pointillé correspondent aux opérations effectuées par cette machine lorsqu'elle interprète la primitive SAUVERFICHIER, pour un fichier donné.
Les flèches en trait continu décrivent la structure des catalogues.

D - Opérations primitives :

Les opérations primitives définies par la machine à fichiers sont les suivantes :

CREERFICHIER (nom unique, caractéristiques).

Les caractéristiques fournies en paramètre sont transmises telles quelles à la machine à enregistrements lors de la création du sous-catalogue. Ces caractéristiques peuvent être par exemple la taille des enregistrements ou la description des clés d'accès aux enregistrements :

DETRUIRE FICHIER (nom unique).

L'objet fichier est retiré du catalogue et l'original est détruit, mais la destruction de la copie est différée jusqu'au moment où elle n'est plus utilisée :

SAUVER FICHIER (nom unique).

La copie du fichier désigné remplace l'original. Les opérations mises en oeuvre dans l'exécution de cette primitive sont représentées par les flèches en pointillé sur le schéma précédent. La sauvegarde d'un fichier se traduit matériellement par la sauvegarde des segments contenant ce fichier (valeur des enregistrements et sous-catalogue). Mais elle peut également nécessiter la sauvegarde du catalogue puisque ces segments sont répertoriés dans un enregistrement de ce catalogue :

LIRE FICHIER (nom unique) → référence de la copie

Cette opération permet d'obtenir une copie de l'enregistrement qui est associé à la copie du fichier dans le catalogue des copies :

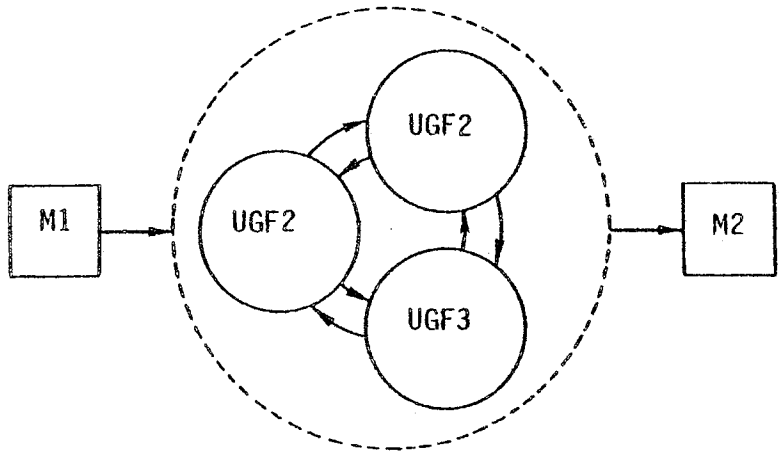
ECRIRE FICHIER (nom unique, copie d'enregistrement).

La copie fournie en paramètre remplace l'enregistrement qui est associé à la copie du fichier dans le catalogue des copies.

III.3.22. Structure de la machine à fichiers

Ayant ainsi défini la machine à fichiers de façon fonctionnelle par les objets et opérations qu'elle offre, nous allons maintenant décrire sa structure.

Sur la machine ME décrite précédemment, nous programmons un certain nombre d'unités qui réalisent en commun les fonctions de gestion des fichiers. Soient UGF1, UGF2, UGF3 ces unités ; nous les regroupons dans une première machine MGF dite de gestion de fichiers qui a la structure suivante :



MGF : Machine de gestion de fichiers

Les messages M1 et M2 ont la structure suivante :

M1 : entête
code opération [+ paramètres]
[copie d'enregistrement du catalogue des copies]
nom unique de fichier

M2 : entête
compte-rendu d'exécution de l'opération
[copie d'enregistrement du catalogue des copies]

Ce n'est pas directement cette machine MGF que nous offrons aux utilisateurs, mais une machine plus riche offrant notamment un mécanisme que l'on trouve dans toutes les méthodes d'accès fichier classiques et qui correspond aux opérations OUVRIR et FERMER. On trouve d'ailleurs également ce mécanisme dans la machine MAS au niveau de l'accès aux segments via les espaces virtuels (primitives LIER et DELIER).

Les objets mis en jeu par ce mécanisme ont une durée de vie qui est relativement limitée ; ce sont les contextes d'utilisation de fichiers, que nous appelons CAF (Contextes d'Accès Fichier).
Si l'on admet un instant qu'une suite d'appels à la machine MAF définit un processus d'accès fichier, alors on peut dire que le CAF est le contexte de ce processus.

De façon plus concrète, un contexte d'accès fichier est composé d'informations telles que :

- pointeur courant dans le cas d'un accès séquentiel,
- pouvoirs du processus d'accès fichier, c'est-à-dire liste des opérations autorisées dans le cadre de ce contexte,
- nom unique du fichier,
- références du sous-catalogue associé au fichier.

Chaque contexte d'accès fichier est désigné par un nom unique choisi par le créateur dans l'opération OUVRIR. Les opérations relatives à la manipulation de ces contextes d'accès fichier sont les suivantes :

OUVRIR (nom de CAF, nom de fichier, autres paramètres).

Cette opération a pour effet de créer un nouveau contexte d'accès fichier.

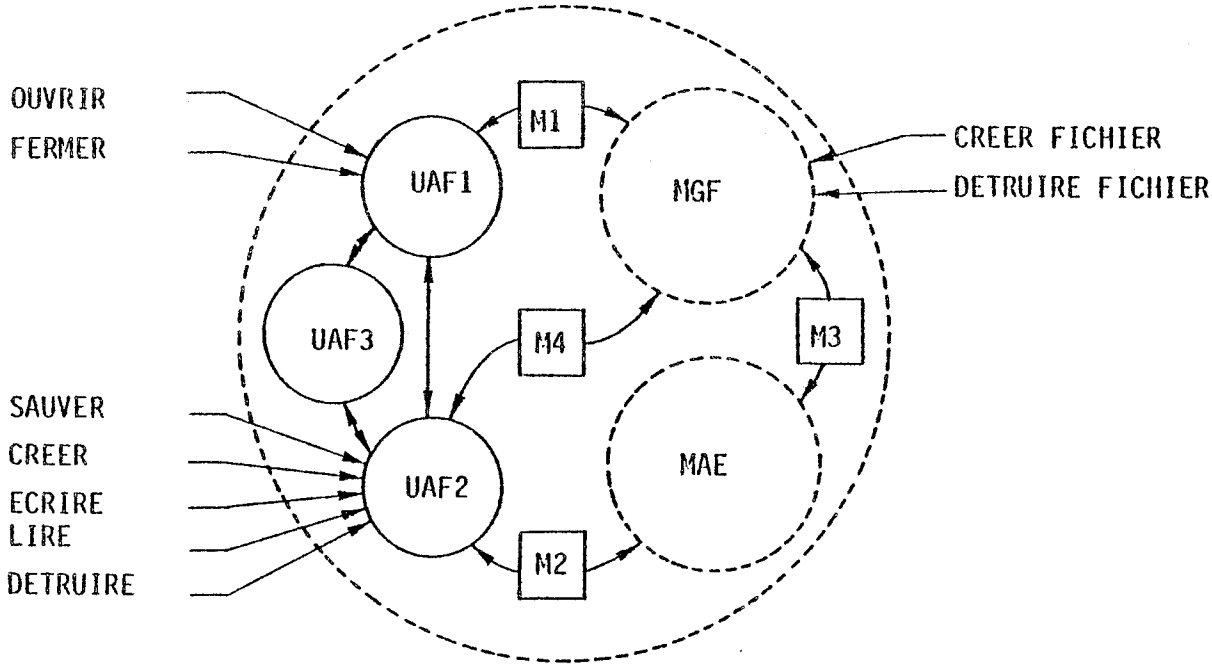
FERMER (nom de CAF).

Cette opération a pour effet de détruire le contexte d'accès fichier désigné.

Les unités effectuant la gestion de ces objets sont réalisés sur la machine ME de façon à bénéficier des services offerts par la machine à enregistrements qui y est incluse. Par exemple, chaque contexte d'accès fichier peut être un enregistrement.

Soient par exemple UAF1, UAF2 et AUF3 ces unités.

La machine à fichiers MAF que nous fournissons en définitive aux utilisateurs est composée de ces unités, de la machine MGF et de la machine MAE. Elle a la structure suivante :



Contenu des messages M1 : LIRE FICHIER
ECRIRE FICHIER

Contenu des messages M2 : CREER
ECRIRE
LIRE
DETRUIRE
} Opérations accompagnées de la référence du sous-catalogue associé au fichier.

Contenu des messages M3 : CREER CATALOGUE

Contenu des messages M4 : SAUVER FICHIER

L'opération SAUVER FICHIER et les opérations relatives aux enregistrements sont définies au niveau de la machine MAE et ne sont donc pas offertes directement aux utilisateurs. Elles sont préalablement traitées par une unité d'accès fichier (UAF3 ou UAF2 sur notre schéma) avant d'être transmises à la machine à enregistrements pour exécution. C'est en particulier dans cette unité d'accès fichier qu'est réalisé l'accès séquentiel.

Pour résumer la définition fonctionnelle de la machine à fichiers, nous dirons qu'elle définit un répertoire d'objets et d'opérations qui est ainsi constitué :

- objets : fichiers, contextes d'accès fichiers, enregistrements.

- opérations :

CREER FICHIER (nom unique, caractéristiques)
DETRUIRE FICHIER (nom unique)

OUVRIR (nom unique de CAF, nom unique de fichier, paramètre)
FERMER (nom unique de CAF)

SAUVER (nom unique de CAF)

CREER (nom unique de CAF, nom local d'enregistrement,copie)
ECRIRE (nom unique de CAF, nom local d'enregistrement,copie)
LIRE (nom unique de CAF, nom local d'enregistrement)
DETRUIRE (nom unique de CAF, nom local d'enregistrement).

Pour être plus complet, il faudrait rajouter à cette liste les opérations d'accès séquentiel aux enregistrements. Nous ne les avons pas mis pour ne pas trop entrer dans le détail des services proposés par la méthode d'accès.

CHAPITRE IV

LE NOYAU MAS

IV.1. INTRODUCTION

On peut dire d'une manière générale qu'un système informatique est à la fois :

- l'*interpréteur* d'un langage particulier répondant aux besoins de ses utilisateurs. Le langage définit une *machine abstraite* [BRI6]. Une machine abstraite fournit - à travers son langage - des fonctions d'accès aux données qu'elle gère. D'un point de vue externe, ces données apparaissent comme des *ressources logiques* manipulables par les instructions de la machine abstraite.

- l'*allocateur* des ressources logiques qu'il fournit à ses utilisateurs.

On peut construire des machines abstraites en termes d'autres machines abstraites. Nous avons présenté au Chapitre II des procédés de construction de machines par connexion de "boîtes noires", unités ou machines précédemment construites. Ces mécanismes permettent de mettre en oeuvre des *niveaux d'abstraction* [DIJ].

IV.1.1. Le noyau MAS

L'expérience [BEK, ORG, WUL] prouve qu'on peut définir une machine de base qui soit commune à une grande variété de systèmes. On appelle *noyau* une telle machine.

Les noyaux connus présentent de nombreux points communs au niveau de la gestion des ressources physiques. Ce niveau peut être constitué en noyau élémentaire.

Nous avons voulu réaliser un tel noyau au moyen des mécanismes présentés au Chapitre II. Nous l'avons appelé MAS (MAchine Système).

MAS est destiné à faciliter la construction de machines plus évoluées. Les instructions de la machine MAS se répartissent en trois groupes :

- les instructions de l'IRIS 80 esclave en modes B ou C [CII 1] ;
- des instructions permettant de construire, de faire communiquer et d'utiliser des machines construites sur MAS (voir Chapitre II) ;
- des instructions de manipulation des ressources logiques

- segments,
- appareils standards (périphériques),
- unités centrales.

Dans ce chapitre, nous nous intéresserons uniquement aux ressources logiques fournies par MAS. Le chapitre suivant montrera la construction de la machine MAS au moyen des outils présentés au Chapitre II.

L'exposé qui va suivre est assez général car les problèmes posés sont assez classiques. Nous n'entrerons pas dans les détails de réalisation des algorithmes mis en oeuvre dans MAS, sauf là où ces algorithmes présentent quelque originalité.

Nous parlerons succinctement des primitives du langage de la machine MAS, des objets fournis (les ressources logiques) et nous aborderons quelques problèmes d'efficacité posés par leur mise en oeuvre. L'efficacité est un objectif important dans la réalisation d'un noyau élémentaire : c'est une condition primordiale pour l'efficacité globale des systèmes qu'il supporte.

IV.1.2. Désignation des objets

Les ressources logiques "segments" et "périphériques" de la machine MAS sont allouées aux utilisateurs sur leurs demandes. MAS fabrique à cette occasion un *nom unique* servant à désigner la ressource de manière non ambiguë. L'utilisateur peut, s'il le veut, transmettre ce nom unique à d'autres utilisateurs, leur donnant ainsi le pouvoir d'utiliser la ressource.

Le nom unique contient une clef fabriquée par le système et des informations facilitant l'accès à la représentation physique de l'objet. Le nom unique d'un segment occupe 64 bits (dont 32 de clef) et celui d'un périphérique occupe 32 bits (dont 16 de clef).

Le nom unique est le seul moyen d'accès à un objet. C'est donc aussi un moyen de protection des utilisateurs entre eux puisque celui qui ignore le nom ne peut utiliser l'objet.

La durée de validité d'un nom unique est celle de l'objet désigné : infinie pour un segment, égale à une session de travail pour un périphériqu
Un utilisateur peut détruire un segment ou libérer un périphérique ;

le nom unique est alors détruit en même temps.

La protection qu'offre le nom unique est-elle efficace ?

La protection est probabiliste et dépend du rapport entre le nombre N d'objets et le nombre de combinaisons possibles pour les noms de ces objets (2^{64} pour un segment, 2^{32} pour un périphérique). Il n'est donc pas totalement exclu que - soit erreur, soit malveillance - un utilisateur puisse disposer d'un nom unique qu'il ne devrait pas connaître. Il y a là une source d'erreurs système qu'il serait difficile de diagnostiquer.

Il faut toutefois rappeler qu'un processus utilisateur est créé sur une certaine machine contenant la machine MAS et que par conséquent les segments, comme les périphériques, ne sont accessibles par les utilisateurs qu'à travers des couches de logiciel développées sur MAS. Une partie du rôle de ce logiciel doit être de contrôler la validité des actions des utilisateurs qu'il supporte. C'est à ce niveau, à travers des mécanismes de contrôle ad hoc, qu'il est possible de prévenir l'utilisation illicite des ressources logiques.

Une façon de réaliser ce contrôle est que ce logiciel gère des désignations symboliques des objets qu'il fournit. Les noms uniques sont alors inutilisables par les utilisateurs des systèmes.

Une autre approche consiste à uniformiser l'accès aux objets d'un système en utilisant des descripteurs d'accès ("capability"). Cette approche s'est montrée productive dans HYDRA [WUL], GEMAU [GUI] et Plessey 250 [FER].

Dans un autre domaine, cette méthode serait utile pour la mise en place de protocoles de communication dans un réseau, en ce qu'elle permettrait un seul type d'accès à des informations réparties. Ceci ne se justifie pas au niveau du noyau MAS, qui doit conserver pour d'autres genres d'applications les distinctions de types entre segments et périphériques, mais pourrait être réalisé dans un système supporté par MAS.

IV.2. LA MEMOIRE SEGMENTEE

IV.2.1. Le choix des objets

Le système MAS veut offrir un support homogène pour les programmes, données et fichiers de ses utilisateurs. Une gestion homogène de l'information est déjà présentée dans quelques systèmes (ESOPE [KRA], MULTICS [ORG], GEMAU [GUI], SOCRATE [ABR]). Elle repose sur une structuration de l'information en segments ou unités logiques équivalentes.

Ces segments sont répertoriés dans un catalogue général surtout utilisé pour en contrôler les accès (cf. Figure 1).

Le système MAS travaille aussi sur des segments. Il ne gère pas de catalogue général mais un catalogue pour chaque unité de disque. Le contrôle d'accès aux segments est obtenu par la vérification du Nom Unique (cf. § IV.1). C'est une protection d'accès rudimentaire, mais justifiable quand l'unité d'information à protéger est un sous-ensemble ou un sur-ensemble de segments (entité de base de données, gros fichiers, ...).

Le système MAS contrôle la cohérence des segments qu'il conserve sur ses unités de disque. L'utilisateur travaille sur des copies temporaires de segments conservés (comme dans le système CMS [AUR]) et ne modifie que ponctuellement les versions originales.

Ce procédé protège l'utilisateur contre ses propres erreurs de manipulation et minimise l'effet des cassures du système. La mémoire segmentée est alors divisée en deux espaces (cf. Figure 2) :

- la banque de segments d'une part : elle regroupe des segments conservés sur des unités de disque appelées Volumes MAS ;
- l'espace de travail d'autre part : il regroupe essentiellement les copies des segments conservées. L'ensemble des copies utilisées par un processus utilisateur est déterminé par un espace virtuel.

Les objets manipulés sont les segments, les volumes MAS, les espaces virtuels.

IV.2.11. Les segments

L'unité d'information désignable est le segment. Ses caractéristiques sont fixées par le mécanisme d'adressage de l'Iris 80 (cf. Figure 3), c'est-à-dire qu'il est constitué de 1 à 256 pages de 512 mots. Comme tous les objets systèmes, un segment reçoit un Nom Unique à sa création. Ce nom unique est composé d'une clé et d'un nom interne qui permet au système de localiser le segment.

Un segment possède les caractéristiques suivantes :

- un type : permanent ou temporaire selon que sa durée de vie dépasse ou non celle d'une session de travail ;
- un mode : B ou C selon qu'il contient du code Iris 80 exécutable en mode B ou en mode C [CII 1] ;
- une taille maximum : la limite supérieure du nombre de pages que peut atteindre le segment ;
- une enveloppe : le créateur peut éventuellement spécifier la taille de l'espace de travail du segment. Cette caractéristique peut être modifiée par le système (cf. § IV.2.413).

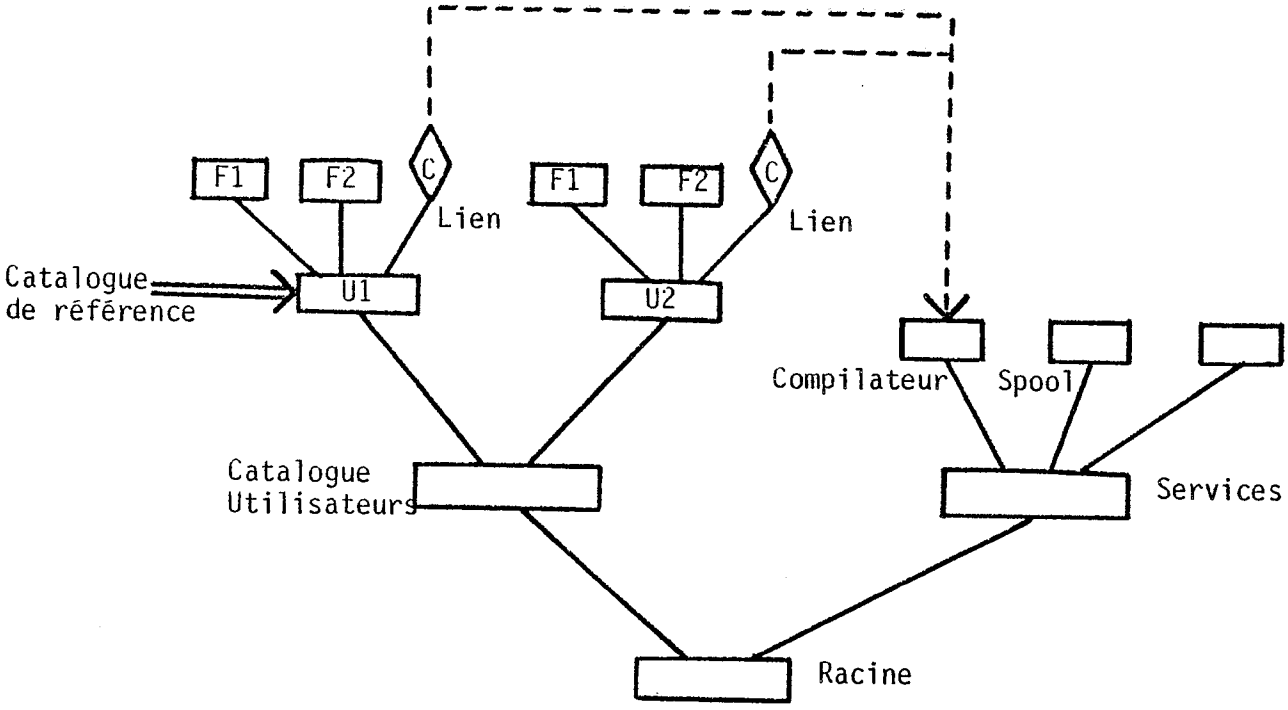
IV.2.12. Les Volumes MAS

Les segments de type permanent sont conservés sur un ensemble de disques d'un format spécial : ce sont les Volumes MAS.

L'ensemble des segments conservés et, par abus de langage, l'ensemble des volumes MAS sont appelés Banque de Segments.

La notion de volume MAS est nécessaire pour trois raisons :

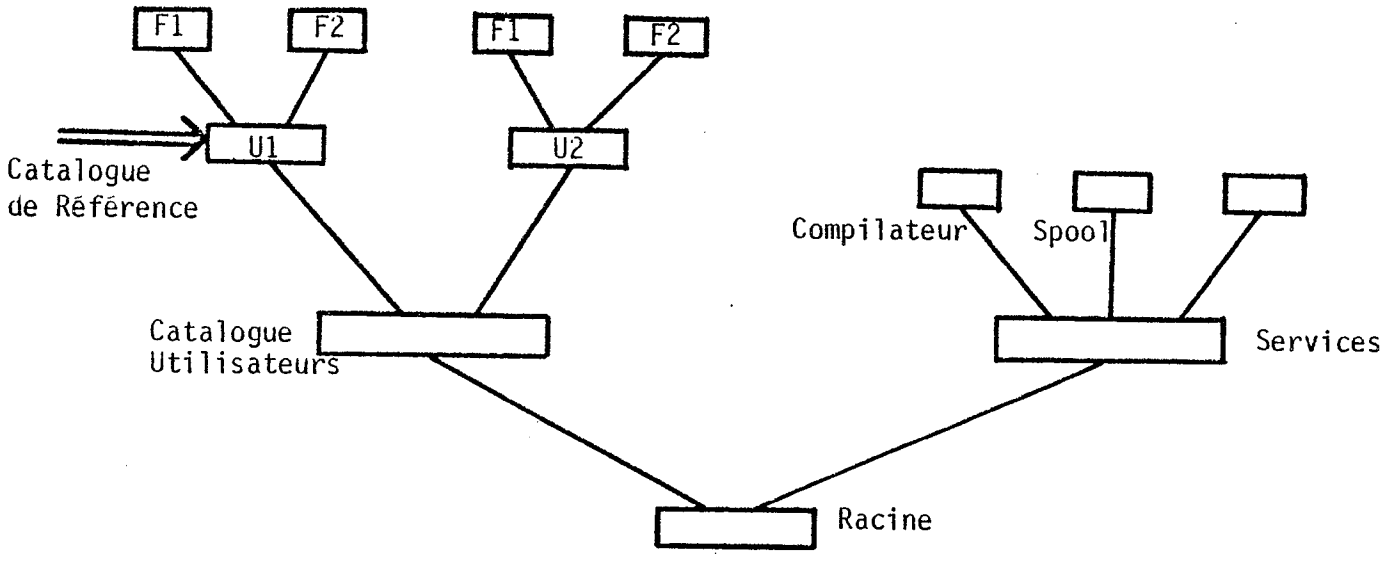
- le nombre de volumes MAS peut être supérieur au nombre d'unités de disques disponibles. On peut donc ne disposer en ligne que d'une partie de la banque de segments, appelée Espace permanent (cf. Figure 2) ;
- un disque de la banque de segment peut être accidentellement détruit. Un tel accident ne doit pas entraîner la reconstruction de toute la banque de segments ;
- un utilisateur (programmeur système) peut vouloir regrouper sur un même disque les segments d'une même application (par exemple une machine Système) ;



Arborescence dans GEMAU

L'objet compilateur peut être désigné dans le catalogue de référence par :

C



Arborescence dans MULTICS

L'objet compilateur peut être désigné dans le catalogue de référence par :

** SERVICES.COMPILATEUR

Figure 1

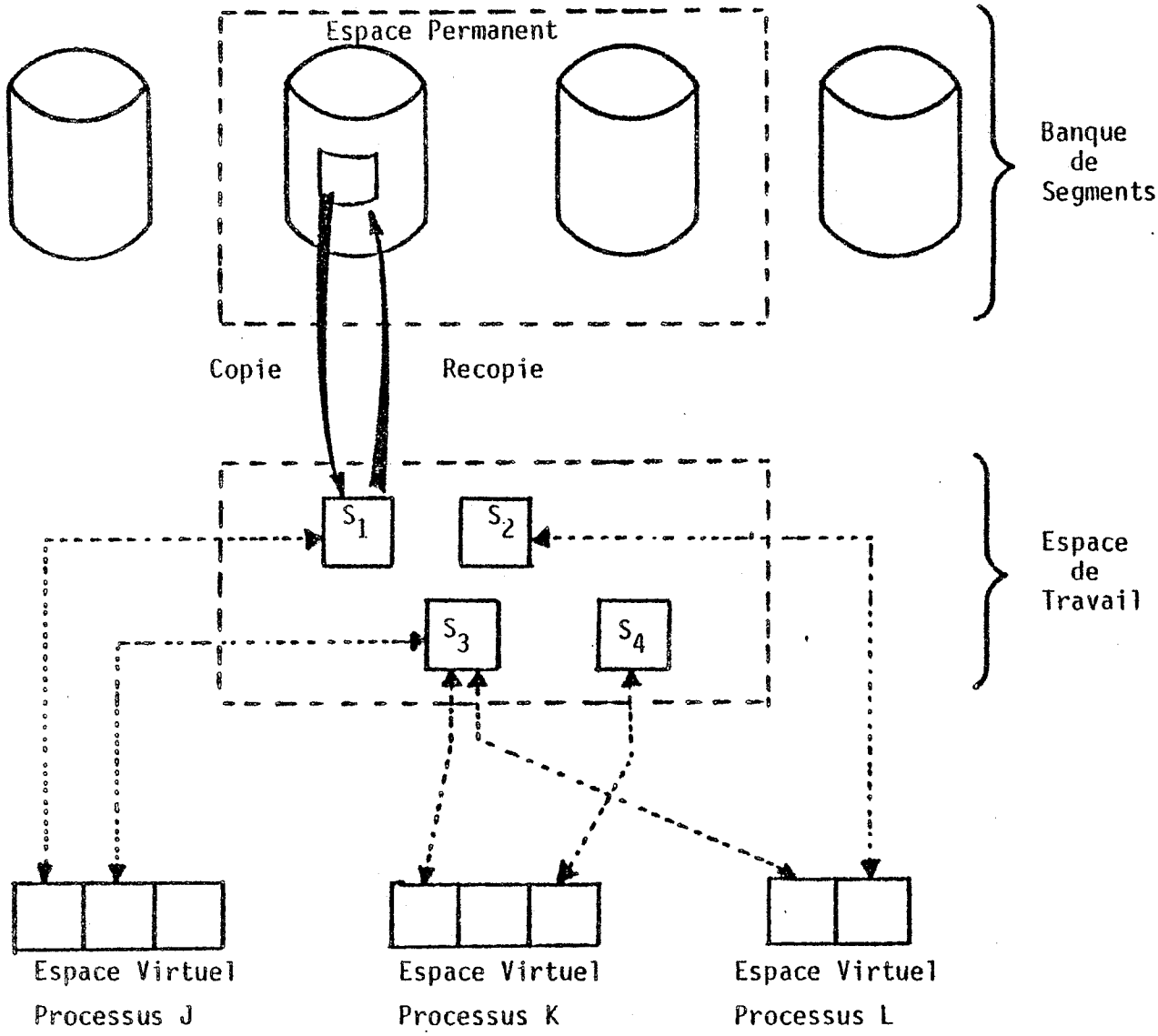

Espace Permanent
Banque de Segments
Copie
Recopie
S1
S2
S3
S4
Espace de Travail
Espace Virtuel Processus J
Espace Virtuel Processus K
Espace Virtuel Processus L

Figure 2

Un volume MAS reçoit à sa création une étiquette composée d'un Nom Symbolique (utile à l'exploitation) et d'une clé fixée par le propriétaire. Parallèlement le volume MAS reçoit un nom interne permettant au système d'associer les segments au volume MAS qui les contient. On impose pour les raisons précédemment énumérées qu'un segment soit entièrement contenu dans un Volume MAS.

