Un outil de recherche en système - le modèle de décomposition

Irène Vatton

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR ingénieur et 3ème cycle

Spécialité : GENIE INFORMATIQUE

par

Irène VATTON

UN OUTIL DE RECHERCHE EN SYSTEMES INFORMATIQUES

LE MODELE DE DECOMPOSITION

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

A Pierre, Pascal et Magali.

S O M M A I R E

1. INTRODUCTION

Il est couramment admis que la décomposition en modules est une bonne méthode de conception de systèmes (MOS). La méthode d'abstraction raffinement (DIJ) est très utilisée pour construire progressivement des systèmes classiques par adjonction de modules.

Aujourd'hui, l'apparition des microprocesseurs sur le marché permet d'envisager de réaliser des systèmes en les éclatant sur une grappe de processeurs (ANC, DAR).

Notre but est de montrer que la méthode d'abstraction raffinement facilite l'étude et la mise en oeuvre de systèmes construits sur une coopération d'unités spécialisées. Pour présenter notre modèle de décomposition de systèmes nous abordons d'abord l'aspect communication puis l'aspect connexion d'unités.

2. COMMUNICATION INTER-UNITES

Une unité est un regroupement de données rémanentes et de fonctions de manipulations de ces données.

Vis-à-vis du monde extérieur l'unité apparaît comme une boîte noire disposant de pattes d'entrées (les fonctions externes) et de pattes de sorties (les appels des fonctions externes d'autres unités). Ces unités indépendantes doivent pour communiquer échanger des messages. On peut obtenir différents comportements d'unités selon la sémantique des messages échangés.

Cette sémantique dépend du type de communication utilisé :

- la communication de type procédural
- ou bien la communication par message.

2.1. COMMUNICATION DE TYPE PROCEDURAL

La communication de type procédural est très utilisée dans le domaine des langages de programmation. Elle permet d'élaborer les conditions de reprise de l'exécution d'une unité au moment de l'appel d'une autre unité.

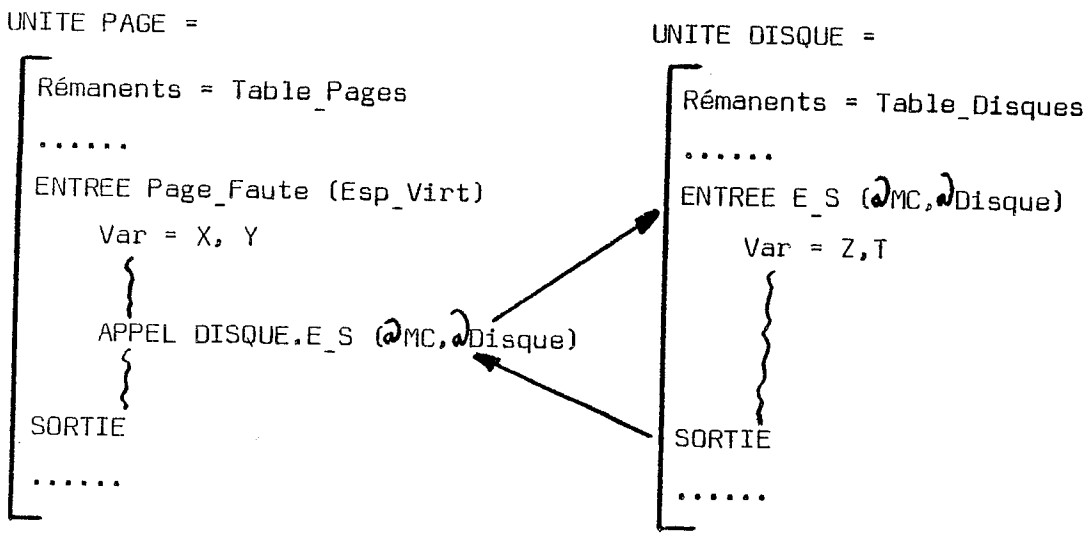
Son fonctionnement est défini par les trois primitives :

ENTREE : cette primitive indique au mécanisme de communication l'emplacement d'une fonction externe de l'unité et ses paramètres formels.

SORTIE : cette primitive indique la terminaison de la fonction externe et engendre l'exécution du retour procédural.

APPEL : cette primitive provoque l'exécution d'une fonction externe d'une autre unité pour les paramètres réels donnés.

Si l'on prend l'exemple suivant de deux unités PAGE et DISQUE :



L'exécution de l'instruction APPEL DISQUE.E_S (∂MC,∂Disque) de l'unité PAGE apparaît comme le lancement d'une exécution sur une unité distante (exécution de la fonction E_S sur l'unité DISQUE) et l'attente de la fin de cette exécution.

Cette synchronisation des exécutions de fonctions d'unités distinctes est résolue différemment suivant l'organisation en processus choisie :

a) Une organisation en processus de traitement

Avec une organisation en processus de traitement, les fonctions d'unités appelées en cascade sont déroulées par le même processus (CHEV).

Le déroulement séquentiel des instructions du processus assure par lui-même la synchronisation des exécutions des fonctions.

On utilise généralement une pile d'exécution pour conserver à la fois l'état courant du contexte d'exécution (variables de travail) du processus dans chaque unité et les paramètres échangés entre les fonctions des unités.

Le mécanisme de communication doit alors transporter cette pile d'exécution de l'unité appelante à l'unité appelée et inversement au moment de l'APPEL et de la SORTIE d'une fonction. Ce transport des piles d'exécution des processus est peu coûteux quand les unités ont accès à une mémoire commune. Le contexte échangé par les unités est le pointeur de pile. Si les unités sont des boîtes noires fermées le découpage en processus de traitement implique le transfert d'un gros volume d'informations (la pile).

b) Une organisation en processus serveurs

Avec une organisation en processus serveurs chaque processus est chargé de dérouler un ensemble déterminé de fonctions, celles d'une unité (BRIN).

Les unités PAGE et DISQUE présentées plus haut se présentent alors comme des processus communicants.

Le mécanisme de communication doit dans ce cas contrôler la synchronisation des processus (GRA) :

Le processus PAGE ne peut continuer tant que le processus DISQUE n'a pas achevé le service d'Entrée_ Sortie.

Avec le découpage en processus serveurs l'information transmise entre unités est assez réduite. Les contextes d'exécution des processus sont maintenus dans chaque unité. Les fonctions échangent des messages contenant les paramètres plus l'information de contrôle (APPEL ou SORTIE, identification de l'unité).

