50376
1979
158

N° d'ordre : 242

50376
1979
158

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR INGÉNIEUR

(TRAITEMENT DE L'INFORMATION)

par

Eric DELATTRE

CONTRIBUTION A LA REPARTITION D'UN SYSTEME A STRUCTURE DE DOMAINES SUR UN RESEAU LOCAL DE MICRO-ORDINATEURS FAIBLEMENT COUPLES

BIBLIOTHÈQUE UNIVERSITAIRE
Section de SCIENCES
LILLE

Thèse soutenue le 21 décembre 1979 devant la Commission d'Examen

Membres du Jury
MM. C. CARREZ — Président et Rapporteur
V. CORDONNIER — Examinateur
D. HERMAN — Examinateur
D. LANCIAUX — Examinateur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE

01.02.1976

DOYENS HONORAIRES de l'Ancienne Faculté des Sciences

MM. R. DEFRETIN, H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES des Anciennes Facultés de Droit et Sciences Economiques, des Sciences et des Lettres

M. ARNOULT, Mme BEAUJEU, MM. BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, CORSIN, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, P. GERMAIN, HEIM DE BALSAC, HOCQUETTE, KAMPE DE FERIET, KOUGANOFF, LAMOTTE, LASSERRE, LELONG, Mme LELONG, MM. LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, NORMANT, PEREZ, ROIG, ROSEAU, ROUBINE, ROUELLE, SAVART, WATERLOT, WIEMAN, ZAMANSKI.

PRESIDENTS HONORAIRES DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. R. DEFRETIN, M. PARREAU.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

M. J. LOMBARD.

PROFESSEURS TITULAIRES

M.	BACCHUS Pierre	Astronomie
M.	BEAUFILS Jean-Pierre	Chimie Physique
M.	BECART Maurice	Physique Atomique et Moléculaire
M.	BILLARD Jean	Physique du Solide
M.	BIAYS Pierre	Géographie
M.	BONNEMAN Pierre	Chimie Appliquée
M.	BONNOT Ernest	Biologie Végétale
M.	BONTE Antoine	Géologie Appliquée
M.	BOUGHON Pierre	Algèbre
M.	BOURIQUET Robert	Biologie Végétale
M.	CELET Paul	Géologie Générale
M.	CONSTANT Eugène	Electronique
M.	DECUYPER Marcel	Géométrie
M.	DELATTRE Charles	Géologie Générale
M.	DELHAYE Michel	Chimie Physique
M.	DERCOURT Michel	Géologie Générale
M.	DURCHON Maurice	Biologie Expérimentale
M.	FAURE Robert	Mécanique
M.	FOURET René	Physique du Solide
M.	GABILLARD Robert	Electronique
M.	GLACET Charles	Chimie Organique
M.	GONTIER Gérard	Mécanique
M.	GRUSON Laurent	Algèbre
M.	GUILLAUME Jean	Microbiologie
M.	HEUBEL Joseph	Chimie Minérale
M.	LABLACHE-COMBIER Alain	Chimie Organique
M.	LANSRAUX Guy	Physique Atomique et Moléculaire
M.	LAVEINE Jean-Pierre	Paléontologie
M.	LEBRUN André	Electronique
M.	LEHMANN Daniel	Géométrie

.../...

Mme	LENOBLE Jacqueline	Physique Atomique et Moléculaire
M.	LINDER Robert	Biologie et Physiologie Végétales
M.	LOMBARD Jacques	Sociologie
M.	LOUCHEUX Claude	Chimie Physique
M.	LUCQUIN Michel	Chimie Physique
M.	MAILLET Pierre	Sciences Economiques
M.	MONTARIOL Frédéric	Chimie Appliquée
M.	MONTREUIL Jean	Biochimie
M.	PARREAU Michel	Analyse
M.	POUZET Pierre	Analyse Numérique
M.	PROUVOST Jean	Minéralogie
M.	SALMER Georges	Electronique
M.	SCHILTZ René	Physique Atomique et Moléculaire
Mme	SCHWARTZ Marie-Hélène	Géométrie
M.	SEGUIER Guy	Electrotechnique
M.	TILLIEU Jacques	Physique Théorique
M.	TRIDOT Gabriel	Chimie Appliquée
M.	VIDAL Pierre	Automatique
M.	VIVIER Emile	Biologie Cellulaire
M.	WERTHEIMER Raymond	Physique Atomique et Moléculaire
M.	ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS SANS CHAIRE

M.	BELLET Jean	Physique Atomique et Moléculaire
M.	BODARD Marcel	Biologie Végétale
M.	BOILLET Pierre	Physique Atomique et Moléculaire
M.	BOILLY Bénoni	Biologie Animale
M.	BRIDOUX Michel	Chimie Physique
M.	CAPURON Alfred	Biologie Animale
M.	CORTOIS Jean	Physique Nucléaire et Corpusculaire
M.	DEBOURSE Jean-Pierre	Gestion des entreprises
M.	DEPREZ Gilbert	Physique Théorique
M.	DEVRAINNE Pierre	Chimie Minérale
M.	GOUDMAND Pierre	Chimie Physique
M.	GUILBAULT Pierre	Physiologie Animale
M.	LACOSTE Louis	Biologie Végétale
Mme	LEHMANN Josiane	Analyse
M.	LENTACKER Firmin	Géographie
M.	LOUAGE Francis	Electronique
Mle	MARQUET Simone	Probabilités
M.	MIGEON Michel	Chimie Physique
M.	MONTEL Marc	Physique du Solide
M.	PANET Marius	Electrotechnique
M.	RACZY Ladislas	Electronique
M.	ROUSSEAU Jean-Paul	Physiologie Animale
M.	SLIWA Henri	Chimie Organique

MAITRES DE CONFERENCES (et chargés d'Enseignement)

M.	ADAM Michel	Sciences Economiques
M.	ANTOINE Philippe	Analyse
M.	BART André	Biologie Animale
M.	BEGUIN Paul	Mécanique
M.	BKOUCHE Rudolphe	Algèbre
M.	BONNELLE Jean-Pierre	Chimie
M.	BONNEMAIN Jean-Louis	Biologie Végétale
M.	BOSCQ Denis	Probabilités
M.	BREZINSKI Claude	Analyse Numérique
M.	BRUYELLE Pierre	Géographie

.../...

M. CARREZ Christian	Informatique
M. CORDONNIER Vincent	Informatique
M. COQUERY Jean-Marie	Psycho-Physiologie
Mle DACHARRY Monique	Géographie
M. DEBENEST Jean	Sciences Economiques
M. DEBRABANT Pierre	Géologie Appliquée
M. DE PARIS Jean-Claude	Mathématiques
M. DHAINAUT André	Biologie Animale
M. DELAUNAY Jean-Claude	Sciences Economiques
M. DERIEUX Jean-Claude	Microbiologie
M. DOUKHAN Jean-Claude	Physique du Solide
M. DUBOIS Henri	Physique
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
Me EVRARD Micheline	Chimie Appliquée
M. FONTAINE Jacques-Marie	Electronique
M. FOURNET Bernard	Biochimie
M. FORELICH Daniel	Chimie Physique
M. GAMBLIN André	Géographie
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GUILLAUME Henri	Sciences Economiques
M. HECTOR Joseph	Géométrie
M. JACOB Gérard	Informatique
M. JOURNEL Gérard	Physique Atomique et Moléculaire
Mle KOSMAN Yvette	Géométrie
M. KREMBEL Jean	Biochimie
M. LAURENT François	Automatique
Mle LEGRAND Denise	Algèbre
Mle LEGRAND Solange	Algèbre
M. LEROY Jean-Marie	Chimie Appliquée
M. LEROY Yves	Electronique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique Théorique
M. LOUCHET Pierre	Sciences de l'Education
M. MACKE Bruno	Physique
M. MAHIEU Jean-Marie	Physique Atomique et Moléculaire
Me N'GUYEN VAN CHI Régine	Géographie
M. MAIZIERES Christian	Automatique
M. MALAUSSENA Jean-Louis	Sciences Economiques
M. MESSELYN Jean	Physique Atomique et Moléculaire
M. MONTUELLE Bernard	Biologique Appliquée
M. NICOLE Jacques	Chimie Appliquée
M. PAQUET Jacques	Géologie Générale
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie Physique
M. PERROT Pierre	Chimie Appliquée
M. PERTUZON Emile	Physiologie Animale
M. PONSOLLE Louis	Chimie Physique
M. POVY Lucien	Automatique
M. RICHARD Alain	Biologie
M. ROGALSKI Marc	Analyse
M. ROY Jean-Claude	Psycho-Physiologie
M. SIMON Michel	Sociologie
M. SOMME Jean	Géographie
Mle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. STEEN Jean-Pierre	Informatique

.../...

M.	THERY Pierre	Electronique
M.	TOULOTTE Jean-Marc	Automatique
M.	TREANTON Jean-René	Sociologie
M.	VANDORPE Bernard	Chimie Minérale
M.	VILLETTE Michel	Mécanique
M.	WALLART Francis	Chimie
M.	WERNIER Georges	Informatique
M.	WATERLOT Michel	Géologie Générale
Mme	ZINN-JUSTIN Nicole	Algèbre

Je tiens à remercier

Monsieur le Professeur C. CARREZ qui me fait l'honneur de présider le Jury de cette thèse et qui, par ses encouragements et ses multiples conseils et critiques m'a permis de conclure ce travail,

Monsieur le Professeur V. CORDONNIER, Monsieur D. LANCIAUX, ingénieur de recherche chez CII-HB, et Monsieur D. HERMAN de l'Université de Rennes, qui se sont intéressés à cette étude et qui ont accepté de participer au Jury.

Madame P. CARON ainsi que Madame et Monsieur P. DEBOCK, qui ont assuré avec gentillesse et compétence la réalisation matérielle de ce document.

TABLE DES MATIÈRES

INTRODUCTION

CHAPITRE 1 : LA DÉCOMPOSITION DES PROGRAMMES EN MODULES

1.1 UN EXEMPLE DE CONCEPTION MODULAIRE DES SYSTEMES D'EXPLOITATION
LE PROJET SESAME

1.2 UN EXEMPLE DE LANGAGE SUPPORTANT LES TYPES ABSTRAITS : LE SYSTEME CLU

1.3 METHODOLOGIE DE DECOMPOSITION MODULAIRE

CHAPITRE 2 : UN SYSTÈME À STRUCTURE DE DOMAINES

2.1 UNE STRUCTURE DE DOMAINES

2.1.1 Préliminaires relatifs à la notion de type construit

2.1.2 Notion de domaine de protection

2.1.3 Adressage par capacités

Définition - Réalisation d'un domaine de protection - Intégrité des capacités - Compléments sur l'adressage

2.1.4 Protection des objets construits

Autonomie des objets par rapport aux domaines - Accès à la représentation d'un objet construit

2.2 DOMAINES ET FIABILITE DU LOGICIEL

2.2.1 Limitation de la production d'erreurs

2.2.2 Limitation de la propagation des erreurs

Limitation des interactions entre modules - Conséquences sur les relations entre modules et domaines.

2.3 CONTRAINTES IMPOSEES PAR L'EXECUTION DES DOMAINES

2.4 IMPLEMENTATION DE LA STRUCTURE DE DOMAINES

2.5 DISCUSSION

CHAPITRE 3 : CRÉATION ET DESTRUCTION DES OBJETS

3.1 OBJETS FORTEMENT TYPES

Définition - Conséquences sur la représentation des objets.

3.2 CREATION DES OBJETS

3.2.1 Préliminaires relatifs à la manipulation des capacités

3.2.2 La maquette comme modèle de création des objets

Création des objets primitifs - Interprétation de la maquette d'un objet - Opération de Composition - Création des maquettes

3.2.3 *Mise en place des domaines à partir de l'objet-type*
Dissociation de la création et de l'activation des domaines - Conséquences
3.2.4 *Opérations de création*
3.3 DESTRUCTION DES OBJETS
3.3.1 *Position du problème*
3.3.2 *Localisation des objets perdus*

CHAPITRE 4 : DÉFINITION D'UNE ORGANISATION MATÉRIELLE ADAPTÉE À LA STRUCTURE DE DOMAINES

4.1 Panorama des structures matérielles des systèmes munis d'un adressage par capacités
4.1.1 *Le Plessey 250 : un multi-processeur muni d'un adressage matériel par capacités*
4.1.2 *La réalisation de Spier : la simulation d'une structure de domaines sur un mini-ordinateur*
4.1.3 *HYDRA/Cmmp : l'implantation d'un adressage par capacités sur un multi-processeur doté d'une matrice "crossbar"*
4.1.4 *Cm* : un multi-processeur réparti supportant un adressage par capacités*
4.1.5 *Les travaux de Casey sur la répartition d'une structure de domaines sur un multi-ordinateur*
4.1.6 *Autres propositions*
4.2 Adéquation de l'organisation matérielle du système à la structure de domaines
4.2.1 *Propriétés des multi-processeurs*
4.2.2 *Implémentation d'une structure de domaines sur un réseau d'ordinateurs : étude de faisabilité*
Position du problème : couplage entre processeurs - Banalisation ou spécialisation fonctionnelle des sites - Régulation de la charge dans un système homogène.
4.3 STRUCTURE D'UN SITE
4.3.1 *Organisation de la mémoire*
4.3.2 *Chemins d'adressage*
Ensemble des objets d'un domaine - Espaces logiques "vus" par le processeur du site - Logique de topographie - Choix d'un micro-processeur.

4.3.3 *Structure d'un site*

4.4 Discussion

CHAPITRE 5 : GESTION RÉPARTIE DES OBJETS

5.1 PROPRIETES DES OBJETS

5.1.1 Partage

5.1.2 Mobilité

5.1.3 Catalogage

5.1.4 Ré-entrance

5.1.5 Récursivité

5.2 MECANISMES DE GESTION DES NOMS

Position du problème - Noms spatiaux - Noms globaux - Emploi des noms

5.3 PARTAGE DES OBJETS

Limitation des possibilités de partage - Répartition de l'opération de partage

5.4 CATALOGAGE DES OBJETS

5.4.1 Préliminaires : connexion d'un disque au système

5.4.2 Gestion des noms des objets catalogeables

5.5 MOBILITE DES OBJETS

5.5.1 Répartition de l'opération d'immobilisation

5.5.2 Répartition de l'opération de libération

5.5.3 Restriction de l'emploi des opérations de libération et d'immobilisation

5.6 REPARTITION DES OPERATIONS DE CREATION DES OBJETS

5.6.1 Répartition de l'opération d'interprétation de maquettes

5.6.2 Copie de l'objet-type

5.6.3 Efficacité du processus créateur

5.7 DESTRUCTION DES OBJETS ET REPARTITION

5.7.1 Répartition de l'opération de destruction

5.7.2 Destruction des représentations des objets

5.7.3 Répartition du processus de localisation des objets perdus

5.7.4 Caractérisation de l'ensemble des domaines D(n)

5.8 REMARQUES DIVERSES

5.8.1 Opération de transfert de capacité

5.8.2 Copie des segments de code

5.8.3 *Caractéristiques quantitatives des objets*
Encombrement-mémoire d'un objet - Encombrement-Message d'un objet.

CHAPITRE 6 : GESTION RÉPARTIE DES DOMAINES

6.1 TRANSFERT DE CONTROLE ET SYNCHRONISATION

6.2 OPERATIONS DE TRANSFERT DE CONTROLE

6.2.1 Opération d'activation
Répartition de l'opération d'activation - Cohérence des mécanismes de transfert de contrôle - Composants utiles à l'exécution d'un domaine - Informations guidant le choix de site

6.2.2 Opérations d'attente et de réactivation
Opération d'attente - Opération de réactivation

6.2.3 Opérations de fin d'exécution

6.2.4 Opérations de manipulation des paramètres

6.2.5 Création des objets locaux

6.2.6 Opérations de synchronisation directe de l'exécution de domaines frères

6.3 SYNCHRONISATION PAR SEMAPHORES

6.4 ALLOCATION DES RESSOURCES MEMOIRE ET TEMPS DE CALCUL
Allocation de temps de calcul - Allocation de mémoire - Domaines Allocateurs

6.5 EXEMPLES DE PROGRAMMATION DES APPELS DE DOMAINE

6.5.1 Organisation d'un domaine

6.5.2 Appel procédural

6.5.3 Appel "bifurquer"/"rejoindre"

6.5.4 Synchronisation directe de l'exécution de domaines frères

6.6 REMARQUES

6.6.1 Etats d'un domaine

6.6.2 Appels imbriqués de domaines

Conclusion

Annexe 1 : Preuve de l'identité de l'ensemble des objets perdus à l'ensemble des objets récursifs non marqués

Annexe 2 : Notice technique du micro-processeur Z8000

Annexe 3 : Gestion des objets

Annexe 4 : Cahier des charges du Sous-Système de Communication

Bibliographie

INTRODUCTION

Le projet OMPHALE a pour but l'étude d'une architecture matérielle et logicielle adaptée à la programmation modulaire et structurée. Ce but est motivé par la souci de développer des méthodologies de programmation conduisant à un logiciel lisible, facile à produire et à maintenir.

Ces qualités émergentes, recherchées par la programmeur, dépendent d'une propriété qui intéresse, cette fois, le concepteur de système : la fiabilité (une forte fiabilité ne constitue un objectif explicite consigné dans le cahier des charges d'un système que pour des applications bien particulières utilisant, par exemple, en temps réel, des calculateurs embarqués). Une définition restrictive réduit la fiabilité à la faiblesse de la probabilité de panne d'un système (Wulf [70]). Plus généralement, un système est considéré comme fiable s'il est apte à fournir un niveau minimum de services, conformes à ses spécifications fonctionnelles, en toutes circonstances (Linden [38]). En fait, l'étude de la fiabilité des systèmes d'insère entre deux approches extrêmes. La première, adoptée par l'industrie aérospatiale, refuse toute occurrence de panne, d'où qu'elle provienne. Cette démarche s'avère doublement irréaliste. D'une part, il est impossible de garantir une fiabilité totale du matériel, les composants présentant une probabilité de défaillance non nulle. D'autre part, il est fort difficile et coûteux de traquer et de corriger les dernières fautes entachant la conception du matériel ou l'écriture des programmes. Mieux vaut - et c'est là la seconde approche, développée par les industriels du téléphone - tolérer la présence des pannes inévitables et tenter de limiter la propagation de leurs manifestations phénoménales, les erreurs. Car une erreur est d'autant plus difficile à détecter et à réparer qu'elle se trouve éloignée de sa source.

Par ailleurs, la fiabilité d'un système doit être distinguée de sa correction - de sa conformité à ses spécifications - cette dernière propriété étant une condition nécessaire, mais non suffisante, à l'expression de la première ; en aucun cas, la correction d'un système n'empêche l'apparition de pannes du matériel.

Le système d'exploitation constitue certainement la partie la plus fréquemment utilisée du logiciel supporté par une machine. Aussi la fiabilité globale d'un système dépend fortement de celle de son système d'exploitation. Or, les systèmes d'exploitation sont devenus des programmes volumineux et complexes, dans lesquels la difficulté de conserver une compréhension globale conduit à des erreurs, dont la correction n'a souvent pour résultat que de génèrer de nouvelles erreurs (Lanciaux [34]). La domination de cette complexité peut s'appuyer sur une démarche, certes peu originale, mais efficace, de décomposition en parties. Des techniques de décomposition sont, alors, appliquées durant la phase de conception du logiciel - ce qui incite à mettre à la disposition de l'utilisateur des outils qui facilitent la décomposition des programmes en modules - et même durant son exécution - ce qui nécessite l'écriture d'un logiciel de base capable d'enchaîner l'exécution des modules.

Le chapitre 1 de cette présente étude illustre par des exemples ces premières techniques de décomposition du logiciel, alors que les secondes constituent proprement le sujet de ce travail : il s'agit de définir une architecture matérielle et logicielle supportant l'exécution de modules et limitant la propagation des erreurs, dans le but d'assurer une bonne fiabilité, i.e. une forte tolérance aux pannes.

Cette architecture est fondée sur le concept de domaine. Au départ, les promoteurs de cette notion voyaient en elle un moyen d'établir une protection des entités (appelées objets) composant le système, c'est à dire une possibilité de contrôler la validité des accès aux objets. De plus, l'élaboration de la notion de domaine est liée au développement du moyen matériel de son implantation - l'adressage par capacités ("capability-based addressing") - qui permet de nommer, de partager et de protéger tous les objets du système de manière uniforme. Puis, les travaux menés à l'IRIA (Ferrié [14]) et à la Carnegie Mellon University (Wulf [67]) assurèrent la liaison entre la notion de type, issue des études sur les langages supportant les types abstraits, et celle de domaine. Après avoir formé seulement les unités de protection du système, les domaines devenaient, de plus, des unités d'exécution, résultant de la décomposition du logiciel en modules. Le chapitre 2 détaille les multiples aspects de la notion de domaine, évalue l'influence

de son emploi sur la qualité du logiciel et discute de l'efficacité de son implémentation au moyen d'un adressage par capacités.

Lanciaux ([34]) a fait aboutir la genése du concept de domaine en rendant les domaines autonomes et indépendants de toute structure de données centralisée. Le chapitre 3, en étudiant en particulier les mécanismes de création dynamique des objets et l'articuliation des notions de domaine et de module, et, ensuite, le chapitre 6 examinent ce point.

Les systèmes munis d'un adressage par capacités ont généralement été implantés sur des multi-processeurs dotés d'une mémoire commune partagée. Seul Casey ([4, 5]) a étudié les avantages résultant de l'emploi de réseaux locaux d'ordinateurs faiblement couplés et a effectué des simulations qui montrent la faisabilité de sa proposition. Le chapitre 4 recherche une architecture matérielle adaptée à la mise en place des domaines et adopte la configuration originale introduite par Casey. Mais la question de la faisabilité reste posée, car il semble que Casey ait négligé certaines contraintes portant sur la taille des domaines et provenant des impératifs de protection. Aussi, les résultats de ses simulations ne peuvent pas être transposés au système OMPHALE. D'autre part, la répartition des domaines dans les mémoires locales d'ordinateurs formant un réseau, exige la redéfinition des mécanismes de gestion des objets, habituellement implantés dans la mémoire commune d'un multi-processeur. C'est à quoi s'attachent les chapitres 5 et 6.

Enfin, le cahier des charges du projet OMPHALE comporte des spécifications autres que celles relatives à la qualité du logiciel. D'une part, l'organisation matérielle du système doit être facilement extensible : l'ajout d'un ordinateur ne doit pas entraîner le boulversement du matériel et du logiciel existants. D'autre part, le système doit présenter une bonne disponibilité : la panne d'un sous-ensemble ne doit pas provoquer la mise hors service du système. Et, la réalisation matérielle entend bénéficier des derniers progrès des technologies LSI, en particulier par l'emploi des microprocesseurs les plus modernes. Par ailleurs, il est prévu d'exploiter OMPHALE en temps partagé.

CHAPITRE 1

LA DECOMPOSITION DES PROGRAMMES EN MODULES

De nombreuses techniques utilisent une décomposition en parties pour faciliter la compréhension, la conception et la construction d'ensembles complexes. En informatique, cette démarche a abouti à l'énoncé de méthodes de décomposition des programmes en modules. Les recherches concernant, d'une part, la conception des systèmes d'exploitation, et d'autre part, les langages de programmation supportant des types abstraits, ont tiré des avantages multiples de l'emploi de ces méthodes. La présentation de deux exemples les mettra en évidence.

1.1 UN EXEMPLE DE CONCEPTION MODULAIRE DES SYSTEMES D'EXPLOITATION : LE PROJET SESAME

J. Montuelle ([47]) remarque que la complexité des systèmes d'exploitation réside autant dans le produit lui-même (taille, fonctions à réaliser, ressources à gérer, parallélisme), que dans le processus de sa production (définition du cahier des charges, conception, spécification, écriture, documentation, vérification, entretien, modification). Aussi propose-t-il un logiciel d'aide à la conception, appelé SESAME (M. Lucas et J. Montuelle [43]), qui facilite le processus de production. SESAME fournit les outils suivants :

- Un langage d'écriture de modèles de modules. Un modèle est un texte source comportant des paramètres. Par affectation de valeurs aux paramètres, il est possible d'engendrer un module à partir d'un modèle.

- Un langage de connexion. L'écriture de programmes de connexion permet de rassembler des modules pour former un système.

- Un bibliothécaire. Il est chargé de conserver les textes source et les binaires objet des modules, ainsi que les programmes de connexion représentant des systèmes.

La nature des **modules** utilisés dans SESAME recouvre plusieurs notions : au stade de la conception, un module représente un **objet abstrait**, i.e. un objet défini par un ensemble de propriétés et manipulable par un ensemble de procédures ou interface. La définition des propriétés ne fait intervenir que l'interface, et non la représentation interne du module, et est décrite par des **spécifications**. Au stade de la réalisation, un modèle de module matérialise une collection d'**objets protégés**, constituée d'une structure de données paramétrée et d'une liste des seules procédures autorisées à la manipuler. Le module forme également une unité de compilation et une unité de conservation des programmes : un module peut être compilé et stocké indépendamment des autres modules constituant un système. Enfin, dans un système, le module reste une unité d'exécution et de synchronisation.

Le découpage d'un système d'exploitation en modules similaires à ceux générés par SESAME procure les avantages suivants (M. Lucas [42], T.A. Linden [38]) :

- Lisibilité : La décomposition facilite la compréhension du fonctionnement du système, que ce soit au niveau de son comportement global, les détails de réalisation des fonctions qu'il fournit étant cachés dans les modules (Parnas [53]), ou, inversement, au niveau de l'implémentation d'une de ses fonctions, les spécifications permettant, alors, de reconstituer le contexte d'activation d'un module particulier. La rédaction d'une documentation réellement utile à l'équipe de conception du système, tout comme aux usagers, s'en trouve simplifiée.

- Facilité de production : Grâce aux spécifications, les modules sont produits et testés indépendamment les uns des autres. La complexité des preuves de programme est réduite.

- facilité de maintenance : La programmation modulaire permet de localiser aisément le module dans lequel existe une erreur. Un module peut être modifié, voire remplacé, sans perturbation de l'ensemble du système, à condition que la nouvelle réalisation respecte les spécifications. Les modules existant sont utilisés lors de l'ajout d'une nouvelle fonction au système.

Finalement, la décomposition en modules renforce la **fiabilité** du logiciel.

1.2 UN EXEMPLE DE LANGAGE SUPPORTANT DES TYPES ABSTRAITS : LE SYSTEME CLU

(B. Liskov et al. [39, 40])

En plus des constructions classiques des langages de programmation structurée, telles que SI... ALORS... SINON, TANT QUE ... FAIRE, etc ... le langage CLU fournit deux sortes de modules correspondant à deux sortes d'abstractions : les procédures et les "clusters". Ces entités sont abstraites dans le sens où elles sont utilisées indépendamment de leur réalisation et sans que leur utilisateur n'ait besoin de connaître leur fonctionnement détaillé. Une procédure supporte une opération abstraite. Un cluster implémente un **type de donnée abstrait**, lui-même défini comme un ensemble d'objets manipulables par une liste d'opérations. Par exemple, le type de donnée abstrait "ensemble d'entiers", appelé intset, est représenté par le cluster :

```
(1)   intset = CLUSTER IS create, insert, remove, has, equal, similar, copy ;

(2)   REP = ARRAY [INT] ;

(3)         create = OPER ( ) RETURNS (CVT) ;
              :
              :
            END create ;

(4)         insert = OPER (s : CVT, i : INT) ;
              :
              :
            END insert ;

(5)         search = OPER (s : REP, i : INT) RETURNS (CVT) ;
              :
              :
            END search ;
              :
              :
      END intset ;
```

La ligne (1) décrit l'interface du cluster. L'interface est formée du nom du cluster et de la liste des noms des opérations constituant le type. La ligne (2) décrit la représentation des objets du type : ici, l'ensemble d'entiers est représenté par un tableau d'entiers. REP constitue un type plus primitif que intset et il peut être utilisé comme tel par les opérations à l'intérieur du cluster. De plus, une opération interne - une opération non accessible à l'extérieur du cluster, comme search, lignes (1) et (5) - admet REP comme type de ses paramètres, voir la variable s ligne (5). Le type intset est utilisé par un appel d'opération, noté

intset $ insert (s, i)

pour l'opération insert, par exemple. Par ailleurs, le type REP est inaccessible à l'extérieur du cluster. Ainsi, l'utilisateur d'un type abstrait n'a pas connaissance de la représentation en mémoire des objets du type, et il n'a accès à ces objets que par les opérations du type.

Le système CLU est doté d'une bibliothéque qui contient, pour chaque abstraction, une "unité de description", formée des **spécifications** concernant les types des données passées aux opérations réalisant une même abstraction. Une abstraction peut être ajoutée à la bibliothèque et utilisée même si aucune représentation n'est définie. Un module est compilé individuellement. La coïncidence des types de ses paramètres et des spécifications contenues dans l'unité de description est testée et la validité des appels d'opérations d'autres abstractions est contrôlée. La sélection d'un module particulier pour représenter une abstraction peut intervenir après la compilation.

Les programmes sont développés en répétant la décomposition d'un problème en sous-problèmes, chacun d'entre eux étant résolu par la définition d'abstractions et la rédaction de modules. Chaque module réalise une **machine abstraite** dotée des seules abstractions nécessaires à la résolution du problème considéré. Les avantages d'une telle méthode de décomposition, appuyée sur les facilités fournies par la bibliothèque, se révèlent fort similaires à ceux procurés par SESAME : lisibilité, facilité de maintenance et de production du logiciel. Mais, de plus, la décomposition en termes d'abstractions montre en quoi les phases de test et de preuve de programme sont simplifiées : le jeu des spécifications portant sur les types des données passées en paramètres permet de reconstituer le contexte d'activation d'un module ; les modules étant supposés corrects, un programme est correct dans la mesure où, à chaque niveau de la décomposition, ils vérifient les spécifications de type des abstractions. B. Liskov envisage de compléter l'unité de description d'une abstraction par des spécifications formelles décrivent le comportement d'un module (un cluster ressemblerait alors fortement à la "form" employée par Alphard - Wulf et al [68]). Par ailleurs, Linden ([38]) montre que les types abstraits facilitent la détection et la réparation des erreurs ; les opérations de traitement des conditions d'erreur font partie du type. Enfin, les possibilités de développement par étapes des programmes sont ici mises à la disposition des utilisateurs et ne sont plus réservées aux seuls concepteurs du système. En plus de la fiabilité du logiciel, les types abstraits renforcent son extensibilité.

Remarque : La terminologie relative aux systèmes d'exploitation remplace les termes "type abstrait" par "type construit". Cette dernière expression sera employée dans toute la suite.

L'emploi d'une méthodologie de conception modulaire, ainsi que l'utilisation de types construits procurent des avantages complémentaires quant à la fiabilité et l'extensibilité du logiciel. Cependant, la mise en oeuvre de méthodes de décomposition différentes donnent des résultats divers sur le plan de la qualité et de l'efficacité du logiciel produit. Ce point mérite d'être développé.

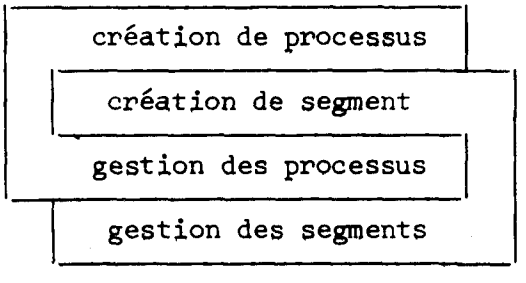
1.3 METHODOLOGIE DE DECOMPOSITION MODULAIRE

Généralement, l'analyse d'un problème complexe conduit à le subdiviser en sous-problèmes, afin d'en faciliter la résolution. Cette forme de modularisation "horizontale" (Linden [38]) est insuffisante car les sous-problèmes résultant du découpage se révèlent souvent de taille trop importante pour bénéficier des avantages décrits ci-dessus. De plus, les questions de partage d'information ne sont pas prises en compte. Enfin, la solution d'un sous-problème peut utiliser des objets de niveaux d'abstraction fort différents, ce qui nuit à la lisibilité. Aussi, il est nécessaire d'employer conjointement une modularisation "verticale" des programmes. Cette approche consiste à concevoir le logiciel par niveaux successifs. Chaque niveau fournit une **machine abstraite** correspondant fortement aux besoins de l'utilisateur. Les fonctions abstraites procurées par la machine d'un niveau permettent de construire une machine abstraite à un niveau supérieur.

De nombreux systèmes sont conçus selon de telles **méthodes hiérarchiques**. Souvent, la hiérarchie des fonctions réalisée pour les machines abstraites est conservée sous une forme ou une autre à l'exécution : dans le système THE (Dijkstra [10]), il existe une hiérarchie de "processus séquentiels", chaque processus exécutant une fonction d'un niveau donné ; Brinch Hansen complète ceci, d'abord en confiant aux processus la responsabilité d'allouer les ressources ([20]), puis en transmettant cette charge à des "moniteurs", qui permettent de synchroniser la progression des processus ([21]) ; le concept introduit par Brinch Hansen est ensuite généralisé par Hoare ([24]). Cependant

cette démarche est critiquée pour sa rigidité (Wulf et al. [67]) et pour son inefficacité (Habermann,et al. [18]). Il semblerait qu'il soit préférable de **réserver la décomposition verticale au stade de la conception et d'étudier séparément le passage à l'implémentation.** Ainsi, Neuman et ses collègues spécifient, écrivent et vérifient complétement un système comportant 13 niveaux de machines abstraites, puis relient ensuite le niveau le plus bas au matériel disponible ([52]).

Par ailleurs, Parnas ([53]) propose un critère de décomposition qui parait plus opérationnel : **un module est composé d'une structure de données et d'un ensemble de fonctions qui partagent la connaissance d'une décision de conception particulière,** la décision étant cachée aux utilisateurs du module. Dès lors, un niveau ne correspond plus nécessairement à un module : par exemple, les modules réalisant la gestion de la mémoire et la gestion des processus sont entrelacés vis à vis de la hiérarchie des fonctions qu'ils réalisent (Habermann et al [18]).



Les deux parties du gérant de la mémoire forment le même module parce qu'elles partagent la connaissance des structures d'implémentation de la mémoire virtuelle (tables de segments, descripteurs, etc ...)

Plusieurs modules implémentent efficacement un seul niveau, mais un module particulier peut traverser plusieurs niveaux. Il n'existe plus de relation évidente entre décomposition modulaire et hiérarchie fonctionnelle (Parnas [54]). Les deux méthodes interviennent à des étapes différentes de la conception du logiciel et il parait souhaitable de les employer toutes deux.

La suite de l'étude repose sur trois hypothèses :

- Les phases de conception puis d'écriture d'un logiciel donnent une collection de modules.

- Le processus de production du logiciel possède un bon degré de fiabilité et l'extensibilité du logiciel produit est garantie.

- Un module implémente un type construit. Les objets sont regroupés en classes d'équivalence d'objets du même type, i.e. d'objets manipulables par les mêmes opérations. Il n'existe pas de moyen d'accès à un objet d'un type donné autre que l'utilisation d'une opération du type.

Ce présent chapitre avait pour but de montrer que ces hypothèses sont réalisables. Mais la suite ne préjuge pas des choix faits lors de la production du logiciel tels que l'emploi ou non d'une méthode hiérarchisée, la sélection d'un critère de décomposition, l'utilisation ou non d'utilitaires comme un bibliothécaire, etc ...

Les autres chapitres traitent de l'exécution des modules, de son efficacité, de son influence sur les avantages issus de la phase de production.

CHAPITRE 2

UN SYSTEME A STRUCTURE DE DOMAINES

Généralement, un système est vu comme un ensemble d'objets. Le système OMPHALE structure les objets qui le constituent en domaines. Un domaine forme, à la fois, une unité de protection et une unité d'exécution. En matière de systèmes d'exploitation, la protection est le terme générique désignant l'action des mécanismes cablés et programmés qui ont pour rôle de contrôler l'accès aux objets (Ferrie [14]).

Ce chapitre traite de la notion de domaine. Il examine les moyens de son implémentation, évalue l'influence de son emploi sur la qualité du logiciel et discute de sa nécessité et de son efficacité.

2.1 UNE STRUCTURE DE DOMAINES

La protection s'étendant à tous les objets du système, y compris les objets d'un type construit, il est utile de détailler cette première notion avant d'introduire le concept de domaine.

2.1.1 *Préliminaires relatifs à la notion de type construit*

- Chaque type est désigné par un nom qui l'identifie sans ambiguité parmi toutes les entités du système. Ainsi l'expression "le type T" signifie, en fait, "le type de nom T".

- La notion de type réalise une partition de l'ensemble des objets du système en classes d'équivalence d'objets du même type. Deux objets sont du même type si ils sont manipulés par les mêmes opérations. Un **mécanisme de construction de type** permet de bâtir un nouveau type T, appelé **type construit**, à partir des types T_1 T_2, ..., T_n existants. Les opérations du type T sont définies par logiciel et emploient les opérations des types T_1, T_2, ..., T_n. Un objet d'un type construit est qualifié lui-même d'**objet construit**.

- Des **types**, dits **primitifs**, préexistent à toute construction de type et permettent d'amorcer le processus d'extension. Les opérations de ces types sont fournies par le matériel ou le logiciel de base du système. Un objet d'un type primitif est dit primitif.

- Le système OMPHALE emploie trois classes d'objets primitifs, dont les éléments sont appelés segments, C-listes et maquettes, et dont les noms de type sont respectivement SEG, CL et MAQ. Un **segment** est un ensemble de valeurs élémentaires. Une **C-liste** participe à l'adressage des objets dans le système. Une **maquette** est un modèle d'objet construit utilisé lors de la création de l'objet.

2.1.2 *Notion de domaine de protection*

Certains objets du système, appelés **agents** du système sont capables d'activer des opérations qui portent sur d'autres objets. La protection repose sur la règle suivante : un agent ne peut déclencher une opération sur un objet que s'il y est explicitement autorisé par la possession d'un **droit** relatif au type de l'opération et à cet objet. A un instant donné, l'état de la protection dans le système peut être représenté par une **matrice de protection** : chaque objet du système occupe une colonne de la matrice, chaque agent, une ligne. A l'intersection d'une ligne et d'une colonne figurent les droits (éventuellement inexistants) d'un agent donné sur un objet donné. Ainsi, par exemple, dans la matrice figurant ci-dessous l'utilisateur A posséde le droit de lire et d'écrire dans le fichier X :

Objets / Agents	Editeur	Fichier X	Fichier Y	Catalogue	....
Utilisateur A	Appel	Lecture Ecriture	Lecture	-	
Editeur	-	Lecture Ecriture	-	Lecture	
⋮					

A un instant donné, un agent exécute des opérations dans un **domaine de protection** ; il n'a accès qu'à une liste limitée d'objets sur lesquels il possède des droits. Un domaine de protection est représenté par une ligne dans la matrice de proctection. Dans l'exemple ci-dessus, l'utilisateur A évolue dans le domaine de protection formé de l'éditeur et des fichiers X et Y. En cours d'exécution les droits d'un agent sur les objets du système se modifient, la matrice de protection se transforme. Ainsi, un calcul se traduit par l'exécution d'opérations dans des domaines de protection successifs distincts.

Intuitivement, le passage d'un domaine à un autre étant contrôlé, la protection sera d'autant mieux assurée que les domaines seront "petits", c'est à dire qu'ils renfermeront peu d'objets. Ainsi, un agent ne possédera pas plus de droits sur les objets que ceux nécessaires pour le déroulement de sa tache immédiate. C'est le **principe du moindre privilège** (le paragraphe 2.2 fait le bilan des avantages attendus du respect de ce principe).

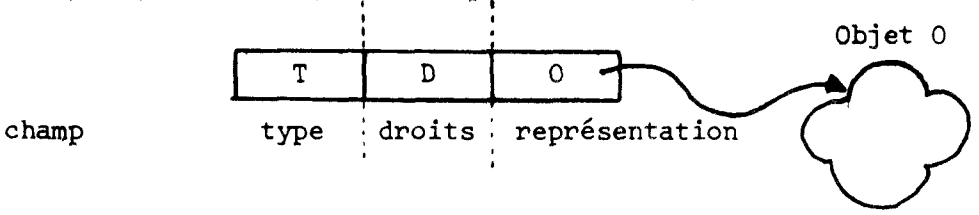
Plusieurs méthodes permettent la réalisation des domaines de protection. Le système OMPHALE, disposant d'un adressage par capacités, inclue le contrôle des droits dans les mécanismes d'adressage des objets et fait du domaine de protection une unité d'exécution.

2.1.3 Adressage par capacités (*Fabry* [J3])

* *Définition*

Tout objet posséde un nom, noté O et appelé **nom unique** de l'objet, qui le distingue de tous les objets du système (le nom d'un type - voir § 2.1.1 - est aussi un nom unique). Le nom unique constitue le début de la chaîne d'accès qui méne à la représentation en mémoire de l'objet.

Un agent A manipule un objet O par l'intermédiaire d'une **capacité** ("capability") pour cet objet. La capacité est composée de trois champs :



- Le nom T du type de l'objet repéré occupe le champ type.
- Les droits de l'agent A sur O sont codés dans le champ droit.
- Le champ représentation renferme le nom unique O.

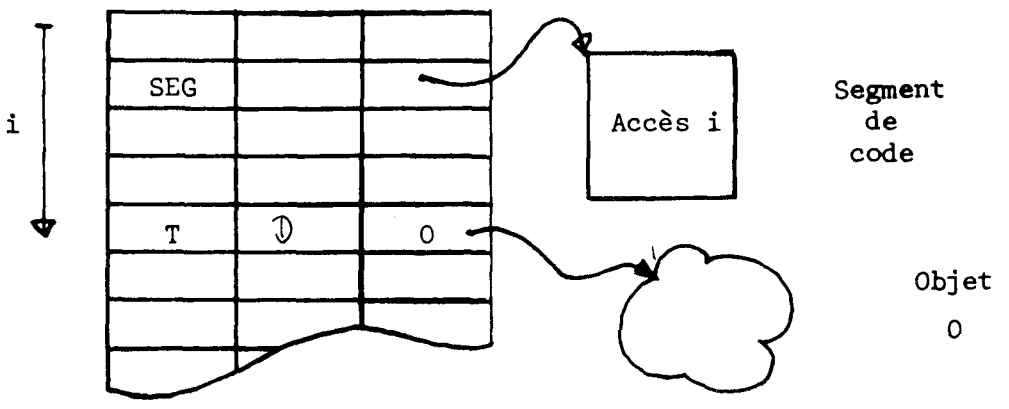
L'accès à l'objet n'est possible

- que par l'intermédiaire de la capacité,
- que si les droits sont suffisants pour autoriser l'accès demandé.

Remarque : Le champ droit peut contenir une clé qui est associée aussi à l'objet. A chaque accès la coïncidence des clés est testée. Des contrôles d'accès plus complexes sont imaginables. De plus, destruction de la clé associée à un objet permet de révoquer les droits des agents sur les objets (Kieburtz et Silberschatz [30]).

* *Réalisation d'un domaine de protection*

Un domaine de protection est alors représenté par une **liste des capacités**, ou **C-liste**, pour les objets qui le composent. L'index i d'une capacité dans cette liste constitue le **nom local** de l'objet repéré dans le domaine considéré. Par ailleurs, une opération est représentée dans le système par un segment de code (ensemble ordonné de valeurs élémentaires exécutables) qui désigne les objets par des noms locaux dans le domaine dans lequel est exécutée l'opération, comme le montre la figure ci-après :



C-liste représentant le domaine

Le matériel et le logiciel de base sont conçus de manière à ce qu'un processeur exécute une de ces listes à la fois. Le mécanisme de contrôle de l'accès à un objet à partir de sa capacité étant réalisé, le principe du moindre privilège est, ainsi, satisfait par une limitation de l'espace d'adressage aux seuls objets composant un domaine de protection. Le domaine devient alors l'unité d'exécution du système et un calcul se traduit par une succession d'exécutions de domaines distincts. Il est, alors, nécessaire de contrôler les changements de domaine. Ceci est réalisé en faisant du domaine un objet du système, de type DOM, dont l'emploi est contrôlé par les mécanismes de protection.

* *Intégrité des capacités*

Cependant, pour que la protection soit effective, il faut assurer l'intégrité des capacités. Aussi, il n'existe aucun moyen de créer de toutes pièces une capacité ; la création d'une capacité est effectuée par le logiciel de base au moment de la création de l'objet qu'elle repère. Et il est impossible d'altérer arbitrairement l'un des champs d'une capacité : les modifications nécessaires sont elles aussi, prises en charge par le logiciel de base. Un agent possède, toutefois, certains degrès de liberté relatifs à la restriction des droits figurant dans les capacités, à la copie des capacités et à leur transmission à des domaines. Les capacités sont elles-mêmes protégées pour éviter leur altération par un programme malveillant (sur la manière dont est réalisée cette protection, voir § 4.3.1).

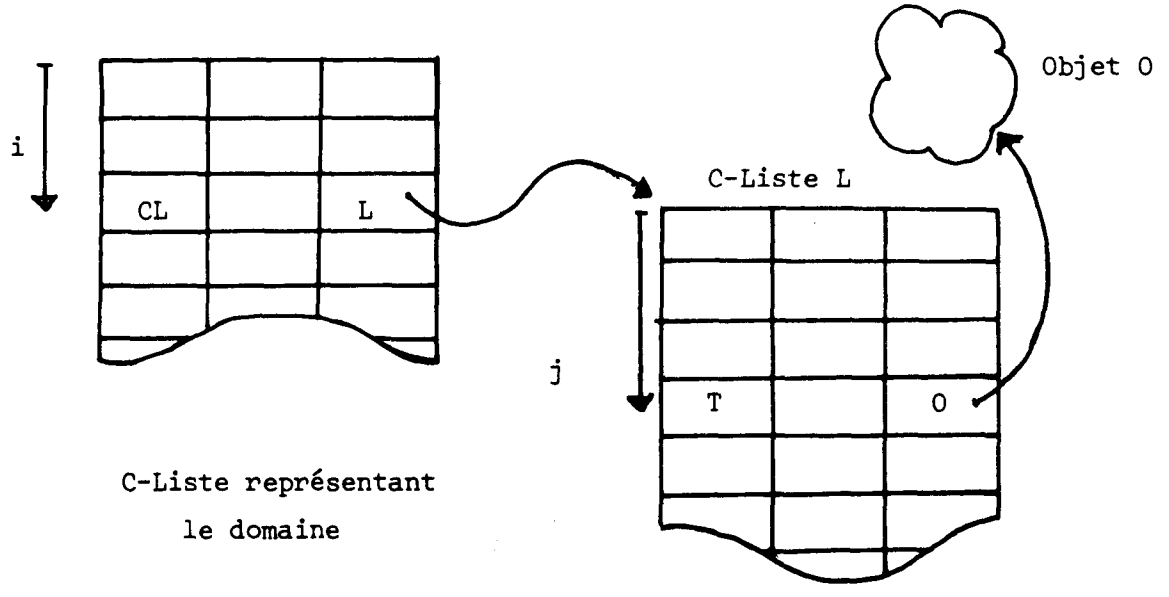
Mais il faut aussi envisager le cas d'une panne conduisant à la modification abusive d'une capacité par le système. Un tel incident peut se traduire par une augmentation illégale des droits d'un agent sur les objets et, donc, violer la règle de protection. Aussi convient-il de détecter ce genre d'erreur. Pour ce faire, Linden ([38]) propose d'insérer des codes autocorrecteurs dans les capacités, et Denning ([9]), en plus du mécanisme de clés cité à la remarque précédente, suggère d'utiliser des noms uniques séparés par des distances de Hamming élevées, afin de rendre peu probable la transformation erronée d'un nom en un autre.

Remarque : Il n'est pas prévu de doter OMPHALE de l'un de ces mécanismes. Les capacités résident, soit en mémoire primaire, où la probabilité de

modification accidentelle est faible, soit dans le Sous-Système de Communication, où une altération est détectée par les contrôles d'erreur des protocoles de transmission de messages.

* *Compléments sur l'adressage*

Les C-listes ne servent pas seulement à représenter des domaines. Elles sont aussi utilisées comme structures de regroupement des références à des objets. Une C-liste peut contenir une capacité pour une autre C-liste et la C-liste représentant un domaine devient la racine d'une arborescence de C-listes. Pour des raisons qui seront exposées au paragraphe 4.3.2 le système **OMPHALE n'accepte que deux niveaux d'arborescence** : un objet O peut être désigné par deux index, le premier, i, étant l'index d'une C-liste L et le second, j, l'index d'une capacité pour l'objet dans la C-liste L, selon le schéma suivant :



Dans ce cas, le nom local de l'objet O dans le domaine est noté i.j. Les mécanismes d'adressage contrôlent que le champ type de la capacité désignée par i renferme le nom CL.

Remarque : L'expression "une capacité de type T" sera employée au lieu de "une capacité dont le champ type renferme le nom T".