IV.2.13. Les Espaces Virtuels

Chaque processus dispose d'un espace virtuel qui permet de réunir dans un espace contigu jusqu'à 127 segments. C'est au travers de son espace virtuel qu'un processus peut exécuter les instructions d'un segment (segment de code) ou référencer les données d'un segment (segment de données).

Un espace virtuel est une suite de pointeurs sur des valeurs courantes (copies de segments permanents ou segments temporaires) de l'espace de travail. Momentanément, un pointeur sur une valeur courante peut être invalidé (le segment est détruit ou n'est plus utilisé).

Pour le processus, l'espace virtuel apparaît comme une suite de blocs de 128 K mots.

IV.2.2. Les opérations sur les objets

IV.2.21. Opérations sur les segments

Les opérations de manipulation des segments sont les suivantes [EST 2] :

CREER ([Etiquette de volume MAS pour les segments permanents],
Clé d'accès au segment,
Liste des caractéristiques associées au segment,
[Valeur initiale = valeur courante ou permanente d'un segment préexistant]).

Cette opération crée un segment et retourne au processus créateur le nom unique du segment créé.

METTRE-A-JOUR (Nom unique de segment)

Cette opération permet de remplacer la valeur permanente d'un segment par sa valeur courante. C'est une opération nulle sur les segments temporaires.

SAUVER (nom unique de segment)

Cette opération permet de conserver une valeur courante intermédiaire sans perdre la valeur permanente. On l'appelle Nouvelle Version, la valeur permanente représentant l'Ancienne Version. La nouvelle version d'un segment ne peut être rendue accessible que par la destruction de l'ancienne version. Chaque nouvelle sauvegarde annule la sauvegarde précédente s'il y a lieu.

DETRUIRE (nom unique de segment)
[Version])

Cette opération permet de détruire soit une des versions ancienne ou nouvelle, soit le segment.

IV.2.22. Opérations sur les volumes

Les opérations permettant de manipuler les volumes MAS sont les suivantes :

FORMATER (nom unique d'unité de disque,
nom symbolique de volume,
clé d'accès au volume)

Cette opération a pour but d'initialiser un volume MAS. L'unité de disque doit être déjà allouée et un volume privé doit être déjà monté sur cette unité (opérations ALLOUERPERIPH et MONTERSURPERIPH). Une demande d'autorisation est envoyée à l'opérateur ; si le nom symbolique est agréé, le disque reçoit le format standard des volume MAS ; une étiquette est créée avec le nom symbolique et la clé d'accès indiquée.

MONTER (étiquette de volume MAS)

Cette opération permet de rendre accessible un volume MAS et de l'introduire dans le parc des volumes MAS s'il n'existe pas déjà, en lui affectant un numéro de série.
Le choix de l'unité où le disque doit être monté est fait par le noyau. Le disque étant monté, le noyau vérifie qu'il présente un format correct ; si le nom symbolique est inconnu dans la table des volumes MAS, il y est introduit après choix d'un numéro de série et accord de l'opérateur.

DEMONTER (Etiquette du volume à démonter,
[Etiquette du volume à monter])

Cette opération permet de supprimer un volume MAS, ou de le remplacer par un autre volume MAS sur une unité de disque. Dans le premier cas, l'unité est libérée ; dans le deuxième cas, tout se passe comme pour l'opération MONTER.

PROTEGER (Etiquette d'un volume MAS)

Cette opération permet de rendre exclusif l'accès à un volume. Les demandes d'accès émanant d'autres processus (utilisation des segments de ce volume),postérieurs à la demande de protection seront refusées.

BANALISER (Etiquette de volume)

Cette opération rend le volume accessible à tous les processus.

IV.2.23. Opérations sur les espaces virtuels

Un espace virtuel est constitué de 2 à 128 blocs. Chaque bloc peut recevoir un segment ou rien du tout. Les opérations fournies sont :

CREER-EV (Identification d'un processus,
Adresse de début d'exécution,
Liste décrivant l'état initial des blocs d'espace virtuel).

Cette opération permet de créer un espace virtuel : les blocs d'espace virtuel décrits sont du 1er au $N^{ième}$ caractérisés par une protection et si la protection n'interdit pas tout accès,une identification du segment associé au bloc. Le $0^{ième}$ bloc de tout espace virtuel est implicitement déclaré par le noyau et permet la liaison entre processus et noyau.

LIER (Identification d'un processus,
Numéro de bloc d'espace virtuel,
Protection d'accès,
Identification de segment).

Cette opération permet d'introduire dans l'espace virtuel d'un processu un segment en le fixant dans le bloc spécifié par son numéro. Préalablement le bloc doit être vide.

DELIER (Identification d'un processus,
Numéro de bloc d'espace virtuel).
Cette opération détruit l'association bloc d'espace virtuel-segment. Le bloc d'espace virtuel devient vide (tout accès interdit).

IV.2.3. Réalisation des objets

IV.2.31. L'espace permanent

Le système MAS gère une table des volumes accessibles qui permet d'obtenir, à partir de l'étiquette d'un volume MAS ou du nom interne d'un volume MAS, l'unité de disque qui matérialise ce volume.

Il vérifie la validité de l'association unité de disque-volume MAS grâce à l'étiquette de volume reproduite sur le volume lui-même.

Chaque volume conserve son catalogue des segments permanents.

Chaque entrée du catalogue contient la clé d'accès du segment et l'adresse du descripteur de la valeur permanente du segment. Le descripteur d'une valeur permanente de segment est une page du volume (de même taille qu'une page de segment) qui regroupe toutes les informations relatives à la valeur du segment : taille, mode, enveloppe, ... Il localise les pages du segment dans le volume. La table des volumes accessibles et le catalogue des segments des volumes permettent de localiser les segments à partir de leur nom unique (cf. Figure 4).

Quand un segment possède deux versions conservées comme sur la Figure 4, le descripteur de la valeur permanente contient le pointeur vers le descripteur de la nouvelle version. Les deux versions peuvent référencer des pages communes : ce sont les pages non modifiées d'une version à l'autre.

IV.2.32. L'espace de travail

Un espace virtuel est décrit par une table de segments du mécanisme d'adressage de l'Iris 80 (cf. Figure 3). A chaque bloc de l'espace virtuel correspond une entrée de la table de segments.

Quand le bloc d'espace virtuel est vide, tout accès à l'entrée table de segments correspondante est interdit. Quand le bloc d'espace virtuel a reçu un segment, l'entrée de la table de segments correspondante pointe

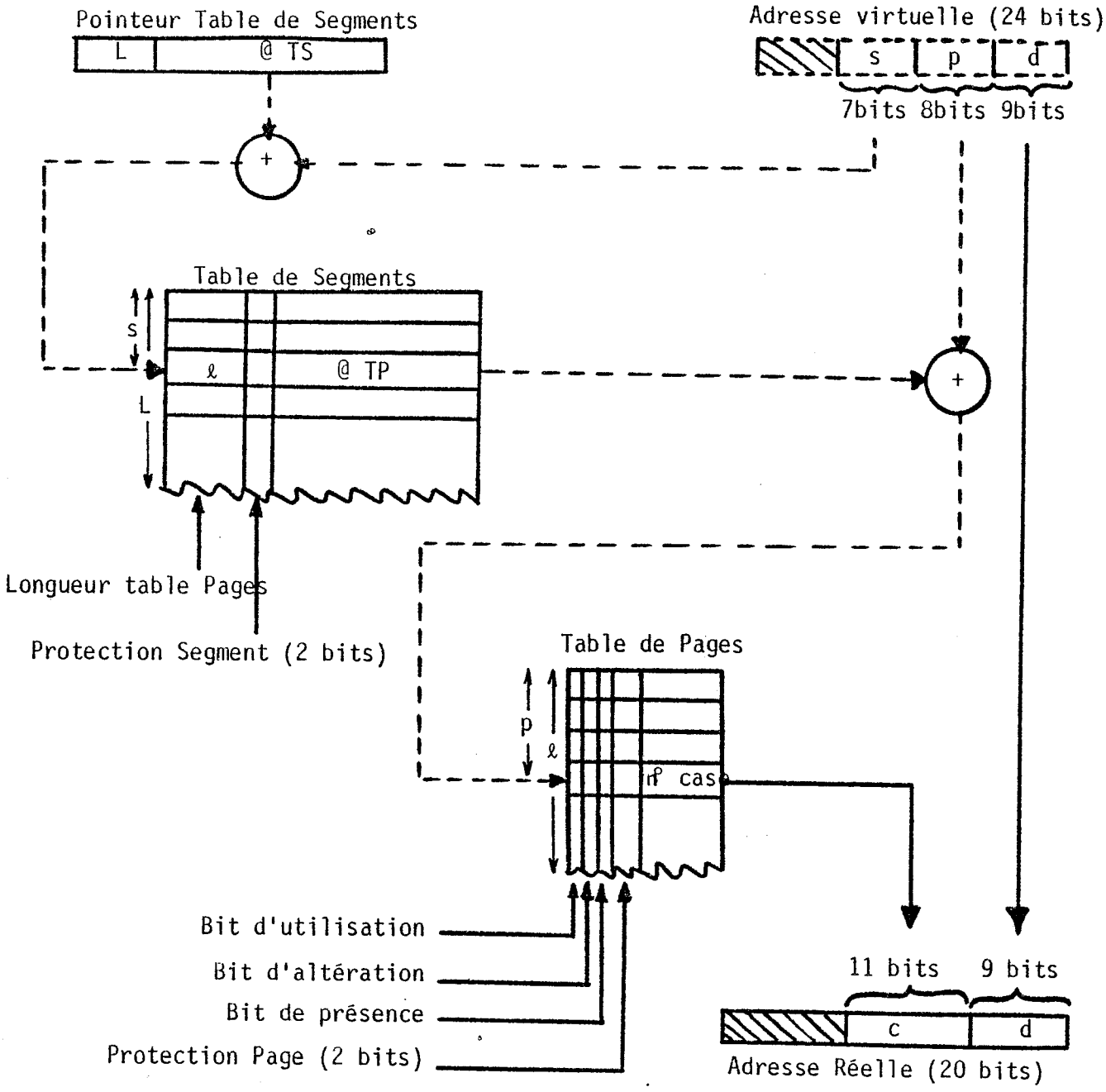

Pointeur Table de Segments
L
@ TS
Adresse virtuelle (24 bits)
s
p
d
7bits 8bits 9bits
+
Table de Segments
s
ℓ
@ TP
L
+
Longueur table Pages
Protection Segment (2 bits)
Table de Pages
p
ℓ
nº case
Bit d'utilisation
Bit d'altération
Bit de présence
Protection Page (2 bits)
11 bits
9 bits
c
d
Adresse Réelle (20 bits)

Figure 3

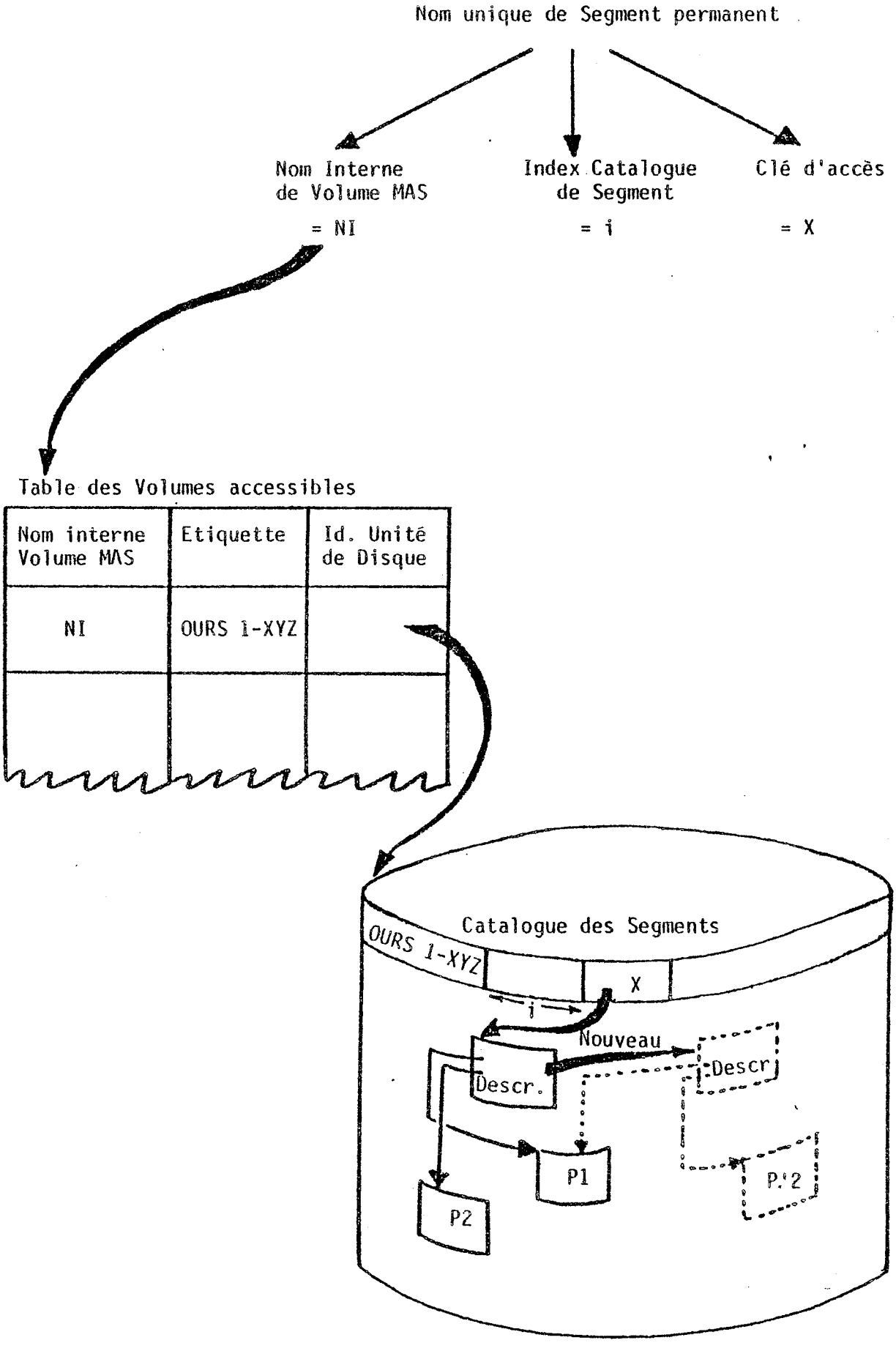

Nom unique de Segment permanent
Nom Interne de Volume MAS
= NI
Index Catalogue de Segment
= i
Clé d'accès
= X
Table des Volumes accessibles
Nom interne Volume MAS
Etiquette
Id. Unité de Disque
NI
OURS 1-XYZ
Catalogue des Segments
OURS 1-XYZ
X
i
Nouveau
Descr.
Descr
P1
P2
P'2

Figure 4

sur une table de pages descriptive de la valeur courante du segment. Le nombre de valeurs courantes utilisées à un moment donné peut être très grand. Toutes les pages de ces valeurs courantes ne peuvent résider ensemble dans la mémoire centrale. On étend donc l'espace de travail à une mémoire secondaire (tambour et disques) gérée par un mécanisme de pagination.

Le nombre très grand de valeurs courantes de segments peut amener momentanément à chasser en mémoire secondaire toutes les pages d'une valeur courante. Dans ce cas, on aurait aimé disposer d'un indicateur table de pages absente, plutôt que de définir une table de page du type "vide". La Figure 5 regroupe les différents cas rencontrés dans la description des espaces virtuels. Les processus J, K, L travaillent sur les segments S1, S2 absents de la mémoire centrale et S3, S4 présents. Les blocs 0 de chaque espace virtuel sont réservés au système ; les blocs, 3 pour le processus J et 2 pour le processus K, sont vides. A chaque valeur courante comme à chaque valeur permanente est associé un descripteur qui permet de retrouver les pages du segment sur les supports de pagination. Par mesure de standardisation, on a choisi de prendre pour un descripteur la taille d'une page. Les 512 mots du descripteur permettent d'y inclure la table de pages de la valeur courante du segment.

Les segments sont partageables ; il faut donc un moyen de reconnaître qu'un segment dispose déjà d'une valeur courante. La table des valeurs courantes qui associe au nom unique d'un segment l'adresse en mémoire centrale du descripteur de sa valeur courante remplit ce rôle.

Grâce à cette table des valeurs courantes, il n'est plus nécessaire de conserver en mémoire centrale le descripteur d'une valeur courante de segment dont toutes les pages sont en mémoire secondaire.

Dans ce cas, la table des valeurs courantes fournit l'adresse du descripteur sur les supports de pagination et la table de pages "vide" puisque le mécanisme d'adressage de l'Iris 80 suppose qu'il existe toujours une table de pages.

La Figure 6 décrit l'accès aux valeurs courantes des segments S1 présent en mémoire centrale et S2 sur supports de pagination.

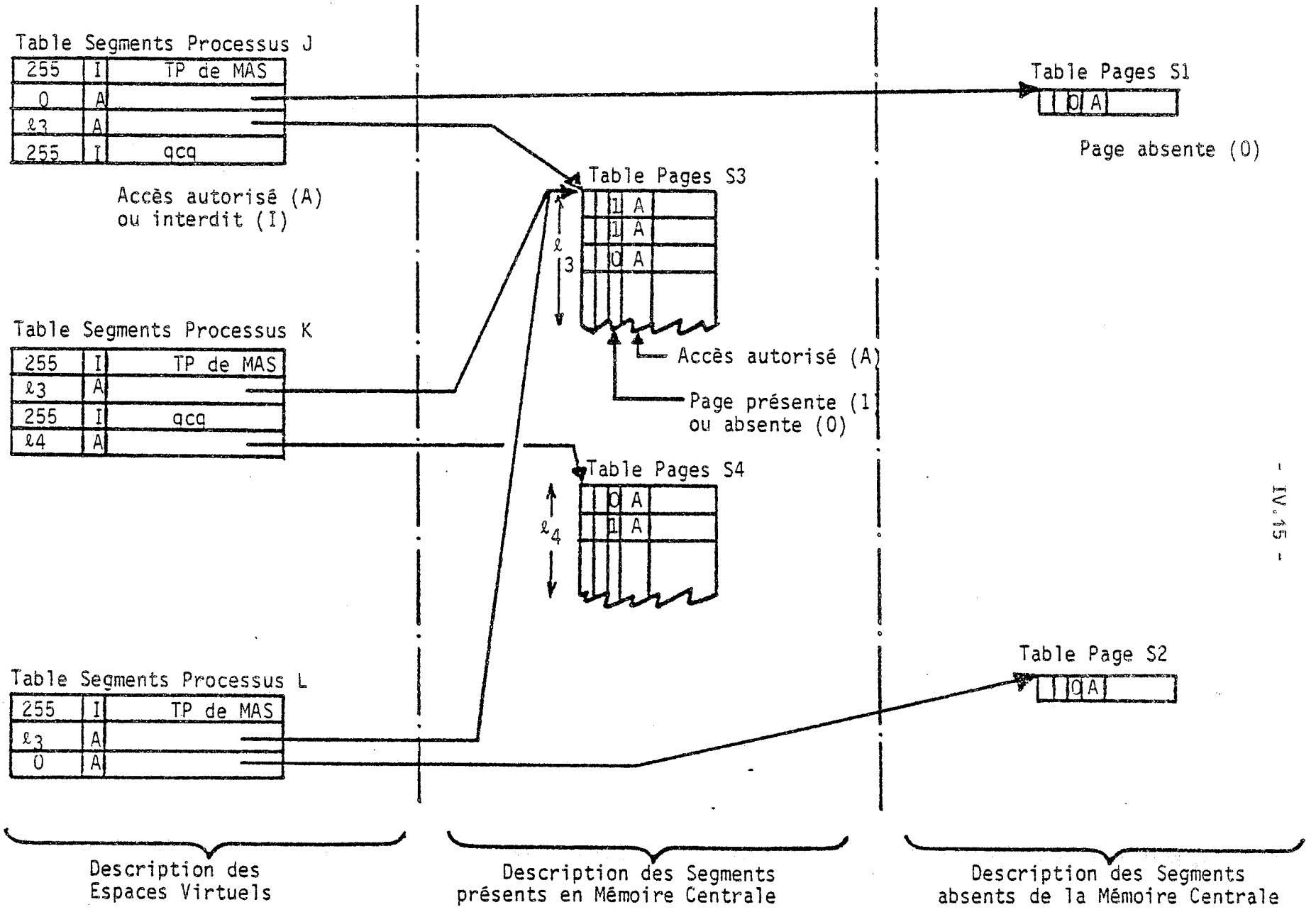


Table Segments Processus J
255 I TP de MAS
0 A
ℓ3 A
255 I qcq
Accès autorisé (A)
ou interdit (I)
Table Pages S1
1 0 A
Page absente (0)
Table Pages S3
1 A
1 A
0 A
ℓ3
Accès autorisé (A)
Page présente (1)
ou absente (0)
Table Segments Processus K
255 I TP de MAS
ℓ3 A
255 I qcq
ℓ4 A
Table Pages S4
0 A
1 A
ℓ4
Table Page S2
1 0 A
Table Segments Processus L
255 I TP de MAS
ℓ3 A
0 A
Description des Espaces Virtuels
Description des Segments présents en Mémoire Centrale
Description des Segments absents de la Mémoire Centrale

Figure 5

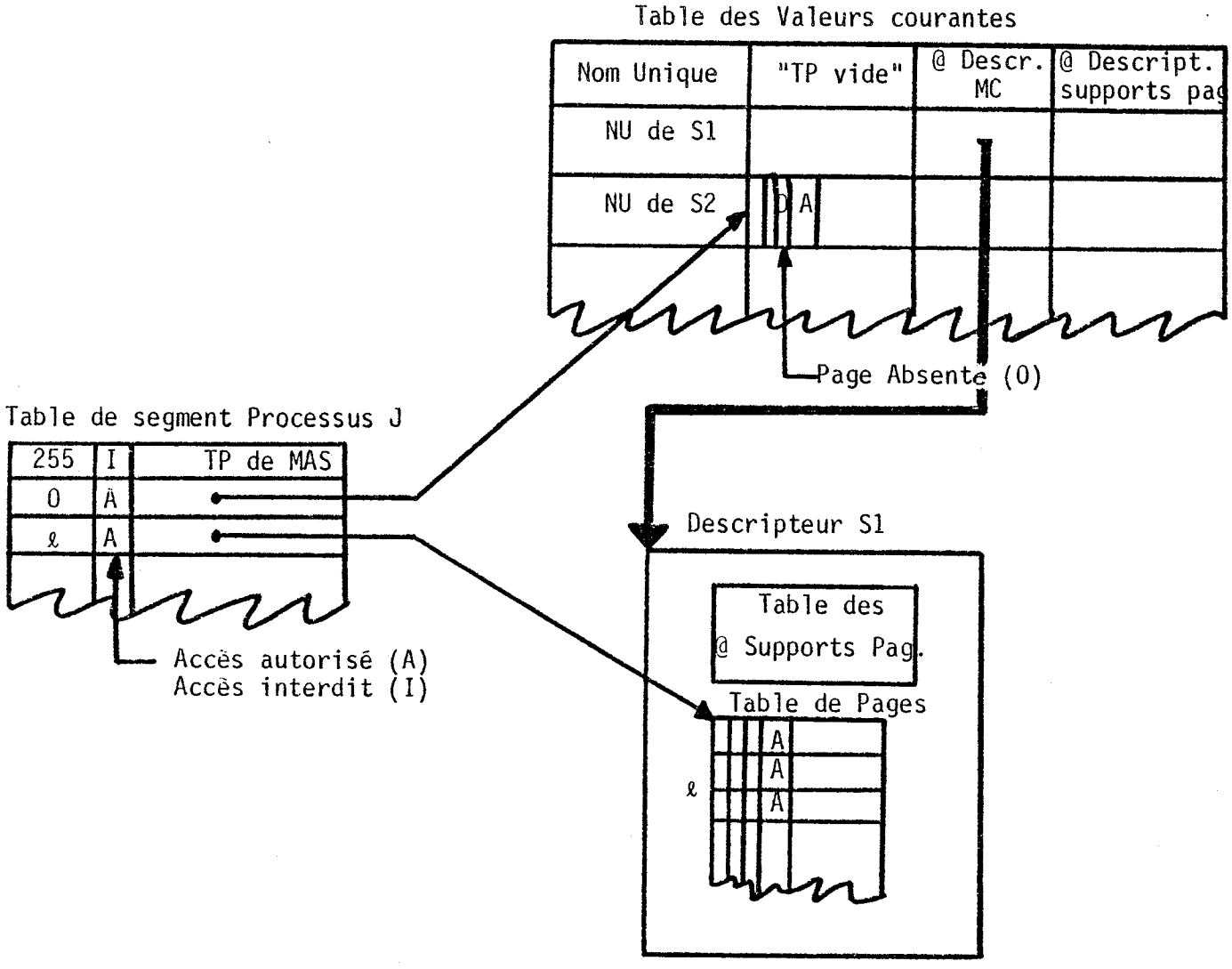


Table des Valeurs courantes
Nom Unique
"TP vide"
@ Descr. MC
@ Descript. supports pag.
NU de S1
NU de S2
D
A
Page Absente (0)
Table de segment Processus J
255
I
TP de MAS
0
A
ℓ
A
Accès autorisé (A)
Accès interdit (I)
Descripteur S1
Table des
@ Supports Pag.
Table de Pages
A
A
A
ℓ

Figure 6

IV.2.4. Efficacité

Le nombre de valeurs courantes de segments contenus dans l'espace de travail du système MAS peut être très important. Dans une tranche de temps donnée, l'ensemble des pages de segments de cet espace de travail effectivement utilisées est forcément restreint. Le but de l'algorithme de gestion de mémoire est de fournir cet ensemble de pages au fur et à mesure des besoins et ceci à un moindre coût.