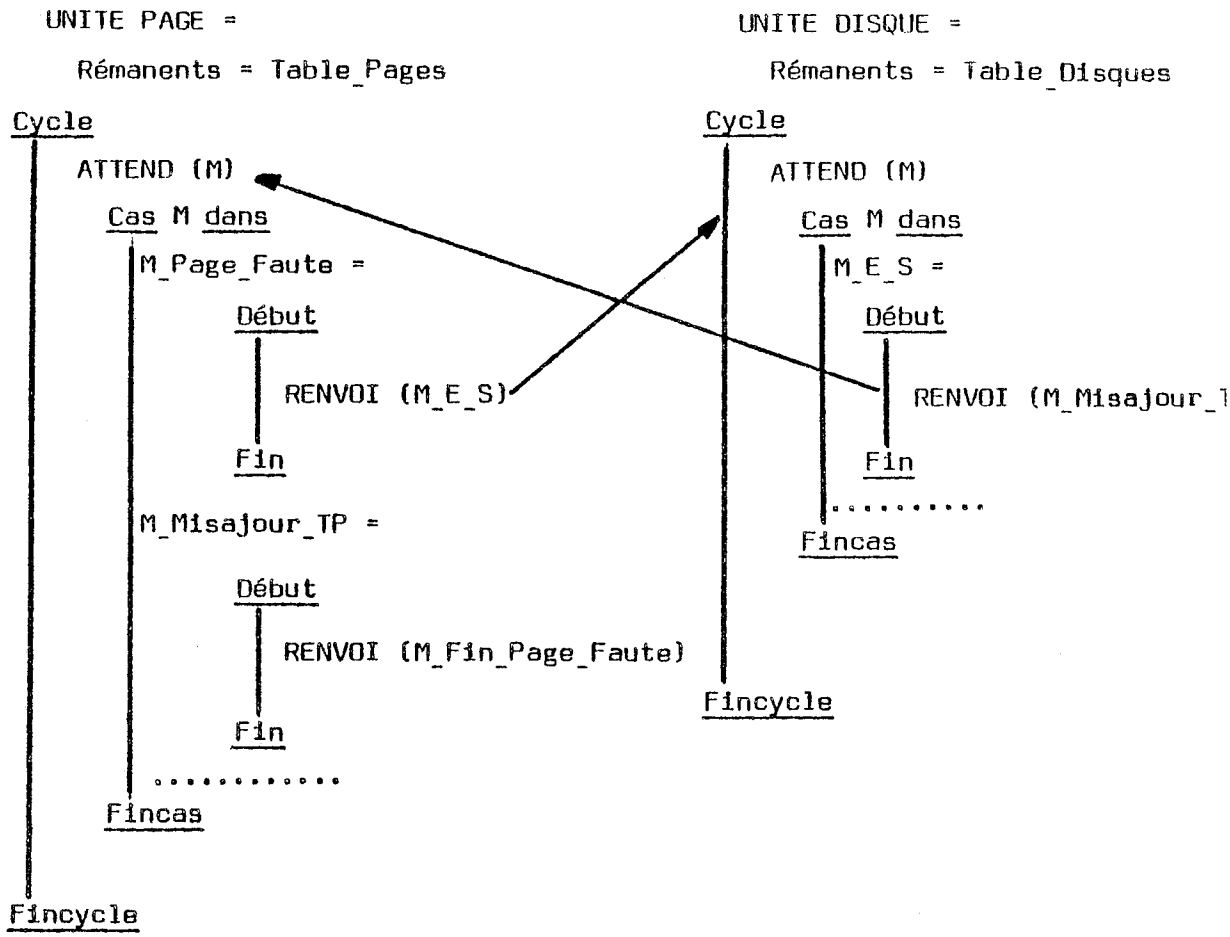
2.2. COMMUNICATION PAR MESSAGE

La communication par message est surtout employée dans les réseaux d'ordinateurs (ELI). Elle est plus générale que la précédente puisqu'elle permet de développer tout type de contrôle, y compris le contrôle procédural. Sa mise en oeuvre nécessite deux primitives :

ATTEND : cette primitive indique au mécanisme de communication qu'un message est attendu.

RENVOI : cette primitive soumet un message au mécanisme de communication. Ce message contient une identification du destinataire.

Si l'on reprend l'exemple précédent des deux unités PAGE et DISQUE on obtient la description suivante :



Ce type de communication suppose une organisation de système en processus serveurs (HOAR, GREEN). Ici ce sont des processus serveurs cycliques. Les messages peuvent être tamponnés ou échangés sur rendez-vous. Dans ce dernier cas, l'exécution du RENVOI d'un message n'est achevée que quand le destinataire a consommé le message par un ATTEND.

Notons que l'on peut associer à la primitive RENVOI l'ordre de destruction du processus serveur qui l'exécute. La primitive ATTEND devient alors superflue : le mécanisme de communication crée un processus serveur pour chaque message reçu sur l'unité.

Un traitement du système peut mettre en jeu différentes unités. La réalisation d'un traitement basé sur la coopération d'unités nécessite, avec une communication par message, l'introduction d'un nouveau type de variables : ces variables ne sont pas liées à une unité (variables rémanentes) ni à une fonction (variables de travail) mais à l'exécution d'un traitement.

Reprenons l'exemple des unités PAGE et DISQUE :

La résolution d'une faute de page se traduit par l'émission successive du message M_Page_Faute vers l'unité PAGE, du message M_E_S vers l'unité DISQUE et du message M_Misajour_TP vers l'unité PAGE.

Les messages M_Page_Faute et M_Misajour_TP doivent contenir l'identification de l'espace virtuel concerné. Par contre, le contenu du message M_E_S peut se réduire aux adresses de mémoire centrale et de mémoire secondaire impliquées par l'entrée/sortie.

Si la fonction d'Entrée/Sortie de l'unité DISQUE veut pouvoir engendrer le message M_Misajour_TP elle doit détenir l'identification de l'espace virtuel. Le message M_E_S doit donc contenir aussi cette identification.

Pour éviter de faire manipuler par la fonction d'Entrée/Sortie une information que ne la concerne pas (l'identification de l'espace virtuel) on peut procéder comme suit :

On définit un format de message pour chaque traitement du système (un message de résolution de faute de page) et chaque fonction d'unité impliquée reçoit ce message sans pouvoir accéder les champs qui ne la concerne pas.

Ce type de message se rapproche beaucoup de celui défini dans PLITS (FELD).

2.3. LE MECANISME CHOISI

Le mécanisme de communication de type procédural parait bien adapté à la construction progressive de systèmes, il est d'ailleurs largement utilisé (LAF, BEK, ORG).

Ce mécanisme présente malgré tout quelques inconvénients quand on essaie de l'appliquer sur une structure d'unités réparties sur différents microprocesseurs (BRIA 2, EST).

Ce mécanisme est coûteux en espace mémoire. Il implique la conservation de la totalité des contextes d'exécution pour chaque unité traversée. Ces contextes contiennent beaucoup plus d'informations que ce qui est nécessaire pour achever le traitement dans les unités concernées (cf. 2.2.).

Ce mécanisme annule aussi un des avantages d'une structure répartie à savoir l'augmentation du débit du système grâce à l'exécution "pipe-line" ou parallèle (JAM, LAUE) des fonctions de différentes unités impliquées dans un même traitement du système. Nous avons donc choisi un mécanisme de communication par message.

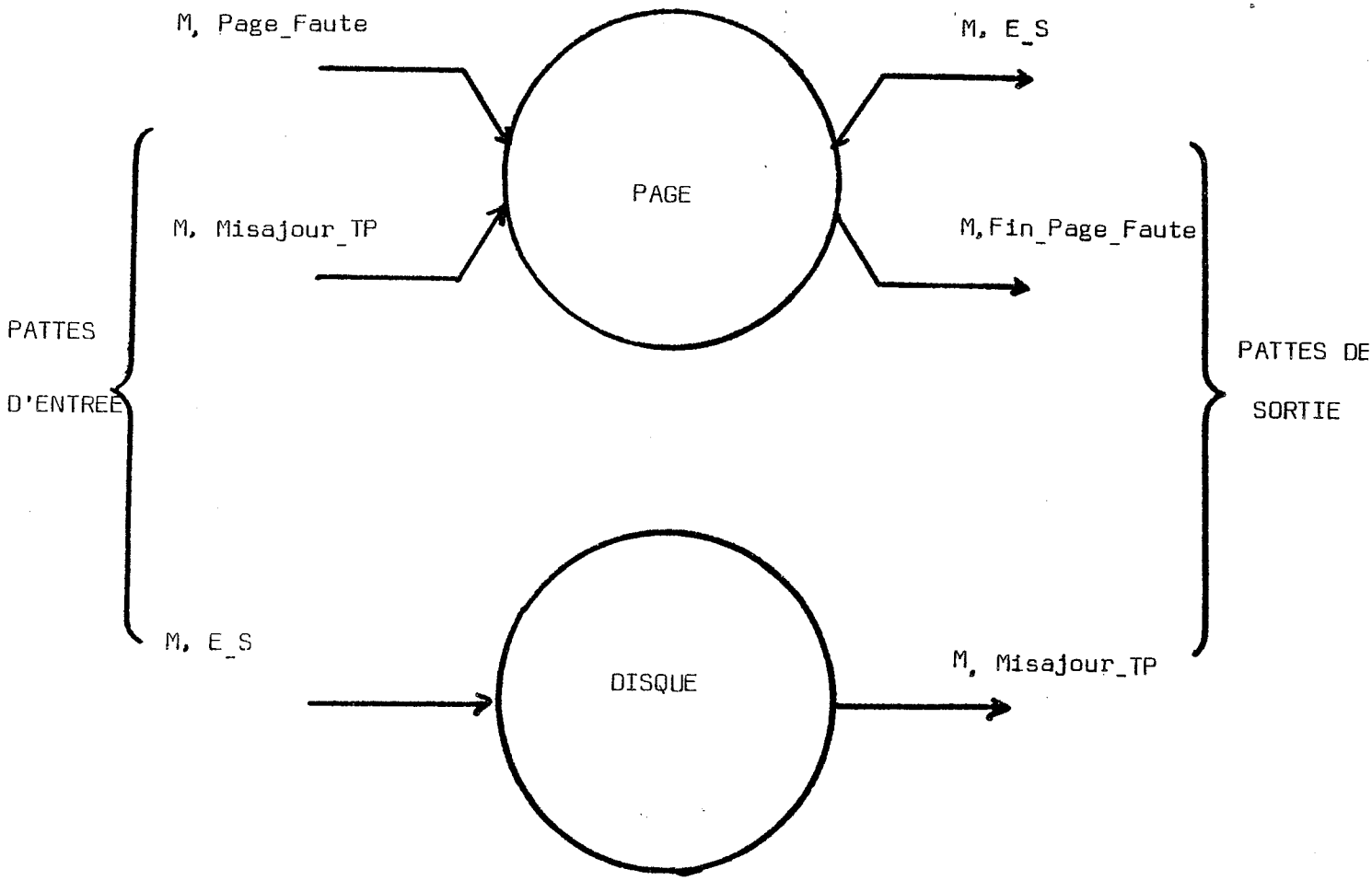
Chaque unité est définie par l'association d'une boîte aux lettres et d'un processus cyclique (semblable à ceux décrits au § 2.2.). La boîte aux lettres sert à tamponner les messages. Les processus ne sont donc jamais bloqués sur un RENVOI de message.