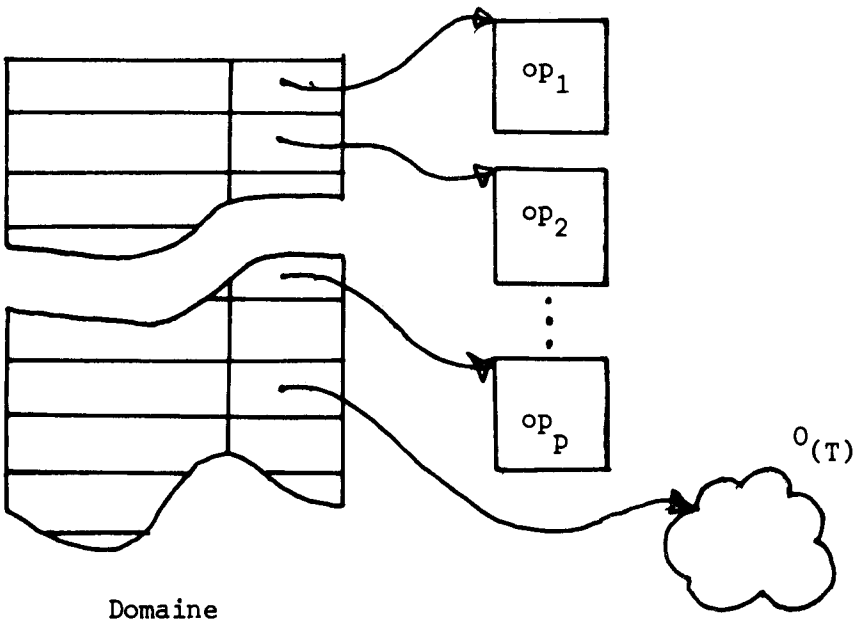
Si la présence dans une capacité des champs droits et représentation découle directement de la définition donnée pour un domaine, il n'en est pas de même pour le champ type. Par ailleurs, le mécanisme de contrôle des accès à un objet à partir d'une capacité n'a pas été examiné. Le paragraphe suivant aborde ces deux questions.

2.1.4 Protection des objets construits

* *Autonomie des objets par rapport aux domaines*

Tout d'abord, il faut assurer qu'un objet construit $O_{(T)}$ - un objet O du type construit T - n'est manipulé que par les opérations op_1, op_2, ..., op_p du type T. Une première approche consiste à enfermer les opérations du type et l'objet dans un seul domaine et à associer à chaque opération un point d'entrée du domaine, comme représenté dans la figure ci-dessous :

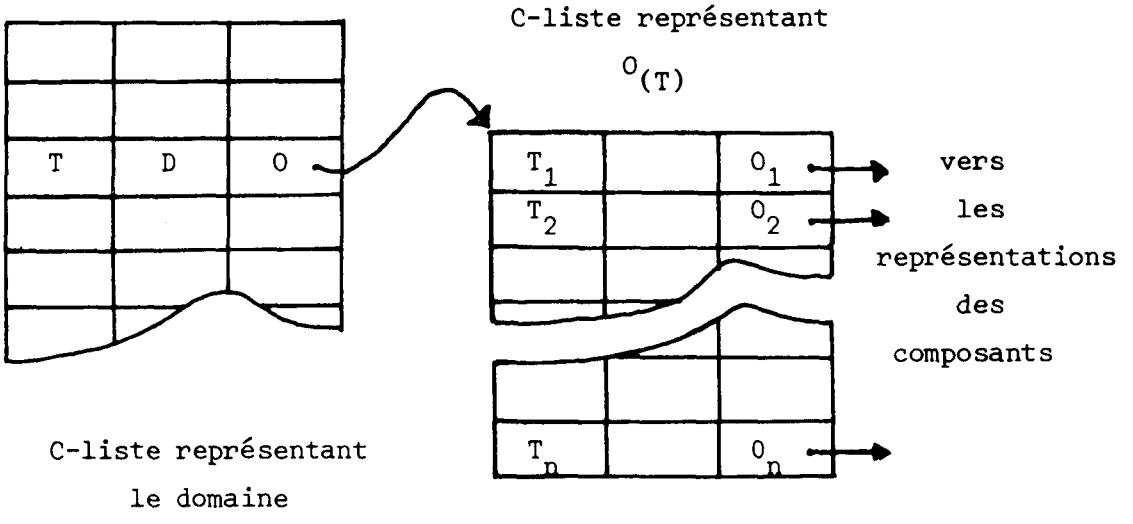
Dans ce cas les capacités ne comportent pas de champ type.



Domaine

Cette méthode, utilisée dans le System 250 de Plessey (England [11]) est critiquée par Ferrié ([14]), parce qu'elle ne permet d'appliquer des opérateurs polyadiques qu'à des configurations fixes d'opérateurs. Il est préférable de séparer les opérations op_1, op_2, ..., op_p de l'objet O et de transmettre une capacité pour l'objet O en paramètre à l'une des opérations quand nécessaire. Il faut alors vérifier que l'objet est bien du type attendu. Le nom du type figurant dans la capacité permet ce contrôle.

Ainsi, un **objet construit $O_{(T)}$ possède une existence autonome par rapport aux opérations qui le manipulent.** Tout objet étant repéré par une capacité, $O_{(T)}$ est représenté par une liste de capacités pour des objets O_1, O_2, ..., O_n de type respectif T_1, T_2, ..., T_n selon le schéma ci-après. O_1, O_2, ..., O_n constituent les **composants** de l'objet O.



Dans ce cas, l'objet $O_{(T)}$ est partagé entre les domaines successifs qui l'utilisent. Ce partage est réalisé par le transfert d'une capacité pour l'objet d'un domaine à un autre. Mais, plusieurs domaines exécutant des calculs différents peuvent aussi **partager simultanément** un objet : alors, leur C-liste renferme chacune une copie d'une capacité pour l'objet. Ainsi un adressage par capacités facilite le partage des objets.

Remarque : Dans la suite, le terme "partage" est réservé au seul cas du partage simultané.

* ***Accès à la représentation d'un objet construit***

Si, par l'intermédiaire d'une capacité pour $O_{(T)}$, le contrôle du type de l'objet, quand il est passé en paramètre à une opération, est facilement effectué, par contre la capacité ne donne pas accès à la C-liste des composants de $O_{(T)}$, ce qui interdit leur manipulation. Pour que les composants soient accessibles, il faut que la C-liste qui les contient soit repérée par

une capacité de type CL (voir "compléments sur l'adressage", § 2.1.3). Une **opération de décomposition** réalise à la fois :

- le contrôle du type de l'objet $0_{(T)}$
- le contrôle des droits figurant dans la capacité pour $0_{(T)}$,
- la fourniture d'une capacité de type CL pour la représentation de l'objet.

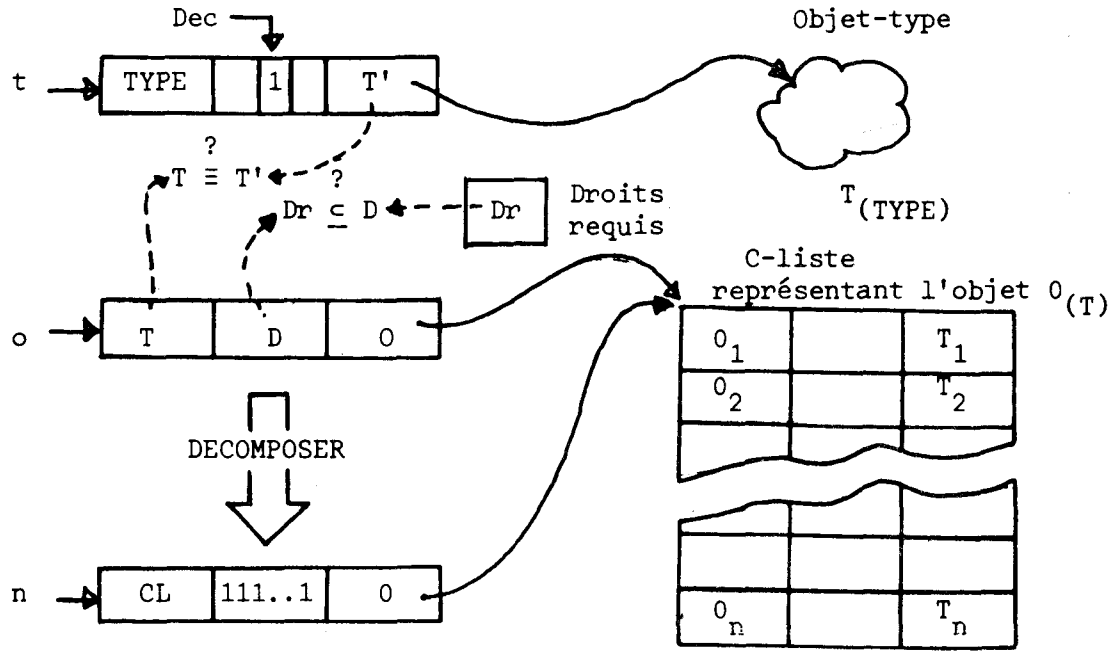
Cette opération participe à la définition d'un type construit qui n'est autre que la classe des modules-des types. Ainsi, le module - le type - est représenté par un objet du système (ce qui est justifié au paragraphe 2.2.2). Dans la suite, le module est appelé **objet-type** ; la classe des objets-type est nommée TYPE ; un objet-type est nommé par le nom T du type qu'il représente.

Soient, dans un domaine, t le nom local d'un objet-type,
o le nom local d'un objet construit,
n un autre nom local
Dr un ensemble de droits
Dec le droit associé à l'activation de l'opération de décomposition.

L'opération de décomposition s'écrit :

```
n <= DECOMPOSER (o, t, Dr)
```

et son action est schématisée ci-dessous (Lanciaux [34]) :



Une capacité de type CL résulte de la décomposition. Les droits qui y figurent sont maximaux (en première approximation ; voir § 3.2.3).

Le logiciel de base est conçu de manière telle que seules les opérations du type T accèdent à une capacité pour l'objet-type T munie du droit de décomposer. Ainsi la décomposition n'est entreprise que par les opérations du type T. Par contre l'agent utilisateur de l'opération ne la voit que comme une abstraction : il n'a pas accès à son code et ne connait pas son fonctionnement interne. Tout ceci explique, d'une part, que les droits requis puissent figurer en paramètre dans l'opération DECOMPOSER, et que, d'autre part, les contrôles des droits sur l'objet paramètre et de la coïncidence du type de l'opération et du type de l'opérande, soient effectués à l'initiative de l'opération elle-même. De plus il faut remarquer que dans les champs droits D et Dr, un bit ne correspond pas nécessairement au droit d'activer une opération du type. Le codage des droits est le résultat d'une convention de programmation dont les opérations du type partagent la connaissance. Un même bit du champ droit peut regrouper les droits d'activation de plusieurs opérations. Cependant, le codage adopté doit vérifier la relation d'ordre pour l'inclusion des champs.

Remarque : Au cas où un domaine manipule simultanément un objet de type T_1 et un objet de type T_2, il doit posséder une capacité pour l'objet-type T_1 et une capacité pour l'objet-type T_2, nunies toutes deux du droit de décomposer. Les objets dont les types sont définis par récursivité croisée - T_1 est construit sur T_2 et T_2 est construit sur T_1 - sont accédés par des domaines entrant dans cette catégorie.

En conclusion, un adressage par capacités assure la protection et le partage de tous les objets du système, quel que soit leur niveau d'abstraction.

2.2 DOMAINES ET FIABILITE DU LOGICIEL

L'"encapsulation" des objets dans des domaines de protection et le respect du principe du moindre privilège renforcent la fiabilité du logiciel en limitant, d'une part, la production des erreurs, et d'autre part, leur propagation.

2.2.1 *Limitation de la production d'erreurs*

Un domaine constitue un **"environnement fermé"**, par opposition à un **environnement ouvert** (Denning [9]) : chaque agent du système ne posséde que les droits - les capacités - qui lui ont été explicitement accordés. Par contre, dans un environnement ouvert un agent n'est privé que des droits qui lui ont été explicitement refusés, i.e. que ce sont les restrictions sur les actions des agents qui doivent être spécifiées. Aussi, un environnement fermé limite la production des erreurs. D'une part, la production d'**erreurs de conception** est restreinte : l'oubli d'un droit se traduit par l'absence d'une capacité, mais aucune action illégale ne peut être entreprise ; par contre, dans un environnement ouvert, l'oubli d'une restriction peut déclencher une opération interdite - d'où erreur. D'autre part, une **panne** du matériel ou du logiciel peut causer une erreur quand il y a destruction d'une restriction, mais non quand il y a destruction d'un droit (dans la mesure où l'intégrité des capacités est assurée : voir § 2.1.3). En général, dans un environnement ouvert toute erreur ou omission tend à augmenter les droits d'un agent, tandis que dans un environnement fermé ces droits diminuent.

2.2.2 *Limitation de la propagation des erreurs*

* *Limitation des interactions entre modules*

Le logiciel est décomposé en modules, chaque module regroupant les opérations d'un type. Pour conserver la fiabilité du logiciel lors de l'exécution, il est nécessaire de limiter les interactions entre modules (Linden [38]). Il faut éviter que le fonctionnement défectueux d'un module - et il est impossible de garantir qu'un module est exempt d'erreurs, ou que le matériel ne tombera jamais en panne - entraîne des erreurs dans les modules voisins et, progressivement, un comportement imprévisible de tout le système. Aussi est-il nécessaire de limiter la propagation des erreurs entre les modules.

Le contrôle des interactions ne peut pas être basé sur les fonctions réalisées par le module, car, dans ce cas, la panne d'un module conduit à des modifications des interactions avec les modules voisins. En particulier, les méthodes de "programmation défensive", qui consistent à insérer des tests dans les opérations du module, manquent de généralité (en plus de la complexité et des délais d'exécution qu'elles entrainent). Il parait préférable

de protéger le module lui-même, i.e. de contrôler l'emploi des opérations qui le composent et l'accès aux objets qui sont passés en paramètres. Cette protection est indépendante du découpage fonctionnel en modules. Cela n'implique pas que toutes les erreurs sont rapidement détectées, mais que, quand elles ne le sont pas, le système continue à se comporter globalement comme prévu. Tel est le sens à donner à l'expression "la protection limite la propagation des erreurs".

* *Conséquences sur les relations entre modules et domaines*

La nécessité de protéger les modules explique la présence des objets-type dans le système. L'unité de protection étant le domaine il est possible, soit d'inclure l'objet-type dans un domaine et, donc, d'enfermer toutes les opérations d'un type dans un même domaine, soit de réaliser chaque opération d'un type dans un domaine différent. La deuxième solution est préférable, pour deux raisons :

- l'observation du principe du moindre privilège conduit à minimiser le nombre d'objets composant un domaine.

- La structure matérielle d'OMPHALE impose de réduire autant que possible la taille des domaines (il s'agit, cette fois, de leur encombrement en mémoire).

La limitation des interactions entre modules est alors réalisée, directement, par le contrôle de la mise en place des domaines à partir des objets-type (voir § 3.2.3), ou, indirectement, par les contrôles d'activation des domaines ou le test des paramètres reçus.

L'observation du principe du moindre privilège pousse à réduire au maximum la taille des domaines. Cependant, puisqu'il réalise une opération d'un type, un domaine doit contenir un minimum d'objets, ne serait-ce pour que le calcul puisse se dérouler correctement. Le paragraphe suivant s'attache à caractériser les objets qui composent un domaine.

2.3 CONTRAINTES IMPOSEES PAR L'EXECUTION DES DOMAINES

Dans la plupart des systèmes, une opération est représentée par une **procédure** (l'objet procédure n'existe pas dans OMPHALE - voir § 3.2.3 -, mais cela ne remet pas en cause les résultats présentés ici). La procédure comprend un segment de code et les objets qui lui sont associés en permanence - ou objets permanents -. D'autres objets sont nécessaires à l'exécution d'une procédure (CROCUS [7]) : les objets locaux qui sont créés à chaque activation de la procédure ; les objets passés en paramètres ; et les objets rémanents, qui sont réutilisés à chaque activation de la procédure. Ces objets constituent l'**environnement** d'exécution de la procédure. ils sont strictement nécessaires à son exécution. Aussi, dans la suite de cette étude, **un domaine** est confondu avec un environnement et **est représenté par la C-liste des objets composant l'environnement d'exécution d'une opération d'un type.**

Les machines disponibles actuellement sont incapables d'exécuter des domaines. Aussi, un noyau *réalise l'interface entre la structure de domaine et la structure matérielle du système.*

2.4 IMPLEMENTATION DE LA STRUCTURE DE DOMAINES

Le noyau comprend le matériel et le locigiel de base. Il peut être vu, tout d'abord, comme un noyau de protection, dans le sens où il fournit les opérations des trois types primitifs, ainsi que les mécanismes associés, tels l'adressage par capacités ou les programmes de gestion des noms uniques. Mais, ensuite, des considérations d'efficacité et de fiabilité relatives à la structure matérielle d'OMPHALE obligent le noyau (ou plutôt, les noyaux, chaque site en possédant une copie) à prendre en charge d'autres fonctions. Ainsi, l'absence de mémoire commune et un parti pris de décentralisation, consuisent à confier la gestion de la mémoire, des processeurs et des périphériques d'un site au noyau de ce site. Afin d'éviter toute centralisation préjudiciable à la fiabilité, la gestion des interfaces au SSC est effectuée, elle aussi, par le noyau de chaque site. Enfin, l'inefficacité qui résulterait des transferts répétés entre les sites de programmes fortement utilisés, incite à munir le noyau des opérations de quelques types construits. Ces types et les types primitifs constituent les **types nucléaires.**

L'utilisation d'un noyau résulte de l'application d'un principe énoncé par Wulf et al. ([67]) : le système ne doit fournir que les mécanismes de base nécessaires à la réalisation efficace et fiable des besoins très diversifiés des usagers du système. Par exemple, au lieu de fournir un certain nombre arbitraire de types, le noyau d'OMPHALE procure à l'usager un mécanisme de construction de type. L'usager conçoit les types qui satisfont directement ses besoins. De plus, il bénéficie de la protection, qui est assurée pour tous les objets construits. Par ailleurs, comme un système universel ne peut pas traiter efficacement tous les problèmes qui lui sont soumis, et que, d'autre part, l'écriture complète d'un système d'exploitation dédié à chaque application est économiquement irréaliste (Habermann et al. [18]), le noyau établit un compromis entre l'efficacité et l'adaptation aux besoins des usagers. La distinction entre programmes d'application et système d'exploitation est, donc, supprimée. En conséquence, l'usager dispose de facilités autrefois réservées au programmeur système. Et plusieurs systèmes d'exploitation peuvent être utilisés simultanément et remplacés dynamiquement (Brinch Hansen [20]). Enfin, la faible taille du noyau permet de prouver que son fonctionnement est correct. Le noyau doit bénéficier du haut degré de fiabilité nécessaire à une protection effective.

2.5 DISCUSSION

L'emploi d'une structure de domaine peut être discutée sous deux aspects : sa nécessité, compte-tenu des autres méthodes permettant d'assurer une protection, son efficacité.

En ce qui concerne le premier point, Ferrié ([14]) différencie les contrôles de protection statiques, effectués une seule fois à la compilation ou lors de l'édition de liens, des contrôles dynamiques, répétés systématiquement à chaque accès durant l'exécution. Malgré la conception de langages de programmation prenant en compte la protection des données (voir, par exemple A.K. Jones et B. Liskov [28]), ou encore la définition d'un interpréteur protégeant les communications entre des compilateurs de langages distincts (P. Darondeau [8]), le contrôle statique se révéle incomplet ; une panne, qui annulera la preuve de correction d'un programme, peut toujours survenir (Lanciaux [34], Denning [9]). De plus, l'existence de ce seul cas de violation de la protection suppose que tous les utilitaires du système sont corrects

et que le système d'exploitation assure de manière infaillible la prise en charge des problèmes de partage des données en temps réel non résolus par les utilitaires. Ces conditions semblent difficiles à réunir actuellement. Il faut cependant remarquer que les deux types de contrôle ne sont pas incompatibles.

La présence de contrôles dynamiques une fois justifiée, deux raisons imposent le choix d'un adressage par capacités, plutôt que celui d'un contrôle par listes d'accès (à un objet est associée la liste des agents autorisés à le manipuler - CROCUS [7]), autre méthode utilisable pour construire un environnement fermé : l'adressage par capacités est plus efficace que la liste d'accès ; la liste d'accès ne permet pas une modification aisée des droits d'un agent(Denning [9], Ferrié [14]). Mais ces deux moyens sont parfois combinés, en particulier pour la réalisation de catalogues (Redell [58], Lampson [31]).

Face à l'avantage d'unicité des contrôles statiques, se pose la question de l'efficacité d'une protection dynamique, et, plus généralement, de l'exécution de domaines, le domaine étant considéré comme une forme de conservation des modules à l'exécution. Il faut remarquer que Parnas recommande de dissocier conception modulaire des programmes et implémentation parce que, justement, l'exécution des modules eux-mêmes lui parait trop inefficace ([53]), compte tenu du critère de décomposition retenu (voir chapitre 1.). Par ailleurs, tirant les conclusions de leurs expérimentations sur le système CAL, Lampson et Sturgis rendent l'"overhead" consommé par certaines opérations du noyau réalisées par logiciel responsable de l'inefficacité globale de CAL ([32]). Enfin, de la petitesse des domaines dépend la fiabilité du logiciel. Or les domaines ne pourront être maintenus petits que si les opérations d'activation sont efficaces. Ces remarques, à priori défavorables à un système à structure de domaines, ne resteraient pas pertinentes, si le matériel supportait directement l'exécution des domaines. En ce qui concerne OMPHALE, la conclusion de cette discussion est subordonnée à l'examen de sa structure matérielle (chapitre 4).

A ce stade de l'étude, le système OMPHALE est structuré comme suit : les objets ne sont accessibles qu'à l'intérieur de domaines. Un domaine est représenté par la C-liste des objets constituant l'environnement d'exécution d'une opération d'un type. L'opération reçoit l'objet-opérande sous forme d'une capacité passée au domaine. Une opération de décomposition fournit une capacité de type C-liste permettant d'accéder aux composants de l'objet opérande.

Un calcul est constitué par l'exécution d'une suite de domaines. L'activation d'un domaine à partir d'un autre est contrôlée, le domaine étant un objet du système.

Le module est conservé à l'exécution sous forme d'un objet-type. La mise en oeuvre d'un domaine à partir de l'objet-type doit être contrôlée.

CHAPITRE 3

CREATION ET DESTRUCTION DES OBJETS

Le présent chapitre décrit, pour partie, le processus de création dynamique des objets dans OMPHALE. L'importante question de la mise en place des domaines, laissée en suspens au chapitre 2, pourra, alors être résolue. Mais les possibilités de création dynamique entrainent une prolifération des objets dans le système. Généralement, ce phénomène est combattu par la mise en oeuvre d'un programme de "ramasse-miettes", qui tente de détruire les représentations des objets inutiles. Le manque de performances de ce genre de programme est aggravé, dans le cas d'OMPHALE, par l'absence de mémoire commune à tous les sites du système. Aussi, il est proposé de n'appliquer le ramasse-miettes qu'au nombre restreint des objets dont il est difficile de décider de l'utilité. Pour ce faire, une hypothèse restrictive devra être énoncée quant à la structuration des objets.

3.1 OBJETS FORTEMENT TYPES

* *Définition*

Un **objet** construit est dit **fortement typé**, et son type fort, si ses n composants sont astreints à vérifier des **spécifications** "sévères" relatives :

- aux types des composants
- aux droits requis sur les composants,

et propres à son type. La sévèrité des spécifications peut être évaluée selon l'échelle suivante ; pour tout entier i, $1 \leq i \leq n$, la capacité pour le i-ème composant, quand elle existe, renferme le nom de type T_i et les droits D_i tels que :

1) Le type T_i est identique à un type imposé T'_i.

Les droits D_i sont supérieurs à des droits requis Dr_i.

ou bien,

2) la régle 1) ou la suivante sont vérifiées ;

Le type T_i est identique à un type T_{ij} appartenant à une union de types imposés :

$$T_{i1} \cup T_{i2} \cup \ldots \cup T_{in_i}$$

Les droits D_i sont supérieurs à des droits requis Dr_{ij} associés au type T_{ij}.

Ou bien,

3) Les règles 1), 2) ou la suivante sont vérifiées :

Le type T_i et les droits D_i sont quelconques.

Quand les règles 1) ou 2) sont respectées l'objet est considéré comme fortement typé.

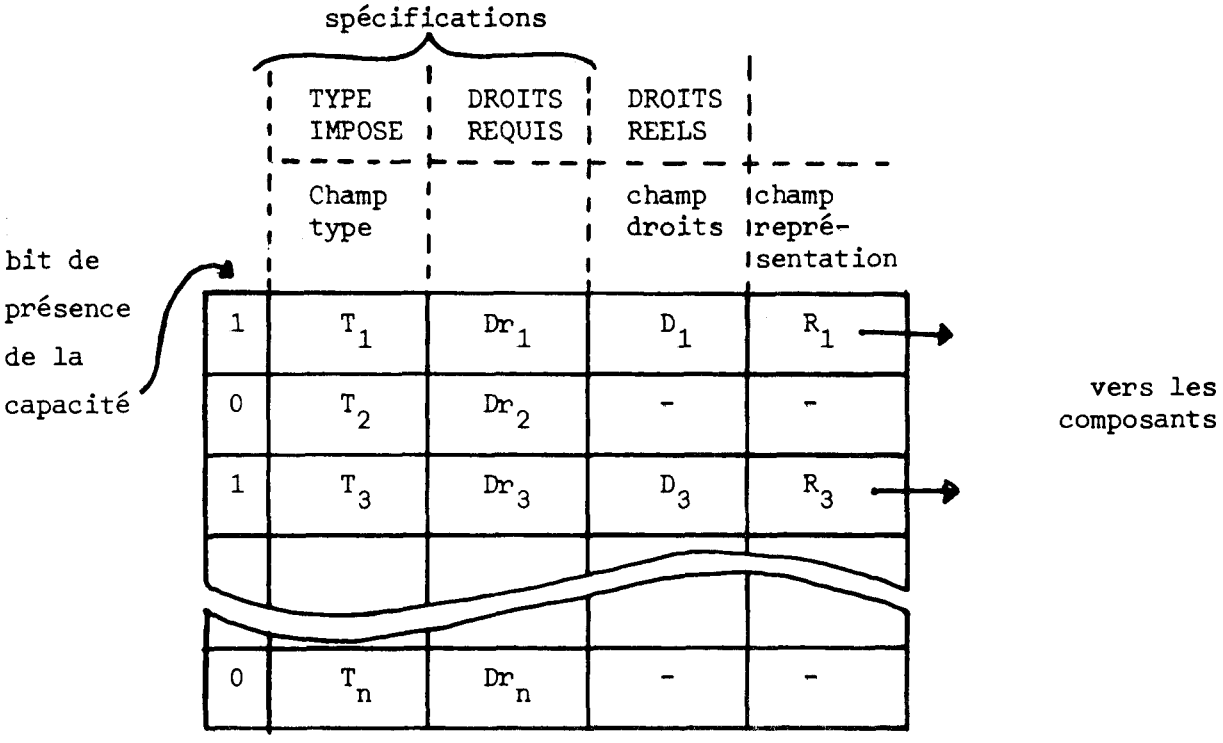
Pour cette étude il est supposé que les objets composant OMPHALE suivent la régle 1). Cette hypothèse peut sembler fort restrictive et trop rigide en pratique. Il est d'ailleurs envisagé de la dépasser au stade de l'implémentation du système. En fait, la règle 2) correspond à la possibilité de disposer de plusieurs représentations différentes pour les objets d'un type donné (bien qu'à un instant un objet du type possède une et une seule représentation). Ce problème caractéristique des langages supportant les types abstraits a reçu des solutions diverses. Aussi, adopter la régle 2) sous la forme énoncée, alors que les travaux engagés pour définir un langage de haut niveau adopté à OMPHALE n'ont pas encore abouti, imposerait une solution manquant de généralité et peu conforme à la philosophie qui préside à la conception du noyau (voir § 2.4). L'examen de langages de haut niveau devra dégager les mécanismes indispensables à la gestion des représentations multiples. Une fois ces mécanismes définis, l'implémentation de la structure d'un objet sera arrêtée et le processus de destruction - pour la simplification duquel l'hypothèse de force des types est actuellement nécessaire (voir § 3.3) - sera généralisé, si possible.

* *Conséquences sur la représentation des objets*

Les spécifications portant sur les composants d'un objet $O_{(T)}$ doivent figurer dans la représentation ; en regard de l'emplacement de la capacité pour un composant se trouvent :

- le type imposé T'_i
- les droits requis Dr_i.

La capacité et les spécifications pour un composant peuvent être partiellement confondues, ce qui transforme la C-liste représentant $O_{(T)}$ en la structure de données suivante :



Un emplacement ne renfermant pas de capacité est appelé **emplacement nul**. Par abus de langage, le même emplacement sera souvent désigné comme une **capacité nulle**. Le bit de présence de la capacité est alors appelé **indicateur de nullité**.

Remarques

1) La schématisation d'une capacité sous forme de ses champs type, droits et représentation continuera d'être utilisée partout où la présence des droits requis n'intervient pas.

2) La définition du type DOM contredit l'hypothèse de force des types : à priori un composant d'un domaine est d'un type quelconque. Cependant un domaine est représenté de la même manière que les autres objets et sa représentation renferme des spécifications. Mais les spécifications diffèrent d'un domaine à un autre.

3.2 CREATION DES OBJETS

Parce qu'elle intervient directement sur la stucture d'un objet et, donc, sur sa C-liste, l'opération de création manipule intensivement les capacités. Aussi, il est utile de détailler les possibilités de copie et de transfert des capacités avant d'examiner le processus de création des objets.

3.2.1 Préliminaires relatifs à la manipulation des capacités

- Soient d et s deux noms locaux dans un domaine. Le noyau fournit une opération de **transfert** et une **opération de copie de capacités** entre un emplacement source s et un emplacement destination d. L'activation de l'oparation de transfert est soumise à la possession d'un **droit de transfert,** noté T, et celle de l'opération de copie à un **droit de copie,** noté C. Les droits T et C forment les droits sur la capacité, par opposition aux autres droits qui constituent les droits sur le type. Les droits sur la capacité sont codés dans les champs droits de toutes les capacités. Un transfert ou une copie de capacité n'est réalisable que si l'emplacement destination est nul (ce que l'indicateur de nullité signale) ; il n'est pas permis d'"écraser" une capacité par une autre. Ce qui ne constitue pas une limitation ; il est toujours possible de détruire la capacité préexistante (voir, opération DETRUIRE, § 3.3.1). La gestion des références multiples à un objet en est facilitée. Les deux opérations n'affectent pas les spécifications contenues dans les emplacements s et d. Mais le type de la capacité à transférer ou à copier, ainsi que les droits réels qui y figurent doivent respecter les spécifications de l'emplacement destination. Dans les deux cas, le champ droits (réels) et le champ représentation de l'emplacement source sont transmis à l'emplacement destination dont l'indicateur de nullité est positionné, ensuite, à 1. Le transfert d'une capacité diffère de la copie en ce que la première opération annule l'emplacement source alors que la seconde ne le modifie pas. Les opérations de transfert de capacité et de copie de capacité sont, respectivement, notées :

d <- s	et	d <= s

- De plus, il n'est possible d'extraire une capacité d'une C-liste C (de placer une capacité dans une C-liste C) que si la capacité repérant la C-liste C renferme un **droit de lecture**, noté Lec (un **droit d'écriture**, noté Ecr). Car, dans certains cas, il est nécessaire de contrôler les mouvements de toutes les capacités composant une C-liste.

- Pour éviter l'emploi de segments de taille réduite à quelques valeurs élémentaires, les capacités de type SEG peuvent contenir des **valeurs immédiates** (leur nombre exact dépendra de l'implémentation définitive des capacités) en lieu et place de leur champ représentation. Deux opérations

v ; = LIRE (s, n)	et	ECRIRE (s, n) : = v

permettent, respectivement, de lire ou d'écrire la n-ième valeur immédiate du segment repéré par la capacité de nom local s dans un domaine.

3.2.2 *La maquette comme modèle de création des objets*

Un objet est entièrement défini une fois que tous ses composants - ou, au moins, les spécifications sur ses composants - existent, qu'ils sont référencés dans sa C-liste et que l'agent qui a déclenché l'opération de création possède une capacité pour l'objet nouvellement créé. Aussi le processus de création est décomposé en trois phases : 1) **la création d'une C-liste.** *La C-liste étant un objet du type primitif, elle fait appel à une opération fournie par le noyau.* 2) **la création des composants et l'insertion des capacités les repérant dans la C-liste.** *Une maquette sert de modèle à l'élaboration de la C-liste de l'objet à créer ; son interprétation déclenche les créations successives des composants.* 3) **La création d'une capacité pour l'objet créé.** *Elle est réalisé par l'appel d'une opération de composition.*

* *Création des objets primitifs*

Le noyau fournit des opérations de création des objets primitifs. Soient n un nom local dans un domaine, l et c deux entiers. L'**opération de création des segments** s'écrit :

n < - CREER_SEGMENT (ℓ)

Un segment de ℓ valeurs élémentaires nulles est créé. Une capacité pour le segment est placée à l'emplacement de nom n. L'**opération de création des C-listes** s'écrit :

n < - CREER_C_LISTE (c)

Une C-liste de c emplacements nuls est créée. Une capacité pour la C-liste est placée à l'emplacement n.

Les droits figurant dans les capacités pour les objets nouvellement créés sont maximaux. Les opérations ne sont réalisées que si leur résultat s'accorde avec les spécifications de l'emplacment n. De plus, les valeurs ℓ et c sont soumises à des contrôles d'utilisation des ressources (voir § 6.4).

Quant à la description du processus de création des maquettes, elle est subordonnée à l'étude de l'organisation des maquettes.

* *Interprétation de la maquette d'un objet* (Lanciaux [34])

Les objets primitifs étant créés par le noyau, seuls les objets construits sont créés par interprétation d'une maquette. Soit $O_{(T)}$ un objet construit à créer. Il est supposé que $O_{(T)}$ possédera n composants, notés $O_{1(T_1)}$, $O_{2(T_2)}$,..., $O_{n(T_n)}$ dans l'ordre de rangement de leur capacité dans la C-liste.
La **maquette**, objet primitif du type MAQ, est caractéristique du type T de l'objet à créer. Elle est formée d'une liste de n **descripteurs de composants**. Le i-ème descripteur de composant renferme :

- une **commande d'interprétation** qui indique les actions à entreprendre pour créer le composant.
- les **spécifications** T_i, Dr_i relatives au composant.
- d'autres informations dépendant des valeurs des précédentes.

Une C-liste de n capacités nulles ayant été créée, l'**interprétation** de la maquette se déroule comme suit ; la liste des descripteurs est parcourue ; pour chaque descripteur un composant est créé conformément à la commande d'interprétation et la capacité pour le composant est placée dans la C-liste en cours d'élaboration.

Les commandes d'interprétation sont au nombre de trois ; une commande de copie utilisée pour les composants existant avant la création de $0_{(T)}$; une commande d'annulation utilisée pour les composants qui seront créés ultérieurement ; une commande de création utilisée pour les autres composants. De plus, une maquette peut être paramètrée : un paramètre est soit un objet d'un type déterminé, soit une taille de segment ou de C-liste. Ces paramètres sont contenus dans une C-liste des paramètres d'interprétation, sous forme d'une capacité pour l'objet dans le premier cas, ou sous forme d'une capacité à valeur immédiate dans le second. Les possibilités de combinaison des paramètres et des commandes, vis à vis du i-ème descripteur, sont les suivantes :

- la **commande de copie** peut porter sur un objet dont une capacité figure :

1) dans le descripteur lui-même. Cette capacité est alors copiée du descripteur à l'emplacement i de la C-liste de l'objet à créer.

2) A l'emplacement p dans la C-liste des paramètres d'interprétation. Cette capacité est alors copiée de l'emplacement p à l'emplacement i.

- la **commande d'annulation** procéde à la copie des seules spécifications. L'indicateur de nullité de l'emplacement i reste positionné à 0.

- la **commande de création** peut porter :

1) sur un objet primitif de type SEG ou CL. L'une des opérations

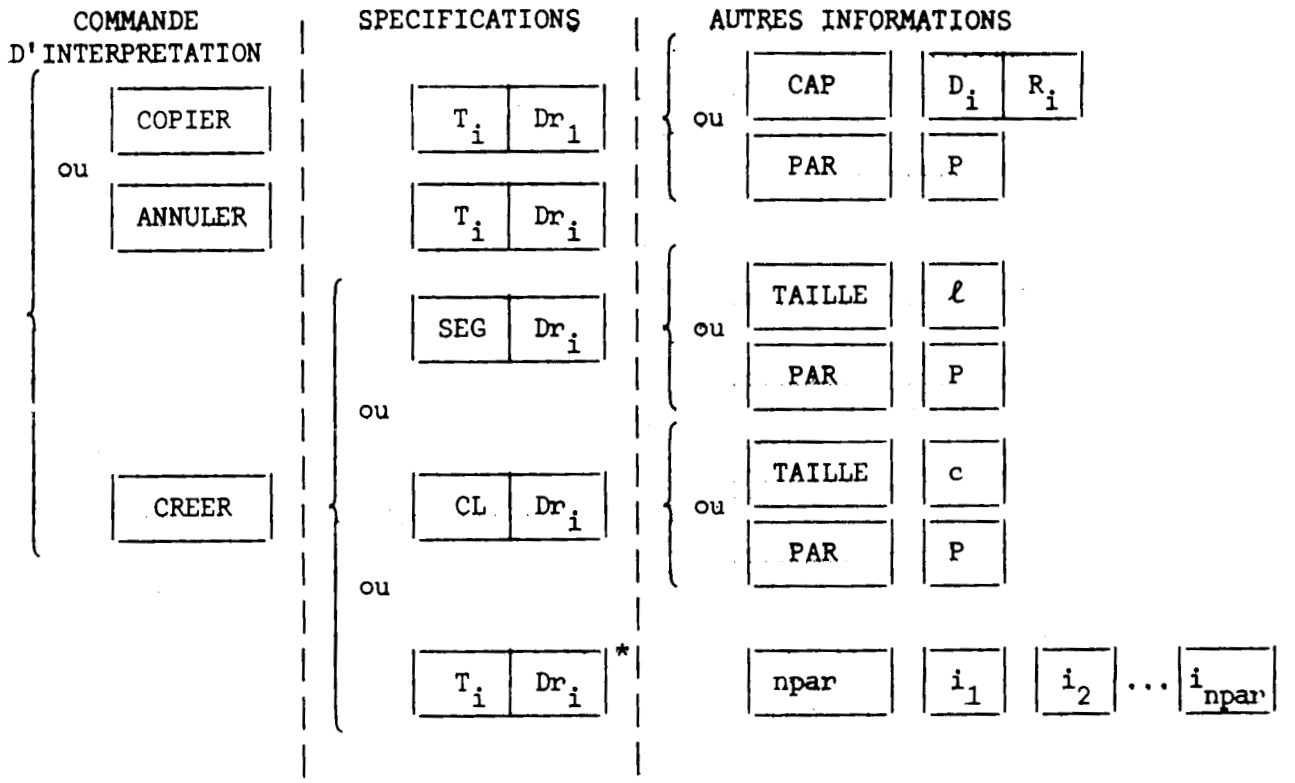
$$i \leftarrow \text{CREER_SEGMENT } (\ell) \quad \text{ou} \quad i \leftarrow \text{CREER_C_LISTE } (\ell)$$

est alors appelée. La valeur ℓ ou c est extraite soit du descripteur, soit de l'emplacement p de la C-liste des paramètres.

2) Sur un objet construit de type T_i. Une opération d'interprétation de la maquette de l'objet T_i est alors activée (voir § 3.2.4). La capacité résultante est placée à l'emplacement i de la C-liste. Le descripteur renferme une liste des npar informations paramétrant l'interprétation de la maquette du composant . Ces informations sont, soit des index dans la C-liste des paramètres d'interprétation de la maquette de $0_{(T)}$, soit, directement, les valeurs des paramètres.

Dans les cas 1) et 2) les spécifications sont copiées du descripteur à la C-liste.

La figure ci-après résume des les différentes possibilités de combinaisons de commandes et de paramètres :



Observation : *Dans ce cas, T_i est nécessairement différent de CL ou SEG.
npar peut être nul.

Notations :
- T_i, Dr_i : spécifications
- D_i : droits réels
- R_i : champ représentation

} de la capacité pour le i-ème composant

npar : nombre de paramètres

ℓ, c : taille de segment, de C-liste

$i_j = (b_j, f_j)$ pour j = 1, ..., npar ; Si $b_j = 0$, $f_j = p_k$ index dans les C-listes des paramètres d'interprétation.

Si $b_j = 1$, $f_j = V_L$ valeur de paramètre d'interprétation.

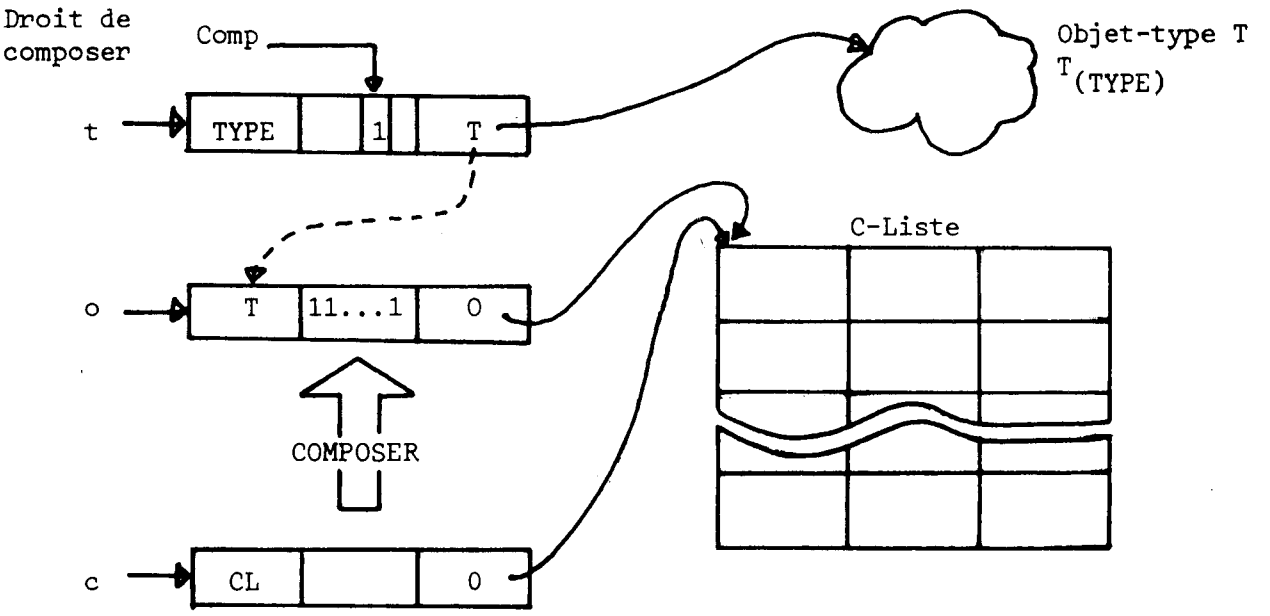
p : index dans la C-liste des paramètres d'interprétation.

* *Opération de composition*

L'**opération de composition** fait partie du type TYPE. A l'inverse de l'opération de décomposition, elle produit une capacité de type T à partir d'une capacité de type CL. Son activation est subordonnée à la présence d'un **droit de composition**, noté Comp, dans une capacité pour l'objet-type T. Soient c, o, t des noms locaux dans un domaine, L'opération de composition s'écrit :

o < = COMPOSER(c, t)

et son activation peut être schématisée comme suit ;



Les droits obtenus dans la capacité o sont maximaux.

*Création des maquettes

La création des maquettes est effectuée par un domaine faisant partie du compilateur. Ce domaine reçoit en paramètres un segment renfermant un texte source décrivant les commandes d'interprétation, et les capacités pour les objets existant au moment de l'interprétation de la maquette (commande de copie). Au besoin, le domaine crée de tels composants. De plus, il possède une capacité pour l'objet-type MAQ munie du droit de composition

Les trois phases du processus de création - création d'une C-liste, interprétation d'une maquette, composition - ont été décrites. Mais, pour expliquer le fonctionnement de l'opération de création elle-même, il est nécessaire de connaitre l'organisation de l'objet-type, qui représente le type dans le système. Or les relations entre les domaines - environnements d'exécutions des opérations des types - et les objets-type n'ont pas encore été exposées.

3.2.3 *Mise en place des domaines à partir de l'objet-type*

* *Dissociation de la création et de l'activation des domaines.*

Dans les systèmes manipulant des procédures, chaque appel d'une procédure déclenche la création, puis l'activation d'un nouvel environnement, Plus précisément, l'appel procédural se déroule, généralement, comme suit :

1) première étape : **création de l'environnement** ; les objets permanents de la procédure sont inclus dans l'environnement ; les objets locaux sont créés ; les objets rémanents et les objets paramètres sont insérés dans l'environnement.

2) deuxième étape : **activation de l'environnement** ; le contrôle est transféré au nouvel environnement.

Lanciaux souligne les insuffisances d'un tel schéma ([34]) :

- l'incorporation des objets rémanents à l'environnement est souvent une opération complexe et coûteuse en temps de calcul. Ainsi, par exemple, le système MULTICS (CROCUS [7]) consulte un segment de liaison pour retrouver les rémanents.

- La fourniture d'un seul appel procédural manque de généralité.

Il propose de dissocier création de l'environnement et activation de l'environnement. Une opération de création de domaine et des opérations de transfert de contrôle sont fournies. La première est réalisée par l'interprétation d'une maquette de domaine. Il n'existe qu'un nombre restreint de règles partant sur la structure des domaines, afin d'en garantir l'universalité. Les opérations de transfert de contrôle comprennent des opérations élémentaires de transfert de paramètres, d'activation de domaine, de retour de domaine, etc ... (elles sont décrites au § 6.2) qui sont utilisées pour mettre en place aussi bien des appels procéduraux que des activations en parallèle de domaines, par exemple. L'accès aux rémanents est assuré en conservant tel quel le domaine qui les contient après exécution de l'opération qu'il réalise. La capacité qui repère le domaine est transmise à d'autres domaines qui provoquent d'autres activations. La création des objets locaux, dans ce cas, est déclenché par des instructions de création figurant dans le code de l'opération.

puissent pas être diffusées à l'extérieur du domaine D_i, elles ne possèdent pas les droits de copie et de transfert. De même, la capacité de type CL résultant d'une décomposition est privée des droits C et T, pour éviter une divulgation de la structure d'un objet du type hors du domaine D_i qui le manipule. Les descripteurs munis de la commande de création et inclus dans la maquette du domaine D_i correspondent, soit à des objets locaux, soit à des objets rémanents, suivant qu'un nouveau domaine est créé à chaque appel de l'opération op_i, ou que le même domaine D_i est réactivé. Les descripteurs portant la commande d'annulation sont vraisemblablement associés à de futurs objets paramètres.

L'opération de création des objets joue un rôle central dans le système : non seulement, elle est bien évidemment utilisée pour créer les objets, mais, de plus, elle participe à la mise en place des domaines.

3.2.4 *Opérations de création*

La création d'un domaine diffère peu de la création d'un objet d'un type quelconque T. Dans les deux cas, il s'agit de créer une liste de capacités nulles, d'interpréter la maquette, puis de créer une capacité soit de type T, soit de type DOM, par composition. Cependant, la sélection de la maquette et le déclenchement de l'opération imposent de fournir deux opérations distinctes : dans le cas de la création d'un objet de type T, le type est spécifié par une capacité repérant l'objet-type T et munie d'un **droit de création** (noté Cr) ; par contre un domaine réalisant une opération d'un type T est créé à partir d'un objet quelconque du type T. Il est alors nécessaire d'indiquer l'opération désirée : un numéro d'opération y pourvoit (ce n'est pas la seule solution possible).

Soient c, d, o, t et p des noms locaux dans un domaine et i un entier, l'**opération de création** s'écrit :

```
c <- ∿ CREER_OBJET (t, p)
```

t désigne une capacité pour l'objet-type T.
p désigne une capacité pour la C-liste des paramètres d'interprétation de la maquette.

La maquette est sélectionnée par son emplacement conventionnel dans l'objet-type. De même, la composition est effectuée grâce à la capacité pour l'objet-type T qui figure dans l'objet-type lui-même (c'est la seule capacité du système qui doit posséder le droit de composer pour le type T ; elle est privée des droits C et T). La capacité résultante est placée en c. Par contre l'**opération de création des domaines** s'écrit :

```
d <- ∿ CREER_DOMAINE (o, i, p)
```

o désigne une capacité pour un objet de type T.
i est un numéro d'opération
p désigne une capacité pour la C-liste des paramètres d'interprétation de la maquette.
Ici, la i+1-ème maquette de l'objet-type T, correspondant à l'opération op_i, est interprétée. La capacité pour l'objet-type DOM qui permettra de créer une capacité de type DOM figure dans le domaine même qui réalise l'opération de création et n'a pas besoin d'être placée dans l'objet-type T. La capacité résultante est transférée à l'emplacement d.