Un grand nombre d'algorithmes ont été définis à partir des observations du comportement des programmes en milieu paginé [DEN 4] :

- Certains de ces algorithmes ne tiennent pas compte du comportement des programmes (algorithmes du type aléatoire ou premier entré premier sorti). Ces algorithmes sont très faciles à mettre en oeuvre mais pas toujours bien adaptés [BEL 2].

Il existe une version améliorée de l'algorithme du type premier entré premier sorti : l'algorithme d'ATLAS [KIL] introduit un tampon de pagination qui définit un niveau intermédiaire entre l'ensemble des pages virtuelles disponibles en mémoire centrale et l'ensemble des pages virtuelles absentes de la mémoire centrale. Ce tampon de pagination permet de retrouver rapidement une page virtuelle qui vient d'être éliminée de l'ensemble des pages virtuelles disponibles.

- D'autres algorithmes, plus récents, classent les pages réelles en fonction de l'histoire de leur utilisation (classement par ordre de référence des pages dans le temps, par fréquence de référence des pages).

Les algorithmes basés sur l'évaluation de l'espace de travail [DEN, MOR 2] et le contrôle de la fréquence des fautes de page [CHU] sont les plus efficaces de cette classe d'algorithmes.

Malheureusement la mise en oeuvre de ces algorithmes est très coûteuse, si ce n'est impossible, quand le matériel n'est pas conçu pour eux [MOR 2]. Les systèmes HYDRA et SAR ont utilisé une gestion de mémoire préventive plutôt qu'adaptative [WUL, KER 1] : ils utilisent la phase de compilation pour tirer des renseignements sur l'exécution future des programmes. C'est là à notre avis une méthode efficace et peu coûteuse, mais qui suppose l'adaptation du ou des compilateurs au système développé.

IV.2.41. La gestion de mémoire dans MAS

L'algorithme de gestion de mémoire que nous avons retenu est un mélange de différents algorithmes connus. Il est partiellement préventif : il admet des renseignements sur le comportement des programmes (taille d'espace de travail, liste de pages à charger prioritairement en mémoire). Il est indépendant du comportement des programmes quand la ressource mémoire est peu sollicitée et basé sur le contrôle de la fréquence des fautes de page quand la ressource mémoire est très sollicitée. Deux politiques d'allocation de mémoire sont en effet nécessaires pour assurer un bon rapport coût-efficacité :

- une allocation à la demande attribue les cases de la mémoire au fur et à mesure des demandes ;
- une allocation régulée attribue les cases de la mémoire en fonction d'une estimation de l'espace de travail des programmes.

IV.2.411. Le rôle de la gestion de mémoire

Dans le système MAS, le pouvoir de régler la charge de la ressource mémoire est laissé au contrôleur (cf. § IV.4.2). Le rôle de la gestion de mémoire est de répartir au mieux les pages de la mémoire physique entre des processus dont le nombre est fixé par le contrôleur. Ceci répond au souci de ne pas optimiser l'utilisation d'une ressource au détriment d'une autre ressource.

Le contrôleur de MAS doit surveiller en permanence le comportement de chaque ressource. En particulier, il doit prévenir la gestion de mémoire du phénomène d'écroulement [DEN 2]. J. Leroudier et D. Potier [LER 1, LER 3] ont étudié en modélisation un moyen de contrôler facilement la charge de la ressource mémoire et d'éviter l'écroulement du système. Leur modèle a montré que le maintien de l'activité du canal de pagination entre 40 % et 60 % de sa capacité assurait une bonne utilisation de la mémoire. Le contrôleur de MAS utilise ce principe. Il asservit le degré de multiprogrammation du système à l'activité du canal de pagination. L'algorithme de gestion de mémoire agit indirectement sur ce degré de multiprogrammation puisqu'il commande les transferts entre mémoire centrale et supports de pagination. Le contrôleur de MAS doit donc déterminer si la charge du système nécessite ou non une allocation régulée.

IV.2.412. Une allocation de mémoire à la demande

L'allocation à la demande laisse les segments s'étendre dans la mémoire. Chaque fois qu'il est fait référence à une page de segment absente de la mémoire,une nouvelle case de mémoire est attribuée. Quand il ne reste plus de cases à attribuer, il faut récupérer une case déjà attribuée au segment concerné en vidant l'ancien contenu.

Si le segment ne possède encore aucune case, on procède à un retassage collectif. Pour chaque segment utilisé, le nombre de cases attribuées est ramené à la valeur de son enveloppe moyenne définie à l'initialisation ou par l'allocation régulée (cf. § IV.2.413). Notons que l'enveloppe moyenne a été choisie comme référence uniquement pour faciliter le changement de politique d'allocation.

L'ordre de remplacement des pages d'un segment en mémoire est donné par la règle "première entré - première sorti", sauf pour les pages classées prioritaires qui sont les dernières à être sorties de la mémoire.

IV.2.413. Une allocation régulée

Quand le système est chargé, il y a inévitablement beaucoup de segments utilisés. Pour assurer un bon degré de multiprogrammation du système, il faut que la répartition de la mémoire soit plus fine. L'idéal serait de connaître l'espace de travail d'un processus tout au long de son exécution. Malheureusement, l'Iris 80 ne permet pas d'obtenir facilement les chaînes de références des pages d'un programme. Nous nous contenterons donc d'évaluer les tailles des espaces de travail.

a) Evaluation des enveloppes de segment

Jusqu'à présent, les systèmes existants se sont intéressés au contrôle du comportement des processus dans les mémoires virtuelles. Or, il a été constaté que plusieurs processus déroulant le même programme (un compilateur en l'occurrence) présentaient des comportements semblables [BUL, RAV]. Il paraît donc plus intéressant d'associer le comportement aux programmes ou données plutôt qu'aux processus. Dans notre système, les segments sont conservés et manipulés par le noyau. Il est donc possible d'enregistrer des informations sur le comportement des segments et de les exploiter quand un processus utilise ces segments.

Notons que le système MAS permet aussi d'obtenir des renseignements statistiques sur des segments très utilisés comme les compilateurs, les éditeurs, ... Par exemple, le nombre moyen de processus utilisateurs, le pourcentage de chance d'avoir un, deux ou trois utilisateurs à la fois ....

Pour répartir au plus juste les cases de mémoire, on évalue les tailles des espaces de travail des segments : ce sont les enveloppes évaluées des segments. Une enveloppe évaluée est réajustée au cours de l'utilisation du segment en fonction du taux de défauts de page. Ce taux de défauts de page est mesuré toutes les N (nombre que nous n'avons pas fixé) occurrences de défaut de page. D'où la formule :

$$\tau = \frac{N}{\text{Temps d'utilisation}}$$

Pour que le taux mesuré matérialise l'activité de pagination du segment, pour une taille d'espace de travail donnée, on ne comptabilise que les défauts de page correspondant au renouvellement de l'espace de travail. Le taux mesuré est comparé à un taux de référence T (taux "idéal" intuitivement très dépendant du taux d'activité espéré pour le canal de pagination (cf. § IV.3.5)).

Si l'on définit :

- une "baisse du taux" par une diminution non consécutive à l'augmentation de l'enveloppe du segment,
- une "montée du taux" par une augmentation non consécutive à la diminution de l'enveloppe du segment,
- un "tassement du taux" par une variation du taux peu sensible,
- une "détérioration du taux" par une augmentation consécutive à la diminution de l'enveloppe du segment,

alors l'algorithme d'évaluation des enveloppes de segments peut se décrire comme suit :

- Soit $\tau < T$: si l'on détecte une "baisse du taux", on diminue l'enveloppe du segment jusqu'à ce qu'il y ait "détérioration du taux".
- Soit $\tau \approx T$: l'enveloppe du segment n'est pas modifié.
- Soit $\tau > T$: si l'on détecte une "montée du taux", on augmente l'enveloppe du segment jusqu'à l'obtention d'un taux inférieur ou égal au taux de référence ou bien un "tassement du taux".

Si l'on se réfère aux résultats des mesures de comportements de programmes [BUL, RAV], les évaluations d'enveloppe seront moins fréquentes pendant les phases de traitement (phases dites stables) que pendant les phases d'initialisation et de terminaison du traitement (phases très courtes de changement de localité).

Après chaque évaluation d'enveloppe E_i, on calcule l'enveloppe moyenne $\bar{E}_i$ correspondant au comportement moyen du segment :

$$\bar{E}_i = \alpha E_i + (1-\alpha)\ \bar{E}_{i-1}, \text{avec } 0 < \alpha \leq 1.$$

Notons que cette valeur moyenne $\bar{E}_i$ déterminera la valeur initiale de l'enveloppe évaluée du segment lors de sa prochaine utilisation.

b) Répartition des cases de mémoire

Nous avons dit (cf. § IV.2.411) que le nombre de segments utilisés dépend de l'ensemble des processus élus par le contrôleur du système. Chaque segment n'est donc pas sûr d'obtenir un nombre de cases de mémoire égal à son enveloppe évaluée : il dispose d'un nombre de cases aussi proche que possible de son enveloppe évaluée ; c'est l'enveloppe courante du segment.

Il en résulte que le taux de défauts de page mesuré pour un segment ne se rapporte pas à son enveloppe évaluée mais à son enveloppe courante. Dans la mesure où l'enveloppe courante d'un segment ne peut être augmentée, on ne peut que découvrir ou confirmer que l'enveloppe évaluée est supérieure à l'enveloppe courante. Si de nouveaux segments utilisés viennent encore s'ajouter, la situation peut se dégrader dangereusement. Il faut alors réduire arbitrairement des enveloppes courantes de segments pour attribuer un minimum de cases à ces nouveaux segments. Normalement on ne devrait pas arriver à de telles situations dégradées si le contrôleur suspend des processus dès que l'activité du canal de pagination dépasse la borne fixée.

Chaque segment utilisé dispose d'un certain nombre de cases de mémoire pour s'exécuter. Les pages virtuelles du segment vont alors se remplacer en mémoire suivant leur ordre d'entrée (il y a une exception pour les pages classées prioritaires).

IV.2.5. Evaluation critique

Nous rencontrons une limitation inhérente à la gestion de l'espace virtuel par pagination : il est difficile de gérer d'une manière optimale l'occupation en mémoire centrale des segments utilisés.

La gestion optimale de la mémoire centrale réclame des informations précises sur le comportement des programmes. On peut envisager plusieurs manières d'obtenir des informations sur les segments au niveau du noyau :

- L'utilisateur peut indiquer (par l'intermédiaire d'un compilateur ou d'un éditeur de lien "intelligent" [ACH]) à la création d'un segment les pages les plus souvent référencées. C'est en quelque sorte le noyau statique du segment. L'utilisateur peut ensuite demander à modifier dynamiquement l'ensemble de ces pages prioritaires pour traduire l'évolution de localité du segment.

- On peut simuler complètement l'exécution d'un programme et relever une trace des références en mémoire que l'on analyse ensuite [BUL] et qui fournit l'évolution de l'ensemble du travail au cours du temps.

- On mesure périodiquement l'activité des segments au niveau de MAS et on essaie d'en déduire leur localité [ROD]. C'est la technique d'évaluation d'enveloppes qui permet de régler l'occupation en mémoire centrale sur le comportement moyen de chaque segment. Cette technique permet dans une certaine mesure de compenser le fait que des informations dont on pourrait disposer au dessus de MAS ne sont pas transmises explicitement à MAS. Mais elle est complexe et sa mise en oeuvre est coûteuse en temps machine. Etant basé sur l'observation du comportement des segments, elle ne peut réagir instantanément aux changements de localité et elle est par conséquent éloignée de l'optimum pendant les périodes de réajustement. L'enveloppe évaluée d'un segment qui est une moyenne lissée de ses diverses localités ne permet pas de réaliser une allocation fine de la mémoire.

IV.2.51. Partage

Une difficulté supplémentaire dans l'évaluation des enveloppes vient du fait que tout segment MAS peut être partagé entre plusieurs processus, chacun pouvant y accéder en parallèle.

Le partage de segments entre processus paraît avantageuse : nous définissons une unité comme étant le support de n processus cycliques [EST1] ; le code déroulé par ces processus est réentrant et occupe un ou plusieurs segments partagés. Le partage de segments de données peut également être avantageux, par exemple dans le cas de la racine d'une structure arborescente. Ce qui paraît déterminant dans le partage de segments n'est pas tant l'économie de place en mémoire centrale - car sous l'effet de la pagination il n'y a économie que lorsqu'il y a partage effectif de page à un instant donné - qu'un avantage qui est inhérent au partage : la gestion d'une seule version du segment plutôt que de n copies, même si cette gestion est un peu alourdie par le fait du partage.

Quoi qu'il en soit, il est difficile de mesurer l'activité d'un segment utilisé par plusieurs processus en parallèle et d'extraire de ces mesures un comportement moyen valable.

Une utilisation de MAS en exploitation aurait eu entre autres intérêts celui de permettre d'évaluer la corrélation entre la taille de l'enveloppe moyenne d'un segment et le degré de partage de ce segment. Il est encore possible de le faire par simulation, avec une charge synthétique [BUL].

Revenons sur les avantages qui découlent du partage de données.
Une base de données peut être considérée comme un ensemble important de données partagées au niveau du système de gestion de la base de données. Dans l'optique où MAS serait le support d'un système de gestion de base de données, la base elle-même serait un sous-ensemble de la banque des segments MAS [FOR]. MAS peut décharger un système gérant un stock de données communes à plusieurs utilisateurs de tout ce qui concerne l'accès physique aux données. Mais il ne peut évidemment pas assurer la cohérence logique des informations communes. Ceci est du domaine du système supporté par MAS qui doit synchroniser les accès des utilisateurs au niveau logique et veiller à la cohérence interne des mises à jour.

Remarquons encore que la communication et la synchronisation au niveau logique entre utilisateurs différents peuvent être réalisées à travers un ou plusieurs segments partagés par ces utilisateurs.

Le matériel IRIS 80 permet de faire coïncider plusieurs pages virtuelles avec la même page réelle, que ces pages virtuelles appartiennent ou non au même espace virtuel.

Nous aurions pu profiter de ce fait pour permettre aussi le partage de page entre segments MAS. Mais alors les problèmes de cohérence, de mise à jour de version, de protection se seraient trouvés multipliés, sans parler des difficultés nouvelles dans l'évaluation des enveloppes de segments. Les fonctions de traitement de la segmentation se seraient trouvées fort alourdies.

Nous n'avons donc pas retenu cette solution qui, en résumé, est difficile à utiliser et à mettre en oeuvre.

IV.2.52. Protection

Nous venons de voir que le partage de segments entre utilisateurs pose des problèmes de cohérence. La cohérence physique est gérée par MAS, alors que la cohérence logique des informations stockées doit être traitée dans des logiciels externes à MAS.

Une part de la cohérence est facilitée par des mécanismes de protection des utilisateurs contre leurs propres erreurs. Par exemple, un utilisateur travaille sur une copie des données qu'il utilise dans son espace de travail et doit en demander explicitement la sauvegarde dans l'espace permanent. La gestion physique des versions de segments est réalisée par MAS.

Dans le domaine de la protection des utilisateurs entre eux, le problème est plus complexe. Nous avons vu que MAS contrôlait les accès grâce à la vérification du nom unique d'un segment. Mais une politique de protection plus fine que le simple "tout ou rien" réclame des mécanismes plus différenciés que celui-ci.

Le matériel IRIS 80 permet d'associer une protection d'accès à un segment et à chaque page du segment : tous accès, lecture et exécution, lecture seulement, aucun accès. Ces protections sont matérialisées par deux bits dans la table des segments d'un processus et dans les entrées de chaque table de page (cf. § IV.2.2). La protection résultante sur une

page d'un segment est celle des deux protections qui est la plus restrictive.

A ce niveau, MAS n'autorise pas tout ce que le matériel permet : il n'y a pas de protection différenciée des pages d'un segment. Nous considérons en effet que l'unité d'information est le segment et non la page.

Du point de vue des systèmes supportés par MAS, un segment n'est que le support des objets qu'ils réalisent. La protection des objets réalisés par les systèmes doit donc être mise en oeuvre au niveau des systèmes eux-mêmes et non au niveau de MAS.

MAS n'aurait guère pu offrir des mécanismes de protection plus perfectionnés sans augmenter les coûts en temps d'interprétation. L'expérience des anneaux de MULTICS [ORG] et celle des capacités d'HYDRA [WUL] ont montré que pour maintenir les coûts dans des bornes raisonnables, les mécanismes de protection fine devaient être câblés.

IV.3. LES PERIPHERIQUES

Un programme quel qu'il soit a besoin de communiquer avec son environnement, ne serait-ce que pour recevoir des données d'entrée et fournir des données de sortie. Une bonne part des communications s'effectue avec les fournisseurs ou récepteurs de données de l'ordinateur : lecteur de carte, imprimante, télétype, disque magnétique, ... Ce sont les *organes périphériques* de l'ordinateur, par opposition à ses organes centraux : unités centrales et mémoire centale.

Le tableau qu'offrent les organes périphériques est d'une grande diversité. Leur utilisation est complexe, voire malaisée. L'un des objectifs des systèmes de traitement est d'aider les utilisateurs à réaliser les opérations qu'ils souhaitent sur les organes périphériques. On a pris l'habitude de grouper les fonctions de traitement des entrées/sorties sous le terme générique de *superviseur d'entrée/sortie*, terme vague qui recouvre une grande variété de situations.

Cette variété vient en partie du fait que la complexité des superviseurs croît avec celle des systèmes de traitement dont ils font partie : les systèmes modernes gèrent des usagers en parallèle, chacun pouvant émettre des ordres de transfert de données ; les superviseurs doivent

donc organiser et synchroniser les demandes d'accès aux organes périphériques en protégeant l'intégrité des informations transmises.

L'apparition des systèmes de pagination a d'autre part donné une particulière importance à la réalisation d'algorithmes efficaces pour les transferts de pages entre la mémoire centrale de l'ordinateur et les supports externes de la mémoire secondaire, c'est-à-dire les disques et tambours magnétiques.

Les fonctions de gestion des organes périphériques débordent donc du cadre de la gestion des transferts physiques de données qui est celui des superviseurs d'entrée/sortie. C'est pourquoi nous appellerons *gestionnaire des organes périphériques* l'ensemble de ces fonctions. Le gestionnaire contient le superviseur d'entrées/sorties.

Ce que fait un gestionnaire dépend étroitement des spécifications fonctionnelles qui fixent son rôle dans l'ensemble du système.

IV.3.1. Les objets : périphériques standard

Nous allons situer les principes qui ont guidé la réalisation de notre gestionnaire parmi ceux qui caractérisent d'autres systèmes connus.

Le système CP/67 [MEY] fonctionne comme un "générateur de machines virtuelles" et fournit à chacun de ses utilisateurs des copies conformes de la machine physique. Le rôle du superviseur de CP/67 est alors de réaliser l'interface entre les périphériques virtuels des utilisateurs et les périphériques physiques [BE]. Les utilisateurs font usage de leur machine virtuelle comme ils le feraient de la machine réelle.

Tel n'est pas le but que nous nous sommes fixés. Nous voulons au contraire dégager l'utilisateur de la gestion matérielle des organes périphériques physiques, c'est-à-dire lui présenter un interface éloigné du matériel.

Comme MAS est un noyau de système, cet interface doit être assez général pour convenir aux systèmes que nous voudrions implanter. Nous avons donc défini un niveau intermédiaire entre l'accès fichier et l'accès par programme canal.

C'est pourquoi nous définissons les *périphériques standard* comme étant les *ressources logiques* mises à la disposition des utilisateurs. Nous suivons en cela la politique générale adoptée pour la réalisation de MAS

[VAT2] qui, comme MULTICS [ORG] ou EMAS [WHI], n'alloue pas directement des ressources physiques aux utilisateurs, mais plutôt des *fonctions d'accès* aux ressources, qui apparaissent comme des objets typés : les ressources logiques.

La définition de périphériques standard dégage donc l'utilisateur de la gestion matérielle des organes physiques et lui permet également d'accéder à tous les périphériques de la même façon, donc de changer de périphérique sans devoir reconsidérer la structure de son programme.

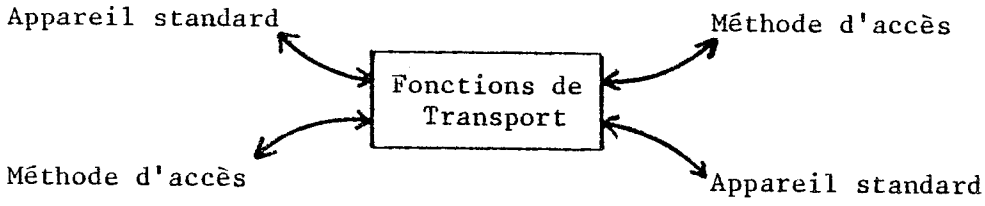
La syntaxe d'une instruction de transfert sur un périphérique standard est la même, quel que soit le périphérique matériel auquel elle s'applique, à la différence de systèmes comme OS360 [IBM1] ou SIRIS 8 [CII2] pour lesquels les dissemblances matérielles entre périphériques se retrouvent à certains niveaux de l'interface avec les utilisateurs.

L'interface standard et polyvalent que nous offrons est appelé *protocole de transfert appareil standard*.

IV.3.2. Protocole de transfert appareil standard

Les principes que nous venons d'invoquer sont les mêmes que ceux qui ont présidé à l'élaboration du "protocole appareil virtuel" (PAV) dans le réseau Cyclades [VIV]. Ces principes, appliqués à deux cas distincts, ont donné lieu à des réalisations différentes.

Le "protocole appareil virtuel" fournit une méthode standardisée pour accéder à des périphériques à travers le réseau : chaque périphérique physique est muni d'un dispositif logiciel nommé adaptateur dont le rôle est de lui prêter l'apparence d'un appareil standard ; la méthode d'accès n'a donc besoin que d'avoir un interface avec un tel périphérique standard ; la communication entre adaptateur et méthode d'accès se fait par l'intermédiaire des fonctions de transport standard du réseau.
Un tel protocole peut également être utilisé pour la communication entre appareils standards ou entre méthodes d'accès ; on a alors un protocole d'échange de niveau supérieur.



Les informations échangées sont des lettres ou des télégrammes qui transitent par des tampons du système.

Voyons maintenant ce qu'est notre protocole de transfert appareil standard.

MAS n'a pas pour vocation d'être une composante d'un réseau ; mais on peut parfaitement implanter une station de transport d'un réseau sur MAS [SEF].

Le protocole de transfert appareil standard ne tient donc pas compte des particularités apportées par la présence ou l'absence de lignes de communications avec un réseau. Il s'agit de transférer une certaine quantité d'informations entre l'espace virtuel d'un utilisateur et un support externe, dans un sens ou dans l'autre. La primitive de transfert est l'instruction TRANSFERER dont la forme la plus générale est TRANSFERER (Emplacement Externe, Sens, Zone Virtuelle).

L'Emplacement Externe est désigné sous la forme

(Id. Périph. [, Loc])

où Id. Périph. désigne l'organe périphérique (cf. § IV.3.3)
et Loc (facultatif) est une adresse sur le support externe monté sur l'organe périphérique.

La Zone Virtuelle est la partie de l'espace de travail en mémoire virtuelle (cf. §IV.2.32) impliquée dans le transfert. Elle s'indique sous la forme d'un triplet

(Segment, Déplacement, Longueur)

où Segment identifie le segment contenant la Zone Virtuelle,
Déplacement donne la distance du début de la Zone Virtuelle par rapport au début du segment,
Longueur est la longueur de la Zone Virtuelle.

L'instruction TRANSFERER qui matérialise le protocole de transfert appareil standard se présente donc sous la forme générale

TRANSFERER ((Id.Périph [,Loc]), Sens, (Segment, Déplacement, Longueur))

Revenons brièvement sur le "protocole appareil virtuel" du réseau Cyclades pour remarquer que l'on peut prendre la fonction TRANSFERER comme fonction de transport, en identifiant la Zone Virtuelle avec le tampon contenant les lettres et télégrammes à transmettre et en donnant comme Id.Périph l'identification du coupleur réseau [SEF]. Noter qu'on n'utilise alors qu'une faible partie des possibilités offertes par la primitive TRANSFERER.