3. CONNEXION D'UNITES

3.1. CONNEXION STATIQUE ET DYNAMIQUE

Nous avons défini avec le mécanisme de communication les conventions d'échange d'information entre unités. Nous n'avons pas jusqu'à présent précisé comment les unités sont connectées entre elles. A priori le problème paraît simple. Il suffit d'associer à chaque unité une patte d'entrée par type de message admis et une patte de sortie par type de message sorti . Une boîte aux lettres d'une unité comporte alors autant de files de messages qu'il y a de pattes d'entrée dans l'unité.

Soit pour les unités PAGE et DISQUE présentées plus haut :



La connexion des unités est définie par la liste des couples patte d'entrée-patte de sortie.

a) Connexion statique

La constitution des couples peut être statique. Dans ce cas, une patte de sortie ne peut être associée qu'à une seule patte d'entrée. Cette connexion statique est très contraignante quand le mécanisme de communication ne gère pas le retour procédural : deux unités ne peuvent pas utiliser une même troisième comme un sous-programme, si cette troisième n'a pas prévu la multiplication de ses pattes d'entrée et de sortie.

b) Connexion dynamique

L'association patte d'entrée - patte de sortie peut être rendue dynamique. Si l'on adjoint aux messages renvoyés une information de routage, le mécanisme de communication peut les acheminer vers les différentes pattes d'entrée désirées. Cette connexion dynamique laisse les unités maîtres de la destination des messages renvoyés. Le mécanisme de communication ne peut pas s'apercevoir qu'une unité a interverti les informations de routage.

c) Connexion semi-statique

On voit que si la connexion statique est très contraignante, la connexion dynamique ne permet aucun contrôle sur l'acheminement des messages. Une solution intermédiaire consiste à définir des graphes de connexion de façon statique. Chaque traitement du système s'associe un graphe de connexion c'est l'information de routage du message de traitement. Au moment d'un RENVOI d'une unité, l'acheminement du message est déterminé par le graphe de connexion suivi. C'est cette solution que nous avons adoptée pour éliminer les deux inconvénients énoncés plus haut.

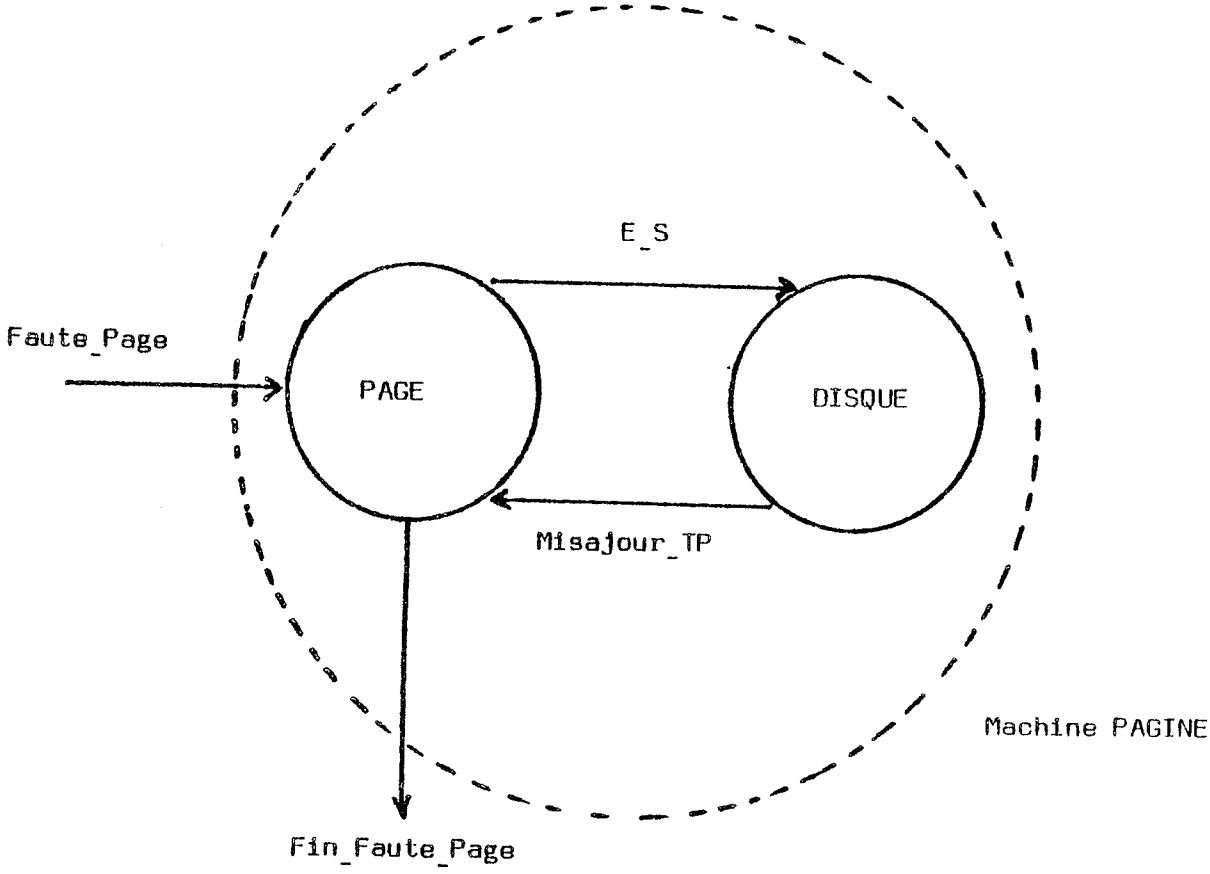
3.2. UTILITE DES CONNEXIONS

Notre but est de permettre la construction de systèmes à partir d'unités définies indépendamment les unes des autres. Pour construire ces systèmes nous voulons exploiter la méthode d'abstraction-raffinement.

3.2.1. L'abstraction

Définir une machine abstraite revient à définir son jeu d'instructions. Le traitement d'une instruction d'une machine abstraite peut être réalisé par une fonction d'une unité ou bien nécessiter la coopération de plusieurs unités. Une machine abstraite est alors définie par l'ensemble des unités qu'elle utilise et un graphe de connexion qui traduit les coopérations d'unités qu'elle implique. Nous avons matérialisé une machine abstraite par un graphe de connexion complet où apparaissent toutes les unités impliquées ainsi que les pattes d'entrée et de sortie d'unités correspondantes aux lancements et terminaisons des instructions de la machine abstraite.

Par exemple, la définition d'une machine abstraite PAGINE en terme d'unités PAGE et DISQUE (cf. § 3.1.) correspond au graphe de connexion suivant.


E_S
Faute_Page
PAGE
DISQUE
Misajour_TP
Machine PAGINE
Fin_Faute_Page

L'unique instruction de cette machine abstraite est lancée par l'envoi d'un message sur la patte d'entrée de la machine (Faute_Page) et terminée par le retour du message sur la patte de sortie de la machine (Fin_Faute_Page).

En associant à chaque message transmi l'identification de la machine abstraite, donc du graphe de connexion, concernée le mécanisme de connexion peut déterminer la patte d'entrée de l'unité destinataire du message.

On peut dire que la définition d'une machine abstraite présente l'aspect statique du mécanisme de connexion et l'exécution des instructions l'aspect dynamique.

3.2.2. Le raffinement

La méthode de raffinement consiste à exprimer la réalisation d'une machine abstraite en termes d'autres machines abstraites. Plus généralement, on utilise le raffinement quand on remplace une séquence d'instructions par une instruction plus évoluée (c'est le cas d'un appel de procédure). Dans la structure que nous avons présenté les instructions des machines abstraites sont interprétées par les unités de la machine. Pour permettre la réalisation d'une machine abstraite en terme d'autres machines abstraites, il faut permettre de construire des unités sur des machines abstraites.

En effet, jusqu'à présent nous avons supposé que les unités préexistaient. Or, une unité n'est rien d'autre qu'une séquence d'instructions encadrée par les primitives de communication ATTEND et RENVOI. La création de nouvelles unités sur une machine abstraite est possible si la machine est d'une part capable de conserver les programmes de ces unités et d'autre part capable d'interpréter les instructions des programmes.

Si l'on prédéfinit deux unités appelées ATTEND et RENVOI capables d'interpréter les primitives de même nom, l'inclusion de ces deux unités dans une machine abstraite lui permet d'interpréter de nouvelles unités (nous n'abordons pas ici le problème de la mémoire).

3.2.3. L'enrichissement et la réduction

Avec l'abstraction et le raffinement nous avons défini une méthode de construction progressive de machines abstraites.

Avec l'enrichissement et la réduction nous offrons la possibilité de définir facilement un éventail de machines abstraites de même niveau, plus ou moins puissantes.