Par ailleurs, il faut assurer la récursivité du processus de création, ce qui implique qu'une opération de création puisse être déclenchée par l'interprétation d'un descripteur de composant muni d'une commande de création portant sur un type construit. En fait, le noyau est doté d'une **opération d'interprétation de maquette** qui lui est interne, qui n'est pas accessible aux utilisateurs du système, et qui est employée pour la réalisation des opérations CREER_OBJET et CREER_DOMAINE. Cette opération s'écrit :

```
c <- ∿ INTERPRETER_MAQUETTE (T, n, p)
```

où T est le nom unique de l'objet à créer
n est un numéro de maquette
p est le nom local d'une capacité pour la C-liste des paramètres d'interprétation de la maquette
c est le nom local de l'emplacement destiné à la capacité pour l'objet nouvellement créé.

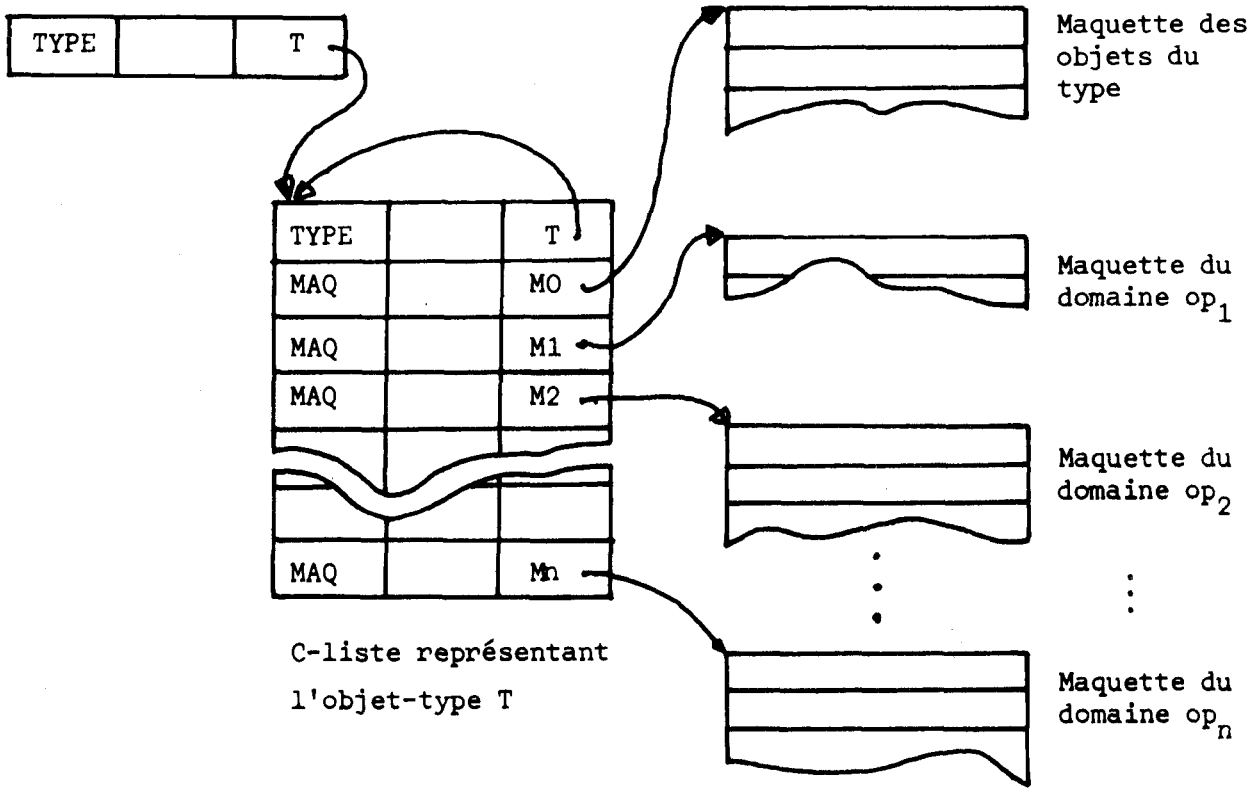
L'exécution de

```
c <- ∿ CREER_OBJET (t, p)
```

déclenche l'extraction de T dans la capacité pour l'objet-type de nom local t,

* *Conséquences*

Le paragraphe 3.2.2 montre que la structure des objets du type T est représentée par la maquette qui permet de les créer par interprétation. Il est également possible de représenter les opérations op_1, op_2, ..., op_n du type T par des maquettes : un domaine D_i réalise l'environnement d'exécution d'une opération op_i ; comme la création et l'activation d'un domaine sont dissociées, l'opération op_i peut être représentée par une maquette du domaine D_i. Dès lors, l'objet-type peut être vu comme un objet construit composé de n+1 maquettes : une maquette par opération du type, plus une maquette des objets du type. Par convention, la première capacité dans la C-liste représentant l'objet type repère l'objet type lui-même, la seconde capacité désigne la maquette des objets du type. Ainsi, l'objet-type possède la structure suivante :



Les objets permanents d'une opération op_i sont alors représentés par des capacités figurant dans la maquette du domaine D_i sous forme de descripteurs munis de la commande de copie. Une capacité de type SEG pour un segment de code ou une capacité de type TYPE permettant de décomposer les objets du type T entrent dans cette catégorie. La maquette permet ainsi de se passer d'objet prodédure. Afin que les capacités repérant les objets permanents ne

l'appel

c <- ∿ INTERPRETER_MAQUETTE (T, 1, p)

Et l'exécution de

c <- ∿ CREER_DOMAINE (o, i, p)

provoque l'obtention du nom unique T à partir de la capacité pour l'objet $O_{(T)}$ de nom local o et l'activation

c <- ∿ INTERPRETER_MAQUETTE (DOM, i+1, p)

Comme plusieurs sites sont impliqués dans l'opération d'interprétation, il est prématuré d'en expliquer le fonctionnement dans ce chapitre (voir 5.6.1). De même, la discussion de l'efficacité des opérations de création - la création d'un objet construit déclenche des créations en cascade de ses composants jusqu'aux objets primitifs - est remise.

Enfin pour certains types (types récursifs, objets dont la création nécessite l'allocation de ressources autres que l'espace mémoire qu'ils occupent), la création d'un objet ne peut pas être réduite à l'interprétation d'une maquette. Dans ce cas la création est prise en charge par un domaine (la création des objets du type est une opération du type).

Remarque : l'activation de l'opération d'interprétation de maquette est subordonnée à un contrôle des ressources consommées par la création (voir § 6.4).

3.3 DESTRUCTION DES OBJETS

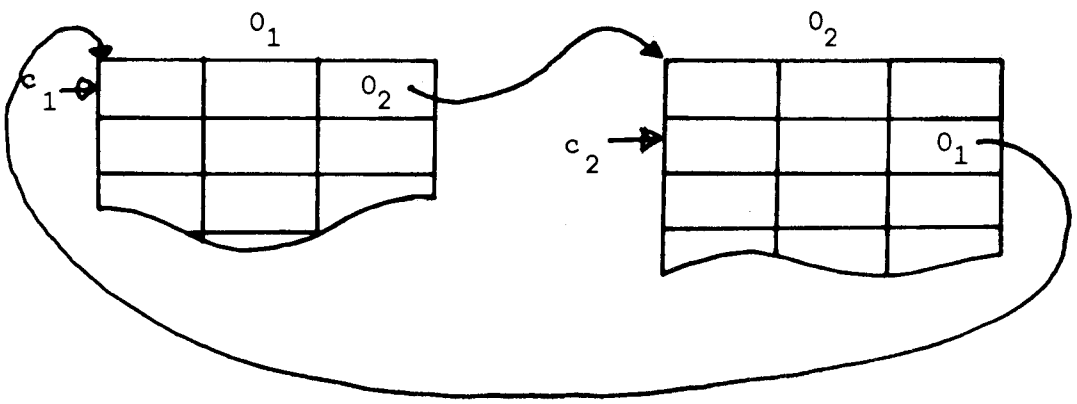
3.3.1 Position du problème

Par définition, un objet n'existe pour le système que s'il est référencé par une capacité. Généralement, la destruction de la représentation des objets - i.e., la récupération de la place qu'ils occupent en mémoire - n'est pas effectuée explicitement : les objets non référencés sont marqués comme tels et un **programme de "ramasse-miettes"**, qui est exécuté cycliquement et, peut-être, en parallèle avec le traitement, parcourt la mémoire et libère les emplacements marqués. Le problème est alors de décider si un objet est encore référencé ou non. Une solution partielle consiste à associer à l'objet un **compteur de références**. A chaque fois qu'une capacité pour un objet est copiée, le compteur est incrémenté de un. A la création de l'objet, il est

positionné à un. Une **opération de destruction**, notée

DETRUIRE c

permet de détruire la capacité de nom local c dans un domaine. Chaque destruction de capacité donne lieu à une décrémentation du compteur de références de un. Quand la valeur du compteur devient nulle, l'objet n'est plus référencé et sa représentation peut être détruite par le ramasse-miettes. Cependant, cette méthode se révèle incomplète, car les compteurs de références ne prennent pas en compte les boucles de références. Soient deux objets O_1 et O_2, chaque objet repérant l'autre grâce à une capacité pour l'autre figurant dans sa propre C-liste. Par hypothèse, chaque capacité est l'unique référence à l'objet qu'elle repère.



Les compteurs de références des deux objets valent un. Mais ils ne passeront jamais à zéro puisqu'aucun des deux objets n'est référencé dans un domaine qui aurait pu détruire soit c_1, soit c_2. Les objets O_1 et O_2 sont des **objets perdus** : ils existent mais sont inacessibles. Aussi la technique des compteurs doit être complétée.

Au lieu de noter les objets inaccessibles, il est possible de marquer les objets accessibles, grâce à un **programme de parcours de références**, le ramsse-miettes détruisent, ensuite, les objets non marqués. Le parcours démarre d'un ensemble d'**objets visibles**, directement accessibles. L'arbre des références constitué par les C-listes composant les objets visibles est parcouru jusqu'à ses feuilles. Chaque objet rencontré durant le parcours est marqué accessible. Mais la lourdeur de cette méthode la rend inutilisable : le temps de calcul consommé par le programme de parcours est environ proportionnel au nombre d'objets du système. C'est, par contre, la seule façon de

détecter les objets perdus. En fait, cette deuxième méthode ne devient viable que si elle ne s'applique qu'à un nombre restreint d'objets. Or, certains objets du système ne seront jamais perdus ; les segments ne peuvent pas former des boucles de références, par exemple, car ils ne contiennent pas de capacités. Pour ces derniers objets la méthode des compteurs de références fonctionne (Birrell et Needham [3]). D'où l'idée de caractériser les objets susceptibles de devenir des objets perdus, de signaler ce fait dans leur représentation, de les traiter, et eux seulement, par le programme de parcours des références, puis de briser les boucles de références liant les objets détectés comme perdus, la destruction de tous les objets restant déclenchée par la technique des compteurs. L'hypothèse de force des types facilite la catégorisation des objets susceptibles d'être perdus.

3.3.2 *Localisation des objets perdus*

Soit un objet $O_{1(T_1)}$ appartenant à une boucle de références. Par hypothèse il existe n objets (n entier), notés $O_{1(T_1)}$, $O_{2(T_2)}$, ..., $O_{n(T_n)}$ tels qu'une capacité de la C-liste représentant $O_{i(T_i)}$ référence $O_{i+1(T_{i+1})}$, pour tout entier i, $1 \leq i \leq n$ (si, par convention $O_{n+1(T_{n+1})} = O_{1(T_1)}$). Les objets étant fortement typés, le type d'un objet détermine sans ambiguïté les types de ses composants. Aussi, puisque $O_{i+1(T_{i+1})}$ se présente comme un composant de $O_{i(T_i)}$, le type T_i est construit sur le type T_{i+1} ; puis, de même, T_{i+1} est construit sur T_{i+2} ; et ainsi de suite. Par récurrence le type T_i est construit à partir de lui-même. Un type construit sur lui-même est qualifié de **récursif** (et les objets de ce type sont des **objets récursifs**). Seuls les objets récursifs sont susceptibles d'être perdus, car ce sont les seuls objets capables de former des boucles de références. Il est ainsi établi que l'ensemble des objets perdus est inclus dans l'ensemble des objets récursifs.

De plus, l'hypothèse de force des types implique que quand un objet $O_{(T)}$ est un composant d'un objet $O'_{(T')}$, le type T' est construit sur le type T. Les types récursifs et les types construits sur des types récursifs sont appellés types prérécursifs (et les objets sont, eux aussi, des **objets prérécursifs**). L'ensemble des objets récursifs n'est potentiellement accessible

qu'à partir de l'ensemble des objets prérécursifs. Aussi, pour isoler les objets perdus, il suffit de partir de l'ensemble des objets à la fois prérécursifs et visibles, puis de parcourir l'arbre des références en ne marquant que les objets récursifs. A l'issue du processus, les objets récursifs non marqués sont les objets perdus (une preuve, due à C. Carrez, est donnée en annexe 1). L'algorithme de parcours et de marquage est détaillé au paragraphe 5.7.3.

Remarque La vérification des propriétés de récursivité, spécifiques du type, est détectée dès la compilation des maquettes. Aussi, des indicateurs présents dans les maquettes et dans les représentations des objets signalent les objets prérécursifs ou récursifs.

En résumé.

Le système OMPHALE est constitué d'objets fortement typés : le type d'un objet détermine le type de ses composants. Un type est représenté par un objet-type, structure de regroupement de maquettes de création des objets. L'interprétation de l'une de ces maquettes délivre, soit un objet du type, soit un domaine. Ce dernier réalise l'environnement d'exécution de l'une des opérations du type.

Le système est doté d'une structure de domaines : des mécanismes de protection assurent que les objets ne sont pas accessibles à l'extérieur de domaines autorisés à les manipuler.

Un calcul se traduit par l'exécution d'une suite de domaines. Un domaine donné participant au calcul active un autre (des autres) domaine(s) grâce à des opérations de transfert de contrôle. L'opération de création de domaine (l'interprétation de sa maquette) est dissociée de toutes les opérations de transfert de contrôle. Les capacités pour les objets permanents d'une opération sont copiées de la maquette au domaine. Les objets locaux peuvent être créés à chaque activation du domaine ou en même temps que le domaine, au gré de l'utilisateur. Durant toute leur vie, les objets rémanents restent enfermés dans le domaine qui les a créés ; ainsi ils sont facilement retrouvés à chaque activation du domaine qui les supporte.

Un objet d'un type T donné est passé en paramètre à l'un des domaines qui réalise une opération du type T. La légitimité du déclenchement de l'opération est contrôlée par la décomposition d'une capacité pour l'objet, opération qui, par ailleurs, permet l'accès à la C-liste repérant les composants de l'objet.

Un objet est détruit soit quand il n'est plus référencé par aucune capacité, soit quand il n'est devenu définitivement inaccessible.

CHAPITRE 4

DEFINITION D'UNE ORGANISATION MATERIELLE ADAPTEE A LA STRUCTURE DE DOMAINES

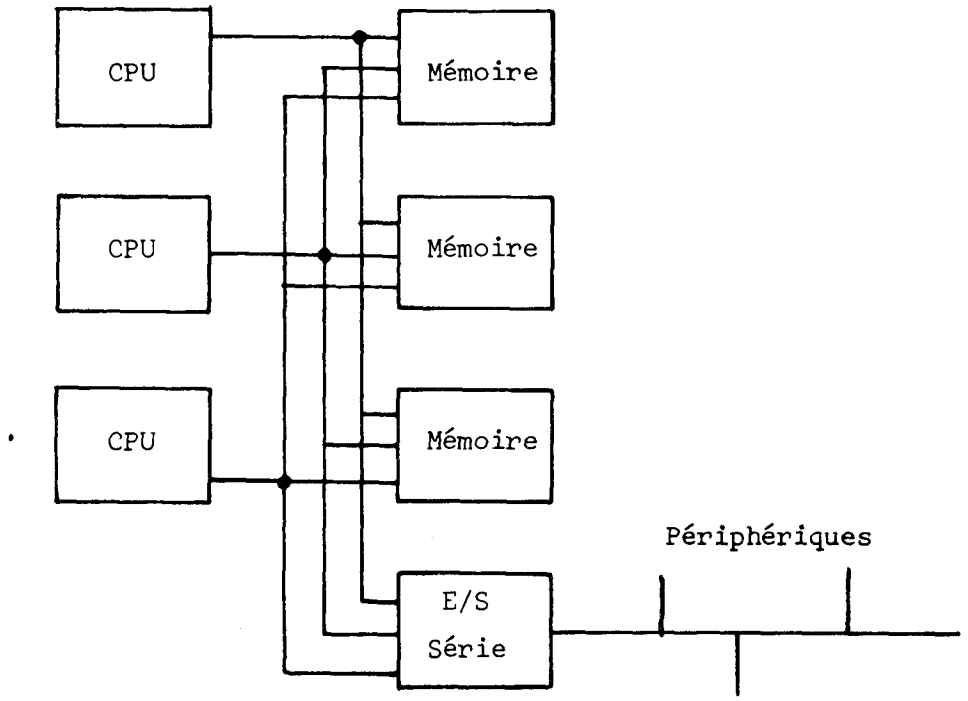
Par l'abondance de ses références à ces projets, la littérature spécialisée a érigé, sans doute trop rapidement, HYDRA et le PLESSEY 250 au rang de paradigmes des systèmes dotés d'un adressage par capacités. Aussi, ces exemples, tous deux implantés sur des multi-processeurs, conduisent à réduire les systèmes supportant une structure de domaines à des machines complexes, lourdes et coûteuses. La première partie de ce quatrième chapitre rendra justice à cette impression : le panorama des sytèmes à capacités qui y est déroulé montre la diversité des organisations matérielles employées. Dès lors, la deuxième partie aborde la question du choix d'une structure adaptée à l'application envisagée pour OMPHALE, ainsi qu'aux qualités qui lui sont demandées. Puis, après avoir détaillé l'architecture matérielle dans une troisième partie, il sera possible de reprendre la discussion laissée en suspens au terme du chapitre 2, et de conclure sur l'efficacité supposée du système.

4.1 PANORAMA DES STRUCTURES MATERIELLES DES SYSTEMES MUNIS D'UN ADRESSAGE PAR CAPACITES

Sans pour autant prétendre à une revue exhaustive des systèmes munis d'un adressage par capacités, la plupart des projets seront cités, qu'ils aient abouti ou non à une réalisation. Mais seules les descriptions des propositions les plus significatives seront développées.

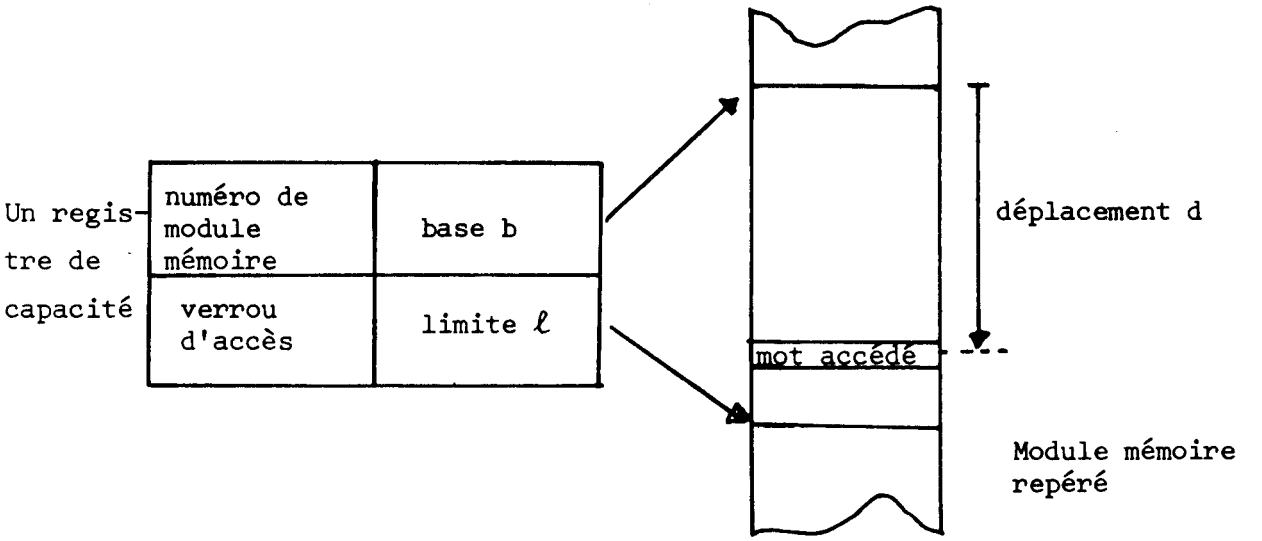
4.1.1 *Le Plessey 250 : un multi-processeur muni d'un adressage matériel par capacités* (*England* [11], *Cosserat* [6]).

Le Plessey 250 est un multi-processeur développé pour des applications de télécommunications fonctionnant en temps réel. Il semblerait qu'il soit le seul système à capacités qui ait été commercialisé. Les modules CPU constituant le système partagent, par l'intermédiaire de bus parallèles, des modules de mémoire et un module d'entrées/sorties séries, selon la configuration typique suivante :



La mémoire se présente comme une mémoire virtuelle segmentée organisée en mots de 24 bits. Les segments contenant des données sont distingués des segments renfermant des capacités.

En plus des registres de données, un CPU est équipé de 8 registres de capacités (RC) qui entrent dans le schéma d'adressage de la machine : un RC localise un segment dans la mémoire commune et contient un verrou qui permet de contrôler les accès :



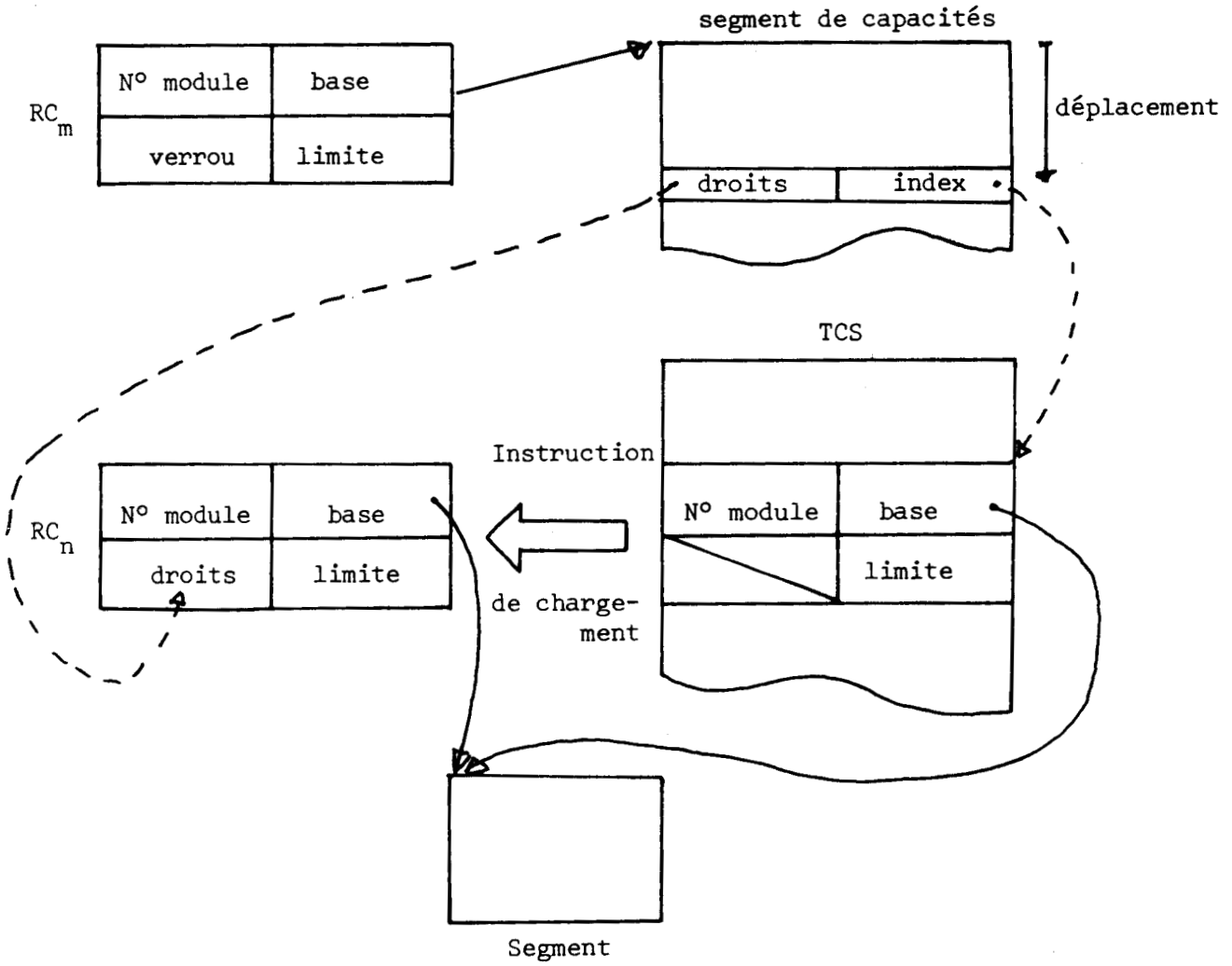
Les instructions sont formatées comme suit (en simplifiant quelque peu) :

code opération	registre de donnée	registre de capacité	déplacement d

Le registre de capacité nommé dans l'instruction sélectionne un module de mémoire et une base b dans ce module. L'adresse effective générée par le processeur dans le module désigné est b+d. A chaque accès le matériel vérifie que la valeur b+d est inférieure à la limite ℓ et que l'opération est compatible avec le verrou d'accès.

La copie des capacités d'un segment (de capacités) à un autre est libre, mais contrôlée par des droits de lecture et d'écriture dans les segments.

Une capacité (48 bits) est constituée de deux champs ; un champ droit et un champ représentation. Le champ représentation renferme un index dans une Table des Capacités du Système (TCS). Chaque entrée de la TCS comprend les informations localisant le segment en mémoire. Pour rendre un segment accessible à un CPU, des instructions de chargement des capacités dans les RC sont fournies. L'action d'une telle opération est décrite ci-dessous :



Chargement d'une capacité dans RC_n à partir de (RC_m) + déplacement

Certains RC sont spécialisés et repèrent l'environnement, la structure de données représentant le processus et le segment de code en cours d'exécution. Un registre spécial désigne la pile associée au processus.

Il faut noter que les opérations de manipulation de capacités sont intégralement réalisées par le matériel.

Remarque : D'après Fabry ([13]), le Chigaco Magic Number Computer, dont la construction a été abandonnée, utilisait des dispositifs matériels similaires à ceux du Plessey 250, si ce n'est que la TCS n'existait pas et que le champ représentation d'une capacité renfermait directement une adresse absolue.

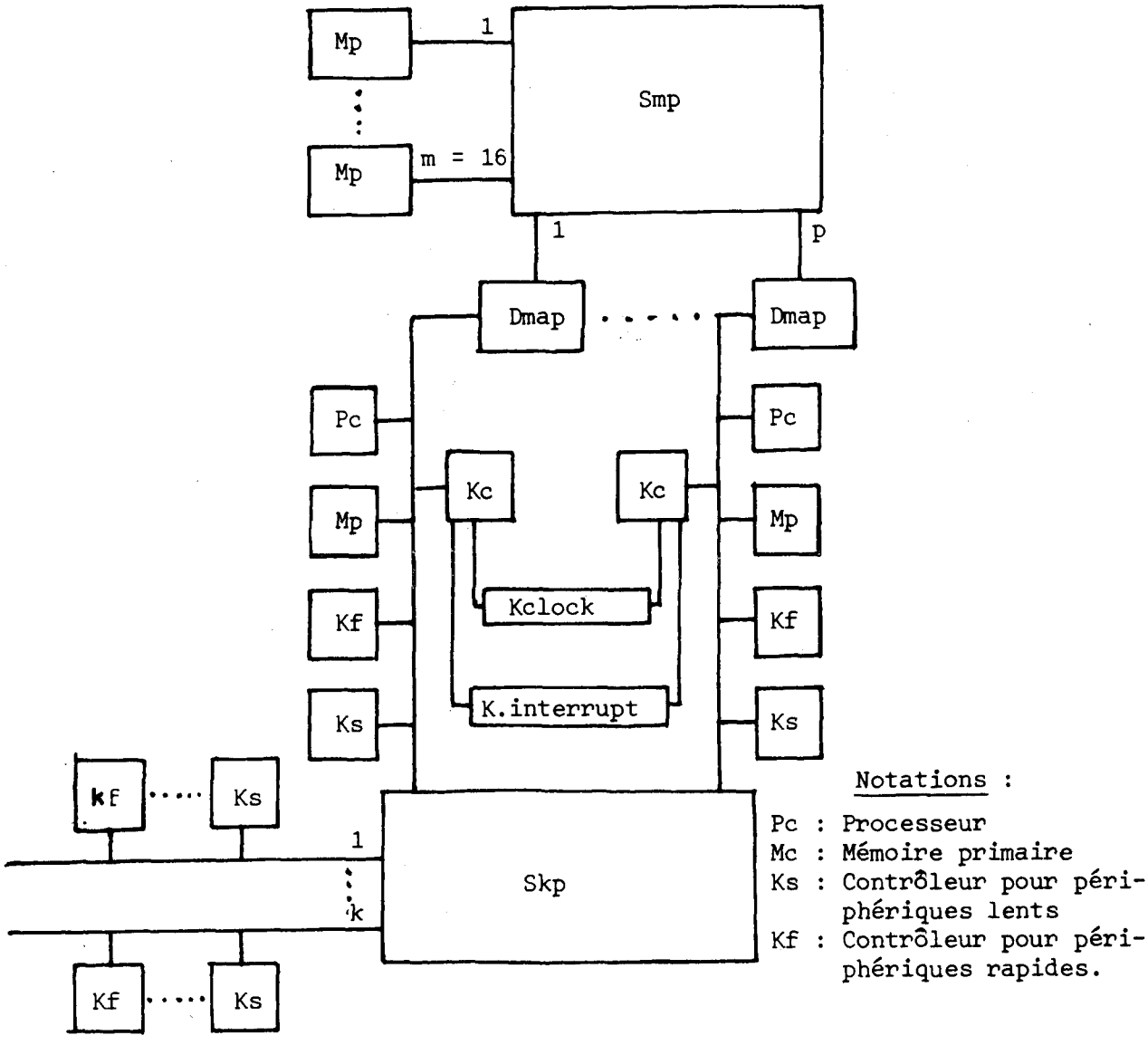
4.1.2 *La réalisation de Spier : la simulation d'une structure de domaines sur un mini-ordinateur* ([62, 63])

Spier implémente une structure de domaines sur un mini-ordinateur. Dans son modèle, un domaine est composé de cinq segments renfermant respectivement, une procédure ré-entrante, une structure de données propre à la procédure et gérée par elle, une zone de travail pour les valeurs temporaires locales, une zone de travail pour les valeurs rémanentes et une zone de communication d'arguments, associée au processus qui exécute le domaine. La machine utilisée, un PDP-11/45, distingue deux espaces d'adressage : un espace d'instructions et un espace de données. A chacun d'entre eux correspond un jeu de 8 Registres de Pages Actives (RPA). Ces registres tranforment l'espace de références constitué par le domaine en un espace mémoire accessible au processeur. Un RPA est un registre de base protégé qui désigne une aire de mémoire de taille variable, par bonds de 64 octets, entre 64 octets et 8 K octets. Un RPA est muni d'un verrou d'accès. Les registres délimitant l'espace d'instruction sont sous-utilisés ; ils ne repèrent que le segment de code. Pour l'espace de données, une convention de programmation partage sept RPA entre les segments de valeurs locales, de valeurs remanentes et d'objets propres à la procédure. Le huitième RPA est réservé aux arguments. Le chargement des registres est effectué par le logiciel de base, qui est exécuté, ainsi que toutes les opérations de gestion des domaines, en mode noyau (le PDP-11/45 dispose, de plus d'un mode superviseur et d'un mode utilisateur moins privilégiés). La maintenance de la structure de domaine est donc "simulée" par logiciel.

Remarque : Au moins deux autres systèmes à capacités ont été implantés sur des mono-processeurs, fort différents, cependant, du PDP-11. Il s'agit du système SUE, monté sur un IBM 360 (Casey [4]) et du système CAL supporté par un Control Data 6400 (Lampson et al. [32]). Ce dernier ordinateur est équipé d'un "Extended Core Store", utilisé comme intermédiaire dans la hiérarchie mémoire entre la mémoire primaire et la mémoire de masse.

4.1.3 HYDRA/Cmmp : l'implantation d'un adressage par capacités sur un multiprocesseur doté d'une matrice de connexion "crossbar" (Wulf et al. [67, 69]).

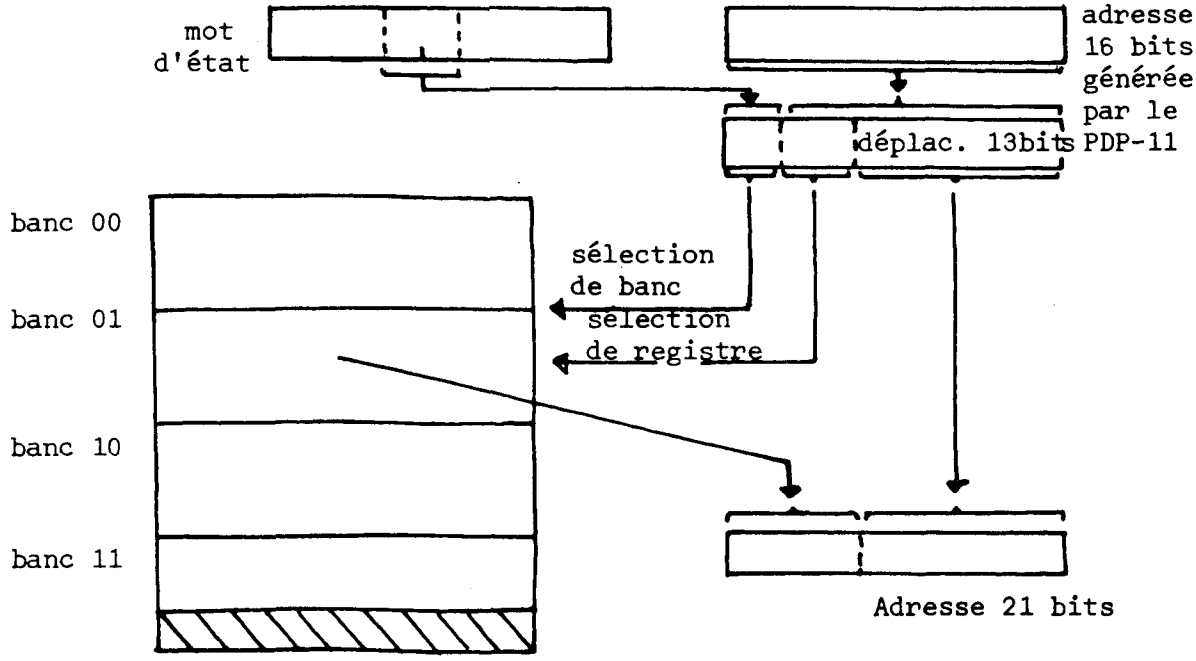
Cmmp est un multiprocesseur dans lequel m modules de mémoire primaire Mp (65 Kmots de 16 bits entrelacés sur 8 bancs ; au maximum m = 16) sont connectés à p processeurs P_c au moyen d'une matrice d'interconnexion " crossbar" Smp. Chaque processeur est, en fait, un mini-ordinateur complet (DEC PDP-11) disposant, en particulier, d'une mémoire locale de 8 Kmots. Un autre "Crossbar" SKp connecte les processeurs à des contrôleurs de périphériques :



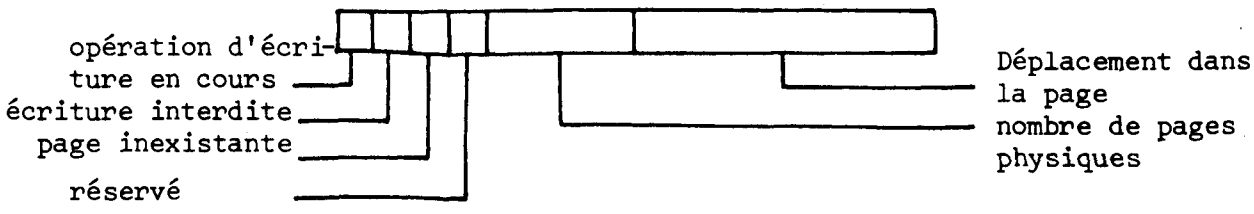
Smp et Skp peuvent être programmés par l'un des processeurs ou configurés manuellement. P communications mémoire/processeurs sont menées parallèlement.

K.clock est une horloge de forte résolution commune à tous les processeurs : elle permet de générer des noms uniques par lecture de sa valeur.

Un Dmap transforme les adresses 16 bits générées par un processeur en adresses en mémoire commune. Il contient 4 bancs de 8 registres de 16 bits. Les 13 bits de poids faibles de l'adresse générée sont considérés comme un déplacement dans une page. Les 3 bits restants sélectionnent un registre parmi 8. Le banc accédé est déterminé au moyen de la valeur de 2 bits contenus dans le mot d'état. Chaque registre renferme une adresse de page physique sur 8 bits. Ainsi, chacun des 8 blocs de 4 Kmots (8 K octets) "vus" par le PDP-11 est appliqué dans une mémoire physique de 2^{20} mots (2^{21} octets) :



Le banc 11 est privé de 2 registres qui correspondent à la mémoire locale (1 bloc de mémoire, 1 bloc pour les entrées/sorties). Il n'existe, donc, que 30 registres dans le Dmap. Le format de chaque registre est le suivant.

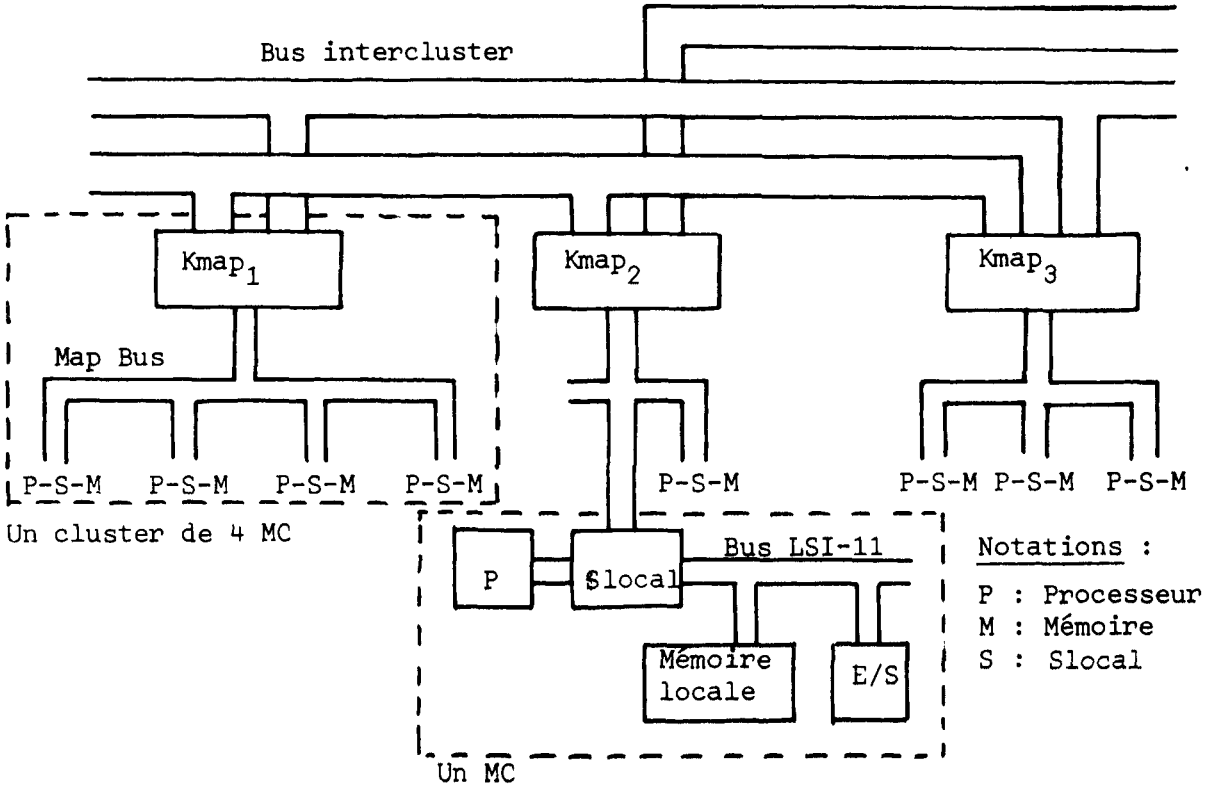


Le contenu des registres peut être modifié par certaines opérations du moniteur.

4.1.4 Cm* : un multi-processeur réparti supportant un adressage par capacités (Swan et al. [64, 65]).

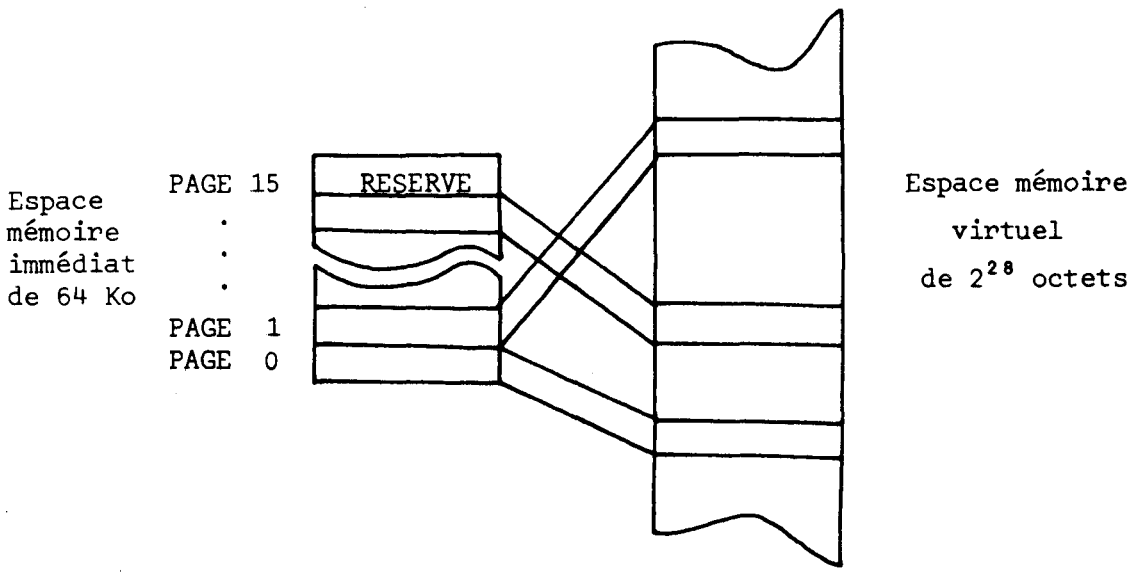
Au niveau logique, Cm* se présente comme un multi-processeur, i.e. que tout processeur a accès à la totalité de la mémoire primaire du système. Mais cette mémoire ne forme pas un vaste bloc continu. Au contraire elle est fractionnée, chaque processeur disposant d'une mémoire locale accessible aux autres processeurs, au prix d'une perte de performances pour ces derniers.

Le composant de base du système est le Module de Calcul (MC). Il est formé d'un micro-processeur - un LSI-11 - , de son bus, de la mémoire locale, d'un commutateur - noté Slocal - capable de traiter les références mémoire non locales, et des interfaces aux périphériques. Cm* est conçu pour gérer une centaine de MC. Un "cluster" regroupe entre 1 et 14 MC autour d'un "map bus". Chaque cluster est muni d'un processeur de transformation des références -le Kmap - dont une fonction est de connecter plusieurs clusters sur un "bus intercluster". La structure du système est, donc, la suivante :

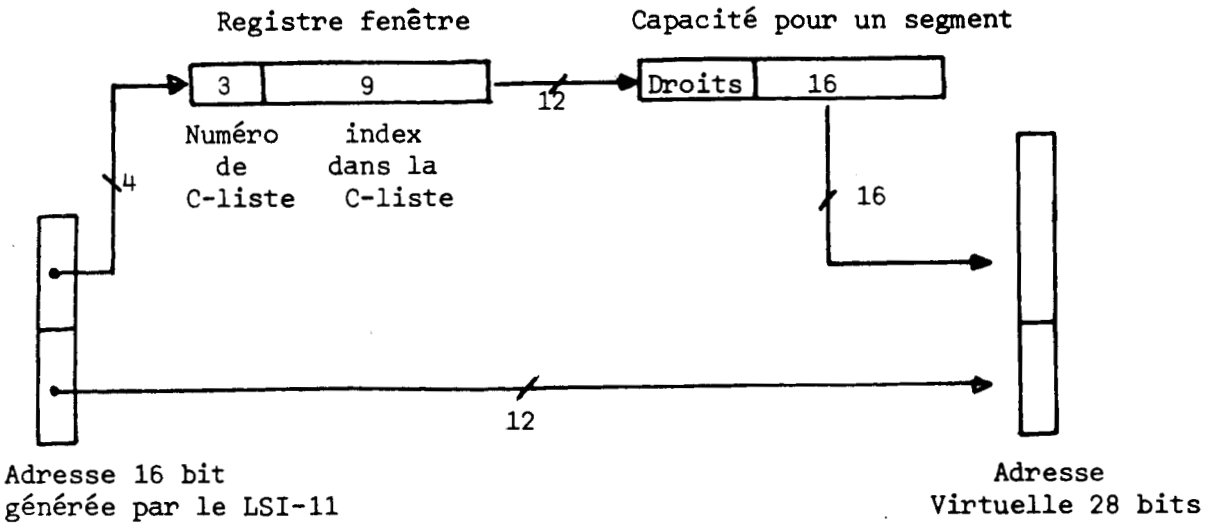


Le Kmap est un processeur spécialisé micro-programmé doté d'une mémoire de programme de 2 k mots de 80 bits. Toute référence non locale est détectée par le Slocal, qui prévient le Kmap, lequel prend ensuite en charge le transfert entre la mémoire locale et un autre MC du même cluster ou un autre cluster. Durant ce temps le LSI-11 est placé en attente. Ce processus est transparent au programmeur.

Pour l'utilisateur du système, la mémoire apparait comme un espace virtuel segmenté de 2^{28} octets, adressé par capacités. Cm* supporte jusqu'à 2^{16} segments de taille variable entre 2 octets et 4 K octets. Le processeur d'un MC "voit" un espace mémoire immédiat de 64 K octets organisé en 16 pages de 4 K octets. La page 15 est réservée au noyau et contient 15 pseudo-registres fenêtre qui définissent une application de l'espace mémoire immédiat dans l'espace virtuel :

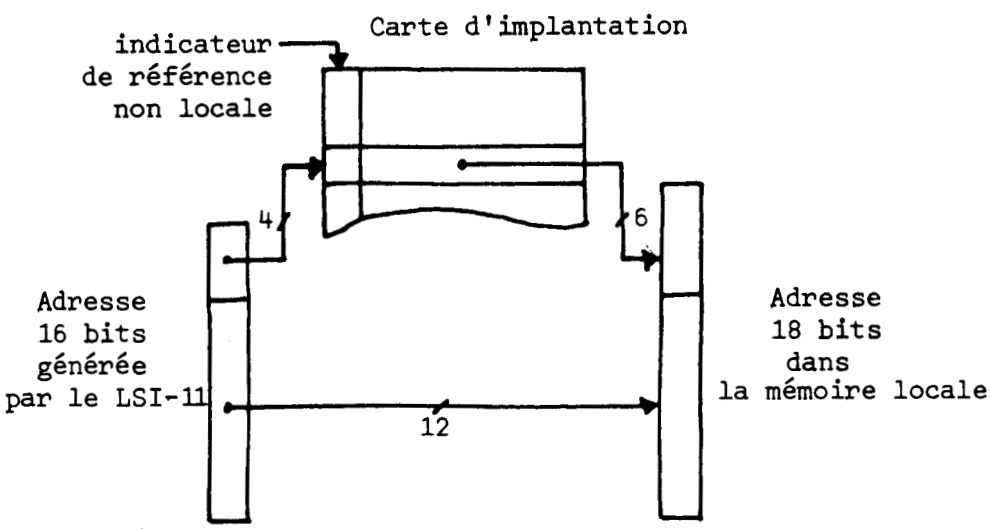


L'environnement en cours d'exécution est repésenté par un arbre de C-listes. Grâce à une convention d'adressage 8 C-listes sont accessibles. Un registre fenêtre sélectionne une capacité par un numéro de C-liste et un index dans la C-liste repérée. La concaténation du nom du segment figurant dans la capacité et du déplacement dans la page fournit une adresse virtuelle de 28 bits :



Note : Tous les chiffres sont des nombres de bits

Ensuite, l'adresse virtuelle permet d'accéder à un descripteur du segment qui contient toutes les informations localisant le segment. Le Kmap traite le cas où le descripteur du segment réside dans un cluster différent de celui qui a généré l'adresse virtuelle. Afin d'accélérer ce long calcul d'adresse, le Slocal est doté d'une carte d'implémentation de la mémoire locale : toutes les références locales sont directement traduites en adresses de 18 bits dans la mémoire locale :

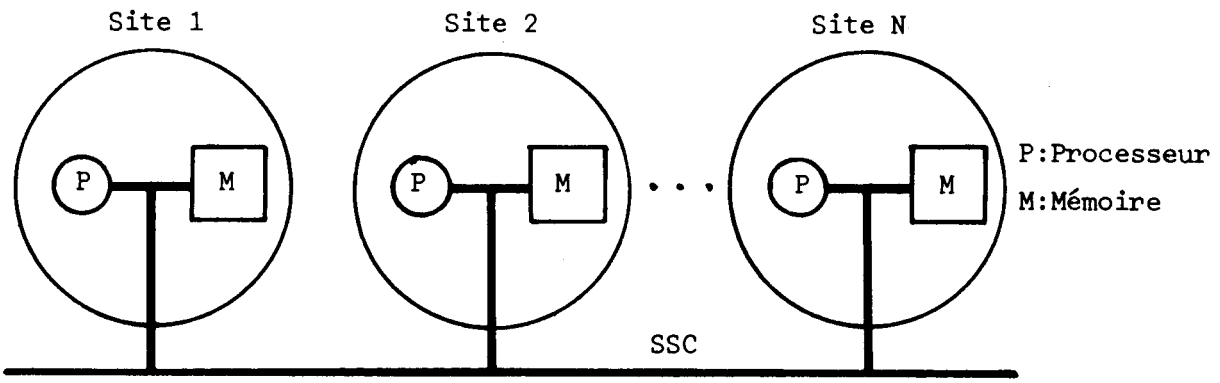


Quand l'indicateur de référence non locale est positionné, le processeur est placé en attente et le Kmap du cluster auquel il appartient effectue le calcul d'adresse.