On peut imaginer - mais la chose n'a pas été faite - que l'on puisse inversement utiliser la primitive TRANSFERER comme méthode d'accès à des périphériques accessibles par le réseau en l'interfaçant avec la station de transport.

Il y a donc une certaine réciprocité dans l'usage du "protocole appareil virtuel" et du "protocole de transfert appareil standard", à un interface près. Mais leur vocation est différente.

Le gestionnaire des organes périphériques fournit deux primitives liées à la primitive TRANSFERER :

- la primitive ARRETER permettant d'arrêter l'exécution d'une ou de plusieurs instructions TRANSFERER émises antérieurement ;
- la primitive TESTER permettant d'obtenir l'état courant de l'exécution d'une instruction TRANSFERER émise antérieurement.

IV.3.3. Aspects de l'allocation

Les utilisateurs disposent d'appareils standard. Le rôle du gestionnaire est de transformer des demandes d'opérations logiques (TRANSFERER) sur ces appareils standard en des suites d'actions correctement synchronisées sur des organes périphériques physiques.

Or, pour des raisons d'encombrement et de coût de la configuration, les organes périphériques sont peu nombreux par rapport aux besoins. MAS est conçu pour gérer de nombreux usagers en parallèle ; chacun peut émettre des ordres de transfert de données. La pénurie d'organe va obliger à en organiser le partage entre les usagers.

Notre hypothèse, analogue à celle d'HYDRA [WUL, LEV], d'allocation des ressources par niveau nous fait reporter à une autorité compétente au-dessus de MAS - que ce soit un répartiteur (cf. IV.4), ou un symbiont, ou un système de gestion de fichiers, ou toute autre unité ou machine chargée de réaliser une certaine politique d'allocation de ressources - toutes les décisions que le partage rend nécessaires en fonction de critères de répartition qui lui sont propres.

La tâche du gestionnaire se borne à fournir les mécanismes d'allocation/libération des périphériques et à garantir la sécurité des usagers. Il alloue les organes à ceux qui en font la demande et en interdit l'accès à ceux qui n'y sont pas autorisés (cf. § IV.1).

L'allocation se fait au moyen de la primitive

ALLOUERPERIPH (Type [, Nom Symbolique]).

L'utilisateur peut demander l'allocation d'un périphérique quelconque du type qu'il donne en paramètre (paramètre "Nom Symbolique" absent), ou bien l'allocation du périphérique particulier identifié par un Nom Symbolique si celui-ci est transmis en paramètre. Un nom symbolique, fixé à la génération de la configuration, sert à désigner "en clair" un périphérique ; il peut correspondre à une localisation géographique. Exemple : MT03, LP01, ...

Le nom unique du périphérique alloué est fourni en retour au demandeur si l'allocation s'est déroulée avec succès. C'est ce nom unique qui sera transmis en tant qu'Id.Périph (cf. § IV.3.2) en paramètre de la primitive TRANSFERER.

L'utilisateur peut libérer un organe périphérique dont il n'a plus besoin au moyen de la primitive

LIBERERPERIPH (Nom Unique).

Le nom unique transmis en paramètre est alors détruit et le périphérique devient allouable.

IV.3.4. Réalisation : le gestionnaire des organes périphériques

Plaçons-nous dans le cas où un utilisateur de MAS, c'est-à-dire un processus, fait une demande de transfert par l'intermédiaire de l'instruction TRANSFERER (cf. § IV.3.2). La zone de travail qu'il désigne se trouve dans son espace virtuel. Elle n'est donc a priori pas entièrement dans la mémoire centrale de l'ordinateur.

Or, les *unités d'échange* [CII1] de l'IRIS 80 qui réalisent les transferts effectifs, exigent des adresses réelles en mémoire centrale.

Le gestionnaire doit donc dans un premier temps transformer le triplet (Segment, Déplacement, Longueur) identifiant la zone virtuelle de l'utilisateur en une liste d'adresses réelles en mémoire centrale ; la zone virtuelle devra être bloquée en mémoire centrale pendant tout le temps que durera le transfert.

Le gestionnaire fait appel à une fonction du module de segmentation/pagination [VAT1] pour réaliser le blocage et la traduction d'adresse. Remarquons qu'à cette occasion, le module de segmentation/pagination peut émettre une ou plusieurs demandes de transfert depuis la mémoire secondaire vers la mémoire centrale.

Le gestionnaire est donc constitué de deux modules, l'un réalisant le traitement préalable exposé plus haut, l'autre traitant les transferts effectifs [TAR1].

Le premier module, nommé ESLOGIQ, réalise l'interface entre le protocole de transfert appareil standard qui se réfère à des espaces virtuels, et le monde réel qui est celui du second module.

Le second module, nommé ESPHYSIQ, réalise

- la construction des *programmes canaux* qui commandent les transferts effectifs d'information,
- le lancement des opérations sur les périphériques,
- le traitement des interruptions issues des organes périphériques.

Son travail est donc celui d'un superviseur d'entrées/sorties selon la terminologie classique.

Il traite les demandes de transfert que lui transmet le module ESLOGIQ, mais aussi les demandes issues du module de segmentation/pagination et du module de gestion des volumes MAS. Ceci lui confère une position de clef de voûte (voir Figure 7) dans l'édifice utilisateurs-MAS.

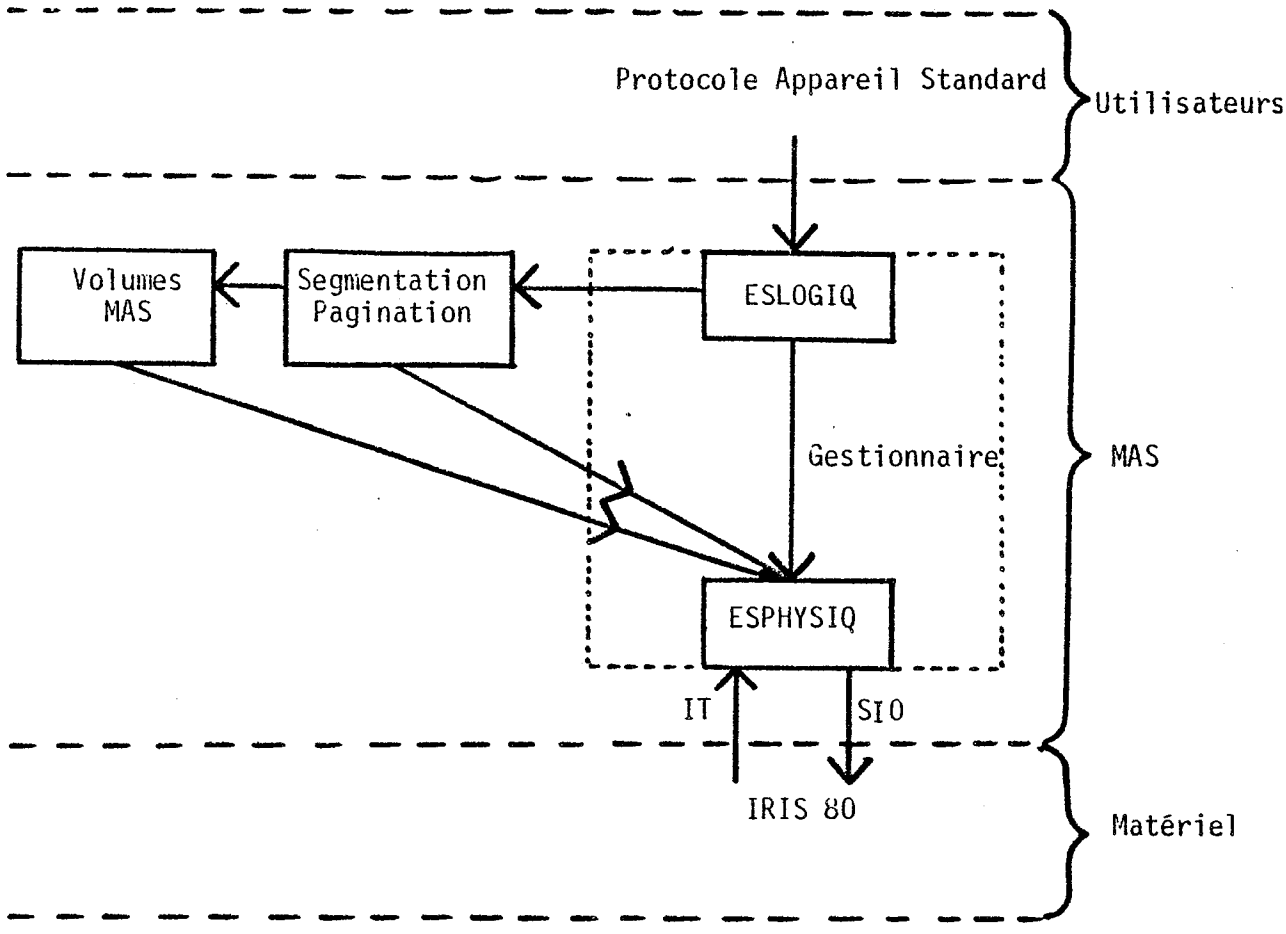


Figure 7 - MAS et le gestionnaire des organes périphériques

IV.3.5. Efficacité

Des études importantes ont été menées sur les systèmes multiprogrammés à mémoire gérée selon une technique de page à la demande afin de déterminer les conditions de l'efficacité de tels systèmes [DEN1, BUZ].

La pagination a pour effet d'imposer un échange important d'informations entre la mémoire centrale et la mémoire secondaire (disques et tambours).

L'augmentation de la concurrence entre les processus sur la mémoire - ou vol de page - peut provoquer une dégradation brutale des performances connue sous le nom d'*écroulement* [DEN2]. Les études ont montré que ce phénomène était conditionné par un déséquilibre entre les intervalles de temps entre fautes de pages et le temps de service de pagination [BELA, LER1, DEN1].

La recherche de l'efficacité maximum du système doit intervenir à plusieurs niveaux :

- régulation de la charge de pagination,
- politique de partage de la mémoire,
- efficacité des transferts de pages.

a) Régulation de la charge de pagination

On a observé, soit par des mesures de comportement de systèmes réels [ADA], soit par simulation de modèles [LER3], qu'il existait un *seuil d'activité optimale* du système correspondant à une certaine activité de la pagination. Cette activité est évaluée, dans [LER3], en mesurant la charge des canaux utilisés par les E/S de pagination. Le modèle est basé sur le taux d'utilisation du canal fictif équivalent à l'ensemble de ces canaux, le taux optimum étant supposé voisin de 50 %. Dans notre cas, cependant, il est impossible de déterminer un tel canal fictif, notamment à cause de l'utilisation possible d'un même canal pour la pagination et pour des transferts d'une autre sorte.

Le mécanisme que nous avons retenu est la mesure du *taux de pagination*, c'est-à-dire le décompte du nombre de transferts de page par unité de temps. Le taux de pagination est donc mesuré au sein du module ESPHYSIQ. La comparaison de ce taux mesuré avec un taux de référence - ou plutôt avec une fourchette de référence - permet de déterminer si l'activité du système est satisfaisante ou si le risque apparaît de la voir se dégrader.

Si le taux de pagination mesuré s'écarte de la fourchette de référence, le contrôleur de MAS (cf. § IV.4.2) en est immédiatement avisé. C'est lui qui peut intervenir pour faire baisser ou augmenter le taux de pagination en jouant sur le degré de multiprogrammation du système.

b) Politique de partage de la mémoire

Ce point est étudié au paragraphe IV.2.4.

c) Efficacité des transferts de pages

La réduction du *temps de service moyen* pour les transferts de page permet de diminuer le nombre des processus en attente d'un transfert de page et autorise une utilisation plus efficace des unités centrales. Le temps

de service moyen peut être réduit, lorsque le débit devient important, par la mise en oeuvre de techniques d'optimisation des séquences de transferts.

Ainsi, pour les tambours ou disques à têtes fixes, nous adoptons une stratégie d'optimisation conforme à l'algorithme d'Eschenbach [WEIN], et qui est peu coûteuse.[COF1] et [BUR] ont développé des modèles de cet algorithme qui corroborent le point de vue de [LEN], obtenu par des méthodes analytiques, selon lequel il offre les meilleurs temps de service.

Pour un disque à bras mobile, les politiques d'optimisation des séquences de transferts par réordonnancement des requêtes entre lesquelles le choix est possible, sont les suivantes [TEO1] :

- PAPS : Premier Arrivé, Premier Servi. Les requêtes sont traitées selon leur ordre d'apparition.
- PPD : Plus Petite Distance. Cet algorithme choisit comme prochaine requête celle qui est relative au cylindre le plus proche de la position actuelle des têtes de lecture/écriture.
- Ascenseur : Le bras de lecture/écriture se déplace de manière monotone dans un sens puis dans l'autre, s'arrêtant au passage pour servir toutes les requêtes. Cette politique admet plusieurs variantes :
  - . Variante SCAN : le bras se déplace du cylindre 0 au cylindre Max et inversement, servant les requêtes chaque fois qu'il s'en présente.
  - . Variante C-SCAN : les requêtes ne sont traitées que lors du déplacement du cylindre 0 au cylindre Max. Le bras ne s'arrête pas lorsqu'il revient du cylindre Max vers le cylindre 0.
  - . Variantes LOOK et C-LOOK : ce sont respectivement un SCAN et un C-SCAN pour lesquels le mouvement du bras s'inverse lorsque la dernière requête dans le sens courant a été atteinte, et non plus le cylindre extrême.
  - . Variante N-step SCAN : les demandes sont réunies par groupes d'au plus N éléments. Lorsque le service d'un tel groupe s'achève, les N plus anciennes demandes constituent le groupe suivant.
- Eschenbach : Le bras de lecture s'arrête au-dessus de chaque cylindre de 0 à Max. Une demande au plus est traitée sur un cylindre en minimisant le délai rotationnel. Le retour du bras du cylindre Max au cylindre 0 est ininterrompu comme dans la variante C-SCAN de l'Ascenseur.

Ces politiques ont été évaluées de diverses manières. Des résultats partiels ont été obtenus par analyse [TEO2, MAN, COF2]. Une bonne étude par simulation, sous des hypothèses acceptables en l'absence de mesures (requêtes indépendantes équiréparties sur le disque), a été publiée dans [LEN].

Ces résultats montrent que PAPS conduit rapidement à saturation, ce qui ne choque pas l'intuition : l'absence d'optimisation n'est justifiée que pour de faibles fréquences de transfert.

PPD et l'Ascenseur-SCAN offrent de bons débits au prix d'une variance des temps de réponse assez élevée. La variante N-step SCAN donne une plus faible variance mais un débit moins élevé.

Eschenbach ne se justifie qu'à forte charge. Ce n'est pas pour nous intéresser puisque la régulation adaptative que nous visons [LER3] exige une charge moyenne.

Il ressort de cette étude que la variante LOOK ou C-LOOK de la stratégie de l'Ascenseur est la meilleure, C-LOOK étant préférable à forte charge.

Nous avons choisi [TAR2], en conformité avec ces résultats et l'avis de [LEN], la variante LOOK de l'Ascenseur.

IV.3.6. Evaluation critique

On peut formuler une remarque qui a déjà été faite dans le cas des segments : c'est que le partage et la répartition des organes périphériques ne sont pas assurés par MAS. Le gestionnaire des organes périphériques ne fait que vérifier les droits des processus utilisateurs par le moyen du nom unique. Il est nécessaire de mettre en oeuvre sur la machine MAS des couches de logiciel gérant le partage des périphériques entre plusieurs usagers [TAR1].

Ainsi, l'utilisation des organes lents d'entrée/sortie (lecteurs de cartes, imprimantes, perforateurs de cartes) ne se conçoit guère sans la réalisation d'un symbiont d'entrées/sorties qui serait le seul à posséder les noms uniques des périphériques lents. Il ferait appel pour la gestion de l'espace disque dont il disposerait aux services d'un système de gestion de fichiers assurant la gestion d'une partie des volumes MAS. Le tout

utiliserait des fonctions de gestion de la console de l'opérateur.

Une remarque d'importance concernant l'allocation des périphériques : la primitive ALLOUERPERIPH (cf. § IV.3.3) ne permet d'allouer qu'un seul périphérique à la fois. C'est un gros inconvénient lorsque des utilisateurs demandent à utiliser plusieurs périphériques en même temps. Il y a alors danger d'interblocage.

Le moyen d'éviter cela aurait été de définir une primitive permettant d'allouer une liste de périphériques (allocation globale) : si tous les périphériques de la liste sont allouables, ils sont alors alloués en même temps et le message d'allocation est renvoyé ; sinon, aucun périphérique n'est alloué et le message est bloqué en attente de libérations ultérieures de périphériques (auquel cas le scénario d'allocation se répète), ou bien optionnellement renvoyé avec un code d'erreur. Il est évidemment possible de définir une unité spécialisée réalisant une telle primitive et de faire transiter toute demande d'allocation par cette unité. En fait, nous avons considéré que la suite des demandes d'allocation de périphériques, lors de la mise en route d'un système, provenait d'unités spécialisées ou de machines (symbionts, SGF, répartiteurs, ...) étant elles-mêmes des composantes d'autres machines et réalisant des services systèmes. On peut alors supposer que le scénario de mise en route d'un tel système est étudié de manière à éviter l'interblocage ; sinon le scénario est à refaire ....

Contrairement à ce qui se passe pour les segments MAS, on ne conserve pas dans le descripteur d'un périphérique l'identité du processus auquel le périphérique a été alloué. Si, à la suite d'une erreur système, le processus en question est détruit, on court le risque de voir le périphérique alloué (donc non allouable à nouveau) à un processus désormais inexistant sans que MAS puisse intervenir. Il faut donc qu'en cas d'erreur dans une machine supportée par MAS, une unité de traitement des pannes dans cette machine ait entre autres pour tâches de libérer tous les organes périphériques éventuellement alloués aux processus détruits [cf. § II.4.2].

IV.4. LES UNITES CENTRALES

Nous avons vu jusqu'à présent que MAS fournit les ressources logiques "segments" et "appareils standard". Les processus supportés par MAS ont également besoin de temps d'exécution sur un processeur physique IRIS 80. MAS doit donc organiser le partage des deux unités centrales physiques de l'IRIS 80 entre les processus demandeurs.

MAS fournit des *unités centrales logiques*, chacune possédant des caractéristiques de performances - c'est-à-dire de rapidité de calcul - différentes. Ces unités centrales logiques sont obtenues par multiplexage des unités centrales physiques.

Les utilisateurs de la machine MAS peuvent définir leurs propres politiques d'affectation des unités centrales logiques aux processus au-dessus de ce premier niveau.

Notre hypothèse de répartition par niveaux des ressources de base (mémoire et unités centrales) est voisine de celle qui a été adoptée dans le projet HYDRA [WUL]. Mais les mécanismes de réalisation sont différents, ainsi que nous l'avons vu dans le cas de la mémoire segmentée (cf. § IV.2.4). Nous allons voir maintenant que c'est aussi le cas des unités centrales.

Les politiques de répartition doivent tenir compte des contraintes caractéristiques des applications prévues. Ces contraintes ne sont pas connues au niveau de la machine MAS.

Il nous semble intéressant de permettre la définition de plusieurs politiques, en dehors de la machine MAS, dont chacune soit adaptée à un type donné de travaux. Nous introduisons la notion de *classe* de processus pour caractériser l'ensemble des processus gérés selon une politique de répartition donnée. Une classe correspond à un type d'application : par exemple, une classe de processus conversationnels, une classe d'applications scientifiques en traitement par lot, une classe pour la recherche documentaire, ...

Cette notion de classes de machines est bien adaptée à la résolution des problèmes d'exploitation ; VM370 [IBM2] et CP67 [AUR] en témoignent.

Il y a donc deux niveaux dans la répartition des ressources de base :

- Le premier, externe à MAS, concerne la répartition à l'intérieur de chaque classe. Une unité est chargée d'appliquer la politique de répartition à l'intérieur d'une classe ; nous appelons *répartiteur* une telle unité. Il appartient au constructeur d'une certaine machine de définir la classe des applications qu'il veut réaliser en définissant les tâches du répartiteur de sa machine et en le programmant en conséquence. Il peut aussi évidemment utiliser un répartiteur existant.

- Le deuxième concerne la distribution des ressources de base - dont l'unité centrale physique - entre les classes. Ce niveau est interne à MAS. Chaque classe a droit à un pourcentage donné de temps d'exécution sur les unités centrales.
Ainsi, les classes paraissent disposer chacune d'un processeur semblable au processeur réel, mais plus lent que lui : c'est l'unité centrale logique. L'unité de la machine MAS qui définit et contrôle ce partage s'appelle le *contrôleur*. Les mécanismes qui réalisent le multiplexage de l'unité centale entre les processus demandeurs sont groupés au sein de HARDEXT dans le *distributeur*.

Nous allons examiner plus en détail le rôle de ces différents composants.

IV.4.1. Les répartiteurs de classe

Une classe dispose donc d'une sorte de processeur fictif unique obtenu par un partage des ressources de la machine physique. Nous n'effectuons pas une simple partition des ressources : lorsqu'une classe n'utilise pas tous ses droits, les autres classes peuvent en profiter momentanément.

Les mécanismes mis en jeu dans la machine MAS permettent à un répartiteur de classe de mener la politique de répartition qu'il entend en affectant à chaque processus de la classe les deux grandeurs caractéristiques suivantes :

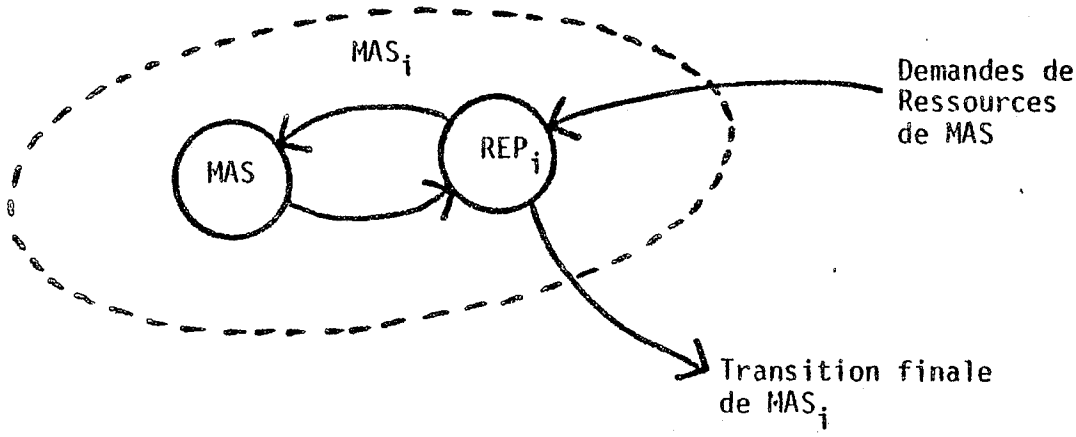
- une *priorité* relative à tous les autres processus de la classe (nombre de 0 à 15) ;
- une durée maximum d'exécution sur une unité centrale de la machine physique. Cette durée est nommée *quantum*.

Nous avons décidé, pour des raisons de simplicité de réalisation au niveau de MAS, de limiter a priori le nombre des classes à cinq. Ce nombre peut être remis en question, mais nous semble suffisant à l'heure

actuelle.

Nous construisons des machines MAS_i composées chacune de la machine MAS et de l'un des répartiteurs de classe. Il existe cinq machines MAS_i : MAS_1, MAS_2, MAS_3, MAS_4, MAS_5. Ces machines sont nécessaires pour la raison suivante :
Si nous voulons garantir que tous les processus d'une certaine classe C_i respectent la politique de répartition menée par le répartiteur REP_i, il importe de ne donner aux utilisateurs aucun accès direct à la machine MAS, mais d'inclure celle-ci dans une machine MAS_i dont l'automate de contrôle asservisse à REP_i toutes les demandes de ressources adressées à la machine MAS.

Aucun processus ne pouvant échapper à un répartiteur de classe, on élimine ainsi les comportements erratiques.



En sus du droit qu'il a de modifier les priorités et les quanta affectés aux processus qu'il gère, un répartiteur REP_i doit encore choisir le moment où il les transfère sur MAS, et interpréter leurs retours de MAS pour les renvoyer soit vers MAS (selon les transitions d'entrée de MAS), soit vers la machine de niveau supérieur (selon les transitions finales de MAS_i).

Exemple 1 : Réception d'un message correspondant à l'arrivée d'un processus sur MAS_i :
Le message est :

- engendré lors de l'interprétation d'une instruction CREERPROCESSUS sur MAS_i ou CREERUNITE sur MAS_i,
- ou renvoyé suivant une transition de l'automate d'une machine incluant MAS_i.

Le répartiteur REP_i attribue alors trois paramètres au nouveau processus :

- le numéro de classe i,
- une priorité, le cas échéant celle qui est demandée et qui figure alors dans le message,
- un quantum, le cas échéant celui qui est demandé ou bien une fraction seulement de celui-ci. Dans le cas d'une création d'unité sur MAS_i, le quantum demandé est infini.

Il renvoie ensuite, s'il le veut, le message vers la machine MAS après y avoir inséré ces trois paramètres, plus une clef permettant au contrôleur de l'identifier.

Exemple 2 : Réception d'un message renvoyé par le distributeur indiquant une fin de quantum :
L'en-tête du message identifie un processus P.

- Dans le cas où REP_i avait attribué à P le quantum initialement demandé, il renvoie P suivant une transition finale de MAS_i. Le processus est terminé en ce qui concerne le répartiteur.
- Dans le cas où le répartiteur n'avait attribué à P qu'une fraction du quantum demandé, il peut par exemple renvoyer sur MAS un autre processus P' ayant même priorité que P et différer le renvoi de P lui-même ; ce procédé permet d'alterner la progression d'un grand nombre de processus en n'envoyant qu'un sous-ensemble sur MAS, après avoir diminué les quanta pour pouvoir renouveler fréquemment le sous-ensemble.

IV.4.2. Le contrôleur

Le contrôleur filtre les demandes d'exécution sur la machine IRIS.
Les ressources "unité centrale" et "mémoire centrale" ne sont pas indépendantes : dès qu'il dispose de l'unité centrale, un processus est demandeur de mémoire.