En effet, si l'on veut définir deux machines abstraites identiques à une instruction près, on doit normalement décrire chacune des machines abstraites par un graphe de connexion incluant ou non l'unité qui interprète cette instruction.

Quand ces graphes de connexion sont très complexes il parait intéressant de factoriser le sous-ensemble commun à ces graphes. Cette factorisation est obtenue par l'inclusion d'une machine abstraite dans une autre machine abstraite.

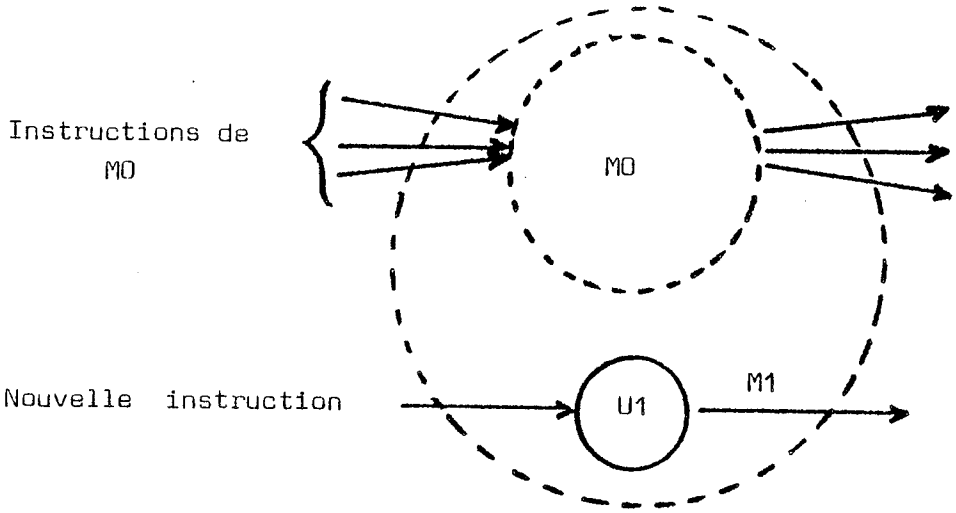
Cette inclusion est possible puisqu'une machine abstraite présente le même interface externe qu'une unité (pattes d'entrée et pattes de sortie).

L'enrichissement d'une machine abstraite correspond à l'adjonction d'une ou plusieurs unités interpréteurs de nouvelles instructions.

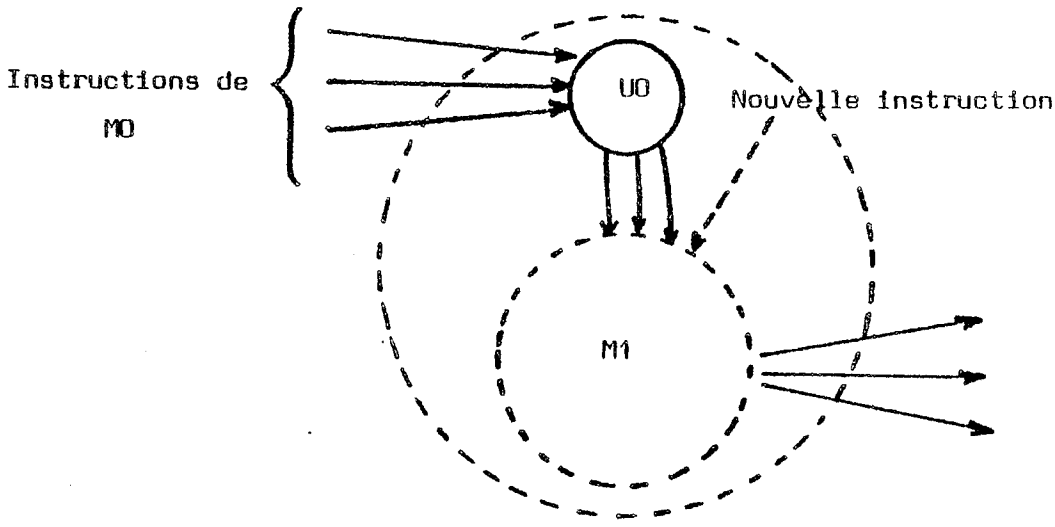
Inversement, la réduction d'une machine abstraite correspond à l'adjonction d'une unité filtre.

Soit deux machines abstraites M0 et M1 identiques à une instruction près.

La machine M1 peut être présentée comme un enrichissement de la machine M0 :



Ou bien inversement, la machine M0 peut être présentée comme une réduction de la machine M1 :



Pour réaliser l'inclusion de machines le mécanisme de communication gère l'empilement des graphes de connexion : quand un RENVOI d'une unité adresse une patte d'entrée d'une machine abstraite incluse (c'est le cas pour l'unité U0 connectée à la machine abstraite M1 dans le graphe de connexion précédant), le mécanisme de communication abandonne momentanément le graphe de connexion courant pour prendre en compte celui de la machine incluse.

Quand un RENVOI d'une unité adresse une patte de sortie de la machine incluse, le mécanisme de communication reprend le graphe de connexion abandonné.

3.3. DESCRIPTION DU MATERIEL

Jusqu'ici nous avons présenté les outils de construction de niveaux de machines abstraites suivant les méthodes d'abstraction, de raffinement, d'enrichissement et de réduction.

Ce modèle de décomposition ne serait pas complet s'il ne permettait pas de prendre en compte la structure du matériel.

Généralement, le matériel se prête bien à une décription en unités ou en machines abstraites. Un matériel constitué d'une grappe de processeurs spécialisés peut émuler un système d'unités ou de machines abstraites initial.

Un processeur peut être spécialisé dans le contrôle de la communication et de la connexion (JAM) : c'est alors lui qui gère les graphes de connexion des machines définies et connait les implantations des unités sur les autres processeurs.

Dans notre cas, le matériel utilisé est un ordinateur CII IRIS 80. Nous avons codé sur cet IRIS 80 un noyau de système réalisant entre autres le mécanisme de communication et de connexion.

Ce noyau présente aux concepteurs de systèmes une machine abstraite initiale appelée Machine IRIS.

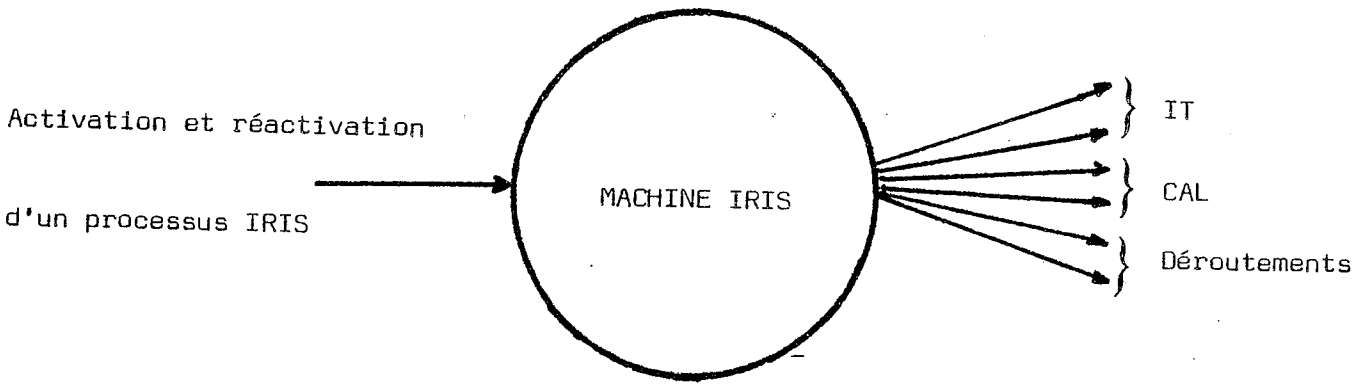
La machine IRIS interprète deux instructions qui sont :

. L'activation d'un processus IRIS : cette instruction soumet à la machine IRIS un programme écrit en code IRIS 80 pour qu'elle le stocke et l'exécute. Le code IRIS 80 contient des instructions d'appel superviseur (CAL).

Certaines instructions d'appel superviseur sont associées à l'exécution de primitives de communication et de connexion (création d'unité création de machine, ATTEND et RENVOI).

Les autres instructions d'appel superviseur, les déroutements (erreurs) et les interruptions sont des pattes de sortie de la machine IRIS.

. La réactivation d'un processus IRIS : cette instruction permet de relancer l'exécution du programme associé au processus. Si le processus a été interrompu c'est qu'une instruction CAL, un déroutement ou une interruption l'a fait sortir de la machine.



Partant de cette machine IRIS le concepteur de système peut construire des unités spécialisées et de nouvelles machines abstraites.