Remarque : La description de l'architecture et de l'adressage de Cm* a été idéalisée, afin de conserver à ce paragraphe une taille raisonnable.

4.1.5 *Les travaux de Casey sur la répartition d'une structure de domaines sur un multi-ordinateur* ([4, 5]).

Casey propose d'implémenter une structure de domainessur un multi-micro-ordinateur - un réseau local de micro-ordinateurs faiblement couplés - dépourvu de mémoire commune partagée (la mémoire de masse mise à part). Les micro-ordinateurs, ou sites, communiquent par message via un Sous-Système de Communication (SSC) :



Casey envisage des systèmes comptant jusqu'à 20 sites.

Les objets et les domaines sont mobiles ; ils peuvent se déplacer d'un site à un autre. Aussi les sites sont banalisés ; ils sont capables d'exécuter n'importe quel domaine (tout au moins en première approximation). Le mouvement d'un objet est guidé par les résultats d'un calcul de choix de site dont le but est de réguler la charge globale du système ; un compromis est recherché entre l'équilibrage des charges des sites et le trafic sur le SSC.

Casey ne s'interesse pas à l'aspect protection contenu dans la notion de domaine mais à la modularisation des traitements que l'emploi des domaines permet (modularisation qui rend possible une répartition des calculs sur les sites). Aussi il ne détaille pas l'adressage physique au niveau des micro-ordinateurs, mais les mécanismes, plus généraux, de gestion des noms des objets. Il distingue les objets non partageables des objets partageables. Chacun des premiers est repéré par une capacité unique qui désigne le site qui posséde l'objet et la localisation de la représentation de l'objet dans la mémoire de ce site. Les objets partageables sont repérés par des noms dits

"globaux", qui doivent avoir un sens pour tout le système , car plusieurs sites différents peuvent posséder une capacité pour le même objet. L'accès à un objet désigné par un nom global peut recourir : à la mise à jour sur chaque site, et à chaque transfert, d'un catalogue nom de l'objet-localisation de l'objet ; à la consultation d'un catalogue central ; à la lecture d'une mémoire associative ; ou à une procédure d'interrogation des sites.

Les propositions de Casey seront analysées plus en détail dans la suite.

4.1.6 *Autres propositions*

- Au lieu de distinguer des segments de données et des segments de capacités Fabry ([13]) et Feustel ([16]) proposent de permettre la cohabitation dans un même segment de capacités et de données, les deux éléments étant différenciés par la valeur d'un bit de marque. Une capacité est alors manipulée par certaines instructions du processeur (copie de capacité, restriction des droits, décomposition, ...) de la même manière qu'une donnée. L'adressage est uniforme pour les capacités et les données, mais le bit de marque est testé pour vérifier que l'emploi d'une instruction est légitime. Et il n'est pas possible d'obtenir, par copie, une capacité à partir d'une donnée, et inversement.

- Lanciaux et Martin ([33]) définissent un adressage utilisant deux jeux de registres : les registres de données et les registres de capacités. Les seconds repèrent des segments de capacités par l'intermédiaire de noms uniques. Les premiers contiennent des index. Une instruction porte nécessairement sur deux opérandes, le premier étant un registre, et s'écrit :

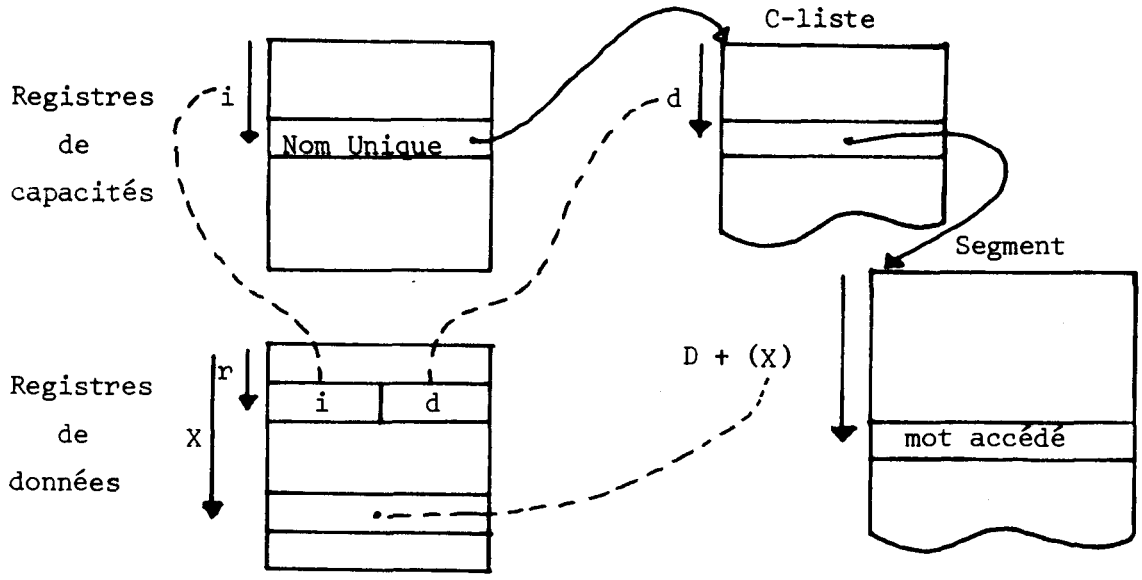
code opération, R, mode (r, D), X

avec R : numéro du registre du premier opérande
X : numéro d'un registre d'index
r : numéro du registre utilisé pour désigner le second opérande
D : adresse du second opérande dans le segment désigné par r.
mode : mode d'adressage du second opérande.

Ces dispositions permettent de varier les modes d'adressage en combinant les indirections et les accès directs par un registre de capacité ou un registre de donnée.

Exemple : Adressage par registre de donnée en mode indirect (noté $\mathcal{D}^*$) à travers une C-liste :

$$\mathcal{D}^*(r, D)\ X$$



Par ailleurs certains registres de capacités sont réservés au repérage des C-listes regroupant les objets d'une même classe (locaux, remanents, ...) qui constituent un environnement. A chaque classe d'objet correspond un mode d'adressage.

Pour accélérer les accès, d'une part, un registre de donnée est couplé à un registre de capacité et contient la valeur de la base du segment en mémoire ; d'autre part, une mémoire associative mémorise les index issus d'indirections à travers de C-listes matérialisent l'environnement en correspondance avec la base du segment en mémoire.

- Il semblerait qu'aucune des propositions ci-dessus n'ait abouti à une réalisation.

Remarque : Le Cambridge CAP Computer (Needham et al. [48 à 51]) n'a pas été cité car, comme le fait remarquer Lauer ([35]),il repose sur une philosophie totalement différente de celle qui sous-tend l'utilisation des capacités dans des systèmes comme HYDRA. Dans ces derniers les objets sont autonomes et une capacité repère un objet parmi tous les objets du système. Par contre, CAP manipule des espaces d'adressage imbriqués associés à une hiérarchie de processus. Une capacité ne rend alors accessible qu'une partie d'un segment ("refinements").

4.2 ADEQUATION DE L'ORGANISATION MATERIELLE DU SYSTEME A LA STRUCTURE DE DOMAINES

Les exemples précédents montrent qu'il est possible d'implémenter un adressage par capacités sur des machines de structures diverses, du multi-processeur au mini-ordinateur en passant par le réseau local d'ordinateurs faiblement couplés. L'opportunité de ces différentes organisations matérielles doit être évaluée vis à vis de l'application que supportera OMPHALE - un système de temps partagé - et en fonction des qualités émergentes attendues du produit fini : fiabilité, disponibilité, extensibilité.

Bien que Spier en ait démontré la faisabilité, la solution d'un mini-ordinateur est à écarter immédiatement du fait de la faiblesse de sa puissance de calcul. Plus généralement, tout système basé sur un ordinateur unique doit être rejeté : Pour un mono-processeur, le seul moyen de passer d'une puissance donnée à une puissance plus élevée est d'accroitre la vitesse des traitements, ce qui nécessite l'emploi de technologies de plus en plus sophistiquées. Les "grands" ordinateurs bénéficient, certes, des progrès d'intégration, de vitesse et de coût des technologies de circuits LSI. Mais cette évolution a un impact beaucoup plus important sur les caractéristiques des micro-ordinateurs, à cause de la masse de leur production et de leur commercialisation. Aussi il parait préférable de bâtir un système de puissance moyenne à partir des micro-processeurs commercialement disponibles, plutôt que d'entreprendre la conception d'un processeur unique. Il faut, alors, s'interroger sur les propriétés des structures de micro-processeurs interconnectés.

Deux organisations matérielles sont envisageables : la première, le multi-processeur, permet à plusieurs processeurs de communiquer au travers d'une mémoire primaire commune et partagée. Ses propriétés sont examinées ci-après. L'autre regroupe plusieurs ordinateurs en un réseau. Elle est parfois appellée multi-ordinateur. Les processeurs échangent, alors, de l'information par envoi de messages.

4.2.1 Propriétés des multi-processeurs

Du partage de m bancs mémoire entre n processeurs au travers d'une circuiterie d'interconnexion découlent à la fois les avantages et inconvénients des multi-processeurs. Trois organisations sont distinguées, qui emploient respectivement, des mémoires "multi-port" et des bus multiples, ou une matrice de connexion "crossbar", ou encore un bus commun unique et multiplexé, comme matériel de connexion (Enslow [12], Swan et al. [64]). Le tableau de la page suivante, qui résume leurs propriétés, indique :

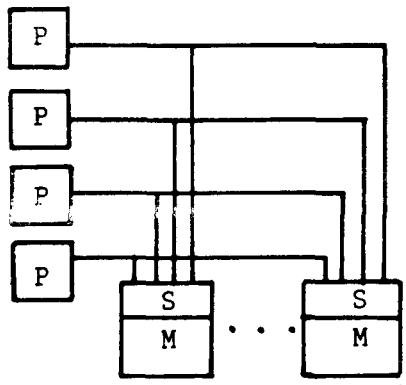
- que les solutions "Multi-bus"/"Multi-port" et matrice "crossbar", complexes dès le départ, paient une amélioration de leur disponibilité et/ou de leur extensibilité par un accroissement notable de cette complexité,

- Que la solution bus commun est facilement extensible dans la limite de la capacité du bus et présente une bonne disponibilité, si le bus est passif. La faiblesse de ses performances peut être compensée par l'ajout de mémoires locales aux processeurs.

Par ailleurs, quel que soit la circuiterie de connexion, les conflits d'accès aux bancs mémoire obligent à distribuer les adresses consécutives sur des blocs de mémoire distincts. Ce qui accroit la complexité de la circuiterie d'accès et conduit à une sous-utilisation des modules d'autant plus prononcée que le temps d'accès à un mot mémoire se rapproche du temps consommé par un processeur pour le traiter. De plus, l'exclusion mutuelle à certaines données nécessite la mise en place de verrous sur le bus unique de sémaphores, etc ... qui entrainent la suspension d'un processeur quand il essaie d'accéder à une donnée déjà utilisée par un autre processeur.

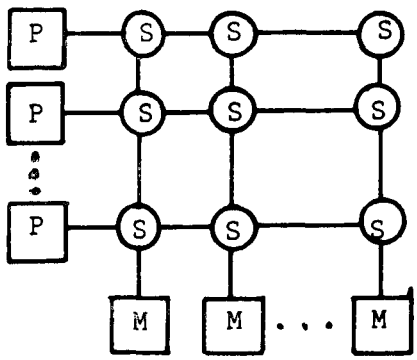
Finalement, un multi-processeur procure deux avantages ; un taux de transfert élevé et une mémoire de grande taille. Le premier doit être tempéré par le fait qu'une partie de son bénéfice est consommé par la gestion de la

MEMOIRES "MULTI-PORTS" / BUS MULTIPLES



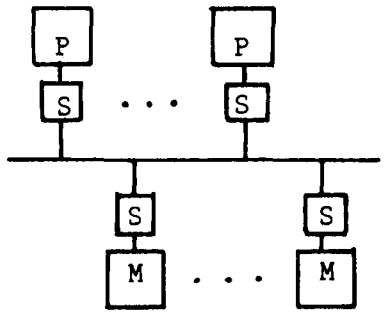
Complexité (nombre de liaisons) : n x m
Parallèlisme (nombre de transferts simultanés possibles) : n
Taux de transfert : le plus élevé (interconnection complète).
Extensibilité : limitée par le nombre et les caractéristiques des blocs mémoire.
Disponibilité : accrue par une distribution des commutateurs (Pluribus).
Améliorations : - distribution des commutateurs accroit l'extensibilité.
- multiplexage de 2 processeurs sur une même liaison.

MATRICE DE COMMUTATION "CROSSBAR"



Complexité : n x m.
Parallèlisme : n
Taux de transfert : grevé par les délais de propagation dans la matrice.
Extensibilité : limitée à la conception par la taille maximale de la matrice.
Disponibilité : dépend de la fiabilité des commutateurs
améliorations : rendre le commutateur passif, utiliser une redondance, partitionner la matrice.

BUS COMMUN UNIQUE MULTIPLEXE



Complexité : n + m
Parallèlisme : 1
Taux de transfert : le plus faible.
Extensibilité : aisée ; dégrade les performances, limitée par la capacité du bus.
Disponibilité : dépend de la fiabilité du bus. Amélioration : bus passif.
Améliorations : -mémoire cache entre bus et processeur
- mémoire locale
- mémoire locale accessible des autres processeurs (Cm*)

BUS LILLE

Propriétés des multiprocesseurs

Notations :

P : processeur
M : mémoire
S : commutateur

n : nombre de processeurs
m ; nombre de mémoires.

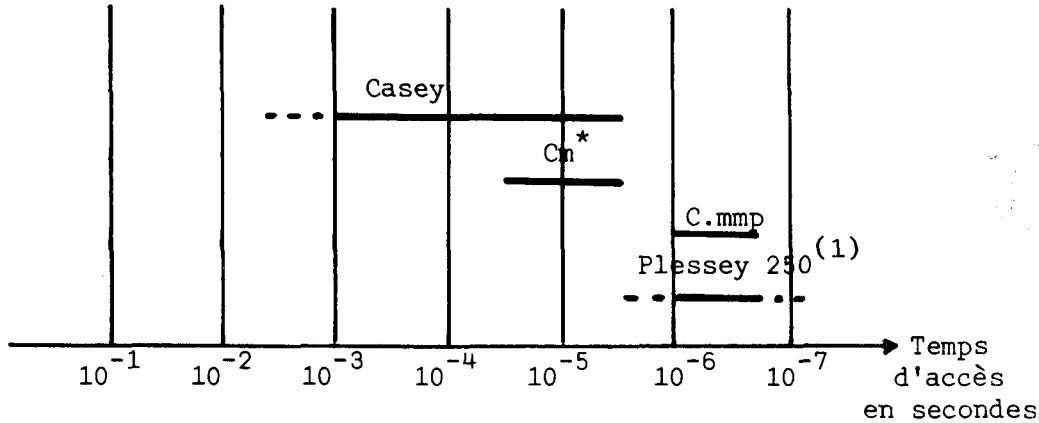
mémoire commune elle-même ou le traitement de certaines données qui, dans un multi-ordinateur, n'entrainerait pas de communications (certaines tables du système, files d'attente, ...). Le second avantage facilite la gestion de la mémoire.

En ce qui concerne les multi-ordinateurs, il est insuffisant de simplement énumérer les qualités de leur organisation matérielle. Car si plusieurs réalisations montrent qu'il est possible d'implanter une structure de domaines sur un multi-processeur, seuls les travaux de Casey concernent les réseaux d'ordinateurs. Et la faisabilité de cette dernière solution ne semble assurée que dans certaines conditions.

4.2.2 Implémentation d'une structure de domaines sur un réseau d'ordinateurs : étude de faisabilité

* *Position du problème : couplage entre processeurs*

Les architectures regroupant plusieurs processeurs sont, généralement, caractérisées par le degré de **couplage** existant entre les processeurs. Sur un axe d'évaluation des couplages, les multi-processeurs, dans lesquels des processeurs partagent une mémoire primaire commune, sont placés à l'extrémité représentant les couplages forts, tandis que les réseaux d'ordinateurs communiquant par des lignes série à faible débit, occupent l'autre bout, celui des couplages faibles (le cas des machines à ALU multiples n'est pas considéré). Pour graduer cet axe Fuller et al. ([17]) proposent de mesurer le couplage par le temps d'accès à une structure de données globale, dans le cas le plus défavorable. Cet exercice, appliqué aux systèmes décrits au paragraphe 4.1, donne les résultats suivants :



(1) = temps d'accès estimé

Par ailleurs, le temps moyen séparant les interactions de deux processeurs exécutant des processus différents est caractéristique de l'application supportée par la machine. Aussi, la comparaison de cette dernière donnée avec le plus mauvais temps d'accès informe sur l'adaptation de la structure matérielle aux programmes exécutés (Fuller et al.). Par exemple, il est évident que deux processus communiquant en moyenne toutes les secondes ne nécessitent pas une exécution sur un multi-processeur dont le plus mauvais temps d'accès sera 10^{-5} seconde, mais se contentera d'un réseau d'ordinateurs. A contrario, dans le Plessey 250, conçu pour gérer des télécommunications en temps réel, les interactions entre processeurs sont fréquentes et il est impossible d'utiliser un réseau d'ordinateurs faiblement couplés.

Le système OMPHALE est destiné à l'enseignement des méthodes de programmation modulaire et structurée, en temps partagé. Il faut donc examiner la fréquence des interactions qu'entraine ce genre d'application. Une remarque préliminaire s'impose : Le logiciel étant, par hypothèse, découpé en modules, la plupart des références générées par l'exécution d'un domaine sont internes au domaine (ce qui est une conséquence du principe de localité des références d'un programme). Ainsi, par exemple, dans le cas de Cm* (Fuller et al.), 90 à 99 % des références à la mémoire sont formées par des accès à des instructions ou à des données temporaires. Aussi, il devient immédiat de placer un domaine entièrement dans la mémoire locale d'un processeur, dans le cas d'un fractionnement de la mémoire, plutôt que de répartir ses constituants dans plusieurs mémoires disjointes. Compte tenu de cette remarque, la fréquence des interactions entre processeurs égale, alors, la fréquence des interactions entre domaines résidant dans les mémoires d'ordinateurs différents (et cela rend la question posée non triviale - les réseaux supportant un système de temps partagé sont courants : la petitesse des domaines accroit la fréquence des interactions entre eux).

Des données décrivant le comportement des usagers de temps partagé sont disponibles (voir, par exemple, Leroudier [37]), et les résultats globaux suivants sont communément admis :

- le temps partagé alterne les périodes de réflexion de l'utilisateur, d'une durée de l'ordre de 30 secondes, et les périodes de quelques secondes constituées par les réponses du système.

- les exécutions des programmes des usagers ne constituent, au plus, qu'un quart des commandes adressées au système et ne consomment, au maximum, que la moitié du temps de calcul. Aussi, toutes les autres commandes (les trois quarts) entraînent l'exécution d'utilitaires (analyseur de commandes, compilateur, éditeur de liens, éditeur de texte, système de gestion de fichiers, ...).

Dans le contexte de l'utilisation du temps partagé pour l'enseignement de la programmation, un allongement du temps de reflexion, une baisse de la fréquence et de la durée des exécutions, sont constatés (dus, en partie, à l'augmentation du nombre des erreurs). Par ailleurs, quand le temps partagé est implanté sur un ordinateur unique multiprogrammé, le degré de multiprogrammation est faible (Betourné et Krakowiak [2]). Aussi, au niveau de l'application, les interactions entre les programmes qui la constituent sont faibles : une session est constitué par l'appel en séquence d'utilitaires et, à un instant donné, il y a peu de chances pour que deux utilisateurs emploient les mêmes programmes (pour simplifier, il est supposé que le système n'effectue pas de copie des programmes).

Cependant, il a déjà été remarqué qu'un programme ne se présente pas comme un tout monolithique, mais qu'il est finement découpé en domaines. Aussi, même si au niveau de l'application la fréquence des interactions se révèle faible (ce qui est nécessaire pour qu'elle puisse rester faible entre les domaines), il se peut que, dans la réalité, au niveau de la structure de domaines, elle soit beaucoup plus importante. Tout dépend de la répartition, entre les mémoires des ordinateurs, des domaines réalisant un même programme. Si il est impossible de regrouper dans une même mémoire une partie des domaines participant à un traitement, il en résultera une prolifération des interactions entre processeurs, jusqu'à rendre le réseau incapable de supporter le système de temps partagé.

Finalement, les conclusions relatives à la faisabilité d'une solution multi-ordinateur sont subordonnées aux résultats d'une étude de la répartition des domaines.

Les paragraphes suivants emploient la terminologie issue des travaux de Casey ([4, 5]) : Le reseau est constitué de **sites** - processeur + mémoire locale + interfaces aux périphériques - connectés à un **Sous-Système de Communication** (SSC).

* ***Banalisation ou spécialisation fonctionnelle des sites***

Une première façon de répartir les domaines sur les sites consiste à **spécialiser les sites**, i.e. à implanter de manière statique dans la mémoire locale d'un site tous les domaines qui réalisent un même programme, une même fonction. Ainsi, le sytème de temps partagé est composé, par exemple, d'un site compilateur, d'un ou plusieurs sites d'exécution, d'un site gérant les fichiers, etc ... Des considérations relatives au temps de réponse du système et sa disponibilité conduisent à rejetter cette solution, malgré sa simplicité, malgré la réponse positive à la question initialement posée (est-il possible, dans ce cas, d'implémenter un sytème de temps partégé sur un multi-ordinateur ?) qu'elle permet.

Sans entrer dans le détail de sa formulation mathématique, la théorie des files d'attente enseigne que, pour un système disposant de plusieurs serveurs, le meilleur temps de réponse est obtenu en ne spécialisant pas les serveurs, à condition que l'équilibre des charges des serveurs, ou mieux leur égalisation, soit recherchée (Casey [4]). Ces deux objectifs diffèrent par le mode d'assignation d'une demande à un serveur. Pour atteindre l'égalisation, il doit être possible de déplacer instantanément, une demande d'une file d'attente à une autre file plus courte, dès qu'une file apparait comme telle. Par contre, un système où une demande est aiguillée vers la file d'attente la plus courte - dans laquelle elle est sensée être la plus rapidement servie - et où elle subsiste jusqu'à sa satisfaction, ne peut être qu'équilibré. L'inconvénient de l'égalisation est qu'elle conduit à d'incessants transferts de demandes entre les sites, si bien que l'équilibre des charges lui sera préféré, malgré la perte de performances que ce choix entraine. Puis un **système hétérogène**, i.e. un système constitué de sites spécialisés supporte mal les fluctuations de charge. Une répartition aléatoire des fonctions sur les sites, sans aucune copie d'une même fonction, fait apparaitre des goulots d'étranglement dans le système : il se crée de longues files d'attente devant les fonctions les plus chargées et, finalement, le

temps de réponse est piloté par les performances du site le plus sollicité. La multiplication des fonctions les plus chargées remédie à cet inconvénient. Mais alors le système est adapté à une charge type et son temps de réponse se dégrade pour toute requête atypique. Enfin, toute extension de la configuration (par exemple : ajout de consoles ou d'un disque) devient problématique du fait des boulversements des caractéristiques de la charge qu'elle entraîne.

En ce qui concerne la disponibilité, les performances d'un système hétérogène se dégradent brutalement lors de la panne de l'un des sites qui le composent, à moins d'avoir prévu une coûteuse redondance de toutes ses fonctions. Par contre, un **système homogène**, i.e. un système où chaque **site "banalisé"** est capable d'exécuter n'importe quel domaine, n'importe qu'elle fonction, peut être conçu de manière à ce que la panne d'un site (ou du moins certaines pannes "graves" aisément détectables et localisables) conduise à sa déconnection du système, tandis que les traitements se poursuivent sur les autres sites avec une perte de performances.

Ainsi, un examen plus détaillé des possibilités de répartition des domaines sur un système homogène - il est maintenant acquis qu'il n'est pas souhaitable que la configuration soit formée de sites spécialisés - est nécessaire à l'énoncé d'une conclusion quant à l'adéquation de l'organisation matérielle du système à sa structure logicielle.

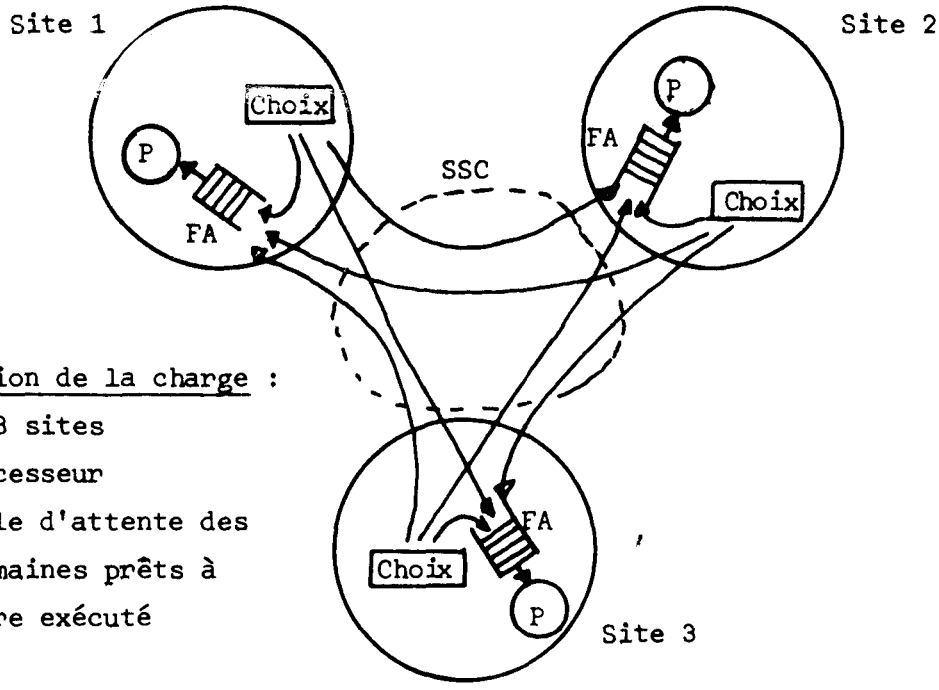
* *Régulation de la charge dans un système homogène (Casey [4, 5])*.

Le choix d'un système homogène n'est justifié que si un équilibre de la charge est assuré. Il faut alors s'interroger, en reprenant la terminologie de la théorie des files d'attente, sur la nature d'une demande. Le chapitre 2 a fait du domaine l'élément atomique du système. Aussi, la satisfaction d'une demande inclue l'exécution d'au moins un domaine. Mais, il est envisageable de regrouper les domaines et de les exécuter sur un site, ce qui réduit les interactions entre processeurs. L'inconvénient de cette proposition est de recouvrir le premier découpage en domaines (lui-même conséquence d'un découpage en modules) par un deuxième niveau de modularisation. Il parait difficile de demander au programmeur d'effectuer cette opération fort éloignée du problème qu'il a à résoudre (contrairement à la modularisation selon Parnas, par exemple ; voir chapitre 1). Et les moyens

de regrouper automatiquement les domaines en "super-domaines" manquent ; il faut donc se résoudre à identifier une demande à une exécution de domaine. Par contre, il est possible de forcer à s'exécuter sur le même site des domaines interagissant fréquemment, et ce, dynamiquement.

D'un autre point de vue, l'objectif d'équilibre des charges des sites impose de rendre les domaines **mobiles** entre les sites ; comme il n'existe pas de mémoire commune aux sites, le domaine - la demande - doit être physiquement déplacé vers la mémoire du site - le serveur. Or tout en étant l'élément atomique du système le domaine est composé de diverses particules : les objets paramètres, les objets locaux, etc ... Aussi un domaine peut être incomplet au moment d'être exécuté (exemple ; un objet paramètre fait partie du domaine qui a appelé le domaine considéré, mais les deux domaines résident sur des sites différents). Les objets du système doivent, donc être, eux aussi, individuellement mobiles, et, avant l'exécution d'un domaine, ses composants éparpillés sur les sites doivent être rassemblés. Par ailleurs, il faut insister sur le fait que les sites constituent un réseau. Et les transferts de domaines, d'objets chargent les liaisons entre les sites, le SSC, de manière antagoniste à la précision de l'équilibre des charges des sites (plus les sites doivent être équilibrés, plus il faut déplacer d'objets, et inversement). Or il est nécessaire de charger le moins possible le SSC (ou, ce qui revient au même, de réduire les interactions entre processeurs), ce qui implique de rechercher un compromis, appelé par Casey **régulation de la charge**, entre, d'une part, l'équilibre des charges des sites et, d'autre part, la charge du SSC. Finalement, le schéma de satisfaction d'une demande est le suivant (voir figure page suivante). Il fait apparaitre une fonction de **choix de site** qui aiguille un domaine à exécuter vers le site réalisant le mieux le compromis de régulation de la charge.

Ainsi, pour conclure, il est possible de réduire autant que nécessaire les interactions entre les sites en biaisant le mécanisme de régulation de la charge en fonction de la fréquence des interactions entre domaines, ceci afin de rassembler sur un même site les domaines qui s'entre-appellent souvent. Ce résultat se paie par un déséquilibre de la charge des sites, donc une dégradation du temps de réponse du système.



Régulation de la charge :
Cas de 3 sites
P : processeur
FA : file d'attente des domaines prêts à être exécuté

Remarques

1) Il est fort coûteux de banaliser intégralement les sites : certains objets relatifs à la gestion des périphériques (par exemple, les segments contenant les mots de commande ou de donnée échangés entre le système et les périphériques) doivent être liés à certains sites, particuliers ; aussi rendre ces objets mobiles nécessite qu'un périphérique donné puisse communiquer avec n'importe quel site, ce qui accroit, bien inutilement, et parfois intensément (cas d'un disque) les communications entre les sites. Ce point est repris au paragraphe 5.4.1.

2) L'absence de mémoire commune rend difficile une égalisation des charges des processeurs ; le transfert d'une demande d'un serveur à un autre nécessite la communication physique d'un domaine d'une mémoire à une autre. La multiplication de ces transferts augmente le traffic sur le SSC et accroit les difficultés de gestion des mémoires locales.

3) La comparaison multi-processeur/multi-ordinateur peut être complétée par une étude des "overheads" produits par les deux solutions :

- "overhead" temporel : l'overhead nécessaire pour gérer une file d'attente croit plus vite que proportionnellement à la longueur de la file. Ce qui avantage les multi-ordinateurs par rapport aux multi-processeurs, puisque les files d'attente sont partagées entre plusieurs sites. Cependant,

la décentralisation introduit des "overheads" de communication.

- "overhead" spatial : la mémoire d'un multi-processeur renferme une copie unique du système d'exploitation. Par contre dans un multi-ordinateur, chaque site posséde une copie du système, ce qui entraîne une perte de mémoire proprotionnelle au nombre des sites. Cet inconvénient est réduit si les programmes constituant le système opératoire sont partagés en séquence entre les ordinateurs (ce que la structure de domaine et la faible fréquence des interactions entre utilitaires permettent). Cependant, chaque site possède au moins une copie du noyau.

Pour conclure les paragraphes précédents ont montré la complexité qu'entraine la recherche d'une bonne disponibilité et d'une bonne extensibilité sur un multiprocesseur, puis la faisabilité de l'emploi d'un réseau d'ordinateurs. Aussi, pour être adoptée, la solution d'un multi-processeur devrait présenter des avantages déterminants. Or, il n'en est rien : dans la comparaison des multi-processeurs et des multi-ordinateurs, avantages et inconvénients des uns et des autres se compensent. OMPHALE sera donc implanté sur un multi-ordinateur qui sera, vraisemblablement, d'une mise en oeuvre moins complexe qu'un multi-processeur.

4.3 STRUCTURE D'UN SITE

Le choix d'un multi-ordinateur ne contribue en rien à l'établissement de la protection dans le système. Or, cette protection repose sur des mécanismes cablés incorporés dans les chemins d'adressage des processeurs et dans l'organisation des mémoires. Aussi convient-il de décrire la structure d'un site d'OMPHALE.

4.3.1 *Organisation de la mémoire*

La sauvegarde de l'intégrité des capacités (§ 2.1.3, "Intégrité des capacités") impose de protéger les capacités elles-mêmes des actions d'un programme malveillant. Aussi, l'accès à la représentation physique en mémoire des capacités est contrôlé et n'est accordé qu'aux routines du noyau. Ce contrôle peut s'appuyer sur deux organisations différentes de la mémoire : un **marquage de la mémoire** ou bien une **partition de la mémoire**

(Redell [58], Fabry [13]). Une mémoire est marquée ("tagged") quand un bit servant à distinguer les capacités des données est ajouté à chacun de ses mots. Des dispositifs matériels simples se chargent alors de la consultation et de la mise à jour de la marque. Ou bien la mémoire peut être partitionnée en segments de données et en segments de capacités, ces derniers renfermant uniquement, et eux seuls,des capacités. Le descripteur du segment contient, dans ce cas, les informations permettant de contrôler les accès.

L'ajout d'un bit de marque entraine, bien évidemment, un accroissement de 1/16 de la taille de la mémoire,supposée composée de mots de 16 bits. Par contre, la partition de la mémoire répartit l'"overhead", tant spatial que temporel, d'identification de la nature des informations sur tout un segment.

Un autre avantage de la partition mémoire est de fournir des formats différents pour les capacités et les données. Il faut remarquer que les représentations des capacités sont des chaînes d'une cinquantaine de bits qui doivent être rangées sur plusieurs mots mémoire, ce qui pose le problème de la reconnaissance du premier mot constituant la capacité. Travaillant sur une mémoire marquée, Redell suggère, pourtant, des solutions diverses (il montre aussi que le bit de marque peut être simulé).

L'inconvénient majeur de la partition est qu'elle se révèle peu "naturelle" : pour une structure de données complexe, les capacités, matérialisant les aspects pointeurs de la structure, et les données doivent être arbitrairement stockés dans des segments différents. Une mémoire marquée permet, par contre, de mélanger dans un même segment données et capacités.

Cependant, la partition présente un avantage décisif : lors de l'appel d'un domaine, les capacités renfermées dans la C-liste du domaine sont consultées pour recueillir diverses informations utiles au choix de site ; si la mémoire est marquée, tous les segments constituant le domaine doivent être balayés pour retrouver les capacités. Cette recherche arborescente est inacceptable, compte tenu de l'inefficacité dont elle greve l'opération d'activation de domaine (voir § 2,5). Elle est limitée aux seules C-listes dans le cas de la partition.

Les mémoires locales des sites d'OMPHALE sont donc partitionnées entre segments de données et segments de capacités.

4.3.2 *Chemins d'adressage*

L'étude des chemins d'adressage amène à distinguer :

- l'ensemble des objets d'un domaine
- les espaces logiques "vus" par le processeur du site
- l'espace mémoire physique

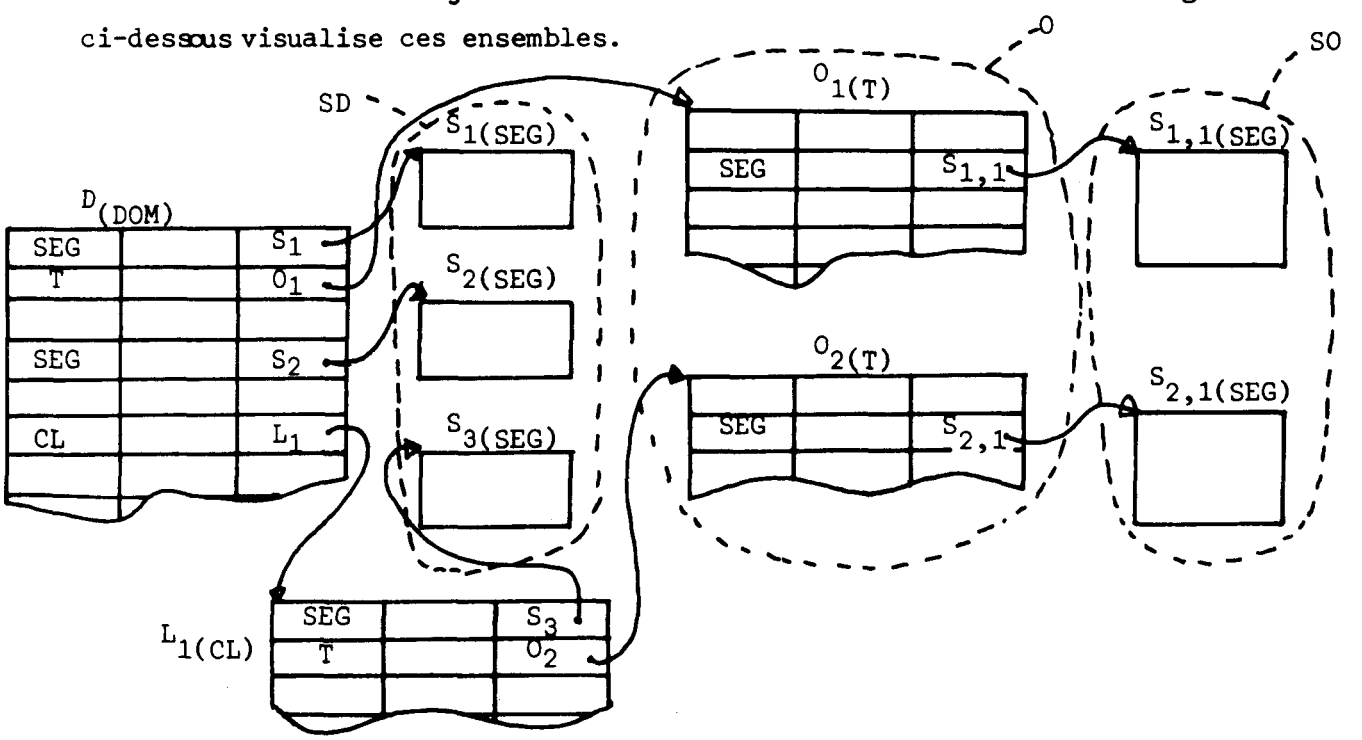
et à définir les passages d'une entité à une autre.

* *Ensemble des objets d'un domaine*

Soit un domaine D réalisant une opération du type T, **L'ensemble des objets du domaine** D est constitué :

- du domaine D lui-même
- du sous-ensemble O des objets $O_{i(T)}$ de type T référencés dans D.
- du sous-ensemble SD des segments S_j référencés dans D.
- du sous-ensemble SO des segments $S_{i,k}$ composants des objets $O_{i(T)}$ référencés dans D.

L'ensemble des segments du domaine D est l'union SD ∪ SO. La figure ci-dessous visualise ces ensembles.



Au cours de son exécution, un domaine ne peut manipuler que les segments de SD ∪ SO, la C-liste qui le représente et les C-listes, soit référencées dans cette première C-liste, soit constituant les représentations des objets de l'ensemble O, ces dernières à condition que les capacités aient été décomposées. Ces limitations proviennent de ce qu'un domaine réalisant une opération d'un type T ne peut décomposer que les objets du type T. En fait l'ensemble des objets d'un domaine doit être défini plus largement. Il est des cas où un domaine est autorisé à manipuler des objets de plusieurs types différents. Par exemple, un domaine manipulant des objets du type T1 construit par récursivité croisée - T_1 est construit sur T_2, qui est, lui-même, construit sur T_1 - peut accéder à la fois aux objets de type T_1 et aux objets de type T_2. Aussi, le sous-ensemble O doit-il être défini comme regroupant les objets $O_{i(T_j)}$ référencés dans D, les types T_j étant les types manipulables par D.

L'ensemble des objets d'un domaine est relié à l'espace mémoire physique par les mécanismes de gestion des noms (décrits au § 5.2), qui du nom unique contenu dans le champ représentation d'une capacité (voir § 2.1.3, "Définition") ménent à la représentation de l'objet repéré.

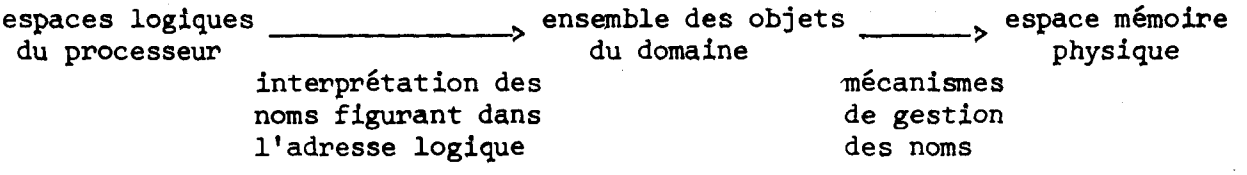
Remarque : Du fait de l'autonomie des objets par rapport aux domaines (voir § 2.1.4), les C-listes de l'ensemble O ne peuvent pas être incluses dans la C-liste du domaine qui les référence. Aussi existe-t-il deux niveaux d'adressage pour former les noms locaux (§ 2.1.3, "Compléments sur l'adressage"). Il était possible de permettre la multiplication des niveaux, soit en fournissant une primitive "walk" qui parcourt les capacités intermédiaires (HYDRA), soit en adoptant des conventions sur les valeurs des indices dans les C-listes (CM^*). Mais ces calculs peuvent être effectués par les compilateurs. Et les C-listes intermédiaires peuvent être supprimées, car les composants intermédiaires n'ont pas à être protégés et ne sont pas partagés (Lanciaux [34]).

* *Espaces logiques "vus" par le processeur du site*

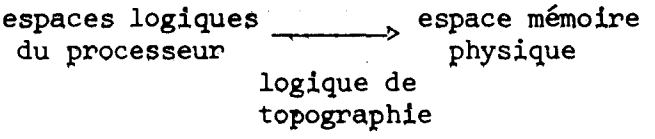
La mémoire étant segmentée, l'adresse issue d'un processeur, appellée **adresse logique**, est interprétée comme la concaténation d'un **nom de segment** et d'un **déplacement** dans le segment nommé. De plus, généralement, un processeur a la possibilité d'accéder à la mémoire selon des modes distincts, par

exemple, en mode normal ou en mode système. Aussi un processeur "voit" plusieurs espaces d'adressage logiques, à raison d'un par mode d'accès.

Bien que concevable, le chemin d'accès à la mémoire



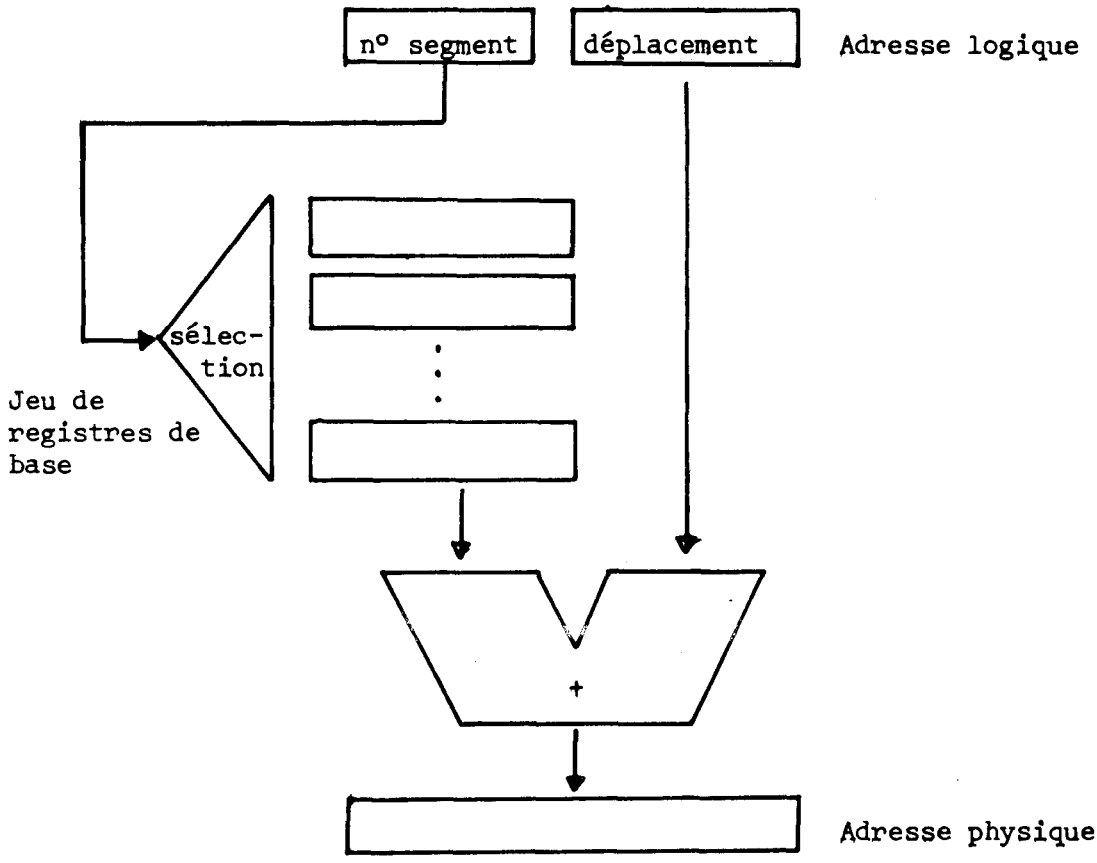
est, bien évidemment, inefficace. Aussi, une **logique de topographie** permet d'accéder aux segments du domaine selon le chemin



Du point de vue de la protection, ce schéma est légitime si, à un instant donné, et lors de l'exécution d'un domaine, la fonction de topographie est telle que l'espace mémoire physique accessible à partir des espaces logiques du processeur est inclus dans l'ensemble des segments du domaine considéré.

* *Logique de topographie*

Les principaux composants de la logique de topographie sont un **jeu de registres de base** et un additionneur d'adresses. Le nom de segment figurant dans l'adresse logique générée par le processeur sélectionne un registre. Cette sélection est soit une simple indexation (cas d'une gestion de la mémoire par zones), soit une recherche associative (cas d'une gestion de la mémoire par pages). Dans un registre de base figure une adresse de base en mémoire qui est additionnée au déplacement pour produire une adresse physique :



Un **verrou d'accès** , mémorisé lui aussi dans le registre de base contrôle la validité de l'accès par comparaison du verrou avec la clé formée des signaux d'écriture/lecture et/ou les signaux d'état - mode normal/système - générés par le processeur. Tout échec déclenche un **déroutement.**

La logique de topographie doit être chargée à partir des C-listes représentant le domaine en cours d'exécution. Ce chargement peut être automatique, mais alors il est nécessaire d'énoncer des conventions sur l'organisation des C-listes. Ou le chargement peut être déclenché par l'exécution d'instructions de chargement, solution plus souple, mais qui oblige à sauvegarder l'état de la logique à chaque suspension du domaine.