Le partage de la mémoire principale se traduit par l'évaluation dynamique du nombre de processus autorisés à s'exécuter. Nous appelons *degré de multiprogrammation* ce nombre. Il est calculé de façon à régulariser le taux de pagination autour d'un certain seuil [LER2]. Le taux de pagination est mesuré en temps réel par le superviseur d'entrées/sorties (cf. IV.3.5). Dès qu'il s'écarte d'une fourchette située autour du seuil, le superviseur en avertit le contrôleur qui augmente le degré de multiprogrammation en

cas de baisse du taux de pagination, ou au contraire diminue le degré de multiprogrammation en cas d'augmentation excessive du taux de pagination.

Le partage des unités centrales se traduit par leur allocation selon la technique dite du *tourniquet*. L'intervalle de temps maximum pendant lequel une unité centrale est allouée à un processus est une *tranche élémentaire*. En principe, un quantum se décompose en plusieurs tranches élémentaires. Une tranche élémentaire n'est évidemment allouée qu'à un processus déjà autorisé à partager la mémoire.

Le partage entre les classes se fait de la manière suivante :
A chaque classe C_i on affecte un pourcentage $\%DM_i$ du degré de multiprogrammation. Ces pourcentages sont des paramètres d'exploitation, fixés selon des critères "politiques" d'exploitation. Si DM(t) est le degré de multiprogrammation total évalué à l'instant t par le contrôleur, le nombre de processus de la classe C_i autorisés à s'exécuter est

$$N_i = \%DM_i \times DM(t)$$

à l'arrondi près. Seule exception : lorsque le nombre de processus exécutables de la classe C_i devient strictement inférieur à N_i.
Pour maintenir constant le degré de multiprogrammation DM(t), le contrôleur choisit un processus exécutable dans une autre classe C_j.

De la même manière, on affecte à chaque classe C_i un pourcentage $\%t_i$ du temps d'exécution total pendant une certaine période correspondant au nombre de tranches élémentaires affectées pendant la révolution d'un *barillet*. Le barillet est une liste circulaire d'identités de classes. Un élément de cette liste spécifie la classe à laquelle appartient le (ou les) processus à exécuter pendant une tranche élémentaire de temps. L'élément suivant spécifie l'affectation de la tranche élémentaire suivante.

Soit n la longueur de cette liste. Le contrôleur constitue le barillet où la classe C_i est désignée $\%t_i \times n$ fois, à l'arrondi près. Il répartit de manière homogène les désignations de classes dans le barillet.

Le contrôleur constitue dans chaque classe le sous-ensemble des N_i processus exécutables en choisissant selon l'ordre des priorités décroissantes. Les processus choisis sont renvoyés sur la machine IRIS où le distributeur les prend en compte.

IV.4.3. Le distributeur

Le distributeur est partie intégrante de la machine IRIS (cf. § II.3) constituée du matériel (instructions mode esclave de l'IRIS 80) et du logiciel HARDEXT.

Le distributeur réalise l'application de la machine IRIS sur la machine physique. Il alloue en tourniquet les deux unités centrales à tous les processus actifs sur la machine IRIS, distribuant ainsi des tranches élémentaires de temps d'exécution.

Une exception : les processus constitutifs de la machine MAS : ils sont prioritaires par rapport aux processus utilisateurs de MAS et on ne leur affecte pas de tranche élémentaire : ils peuvent s'exécuter aussi longtemps qu'ils ne provoquent pas de déroutement (entre autres par l'instruction ATTEND).

Le distributeur utilise le barillet constitué par le contrôleur de MAS pour distribuer des tranches élémentaires de temps aux processus utilisateurs qui satisfont aux conditions suivantes :

- ils sont actifs (c'est-à-dire qu'ils ne sont pas sous l'effet d'une instruction SUSPENDRE) et dans l'état IRIS de l'automate de MAS (ils n'ont pas été déroutés et peuvent dérouler leur propre code) ;
- leurs priorités sont supérieures ou égales à celles des autres processus de la même classe qui satisfont aux conditions précédentes.

Il n'est pas calculé de priorité globale pour les processus car elles fausseraient le partage du temps d'exécution entre les classes. Le distributeur maintient les listes des processus de chaque classe qui sont actifs sur la machine IRIS. Ces listes se décomposent en autant de files d'attente qu'il y a de priorités (16) (le nombre maximum de files est donc 5×16 = 80).

Les files sont gérées par la politique PAPS (Premier Arrivé, Premier Servi). Lorsqu'une classe C est élue par consultation du barillet, le distributeur lance le premier processus de la file non vide correspondant à la plus forte priorité dans la classe C. Si ce processus s'exécute sans déroutement pendant une tranche élémentaire, il est placé en queue de la file à la tête de laquelle il était. Ce mécanisme garantit qu'après un nombre fini de parcours du barillet, tous les processus de priorité

maximale soumis au distributeur dans chaque classe ont été exécutés.

Si un processus est dérouté avant la fin de la tranche élémentaire courante, le distributeur allouera le reste de la tranche à un processus de la même classe jusqu'à ce que l'une des conditions suivantes soit remplie :

- la tranche élémentaire courante est épuisée ou est devenue plus petite qu'une valeur x au-dessous de laquelle il serait coûteux d'activer un processus pour l'interrompre aussitôt ;
- ou bien aucune des files de la classe ne contient de processus actif dans l'état IRIS.

Le distributeur alloue ensuite une tranche élémentaire à un processus de la classe désignée par l'élément suivant du barillet, et ainsi de suite jusqu'à la rencontre d'une classe non vide.

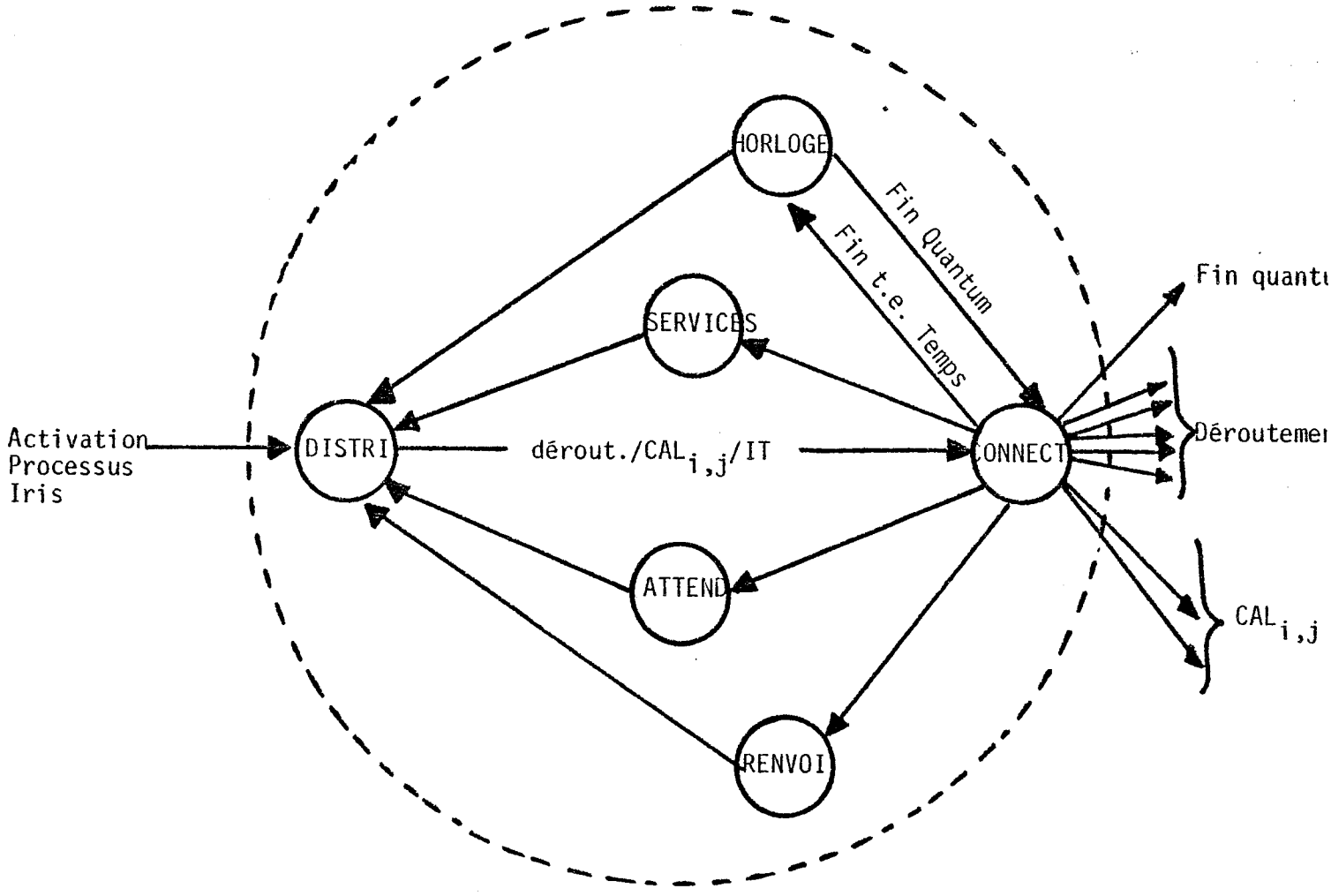
Chapitre V

Décomposition du noyau MAS

V.1. LA MACHINE IRIS

Le noyau MAS que nous venons de présenter au Chapitre IV nécessite pour sa mise en oeuvre et celle de ses processus utilisateurs un support mémoire et un ou plusieurs processeurs d'exécution. Ces supports (mémoire et processeurs) sont fournis par un ordinateur CII Iris 80 : la machine abstraite à la base de la décomposition du noyau MAS apparaît donc comme une machine capable d'enregistrer et d'exécuter des programmes écrits en code Iris 80. Le langage de l'Iris 80 défini par le constructeur ne suffit pas pour construire des niveaux d'abstractions. Il faut encore enrichir ce langage des primitives de communication et d'extension décrites au Chapitre II (création d'unité, création de machine, attente d'un message, renvoi d'un message). Il existe à cet effet une instruction d'appel superviseur : $CAL_{i,j}$. Chaque couple de valeurs i,j de l'instruction CAL peut définir une séquence particulière d'instructions Iris 80 à exécuter. Quatre de ces couples i,j sont réservés pour réaliser les primitives de communication et d'extension. D'autres couples i,j de l'appel superviseur mettent en oeuvre les primitives de gestion des processus (création d'un processus, état d'un processus, ...). En effet tout processus qui consomme du temps d'unité centrale et de la mémoire de l'Iris 80 doit être décrit par un bloc de contrôle dans la machine Iris. Nous avons introduit dans la spécification du système MAS cinq classes de processus (cf. § IV.4) qui définissent un ordre de priorité d'exécution. A ces cinq classes de processus utilisateurs s'ajoute la classe des processus qui réalisent le noyau MAS : ce sont les processus primitifs. Le distributeur d'unité centrale de la machine Iris doit donc gérer six files de processus. Vu du monde extérieur, cela peut apparaître comme six types de machine Iris dont les unités centrales sont plus ou moins rapides.

Si l'on disposait du mécanisme de communication et d'extension présenté au Chapitre II pour décrire la machine Iris, elle pourrait être définie par le graphe de connexion d'unités suivant :



M-IRIS

DISTRI : Cette unité reçoit les messages descripteurs des processus Iris à exécuter dans une des six files de processus. Elle gère les deux unités centrales Iris 80 et les alloue pour des tranches élémentaires aux processus qu'elle exécute, en fonction de leur classe et de leur priorité (cf. § IV.4.3). Les processus sont interrompus dès l'occurrence d'un déroutement, d'une instruction $CAL_{i,j}$ ou d'une interruption de fin de tranche élémentaire de temps.
Leur descripteur est alors transmis à l'unité CONNECT.

CONNECT : Cette unité est chargée d'analyser les causes d'interruption des processus. L'interruption peut être due à l'appel d'une fonction des autres unités de la machine Iris : fin de tranche élémentaire de temps, création de processus, d'unité, ..., primitives ATTEND et RENVOI. Les autres causes d'interruption sont associées à des

numéros de pattes de sortie de la machine Iris. S'il y a sortie de la machine Iris, il faut reprendre le contexte d'exécution associé au message descripteur du processus, au moment de l'entrée dans la machine Iris. Ce contexte comprend l'identification d'un automate de transition d'une machine et un état courant dans cet automate. Le numéro de patte de sortie, sauf erreur, doit conduire à un nouvel état de l'automate donc à une nouvelle unité ou machine.

a) S'il s'agit d'une unité, le message sorti de la machine Iris est chaîné à la file de messages d'entrée de l'unité. Ce dépôt de message peut à son tour débloquer le descripteur d'un processus en attente de message sur cette file d'entrée. Ce descripteur sera alors sorti de l'unité ATTEND où il était entré sur demande d'acquisition de message.

b) S'il s'agit d'une machine, le contexte d'exécution associé au message est à nouveau empilé pour prendre le nouveau contexte d'exécution correspondant à la machine appelée. Ce nouveau contexte d'exécution définit une unité initiale destinataire du message sorti de la machine Iris.

SERVICES : On regroupe dans cette unité tous les services fournis par la gestion des processus et le mécanisme d'extension : création de processus, d'unité, de machine, demande de l'heure, ...

HORLOGE : Cette unité décompte les tranches élémentaires de temps consommées par les processus. Si le processus a consommé un temps égal au quantum qui lui est alloué, son descripteur est envoyé à l'unité CONNECT pour sortir de la machine Iris. Si ce n'est pas le cas, le descripteur du processus est renvoyé au distributeur. Notons que les processus primitifs du noyau MAS ont des quantum infinis.

ATTEND : Cette unité traite les demandes d'acquisition de messages des processus constitutifs d'unités. Elle consulte la file des messages d'entrée de l'unité pour en retirer un,et renvoyer le descripteur du processus muni du message au distributeur. Dans le cas où la file des messages d'entrée est vide, le descripteur du processus est bloqué en attente d'un message.

RENVOI : Cette unité traite les demandes de restitution de messages des processus constitutifs d'unités. Elle consulte l'état de l'automate de transition associé au message pour en déduire, sauf erreur, un nouvel état de l'automate et une unité ou machine destinataire (voir CONNECT). Le descripteur du processus ainsi débarrassé de son message est renvoyé au distributeur.

V.2. LES UNITES DE GESTION DE RESSOURCE

Le but du noyau MAS est de dégager les processus qui s'exécutent sur l'Iris 80 des contraintes matérielles d'accès aux ressources (programme canal, accès disques, partage des ressources). Les processus utilisateurs du noyau MAS voient les ressources mémoires et unité centrale, les périphériques comme des objets du système et disposent d'opérations primitives pour les manipuler (cf. Chapitre IV).

Partant de la machine initiale, la machine Iris (cf. § V.1), il est possible de créer des unités qui gèrent les ressources physiques de l'Iris 80 et fournissent les opérations primitives voulues.

La décomposition du système en unités est assez facile quand elle est calquée sur le découpage physique des ressources. On obtient alors quatre unités de bases appelées Unités de Gestion de Ressource pour gérer respectivement :

- les périphériques
- les volumes MAS
- les mémoires virtuelles
- la répartition de l'unité centrale.

On a vu, lors de la présentation du protocole appareil standard, que le gestionnaire des organes périphériques se décompose en deux niveaux (cf. § IV.3.) : un niveau logique qui traduit les emplacements en mémoires virtuelles en adresses physiques,et un niveau physique qui exécute les Entrées/Sorties et gère les périphériques. Ce qui donne finalement cinq unités de gestion de ressource. Les processus qui réalisent ces cinq unités sont des processus primitifs créés sur la machine Iris.

V.2.1. Unité de gestion des Entrées/Sorties physiques

Cette unité désignée par ESPHI gère les périphériques et met en oeuvre les opérations d'Entrée/Sortie (cf. § IV.3). Elle utilise trois pattes d'entrée :

E1 = Opérations d'Entrée/Sortie (Transfert, Test, Arrêt)

E2 = Opérations d'allocation de périphériques et de libération

E3 = Interruptions d'Entrée/Sortie.

Elle présente quatre pattes de sortie :

S0 = Erreurs de fonctionnement de l'unité (Débordement de Tables ...)

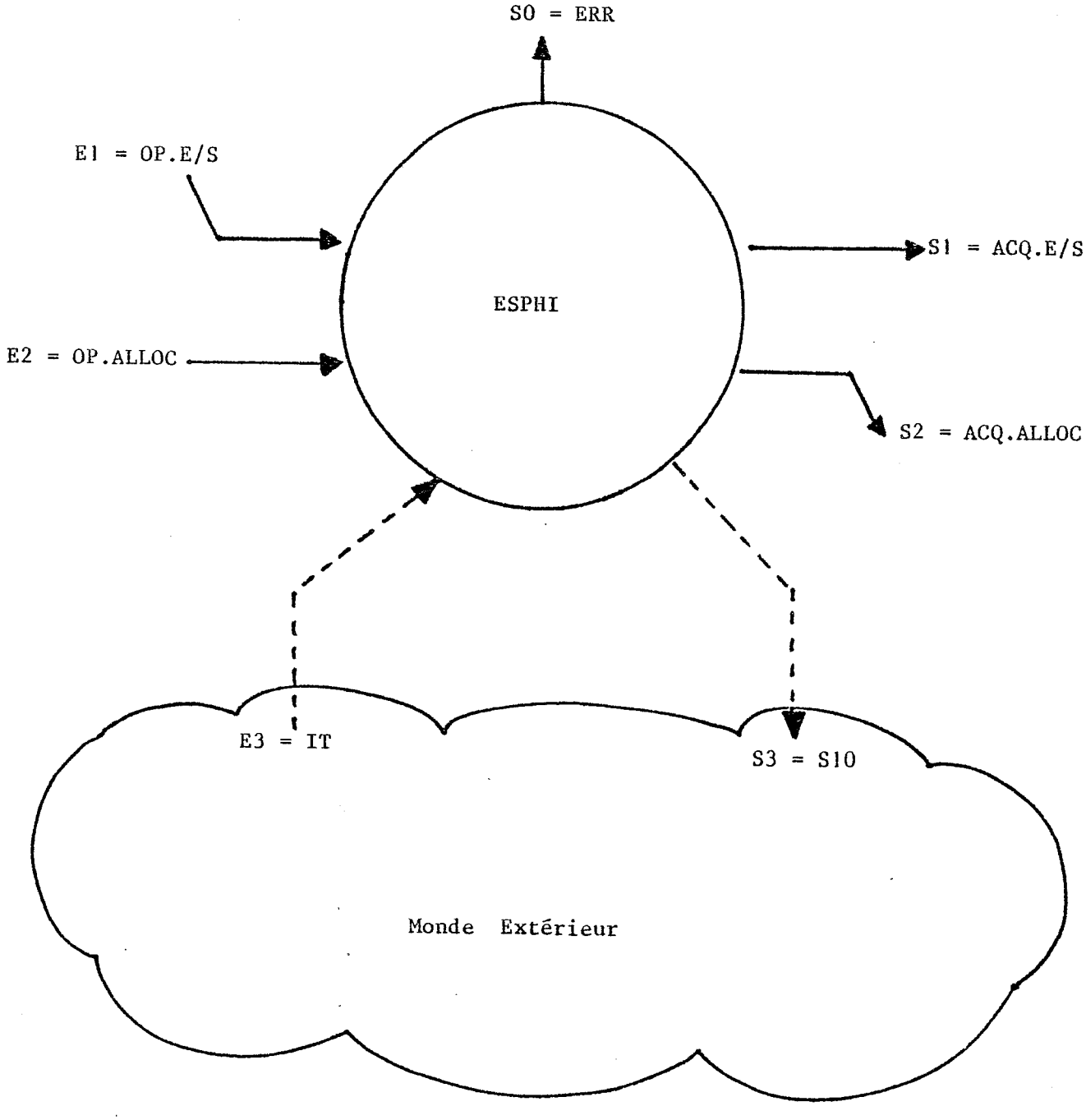
S1 = Acquittements d'Entrée/Sortie

S2 = Acquittements d'allocation

S3 = Initialisation d'un transfert physique (SIO).

Les pattes d'entrée E3 et de sortie S3 communiquent avec le monde extérieur.

On peut représenter l'unité comme suit :



Remarque : Dans cette unité comme dans les suivantes, on ne présente qu'une patte de sortie par type d'opération demandée (Acquittement) pour simplifier les schémas.

En réalité, les unités présentent aussi une patte de sortie pour chaque classe d'erreurs (erreurs d'allocation, sur le format de message ...).

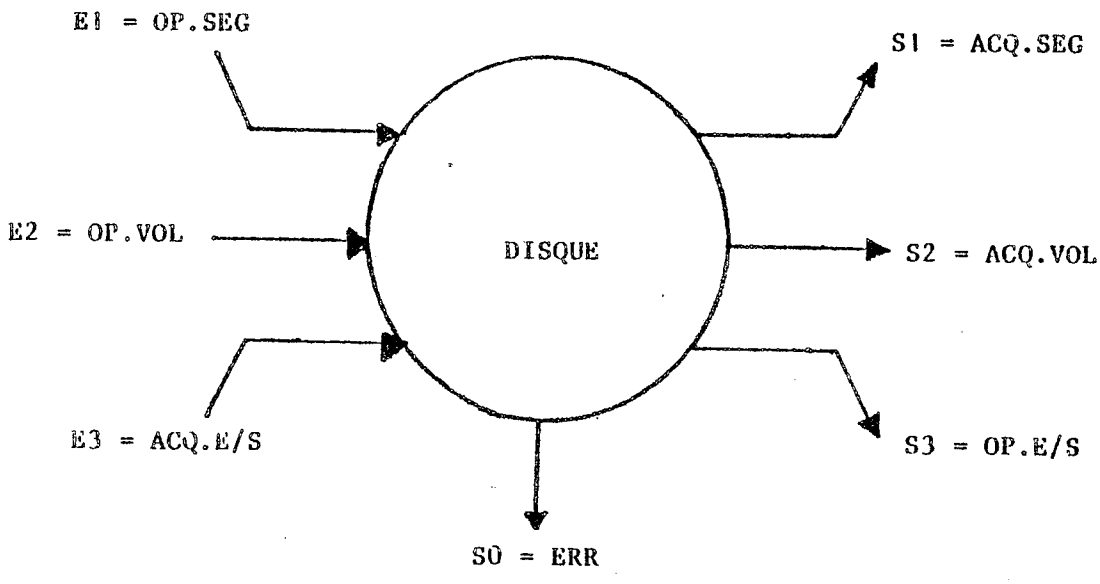
V.2.2. Unité de gestion des Volumes MAS

Cette unité désignée par DISQUE gère les disques qui servent de support à l'espace permanent de MAS et qu'on appelle les Volumes MAS (cf. § IV.2.12). Elle manipule aussi (création, lecture, sauvegarde, destruction) les valeurs permanentes de segments contenues sur ces disques. Cette unité doit être en relation avec une unité (ESPHI en l'occurrence) chargée de mettre en oeuvre les accès aux disques. Les opérations sont réparties sur les trois pattes d'entrée :

- E1 = Opérations sur les valeurs permanentes de segments (création, lecture, sauvegarde, destruction)
- E2 = Opérations sur les volumes MAS (formatage, protection, banalisation)
- E3 = Acquittement d'Entrée/Sortie.

L'unité présente quatre pattes de sortie :

- S0 = Erreurs de fonctionnement de l'unité
- S1 = Acquittements d'opérations sur les segments
- S2 = Acquittements d'opérations sur les Volumes MAS
- S3 = Opérations d'Entrée/Sortie (Transfert).



V.2.3. Unité de gestion des mémoires virtuelles

Cette unité désignée par SEGPAGE réalise la segmentation et la pagination de la mémoire : c'est elle qui crée, modifie et détruit les espaces virtuels des processus et les valeurs temporaires de segment (cf. § IV.2.13)

Cette unité ne se suffit pas à elle-même. Elle utilise une unité qui assure des échanges avec les disques et les tambours de pagination (ESPHI). Parallèlement, elle doit être en relation avec l'unité qui gère les valeurs permanentes de segments (DISQUE) pour en tirer des copies.

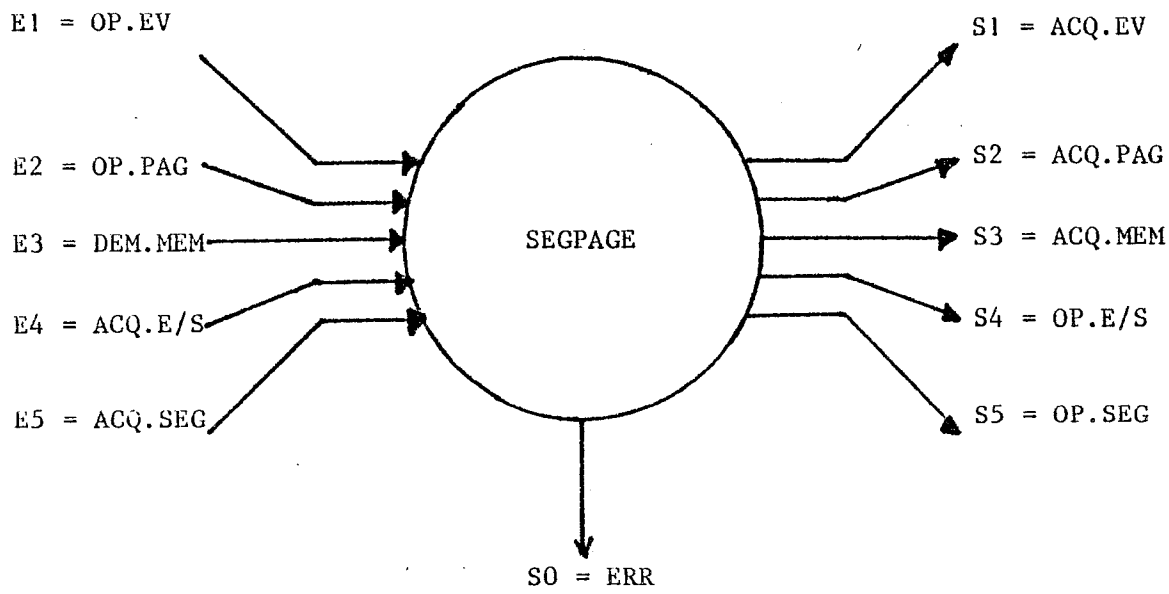
L'unité SEGPAGE présente cinq pattes d'entrée :

- E1 = Opérations sur les espaces virtuels (création, destruction, liaison et déliaison de segments)
- E2 = Opérations de pagination (faute de page, blocage/déblocage de pages)
- E3 = Demande de Ressource Mémoire (activation/désactivation de processu
- E4 = Acquittements d'Entrée/Sortie
- E5 = Acquittements d'opérations sur les segments.