4. EXEMPLE

On désire réaliser un système de consultation et de mise à jour d'une petite base de données (pour un centre d'anlyses médicales par exemple). Ce système doit permettre l'écriture de n-uplets (nom - adresse - date - analyse...) sur disque et la consultation de ces n-uplets. Un utilisateur choisit les n-uplets qu'il veut consulter sur la valeur d'un ou de plusieurs champs (nom = TOTO).

Les utilisateurs ont accès au système par un terminal écran. Ils ont le choix entre afficher le résultat de leur consultation sur l'écran de leur terminal ou bien le lister sur l'imprimante du système. Dans ce dernier cas, la liste des n-uplets sera précédée d'un Titre comprenant : le Nom de l'utilisateur (identification du terminal écran), l'heure, la date, et la consultation demandée.

4.1. DECOMPOSITION DU SYSTEME

Un tel système peut se décomposer en six grandes fonctions qui sont :

. L'acquisition et l'analyse des commandes émises par les utilisateurs. Les commandes reconnues sont :

AJOUTER (Id_Utilisateur, n_uplet)
AFFICHER (Id_Utilisateur, {id_champ =})
IMPRIMER (Id_Utilisateur, {id_champ =}).

. L'insertion d'un n_uplet dans la base de données.

. La recherche d'un n_uplet en fonction d'un contexte: la recherche est terminée dès qu'elle a obtenu un n_uplet qui satisfait les critères de consultation ({id_champ =}). Pour recherche tous les n_uplets qui les satisfassent il faut relancer la recherche tant que le résultat n'est pas vide. A chaque coup le contexte de recherche doit être différent pour ne pas obtenir toujours le même n_uplet.

. L'affichage de lignes sur les écrans des utilisateurs.

. L'impression des titres de listes (Nom de l'utilisateur, l'heure...).

. L'impression des lignes sur l'imprimante.

Le découpage du système en unités ne peut pas être directement calqué sur cette décomposition fonctionnelle. En effet, les fonctions d'insertion et de recherche d'un n-uplet manipulent la même base de données, elles sont donc en conflit pour décider du positionnement du bras du disque.

Si l'on veut isoler chacune de ces fonctions dans une unité (unité INSERT, unité RECHERCHE), il faut introduire une troisième unité pour gérer le disque (unité DISQUE).

Cette solution est assez lourde si l'on considère que chaque unité représente l'utilisation d'un microprocesseur.

Une autre solution consiste alors à réunir ces deux fonctions (d'insertion et de recherche d'un n-uplet) dans une même et seule unité (l'unité B.D).

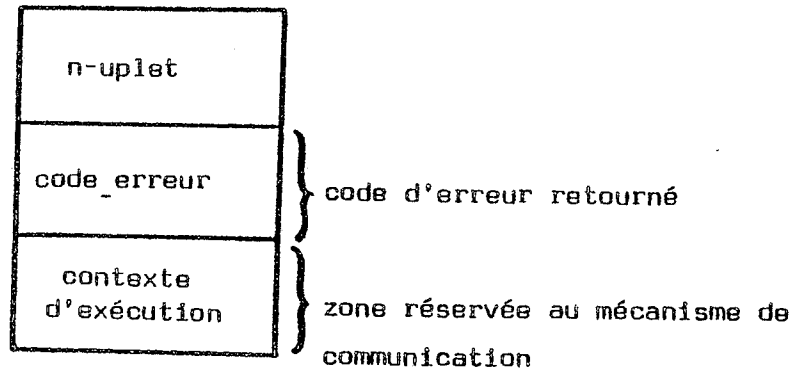
Le problème est identique pour toutes les fonctions qui ont accès aux mêmes ressources physiques : c'est le cas de l'impression des titres et de l'impression des lignes sur l'imprimante, de l'affichage des lignes et de l'acquisition des commandes si l'écran et le clavier sont indissociables.

Ces considérations conduisent à décomposer le système de consultation et de mise à jour d'une petite base de données en trois types d'unités :

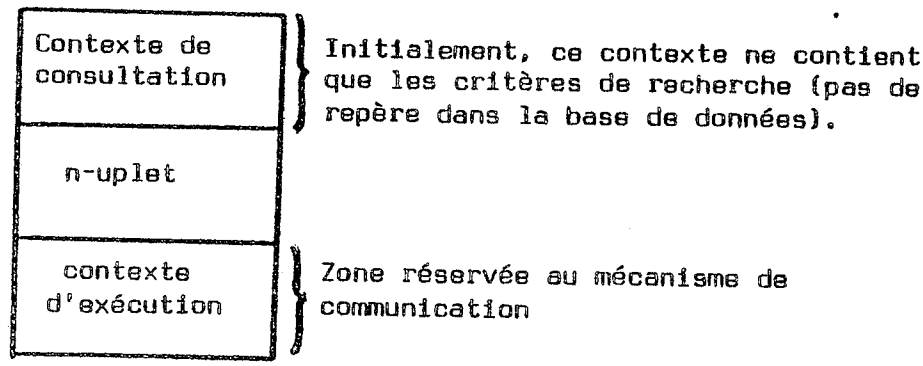
* Une unité pour gérer chaque terminal utilisateur : $UTIL_i$: elle regroupe les fonctions d'acquisition de commandes et d'affichage de lignes sur l'écran. C'est elle qui construit les messages en fonction des commandes tapées sur le clavier de l'utiisateur.

Deux formats de message sont nécessaires (cf. § 2.2.).

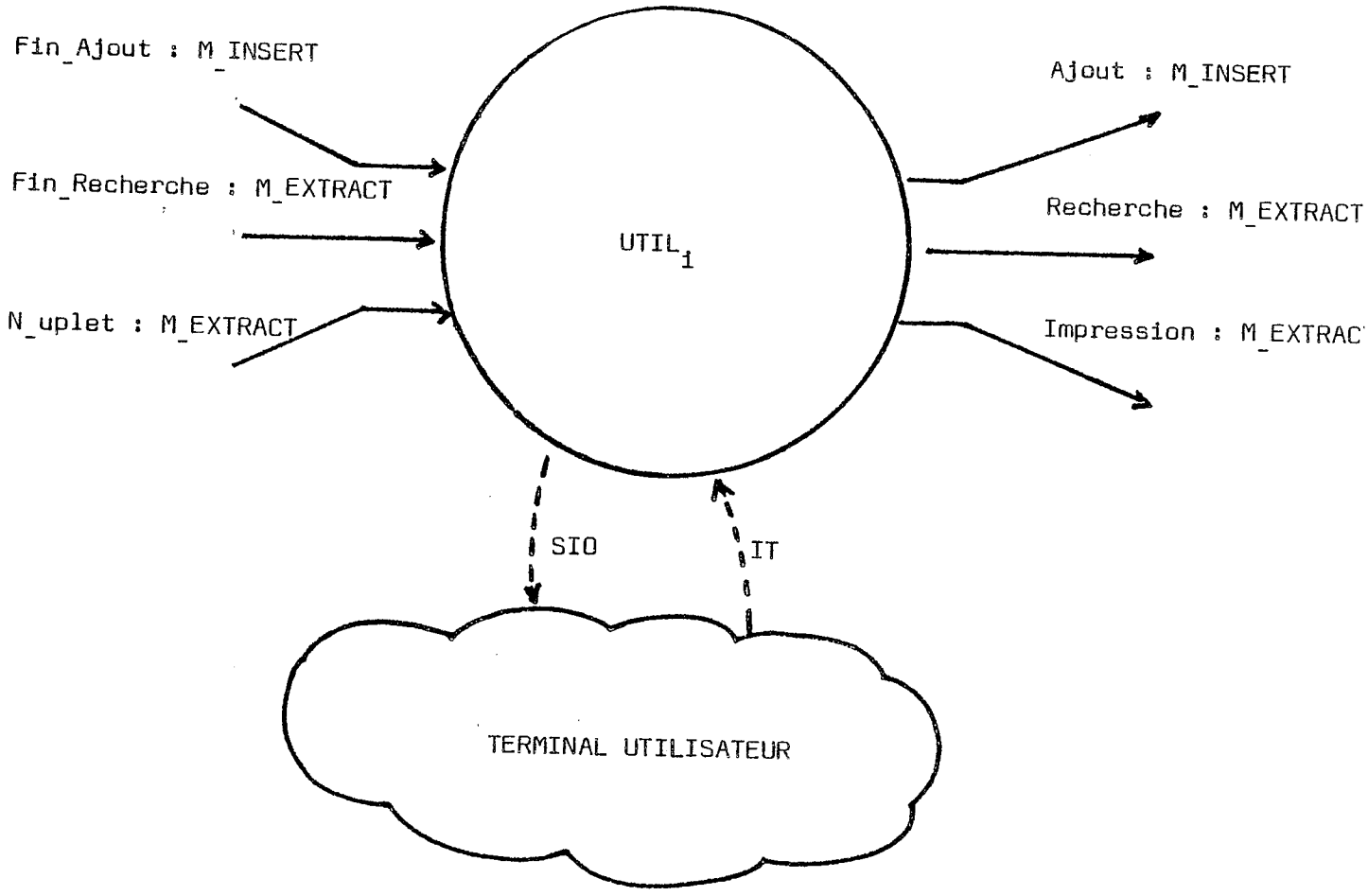
- Un message d'insertion d'un n_uplet dans la base de données : nous le désignons par M_INSERT.