* *Choix d'un micro-processeur*

Les micro-processeurs "classiques" 8 bits sont incapables de supporter un adressage segmenté protégé. Non qu'il soit impossible de disposer sur leur bus d'adresses un jeu de registres de base ; cela a déjà été réalisé (voir, par exemple, la réalisation d'un multi-processeur à partir de micro-processeurs INTEL 8080 par Hoener et Roehder [25]). Mais le cas du déroutement ne

peut être traité que comme une interruption. Par conséquent, l'accès erroné est effectué puisque le déroulement du cycle du processeur ne sera perturbé qu'à la fin du traitement de l'instruction qui a déclenché le déroutement. Si il est facile d'inhiber, à n'importe quel moment du cycle d'exécution, une écriture en mémoire, par contre, lors d'une lecture, il faut fournir une donnée au processeur, donnée qui a de fortes chances de modifier les registres et/ou l'état du processeur. Ainsi, ce n'est pas la détection des violations de protection qui pose problème, mais leur prise en compte.

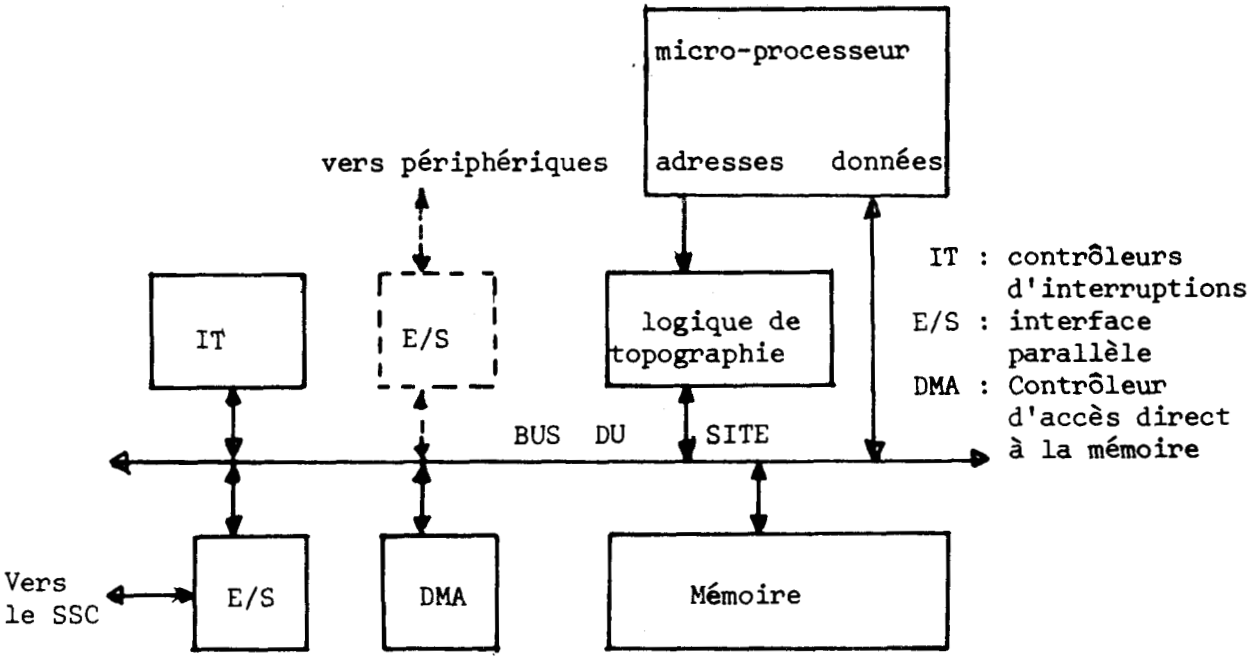
Parmi les récents micro-processeurs 16 bits, le 8086 d'INTEL ([26]) doit être écarté. Il comporte, certes, un adressage segmenté, mais sa protection n'est pas prévue. De plus, il n'est pas possible de dérouter le processeur en cours d'exécution. Ces limitations proviennent de l'organisation même du micro-processeur, conçue pour maintenir une forte compatibilité avec les 8080 et 8085 de la même firme. Aussi, le 8086 constitue l'aboutissement - la fin - d'une gamme de micro-processeurs, tandis que le Z8000 de Zilog peut être considéré comme la première machine d'une nouvelle génération qui se rapproche beaucoup de la gamme des mini-ordinateurs.

Et le Z8000 (voir l'annexe 2) possède toutes les caractéristiques nécessaires à la réalisation d'un adressage segmenté protégé. Il peut adresser 128 segments d'une taille maximale de 64 K octets. Il dispose d'un mode système et d'un mode normal. Certaines instructions privilégiées (gestion des interruptions, entrées/sorties, gestion de l'état du processeur, appel système) ne sont exécutables qu'en mode système. Le boîtier est muni d'une broche "Segment trap" qui permet de dérouter le processeur. Aussi, ce micro-processeur convient parfaitement à la réalisation d'un site d'OMPHALE.

Enfin, il semblerait que le MC 68000 de Motorola (Le Mair [44]), projet encore plus ambitieux, convienne également. Mais il ne sera pas disponible commercialement avant 1980.

4.3.3 Structure d'un site

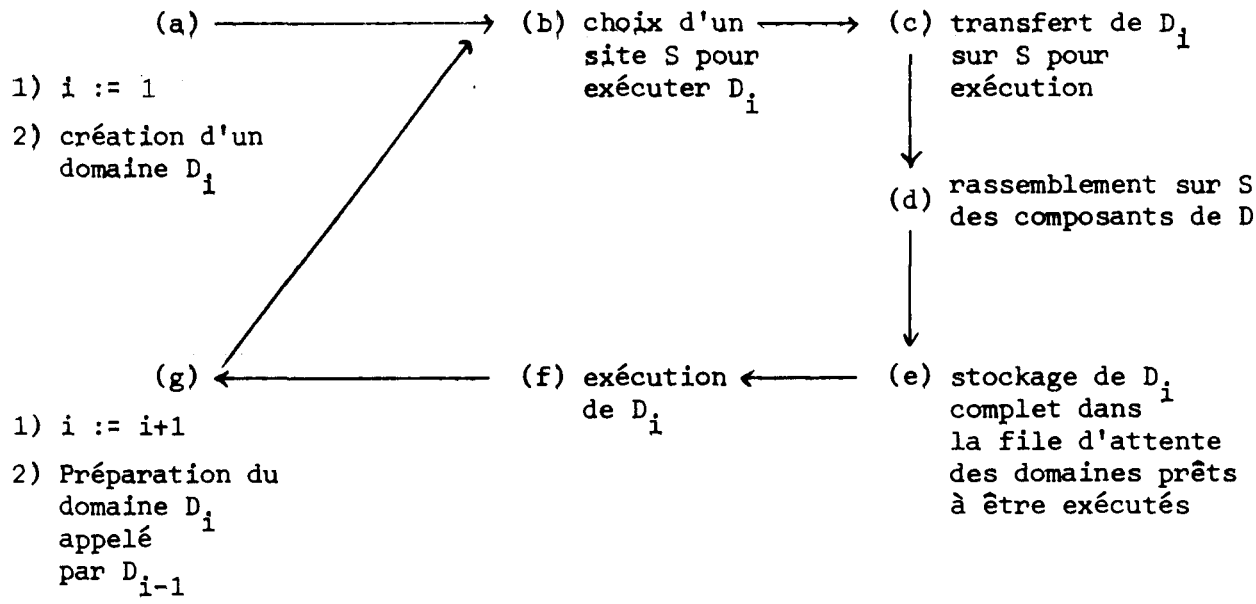
Le schéma ci-dessous résume l'organisation d'un site, telle qu'elle peut être déduite à ce stade de l'étude.



Structure provisoire d'un site

Cette organisation présente une faiblesse ; le traffic sur le SSC étant asynchrone par rapport au déroulement des traitements effectués par le processeur, chaque arrivée de message sur le site interrompt l'exécution. Et il est nécessaire de "multiprogrammer" l'exécution et le traitement des messages. Sinon deux processeurs seront placés en attente pendant toute la durée - de l'ordre de la milliseconde - d'une communication entre deux sites. Cet asynchronisme et donc inévitable.

Par ailleurs, il faut remarquer que les opérations d'activation de domaine manipulent intensément les capacités représentant un domaine, mais n'accédent peu ou pas aux segments du domaine. De plus, dans l'enchaînement suivant :

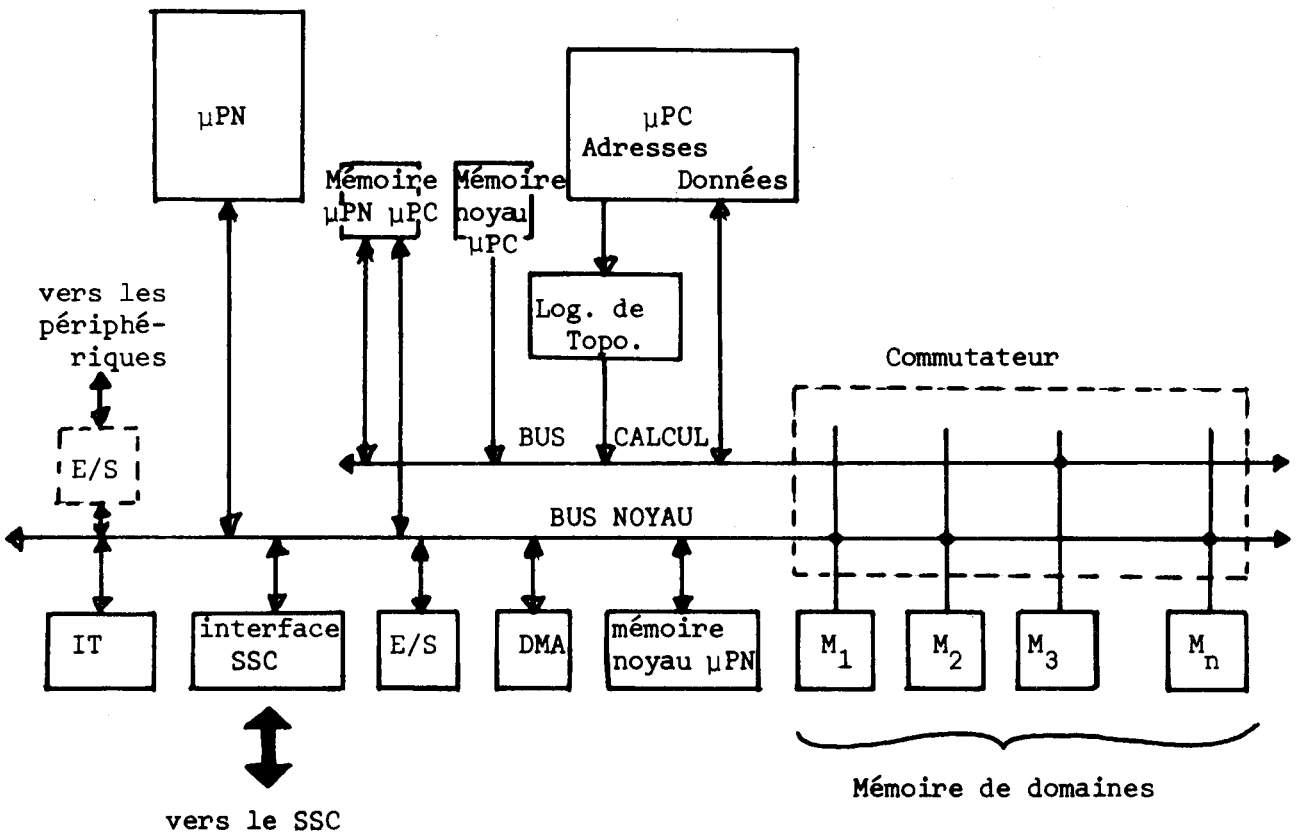


qui n'est autre que le cursus normal, mais simplifié, d'exécution des domaines (voir § 6.2.1), l'exécution d'un domaine D_i peut être menée parallèlement avec les opérations (b), (c), (d), (e) et (g) concernant d'autres domaines D_j, $j \neq i$, non encore exécutés (ou des domaines D'_k indépendants du domaine D_i) puisqu'il est certain que D_i ne les perturbera pas. D'où l'idée d'étendre la partition de la mémoire à une **partition des processeurs**, chaque site étant alors doté de deux processeurs (Z8000). L'un, le **micro-processeur de calcul**, µPC, est chargé d'exécuter "aveuglément" les domaines dans l'ordre de la file d'attente des domaines prêts. Pour ce faire il accède massivement aux segments de données. L'autre, le **micro-processeur noyau**, noté µPN, est chargé de préparer l'exécution des domaines, soit à partir de domaines exécutés par µPC, soit à partir de domaines provenant d'autres sites.

Comme le noyau est exécuté en mode système, alors que les domaines sont exécutés en mode normal, au niveau de l'assembleur toute opération sur les capacités sera représentée par une instruction d'appel système, ce qui permet à la fois, d'émuler le Z8000 qui ne dispose évidemment pas d'opérations de manipulation des capacités et de protéger les capacités en ne faisant agir sur elles que les routines du noyau supposées correctes. Aussi µPN accéde-t-il surtout aux segments de capacités et aux structures de données nucléaires. Si aucun segment de donnée n'entre dans la composition des objets du noyau, µPN n'accédera pas aux segments de données. Inversement, si µPC n'effectue aucune manipulation de capacité et s'en décharge sur µPN (ce qui est un choix à faire : µPC pourrait très bien exécuter en mode système les parties du noyau

synchrones avec l'exécution des domaines), µPC n'accédera pas aux segments de capacités. La partition peut être totale, à une file d'attente près (les domaines prêts).

La relation d'une discussion - toujours laborieuse - sur le choix d'un mode de gestion de la mémoire présente peu d'intérêts ici. Seule une organisation, possible et définitive, d'un site est présentée ;



La mémoire est organisée en n pages et les segments sont gérés par zones à l'intérieur des pages (B. et J. Leguy [36]). Chaque page renferme plusieurs domaines, chaque domaine résidant intégralement dans une page. Les domaines prêts sont distribués dans les pages de manière telle que deux domaines consécutifs dans la file d'attente soient implantés dans des blocs mémoire différents. Au moyen d'un commutateur, µPC se connecte à une page M_i à la fois (cas du bloc M_3 sur la figure), les n-1 pages restantes demeurant accessibles à µPN sans conflit possible avec µPC. Le micro-processeur de calcul traite les parties du noyau synchrones avec l'exécution des domaines. Les demandes de manipulation de capacités concernant M_i et survenant durant

la période où ce bloc est isolé sont placées en attente et traitées dans l'ordre de leur arrivée par μPN dès que μPC libère la page (ce qui survient nécessairement au bout d'un temps fini). Les transferts de segments sont effectués en DMA entre le SSC et la mémoire, suspendant l'exécution de l'un des processeurs. Enfin, une page est organisée de manière telle que μPN puisse procéder à son seul compactage.

4.4 DISCUSSION

Casey a entrepris des simulations qui montrent la faisabilité de sa proposition de répartition d'une structure de domaine sur un multi-ordinateur, sous la direction d'un mécanisme de régulation de la charge ([4]). Le comportement de systèmes fonctionnant en temps partagé et équipés, au maximum, de 9 sites, 54 consoles et 3 disques, a été simulé. Les principaux résultats sont les suivants :

- le système exhibe un comportement stable : absence d'étreinte fatale, absence de phénomène de bouclage infini ou d'oscillation des domaines à la recherche d'un site d'exécution.

- A charge constante par site, le système accroit légèrement son temps de réponse quand sa taille croit entre 1 et 9 sites.

- La bande passante du SSC requise pour un système de 9 sites est de l'ordre de 3 MHz. Et elle croit proportionnellement au nombre des sites. Extrapolant ses résultats, Casey prédit un comportement satisfaisant pour des systèmes comptant une vingtaine de sites.

Mais les nombreux chiffres fournis avec les programmes de simulation et le détail de la structure de domaines que leur rédaction nécessite, permettent d'apprécier les limites des travaux de Casey. Les faits suivants doivent être soulignés :

- les objets ne sont pas typés ; leur nombre total dans le système est limité à quelques centaines ; ils ne sont pas, en général, partageables ; leur création et leur destruction sont statiques.

- le nombre total des domaines composant le système est faible (50 ?) et fixe. La taille de leur représentation varie entre 1,5 K octets et 36 K octets ; les programmes des usagers sont interprêtés par des domaines

superviseurs ; un "processeur virtuel" (représentant le processus) est attribué à chaque usager ; le nombre de processeurs virtuels est fixe.

- Les capacités ne sont pas manipulables par l'usager, mais seulement par le noyau.

- Les seuls objets partageables sont les"domaines génériques", qui correspondent aux objets-type d'OMPHALE.

Bien que ce ne soit pas le sujet de son étude, Casey entend ne pas se priver des avantages relatifs à la fiabilité procurés par une structure de domaines. Cependant, il est à craindre que ces avantages ne puissent pas se manifester. D'une part, la grande taille de la représentation physique des objets et des domaines limite fortement leur nombre total, jusqu'à rendre le découpage du système en domaines trivial (un domaine = un compilateur, par exemple). Ce qui va à l'encontre du respect du principe du moindre privilège, qui suppose un grand nombre d'objets dans le système et un découpage fin en domaines composés de quelques objets. D'autre part, il est impossible pour l'usager de nommer et de partager avec un autre usager n'importe quel objet du système. Enfin, la notion de type est absente (et les conséquences de sa présence ne sont donc pas tirées). Or la fiabilité et l'extensibilité du système sont enrichies par le typage des objets, l'adressage par capacités de tous les objets du système et la petitesse des domaines (voir chapitre 2).

Ce qui explique la différence fondamentale entre OMPHALE et le système proposé par Casey : la taille des domaines. Dès lors, les résultats des simulations ne peuvent pas être transposées immédiatement à OMPHALE. De nouvelles simulations (et/ou des expérimentations sur maquette) devront être entreprises. Avant que les résultats soient connus toute conclusion relative à la faisabilité de la régulation, à l'efficacité du système ou à la modicité de la bande passante requise pour le SSC, est prématurée.

Par ailleurs l'inefficacité des systèmes à capacités réside souvent dans la simulation par logiciel des opérations de manipulation des capacités. En particulier l'opération d'activation de domaine peut se révéler trop coûteuse en temps de calcul (jusqu'à une dizaine de millisecondes). La présence de deux processeurs par site devrait partiellement remédier à cette difficulté. Non

pas que la durée de cette opération soit réduite, elle reste inchangée. Mais l'activation des domaines ne charge pas le processeur de calcul.

Finalement, le problème de l'efficacité d'OMPHALE se réduit à la question : les résultats des simulations de Casey - stabilité de la régulation, efficacité globale du système, modicité de la bande passante requise pour le SSC - sont-ils valides pour de "petits" domaines ?

En résumé de l'organisation matérielle d'OMPHALE.

Le système est composé de sites communiquant entre eux par messages via un Sous-Système de Communication (SSC). Il est dépourvu de mémoire commune.

Les sites sont banalisés : ils sont capables d'exécuter n'importe quel domaine, sauf ceux liés à la gestion des périphériques qui sont associés à des sites particuliers. Aussi les objets, et en particulier les domaines sont mobiles : ils peuvent être déplacés d'un site à un autre. Un mécanisme de régulation de la charge guide le choix du site d'exécution d'un domaine. Le meilleur compromis possible est recherché entre, d'une part, l'équilibre des charges des sites, et d'autre part le traffic sur le SSC. La régulation tend également à regrouper sur un même site des domaines s'inter-appelant fréquemment. Le domaine est transmis au site choisi pour son exécution et, sur ce site, ses composants éparpillés sur le réseau sont rassemblés. Après quoi le domaine est placé dans la file d'attente des domaines prêts à être exécutés.

Un site regroupe une mémoire locale, deux micro-processeurs Z8000 - l'un affecté à la préparation des domaines, l'autre à leur exécution - et les interfaces au SSC et aux périphériques connectés au site.

Au niveau des principes de sa structuration en domaines, le système OMPHALE s'inspire fortement des travaux de Lanciaux. Mais Lanciaux n'envisage que rapidement les problèmes liés à l'implémentation des domaines sur un réseau. Par contre, Casey s'intéresse précisément à ce point, mais néglige les aspects de protection congénitaux à la notion de domaine. Aussi les chapitres 5 et 5 examinent les mécanismes nécessaires à l'implémentation de domaines de protection sur un multi-ordinateur.

L'absence de mémoire commune partagée, de par la perte de la possibilité de centraliser les informations, conduit à redéfinir les mécanismes

- assurant la gestion des noms : le nom unique contenu dans le champ représentation d'une capacité, mène à la représentation en mémoire locale de l'objet repéré ; ou encore à sa représentation en mémoire de masse.
- contrôlant la copie des capacités, et, donc le partage des objets.
- permettant de lier un objet à un site déterminé
- contribuant à l'enchaînement de l'exécution des domaines et à la synchronisation des calculs.

La plupart de ces dispositifs sont répartis : les informations nécessaires à leur fonctionnement résident, en général, sur des sites différents. Ceci provient de ce qu'une capacité, enfermée dans un domaine implanté sur le site n, repère un objet qui se trouve, souvent, sur un autre site m. De même, presque toutes les opérations nucléaires sont réparties : les noyaux d'au moins deux sites différents sont impliqués dans leur exécution. La création des objets, leur destruction, et l'activation des domaines sont, en particulier, des opérations dont le fonctionnement est fortement complexifié par la répartition (il est vrai qu'à l'éparpillement des informations sur les sites, s'ajoute l'objectif de régulation de la charge du système).

Le chapitre 5 traite de la gestion répartie des objets, tandis que le chapitre 6 s'intéresse plus particulièrement aux domaines.

Enfin, les différentes caractéristiques des objets et domaines, les propriétés exigées du SSC, réparties sur ces deux chapitres, sont centralisées en deux annexes : l'annexe 3 résume les opérations applicables aux objets, leurs effets sur les propriétés des objets ; cette annexe donne aussi les formats des descripteurs et des capacités ; l'annexe 4 dresse un cahier des charges du Sous-Système de Communication.

CHAPITRE 5

GESTION REPARTIE DES OBJETS

5.1 PROPRIETES DES OBJETS

Un objet du système présente :

- des propriétés intrinséques liées à sa nature ou à sa fonction,
- des propriétés temporaires dépendant de son utilisation à un instant donné,
- et des propriétés optionelles sélectables au moment de sa création.

Les propriétés d'un objet sont signalées par le positionnement d'indicateurs figurant dans sa représentation. Du fait de leur permanence durant toute la vie d'un objet les indicateurs des propriétés intrinséques se trouvent déjà dans la maquette de l'objet, à partir de laquelle ils sont copiés lors de la création. Les indicateurs de propriétés optionelles sont issus de la liste des paramètres fournis lors de la création d'un objet. Les indicateurs des propriétés intrinséques et optionelles demeurent invariants durant toute la vie d'un objet. Par contre, les indicateurs des propriétés temporaires varient en fonction des opérations appliquées à l'objet, mais les degrés de liberté de cette évolution sont, en général, limités par les propriétés intrinséques et optionelles.

Les propriétés d'un objet peuvent être regroupées suivant :

- le nombre de capacités qui le repèrent (partage),
- la possibilité ou non de le déplacer (mobilité),
- la présence ou non d'une copie de sa représentation en mémoire de masse (catalogage),
- la modification ou non de sa représentation lorsqu'il est manipulé (ré-entrance),
- son aptitude à participer à des boucles de référence (récursivité).

5.1.1 Partage

Un objet est **partagé** (propriété temporaire notée P) quand il est référencé par plus d'une capacité. A sa création, un objet est non partagé. La propriété de partage s'acquiert par application d'une **opération de partage**, qui s'écrit

$$[d.]e <= \sim c$$

où c est le nom local d'une capacité dans le domaine courant (en cours d'exécution).

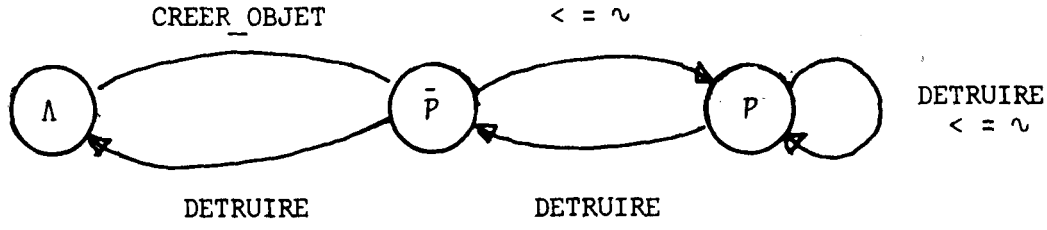
d est le nom local d'une capacité pour un domaine destination.

e est le nom local d'un emplacement dans le domaine destination.

(si d est omis le domaine destination est confondu avec le domaine courant) et qui procéde à la copie de la capacité c à l'emplacement e du domaine d. L'activation de l'opération de partage est subordonnée à la présence d'un **droit de copie** dans la capacité c.

Un **compteur de références** mémorise le nombre de capacités qui repèrent le même objet, par suite d'exécutions répétées de l'opération de partage. Ce compteur permet de décider de la destruction de l'objet (voir § 3.3.4 et § 5.7.1).

La propriété de partage se perd par des destruction successives de capacités, jusqu'à ce qu'il n'en subsiste plus qu'une :



Λ : état inexistant.

Le partage des objets catalogués est soumis à une restriction (voir § 5.1.3)

L'opération de partage est détaillée au paragraphe 5.3.

5.1.2 *Mobilité*

En général, les objets sont **mobiles** : ils peuvent être déplacés d'un site à un autre. Mais certains objets ne jouissent pas de cette propriété intrinséque : ils sont assignés à résidence sur un site donné et sont qualifiés d'**immobiles**. Le numéro du site de résidence d'un objet immobile est reçu en paramètre lors de sa création.

Un objet mobile peut être **immobilisé** (propriété temporaire notée $\bar{I}$) par action d'une **opération d'immobilisation**, notée

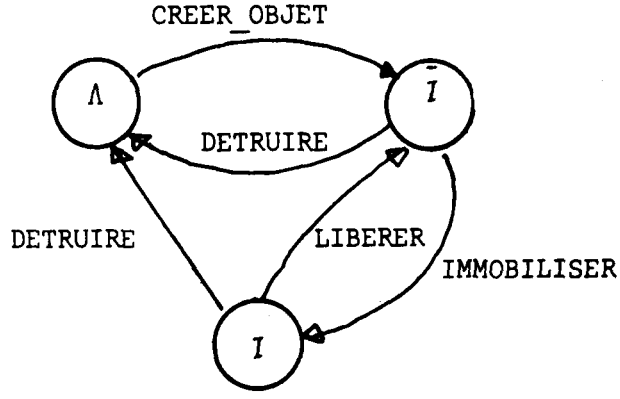
IMMOBILISER (c, n)

où c est le nom local d'une capacité pour un objet O dans le domaine courant,
n est un numéro de site
qui astreint l'objet O à résider temporairement sur le site n. A sa création un objet mobile n'est pas immobilisé.

La propriété d'immobilisation se perd par l'appel d'une **opération de libération**, notée

LIBERER c

où c est le nom d'une capacité dans le domaine courant :



Un **indicateur d'immobilisation** est positionné dans toute capacité pour tout objet immobile et pour tout objet mobile immobilisé. Cette information est utile lors du choix du site d'exécution d'un domaine renfermant un objet immobilisé.

L'utilisation des opérations d'immobilisation et de libération est soumise à une restriction, qui est examinée au paragraphe 5.5, comme les opérations elles-mêmes.

5.1.3 *Catalogage*

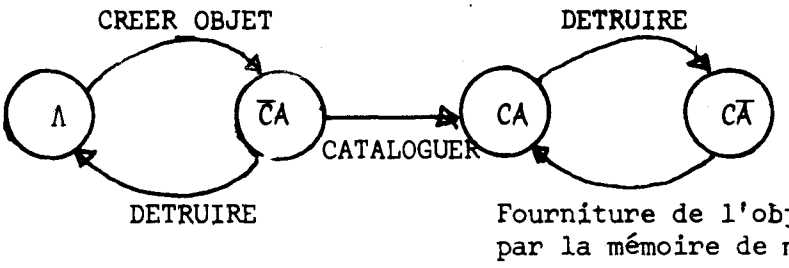
Un objet **catalogué** (propriété temporaire notée C) possède une copie de sa représentation en mémoire de masse. La copie est accessible par l'intermédiaire d'un **nom symbolique**. Des **catalogues** maintiennent les correspondances entre les noms symboliques des objets et des capacités pour les mêmes objets. Lampson ([31]) et Redell ([58]) ont montré que ces structures de données peuvent être entièrement construites à l'aide de l'adressage par capacités, même en ce qui concerne la réalisation de contrôles sur des listes d'accès.

Pour qu'un objet puisse être catalogué, il faut qu'il soit **catalogable**. Un objet non catalogable est dit **passager**. La propriété optionnelle de catalogage d'un objet est choisie lors de la création de l'objet. Au même moment, la décision est prise de permettre ou non le partage ultérieur d'un objet catalogable. Ainsi un objet est soit catalogable, soit passager. Et un objet catalogable est partageable ou non. Ces dispositions simplifient la gestion des noms des objets (voir § 5.2). Elles ne restreignent en rien la généralité du système : par précaution, un objet peut être déclaré comme catalogable et partageable et n'être jamais ni catalogué, ni partagé. L'objet n'est considéré comme catalogué qu'après l'application d'une **opération de catalogage**, notée

CATALOGUER c

où c est le nom local dans le domaine courant d'une capacité pour un objet catalogable.
Cette opération ne copie pas la représentation de l'objet en mémoire de masse ; elle ne sert qu'à signaler l'objet comme catalogué (positionnement d'un indicateur de catalogage) et à établir les chaînes d'accès ad hoc (voir § 5.4.2).

Enfin, un objet posséde la propriété temporaire **d'activité** (notée A) quand sa représentation réside dans la mémoire locale d'un site et/ou quand il est référencé par une capacité résidant dans la mémoire locale d'un site (capacité active). Au contraire, un objet est **passif**, quand il réside seulement en mémoire de masse et quand il n'est référencé que dans les catalogues. Aussi, dès sa création un objet est actif et, si il est catalogué, il devient passif quand la dernière capacité qui le référence est détruite :



L'opération CATALOGUER est détaillée au paragraphe 5.4.2.

5.1.4 Ré-entrance

Certains objets, comme les segments de code ou les objets-type, sont dotés de la propriété intrinséque de **ré-entrance** : leur manipulation n'entraîne pas d'altération des valeurs que contiennent leurs représentations. Aussi, les objets ré-entrants peuvent être copiés sans que se pose le problème de la mise à jour ultérieure des diverses copies.

5.1.5 Récursivité (voir § 3.3.2)

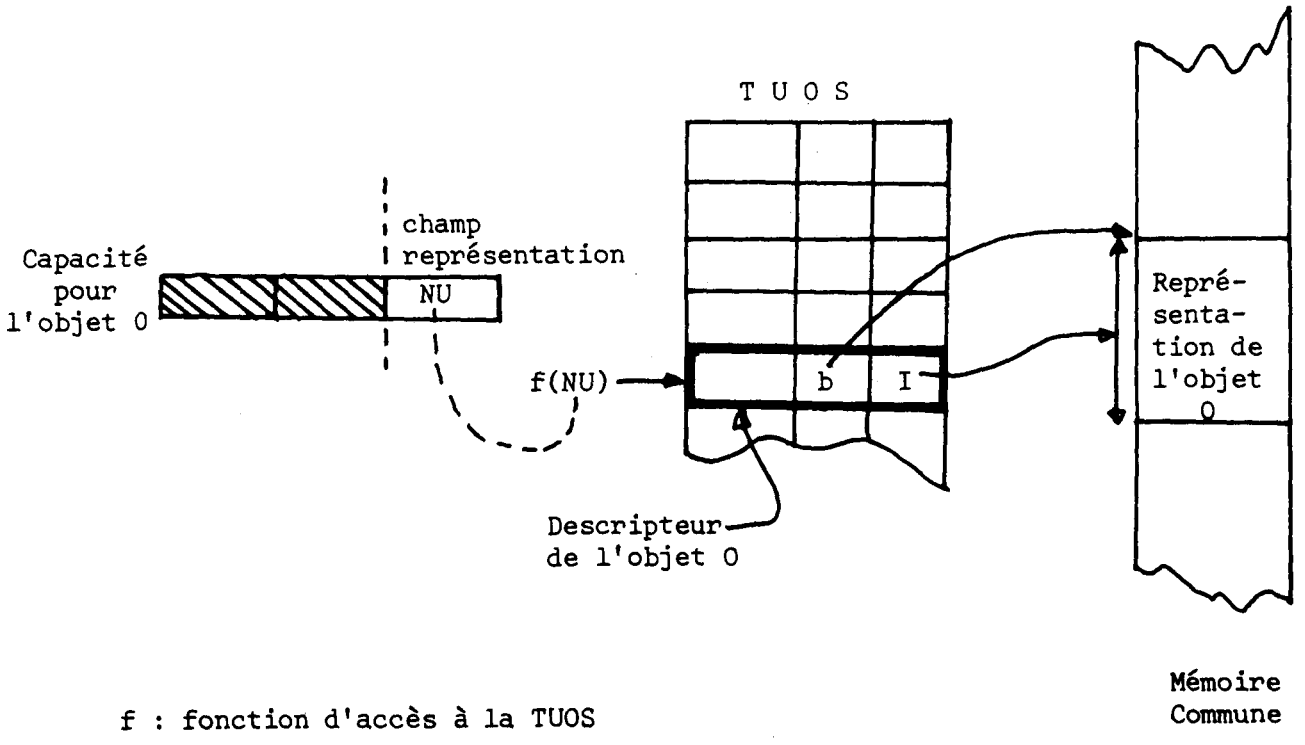
La **récursivité** ou la **prérécursivité** d'un objet sont des propriétés intrinséques, car elles sont liées à son type. Lors de la recherche des objets perdus, un objet prérécursif peut être accédé par l'une des capacités qui le repèrent. Aussi un **indicateur de prérécursivité** figurant dans chaque capacité pour un objet indique si la chaîne d'accès menant à cet objet doit être parcourue ou non. Par contre, l'**indicateur de récursivité** ne figure que dans la représentation d'un objet prérécursif : le marquage d'un objet prérécursif s'effectue en présence de sa représentation.

5.2 MECANISMES DE GESTION DES NOMS

Le champ représentation d'une capacité renferme un nom unique qui est le premier maillon de la chaîne d'accès menant à la représentation physique en mémoire de l'objet repéré. Les dispositifs qui contribuent à la maintenance et à l'établissement des chaînes d'accès sont appelés **mécanismes de gestion des noms**.

* *Position du problème*

Par opposition à un **nom local** qui désigne une capacité pour un objet dans un environnement donné (dans un domaine) et qui n'a de signification que pour le domaine considéré, un **nom** est **unique** quand il désigne un et un seul objet parmi tous ceux que compte le système, Un nom unique doit donc avoir un sens pour l'ensemble du système, Aussi, dans le cas des systèmes dotés d'une mémoire primaire commune le schéma de gestion des noms suivant est appliqué : Chaque entrée d'une **Table Unique des Objets du Système** (TUOS) renferme un descripteur associé biunivoquement à un objet. Le **descripteur** existe en un exemplaire unique et localise la représentation de l'objet en mémoire (par l'intermédiaire d'une adresse de base et une taille de segment, par exemple). Un nom unique sélectionne alors l'entrée dans la TUOS correspondant à l'objet repéré :



f : fonction d'accès à la TUOS

NU : nom unique

Schématiquement les noms uniques peuvent être de deux types, dont dépendent la méthode de génération du nom et le mode de sélection de l'entrée dans la TUOS (Lanciaux [34]) :

- un nom temporel est, typiquement, généré par une horloge du système dont la période est suffisamment faible pour que deux lectures consécutives de sa valeur donnent des noms différents. Les descripteurs renferment alors les noms auxquels ils sont associés et sont rangés dans l'ordre des noms croissants dans la TUOS. Ou l'index d'un descripteur est donné par le calcul d'une fonction de "hash-code" sur le nom unique. Bien d'autres méthodes sont imaginables.

- un nom spatial localise directement le descriteur dans le TUOS, par son index dans la table, ou son adresse. Générer un descripteur revient alors à créer une nouvelle entrée dans la TUOS.

Les difficultés d'implémentation de mécanismes similaires sont bien connues (Lanciaux, Redell [58]). Le grand nombre d'objets constituant le système aboutit à une grande taille de la TUOS incompatible avec son implémentation en mémoire primaire. Et des méthodes de pagination de la table et de stockage en mémoire secondaire ralentissent considérablement les accès. Enfin, la longueur des noms influe sur la taille des capacités : l'emploi de noms temporels conduit à des capacités de presque une dizaine d'octets.

L'absence de mémoire commune dans le système OMPHALE pose quelques problèmes supplémentaires. Tout d'abord, le déplacement d'un objet d'un site à un autre ne doit pas entrainer une recherche et une mise à jour de toutes les capacités pour cet objet, qui peuvent être éparpillées sur tout le réseau. Puis, une table unique est incompatible avec l'objectif de disponibilité du projet. Mais un mouvement d'objet ne doit pas conduire à la mise à jour de tables dans des sites non impliqués par le transfert. Et, bien sûr la question de la taille des tables est d'autant plus préoccupante que les mémoires locales sont plus petites (par rapport à la mémoire commune d'un multi-processeur).

OMPHALE manipule deux sortes de noms : les noms spatiaux et les noms globaux.

* *Noms spatiaux*

Un **nom spatial** est un couple (n, d) où n est le numéro du site qui possède la représentation de l'objet repéré, et d est une information qui permet au site n de retrouver l'objet dans sa mémoire locale. Seul ce site n est apte à interpréter d conformément à son organisation interne. Ainsi, si le site est structuré comme décrit au paragraphe 4.3.3, d est la concaténation d'un numéro de bloc mémoire et d'un index de descripteur dans le bloc (chaque bloc renferme les descripteurs des objets qu'il contient). Dans ce cas, créer un nom spatial nécessite d'allouer une zone dans une page destinée à contenir la représentation de l'objet, et à créer son descripteur. Détruire un nom spatial impose de libérer la zone occupée puis le descripteur. Mais un autre site de la même configuration pourra être organisé différemment.

* *Noms globaux*

Chaque site d'OMPHALE dispose d'un **générateur de valeurs uniques** et d'une **Table des Objets Globaux** (TOG). Un **nom global** est un couple (n, v) où v est une valeur unique et n le numéro du site qui a généré la valeur. La TOG d'un site m renferme les descripteurs des objets repérés par des noms globaux et résidant sur le site m. Lorsque l'un de ces objets doit être transmis à un site m', son descripteur est ôté de la TOG de m, envoyé au site m' et placé dans la TOG de m'.

L'interface entre le SSC et un site est capable d'effectuer des recherches associatives dans la TOG. Par ailleurs le SSC diffuse les messages à tous les sites. Quand un message est destiné "au site qui posséde l'objet de nom global (n, v)", les interfaces consultent leur TOG respective et un seul site m répond - celui qui posséde (n, v) dans sa TOG. Pour accélérer la recherche, le dispositif associatif peut être intégré au matériel de l'interface au SSC (par exemple, une mémoire de mots de 1 bit - présence ou non de l'objet - et adressée directement par le nom global serait suffisante).

Enfin, à chaque nom global est associé un **compteur de références**, qui n'est autre que le compteur des capacités repérant l'objet. Dès que l'objet est partagé un compteur est créé et positionné à deux. A chaque copie du nom - à chaque partage de la capacité le renfermant - la valeur du compteur est incrémentée de un ; elle est décrémentée de un à chaque destruction d'une copie du nom. Quand l'objet n'est plus repéré la valeur v est rendue au site

n qui l'a générée (le numéro n permet d'identifier le site ; de plus la présence de ce numéro assure l'indépendance des générateurs des différents sites). Le site n pourra réutiliser le même nom global (n, v) pour repérer un autre objet.

* *Emploi des noms*

Lors de sa création un objet passager est repéré par un nom spatial. Si l'objet créé est immobile, il gardera le même nom spatial durant toute sa vie. Par contre, si il est mobile, la première opération de partage qu'il subit remplace son nom spatial par un nom global qui le désignera jusqu'à sa destruction.

Un objet catalogable immobile et/ou non partageable est repéré durant toute sa vie par des noms spatiaux. Un objet catalogable, mobile et partageable est désigné par un nom spatial à sa création. Le nom spatial est transformé en nom global dès la première application sur l'objet d'une opération de partage ou de catalogage. Dès que l'objet devient passif il retrouve un nom spatial qui repère sa représentation en mémoire de masse. Lorsqu'il redevient actif, le nom spatial est transformé en un nom global. Et quand il redevient passif, il retrouve le nom spatial en mémoire de masse et ainsi de suite. L'annexe 3 donne les graphes des transitions des propriétés temporaires d'un objet sans l'effet des opérations qu'il subit. Le fait d'être repéré par un nom spatial y est considéré comme une propriété temporaire.

Ainsi, l'emploi des noms globaux est limité aux seuls objets partagés et mobiles. C'est le seul cas où il est difficile de mettre à jour les capacités lors d'un déplacement de l'objet repéré et où l'utilisation d'un nom indépendant de la localisation de l'objet s'impose. Cette question ne se pose évidemment pas pour les objets immobiles. Et elle est résolue simplement pour les objets mobiles non partagés : un tel objet n'est, par définition, repéré que par une seule capacité et, comme un déplacement est toujours dirigé vers le site n qui posséde la capacité (c'est cette capacité qui a déclenché le mouvement), la mise à jour du nom spatial est effectuée lors de l'arrivée de l'objet sur le site n. Ces dispositions réduisent les tailles des TOG et des noms globaux eux-mêmes. De plus, les noms globaux sont réutilisés. Il est inutile de désigner un objet pasif par un nom global, car, dans ce cas, il

ne peut pas être immédiatement partagé. Un nom spatial en mémoire de masse suffit (Herbert [22]). Comme le nombre des objets passifs catalogués et partageables est très supérieur au nombre des objets actifs, ceci contribue aussi à réduire la taille des noms et des tables. Puis, un site ne dispose des descripteurs que des objets qui résident dans sa mémoire locale, ce qui renforce la disponibilité des mécanismes de gestion des noms par rapport à une table unique monolithique. Et, durant le déplacement d'un objet, les sites expéditeurs et destinataire sont les seuls concernés. Enfin, pour un site donné, la taille de la table des descripteurs des objets repérés par des noms spatiaux ne devrait pas se révéler prohibitive, la table était confondue avec la table de gestion de la mémoire locale.

5.3 PARTAGE DES OBJETS

* *Limitation des possibilités de partage*

Tous les objets sont, potentiellement, partageables. Mais les possibilités de partage sont limitées par la configuration matérielle du système. Comme il n'existe pas de mémoire commune à deux processeurs, le partage d'un objet entre deux domaines en cours d'exécution est effectué en **exclusion mutuelle** : il est impossible que deux domaines aient simultanément accès au même objet.

* *Répartition de l'opération de pargage*

En toute généralité, lors de l'exécution d'une opération

$$d \ , \ e < = \sim c$$

- la capacité c réside sur le site n (site sur lequel s'exécute le domaine considéré).
- l'objet O repéré par la capacité c réside sur le site n'.
- le domaine d réside sur le site n"

et n, n', n" sont tous trois différents. Or l'opération de partage doit déclencher

1) a) L'incrémentation de un du compteur de référence , s'il existe, ou son positionnement à deux dans le cas d'un premier partage.

b) La transformation du nom spatial, qui repère éventuellement l'objet, en nom global au cas où l'indicateur d'immobilisation figurant dans la capacité c n'est pas positionné.

Le site n' est donc impliqué dans l'opération, car c'est lui qui posséde le descripteur de l'objet O, et, donc, son compteur de références, s'il existe.

2) La copie sur le site n de la capacité c et son envoi au site n".

Aussi, si le cas 1-b) se présente un nom global (n, v) est généré sur le site n et un **message de service** est envoyé au site n', message qui demande la substitution de (n, v) au nom spatial (n', d). Le site n' procéde à la mise à jour de ses tables de descripteurs et insère dans sa TOG le descripteur de l'objet O. Puis, dans tous les cas, un (autre) **message de service** demandant l'incrémentation du compteur de références est envoyé au site n'. Si le compteur n'existe pas le noyau le crée ; sinon il l'incrémente de un. La capacité c, dont le champ représentation a éventuellement été mis à jour, est alors copiée sur le site n(opération <=) et transmise au site n" au moyen d'un **message de transfert de capacité** (MTC).

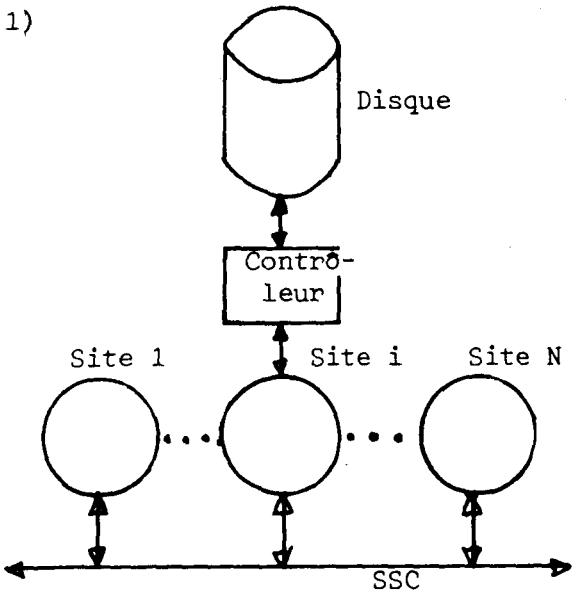
Ainsi, plusieurs domaines résidant sur des sites différents peuvent posséder chacun une capacité pour un objetO. En supposant que ces domaines soient activés simultanément, il faut décider à quel domaine attribuer l'objet O. Pour ce faire, une **File d'Attente des Ordres de Transfert** (FAOT) est associée à tout objet partagé. La phase de rassemblement des objets constituant un domaine donne lieu à l'envoi de **Messages-Ordres de Transfert** (MOT) aux sites possédant ces objets (voir § 6.3.1). Pour un objet partagé, la réception d'un MOT se traduit par le stockage d'un nouvel ordre dans la FAOT. Les ordres de transfert sont servis dans l'ordre de leur arrivée. Toute suspension ou destruction d'un domaine renfermant un objet partagé conduit à "une libération" de l'objet et à la progression de la FAOT. Afin d'éviter les étreintes fatales, il est supposé que le SSC est capable de transmettre tous les MOT concernant un même domaine en une seule opération indivisible. La mise en place de la FAOT est assurée par le noyau lors de la réception d'un message de service demandant d'incrémenter un compteur de références inexistant. La FAOT accompagne les déplacements du descripteur d'un site à un autre.

5.4 CATALOGAGE DES OBJETS

5.4.1 *Préliminaires : connexion d'un disque au système* (Casey [4])

Un ou plusieurs disques constituent la mémoire de masse d'OMPHALE. La prise en charge d'un contrôleur de disque par le système pose des problèmes spécifiques. D'une part, le contrôleur génère des interruptions qui doivent être traitées par un processeur. D'autre part, les données à lire et à écrire sur le disque transitent par des mémoires-tampon, qui, elles sont considérées comme des objets du système, sont immobiles. Quatre solutions sont décrites et discutées, ci-après.

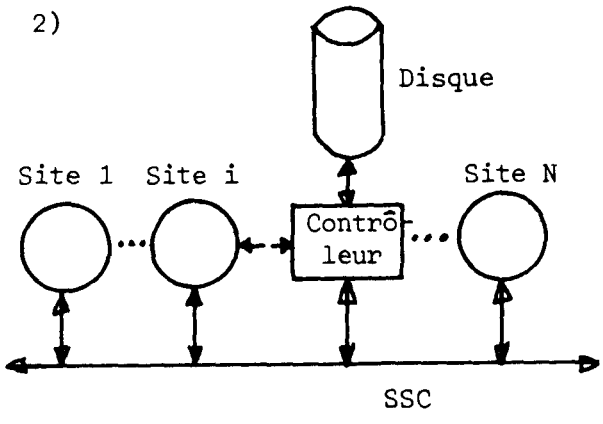
1)



Le disque est connecté via le contrôleur à un site banalisé i. Le site i est capable d'exécuter n'importe quel domaine mobile. Les mémoires-tampon sont des segments immobiles dans la mémoire locale du site i. Le noyau est conçu de manière à traiter les signaux d'état et de contrôle émis par le controleur.

Ce schéma posera problème à forte charge du système. Pour effectuer un transfert un domaine gérant du disque est appelé. Mais ce domaine est immobile sur le site i, puisque les mémoires-tampon le sont. Aussi, à forte charge le nombre de tels domaines croit jusqu'à constituer l'essentiel de la charge du site i. Il risque donc d'y avoir aggravation du déséquilibre des charges des sites. De plus, en raison de cette forte charge sur le site i, il sera difficile de regrouper sur le même site i les domaines participant à une même phase d'un calcul.