Elle utilise six pattes de sortie :

- S0 = Erreurs de fonctionnement de l'unité
- S1 = Acquittements d'opérations sur les espaces virtuels
- S2 = Acquittements d'opérations de pagination
- S3 = Acquittements ressource mémoire
- S4 = Opérations d'Entrée/Sortie (Transfert)
- S5 = Opérations sur les Valeurs permanentes de Segments (lecture, sauvegarde, destruction).

On obtient la représentation suivante :



V.2.4. Unité de gestion des Entrées/Sorties logiques

Cette unité désignée par ESLOG est chargée de traduire les demandes de transferts conformes au protocole de transfert appareil standard (cf. § IV.3.2) en opérations d'Entrées/Sorties. Cette unité demande le blocage en mémoire centrale des pages de segments impliquées dans le transfert et leur déblocage en fin de transfert.

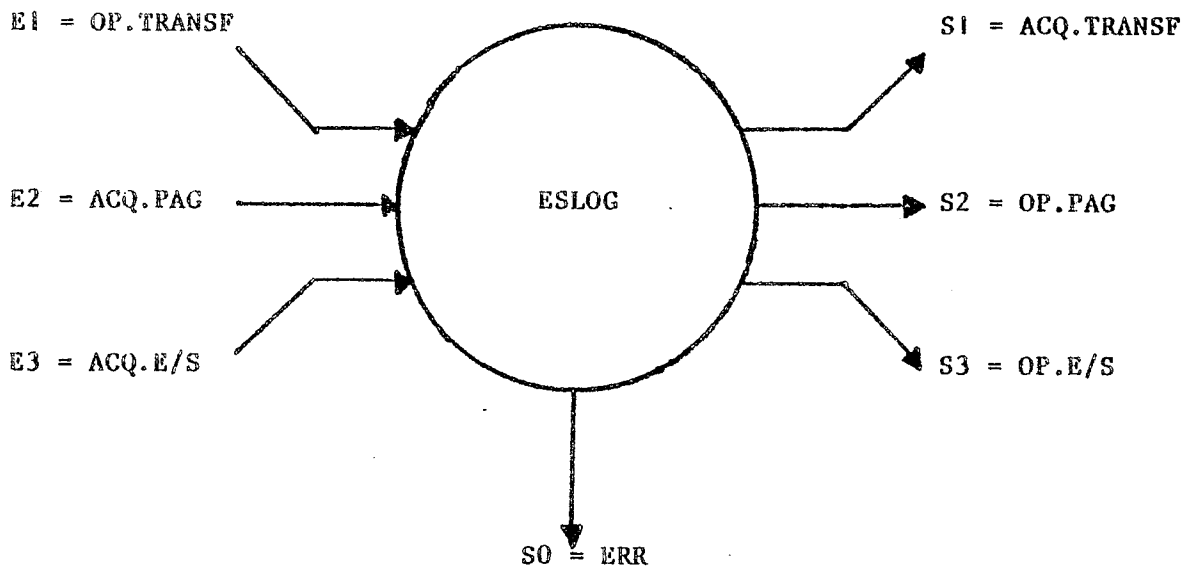
Elle distingue trois pattes d'entrée :

- E1 = Opérations de transferts appareil standard (Transfert-Test-Arrêt)
- E2 = Acquittements d'opérations de pagination
- E3 = Acquittements d'Entrée/Sortie

et quatre pattes de sortie :

- S0 = Erreurs de fonctionnement de l'unité
- S1 = Acquittements de transferts appareil standard
- S2 = Opérations de pagination (blocage/déblocage de pages)
- S3 = Opérations d'Entrée/Sortie (transfert-test-arrêt).

D'où la représentation :



V.2.5. Unité de contrôle des ressources

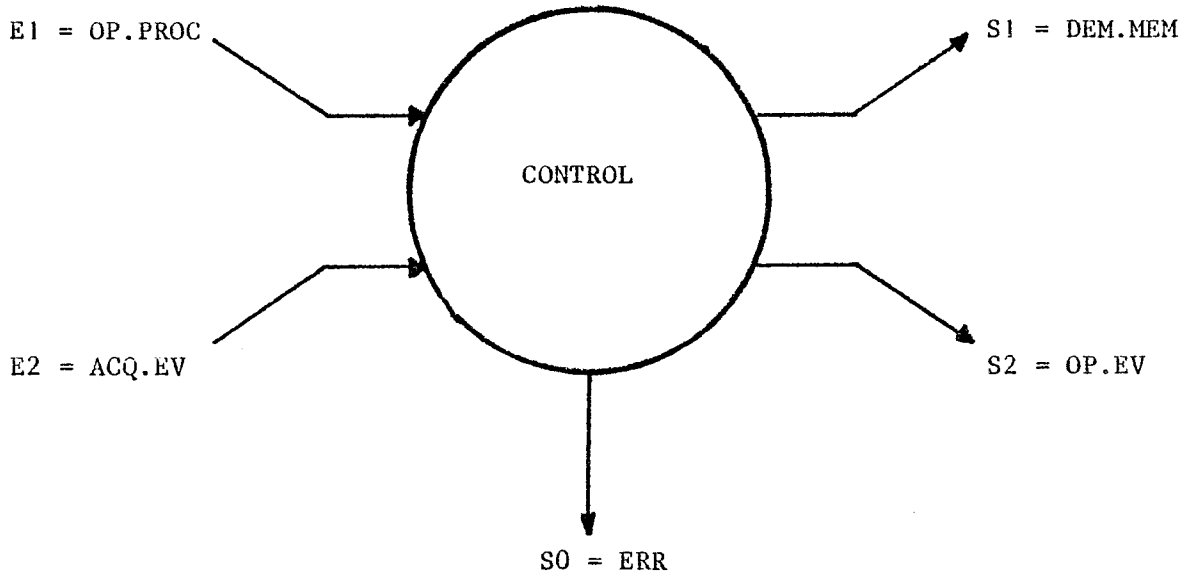
Cette unité désignée par CONTROL contrôle la consommation des ressources unité centrale et mémoire des processus des différentes classes (cf. § IV.4.). Elle reçoit les demandes de création et de destruction des processus MAS et les répercute à l'unité SEGPAGE pour créer et détruire les espaces virtuels des processus. Elle décide de l'activation des processus utilisateurs du système MAS et demande à l'unité SEGPAGE de leur allouer de la mémoire centrale. Elle présente deux pattes d'entrée :

- E1 = Opérations sur les processus (création, destruction, activation, suspension)
- E2 = Acquittements d'opérations sur les espaces virtuels.

Elle utilise trois pattes de sortie :

- S0 = Erreurs de fonctionnement de l'unité
- S1 = Demande de Ressource Mémoire (activation/désactivation de processus)
- S2 = Opérations sur les espaces virtuels (création, destruction).

On obtient donc la représentation suivante :



V.3. LA MACHINE MAS

Le langage de la machine MAS ne doit pas se limiter aux opérations primitives de manipulation des objets du système. Les processus utilisateurs du noyau MAS, qu'on appelle processus MAS, sont d'abord des processus qui exécutent du code Iris 80. La machine MAS est donc construite comme une extension de la machine Iris. Son langage se compose :

- d'instructions du code Iris 80
- de primitives de communication et d'extension
- de primitives de gestion de processus
- de primitives de gestion de mémoire
- de primitives de gestion de périphériques.

Les deux premières composantes du langage sont fournies par la machine Iris (cf. § V.1), les deux dernières sont fournies par les unités de gestion de ressource (cf. § V.2).

Le traitement des primitives de gestion des processus est réparti entre l'unité CONTROL et la machine Iris.

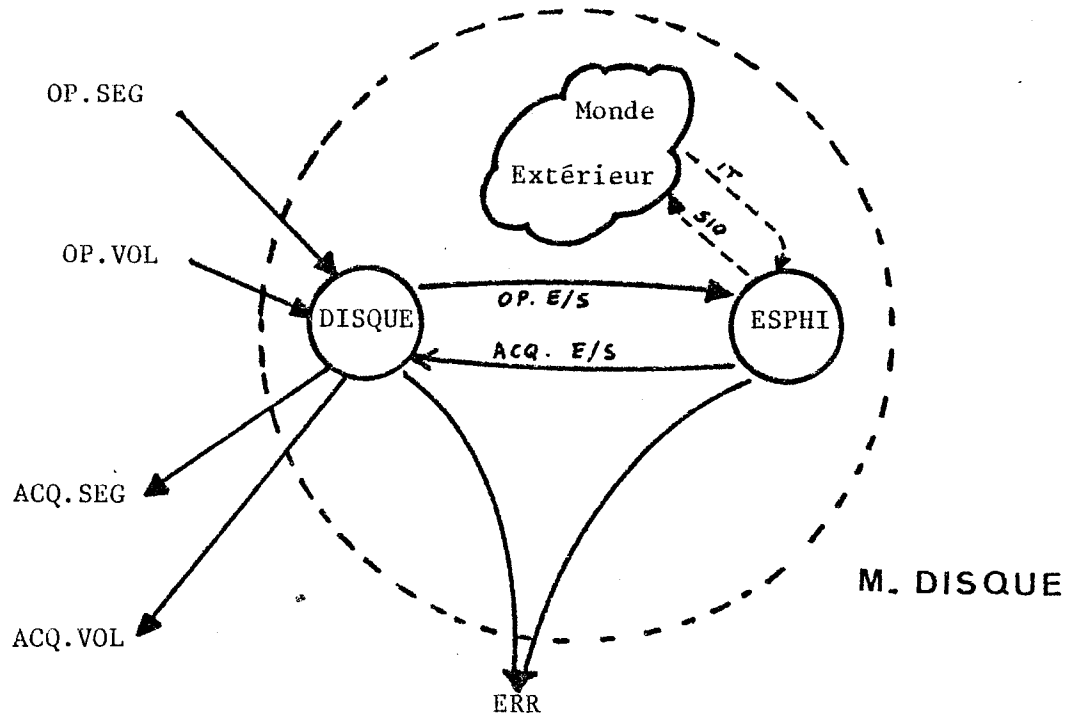
En effet, un processus MAS est un processus IRIS qui dispose des services du noyau MAS. L'unité de gestion de ressource CONTROL détermine si oui ou non un processus MAS peut consommer de l'unité centrale. Elle s'assure que le processus dispose de cases de mémoire pour s'exécuter. Mais c'est la machine Iris qui distribue l'unité centrale. Un processus MAS est donc décrit par un même bloc de contrôle qu'un processus IRIS (cf. § V.1). La différence est que le message descripteur d'un processus MAS contient un contexte d'exécution (l'identification de l'automate de transition de la machine MAS et l'état courant dans cet automate) alors que celui d'un processus IRIS n'en contient pas. Un processus IRIS ne peut pas sortir de la machine Iris, c'est un processus primitif.

Le modèle de décomposition présenté au Chapitre II fournit plusieurs solutions pour assembler les unités de gestion de ressource et donc construire la machine MAS.
Nous allons présenter deux façons de procéder :

- une composition multi-niveaux : on définit des machines abstraites intermédiaires pour construire la machine MAS
- une composition uni-niveau : la machine MAS est directement décrite en termes d'unités de gestion de ressource associées à la machine Iris.

V.3.1. Composition multi-niveaux

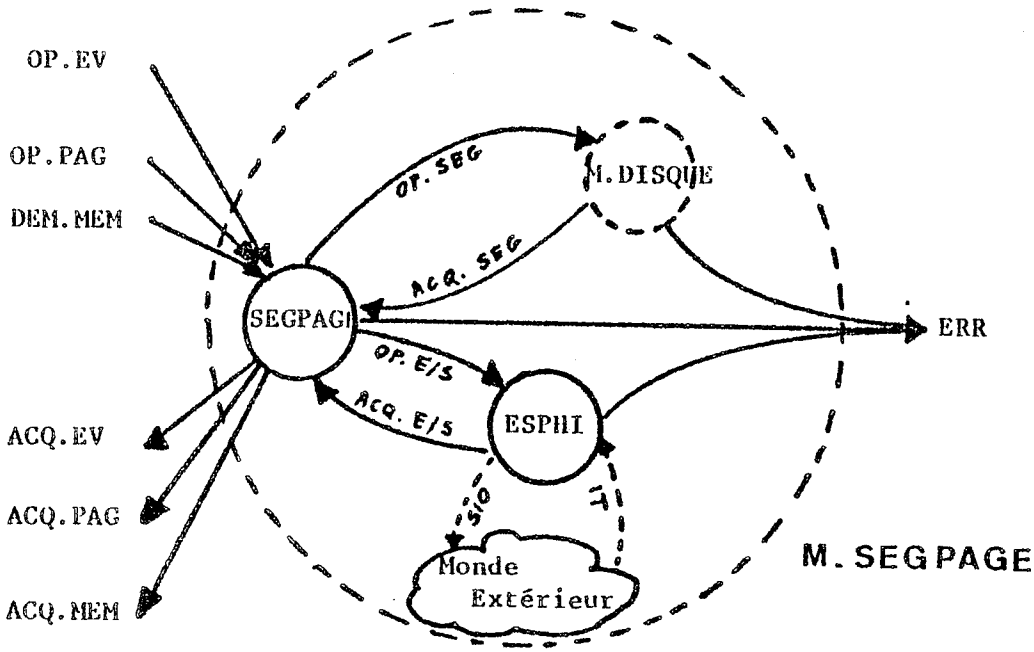
Il s'agit de faire apparaître des machines intermédiaires capables d'interpréter un sous-ensemble du langage de la machine MAS (c'est déjà le cas de la machine Iris). On peut procéder par enrichissement progressif de la machine Iris : on définit une nouvelle machine chaque fois qu'on connecte une nouvelle unité. Une autre façon de procéder et que nous allons présenter consiste à définir une machine pour chaque niveau de service du système. Mis à part l'unité de gestion des Entrées/Sorties physiques, toutes les unités de gestion de ressource utilisent d'autres unités. Pour chacune de ces unités, on définit une machine qui la relie aux unités nécessaires à sa réalisation. Ainsi la machine M-DISQUE relie l'unité ESPHI à l'unité DISQUE :



Chaque unité de gestion de ressource présente une même patte de sortie (SO) en cas d'erreurs de fonctionnement. Ces pattes de sortie d'unités sont aussi des pattes de sortie des machines, mais se trouvent confondues à l'extérieur des machines.

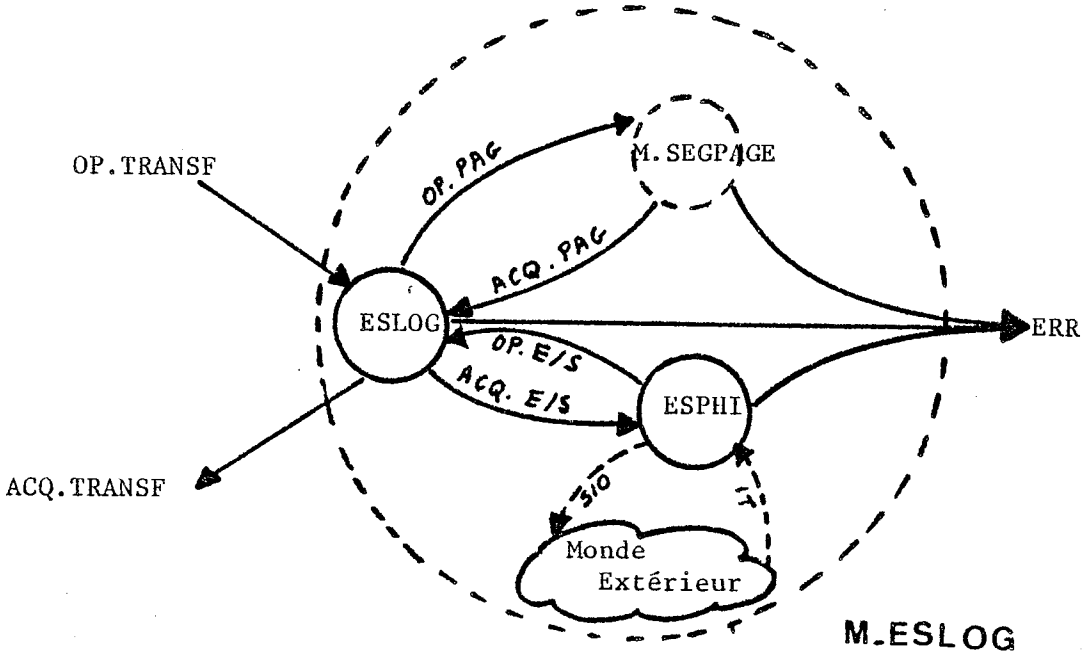
Notons que dans la machine M-DISQUE la patte d'entrée OP-ALLOC et la patte de sortie ACQ-ALLOC de l'unité ESPHI ne sont pas utilisées. Si dans la machine M-DISQUE, l'unité ESPHI renvoie un message sur la patte de sortie S2 = ACQ.ALLOC, le mécanisme de communication considère qu'il y a erreur de fonctionnement. C'est un moyen de contrôler les résultats licites d'un traitement de message dans une unité.

La machine M-DISQUE réalise les opérations sur les segments et l'unité ESPHI les opérations d'Entrées/Sorties nécessaires à la pagination. La machine M-SEGPAGE les relie à l'unité SEGPAGE pour réaliser les opérations de segmentation et de pagination :

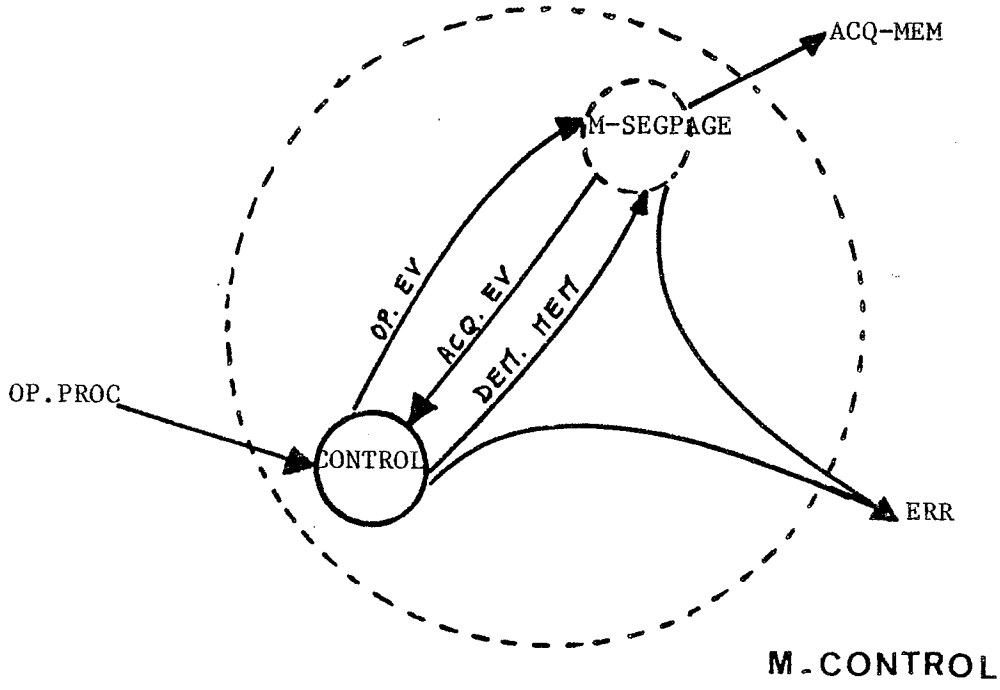


A partir de la machine M-SEGPAGE, on construit les deux machines M-ESLOG et M-CONTROL qui réalisent le protocole appareil standard et contrôlent les processus MAS.

On peut dire que la machine M-SEGPAGE factorise la partie commune de la description des deux machines M-ESLOG et M-CONTROL.

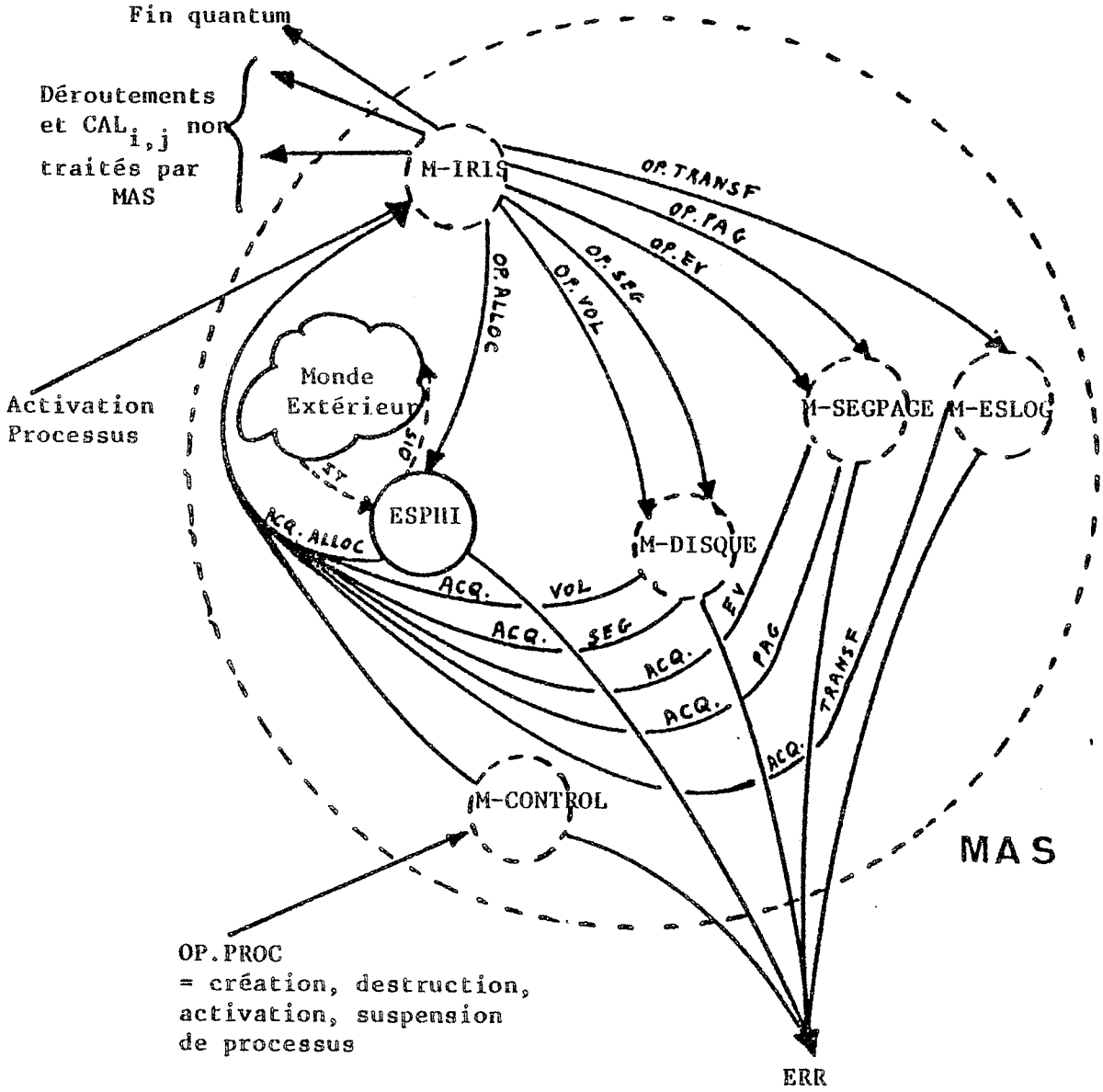


M.ESLOG



M.CONTROL

On obtient finalement la description suivante de la machine MAS :



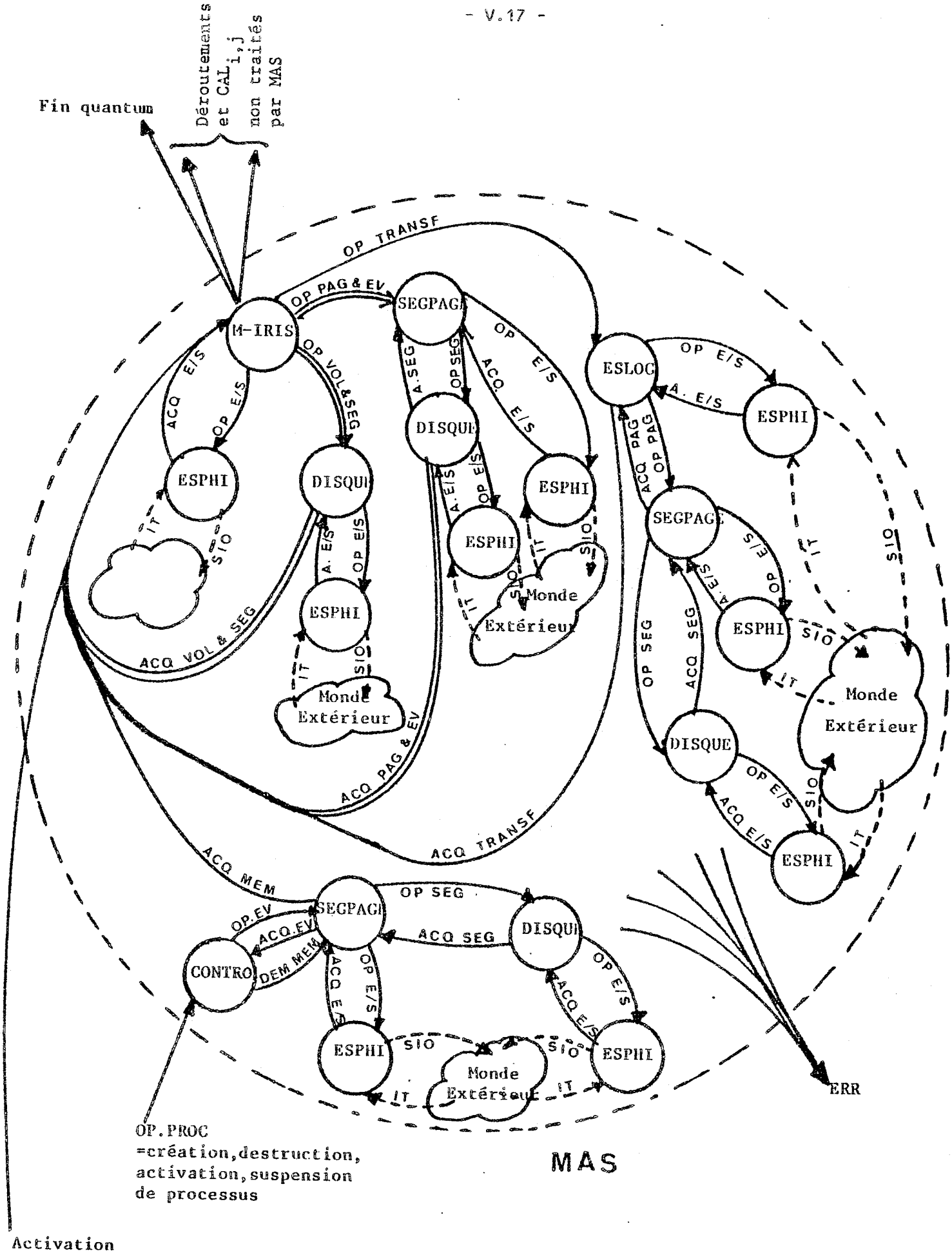
Les messages descripteurs des processus MAS peuvent entrer dans la machine MAS de deux façons différentes : soit par la patte d'entrée "OP-PROC" de la machine M-CONTROL, soit par la patte d'entrée de la machine M-IRIS. Initialement, c'est la machine M-CONTROL qui reçoit les processus MAS (cf. § V.5). Par la suite, l'utilisation de l'une ou l'autre patte d'entrée dépend de la cause pour laquelle le descripteur du processus MAS est sorti de la machine MAS. S'il s'agit d'un déroutement, l'allocation des ressources n'est pas remise en cause, le descripteur rentre alors directeme sur la machine M-IRIS. Dans tous les autres cas, le descripteur du processus rentre sur la machine M-CONTROL.