- Un message d'extraction d'un n_uplet : nous le désignons par M_EXTRACT.

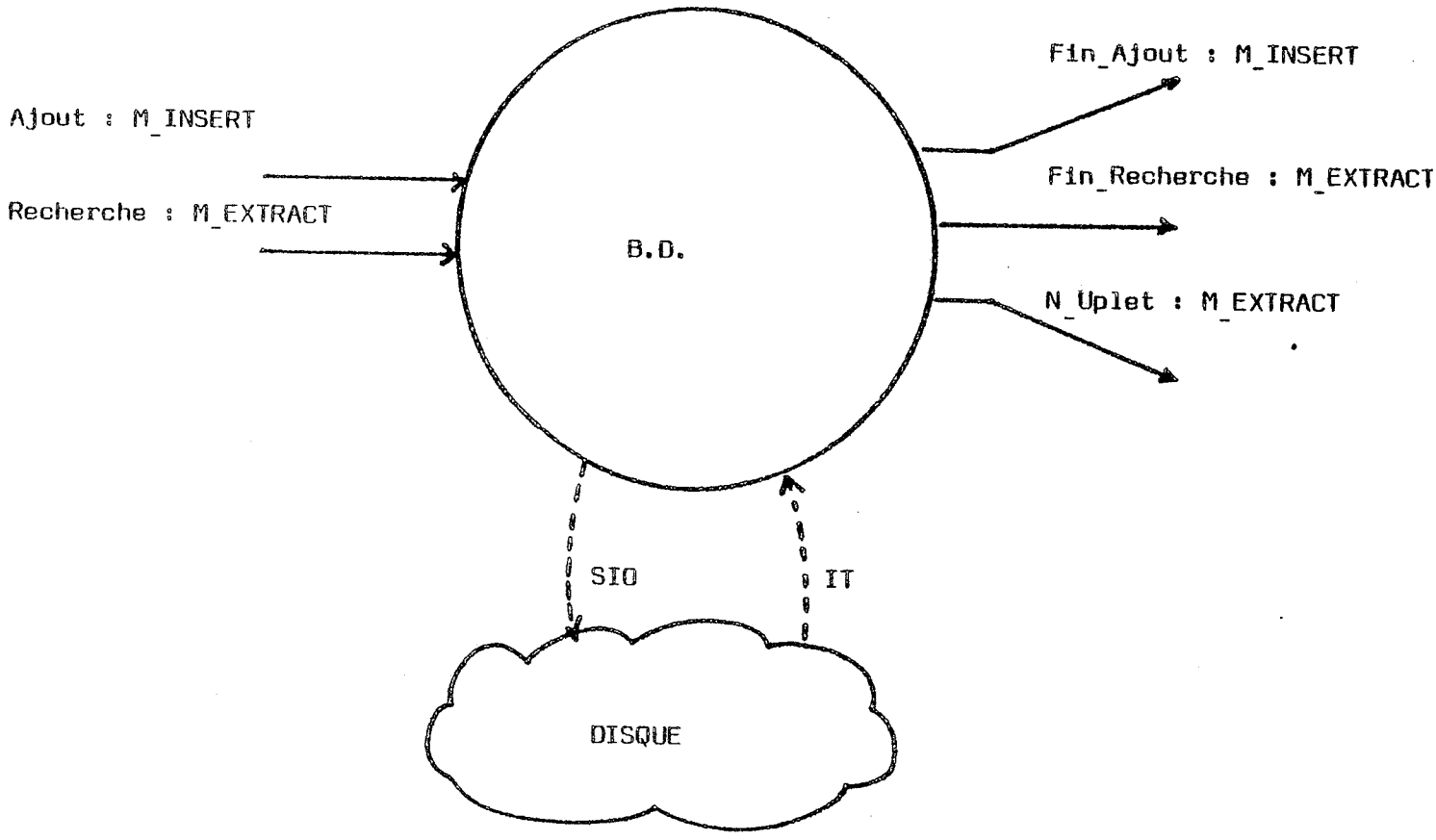


L'unité $UTIL_i$ présente quatre pattes d'entrée et de sortie :

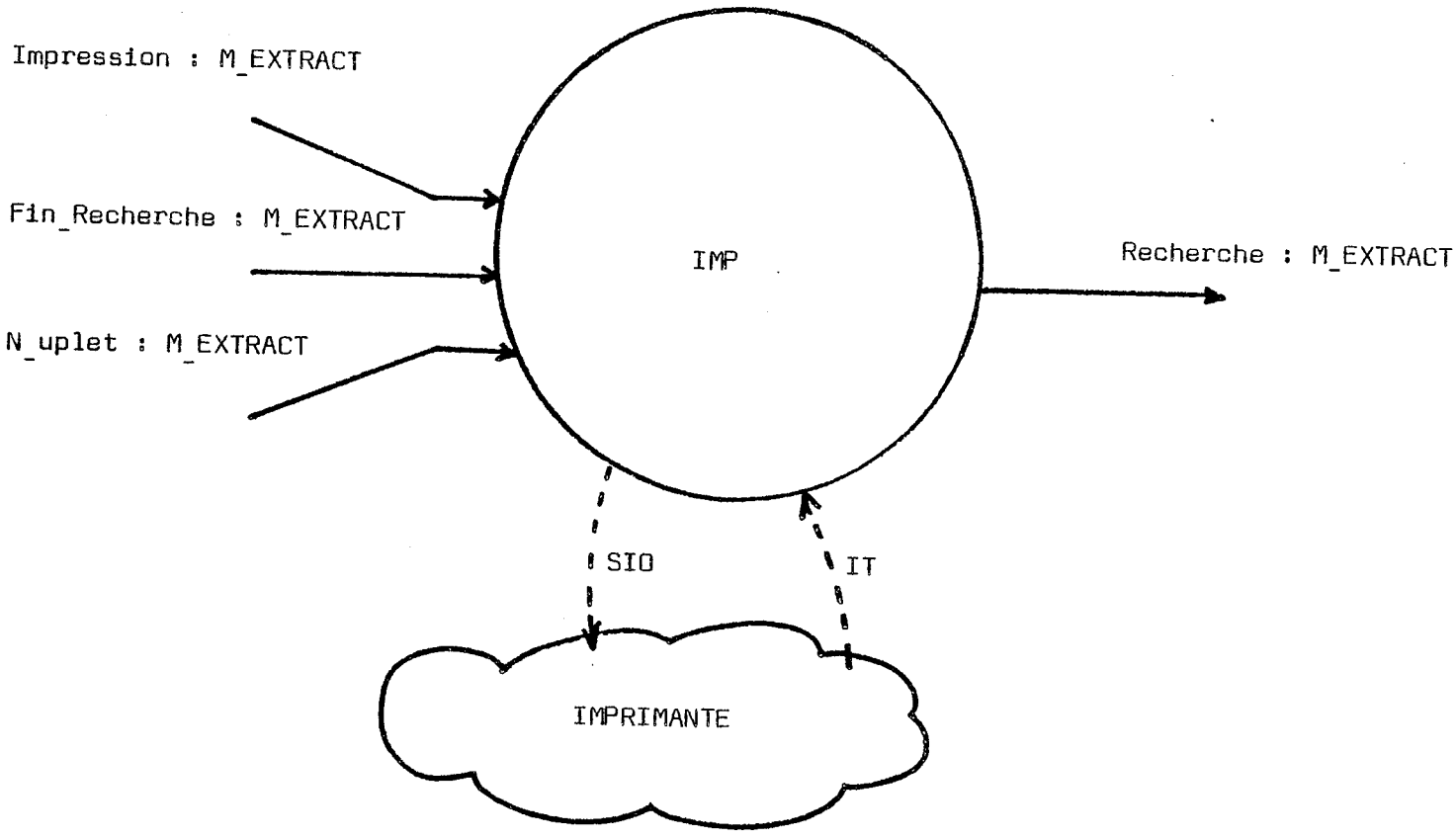


REMARQUE : les pattes d'entrée et de sortie SIO et IT ne sont pas gérées par le mécanisme de communication mais par un mécanisme matériel équivalent.