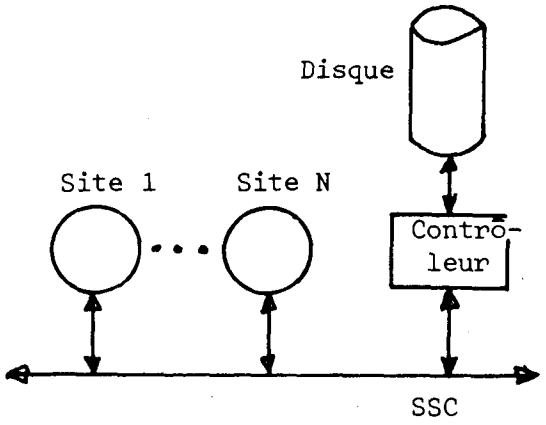
2)



Le site i est banalisé. Son noyau prend en charge le contrôle du disque. Le contrôleur disque dispose de mémoires-tampon qui ne sont pas des objets du système. Seuls les paramètres d'un transfert sont envoyés au site i, les segments de données à transférer restant

sur le site n de résidence du domaine demandeur. Les domaines sont acheminés directement vers le (à partir du) contrôleur via le SSC à partir de (vers) la mémoire locale du site n. La charge de contrôle du disque étant faible, ce schéma ne crée pas de déséquilibre supplémentaire des charges des sites. Mais le contrôleur doit être équipé de mémoires-tampon volumineuses. Et les consultations de catalogues ne sont pas efficaces.

3)



Vu du SSC, le contrôleur est suffisamment "intelligent" pour supporter en partie la structure de domaine. Par exemple, il est capable d'effectuer des choix de site. De plus, il gère les catalogues. Mais ces fonctions ne sont pas forcément assurées par des domaines immobiles. L'essentiel est que, vu du SSC, elles apparaissent commes telles. Les inconvénients des solutions précédentes disparaissent sauf celui de l'importance en volume de la mémoire locale. Cette solution est celle adoptée pour OMPHALE. En raison de sa complexité, le "contrôleur" est baptisé, dans la suite, "**site disque**", bien que ces termes soient impropres.

4) (*même schéma que 3*) . Le contrôleur ne supporte pas la structure de domaines. L'état et les commandes du contrôleur doivent être acheminés par message vers le (à partir) ou les sites possédent le ou les domaines gérant le disque, ce qui est beaucoup trop inefficace.

5.4.2 *Gestion des noms des objets catalogables*

Lors de l'activation de l'opération CATALOGUER, un objet catalogable peut être repéré par :

1) un nom global,
2) un nom spatial et être mobile, partageable et non immobilisé,
3) un nom spatial et être a) immobile
   ou b) immobilisé
   ou c) non partageable.

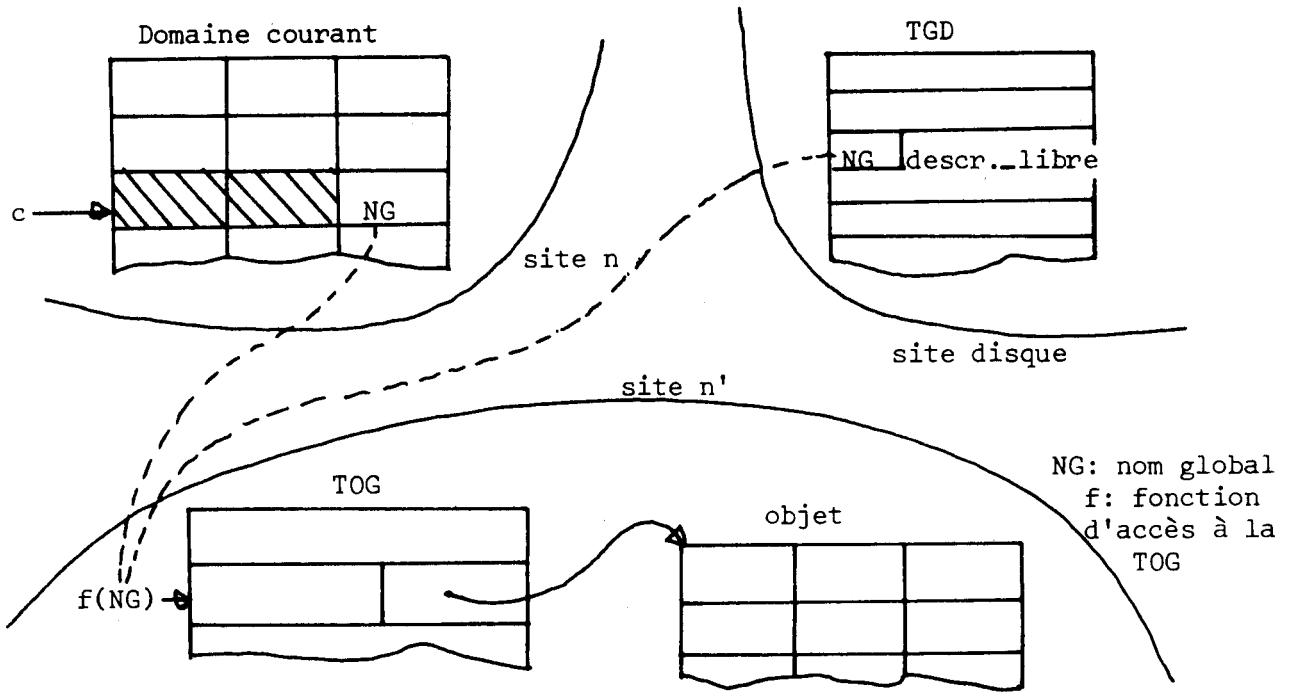
L'opération CATALOGUER établit une chaîne d'accès entre le catalogue et l'objet, chaîne qui permettra un stockage réel ultérieur de la représentation de l'objet.

Le site disque est doté d'une **Table de Gestion du Disque** (TGD) qui posséde les descripteurs des objets stockés sur disque. Aussi, il convient d'établir une liaison entre un descripteur en TGD et la représentation (active) de l'objet. Quand l'objet est repéré par un nom global, il suffit de placer ce nom dans le descripteur. Par contre, quand l'objet est repéré par un nom spatial, il faut adopter une autre solution, puisque le nom spatial varie durant la vie de l'objet. Premièrement, il est possible de générer un nom global et de le placer et dans le descripteur en TGD et dans la représentation de l'objet à un emplacement réservé à cet effet lors de sa création. Deuxièmement, le site disque peut générer un nom dans la TGD et ce nom peut être placé dans la représentation de l'objet. La nécessité de limiter le nombre des noms globaux est peu favorable à la première solution ; l'aller-retour entre site demandeur et site disque qu'elle entraine constitue un inconvénient de la seconde. Sans pour autant trancher définitivement, la deuxième hypothèse est envisagée dans la suite (les schémas sont aisément transposables à la première).

Au moment de l'activation de l'opération

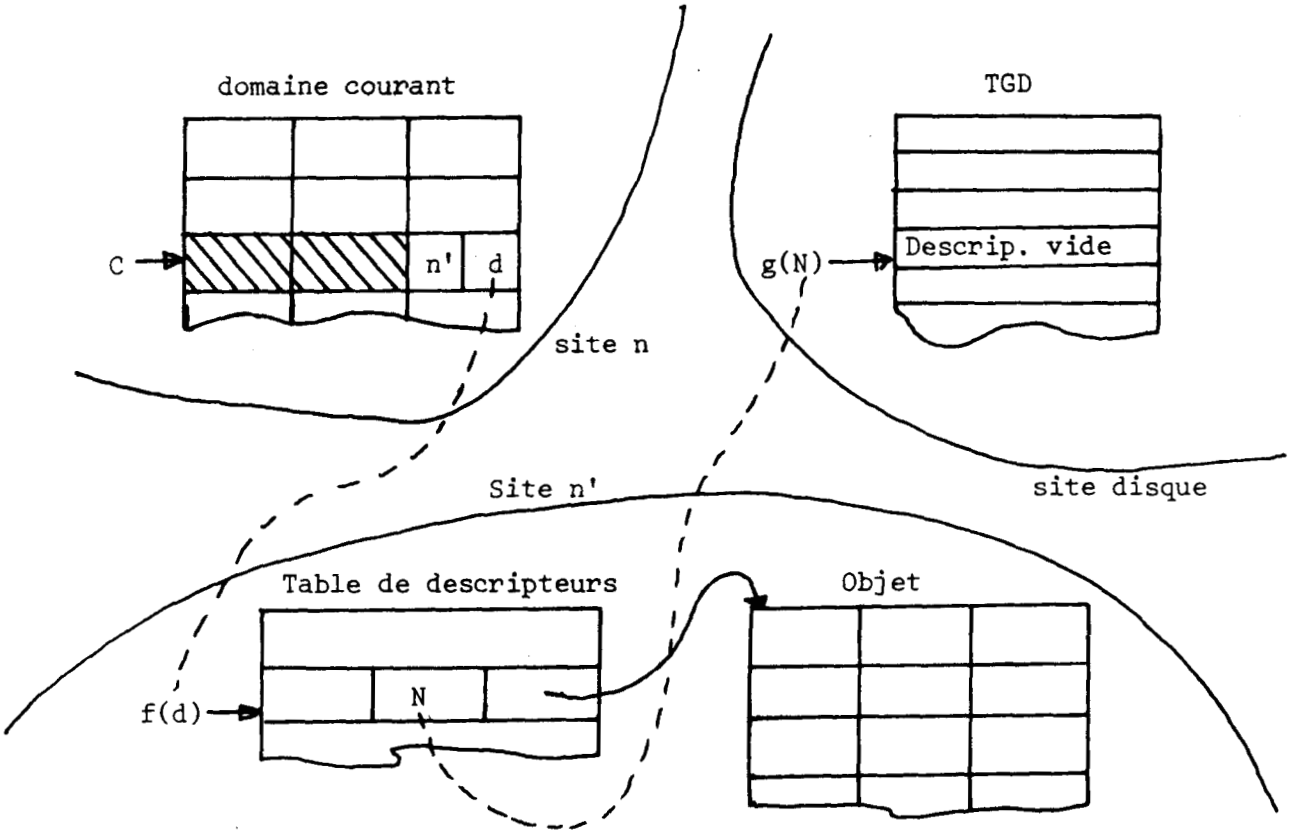
CATALOGUER c

la capacité c réside sur le site courant n, et l'objet repéré sur un site n' ; en général $n \neq n'$. Dans le cas 1) un message de service est envoyé en site disque. Il demande d'associer un descripteur libre dans la TGD au nom global (n, v). A l'issue de l'opération la situation est la suivante :



Dans le cas 2) un nom global (n, v) est généré sur le site n. Un premier message se service est envoyé au site n'. Il demande la substitution de (n, v) ou nom spatial (n', d) (voir § 5.3). Ce cas se ramène ou alors, au cas 1) et un second message de service est envoyé dans les mêmes conditions.

Dans le cas 3) un message de service est envoyé au site disque. La réception de ce message déclenche la génération d'un nom N pour un descripteur libre dans la TGD ; ce nom est retourné au site n' et il est placé dans le descripteur de l'objet. Après opération la situation est la suivante



f, g : fonctions d'accès.

Remarques :

1) Une table annexe possédée par le site disque et maintenant les correspondances NG <—>N,où N est un nom dans la TGD, peut accélérer les accès à la TGD.

2) Un indicateur signalant que l'objet est partageable ou non est transmis à la TGD par le message de service.

3) Si l'opération de catalogage porte sur un objet mobile, partagé et immobilisé, un nom spatial peut être associé au nom dans la TGD. Il convient, alors, qu'une exécution d'une opération de libération transforme le nom spatial et rectifie la liaison.

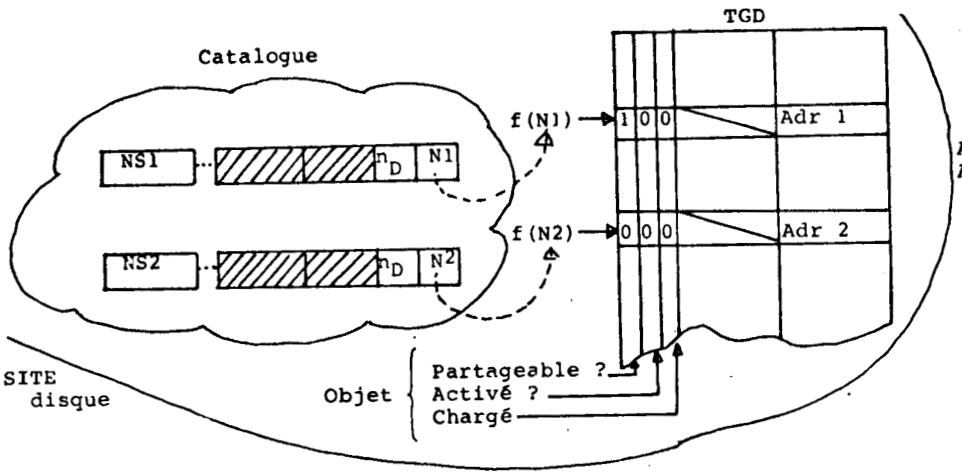
Par la suite le catalogage effectif de la représentation sera déclenché soit explicitement, soit implicitement, lors de la destruction de la représentation active de l'objet. Ce qui suppose qu'entre temps un nom symbolique ait été associé au nom dans la TGD. Aussi, une opération nucléaire qui permette d'extraire le nom dans la TGD doit être définie. Mais cette définition est prématurée, car l'organisation des catalogues n'est pas encore arrêtée.

Quoi qu'il en soit, le catalogue assure, en général, plusieurs liaisons entre des noms symboliques et des capacités qui repèrent la représentation passive d'un objet, au moyen de noms spatiaux (n_D, N) où n_D est le numéro du site disque et N est un nom dans la TGD. Dès qu'un objet est rendu passif, le nom unique qui le repérait est détruit. S'il s'agissait d'un nom global, sa valeur est rendue au site qui l'a générée, ce qui peut nécessiter l'envoi d'un message de service. Un objet passif O est activé dès qu'un objet O' contenant une capacité qui le référence est transféré du disque au système. Les C-listes des objets transférés sont parcourues. Pour chaque capacité pour un composant partageable, un nom global (n_D, v) est généré. Ce nom global remplace le nom spatial (n_D, N). Et le descripteur d'un tel objet est stocké dans la TOG du site disque. Ainsi toute nouvelle sollicitation du catalogue délivre une capacité pour la représentation active de l'objet. Ces dispositions concernent le cas général des objet partageables. Mais, pour certaines applications, il est nécessaire d'accéder au catalogue en exclusion mutuelle : une et une seule capacité en est, alors, issue. Le nom spatial (n_D, N), dans ce cas, est conservé, puisqu'aucune copie ne peut en être effectuée à partir du catalogue. Cependant, les domaines qui manipulent, ensuite, l'objet actif gardent toute liberté de transformer le nom global en nom spatial et de partager l'objet. Le nom spatial figurant dans une capacité pour un composant non partageable est également conservé tel quel, puisque, par définition, cette capacité est la seule à repérer l'objet durant toute sa vie. Les figures de la page suivante visualisent l'évolution des chaînes d'accès aux objets catalogués.

BUS LILLE

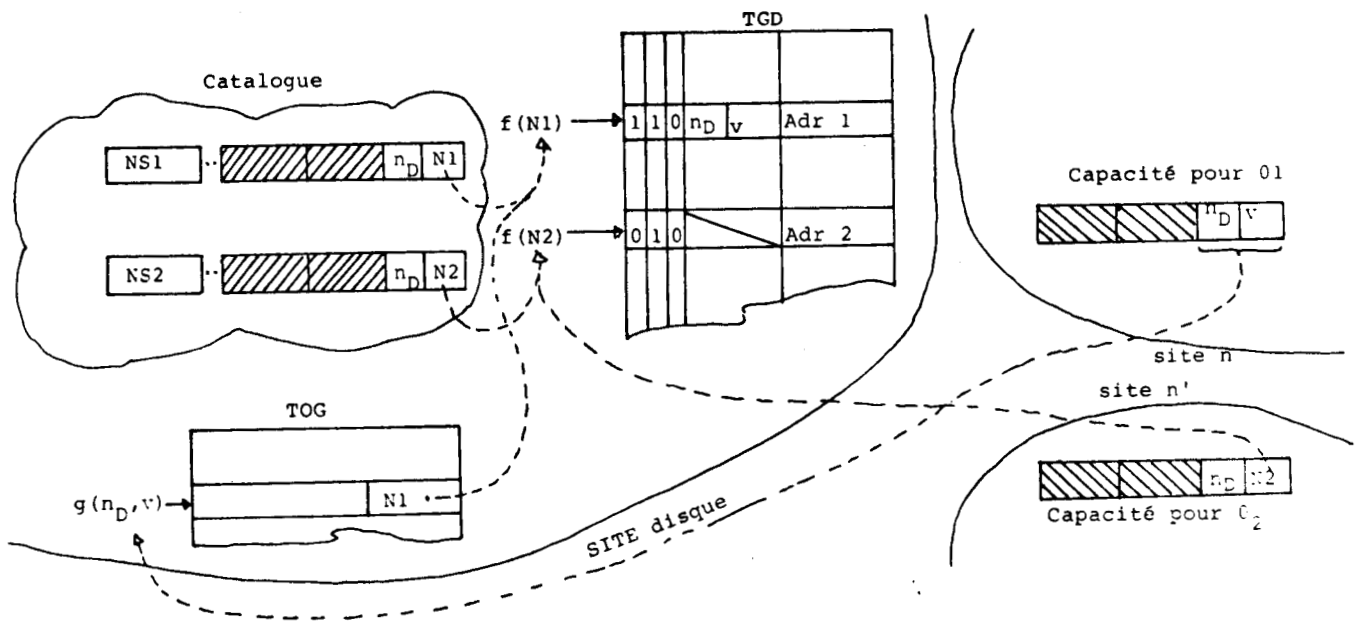
Remarque : Il se peut que les noms dans la TGD soient trop longs pour figurer dans les capacités actives. Ils seront, alors, remplacés par des noms récupérables - similaires aux noms globaux - réutilisés dès que l'objet redevient passif (le nombre des objets passifs est très supérieur au nombre des objets actifs).

* OBJETS PASSIFS

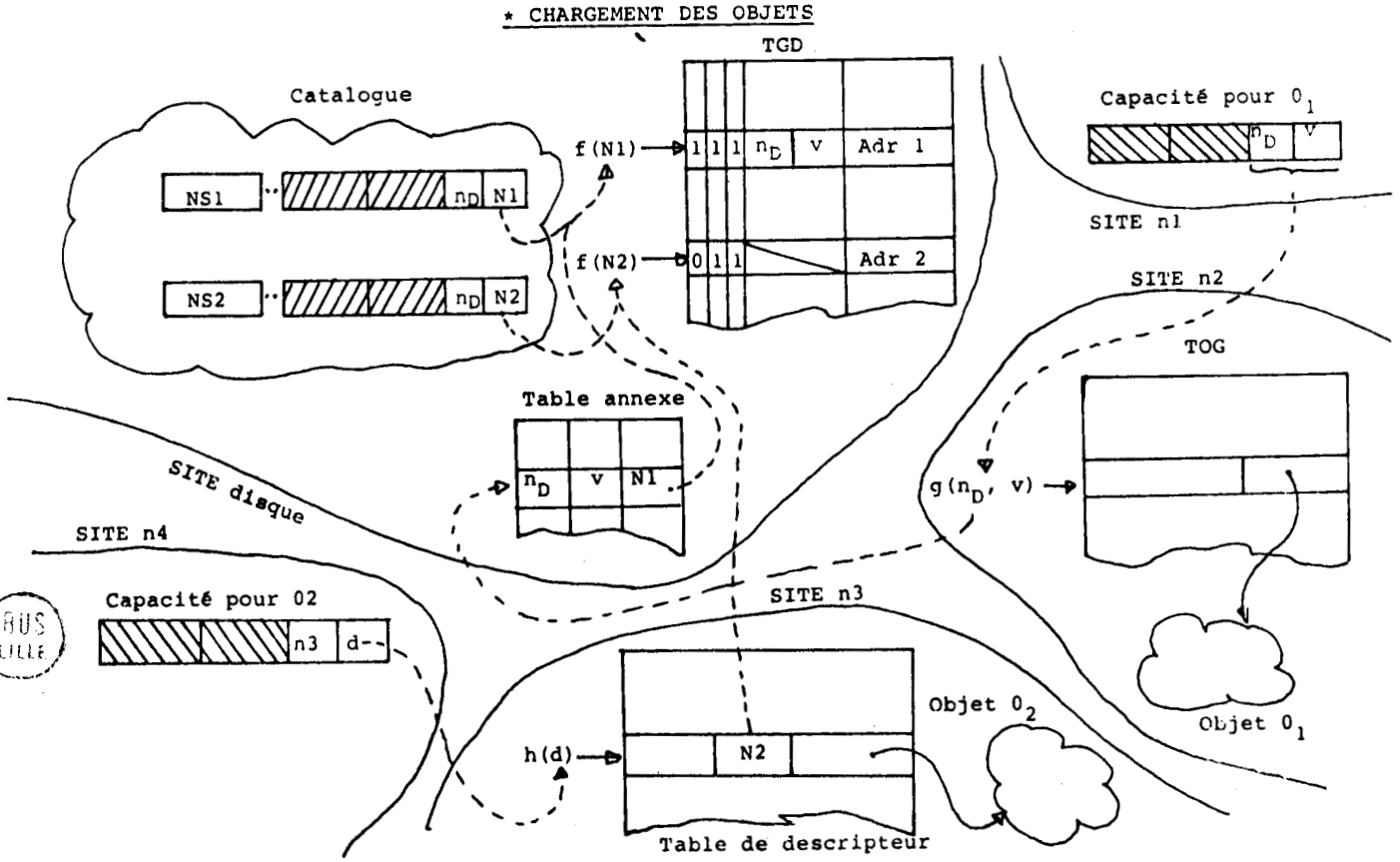


O_1 : Objet catalogeable et partageable
O_2 : Objet catalogeable et non partageable
NS1 : nom symbolique de O_1
NS2 : nom symbolique de O_2
Adr_1 : adresse disque de O_1
Adr_2 : adresse disque de O_2
N1 : nom dans la TGD de O_1
N2 : nom dans la TGD de O_2
n_D : numéro du site disque
f : fonction d'accès
g : fonction d'accès
h : fonction d'accès

* ACTIVATION DES OBJETS



* CHARGEMENT DES OBJETS



BUS LILLE

5.5 MOBILITE DES OBJETS

5.5.1 *Répartition de l'opération d'immobilisation*

Lors de l'activation

IMMOBILISER (c, n)

la capacité de nom c réside sur un site n', l'objet repéré sur un site n", et, en général n, n' et n" sont tous trois différents. Aussi un message de service est envoyé au site n. Le site n envoie, ensuite un Message-Ordre de Transfert d'Objet au site n", qui déplace alors l'objet par l'intermédiaire d'un Message-Transfert d'Objet. Le site n a la possibilité de refuser le déplacement. Une procédure de traitement d'erreur est exécutée dans ce cas.

5.5.2 *Répartition de l'opération de libération*

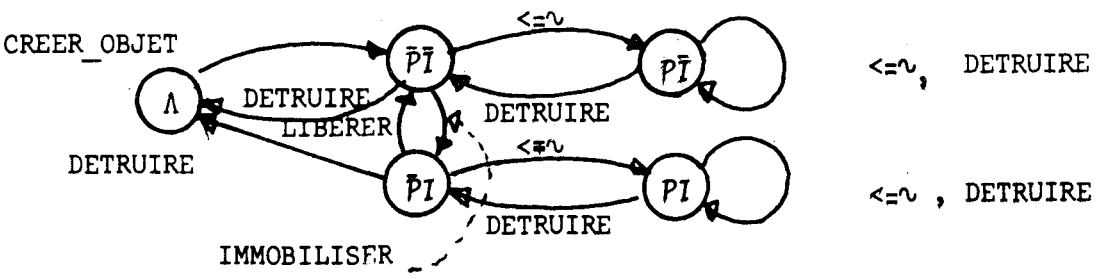
Au moment de l'exécution de l'opération

LIBERER c

la capacité c réside sur le site n et l'objet 0 repéré sur un site n'. La libération de 0 se réduit à la mise à jour de l'indicateur d'immobilisation figurant dans la capacité c et dans la représentation de l'objet. Aussi l'envoi d'un message de service au site n' est suffisant. Sauf au cas où 0 est à la fois catalogué, partageable et repéré par un nom spatial (voir remarque 3, § 5.4.2). Il convient alors de transformer le nom spatial en nom global, ce qui nécessite la génération d'un nom global sur le site n, et la substitution via un message de service. Puis la liaison avec la TGD doit être actualisée, ce qui entraîne l'envoi d'un dernier message au site disque.

5.5.3 *Restriction à l'emploi des opérations de libération et d'immobilisation*

Les opérations d'immobilisation et de libération ne peuvent être appliquées qu'à des objets non partagés :



Cette restriction est techniquement indispensable ; si l'immobilisation ou la libération portaient sur un objet partagé, il serait impossible de mettre à jour les indicateurs d'immobilisation figurant dans les capacités autres que celle désignée par l'opération effectuée. Or l'information d'immobilisation est utile à une activation correcte des domaines (voir § 6.2.1, "Informations utiles au choix de site"). De toutes façons la limitation introduite est faible : l'immobilisation d'un objet est le fait d'un domaine qui constitue, en quelque sorte son gérant, et qui est capable de contrôler les copies de capacités.

5.6 REPARTITION DES OPERATIONS DE CREATION DES OBJETS

Au moment de l'exécution d'une opération d'interprétation d'une maquette du type T, l'objet-type T réside sur un site n', en général différent du site courant n. Cette opération est donc répartie.

5.6.1 *Répartition de l'opération d'interprétation de maquette*

L'appel de l'opération INTERPRETER_MAQUETTE se traduit par l'emission d'un **Message-Demande d'Interprétation de Maquette** (MDIM) destiné au site n' possédant l'objet-type. Le MDIM renferme le numéro de la maquette à interpréter et la C-liste des paramètres d'interprétation. A la réception du message, le noyau du site n' effectue un choix du site de création de l'objet. Deux sites sont éligibles : le site n' et le site le moins chargé, noté m. Le choix du site n' présente l'avantage de permettre la création immédiate de l'objet. Mais il risque d'entraîner un déséquilibre des charges des sites, car l'accumulation sur le site n' d'objets de type T peut conduire à une migration massive vers le même site des domaines qui les manipulent. Au contraire, l'élection du site m ré-équilibre les charges des sites. Mais, alors, l'objet-type doit être transmis au site m, ce qui accroit le traffic sur le SSC. Tels sont les termes du compromis de régulation de la charge. Pour décider le noyau dispose des informations suivantes :

- une **Table de Charge du Système** (TCS), renfermant la valeur de la charge de chacun des sites.
- Une indication de charge du SSC.
- La valeur de "l'encombrement message" de l'objet-type, i.e. la taille totale des messages nécessaires pour le transmettre d'un site à un autre.

- La valeur de "l'encombrement mémoire" de l'objet à créer, i.e. la taille de sa future représentation en mémoire.

(Sur les deux derniers points, voir § 5.8.3).

En fait, l'objet-type est ré-entrant et sa représentation peut être copiée (voir § 5.6.2). Si le site m est choisi un **Message-Demande de Copie de l'Objet-type** (MDCOT) est envoyé. Ce message renferme les informations contenues dans le MDIM initial et un numéro de demande. Le site m a la possibilité de refuser la demande de copie, et de transmettre le MDCOT à un site qu'il estime moins chargé que lui. Mais le nombre de ces transferts est limité. Le site qui accepte la demande répond par l'envoi d'un **Message Ordre de Transfert** destiné au site n'.

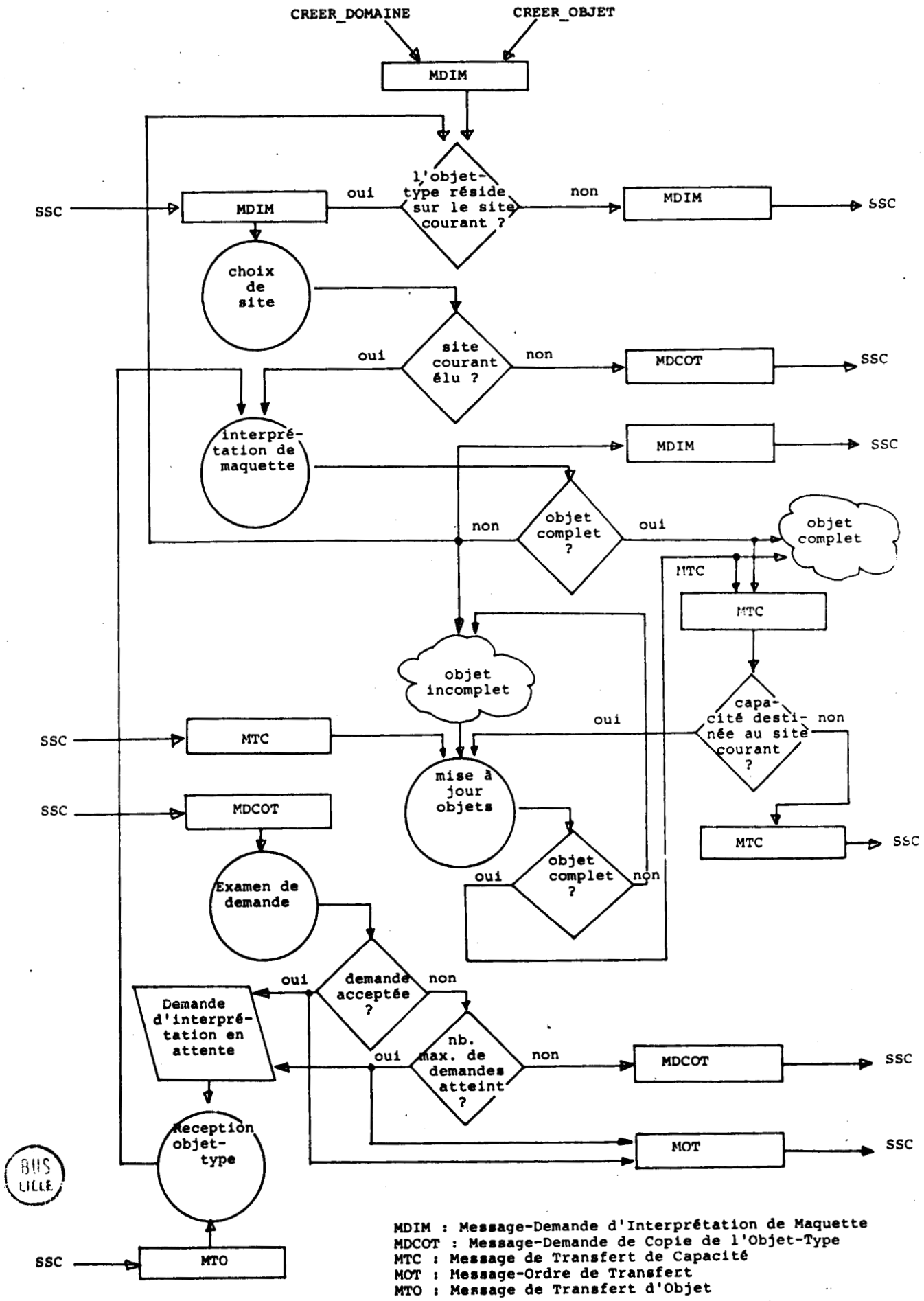
Un site p est donc finalement choisi pour interpréter la maquette. Les actions déclenchées par l'interprétation d'une commande de copie, d'annulation ou de création d'un objet primitif (voir § 3.2.2) sont entreprises sur le site p. Par contre, la création d'un composant de type construit est répartie. L'objet en cours de création est alors considéré comme incomplet et un MDIM du composant est généré. A l'issue du processus de création du composant une capacité est retournée au site demandeur au moyen d'un MTC. L'objet est considéré comme complet quand sa C-liste renferme toutes les capacités pour les composants ; un compteur mémorise le nombre de MDIM de composants émis ; il est décrémenté de un à chaque réception de MTC ; quand il vaut zéro l'objet est complet. Une capacité pour l'objet qui vient d'être créé est générée et transmise au site qui a initié le processus créateur au moyen d'un MTC. Le schéma de la page 102 résume l'enchaînement des traitements liés à la création des objets pour le noyau d'un site n.

5.6.2 *Copie de l'objet-type*

Un objet-type T est repéré par un nom global (n, v). La copie de l'objet-type d'un site n à un site m s'effectue en trois temps :

1) Transfert du nom global. A l'issue de cette opération l'interface du site m au SSC acceptera tout message destiné "au site possédant l'objet de nom (n, v)". Le transfert doit être indivisible, ce qui repose sur le fonctionnement du SSC.

2) Transfert du descripteur. Le descripteur est retiré de la TOG du site n et placé dans la TOG du site m.


CREER_DOMAINE
CREER_OBJET
MDIM
SSC
MDIM
oui
l'objet-type réside sur le site courant ?
non
MDIM
SSC
choix de site
oui
site courant élu ?
non
MDCOT
SSC
interprétation de maquette
MDIM
SSC
non
objet complet ?
oui
objet complet
MTC
MTC
objet incomplet
SSC
MTC
oui
capacité destinée au site courant ?
non
SSC
MDCOT
mise à jour objets
MTC
SSC
Examen de demande
oui
objet complet ?
non
oui
demande acceptée ?
non
Demande d'interprétation en attente
oui
nb. max. de demandes atteint ?
non
MDCOT
SSC
MOT
SSC
Reception objet-type
SSC
MTO
MDIM : Message-Demande d'Interprétation de Maquette
MDCOT : Message-Demande de Copie de l'Objet-Type
MTC : Message de Transfert de Capacité
MOT : Message-Ordre de Transfert
MTO : Message de Transfert d'Objet

BUS LILLE

3) Copie de la représentation de l'objet-type. Une copie de chacune des maquettes composant l'objet-type est envoyée au site m. Sur le site m la C-liste représentant l'objet-type est reconstituée.
Et ce impérativement dans l'ordre 1), 2), 3).

Afin de pouvoir décider de l'utilité de la version originale résidant sur le site n, un compteur de demandes mémorise le nombre d'interprétations de maquettes en cours sur le site n, pour un objet-type T donné. Lors de la copie de l'objet type ce compteur est décrémenté de un, alors que celui de la nouvelle copie est positionné à un. L'original peut être détruit lorsque le compteur de demandes vaut zéro.

5.6.3 *Efficacité du processus créateur*

Le caractère récursif du processus créateur, qui, pour créer un objet construit, génère en cascade tous ses composants jusqu'aux objets primitifs, peut faire douter de son efficacité. Ainsi, Lanciaux ([34]) propose d'imbriquer toute la hiérarchie des maquettes des composants d'un objet construit dans une même structure de données, et de l'interpréter en une seule fois. Dans le cas d'OMPHALE, cette proposition est inapplicable, car elle entrainerait une concentration de toutes les maquettes d'un objet sur le même site et par suite des composants eux-mêmes, puis des domaines qui les manipulent, ce qui nuit à l'équilibre des charges des sites. Pour ce qui est de la possible inefficacité, elle réside dans les "overheads" de communication par messages. Mais ils sont compensés par des interprétations parallèles des maquettes des composants sur des sites différents.

5.7 DESTRUCTION DES OBJETS ET REPARTITION

5.7.1 *Répartition de l'opération de destruction*

L'exécution de l'opération

DETRUIRE c

où c est le nom local dans le domaine courant d'une capacité à détruire,

est répartie, car la capacité c réside sur le site courant n, alors que l'objet repéré réside sur un site n' et que, en général, n ≠ n'. Aussi un message de service signalant la destruction de la capacité est envoyé au site n'. Puis sur le site n, l'emplacement c est annulé. Lors de la réception du message sur le site n' l'objet peut être ou non partagé. Si oui, il possède un compteur de références ; ce compteur est alors décrémenté de un. Dans le cas contraire l'objet n'est plus, en réalité, référencé et sa représentation peut être détruite. Un indicateur est positionné dans le descripteur de l'objet pour signaler ce fait. De plus, toujours dans ce cas, si l'objet était référencé par un nom global, le nom doit être rendu au site qui l'a généré.

5.7.2 *Destruction des représentations des objets*

Il faut rappeler que la mémoire d'un site est organisée en pages indépendantes (voir § 4.3.3). Soit à intervalles réguliers, soit au moment où l'occupation de la page dépasse un certain seuil, le processeur noyau procéde à la destruction des objets non référencés de la page. Pour cela, il parcourt les descripteurs, consulte les indicateurs et détruit un à un les objets inutiles. Si l'objet est représenté par un segment, l'emplacement du segment est simplement libéré. Si l'objet est repésenté par une C-liste, la C-liste est parcourue et l'opération DETRUIRE est appliquée à chaque capacité. La destruction des représentations des objets peut être combinée avec un compactage des objets encore référencés.

5.7.3 *Répartition du processus de localisation des objets perdus*

La localisation des objets perdus exige un parcours de l'arbre des références issues des C-listes représentant les éléments qui appartiennent à l'ensemble des objets pré-récursifs et visibles (noté V), et un marquage des objets récursifs rencontrés durant ce processus (voir § 3.3.2). Tout objet pré-récursif peut occuper trois états différents rendant compte de sa situation vis à vis du déroulement du processus ; objet "non examiné" (NE), objet "à examiner" (AE) ou objet "marqué" (M). Sur un site n donné un objet ne peut être atteint qu'à partir d'un domaine, soit en cours d'exécution, soit référencé dans la file d'attente des domaines prêts (en première approximation, voir § 5.7.4). Soit $D(n)$ l'ensemble de ces domaines.

Tout d'abord, le site n examine successivement chaque élément D de $\mathcal{D}(n)$. La C-liste représentant D est parcourue et chaque objet prérécursif référencé et résidant sur le site n passe dans l'état AE, s'il était auparavent dans l'état NE. Par contre, tout objet O prérécursif et ne résidant pas sur le site n provoque l'envoi d'un **Message-Demande d'Examen** (MDE) au site n' qui le posséde. A la réception de ce message l'objet O passe dans l'état AE s'il était dans l'état NE. Cette phase du processus de marquage est déclenchée par un site i quelconque selon un critère à définir (périodicité ? niveau donné de la charge du site ? ...). Le site i diffuse à tous les sites un message de service les informant du commencement du marquage, ce qui provoque l'examen des domaines de $\mathcal{D}(j)$ sur chaque site j pour $j \neq i$, et, donc, la donstruction de l'ensemble V : à l'issue de cette première phase tous les objets prérécursifs et visibles sont dans l'état AE.

Puis le noyau de chaque site procéde à l'examen de tous les objets "à examiner". Pour ce faire il dispose d'une liste des noms de ces objets, liste élaborée au fur et à mesure de la progression de la phase de construction de V. Ou bien il parcourt toute la mémoire locale du site en ne s'interessant qu'aux objets prérécursifs occupant l'état AE. La C-liste de chacun des objets est examinée. Un composant précursif occupant l'état NE passe dans l'état AE, s'il réside sur le site courant. Sinon un MDE est envoyé au site qui le posséde. A sa réception le MDE subit le même traitement que lors de la première phase de marquage. Quand tous ses composants ont été examinés, l'objet passe dans l'état "marqué". Le noyau maintient une liste des MDE qu'il reçoit durant cette seconde phase, liste qu'il traite ensuite de la même façon.

Le processus est terminé quand tous les noyaux ont examiné tous les objets occupant l'état AE. Pour qu'un noyau puisse décider de la terminaison, il questionne tous les sites pour savoir s'il existe encore un noyau exécutant la deuxième phase. Si non le processus est réellement terminé. Si oui, c'est ce dernier site qui décidera de sa terminaison suivant la même procédure. Cette méthode exige que les messages de questionnement soient acheminés en une seule opération indivisible. Cette forme de synchronisation peut être réalisée par le matériel au moyen de bascules spécialisées. Chaque site posséde une bascule qui est positionnée à un lors du marquage et remise à zéro quand le noyau a terminé son travail sur son site. La consultation du

résultat d'un OU portant sur les états de toutes les bascules (sorties à collecteurs ouverts reliées à une ligne spécialisée du SSC) renseigne sur la terminaison du marquage.

La localisation proprement dite des objets perdus peut alors être entreprise : sur chaque site la mémoire locale est parcourue et tout objet récursif non marqué est considéré comme perdu. La C-liste d'un objet perdu est immédiatement détruite, ce qui a pour effet de rompre les boucles de références. Durant ce parcours de la mémoire tous les objets récursifs non marqués sont réinitialisés dans l'état "Non examiné". Quand chaque site a effectué ce travail un second marquage peut être entamé Une procédure similaire à celle déterminant la fin du marquage, informe sur la terminaison de la phase d'initialisation.

Enfin, durant le marquage, les domaines peuvent créer et déplacer des objets prérécursifs. Ces objets sont nécessairement accessibles. Ainsi lors de la création d'un objet prérécursif, il est placé dans l'état AE. Et lors du déplacement d'un objet prérécursif d'un site n à un site n', son nom est placé dans la liste des objets à examiner sur le site n' (celle qui receuille les MDE reçus par le site n').

5.7.4 *Caractérisation de l'ensemble des domaines D(n)*

La propriété de prérécursité peut être étendue aux domaines : est prérécursif un domaine renfermant au moins un composant prérécursif, compte non tenu des domaines prérécursifs référencés (sinon tous les domaines du système seraient prérécursifs). Cette propriété intrinséque dépendant du type de l'opération réalisée par le domaine, peut être signalée dans la maquette du domaine. Aussi, il est inutile d'examiner tous les domaines référencés dans la File d'Attente des Domaines Prêts : l'examen des seuls domaines prérécursifs suffit. Mais il faut être certain que tous les objets récursifs non perdus peuvent être marqués, quelle que soit la complexité de leurs chaînes d'accès. Or les domaines peuvent être dans l'état bloqué ou suspendu (voir chapitre 6). Plutôt que de suivre de longues chaînes d'accès, mieux vaut examiner tous les domaines prérécursifs résidant sur un site.

5.8 REMARQUES DIVERSES

5.8.1 *Opération de transfert de capacité*

Le paragraphe 3.2.1 décrit deux opérations élémentaires de copie et de transfert de capacités (notées respectivement <= et ←). Ces opérations ne sont pas reparties et sont réservées à l'écriture du noyau. De même qu'une opération répartie de partage est fournie à l'utilisateur du système (système opératoire ou usager), une opération répartie de **transfert de capacité** entre domaines est mise à sa disposition. Cette opération s'écrit :

```
[d,] e <- c
```

où c est le nom local dans le domaine courant de la capacité à transférer
d est le nom local d'une capacité pour le domaine destination
e est le nom local d'un emplacement nul dans le domaine destination.
(Si d est omis, domaine courant et domaine destination sont confondus).
Elle provoque le transfert de la capacité à l'emplacement e dans le domaine de nom d et l'annulation de l'emplacement c dans le domaine courant. Son invocation est subordonnée au positionnement du **droit de transfert** figurant dans le champ droit de la capacité c. Avant exécution, la capacité c réside sur un site n et le domaine d sur un site n'. En général, n ≠ n'. Aussi, l'opération de transfert provoque l'envoi d'un MTC au site n'.

5.8.2 *Copie des segments de code*

De par leur propriété de ré-entrance les segments de code peuvent être partagés par copie (mais cette forme de partage n'est peut être pas souhaitable). Ils sont alors repérés par des noms spatiaux, même si ils sont mobiles et partagés.

5.8.3 *Caractéristiques quantitatives des objets*

La réalisation du compromis de régulation de la charge et le nécessaire contrôle des ressources consommées par un calcul imposent la définition de caractéristiques quantitatives des objets.

* *Encombrement-Mémoire d'un Objet*

La taille des segments de données et des segments de capacités est exprimée en **quanta de mémoire**, unité de mémoire commune à tous les sites. L'**encombrement mémoire** d'un objet est mesuré par le nombre de quanta consommés lors de la création. C'est la somme des encombrements-mémoire de tous ses composants jusqu'aux objets primitifs compris. Le calcul de l'encombrement mémoire est effectué avant la création de l'objet à partir d'informations contenues dans sa maquette : l'encombrement dit statique et des coefficients d'évaluation de l'encombrement dynamique. L'encombrement statique est indépendant des paramètres d'interprétation de la maquette alors que l'encombrement dynamique est calculé par la multiplication de certains de ces paramètres par les coefficients;

* *Encombrement-Message d'un Objet*

La taille des messages véhiculés par le SSC est exprimée en **quanta de communication**, unité commune à tous les sites. L'**encombrement-message** d'un objet est la taille totale des messages nécessaires à sa transmission d'un site à un autre. Cette transmission inclue le transfert du descripteur de l'objet, de sa C-liste, et des segments référencés dans la C-liste. L'encombrement-message d'un domaine cumule les encombrements-message des objets du domaine (définis au § 4.3.2, "ensemble des objets d'un domaine").

CHAPITRE 6

GESTION REPARTIE DES DOMAINES

6.1 TRANSFERT DE CONTROLE ET SYNCHRONISATION (Lanciaux [34])

Soit un domaine D. Dans le cas le plus général, par l'exécution d'**opérations de transfert de contrôle**, le domaine D a la possibilité de déclencher l'activation de domaines D_1, D_2, ..., D_n, en parallèle avec sa propre exécution. D est alors un "domaine appelant", ou encore, un "domaine père". Les domaines D_1, D_2, ..., D_n sont des "domaines appelés" ou des "domaines fils" de D. Ils sont aussi "frères" les uns des autres, car tous fils du même domaine D. Le domaine D a, de plus, la possibilité de suspendre son exécution et d'attendre qu'un (ou plusieurs) de ses fils, ayant terminé une certaine tâche, déclenche sa réactivation, selon un mécanisme à définir.

Afin de maintenir la cohérence des mécanismes de transfert de contrôle et d'éviter les incidents dus à leur mauvais emploi (étreintes fatales, activations multiples d'un même domaine, ...), les règles suivantes doivent être respectées :

- un domaine ne possède qu'un seul père,
- durant son exécution un domaine ne peut réactiver qu'une seule fois son père,
- un domaine ne peut être activé une deuxième fois qu'après avoir réactivé son père et terminé sa propre exécution.

Ainsi, un premier jeu d'opérations de transfert de contrôle régit les relations entre un domaine père et ses fils. Cependant certaines applications requièrent une synchronisation directe des exécutions de domaines. Toujours dans un souci de maintien de la cohérence, cette faculté est limitée à des domaines frères. Un second jeu d'opérations de transfert de contrôle permet, donc, de synchroniser l'exécution de domaines frères.

Par ailleurs, de nombreuses raisons conduisent à dissocier la création d'un domaine de son activation. Certaines ont déjà été indiquées (§ 3.2.3) :

- manque de généralité de l'appel procédural.
- Difficultés d'accès aux objets rémanents.

D'autres, plus générales, peuvent être formulées :

- La fourniture d'opérations élémentaires et universelles de transfert de contrôle, qui permettent de réaliser divers appels de domaines, est conforme à la conception d'ensemble du noyau (§ 2.4).

- L'inclusion des informations, qui supportent les opérations de transfert de contrôle, dans les représentations des domaines, rend les domaines autonomes et indépendants d'une structure de données centralisée (qui représenterait le processus). Cette autonomie accroit la mobilité des domaines et renforce la disponibilité du système.

L'autonomie des domaines est assurée par les dispositions suivantes. Tout d'abord, à chaque C-liste représentant un domaine est associée une zone de données, subdivisée en une zone de sauvegarde et une zone d'initialisation. La **zone de sauvegarde** conserve l'état du domaine quand il est suspendu : elle renferme les registres de données du processeur de calcul, l'état de la logique de topographie et la valeur du compteur ordinal. La **zone d'initialisation** renferme les informations nécessaires pour ré-initialiser un domaine après son exécution. Ces informations de ré-initialisation sont indispensables pour garantir la généralité des mécanismes de transfert de contrôle : avant retour à l'appelant un domaine activé selon un appel procédural doit être ré-initialisé, alors qu'à la fin de l'exécution d'une coroutine, le domaine doit être laissé en l'état. Puis, les liens entre domaine appelant et domaine appelé, ainsi que les chaînes d'accès aux paramètres sont matérialisés par des capacités incluses dans les C-listes représentant les domaines. Il en est de même, enfin, pour les ressources mémoire et temps de calcul consommables par un calcul.

Ainsi, par convention, les cinq emplacements de noms locaux 1 à 5 dans la C-liste représentant un domaine, sont réservés, dans l'ordre des noms croissants, à des capacités pour :

1) le domaine appelant
2) la C-liste des paramètres reçus
3) le nombre de quanta de temps encore consommables par le calcul
4) le nombre de quanta de mémoire encore consommables par le calcul
5) un domaine gérant les déroutements.