Cette méthode de composition multi-niveau est coûteuse en taille de descripteur de processus et en temps d'exécution puisqu'à chaque entrée dans une machine incluse, on doit empiler dans le descripteur du processus l'état courant de l'automate de transition de la machine contenant et prendre en compte l'automate de transition de la machine incluse.

V.3.2. Composition uni-niveau

Nous avons choisi de définir la machine MAS directement en termes d'unités de gestion de ressource. Cette définition paraît plus complexe puisqu'elle ne comporte aucune factorisation. On retrouve dans cette description de la machine MAS exactement la décomposition des machines M-DISQUE, M-SEGPAGE, M-ESLOG et M-CONTROL.

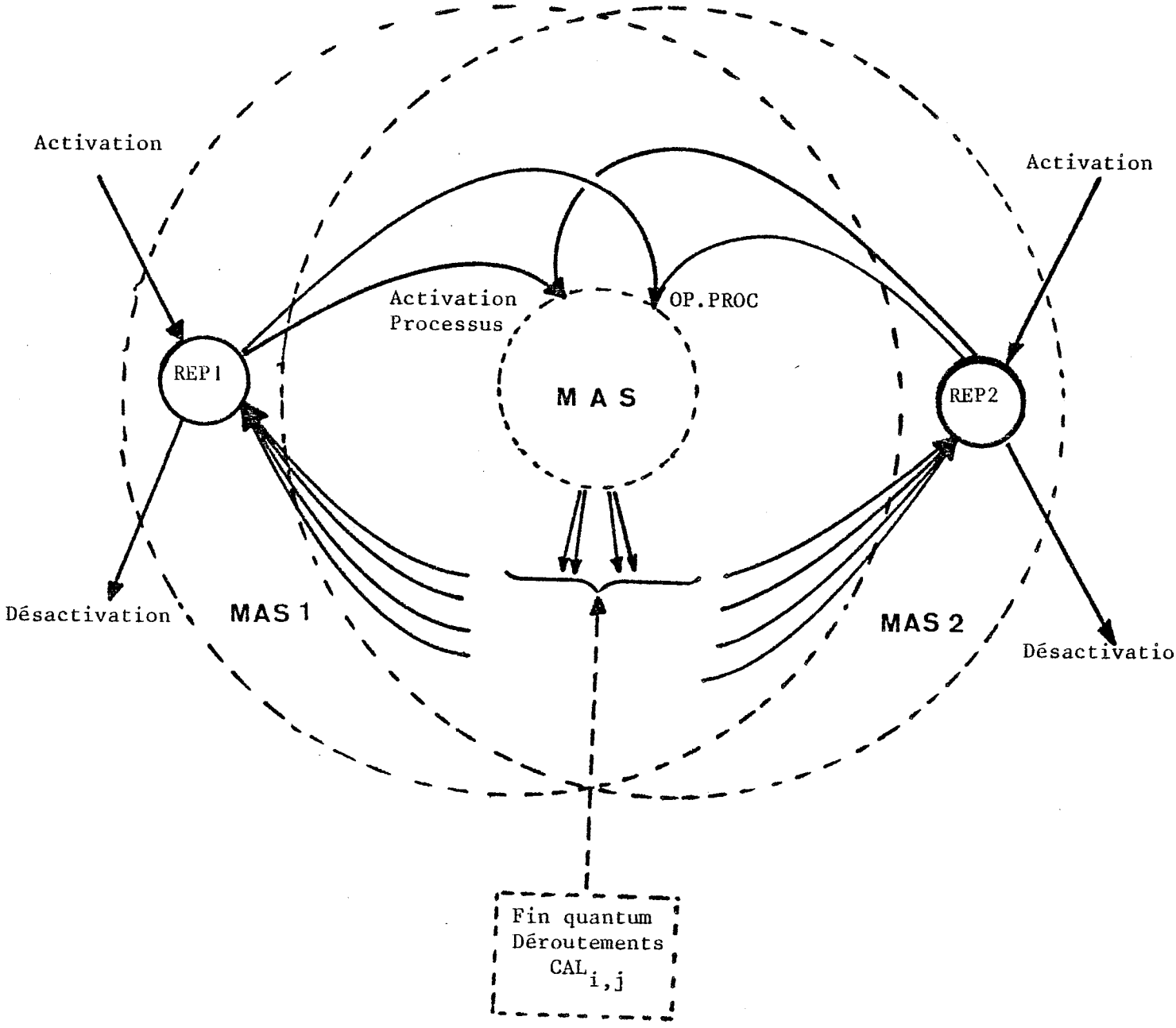
Une unité peut ainsi figurer plusieurs fois dans l'automate de transition d'état de la machine MAS. Chaque occurrence de l'unité sur la figure représente un état distinct de l'automate donc une interprétation différente des conditions de sortie de la même unité.


Fin quantum
Déroutements et CAL i,j non traités par MAS
OP TRANSF
OP PAG & EV
M-IRIS
SEGPAG
OP VOL&SEG
ACQ E/S
OP E/S
A.SEG
OP SEG
ACQ E/S
ESPHI
DISQUE
ESPHI
ESPHI
ESLOC
OP E/S
A. E/S
ESPHI
ACQ PAG
OP PAG
SEGPAG
IT
SIO
A. E/S
OP E/S
ESPHI
ESPHI
IT
SIO
Monde Extérieur
ACQ VOL & SEG
ESPHI
IT
SIO
Monde Extérieur
ACQ PAG & EV
OP SEG
ACQ SEG
ESPHI
IT
SIO
Monde Extérieur
DISQUE
OP E/S
ACQ E/S
ESPHI
ACQ TRANSF
ACQ MEM
OP SEG
SEGPAG
DISQUE
OP.EV
ACQ.EV
ACQ SEG
CONTRO
DEM MEM
ACQ E/S
OP E/S
ACQ E/S
OP E/S
ESPHI
SIO
SIO
ESPHI
Monde Extérieur
IT
IT
ERR
OP.PROC
=création,destruction,
activation,suspension
de processus
MAS
Activation
Processus

V.4. LES MACHINES MAS_i

Nous avons précisé au chapitre précédent (cf. § IV.4) que les processus MAS étaient répartis en classes. Chaque classe (i=1,...,5) dispose d'un pourcentage du temps d'unité centrale et doit se munir d'une unité répartiteur capable de mettre en oeuvre une politique de répartition des ressources entre les différents processus MAS qu'elle gère.

Il importe alors de ne donner aux utilisateurs aucun accès direct à la machine MAS mais d'inclure celle-ci dans une machine MAS_i du type :



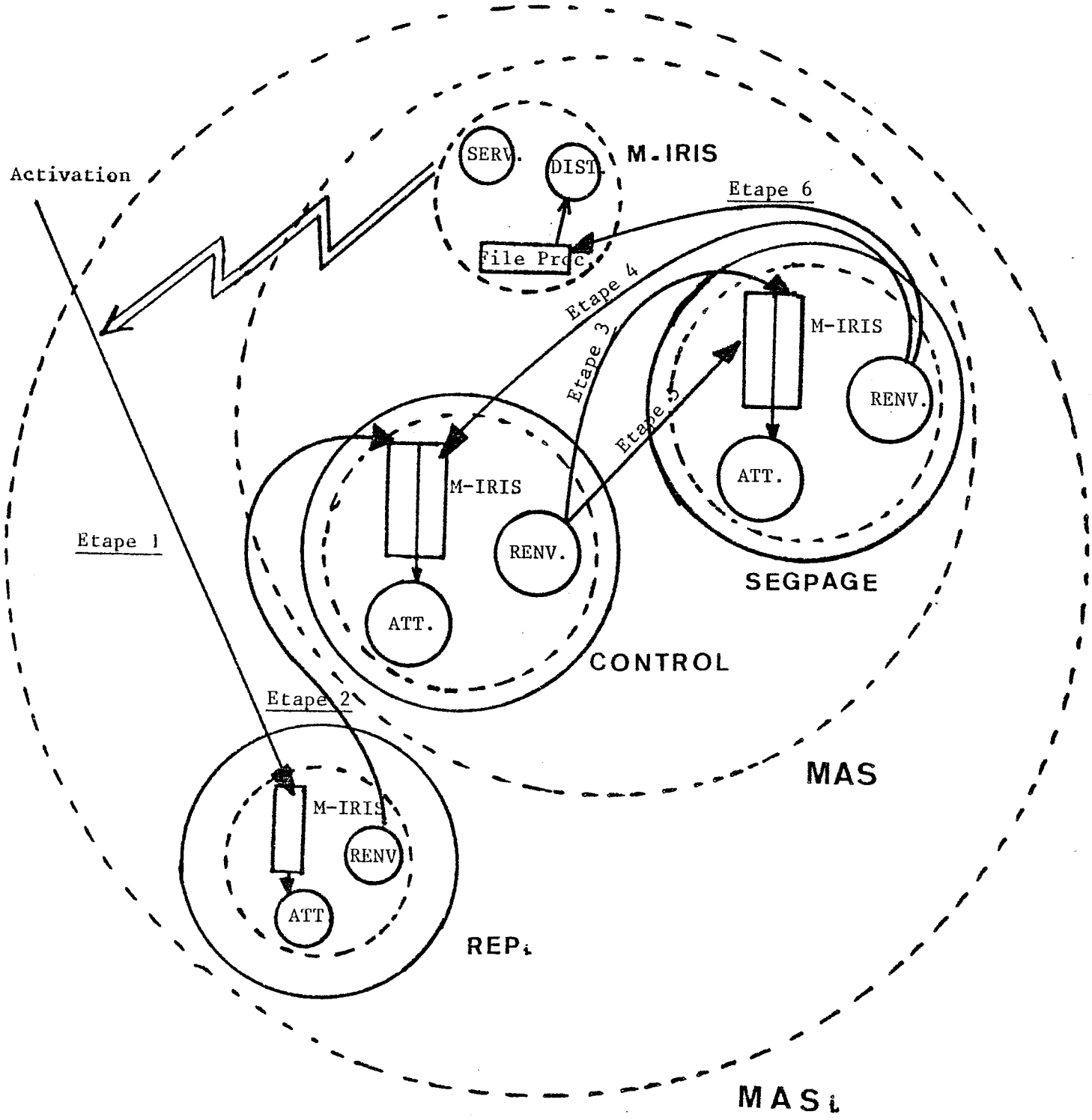
Chaque unité répartiteur REP1, REP2 reçoit les demandes d'activations des processus MAS de leur classe respective et reste libre de décider du moment de leur désactivation. Ainsi un processus de la classe C_1 peut transiter plusieurs fois entre l'unité REP1 et la machine MAS avant d'être désactivé.

V.5. EXEMPLE DE CREATION D'UN PROCESSUS MAS

Supposons qu'un processus MAS détienne le pouvoir de création de processus MAS de la classe i. Ce processus peut donc appeler la primitive de création de processus de la machine Iris (cf. § II.2.31) :

CREERPROCESSUS (id.machine = MAS_i, contexte initial, id.processuscréé).

Le descripteur de ce processus est alors envoyé sur l'unité SERVICES de la machine Iris. Le traitement de cette primitive consiste à engendrer un nom et un message descripteur du processus fils (initialisés avec le contexte initial). Reconnaissant l'identité de la machine MAS_i, l'unité SERVICES adjoint au message l'identification de l'automate de transition de cette machine et l'envoi à son unité initiale : l'unité Répartiteur Rep_i. Le message créé est alors déposé dans la file d'entrée de messages de cette unité (Etape 1 de la Figure). Le processus est considéré comme créé. Le descripteur du processus père est renvoyé au distributeur muni de l'identité du fils. Quand un processus constitutif de l'unité REP_i demande un message à traiter, il exécute un ATTEND. L'unité ATTEND de la machine Iris (cf. § V.1) qui interprète le processus de l'unité REP_i consulte la file d'entrée de messages de l'unité REP_i et prend le message descripteur du processus fils. Tous les traitements et modifications du message prévus sont exécutés : par exemple, découpage du quantum fourni en quantum plus petits. Puis le message descripteur du processus fils est renvoyé sur la patte de sortie "OP-PROC". L'unité RENVOI de la machine Iris qui interprète le processus de l'unité REP_i s'aperçoit que ce message renvoyé doit être déposé dans une certaine file d'entrée de l'unité initiale CONTROL de la machine MAS (Etape 2 de la Figure).


Activation
SERV.
DIST.
M-IRIS
File Proc
Etape 6
Etape 4
Etape 3
Etape 5
M-IRIS
RENV.
ATT.
SEGPAGE
M-IRIS
RENV.
ATT.
CONTROL
Etape 1
Etape 2
M-IRIS
RENV
ATT
REPi
MAS
MASi

De la même façon que pour l'unité REP_i, un processus de l'unité CONTROL acquiert le message en appelant la primitive ATTEND de la machine Iris. Le processus MAS créé et identifié par le message descripteur ne possédant pas encore d'espace virtuel, est renvoyé sur la patte de sortie "OP-EV" de l'unité CONTROL (cf. § V.2.5).

L'unité RENVOI de la machine Iris qui interprète le processus de l'unité CONTROL en conséquence dépose le message descripteur dans une file d'entrée de l'unité SEGPAGE (Etape 3 de la Figure). Quand l'espace virtuel du processus fils MAS est créé, le message correspondant est retransmis à l'unité CONTROL (Etape 4) qui va à un moment donné décider de l'activer.

Le message est alors transmis à l'unité SEGPAGE par la patte de sortie "DEM-MEM" de l'unité CONTROL (cf. § V.2.5). Il est donc déposé dans une autre file d'entrée de l'unité SEGPAGE (Etape 5 de la Figure). Quand l'unité SEGPAGE a redistribué la mémoire pour permettre l'exécution de ce nouveau processus MAS, elle renvoie le message descripteur du processus sur sa patte de sortie "ACQ-MEM" (cf. § V.2.3).

L'unité RENVOI de la machine Iris qui interprète le processus de l'unité SEGPAGE s'aperçoit que le message est destiné à la machine Iris incluse dans la machine MAS. Le message descripteur du processus MAS_i est alors déposé dans la $i^{ème}$ file des processus activés sur la machine Iris (Etape 6 de la Figure).

V.6. RESUME

La décomposition d'un système classique comme MAS en unités et en machines permet d'isoler les grandes fonctions du système (gestion de périphériques, gestion de mémoire, contrôleur). La recherche de cette décomposition nous a permis de constater qu'un système informatique classique est peu répartissable.

En effet un découpage plus fin du système MAS n'est pas applicable sans un accroissement disproportionné du flot de messages échangés et une baisse de performance du système.

Si l'on veut que le gain de performance donné par la multiplicité des processeurs ne soit pas anéanti par l'inertie des outils de communication entre ces processeurs, il ne faut pas découper trop finement les applications.

CHAPITRE VI

CONCLUSION

VI.1. BILAN

Il est toujours difficile de faire le bilan d'une expérience qui n'a pas été menée à son terme.

Si l'on considère le déroulement chronologique de notre réalisation, on s'aperçoit qu'il y a eu un certain conflit entre deux objectifs : il s'agissait d'une part de réaliser un système d'exploitation sur un gros ordinateur et d'autre part d'étudier des outils de réalisation de systèmes. Le premier objectif n'a finalement pas été atteint mais les contraintes qu'il a imposées pendant le développement ont nui au deuxième objectif.

VI.1.1. Ce qui a été réalisé

Le noyau HARDEXT (voir Chapitre II) contenant les mécanismes de construction d'unités et de machines, les mécanismes de connexion par ATTEND et RENVOI et tous les services de base (gestion de la mémoire libre résidente, acquittement des interruptions, gestion des échéanciers, répartiteur, ...) a été programmé et testé.

Les modules de la machine MAS (voir Chapitre IV) réalisant la gestion de la mémoire segmentée et les entrées/sorties sur les périphériques ont été programmés mais non testés.

Divers sous-systèmes :machine relationnelle [FOR], station de transport [SEF], machine fichier (cf. Chapitre III) ont été spécifiés sur la machine MAS, en utilisant les outils de décomposition mis en oeuvre dans HARDEXT.

Il s'agit là de ce qui a été matériellement réalisé. Mais nos travaux apportent en outre une expérience scientifique intéressante dans le domaine de la construction de systèmes informatiques au moyen d'outils universels. Nous reviendrons sur ce point au paragraphe VI.3.

VI.1.2. Ce qui n'a pas pu être réalisé : évaluation quantitative

Les solutions apportées aux problèmes ponctuels d'efficacité (enveloppes de segments, tampon de pagination, algorithmes d'allocation sur disque) n'ont pu être quantitativement évaluées. Nous n'avons donc pas pu apporter la preuve que les paramètres de réglage que nous avions choisis étaient

les bons, ni a fortiori les valeurs que nous leur avons données pour assurer un fonctionnement satisfaisant du système.

Une évaluation complète avant réalisation était difficilement envisageable à moindre coût : d'une part une évaluation analytique n'est pas possible car on ne sait pas construire de modèle d'un système tel que le nôtre en termes d'équations que l'on sache résoudre. D'autre part une évaluation par simulation doit être nécessairement fine si l'on veut que le modèle soit fidèle à la réalité et les résultats probants ; on s'aperçoit alors qu'une telle simulation peut être aussi coûteuse, sinon plus, que la réalisation du système.

Il était possible de faire des évaluations ponctuelles restreintes d'algorithmes par simulation. Mais cela nous aurait demandé un travail supplémentaire relativement important qui aurait été soustrait à la réalisation du système d'exploitation que nous envisagions. Nous avons jugé préférable de développer des algorithmes conçus en nous inspirant des travaux connus : c'est le cas des algorithmes de gestion de la mémoire segmentée, ainsi que des algorithmes de gestion des transferts sur disques et tambours. Les fonctions étant bien délimitées, il était possible de régler les paramètres dans la version expérimentale du système.

Nous avons envisagé, lorsque la réalisation d'un système d'exploitation a été abandonnée, de faire une évaluation du découpage de la machine MAS, dans la version expérimentale qui en était réalisée, au moyen d'un simulateur de charge travaillant à partir de traces de programmes prélevées sous SIRIS8. Nous étions conscients du fait que les résultats auraient été biaisés , le comportement d'un programme sous SIRIS8 ne pouvant être supposé identique à celui du programme équivalent s'exécutant sur MAS. En fin de compte, la réalisation de ce simulateur de charge a été abandonnée en même temps que MAS.

Enfin, une évaluation macroscopique du découpage de MAS, en considérant les unités comme des boîtes noires, était dénuée de sens puisqu'on était dans l'incapacité d'affecter des coefficients de probabilité aux transitions entre les unités de MAS. En l'état actuel de la technique, on ne peut estimer de tels coefficients que par des mesures sur un système expérimental fonctionnant sous un simulateur de charge (mais alors il est difficile d'obtenir des mesures statistiquement significatives), ou en exploitation réelle.

VI.1.3. Quelques réflexions sur le découpage d'un système

Les méthodes de décomposition fonctionnelle d'un système reposent sur la notion de découpe en niveaux d'abstraction [DIJ], ainsi que de structuration des programmes en modules. Ces deux notions ne coïncident pas nécessairement.

Il y a, à l'heure actuelle, une certaine ambiguïté derrière tout principe de décomposition modulaire. Non par manque de critères objectifs de décomposition (l'efficacité, par exemple), mais par insuffisance des outils d'évaluation qui permettraient de quantifier les avantages respectifs de plusieurs décompositions possibles en fonction des critères retenus.

Il n'existe pas a fortiori d'algorithme de décomposition. De nombreux degrés de décomposition d'un système sont possibles. Nous aurions pu, par exemple, déconnecter la segmentation de la pagination dans MAS et donc couper le module SEGPAGE en deux ; ou bien, comme dans GEMAU [BRI1], faire éclater le module ESPHYSIQ en une dizaine de modules spécialisés sur des types particuliers d'entrées/sorties.

Le découpage en modules que nous avons adopté est logique à plus d'un titre et nous n'allons pas revenir sur sa justification. Mais il n'est pas nécessairement unique.

On peut se demander, de manière analogue, quels sont les critères permettant de juger si une certaine décomposition en niveaux d'abstraction est satisfaisante.

Faut-il multiplier les niveaux au risque de perdre de l'efficacité si cela se traduit par des cascades d'interprétations successives ? (Notons toutefois que l'on peut court-circuiter des niveaux d'interprétation : une transition d'un automate de niveau i peut entraîner directement une transition d'une machine de niveau j, avec j << i).

Ou bien faut-il programmer "à plat", avec efficacité mais laborieusement et au prix de réajustements fréquents dont on sait qu'ils sont une source de pannes et de mauvais fonctionnement temporaire ?

En l'absence d'outils d'évaluation, la discussion n'est pas close.

Pour nous, le problème du choix des fonctions à inclure dans le noyau MAS s'est posé dans les termes suivants :

(1) Le noyau doit être suffisamment restreint pour être universel, mais alors il faut prévoir des extensions pour programmer les applications systèmes, ou

(2) Il faut réaliser une gestion perfectionnée des objets qui pourrait se révéler inutile pour certaines applications.

Le deuxième choix était celui de GEMAU [BRI 1], qui s'était révélé être un outil assez lourd et peu éfficace. Nous avons donc opté pour la première solution, celle d'un noyau simple pourvu de mécanismes d'extension assez puissants. Nous offrons moins que MULTICS ou HYDRA sur le plan des ressources logiques mais nous fournissons plus sur le plan des mécanismes d'extension.

VI.1.4. Ce qu'il serait souhaitable d'améliorer

VI.1.41. Langage de connexion

Au vu des premières expériences, il y a une amélioration qu'il serait souhaitable d'apporter pour l'utilisation du modèle de décomposition. Elle concerne le langage de connexion permettant de définir les machines et les automates régissant les transitions d'états à l'intérieur de ces machines. Il semble que les primitives CREERUNITE et CREERMACHINE que nous fournissons soient d'un emploi malaisé, car de trop bas niveau et n'offrant aucune protection.

Il serait souhaitable que le langage de connexion puisse décrire le cheminement du traitement d'une primitive de la machine à construire entre les différents composants de la machine, ou en d'autres termes qu'il puisse décrire la suite des transitions entre les états de l'automate à construire à partir d'une transition initiale (la soumission de la primitive).

L'interpréteur ou le compilateur d'un tel langage de connexion devrait être capable de construire l'automate de la machine à partir des descriptions des chemins parcourus.

Ceci débouche en fait dans le domaine plus vaste de l'expression, dans un langage évolué, du "contrôle" dans un système, le terme de "contrôle" désignant ce qui a trait à la commande des différentes parties d'un traitement dans une machine et à l'émission, réception ou échange de signaux entre les différents composants d'une machine.

Ces problèmes débordent du cadre de cette étude. Par contre, il serait envisageable de fournir, dans le langage de connexion, des moyens pour contrôler le type des données échangées entre les composants d'une machine (voir par exemple GREEN [ABR2] ou SESAME [CHE]). Il est alors possible de vérifier statiquement au moment du CREERMACHINE la validité des connexions du point de vue du type des messages qui les emprunteront.

VI.1.42. Contrôle de flux, régulation

Tel que nous l'avons réalisé, le schéma de communication se prête au contrôle de flux entre les unités. Cela peut se faire de plusieurs manières différentes :

- Si l'on veut contrôler le flux de messages entre deux composants d'une machine, on peut facilement intercaler une unité de contrôle entre les deux composants. Il suffit pour cela de modifier les transitions appropriées de l'automate.

- On peut aussi faire jouer à HARDEXT un rôle de régulateur global pour toute une machine ou local pour une unité, en lui faisant contrôler la longueur des files d'attente à l'entrée de certaines unités. On a ainsi un contrôle centralisé [JAM] de l'activité.

- Enfin, les répartiteurs disposent de l'instruction SUSPENDRE (traitée par HARDEXT) qui leur permet de suspendre l'activité de processus utilisateurs et donc l'émission de messages. Les processus suspendus sont retirés de la liste des processus activables. Les répartiteurs sont ainsi capables de gouverner dans une certaine mesure les files d'attente dans les niveaux de machine au-dessus d'eux.

VI.2. PERSPECTIVES

Les perspectives portent en gros sur trois champs d'exploration :

- les problèmes généraux de distribution de ressources,
- les problèmes d'utilisation de grandes mémoires segmentées pour des bases de données,
- les problèmes d'évaluation d'architectures multiprocesseurs.