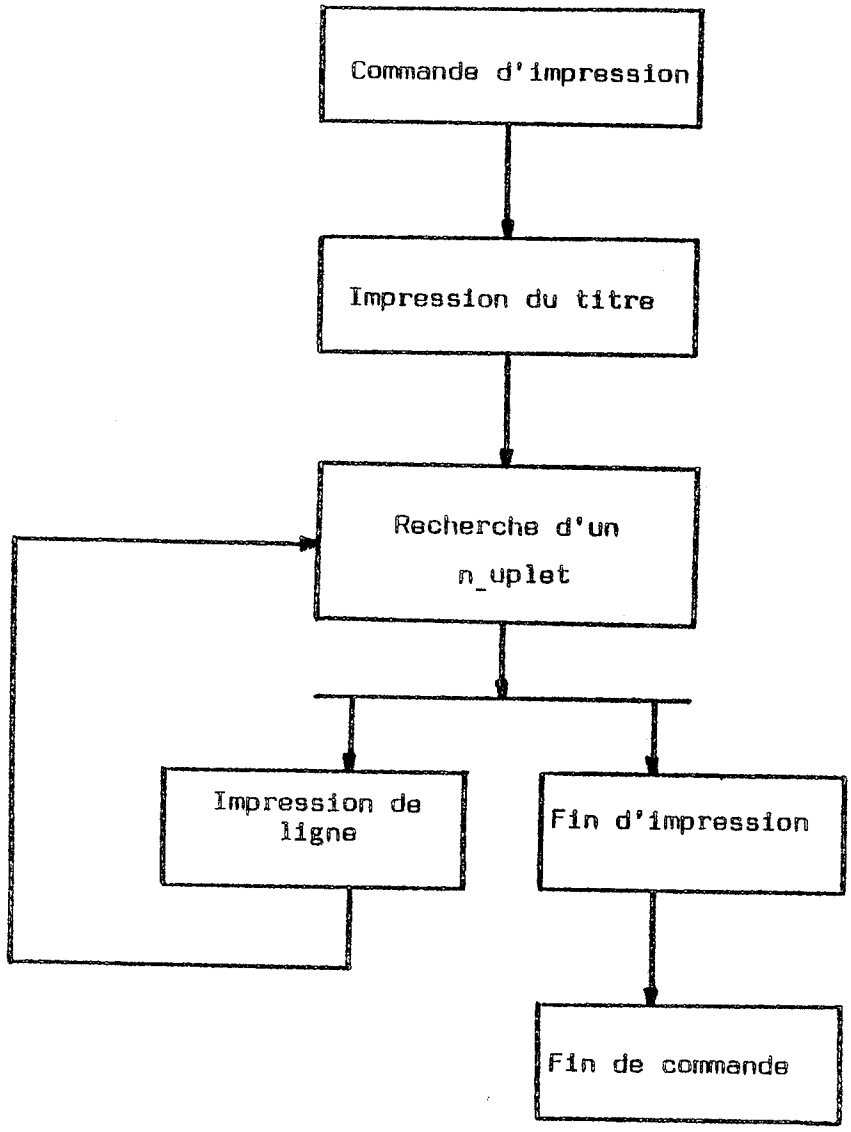
« Une unité pour gérer la base de données : BD. Elle regroupe les fonctions d'insertion et de recherche d'un n_uplet. L'unité présente trois pattes d'entrée et quatre pattes de sortie :



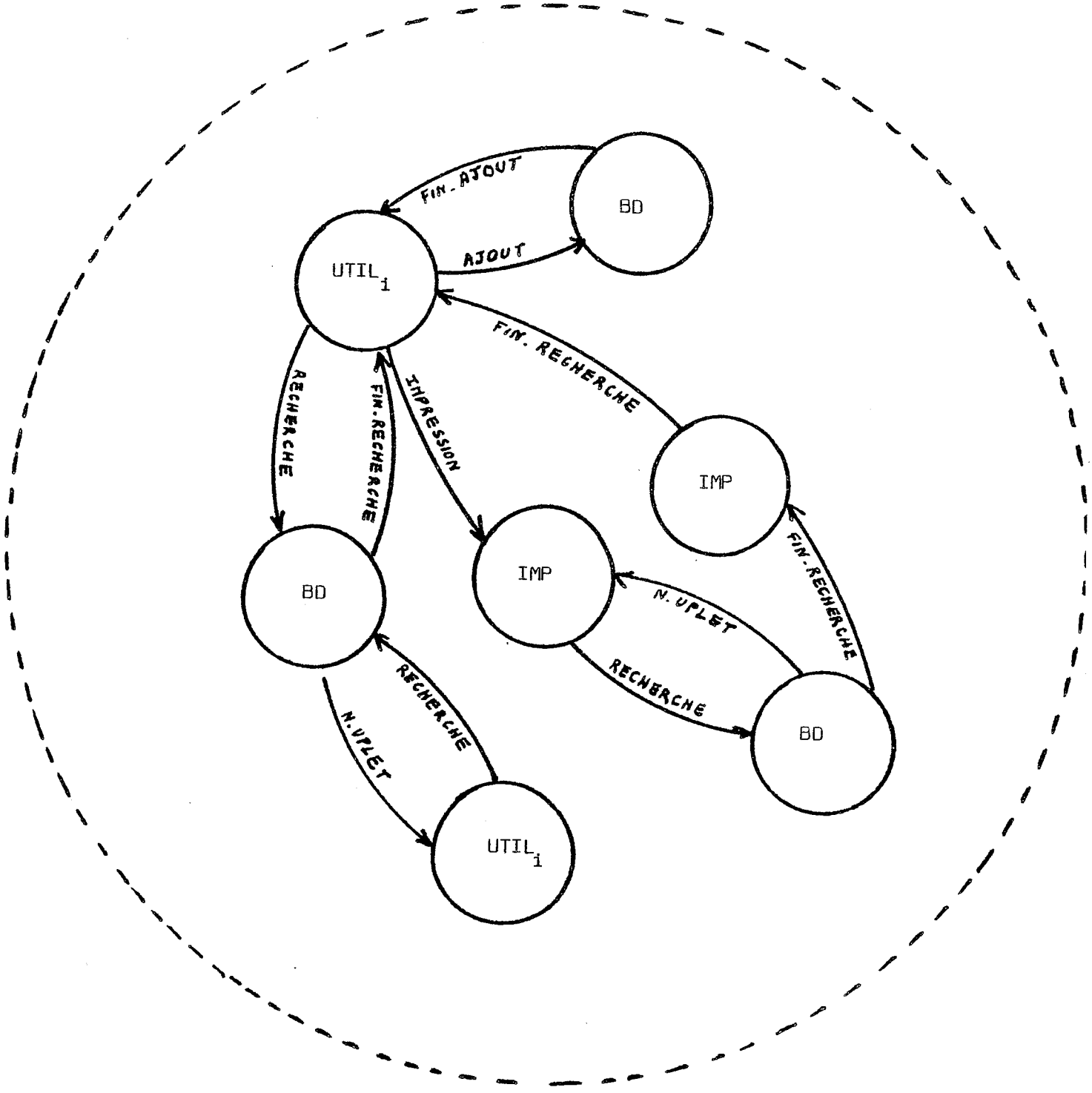
* Une unité pour gérer l'imprimante : IMP. Elle regroupe les fonctions d'impression du titre et des lignes des listes. L'unité présente quatre pattes d'entrée et deux pattes de sortie :



On peut alors décrire le fonctionnement du système en associant dans des machines les pattes d'entrée et de sortie de chaque unité. Les connexions d'unités doivent traduire l'enchainement des fonctions nécessaires à l'exécution d'une commande. Par exemple, une demande d'impression de n_uplets répondant à un certain critère se traduit par l'enchainement de fonctions suivant :



Des enchainements de fonctions nécessaires à l'exécution de chaque commande on déduit le graphe de connexion d'une machine "Système de base de données" (SBD_i) : il existe autant de machines "Système de base de données" qu'il y a de terminal écran.