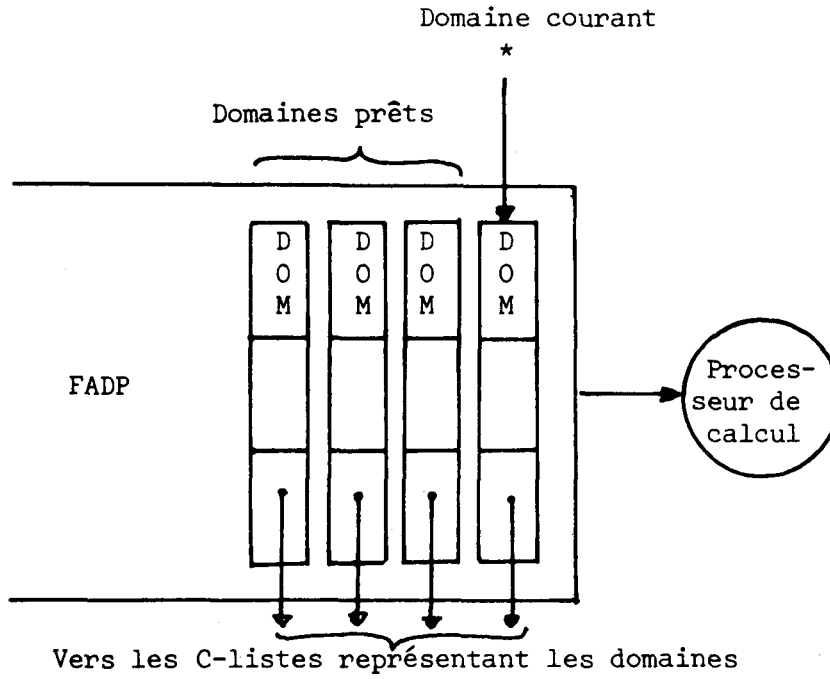
Les opérations de transfert de contrôle permettent de programmer le transfert des paramètres, l'activation d'un domaine, l'établissement de liaisons appelant/appelé entre domaines, etc ... D'autres permettent d'allouer des quanta de temps ou de mémoire à un domaine.

Les opérations de transfert de contrôle sont décrites au paragraphe 6.2 ; les opérations d'allocations des ressources au paragraphe 6.4 ; et le paragraphe 6.5 présente des exemples d'utilisation.

Mais, il ne suffit pas de mettre à la disposition de l'utilisateur du noyau des opérations lui permettant d'enchaîner l'exécution de domaines, il faut aussi lui donner des moyens de synchroniser des séries de domaines réalisant des calculs différents (le problème de l'exclusion mutuelle ne se pose pas - voir § 5.3 "limitation des possibilités de partage"). La fourniture d'un mécanisme de synchronisation "directe" entre domaines - un domaine bloque l'exécution d'un autre domaine en le désignant par son nom (Crocus [7]) - semble insuffisante. Car, dans de nombreux cas, il est nécessaire qu'un domaine puisse bloquer (ou débloquer) l'exécution d'un ou plusieurs domaines dont il ne connait pas l'identité. Bien que la question du choix d'un mécanisme de synchronisation indirecte n'ait pas été examinée en détail, l'exemple de l'implantation de sémaphores est décrit rapidement, dans le seul but de monter la faisabilité du dispositif, au paragraphe 6.3.

6.2 OPERATIONS DE TRANSFERT DE CONTROLE

Chaque site n est équipé d'une **File d'Attente des Domaines Prêts** (FADP) à être exécutés, qui contient les capacités pour les domaines, dont le site d'exécution choisi est n, et dont tous les composants utiles à l'exécution résident sur le site. Le domaine repéré en tête de la FADP est en cours d'exécution : c'est le **domaine courant**. Les autres domaines repérés dans la File sont les **domaines prêts** à être exécutés. Le processeur de calcul du site exécute les domaines prêts dans l'ordre de progression de la FADP. La capacité pour le domaine courant peut être manipulée par certaines opérations nucléaires. Elle est désignée par un nom interne au noyau et noté *.



Soient, dans un domaine courant D_1, d le nom local pour d' une capacité pour un domaine D_2 ; et p_1, p_2, ..., p_q q noms locaux. Les opérations élémentaires de transfert de contrôle sont les suivantes :

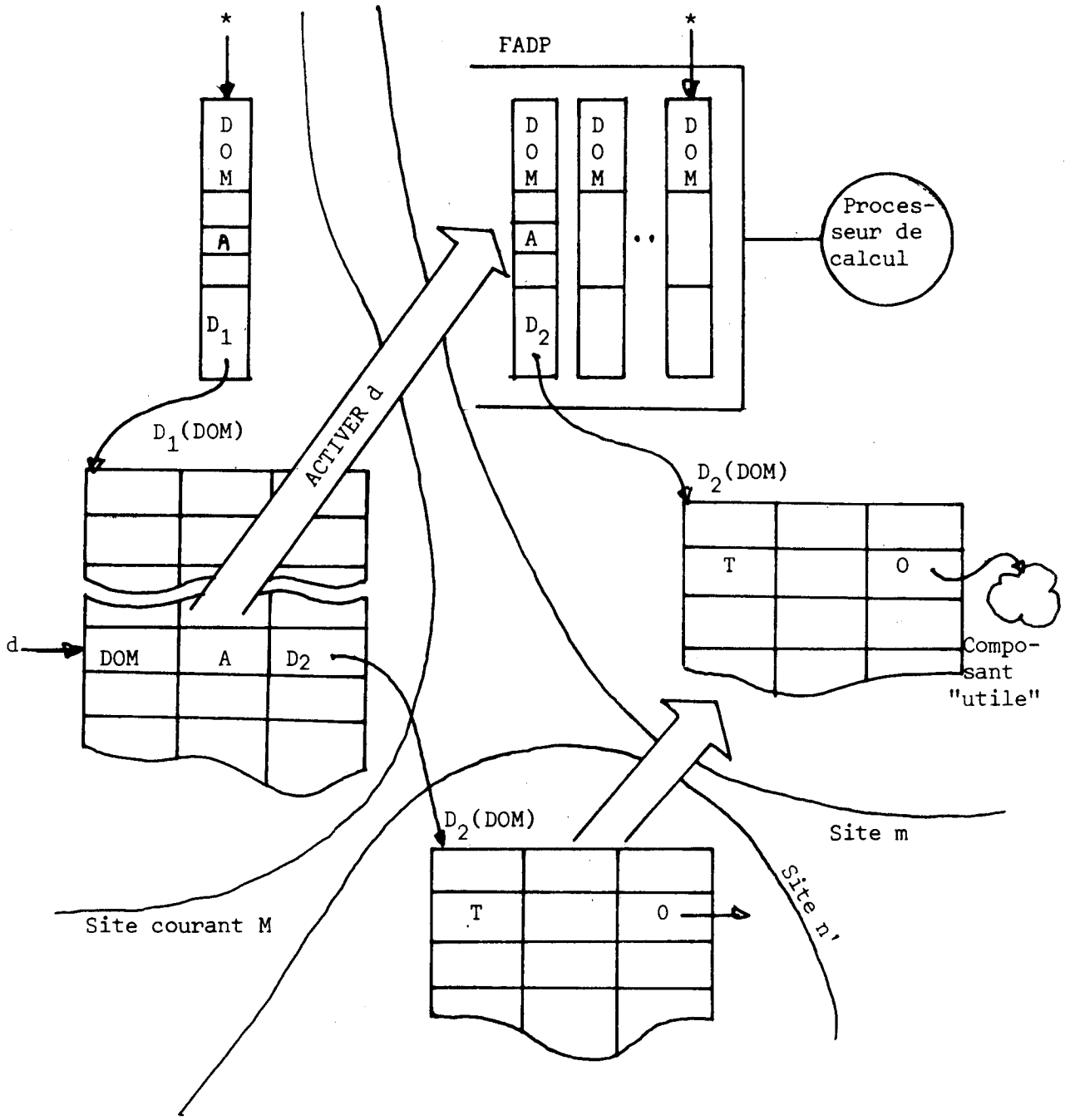
6.2.1 *Opération d'activation*

L'**opération d'activation**, notée

ACTIVER d

provoque le choix d'un site m d'exécution pour D_2, le rassemblement des composants "utiles" à l'exécution du domaine sur ce site, puis la mise en queue d'une capacité pour le domaine dans la FADP du site m. Son déclenchement est subordonné à la présence dans la capacité de nom d d'un **droit d'activation** noté A.

L'opération d'activation est répartie :



* *Répartition de l'opération d'activation*

Au moment du déclenchement de l'opération

ACTIVER d

le domaine D_2 réside sur un site m et tous les composants "utiles" - cette notion d'utilité reste à préciser - se trouvent, à priori, éparpillés sur tout le réseau. Mais seul le site m, qui possède la C-liste représentant le domaine, dispose d'informations sur cette répartition. Aussi, un **Message-Demande de choix de Site d'Activation** est envoyé au site m. Le MDCSA renferme

BUS LILLE

une copie de la capacité de nom d repérant D_2. Après réception de la demande, le site n effectue le choix du site d'exécution de D_2 en respectant le compromis de régulation de la charge. Plusieurs sites sont éligibles

- le site m lui-même
- le site le moins chargé
- tout site possédant un composant "utile" du domaine D_2.

Soit r le numéro du site élu. Si m ≠ r. Un **Message-Demande d'Activation** (MDA) est envoyé au site r. Le MDA renferme la capacité pour le domaine à activer, sa C-liste, et un numéro d'ordre de la demande. Le site r a la possibilité de refuser la demande d'activation et de la transmettre à un site r', qu'il estime capable d'assurer un meilleur compromis de régulation de la charge que lui. Cependant, afin d'éviter des migrations sans fin de la demande d'activation, le nombre de refus est limité grâce à un contrôle sur le numéro d'ordre inclus dans le MDA.

Finalement un site s est retenu comme site d'exécution du domaine D_2. Le noyau du site s procéde alors au rassemblement des composants de D_2 dans la mémoire locale. Pour cela, in envoie autant de **Messages-Ordre de Transfert** (MOT) que de composants "utiles" ne résidant pas sur le site s. Le domaine est considéré comme incomplet. Afin d'éviter les étreintes fatales provoquées par des ordres simultanés portant sur des objets partagés (voir § 5.3, "Répartition de l'opération de partage"), tous les MOT concernant un même domaine sont acheminés en une seule opération indivisible du SSC. Un compteur mémorise le nombre de MOT envoyés pour un même domaine. Les sites impliqués répondent par des MTO (Messages de Transfert d'Objet) contenant les composants. A leur réception, les MTO sont comptés. Quand leur nombre atteint le nombre de MOT émis, le domaine est considéré comme prêt. La capacité pour le domaine D_2, qui résidait, auparavant, dans une structure de données nucléaire, est placée en queue de la FADP.

Le schéma de la page suivante résume les traitements relatifs à l'activation des domaines et réalisés par un site.

Il est envisageable de charger les composants d'un domaine dans la mémoire locale du site choisi, à la demande, lors des premiers accès aux composants. Mais cette manière de procéder imposerait de suspendre l'exécution

du domaine courant à chaque premier accès, jusqu'à ce que le composant parvienne sur le site courant. Comme un délai de plusieurs millisecondes en résulterait, il serait, alors, nécessaire de multiprogrammer le processeur de calcul. Un accroissement de la complexité du noyau s'ajouterait, donc, à une perte de performances. Car le pré-chargement des composants est, ici, effectué par le processeur noyau en parallèle avec la progression de la FADP. De plus, il faudrait se protéger des étreintes fatales.

* *Cohérence des mécanismes de transfert de contrôle*

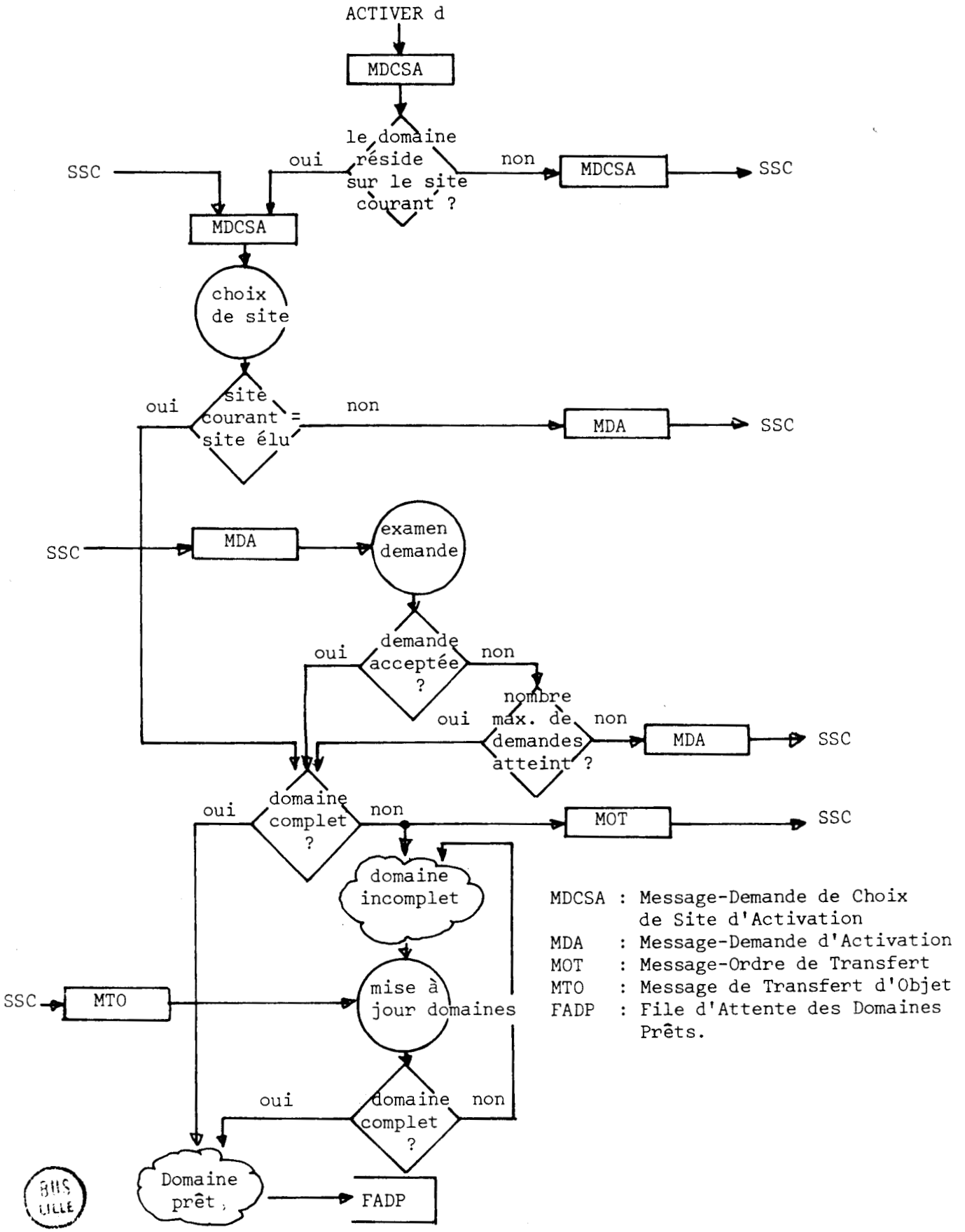
Après copie de la capacité pour le domaine à activer dans le MDCSA, l'original occupant l'emplacement d est privé du droit d'activation. Ainsi, une nouvelle activation du domaine fils est impossible. En fait, les opérations de transfert de contrôle sonc conçues de sorte que, en règle générale, le droit A soit transféré d'une capacité à une autre et qu'il ne soit copié que dans des cas bien définis.

* *Composants utiles à l'exécution d'un domaine*

Les composants utiles à l'exécution d'un domaine sont les objets potentiellement accessibles durant l'exécution. Ils doivent être caractérisés précisément afin de réserver les zones correspondant à leur représentation en mémoire. Par définition, les éléments de l'ensemble des objets d'un domaine (voir § 4.3.2) sont utiles à l'exécution de ce domaine. Dans la C-liste représentant un domaine, les emplacements destinés à des capacités repérant les objets utiles sont munis d'un indicateur. Lors de la phase de rassemblement des composants d'un domaine, sa C-liste est parcourue et la détection d'un emplacement de capacité dont l'indicateur est positionné déclenche l'envoi d'un MOT pour le composant.

* *Informations guidant le choix de site*

Afin d'effectuer un choix de site, le noyau dispose des valeurs des charges des sites contenues dans la Table de Charge du Système et de la valeur de la charge du SSC. De plus, en correspondance avec chaque capacité pour un composant du domaine, se trouve la valeur de son encombrement message. Pour chaque site éligible il est ainsi possible de chiffrer le coût du déplacement des objets, donc la valeur de la charge du SSC due à l'exécution du domaine. De plus, il est supposé que la taille des zones nécessaires pour représenter un objet en mémoire se déduit de son encombrement message par une fonction



MDCSA : Message-Demande de Choix de Site d'Activation
MDA : Message-Demande d'Activation
MOT : Message-Ordre de Transfert
MTO : Message de Transfert d'Objet
FADP : File d'Attente des Domaines Prêts.

BIS LILLE

connue du noyau (il s'agit, vraisemblablement, de retirer de l'encombrement message, la taille des en-têtes des messages). La taille des zones à réserver en mémoire peut donc être calculée. Ce résultat est intégré dans le calcul de l'accroissement de la charge apporté par l'exécution sur le site choisi. Pour chaque site éligible l'augmentation du traffic sur le SSC et le déséquilibre des charges des sites peuvent, donc, être calculés. Le site choisi est, en général, celui qui réalise le meilleur compromis entre ces deux fonctions antagonistes.

Cependant, ces quantités ne correspondent pas toujours à la réalité. Car quand un composant est repréré par un nom global, il est impossible de connaître le numéro de son site de résidence. Aussi, il faut envisager le cas le plus défavorable où tous les composants repérés par des noms globaux sont déplacés. L'augmentation calculée du traffic sur le SSC est, donc, un majorant de la charge réelle.

Par ailleurs, quand un domaine contient un objet immobile ou immobilisé le choix du site d'exécution est imposé (d'où l'utilité de l'indicateur de mobilité figurant dans les capacités). Et, lors de l'activation d'un domaine appelé en parallèle avec l'exécution du domaine appelant, le site de résidence du domaine appelant n'est pas éligible, afin de bénéficier d'un parallélisme d'exécution réel.

6.2.2 *Opérations d'attente et de réactivation*

* *Opération d'attente*

L'**opération d'attente** permet d'établir un lien appelant/appelé entre deux domaines. Elle déclenche la suspension de l'exécution de l'appelant, la sauvegarde de son état dans la zone de données prévue à cet effet et la progression de la FADP. Elle est notée :

ATTENDRE d

et elle provoque la copie de la capacité pour le domaine courant D_1 et son transfert à l'emplacement 1 du domaine D_2, emplacement réservé. La capacité * est détruite. L'emplacement d est annulé.

L'opération d'attente est répartie,car le domaine D_2 réside sur un site m en général différent du site courant n. La figure de la page 134 représente les liaisons entre appelant est appelé avant et après exécution de

ATTENDRE d

Un message de transfert de capacité contenant une copie de la capacité pour D_1 figurant en tête de la FADP est envoyée au site m.

* *opération de réactivation*

Grâce à la capacité figurant à l'emplacement 1, l'appelant peut être réactivé par l'appelé. Pour ce faire D_2 déclenche une **opération de réactivation**, notée

| REACTIVER |

Cet appel provoque la copie de la capacité pour D_1 (appelant) et sa transmission au moyen d'un **Message-Ordre de Réactivation** (MOR). Le noyau du site n examine, ensuite, la C-liste de D_1, et, si le domaine est incomplet - des composants partagés de D_1 ont pu être déplacés durant la suspension de son exécution - des MOT sont envoyés. Lorsque les MTO correspondants ont été reçus, la capacité pour D_1 est placée dans la FADP.

L'exécution de l'opération de réactivation est soumise à la présence du droit d'activation dans la capacité pour le domaine père. Et la réactivation de l'appelant provoque l'annulation de ce droit dans la capacité occupant l'emplacement 1 du domaine appelé. Ainsi le domaine père ne peut être réactivé qu'une seule fois par son fils.

Remarques

1) Afin de protéger l'emplacement 1 de toute malveillance, il est possible, lors de l'exécution de l'opération d'attente, de "sceller" la capacité de type DOM, en construisant sur le type DOM un type LIEN :

Soient l le nom local d'une capacité pour l'objet-type LIEN, munie du droit de composer

t le nom local d'une capacité pour l'objet-type DOM, munie du droit de décomposer

c, c' deux autres noms locaux

l'opération s'écrit :

c' < = DECOMPOSER (*, t, <droits>)

c < = COMPOSER (c, 1)

DETRUIRE c'

L'emplacement de nom c contient une capacité de type LIEN pour le domaine courant. L'opération de réactivation procède, bien sûr, en sens inverse.

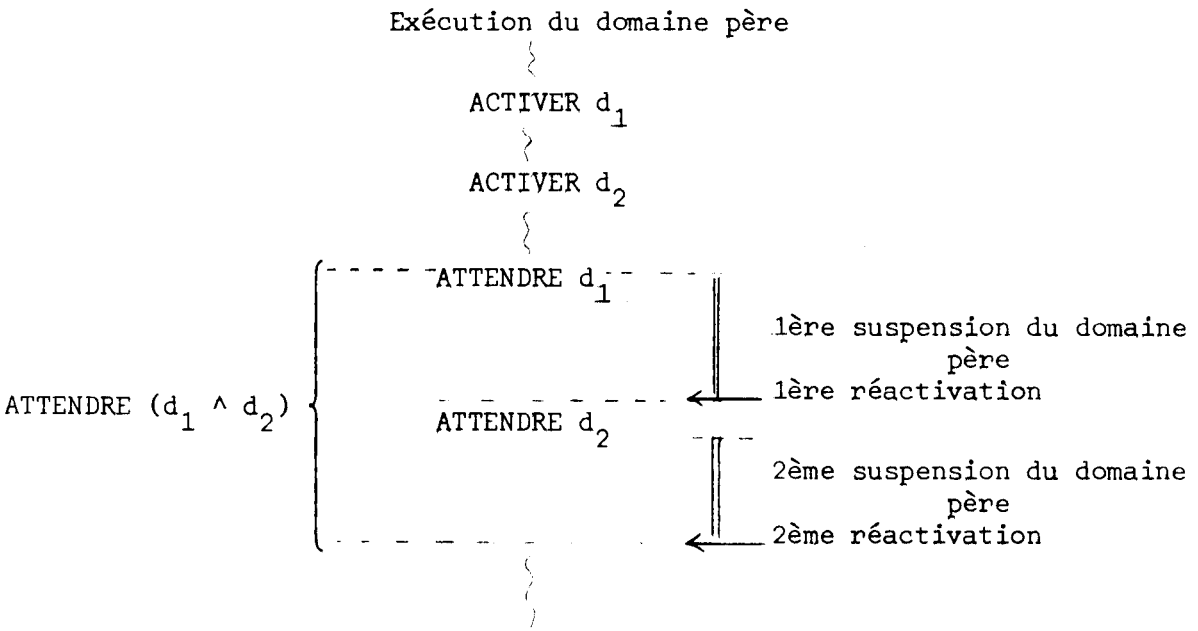
2) Il est souhaitable de fournir une opération d'attente plus élaborée portant plusieurs domaines fils. Pour deux domaines de noms locaux d_1 et d_2.

ATTENDRE ($d_1 \wedge d_2$)

provoquerait la réactivation de l'appelant après le déclenchement par chacun des domaines fils de l'opération de réactivation, alors que

ATTENDRE ($d_1 \vee d_2$)

conduisant à cette réactivation dès l'intervention de l'un quelconque des domaines fils. Le cas "et" est réalisable avec l'opération d'attente préalablement définie, mais il est inefficace :



La nouvelle opération d'attente exige la copie en deux exemplaires, tous deux munis du droit A, de la capacité pour le domaine père. Chaque réactivation conduit à l'examen de la condition d'attente - "et" ou bien "ou". Au cas où la condition n'est pas réalisée au premier examen ("et") la capacité pour le

domaine père (reçue dans le MOR) est détruite. Mais le résultat du premier examen doit toujours être mémorisé : la seconde réactivation doit être ignorée - et la capacité pour l'appelant détruite - dans le cas d'un "ou", alors que c'est cette seconde opération qui est effective dans le cas d'un "et".

L'opération d'attente peut être généralisée à une combinaison de "et" et de "ou" sur n domaines frères.

3) Il est envisageable de diviser l'opération ATTENDRE en deux opérations, l'une de liaison appelant/appelé, l'autre de suspension de l'exécution de l'appelant (Lanciaux [34]). Mais il semblerait que les deux opérations soient toujours employées l'une à la suite de l'autre. La distinction serait, alors, inutile.

6.2.3 *Opérations de fin d'exécution*

L'exécution d'un domaine se termine par l'appel d'une **opération d'abandon**, notée

QUITTER

ou d'une **opération d'arrêt**, notée

ARRETER

Ces deux opérations provoquent la destruction de la capacité pour le domaine et la progression de FADP. Les Files d'Attente d'Ordres de Transfert progressent également (voir § 6.6.2). De plus, l'exécution de QUITTER entraine la mémorisation des contenus des registres de données du processeur de calcul, de l'état de la logique de topographie et de la valeur du compteur ordinal dans la zone de sauvegarde du domaine. Tandis que l'exécution de ARRETER conduit à la ré-initialisation de ces mêmes informations à partir des indications de la zone d'initialisation. La fourniture de deux opérations permet de distinguer la fin d'exécution d'une coroutine de la fin d'exécution d'une procédure.

De plus, il est nécessaire de replacer D_1 et D_2 dans les états qu'ils occupaient avant l'exécution de D_2. En particulier D_2 doit être repéré par

une capacité placée en d dans la C-liste de D_1. Un Message de Transfert de Capacité est acheminé vers le site n. Il contient une capacité, munie du droit d'activation, pour le domaine en cours d'exécution (i.e. D_2 quand le message est généré). L'index d a été transmis à D_2 lors de son activation : il est inclus dans le MDCSA et dans les éventuels MDA. Cet index figure aussi dans le MTC. Il n'est pas possible de conserver d dans la représentation de D_1, car D_1 a pu activer plusieurs domaines et rien ne distingue la fin de l'exécution d'un domaine de celle d'un autre. En ce qui concerne D_2, l'emplacement 1 a été annulé.

6.2.4 *Opérations de manipulation des paramètres*

L'**opération de transfert des paramètres**, notée

d <- ∿ TRANSFERER_PARAMETRES (p_1, p_1, ..., p_q)

provoque, sur le site courant, le rassemblement des capacités de noms locaux p_1, p_2, ..., p_q dans une C-liste et l'envoi de cette liste au site possédant le domaine D_2. Dans ce domaine, la C-liste est repérée par une capacité occupant l'emplacement réservé 2. Cette opération est, bien évidemment répartie.

En fin d'exécution de D_2, l'appel de l'**opération de retour des paramètres**, notée

RETOURNER_PARAMETRES

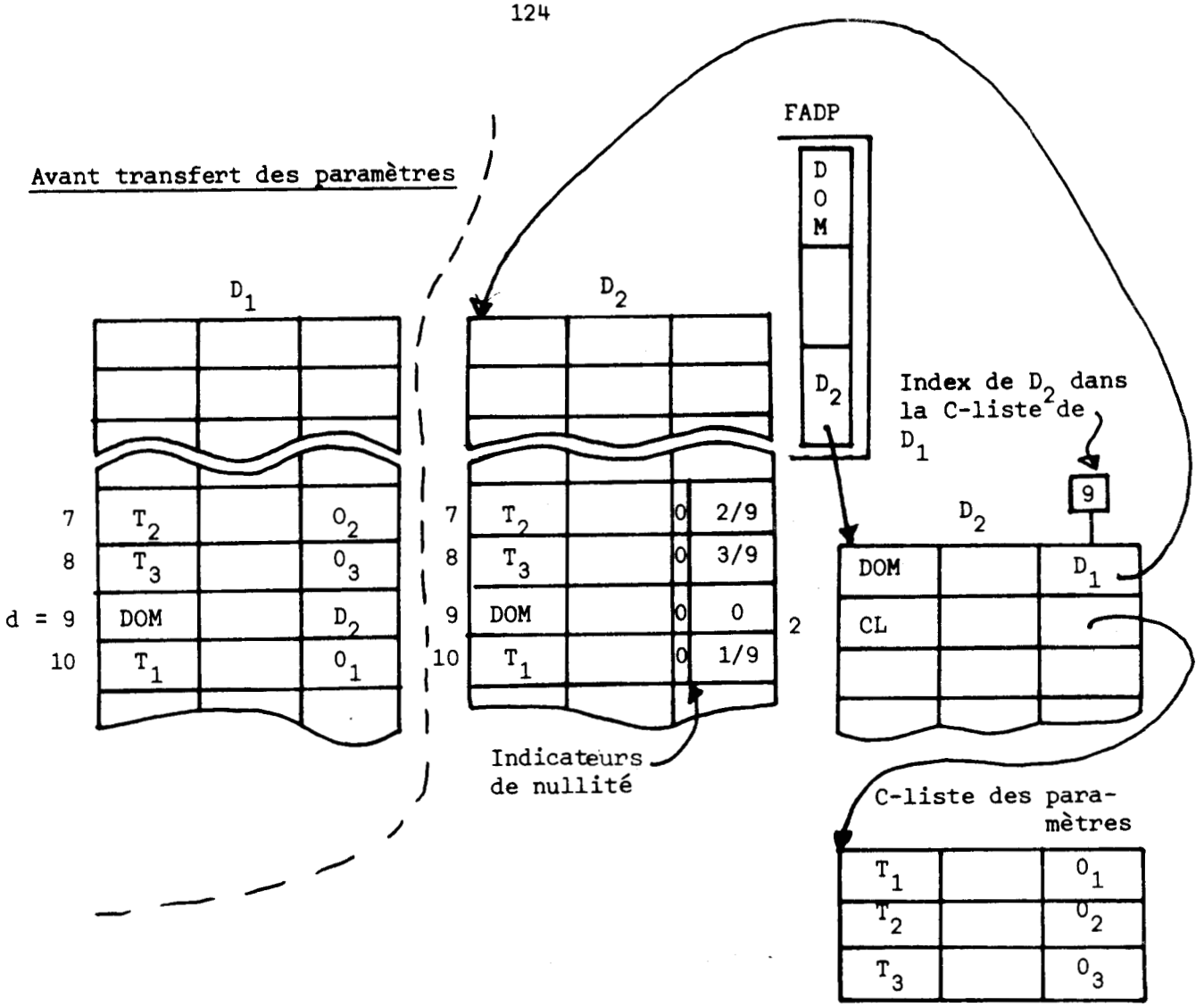
provoque le retour des paramètres au domaine appelant. Afin qu'une capacité puisse réintégrer l'emplacement qu'elle occupait avant l'exécution de l'appellé, son index dans la C-liste des paramètres est mémorisé dans le champ représentation de son emplacement source avec l'index du domaine appelé.

Exemple : voir page suivante.

Remarque :

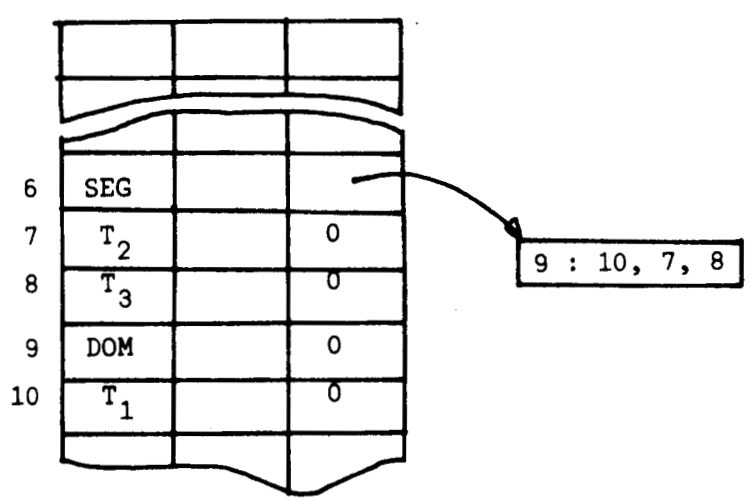
Afin d'éviter un parcours de la C-liste de l'appelant dans le but de retrouver les emplacements des capacités paramètres, il est possible de regrouper les index, soit dans la zone de données associée au domaine, soit dans un segment.

Voir l'exemple page suivante.



Après exécution de 9 < - ∿ TRANSFERER_PARAMETRES (10, 7, 8)

Cas où les index sont mémorisés dans un segment :



BUS LILLE

6.2.5 *Création des objets locaux*

Jusqu'à ce point de l'étude, la création, puis l'accès à un objet durant l'exécution d'un même domaine a de fortes chances de produire une erreur : l'objet est créé sur un site à priori différent du site courant, ce qui le rend inaccessible. Ce problème peut être contourné en considérant ce cas comme une faute de programmation. Mais il correspond à une situation fort courante : dans le cas de l'appel procédural d'un domaine contenant des objets rémanents les objets locaux doivent être explicitement créés lors de l'appel du domaine. Aussi, le noyau fournit une **opération de création des objets locaux**, notée :

$$d <- \sim \text{CREER_LOCAUX } (m, c, p_1, \ldots, p_q)$$

où m est le nom local dans le domaine appelé d'une maquette d'une C-liste des objets locaux,

c est le nom local dans le domaine appelé de la C-liste à créer,

$p_1, \ldots, p_q$ sont les noms locaux des capacités repérant les paramètres d'interprétation de la maquette.

L'opération de création des locaux est déclenchée par le domaine appelant. Une C-liste des paramètres d'interprétation de la maquette des locaux est constituée et est envoyée, ainsi que les index m et c au site possédant le domaine appelé. Le noyau de ce site provoque l'interprétation de la maquette m. La capacité résultante est placée en c.

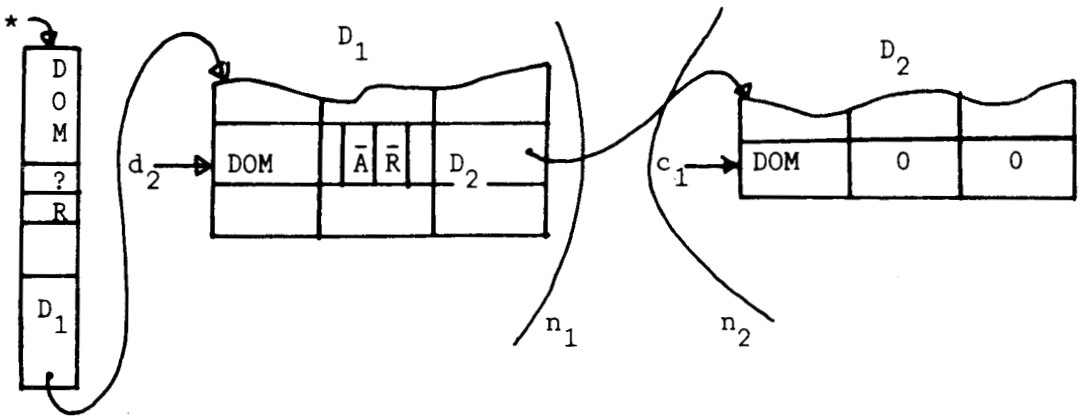
6.2.6 *Opérations de synchronisation directe de l'exécution de domaines frères*

Dans ce paragraphe il est supposé que deux domaines D_1 et D_2 sont fils d'un même domaine père D. Les opérations de synchronisation directe régissent les relations entre D_1 et D_2 et n'interviennent pas sur les liaisons entre appelant et appelés.

Deux opérations sont fournies par le noyau ; une opération d'attente permet à un domaine de suspendre son exécution ; une opération de réactivation permet d'activer un domaine suspendu. La propagation d'un **droit de réactivation**, noté R, contrôle l'emploi des opérations.

Soient d_2 le nom local dans D_1 d'une capacité pour D_2
c_1 un nom local dans D_2
n_1 le numéro du site de résidence de D_1
n_2 le numéro du site de résidence de D_2.

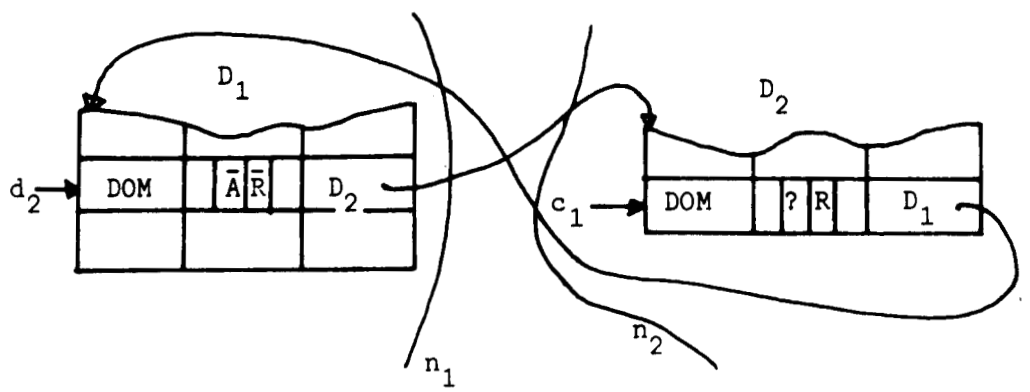
Il est supposé, qu'au départ la situation est la suivante :



Puis D_1 exécute l'opération d'attente, notée

ATTENDRE (d_2, c_1)

La capacité pour le domaine courant est transmise au site n_2 au moyen d'un message de transfert de capacité. Elle est placée en c_1 dans D_2. L'exécution du domaine D_1 est suspendue et son état est sauvegardé :



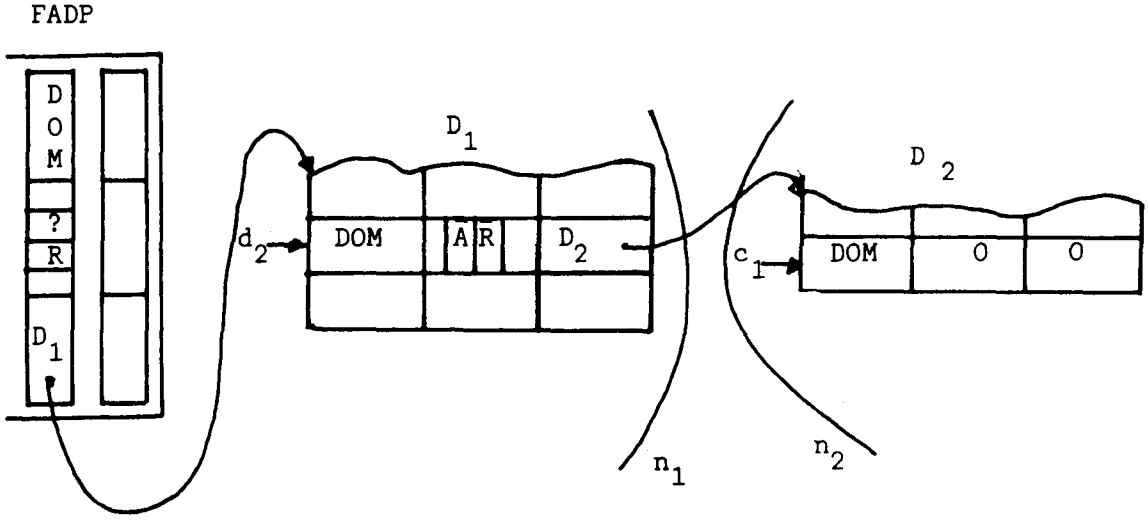
L'exécution par D_2 de l'opération de réactivation, notée

REACTIVER c_1

déclenche, de nouveau, l'exécution de D_1 : la capacité de nom c_1 est transférée au site n_1 au moyen d'un Message-Ordre de Réactivation. L'emplacement c_1 dans D_2 est annulé. Puis le MOR est traité comme dans le cas de la réactivation de l'appelant. La réactivation de D_1 est subordonnée à la présence du droit R dans la capacité de nom c_1. Les manipulations du droit R assurent, qu'un domaine, qui est réactivé, a été, auparavant, suspendu. Par suite des différences des vitesses d'exécution de D_1 et D_2, D_1 a pu exécuter

REACTIVER c_1

avant que la capacité pour D_1 ne soit présente. Dans ce cas, l'exécution du domaine D_2 est bloquée. Elle est débloquée par le noyau dès que la capacité parvient dans le site n_2



Un exemple d'utilisation des deux opérations est donné au paragraphe 6.5.4.

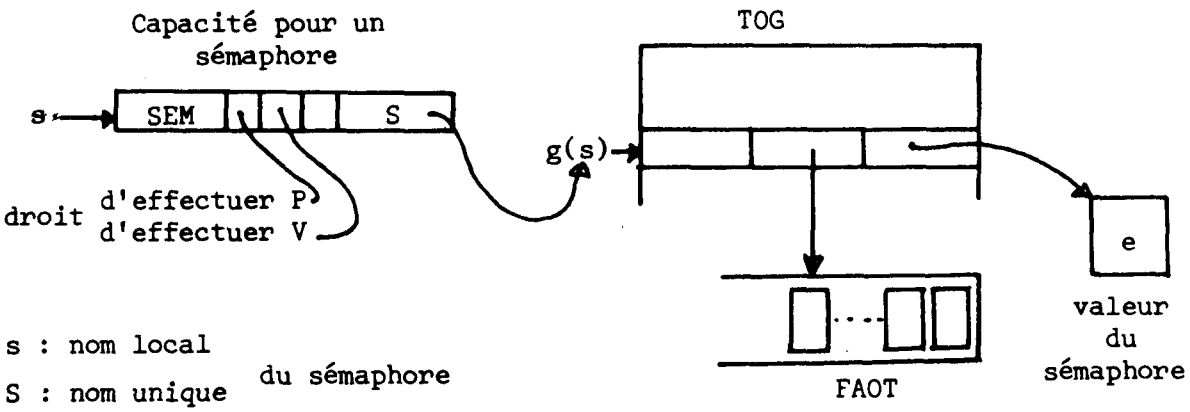
6.3 SYNCHRONISATION PAR SEMAPHORES

Un **sémaphore** s est constitué d'une variable entière e(s), appelée valeur du sémaphore et d'une file d'attente f(s). Deux opérations, notées P(S) et V(S), agissent sur le sémaphore (Crocus [7]) :

```
P(s) : DEBUT                          V(s) : DEBUT
       e(s) : = e(s) - 1 ;                   e(s) : = e(s) + 1 ;
       SI e(s) < 0 ALORS                     SI e(s) ≤ 0 ALORS
          DEBUT                                 DEBUT
          COMMENTAIRE : l'opération P           Sortir un domaine de f(s) ;
                        est exécutée            COMMENTAIRE : soit D ce
                        par un domaine D ;                    domaine ;
          état (D) : = BLOQUE ;                 état (D) : = ACTIF
          mettre D dans f(s)                    FIN
          FIN                               FIN ;
       FIN ;
```

A la création d'un sémaphore sa file d'attente est vide et sa valeur est positive ou nulle.

Des sémaphores peuvent être implantés dans le système OMPHALE. Le sémaphore est, alors, un objet nucléaire de type §EM, manipulable par deux opérations - P et V - dont l'emploi est contrôlé par des droits. La file d'attente du sémaphore est combinée avec la File d'Attente des Ordres de Transfert (FAOT, voir § 5.3) :



s : nom local
S : nom unique du sémaphore
g : fonction d'accès à la TOG

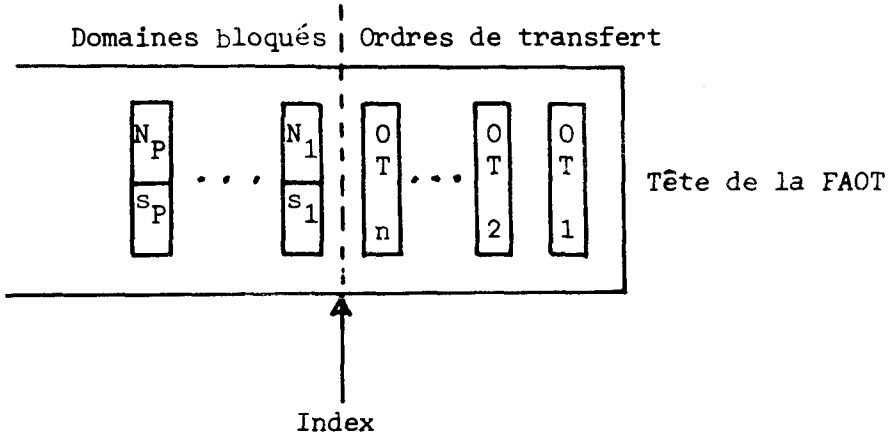
Le noyau fournit deux opération, notées

BLOQUER s	et	DEBLOQUER s

où s est le nom local d'une capacité pour un sémaphore. L'opération BLOQUER provoque la suspension de l'exécution de domaine courant et le transfert du couple

(nom unique du domaine, nom local du sémaphore)

en queue de la FAOT. En fait la FAOT est divisée en deux zones : la zone de tête contenant les ordres de transfert et la zone de queue contenant les couples associés aux domaines bloqués : un index repère la frontière entre les deux zones :



OT : ordre de transfert ; N nom unique de domaine bloqué ; s nom local de sémaphore.

L'opération DEBLOQUER provoque la réactivation du domaine repéré en tête de la zone des domaines bloqués. La zone de queue est décalée dans un sens ou dans l'autre suivant la progression autonome des ordres de transfert. Les opérations P et V s'écrivent, alors :

```
P(s) : DEBUT                    V(s) : DEBUT
       e - : = 1                       e + : = 1 ;
       si e < 0 ALORS BLOQUER s        SI e ≤ 0 ALORS DEBLOQUER s
       FIN ;                           FIN ;
```

6.4 ALLOCATION DES RESSOURCES MEMOIRE ET TEMPS DE CALCUL

Il est nécessaire de contrôler l'utilisation des ressources mémoire et temps de calcul, afin d'éviter qu'un traitement érroné ou malveillant ne monopolise toutes les ressources du système.

Le mécanisme d'allocation des ressources mémoire et temps de calcul est inspiré de la notion de "chunk" - mot traduit, ici, par "portion" - introduite par Prasad et al. ([56]). Un domaine en cours d'éxécution possède une **portion de temps** et une **portion de mémoire**, matérialisées par des capacités de type respectif TPS et MEM occupant les emplacements réservés 3 et 4 de sa C-liste. Ces capacités renferment dans leur champ représentation, sous

forme de valeur immédiate, un nombre positif T de quanta de temps et un nombre positif U de quanta de mémoire.

Soient, p_1 et p_2 les noms locaux de 2 portions ; q un nombre de quanta. Si la portion p_2 est, initialement, annulée, la portion p_1 peut être divisée au moyen de l'opération.

p_2 <- DIVISER (p_1, q)

La valeur de la portion p_1 passe de q_1 à q_1 - q ; celle de la portion p_2 de 0 à q. Deux portions peuvent aussi être fusionnées par l'intermédiaire de l'opération :

P_2 <- FUSIONNER p_1

La portion p_1 passe de la valeur q_1 à 0 ; celle de la portion p_2 de q_2 à $q_2 + q_1$. DIVISER et FUSIONNER sont les deux seules opérations manipulant les portions, mais dès qu'une valeur négative résulte de leur action il y a positionnement d'un indicateur d'anomalie. Un domaine gérant ces anomalies est repéré dans l'emplacement 5 de la C-liste du domaine courant.

* *Allocation de temps de calcul*

Le domaine courant a la possibilité d'allouer du temps de calcul à un autre domaine de nom local d, soit en transférant intégralement sa portion de temps, en éxécutant

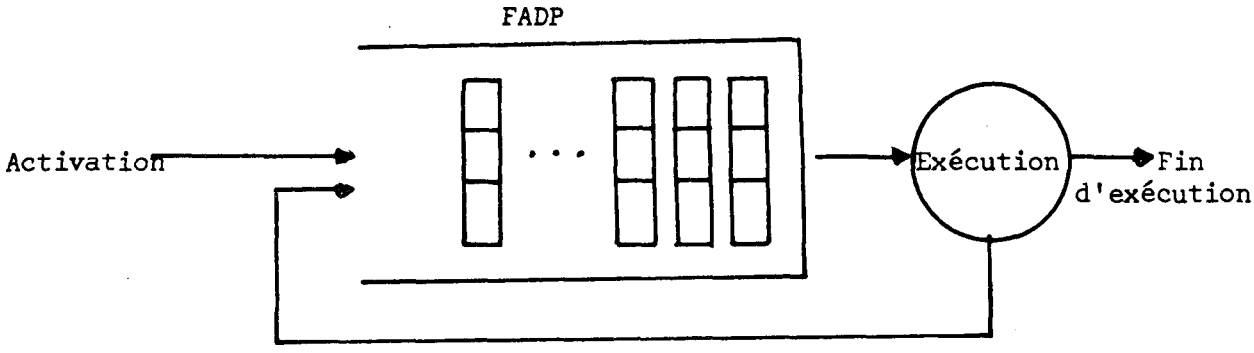
d.3 <~ 3

soit en exécutant une **opération d'allocation de temps**, notée

ALLOUER_TEMPS (d, t)

où t est le nom local d'une portion de temps de valeur T. Cette opération transmet la portion de nom t au site possédant le domaine d'où elle est fusionnée avec la portion de temps préexistante.