VI.2.1. Distribution de ressources

Considérons le premier point sous l'angle suivant :
Soit une machine M contenant des unités U_i chargées de gérer des ressources données, matérielles ou logicielles. Les unités U_i peuvent apparaître elles-mêmes comme des ressources logicielles fournies aux processus exécutés par la machine M.

Une demande de ressource se traduit dans tous les cas par l'envoi d'un message à une unité de la machine M. Si l'unité gère une ressource, l'allocation se traduit par le renvoi du message ; la libération ultérieure doit également être demandée par un message, renvoyé ensuite par cette unité. Si l'unité est elle-même considérée comme une ressource, l'allocation se traduit par la prise en compte du message et la libération par le renvoi du message.

Il est facile de filtrer tous ces messages. Il suffit de prévoir les transitions appropriées dans l'automate associé à la machine M.

Il est alors possible de mettre au point des stratégies fines de répartition des ressources fournies par une certaine machine et de les évaluer en plaçant des points de mesure dans le filtre à messages.

De plus, des ressources d'une machine M pouvant être mises en oeuvre par une machine incluse M', et ainsi de suite en cascade, on a un schéma d'asservissement des ressources les unes par rapport aux autres. L'évaluation des contraintes d'asservissement est un domaine quasiment inexploré dont l'étude serait facilitée par l'usage de nos outils [ANC2].

Le contrôle des transitions d'état dans l'automate d'une machine permettrait aussi la mise au point de politiques de régulation adaptative à un niveau plus fin que celle que nous avons réalisée, puisque nous nous contentons, au vu de la valeur du taux de pagination, d'agir sur le degré de multiprogrammation de l'IRIS 80 ; ce qui est un peu élémentaire ....

VI.2.2. Utilisation de la mémoire segmentée

C'est surtout du côté des systèmes d'implantation et de recherche de données dans de grands ensembles que se situent les perspectives les plus intéressantes. Les recherches par arbres, les algorithmes d'exploration relationnelle peuvent profiter du système de segmentation fourni par MAS

car celui-ci fait disparaître les problèmes de gestion physique de l'accès aux données. Il serait intéressant de voir un système de gestion de base de données implanté sur un noyau tel que MAS [FOR].

L'expérience de MULTICS semble indiquer qu'un noyau fournissant de grands espaces virtuels constitue une bonne base pour la création de machines spécialisées dans la gestion de fichiers ou de bases de données [COR].

VI.2.3. Evaluation d'architectures multiprocesseurs

Les outils de décomposition que nous fournissons peuvent permettre l'évaluation d'architectures multiprocesseurs. On peut en première approche identifier un composant (unité ou machine) avec un processeur (physique ou logique). Il est alors possible d'utiliser les outils d'architecture évolutive que nous fournissons dans les essais et l'évaluation d'architectures de systèmes sur multiprocesseurs : insertion, remplacement, suppression d'unités ou groupes d'unités.

Il y a plus : deux notions de communication coexistent dans notre système. Il y a d'une part la communication par messages entre processeurs (unités ou machines) gérée par automate. Mais il y a aussi des processeurs ayant accès à une mémoire partagée et communiquant au moyen de variables communes : ce sont les unités regroupées en modules.

Il est donc possible d'évaluer des architectures multiprocesseur en tenant compte des divers modes de communication physique entre processeurs : communication par mémoire commune (unités groupées en module) ou communication par bus ou ligne (réseau d'unités ou machines) sans mémoire commune.

On peut naturellement regrouper les deux types de communication pour évaluer des architectures de grappes de processeurs disposant de mémoire commune avec communications par bus ou ligne entre les grappes : CM* [FUL], machines réseau [CHUP].

Il faut toutefois apporter une restriction : les transmissions physiques de messages se font dans notre cas par copie de mémoire entre espaces virtuels et non par fil. On ne peut donc pas tester directement le comportement temporel d'une architecture multiprocesseur soumise à une certaine charge.

On peut par exemple procéder comme suit :

. On commence par écrire une version simplifiée du système en prenant une unité pour un processeur, et en regroupant éventuellement en module les unités à mémoire commune.

. On observe et on mesure (indépendamment du temps) le comportement de l'architecture sur des jeux d'essai. On commence par observer le déroulement dans le réseau d'une commande initiale - on vérifie à ce moment la validité des transitions - on fait de même pour une certaine charge globale du système. On est alors en mesure d'assigner des coefficients de probabilité aux transitions entre les unités. On peut alors fabriquer l'automate probabiliste du réseau de processeurs.

. Cet automate est ensuite utilisé en entrée d'un simulateur de files d'attente [MER] dans lequel les durées correspondant aux divers modes de transmission, ainsi que les temps de service à l'intérieur des unités sont également introduits sous forme de paramètres ajustables. On peut alors analyser les performances du système, compte tenu des lois d'arrivées et des paramètres de l'architecture (temps de service des unités, temps de transmissions).

Une telle approche fournit un procédé puissant de définition de systèmes répartis sur des architectures multiprocesseurs. On pourrait par exemple, en faisant varier les paramètres d'architecture, définir les moyens de communication les plus appropriés pour un certain type d'architecture : modularité de certaines unités regroupées sur mémoires communes, transmissions par bus entre certaines unités ou modules, transmissions par lignes synchrones/asynchrones entre d'autres unités ou modules. On pourrait ainsi déterminer la meilleure manière de cabler un modèle que l'on aurait évalué par la méthode indiquée.

Il faut remarquer qu'il n'existe pas encore à notre connaissance d'outils de simulation et d'évaluation d'architectures de systèmes qui soient décomposés en deux parties comme nous venons de l'indiquer, avec une approche synthétique comme la nôtre. Un outil comme SIMULA [DAH] ne sépare pas la simulation temporelle de la mesure du comportement d'une architecture ; il faut tout simuler en une seule fois. L'évaluation d'une architecture est alors plus coûteuse que sa réalisation : programmation complète du système suivie de la mesure de son comportement et du réglage des paramètres.

D'autres outils d'évaluation existent déjà, dont certains sont complémentaires du nôtre [MER]. Des outils d'évaluation d'architectures réparties existent [BOR] mais ils ne sont encore opérationnels que pour l'évaluation de composants proches du matériel.

VI.3. APPORT DU TRAVAIL PRESENTE

Malgré les avatars rencontrés au cours de la réalisation d'un système d'exploitation - pour des raisons extérieures à cette réalisation - notre travail apporte des éléments nouveaux dans le domaine de l'architecture des systèmes informatiques.

Les méthodes de découpage des systèmes informatiques en composants, et leur construction à partir des composants au moyen de mécanismes d'extension [DIJ], sont appliquées d'ordinaire au découpage en niveaux d'abstraction fondés sur les notions de procédure et d'appel procédural. Notre modèle de décomposition s'appuie à l'inverse sur les notions de processus et de communication par messages.

Ce modèle de décomposition a été appliqué de manière systématique. Il comprend des opérations de construction d'unités et de machines, et un mécanisme simple de connexion de ces unités et machines : l'automate de transition des machines construites. Ces opérations et ce mécanisme permettent l'extensibilité des systèmes.

Nous avons illustré l'utilisation du modèle de décomposition, et nous espérons avoir montré son utilité, en l'appliquant à la construction de la machine MAS et d'une machine à fichiers.

Cette démarche nous a conduits à nous interroger sur la valeur générale du découpage et des outils de décomposition proposés. Les problèmes rencontrés peuvent préfigurer ceux qui se posent dans la réalisation des systèmes répartis. Une condition première pour l'utilisation de notre modèle dans la réalisation d'un système implanté sur un multiprocesseur serait d'adapter la notion d'automate à une architecture répartie, c'est-à-dire de pouvoir faire éclater un automate sur les sites physiques du multiprocesseur. D'où la nécessité d'une méthode et d'un langage d'expression du contrôle pour la réalisation d'applications réparties.

Bibliographie

[ABR1] J.R. ABRIAL & al.
Projet SOCRATE - Spécifications générales.
U.S.M.G., Août 1970.

[ABR2] J.R. ABRIAL
Présentation du parallélisme dans un langage de haut niveau.
Extrait du rapport : Rationale for the design of the GREEN programming language. Février 1978.

[ACH] M.S. ACHARD, J.Y. BABONNEAU, G. MORISSET
Adaptation automatique des programmes au milieu paginé.
Rapport LABORIA n° 196, 1976.

[ADA] J.C. ADAMS, G.E. MILLARD
Performance measurements on the Edinburgh Multiaccess System.
ICS 75 Antibes, Juin 1975.

[ANC1] F. ANCEAU, N. HADJIDAKIS
Organisation logique d'un mécanisme de communication pour un système transactionnel réparti de manière fonctionnelle.
Rapport de Recherche n° 96, I.N.P.G., Novembre 1977.

[ANC2] F. ANCEAU
Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques.
Thèse d'Etat, ENSIMAG, Décembre 1974.

[AUR] A. AUROUX, C. HANS
Introduction au système CP/67 - CMS.
Monographies Informatiques. L. Bolliet, Dunod.

[BAG] J.D. BAGLEY & al.
Sharing data and services in a virtual machine system.
IBM Journal.

[BAL] R. BALTER, J. BRIAT, X. ROUSSET DE PINA
Révocation des droits d'accès et substitution des objets dans les systèmes à adressage par descripteurs.
Séminaire ENSIMAG, Janvier 1975.

[BAN] J.P. BANATRE & al.
Designing an application - oriented distributed system.
Note interne IRISA PI/078, Rennes, 1977.

[BE] J. BELLINO
Mécanismes de base dans les systèmes superviseurs.
Thèse ès-sciences appliquées, USMG, Septembre 1973.

[BEK] Y. BEKKERS, D. HERMAN, M. RAYNAL
Conception et réalisation d'une machine-langage de haut niveau adaptée à l'écriture des systèmes.
Thèse 3ème Cycle, Rennes, Septembre 1975.

[BEL1] L.A. BELADY, C.J. KUEHNER
Dynamic space sharing in computer systems.
CACM, vol. 12, n° 5, 1969.

[BEL2] L.A. BELADY
A study of replacement algorithms for virtual storage computer.
IBM System Journal, Vol.5 n° 2, 1966.

[BIS] R. BISIANI, F. TISATO
A multi-microprocessor system as a set of cooperating processes.
Proc EUROMICRO Workshop, Nice, 1975.

[BOR] D. BORRIONE
LASCAR - Un langage pour la simulation et l'évaluation des architectures d'ordinateurs.
Thèse 3ème Cycle INPG, Avril 1976.

[BRIN] P. BRINCH HANSEN
Distributed processes : a concurent programming concept.
Univ. of Southern California, Septembre 1977.

[BRI1] J. BRIAT, S. GUIBOUD-RIBAUD
Espace d'adressage et espace d'exécution du système GEMAU.
Aspects théoriques et pratiques des systèmes d'exploitation, IRIA, 1974.

[BRI2] J. BRIAT, S. ROUVEYROL, A. TARABOUT
Evaluation du langage LIS.
OURS 9005.2, Octobre 1976.

[BRI3] J. BRIAT
Manuel de présentation MAS-HARDEXT.
OURS 1001.

[BRI4] J. BRIAT, J. ESTUBLIER
Manuel langage MAS-HARDEXT.
OURS 3000, Mars 1977.

[BRI5] J. BRIAT, J.P. VERJUS
Implication de certaines propriétés d'un noyau de système (MAS) sur son langage d'écriture. BIGRE, n° 4, Octobre 1976.

[BRI6] J. BRIAT, X. ROUSSET DE PINA, J.P. VERJUS
Le projet OURS : ses principes.
Congrès AFCET, Juin 1976.

[BUL] P. BURGEVIN, J. LEROUDIER
Characteristics and models of program behavior.
ACM nat. conf., 1976.

[BUR] W.H. BURGE, A.G. KONHEIM
An accessing model.
JACM vol.18 n°3, Juillet 1971.

[BUZ] J.P. BUZEN
Fundamental base of computer system performance.
ACM Sigmetrics, Harvard 1976.

[CHE] J.L. CHEVAL & al.
Conception modulaire de systèmes d'exploitation.
Rapport de recherche n° 32, ENSIMAG, Mars 1976.

[CHU] W.W. CHU, H. OPDERBECK
Performance of the Page Fault Frequency replacement algorithm in a multiprogramming environment.
Information Processing 74 - North-Holland Publishing Company, 1974.

[CHUP] J.C. CHUPIN, J. SEGUIN
MADRE - Objectifs de définition d'une méthode d'accès direct réseau.
Centre Scientifique CII Grenoble, Avril 1974.

[CII1] CII - Ordinateur IRIS 80
Manuel d'utilisation.
N° 4152 E/Fr.

[CII2] CII - Procédures systèmes sous SIRIS7 / SIRIS8
Manuel d'utilisation.
N° 3642 E4/Fr.

[COF1] E.G. COFFMAN
Analysis of drum input/output queue under scheduled operation in a paged computer system.
JACM vol.16 n°1.

[COF2] E.G. COFFMAN, L.A. KLIMKO, RYAN
Analysis of scanning policies for reducing disk seek times.
SIAM Journal on Computing.

[COR] F.J. CORBATÓ, J.H. SALTZER, C.T. CLINGEN
MULTICS - The first seven years.
Spring joint computer conference, 1972.

[CRE] B. CREECH
Architecture of the B6.500.
Infotech report 2, 1971.

[CRO] CROCUS
Systèmes d'exploitation des ordinateurs.
Dunod.

[DAH] O. DAHL, K. NYGAARD
SIMULA : an Algol-based simulation language.
CACM vol.9, Septembre 1966.

[DAR] P. DARONDEAU & al.
Architecture de calculateurs orientés vers les systèmes.
RAIRO, B2, 1974.

[DEL] M. DELAUNAY
Rapport d'évaluation du système LIS.
Note interne, Novembre 1977.

[DEN1] P.J. DENNING & al.
Optimal multiprogramming.
Acta Informatica, Vol.17 fasc.2.

[DEN2] P.J. DENNING
Thrashing : its causes and preventions.
Proceedings AFIPS, FJCC 1968.

[DEN3] P.J. DENNING
The Working Set model of program behavior.
CACM Vol.11 n°5, Mai 1968.

[DEN4] P.J. DENNING
Virtual memory.
Computing Survey, Vol.2 n°3, Septembre 1970.

[DIJ] E.W. DIJKSTRA
Hierarchical ordering of sequential processes.
Acta Informatica, Vol.1, 1, 1971.

[DUP] J.P. DUPUY
Améliorations fonctionnelles dans un système d'opérations
à trois niveaux de hiérarchie.
Thèse CNAM, Mars 1974.

[DUPU] M. DUPUY, A. SCRIZZI
A multiprocessor system based on a multibus/multiport oriented
memory communication.
Proc. EUROMICRO Symp., Venise, Octobre 1976.

[ENG] D.M. ENGLAND
Capability concept and structure in System 250.
Protection in Operating Systems, IRIA, 1974.

[EST1] J. ESTUBLIER
Processus cyclique et appel procédural.
Thèse 3ème Cycle INPG, Avril 1978.

[EST2] J. ESTUBLIER, A. TARABOUT, I. VATTON
Manuel langage MAS-UGR.
OURS 3001.2, Octobre 1976.

[FER] J. FERRIE, D. LANCIAUX
Le système Plessey 250.
Rapport de recherche n° 168, IRIA, Avril 1976.

[FIN] L. FINET
Entrées/sorties dans les systèmes d'exploitation.
Thèse 3ème Cycle USMG, 1973.

[FOR] J.M. FORESTIER
Machine relationnelle pour système segmenté.
Thèse 3ème Cycle INPG, Septembre 1978.

[FRA]] H. FRANK
Analysis and optimization of disk storage devices for time-sharing systems.
JACM Vol.16 n°4, Octobre 1969.

[FRAN] N. DE FRANCESCO, G. VAGLINI, M. VANNESCHI
On deadlocks in networks of asynchronous microprocessors.
Proc. EUROMICRO Symp., Venise, Octobre 1976.

[FUL] FULLER, JONES, DURHAM editors
CM* review.
Juin 1977.

[GEM] Projet GEMAU
Spécifications internes.
Note interne, Grenoble, 1975.

[GUI] S. GUIBOUD-RIBAUD
Mécanismes d'adressage et de protection dans les systèmes informatiques.
Thèse d'Etat INPG, Juin 1975.

[HAY] L.S. HAYNES
The architecture of an Algol 60 computer implemented with distributed processors.
Proc. 4th annual symp. on computer architecture, Silver Springs, Mars 1977.

[HOA] C.A.R. HOARE
Communicating sequential processes.
CACM vol.21 n°8, Août 1978.

[IBM1] IBM Corp.
OS/360 Input/Output supervisor program logic manual.
Form GY28-6616.

[IBM2] IBM Corp.
OS/VS virtual storage access method (VSAM).
Programmer's guide.
GC26-3838-2.

[IBM3] IBM Corp.
OS/VS virtual storage access method (VSAM).
Planning guide. GC26-3799-2.

[JAM] A.J. JAMMEL, H.G. STIEGLER
Structural decomposition and distributed systems.
Note interne du Centre de Calcul Leibniz, Munich, Novembre 1977.

[KAI] C. KAISER
Conception et réalisation de systèmes à accès multiple : gestion du parallélisme.
Thèse d'état, IRIA, 1973.

[KER1] A. KERANGUEVEN
Conception et réalisation du noyau du système SAR : adressage des objets.
Thèse 3ème Cycle, Rennes, Juillet 1974.

[KER2] F. KERANGUEVEN
Conception et réalisation du noyau du système SAR : gestion du parallélisme.
Thèse 3ème Cycle, Rennes, Juillet 1974.

[KIL] T. KILBURN & al.
One level storage system.
IRE Trans. EC-11, 2, Avril 1962.

[KRA] S. KRAKOWIAK
Conception et réalisation de systèmes à accès multiple : allocation de ressources.
Thèse d'état, IRIA, 1973.

[LAF1] P. LAFORGUE
Construction de sous-système utilisant une machine abstraite.
Thèse Docteur-Ingénieur, Grenoble, Février 1975.

[LAF2] P. LAFORGUE
La répartition des ressources.
OURS 1005, Novembre 1977.

[LIS] CII
The system implementation language LIS.
N° 4549 E1/En.

[LEN] J. LENFANT
Comportement des programmes dans leur espace d'adresse.
Thèse d'état, Rennes, Décembre 1974.

[LER1] J. LEROUDIER, D. POTIER
Principles of optimality for multiprogramming.
ACM Sigmetrics, Harvard, 1976.

[LER2] J. LEROUDIER, D. POTIER, M. BADEL
Un modèle d'analyse des performances d'ordinateurs multiprogrammés à mémoire virtuelle.
IRIA - LABORIA.

[LER3] J. LEROUDIER
Systèmes adaptatifs à mémoire virtuelle.
Thèse d'état, Grenoble, 1977.

[LER4] J. LEROUDIER, D. POTIER
A two leivel control scheme for multiprogrammed virtual memory computer systems.
Rapport IRIA n° 141, Novembre 1975.

[LEV] R. LEVIN & al.
Policy / mechanism separation in HYDRA.
Proc. 5th symp. on OS principles, Austin, Novembre 1975.

[LIP] G.J. LIPOVSKI
On virtual memories and micro-networks.
Proc. 4th annual symp. on computer architecture, 1977.

[LUC] M. LUCAS, J. MONTUELLE
Conception modulaire des systèmes d'exploitation.
Thèse 3ème Cycle, Grenoble, Juin 1977.

[MAN] T. MANOCHA
Ordered notion for direct access devices.
SIAM 1971 Fall meeting, Madison.

[MAZ] G. MAZARE
Systèmes multi-microprocesseurs : problèmes de parallélisme, définition et évaluation d'un système particulier.
Thèse d'état, Grenoble, Juin 1978.

[MER] D. MERLE, D. POTIER, M. VERAN
Un outil d'analyse des performances des systèmes informatiques.
Congrès AFCET 1978.

[MEY] P.A. MEYER, L.H. SEAWRIGHT
CP/67 : a virtual machine time-sharing system.
IBM system journal, vol.9 n°3.

[MOR1] J.E. MORRISON
User program performance in virtual storage systems.
IBM system journal n° 3, 1973.

[MOR2] J.B. MORRIS
Demand paging through utilization of Working Sets on the MANIAC II.
CACM vol.15 n° 10, Octobre 1972.

[MOS] J. MOSSIERE
Méthodes pour l'écriture des systèmes d'exploitation.
Thèse d'état INPG, Septembre 1977.

[MUR] K. MURAKAMI, S. NISHIKAWA, M. SATO
Polyprocessor system analysis and design.
Proc. 4th annual symp. on computer architecture, 1977.

[ORG] E.I. ORGANICK
The MULTICS system : an examination of its structure.
MIT Press, Cambridge Mass., 1972.

[PEQ] M. PEQUIGNOT
LP80 - Metteur au point.
Manuel d'utilisation - CICG.

[POT] D. POTIER
Modèles à files d'attente et gestion de ressources dans les systèmes informatiques.
Thèse d'état INPG, 1977.

[POU] G.H. POUJOULAT
The CORAIL building block system.
Proc. 4th annual computer architecture symp., Silver Springs, Mars 1977.

[RAV] F. RAVEAU, P. WAGNER
Etude des espaces de travail des tâches utilisatrices sous le système SIRIS8.
Rapport de projet ENSIMAG, Juin 1977.

[ROD] J. RODRIGUEZ-ROSELL, J.P. DUPUY
The design, implementation and evaluation of a Working Set dispatcher.
CACM, Avril 1973.

[ROH] J. ROHMER, D. TUSERA
Experimental top-down specification of specialized microprocessors networks. Application to an APL Terminal.
Proc. EUROMICRO Workshop, Nice, 1975.

[SEF] B. SEFSAF
Etude et spécification d'une mini-station de transport dans le cadre du projet OURS.
Projet de DEA, INPG, Septembre 1978.

[SEN] D. SENGER
A multiple instruction stream processor.
Proc. EUROMICRO Workshop, Nice, 1975.

[TAN] Y. TANAKA & al.
HARPS, a new hierarchical array processor system.
Proc. EUROMICRO symp., Venise, Octobre 1976.

[TAR1] A. TARABOUT
ESLOGIC et ESPHYSIQ - Réalisation des entrées/sorties.
OURS 2003, Juin 1976.

[TAR2] A. TARABOUT
Gestion optimale des entrées/sorties sur disques et tambours.
OURS 2002, Mai 1976.

[TEO1] T.J. TEOREY, T.B. PINKERTON
A comparative analysis of disk scheduling policies.
CACM vol.15 n°3, Mars 1972.

[TEO2] T.J. TEOREY
Properties of disk scheduling policies in multiprogrammed computer systems.
Proc. AFIPS, FJCC, 1972.

[VAT1] I. VATTON
Segmentation - Pagination.
OURS 2000, Novembre 1976.

[VAT2] I. VATTON
MAS-UGR. Manuel de présentation.
OURS 1002, Janvier 1976.

[VIV] A. VIVIER, H. ZIMMERMANN
Standard virtual device protocol.
Publications Cyclades SCH 550, Octobre 1975.

[VYS] V.A. VYSSOTSKY, F.J. CORBATO, R.M. GRAHAM
Structure of the MULTICS supervisor.
FJCC 1965.

[WEIN] A. WEINGARTEN
The Eschenbach drum scheme.
CACM vol.9 n°7, Juillet 1966.

[WHI] H. WHITFIELD, A.S. WIGHT
EMAS : The Edinburgh multiaccess system.
Computer Journal vol.16 n°3, 1973.

[WUL] W.A. WULF editor
The HYDRA operating system.
Carnegie Mellon University.

[ZUR] F.W. ZURCHER, B. RANDELL
Iterative multi-level modelling. A methodology for computer system design.
IFIP Congress, Edinburgh 1968.

Notes internes

[BRI2] J. BRIAT, S. ROUVEYROL, A. TARABOUT
Evaluation du langage LIS.
OURS 9005.2, Octobre 1976.

[BRI3] J. BRIAT
Manuel de présentation MAS-HARDEXT.
OURS 1001.

[BRI4] J. BRIAT, J. ESTUBLIER
Manuel langage MAS-HARDEXT.
OURS 3000, Mars 1977.

[BRI7] J. BRIAT & al.
Projet OURS - Manuel de présentation.
OURS 1000, Avril 1976.

[DEL] M. DELAUNAY
Rapport d'évaluation du système LIS.
Note interne, Novembre 1977.

[EST2] J. ESTUBLIER, A. TARABOUT, I. VATTON
Manuel langage MAS-UGR.
OURS 3001.2, Octobre 1976.

[GEM] Projet GEMAU
Spécifications internes.
Note interne, Grenoble, 1975.

[LAF2] P. LAFORGUE
La répartition des ressources.
OURS 1005, Novembre 1977.

[TAR1] A. TARABOUT
ESLOGIC et ESPHYSIQ - Réalisation des entrées/sorties.
OURS 2003, Juin 1976.

[TAR2] A. TARABOUT
Gestion optimale des entrées/sorties sur disques et tambours.
OURS 2002, Mai 1976.

[VAT1] I. VATTON
Segmentation - Pagination.
OURS 2000, Novembre 1976.

[VAT2] I. VATTON
MAS-UGR. Manuel de présentation.
OURS 1002, Janvier 1976.

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU les rapports de présentation de Messieurs :

- C. KAISER, Maître de Conférences au C.N.A.M. - PARIS -
- J.P. VERJUS, Professeur à l'Université de RENNES

Monsieur MAILLOT Bernard

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Génie Informatique".

Grenoble, le 4 Décembre 1978

Ph. TRAYNARD
Président
de l'Institut National Polytechnique

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU le rapport de présentation de Monsieur :

- J.P. VERJUS, Professeur à l'Université de RENNES

Monsieur Alain TARABOUT

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie Informatique".

Grenoble, le 4 Décembre 1978

Ph. TRAYNARD
Président
de l'Institut National Polytechnique

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU le rapport de présentation de Monsieur :

- J.P. VERJUS, Professeur à l'Université de RENNES

Madame Irène VATTON née CIUTI

est autorisée à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie Informatique".

Grenoble, le 4 Décembre 1978

Ph. TRAYNARD
Président
de l'Institut National Polytechnique