MACHINE SBD_i

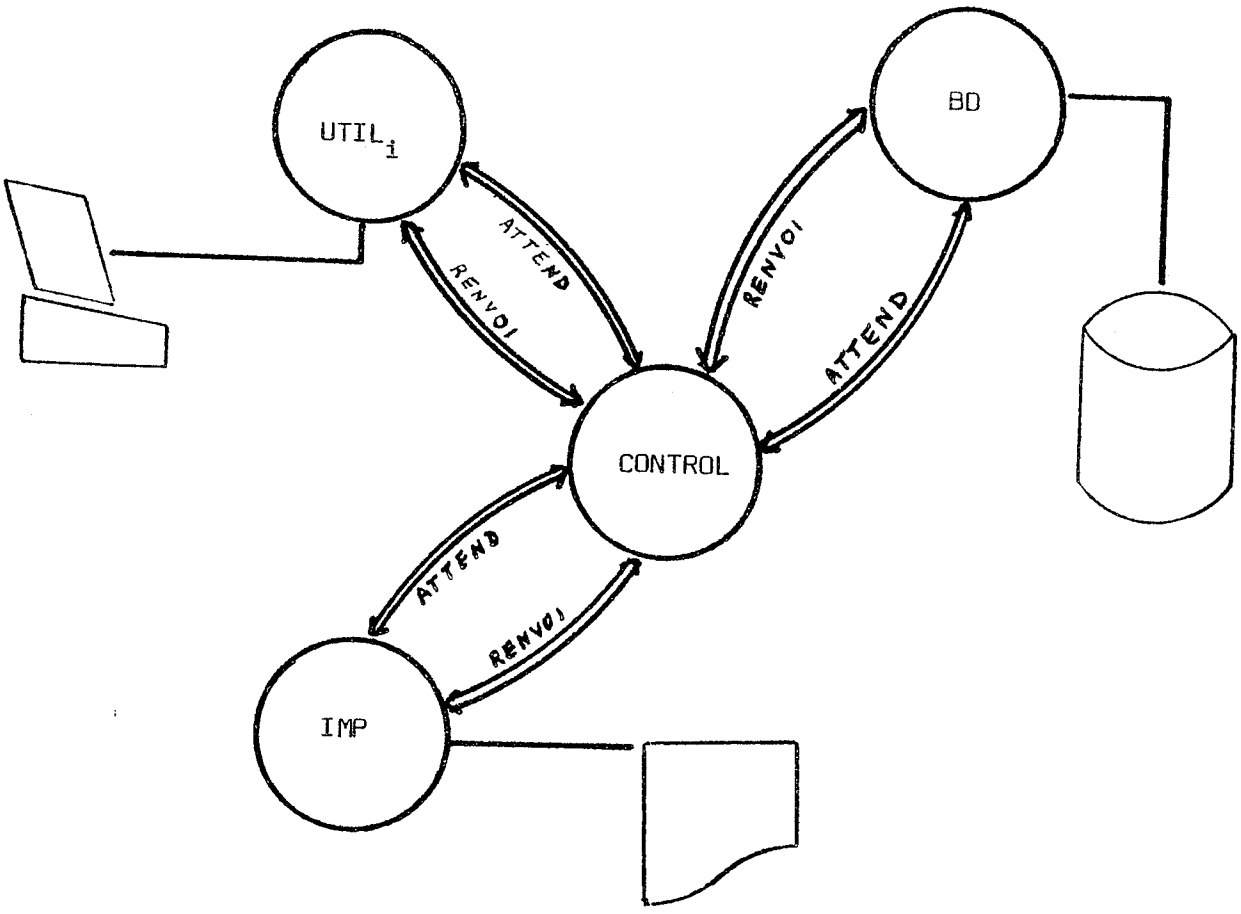
Notons que l'on peut faire apparaitre plusieurs fois une même unité dans le graphe de connexion d'une machine SBD_i. Chaque occurrence d'une même unité correspond à une invocation particulière de l'unité. La multiplicité des occurrences d'une même unité permet de contrôler les réponses valides à une invocation d'unité (la fonction fin_recherche de l'unité IMP ne peut répondre que sur la patte de sortie fin_recherche).

4.2. APPLICATION SUR LE MATERIEL

Naturellement, de la décomposition du système de consultation et de mise à jour d'une petite base de données en unités découle une organisation matérielle : chaque unité est mise en oeuvre par un microprocesseur.

Le système se compose alors de n microprocesseurs pour n terminaux écran, d'un microprocesseur associé au disque et d'un microprocesseur relié à l'imprimante. A ces n + 2 microprocesseurs, il faut ajouter un microprocesseur chargé de gérer la connexion des unités. En effet, un microprocesseur émule parfaitement le fonctionnement d'une unité (il existe en général une instruction d'attente d'interruption et une instruction de transmission de données) mais le système ne peut pas fonctionner sans un contrôleur capable d'interpréter les graphes de connexion des machines SBD_i.

a) Une première solution consiste à introduire un nouveau microprocesseur spécialisé dans le contrôle. On obtient alors un réseau de microprocesseurs en étoile (ALT) :


UTIL$_i$
BD
ATTEND
RENVOI
RENVOI
ATTEND
CONTROL
ATTEND
RENVOI
IMP

Chaque microprocesseur renvoie et attend les messages vers et du microprocesseur de contrôle qui détermine le cheminement des messages en fonction des graphes de connexion des machines SBD_i. Un ATTEND signale au contrôleur que l'unité correspondante est prête à traiter un message. Le contrôleur relance l'exécution de l'unité en lui transmettant un message qui la concerne. Un RENVOI lui signale que l'unité émet un message sur une patte de sortie identifiée par un numéro. Le contrôleur détermine s'il y a un destinataire prévu par le graphe de connexion de la machine utilisée et s'apprête à lui fournir le message.

Cette solution introduit un goulot d'étranglement dans le système : le contrôleur.

b) Une autre solution consiste à répartir le contrôle. Chaque microprocesseur interprète les primitives de communication ATTEND et RENVOI émises par l'unité qu'il supporte ($UTIL_i$, B.D., IMP).

Cette solution implique que les graphes de connexion des machines SBD_i soient copiés dans chaque microprocesseur du système.

Si les redéfinitions de machines ne sont pas fréquentes la répartition du contrôle est tout à fait envisageable. La vérification de la cohérence des copies de graphes de connexion de machines ne pose pas trop de problème.

5. RESUME

En résumé on peut dire que le modèle de décomposition est un bon outil pour étudier des systèmes construits sur une coopération d'unités spécialisé La mise en oeuvre de ces systèmes sur des grappes de processeurs pose par contre un problème au niveau du contrôle de la communication. Les concepteur de systèmes ont le choix entre maintenir un contrôle centralisé et répartir

ce contrôle. Dans le premier cas la bonne marche du système dépend du bon fonctionnement du processeur qui réalise le contrôleur. Dans le deuxième cas il faut assurer que les copies de graphes de connexion réparties sur chaque processeur sont cohérentes.

BIBLIOGRAPHIE

(ALT) J. ALTABER, V. FRAMMERY, C. GAREYTE, J.P. JEANNERET, P. VANDERSTOK
Contrôle en temps réel avec architecture distribuée.
Journées BIGRE, Novembre 1977.

(ANC) F. ANCEAU
Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques.
Thèse d'Etat, ENSIMAG, Décembre 1974.

(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.

(BRIA1) J. BRIAT, X. ROUSSET DE PINA, J.P. VERJUS
Le projet OURS. Ses principes.
Congrès AFCET, Juin 1976.

(BRIA2) J. BRIAT, J.P. VERJUS
Implication de certaines propriétés d'un noyau de système (MAS) sur un langage d'écriture.
BIGRE, n° 4, Octobre 1976.

(BRIN) P. BRINCH HANSEN
Distributed Processes : a concurrent programming concept.
Computer Science Dept., Université Southern California, 1977.

(CHEV) J.L. CHEVAL, F. CRISTIAN, S. KRAKOWIAK, J. MONTUELLE, J. MOSSIERE
Un système d'aide à l'écriture de systèmes d'exploitation.
Congrès AFCET, Paris, 1976.

(DAR) P. DARONDEAU, S. GUIBOUD-RIBAUD, C. OTRAGE, H. SMIT
Architecture de calculateurs orientés vers les systèmes.
RAIRO, B2, 1974.

(DIJ) E.W. DIJKSTRA
Hierarchical ordering of sequential processes.
Acta Informatica, Vol. 1, 1, 1971.

(ELI) M. ELIE, H. ZIMMERMANN
Transport Protocol. Standard End-to-End Protocol for Heterogeneous Computer Networks.
Bibliothèque Cyclades, SCH 519.2, Mai 1975.

(EST) J. ESTUBLIER
Processus cyclique et appel procédural.
Thèse 3ème Cycle INPG, Grenoble, Avril 1978.

(FELD) J.A. FELDMAN
A programming methodology for distributed computing (among other things).
TR9, Computer Science Dept., Université Rochester, Novembre 1976.

(GRA) J. GRACA MARTINS
Communication implicite entre les processus répartis sur un réseau hétérogène.
Rapport DEA, Grenoble, Septembre 1977.

(GREEN) GREEN Language
Rationale for the design of the GREEN programming language, a language designed in accordance with the Ironman requirements.
Février 1978.

(HOAR) C.A.R. HOARE
Communicating sequential processes.
CACM, Vol. 21, 8, Août 1978.

(JAM) A.J. JAMMEL, H.G. STIEGLER
Structural decomposition and distributed systems.
Centre de Calcul Leibniz, Munich, Novembre 1977.

(LAUE) H.C. LAUER, R.M. NEEDHAM
On the duality of Operating System Structures.
2ème colloque international sur les systèmes d'exploitation,
IRIA, Octobre 1978.

(LAF) P. LAFORGUE
Construction de sous-système utilisant une machine abstraite.
Thèse Docteur-Ingénieur, Grenoble, Février 1975.

(MOS) J. MOSSIERE
Méthode pour l'écriture des systèmes d'exploitation.
Thèse d'Etat USMG, Grenoble, Septembre 1977.

(ORG) E.I. ORGANICK
The MULTICS system : an examination of its structure.
MIT Press, Cambridge, Mass., 1972.