L'exécution d'un domaine consomme sa portion de temps. Un quantum de temps correspond à un cycle d'exécution. Avant exécution la portion de temps d'un domaine est decrémentéede un. Quand le cycle est terminé, la portion est de nouveau décrémentée de un. Si sa valeur est encore positive, le domaine est replacé en file d'attente :



Epuisement du quantum de temps

L'épuisement de la portion de temps est traité par le domaine de nom local 5, qui obtient, éventuellement, des portions de temps d'un domaine allocateur du système d'exploitation.

* *Allocation de mémoire*

Le domaine courant a la possibilité d'allouer de la mémoire à un domaine de nom local d en exécutant

d.4 <~ 4 ou | ALLOUER_MEMOIRE (d, m) |

où m est le nom local d'une portion de mémoire de valeur μ. Cette deuxième opération provoque l'envoi de la portion de mémoire au site possédant le domaine d et sa fusion avec celle du domaine.

La création d'objets consomme la portion de mémoire. L'exécution des opérations de création de domaine ou de création d'objet provoque le transfert de la portion de mémoire du domaine courant vers le site possédant l'objet-type (à l'intérieur du Message-Demande d'Interprétation de la Maquette). Sur ce site, avant que le choix du site de création ne soit effectué, l'encombrement mémoire de l'objet à créer, chiffré en quanta de mémoire, est évalué. La portion de mémoire est alors divisée. Si elle n'est pas épuisée le processus de création se poursuit. Sinon un message est retourné au domaine demandeur et le domaine gérant les anomalies est appelé.

Inversement, la destruction des objets doit régénérer la portion de mémoire du calcul courant. Ce qui pose problème, puisque la destruction des objets est effectuée implicitement après la destruction des capacités qui les repèrent : quand la représentation d'un objet est détruite le noyau ne possède plus aucune trace de l'identité du domaine qui renfermait la dernière capacité. Une solution consiste à considérer une et une seule capacité comme "propriétaire" d'un objet partagé et à allouer l'encombrement mémoire d'un

objet au domaine qui détruit cette capacité. Ce qui suppose que la valeur de l'encombrement mémoire d'un objet soit contenu dans sa représentation, que la copie d'une capacité puisse s'accompagner d'un transfert de propriété et que la capacité pour un objet nouvellement créé soit propriétaire de l'objet.

* *Domaines allocateurs*

Ce schéma d'allocation des ressources suppose qu'il existe dans le système d'exploitation un ou plusieurs domaines allocateurs capables de créer des portions de temps et de mémoire.

6.5 EXEMPLES DE PROGRAMMATION DES APPELS DE DOMAINES

Trois exemples sont décrits :

- Un appel procédural classique
- un appel parallèle "bifurquer" ("fork") et son corollaire, l'opération "rejoindre" ("join")
- une utilisation des opérations de synchronisation directe des domaines frères.

Une organisaton des domaines qui admet, en particulier, les deux premières sortes d'appel est tout d'abord examinée.

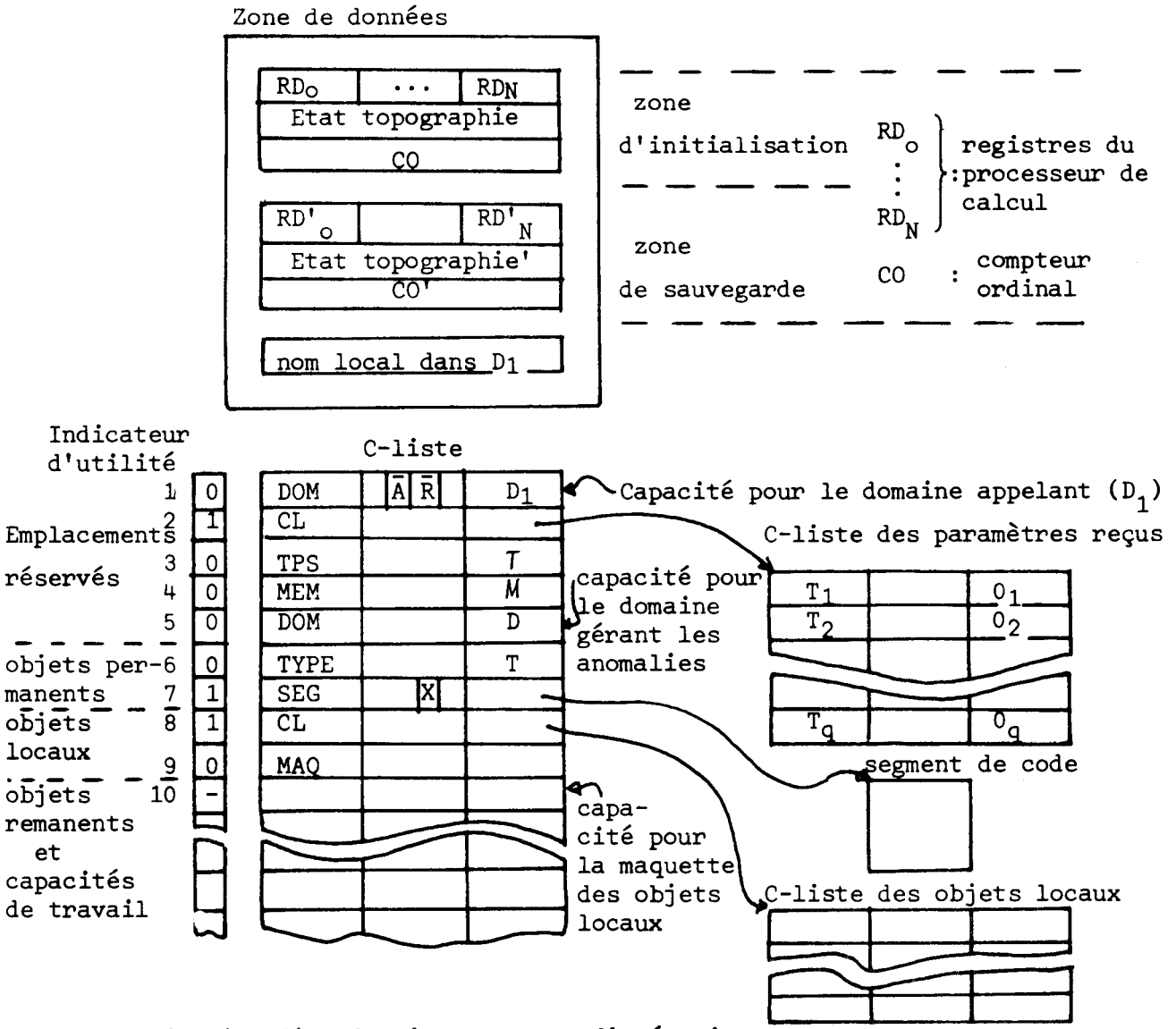
6.5.1 *Organisation d'un domaine*

La figure de la page suivante représente l'organisation d'un domaine D_2, supposé en cours d'exécution après activation par un domaine appelant D_1.

La zone de données et les 5 premiers emplacements de la C-liste de D_2 respectent les règles universelles de structuration des domaines exposées aux paragraphes précédents. Par contre, la disposition des autres capacités provient de conventions établies par le système opératoire, mais inconnues du noyau.

6.5.2 *Appel procédural*

L'appel procédural de D_2 par D_1 et le retour à l'appelant sont programmés comme indiqué page 132. Il est supposé que les objets remanents ont été créés en même temps que le domaine. Les points suivants sont à remarquer :



Organisation d'un domaine en cours d'exécution

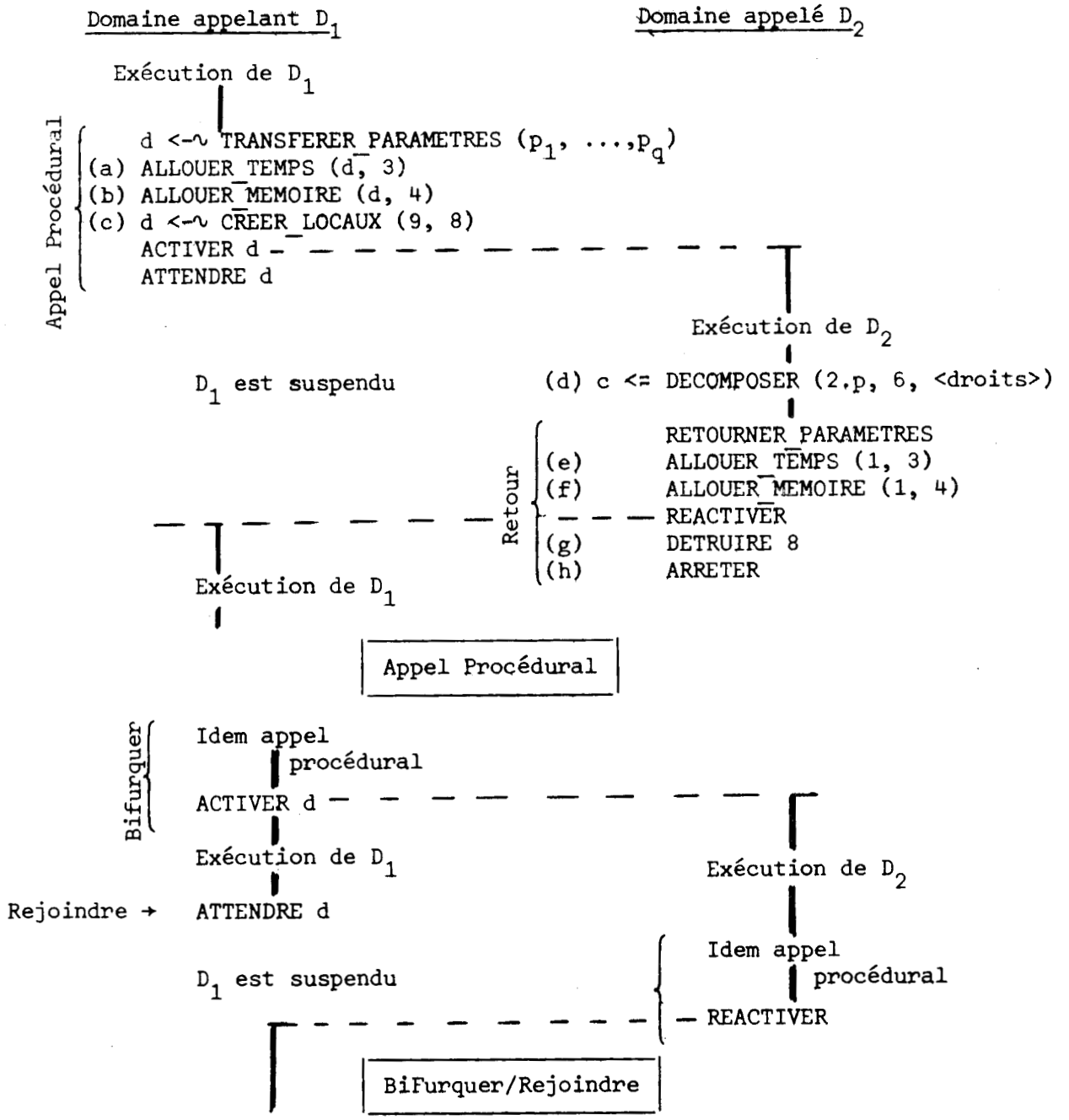
- les objets locaux sont créés au moment de l'activaiton de D_2(c) selon la maquette repérée par la capacité de nom local 9 dans D_2. Il est supposé que la création des locaux n'exige pas la transmission de paramètres d'interprétation.

- Toutes les ressources de l'appelant sont allouées à l'appelé (a et b).

Les ressources restantes après exécution de l'appelé sont rendues à l'appelant (e et f)

- une fois activé, D_2 peut accéder aux objets du type T en décomposant leurs capacités (d).

- en fin d'exécution de l'appelé, les objets locaux sont détruits (g) et le domaine est réinitialisé.



Notations : d : nom local de D_2 dans D_1

p_1, ..., p_q noms locaux dans D_1

p : nom local dans la C-liste des paramètres reçus de D_2

c : nom local dans D_2.

Les mouvements des capacités durant la séquence d'appel de D_2 et de retour à D_1 sont visualisés page suivante.

6.5.3 Appel "Bifurquer"/"Rejoindre"

Les programmes correspondant aux opérations bifurquer et rejoindre sont donnés page 132.

L'appel bifurquer ne diffère de l'appel procédural que par

- la séparation des opérations ACTIVER et ATTENDRE qui permet l'exécution parallèle de l'appelant et de l'appelé.
- l'allocation des ressources qui s'écrit
  - (a) ALLOUER_TEMPS (d, t)
  - (b) ALLOUER_MEMOIRE (d, m)

où t et m sont, respectivement, les noms locaux dans D_1 d'une portion de temps et d'une portion de mémoire. Il n'est pas possible de priver D_1 de ses ressources, puisqu'il doit poursuivre son exécution.

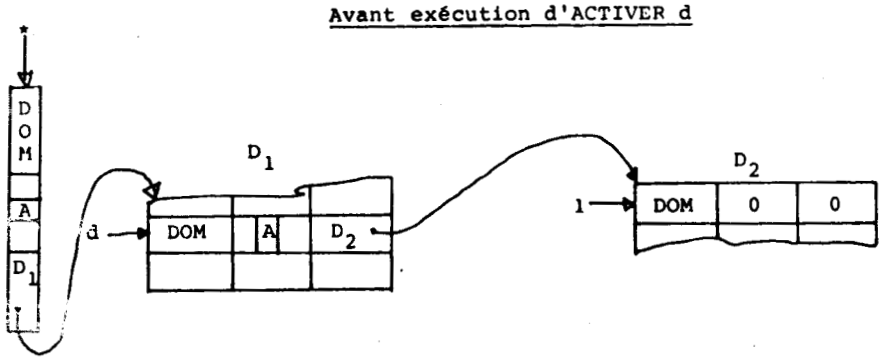
Le retour à l'appelant est identique au retour d'un appel procédural.

En raison de la différence des vitesses d'exécution des deux domaines, il se peut que D_2 exécute l'opération de retour des paramètres, alors que la capacité pour l'appelant n'est pas encore présente dans sa C-liste (l'opération ATTENDRE n'est pas effectuée).Cela correspond à un cas de blocage de D_2 qui doit être detecté par le noyau. L'arrivée de la capacité attendue doit débloquer le domaine.

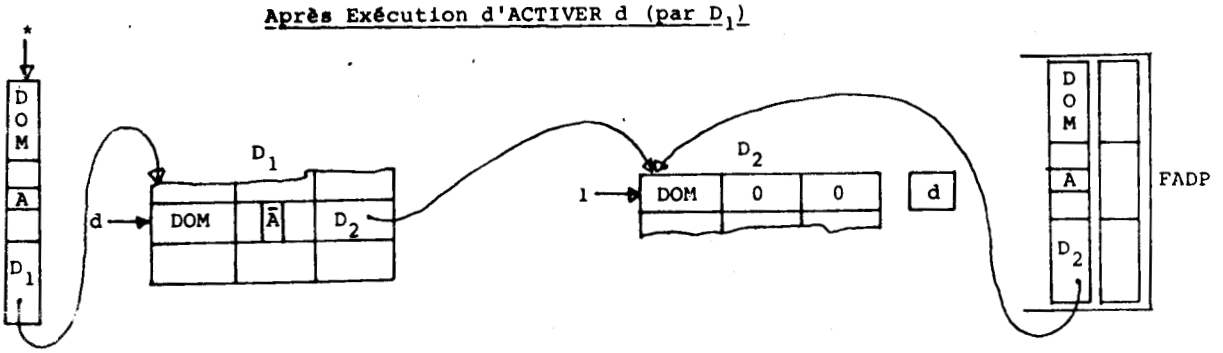
6.5.4 Synchronisation directe de l'exécution de domaines frères

Les programmes de la page 135 présentent un cas de synchronisation directe de deux domaines frères D_1 et D_2. Ces deux domaines ont été activés par un domaine père D (a et b). Un dialogue s'établit entre D_1 et D_2 - qui sont des coroutines - et D attend que ce dialogue soit terminé (c). Ce schéma suppose que D_2 possède une capacité pour D_1, et D_1 une capacité pour D_2.

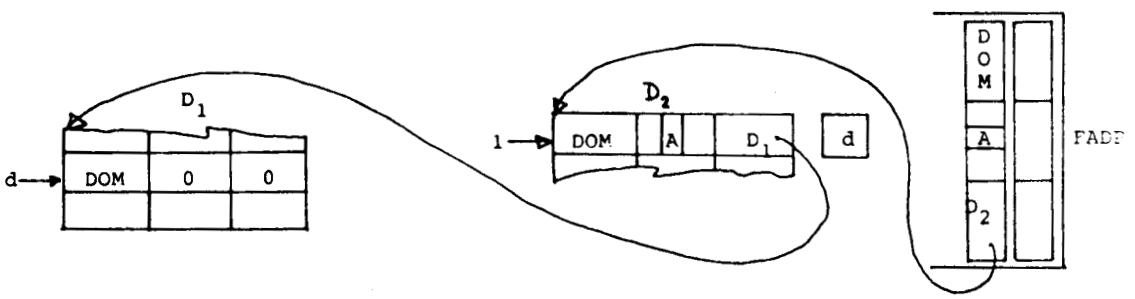
Avant exécution d'ACTIVER d



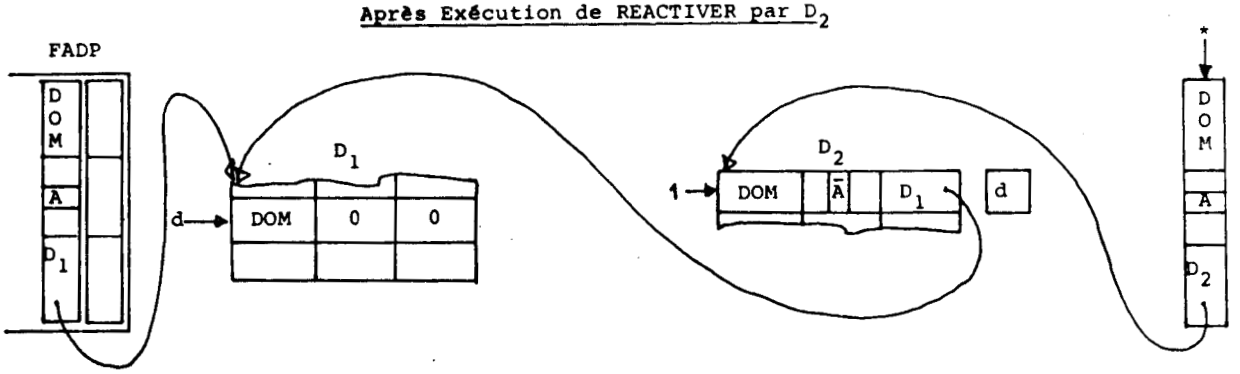
Après Exécution d'ACTIVER d (par D_1)



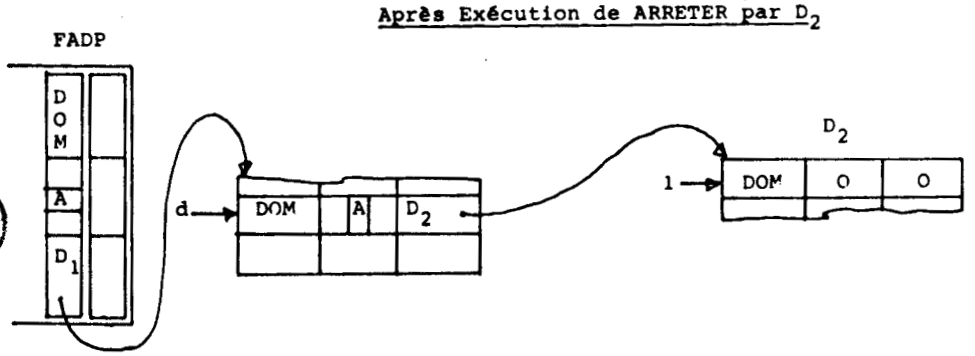
Après exécution de ATTENDRE d (par D_1)



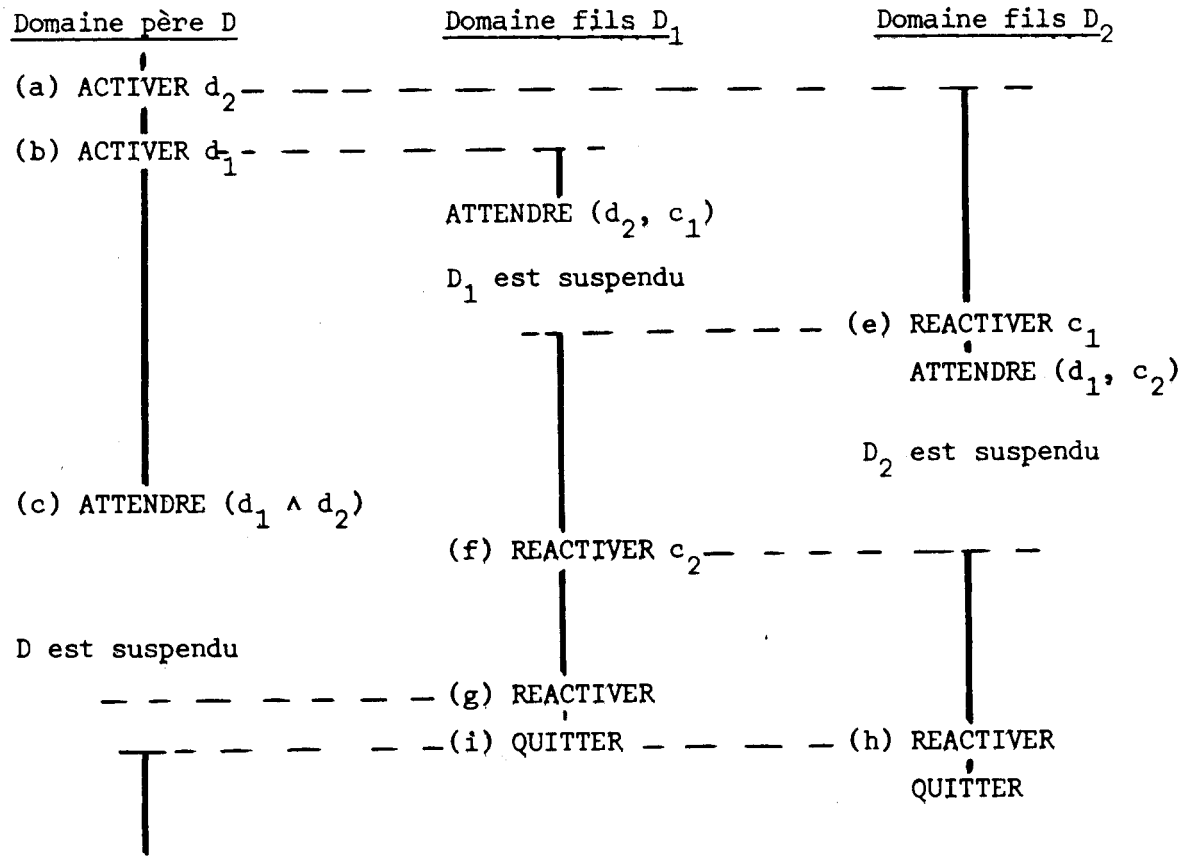
Après Exécution de REACTIVER par D_2



Après Exécution de ARRETER par D_2



BUS LILLE



Notations : d_1 : nom local d'une capacité D_1 dans D et dans D_1
d_2 : nom local d'une capacité pour D_2 dans D et dans D_1
c_1 : nom local dans D_2
c_2 : nom local dans D_1

Il faut remarquer :

- l'emploi de l'opération REACTIVER pour les domaines frères (e et f) ou pour le domaine père (g et h)
- que la deuxième exécution de REACTIVER (h) déclenche l'exécution de D : l'attente porte sur la terminaison de D_1 et de D_2 (c).

Une mauvaise programmation peut conduire à des étreintes fatales : la disposition de l'instruction

ATTENDRE (d_1, c_2)

après la deuxième activation de D_2 suspend son exécution ; mais il ne sera jamais réactivé par D_1 et, donc, D restera indéfiniment en attente. Cependant ce cas d'étreinte fatale est détectable : dans l'exemple, la capacité placée en c_2 de D_1 peut être signalée comme un lien vers D_2, résultant d'une opération

ATTENDRE effectuée par D_2. Aussi, la présence de cette capacité au moment de l'exécution de QUITTER (i) par D_1 constitue une anomalie caractéristique d'une éventuelle étreinte fatale.

6.6 REMARQUES

6.6.1 *Etats d'un domaine*

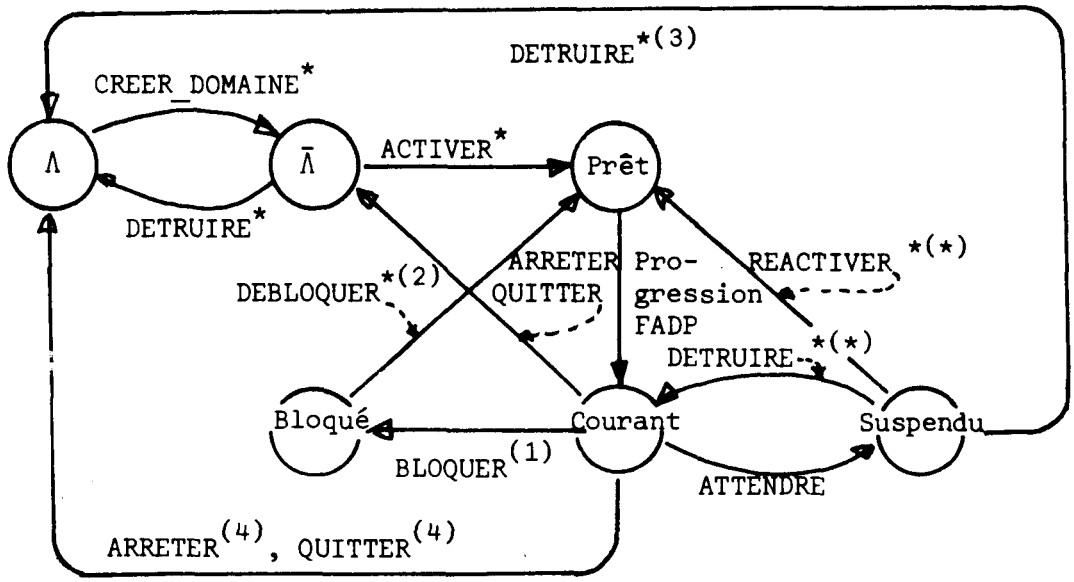
Un domaine peut se trouver dans les états :

- existant, après sa création ou en fin de son exécution
- prêt
- courant
- suspendu, après exécution d'une opération ATTENDRE
- bloqué, après une opération P sur un sémaphore.

Le graphe de la page suivante représente les transitions d'états d'un domaine sous l'action des opérations de transfert de contrôle et de synchronisation.

6.6.2 *Appels imbriqués de domaines*

Les appels imbriqués de domaines contenant des objets partagés s'apparentent aux appels imbriqués de moniteurs. De tels appels posent le problème bien connu (Lister [41], Haddan [19], Parnas [55], Wettstein [66], Joseph et Prasad [29]) du relachement de l'exclusion mutuelle sur les objets partagés. Du fait de la particularité de la structure matérielle d'OMPHALE tous les domaines sont exécutés en exclusion mutuelle. Il importe, alors, de relacher l'exclusion - i.e. de faire progresser les Files d'Attente des Odres de Transfert des Objets partagés référencés dans un domaine - dès que l'exécution du domaine est stoppée, quelle qu'en soit la cause. Sinon, il y a risque d'étreinte fatale. Mais cela impose que l'invariant du domaine soit vérifié dans tous ces cas.



Notes : Λ : état inexistant

* : opération effectuée par un autre domaine que D (si **, un domaine appelé par D)

(1) : résultat d'une opération P sur un sémaphore

(2) : résultat d'une opération V sur un sémaphore

(3) : le lieu appelant/appelé est l'unique capacité repérant D

(4) : D n'est référencé qu'à partir de la FADP

Graphe d'état d'un domaine D

BUS LILLE

CONCLUSION

Au terme de ce travail les divers mécanismes formant le noyau d'un système réparti à structure de domaines ont été examinés. Leur présence a été justifiée, leur efficacité a été discutée et leur influence sur la qualité du logiciel a été évaluée. Cependant deux hypothèses subsistent ; l'une doit être dépassée, l'autre doit être validée :

- hypothèse de force des types. Le processus de détection des objets perdus n'a été étudié que pour des objets fortement typés. Il parait facilement généralisable au cas où le type d'un composant appartient à une union de types déterminés. Par contre, le passage à des composants de type quelconque semble non trivial. Or le système OMPHALE devra supporter des objets dont les composants entrent dans cette catégorie (exemple : catalogues, générateurs de types). La définition d'un langage de haut niveau adapté à la structure de domaine, actuellement en cours (Michael [46]), devrait permettre de préciser les règles de structuration des types et de résoudre, par la suite, le problème des objets perdus dans le cas général.

- hypothèse de faisabilité de la régulation de la charge. Le système OMPHALE manipulant des domaines de petite taille, les résultats des simulations de Casey ne peuvent pas être utilisés. En conséquence, la faisabilité de la régulation de la charge reste à prouver. Cependant, il semblerait que la fréquence des interactions entre les sites puisse être réduite à volonté, au prix, certes, d'un déséquilibre des charges des sites, donc d'une perte de performances (significative ?). Plus précisément, doivent être examinées :

  * la correction des algorithmes de choix de site.
  * l'influence de la charge sur la bande passante requise pour le SSC.
  * l'influence sur les performances
    d'une modification de la configuration
    d'une spécialisation partielle des sites
    de différentes mesures comme la copie des objets-type, la copie des segments de code, ...

Pour ce faire, la réalisation de la maquette d'un site est en cours d'étude.

Une version réduite de noyau sera implantée sur cette maquette, Différents paramètres indispensables : taille moyenne des domaines, nombre moyen de composants d'un objet, durée et fréquence d'appel de certaines opérations nucléaires, etc ..., pourront, alors, être mesurés. L'élaboration d'un modèle mathématique du comportement du système, et/ou la rédaction d'un programme de simulation, voire une expérimentation en vraie grandeur sur maquette, devraient permettre de conclure.

Enfin, la conception du Sous-Système de Communication fait l'objet d'une étude particulière (Messi N'Jangue [45]).

ANNEXE 1

PREUVE DE L'IDENTITÉ DE L'ENSEMBLE DES OBJETS PERDUS A L'ENSEMBLE DES OBJETS RECURSIFS NON MARQUÉS

Let E be the set of existing objects, E_o those which are directly accessible, and E_1 those which are recursive. Let us denote as x R y the fact that object x has a reference to object y, and R_p the restriction of R to prerecursive objects. Let R^* (resp. R_p^*) be the transitive closure of R (resp. R_p). Obviously lost objects are the set $E - R^*(E_o)$ where $R^*(E_o) = \{y \mid \exists\, x \in E_o \;\; x\, R^*\, y\}$

Property 1 $E = R^*(E_o \cup E_1)$

Let $x_o, \ldots, x_p$ be distinct members of $E - (E_o \cup E_1)$ such that for all $i = 1, \ldots, p$, $x_i\, R\, x_{i-1}$. Since $x_p \notin E_o$, it must be referenced by some member of E, say x_{p+1} and $x_{p+1}\, R^*\, x_i$ for all $i = 0, \ldots, p$. Because $x_i \notin E_1$, x_i is not recursive, and $x_{p+1} \neq x_i$ for all $i = 0, \ldots, p$ Because E is finite, there must exist some k for which $x_k\, R^*\, x_o$ and $x_k \in E_o \cup E_1$, which concludes the proof.

Property 2 $E_1 \cap R^*(E_o) = E_1 \cap R_p^*(E_o)$ i.e. the marking process concerns non lost recursive objects.

Obviously $R_p^*(E_o) \subset R^*(E_o)$. Conversely if x R y and y is of prerecursive, then x is also prerecursive and $x\, R_p\, y$. Therefore $x\, R^*\, y$ and $y \in E_1$ implies $x\, R_p^*\, y$ which concludes the proof.

We shall formalize now the destruction process as follows :

Let $x\, R_1\, y \iff x\, R\, y$ and $x \notin E_1 - R^*(E_o)$

Hence R_1 is the relation obtained after destroying capabilities in the recursive non marked objects. Let $F_1 = E$ for purposes of notations. Given (F_i, R_i), let (F_{i+1}, R_{i+1}) defined as follows :

a) there exist $x_i \in F_i - E_o$ such that $\forall\, y \in F_i$, y R x does not hold. i.e. the reference count of x is nul. Then $F_{i+1} = F_i - \{x\}$

$y\, R_{i+1}\, z \iff y\, R_i\, z$ and $y \neq x$

b) otherwise $F_{i+1} = F_i$ and $R_{i+1} = R_i$

Because $F_{i+1} \subset F_i \subset E$ and $F_{i+1} = F_i$ implies $F_k = F_i$ for all $k > i$ and finally that E is finite, the process must end and we denote $(\bar{F}, \bar{R})$ the limit.

Property 3 $R^*(E_o) = \bar{F}$ and $\bar{R}$ is the restriction of R to $\bar{F}$. Let $x \in R^*(E_o)$ i.e. there exist $x_o, \ldots, x_n$ such that $x_o \in E_o$, $x_n = x$ and for all $j = 1, \ldots, n$ $x_{j-1} R x_j$. It follows that $x_o \in F_i$ for all i. From $x_j \in R^*(E_o)$ it follows $x_{j-1} R_1 x_j$ for all j. Moreover if $\{x_j\} \subset F_i$ and $x_{j-1} R_i x_j$ for all j, then $x_j \neq x_i$ for all j and $\{x_j\} \subset F_{i+1}$ and $x_{j-1} R_{i+1} x_j$. Hence, this is true for all i and $x \in \bar{F}$. Furthermore if x, $y \in R^*(E_o)$ such that x R y then x $\bar{R}$ y. Since $\bar{F}$ is such that for all members of $\bar{F} - E_o$, all reference counts are non null, property 1 apply to $\bar{F}$, i.e. $\bar{F} = \bar{R}^*(E_o \cup (E_1 \cap \bar{F}))$. Let $x \in \bar{F}$ and suppose $x \in \bar{R}^*(E_1 \cap \bar{F})$ i.e. $\exists\, y \in E_1 \cap \bar{F}$ such that $y \bar{R}^* x$. But $y \in E_1 \cap R^*(E_o)$ othewise $y \bar{R}_1^* x$ cannot hold and therefore $y \bar{R} x$. From the fact that $\bar{R}$ implies R, we have $y R^* x$ and $y \in R^*(E_o)$. Hence $x \in R^*(E_o)$. QED.

This property shows that $(\bar{F}, E_o, \bar{R})$ is a data structure equivalent to (E, E_o, R), and from $\bar{F} = \bar{R}^*(E_o)$ that it does not contain any lost object.

ANNEXE 3

AmZ8001

16-Bit Microprocessor
Advanced Micro Devices

PRELIMINARY DATA

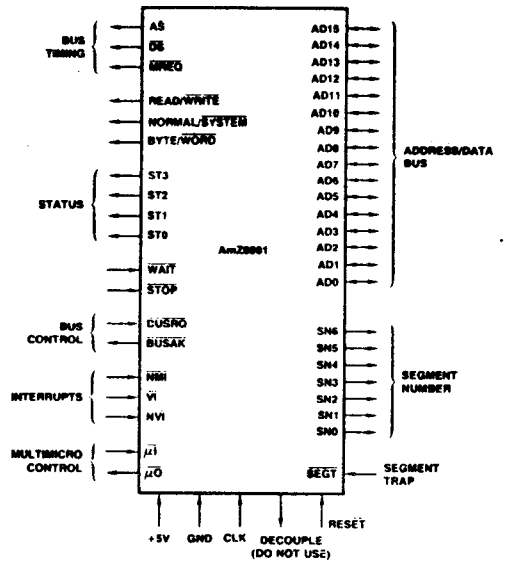
DISTINCTIVE CHARACTERISTICS

- Sixteen general purpose registers
- Direct addressing up to 8MB segmented memory
- Software compatible with AmZ8002 microprocessor
- Powerful instructions with flexible addressing modes
- Privileged/Non-Privileged mode of operation
- Sophisticated interrupt structure
- On-chip memory refresh facility
- TTL compatible inputs and outputs
- Single phase clock
- Single +5V power supply
- 48-pin package

GENERAL DESCRIPTION

The AmZ8001 is a general-purpose 16-bit CPU belonging to the AmZ8000 family of microprocessors. Its architecture is centered around sixteen 16-bit general registers. The CPU deals with 23-bit address spaces and hence can address directly 8MB of memory. The 23-bit address consists of two components: 7-bit segment number and 16-bit offset. Facilities are provided to maintain three distinct address spaces – code, data and stack. The AmZ8001 implements a powerful instruction set with flexible addressing modes. These instructions operate on several data types – bit, byte, word (16-bit), long word (32-bit), byte string and word string. The CPU can execute instructions in one of two modes – System and Normal. Sometimes these modes are also known as Privileged and Non-Privileged, respectively. The CPU also contains an on-chip memory refresh facility. The AmZ8001 is software compatible with the AmZ8002 microprocessor. The AmZ8001 is fabricated using silicon-gate N-MOS technology and is packaged in a 48-pin DIP. The AmZ8001 requires a single +5 power supply and a single phase clock for its operation.

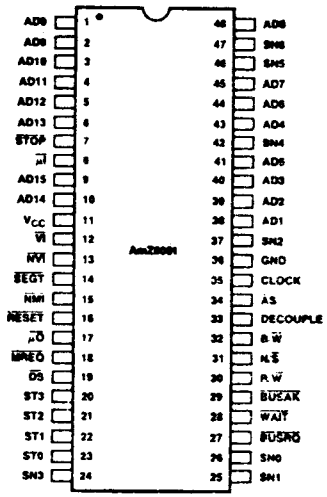
LOGIC SYMBOL



MOS-240

CONNECTION DIAGRAM

Top View



Note: Pin 1 is marked for orientation.

MOS-241

INTERFACE SIGNAL DESCRIPTION

V_{CC}: +5V Power Supply

V_{SS}: Ground

AD0-AD15: Address/Data Bus (Bidirectional, 3-State)

This 16-bit multiplexed address/data bus is used for all I/O and memory transactions. HIGH on the bus corresponds to 1 and LOW corresponds to 0. AD0 is the least significant bit position with AD15 is most significant. The $\overline{AS}$ output and $\overline{DS}$ output will indicate whether the bus is used for address offset or data. The status output lines ST0-ST3 will indicate the type of transaction; memory or I/O.

$\overline{AS}$: Address Strobe (Output, 3-State)

LOW on this output indicates that the AD0-AD15 bus contains address information. The address information is stable by the time of the LOW-to-HIGH transition of the $\overline{AS}$ output (see timing diagrams). The status outputs ST0-ST3 indicate whether the bus contains a memory address or I/O address.

$\overline{DS}$: Data Strobe (Output, 3-State)

LOW on this output indicates that the AD0-AD15 bus is being used for data transfer. The R/$\overline{W}$ output indicates the direction of data transfer – read (or in) means data into the CPU and write (or out) means data from the CPU. During a read operation, data can be gated on to the bus when $\overline{DS}$ goes LOW. A LOW-to-HIGH transition on the $\overline{DS}$ output indicates that the CPU has accepted the data (see timing diagram). During a write operation, LOW on the $\overline{DS}$ output indicates that data is setup on the bus. Data will be removed sometime after the LOW-to-HIGH transition of the $\overline{DS}$ output (see timing diagram).

R/$\overline{W}$: Read/Write (Output, 3-State)

This output indicates the direction of data flow on the AD0-AD15 bus. HIGH indicates a read operation, i.e., data into the CPU and LOW indicates a write operation, i.e., data from the CPU. This output is activated at the same time as $\overline{AS}$ going LOW and remains stable for the duration of the whole transaction (see timing diagram).

B/$\overline{W}$: Byte/Word (Output, 3-State)

This output indicates the type of data transferred on the AD0-AD15 bus. HIGH indicates byte (8-bit) and LOW indicates word (16-bit) transfer. This output is activated at the same stage as $\overline{AS}$ going LOW and remains valid for the duration of the whole transaction (see timing diagram). The address generated by the CPU is always a byte address. However, the memory is organized as 16-bit words. All instructions and word operands are word aligned and are addressed by even addresses. Thus, for all word transactions with the memory the least significant address bit will be zero. When addressing the memory for byte transactions, the least significant address bit determines which byte of the memory word is needed; even address specifies the most significant byte and odd address specifies the least significant byte. In the case of I/O transactions, the address information on the AD0-AD15 bus refers to an I/O port and B/$\overline{W}$ determines whether a data word or data byte will be transacted. During I/O byte transactions, the least significant address bit A0 determines which half of the AD0-AD15 bus will be used for the I/O transactions. The ST0-ST3 outputs will indicate whether the current transaction is for memory, normal I/O or special I/O.

ST0-ST3: Status (Outputs, 3-State)

These four outputs contain information regarding the current transaction in a coded form. The status line codes are shown in the following table:

ST3	ST2	ST1	ST0	
L	L	L	L	Internal Operation
L	L	L	H	Memory Refresh
L	L	H	L	Normal I/O Transaction
L	L	H	H	Special I/O Transaction
L	H	L	L	Segment Trap Acknowledge
L	H	L	H	Non-Maskable Interrupt Acknowledge
L	H	H	L	Non-Vectored Interrupt Acknowledge
L	H	H	H	Vectored Interrupt Acknowledge
H	L	L	L	Memory Transaction for Operand
H	L	L	H	Memory Transaction for Stack
H	L	H	L	Reserved
H	L	H	H	Reserved
H	H	L	L	Memory Transaction for Instruction Fetch (Subsequent Word)
H	H	L	H	Memory Transaction for Instruction Fetch (First Word)
H	H	H	L	Reserved
H	H	H	H	Reserved

$\overline{WAIT}$: Wait (Input)

LOW on this input indicates to the CPU that memory or I/O is not ready for the data transfer and hence the current transaction should be stretched. The $\overline{WAIT}$ input is sampled by the CPU at certain instances during the transaction (see timing diagram). If $\overline{WAIT}$ input is LOW at these instances, the CPU will go into wait state to prolong the transaction. The wait state will repeat until the $\overline{WAIT}$ input is HIGH at the sampling instant.

N/$\overline{S}$: Normal/System Mode (Output, 3-State)

HIGH on this output indicates that the CPU is operating in Normal Mode and LOW indicates operation in System Mode. This output is derived from the Flag Control Word (FCW) register. The FCW register is described under the program status information section of this document.

$\overline{MREQ}$: Memory Request (Output, 3-State)

LOW on this output indicates that a CPU transaction with memory is taking place.

$\overline{BUSRQ}$: Bus Request (Input)

LOW on this input indicates to the CPU that another device (such as DMA) is requesting to take control of the bus. The $\overline{BUSRQ}$ input can be driven LOW anytime. The CPU synchronizes this input internally. The CPU responds by activating $\overline{BUSAK}$ output LOW to indicate that the bus has been relinquished. Relinquishing the bus means that the AD0-AD15, $\overline{AS}$, $\overline{DS}$, B/$\overline{W}$, R/$\overline{W}$, N/$\overline{S}$, ST0-ST3, SN0-SN6 and $\overline{MREQ}$ outputs will be in the high impedance state. The requesting device should control these lines in an identical fashion to the CPU to accomplish transactions. The $\overline{BUSRQ}$ input must remain LOW as long as needed to perform all the transactions and the CPU will keep the $\overline{BUSAK}$ output LOW. After completing the transactions, the device must disable the AD0-AD15, $\overline{AS}$, $\overline{DS}$, B/$\overline{W}$, R/$\overline{W}$, N/$\overline{S}$, ST0-ST3, SN0-SN6 and $\overline{MREQ}$ into the high impedance state and stop driving the $\overline{BUSRQ}$ input LOW. The CPU will make $\overline{BUSAK}$ output HIGH sometime later and take back the bus control.

$\overline{BUSAK}$: Bus Acknowledge (Output)

LOW on this output indicates that the CPU has relinquished the bus in response to a bus request.

BUS LILLE

$\overline{NMI}$: Non-Maskable Interrupt (Input)

HIGH to LOW transition on this input constitutes non-maskable interrupt request. The CPU will respond with the Non-maskable Interrupt Acknowledge on the ST0-ST3 outputs and will enter an interrupt sequence. The transition on the $\overline{NMI}$ can occur anytime. Of the three kinds of interrupts available, the non-maskable interrupt has the highest priority.

$\overline{VI}$: Vectored Interrupt (Input)

LOW on this input constitutes vectored interrupt request. Vectored interrupt is next lower to the non-maskable interrupt in priority. The VIL bit in the Flag and Control Word register must be 1 for the vectored interrupt to be honored. The CPU will respond with Vectored Interrupt Acknowledge code on the ST0-ST3 outputs and will begin the interrupt sequence. The $\overline{VI}$ input can be driven LOW anytime and should be held LOW until acknowledged.

$\overline{NVI}$: Non-Vectored Interrupt (Input)

LOW on this input constitutes non-vectored interrupt request. Non-vectored has the lowest priority of the three types of interrupts. The NVIE bit in the Flag and Control Word register must be 1 for this request to be honored. The CPU will respond with Non-Vectored Interrupt Acknowledge code on the ST0-ST3 outputs and will begin the interrupt sequence. The NVI input can be driven LOW anytime and should be held LOW until acknowledged.

$\overline{\mu I}$: Micro-In (Input)

This input participates in the resource request daisy chain. See the section on multimicroprocessor support facilities in this document.

$\overline{\mu O}$: Micro-Out (Output)

This output participates in the resource request daisy chain. See the section on multimicroprocessor support facilities in this document.

$\overline{RESET}$: Reset (Input)

LOW on this input initiates a reset sequence in the CPU. See the section on Initialization for details on reset sequence.

CLK: Clock (Input)

All CPU operations are controlled from the signal fed into this input. See DC characteristics for clock voltage level requirements.

DECOUPLE: (Output)

Output from the on-chip substrate bias generator. Do not use.

$\overline{STOP}$: Stop (Input)

This active LOW input facilitates one instruction at a time operation. See the section on single stepping.

SN0-SN6: Segment Number (Outputs, 3-State)

These seven outputs contain the segment number part of a memory address. A HIGH on the output corresponds to 1 and a LOW corresponds to 0. SN0 is the least significant bit position and SN6 is the most significant bit position.

$\overline{SEGT}$: Segment Trap (Input)

LOW on this input constitutes a segment trap request. If the line is driven LOW, the CPU will respond with the Segment Trap Acknowledge code on the Status lines, and commence a trap sequence. The $\overline{SEGT}$ input may be driven LOW at any time and is customarily held LOW until acknowledged. This input has priority over the interrupts.

PROCESSOR ORGANIZATION

The following is a brief discussion of the AmZ8001 CPU. For detailed information, see the AmZ8001/AmZ8002 Processor Instruction Set Manual (Publication No. AM-PUB086).

GENERAL PURPOSE REGISTERS

The CPU is organized around sixteen 16-bit general purpose registers R0 through R15 as shown in Figure 1. For byte operations, the first eight registers (R0 through R7) can also be addressed as sixteen 8-bit registers designated as RL0, RH0 and so on to RL7 and RH7. The sixteen registers can also be grouped in pairs RR0, RR2 and so on to RR14 to form eight long word (32-bit) registers. Similarly, the sixteen registers can be grouped in quadruples RQ0, RQ4, RQ8 and RQ12 to form four 64-bit registers.

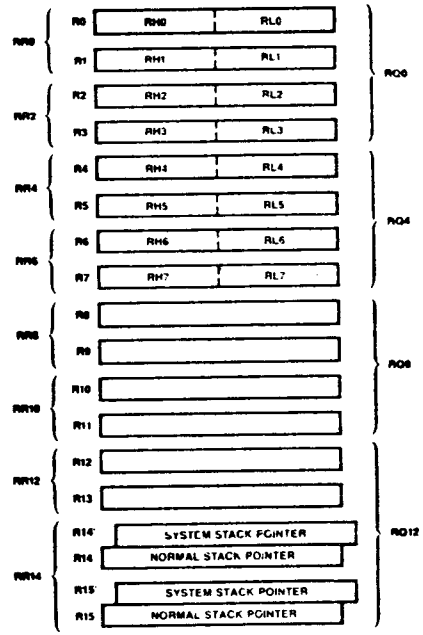


Figure 1. AmZ8001 Registers.

MOS-24

STACK POINTER

The AmZ8001 architecture allows stacks to be maintained in memory. Any general-purpose register pair except RR0 can be used as a stack pointer in stack manipulating instructions such as PUSH and POP. The designated register pair holds a 23-bit segmented address. Certain instructions (such as subroutine call and return) make implicit use of the register pair RR14 as the stack pointer. Two implicit stacks are allowed – normal stack using RR14 as the stack pointer, and system stack using RR14' as the system stack pointer (see Figure 1). If the CPU is operating in the Normal Mode, RR14 is active, and if the CPU is in System Mode, RR14' will be used instead of RR14. The implied stack pointer is a part of the general registers and hence can be manipulated using the instructions available for register operations.

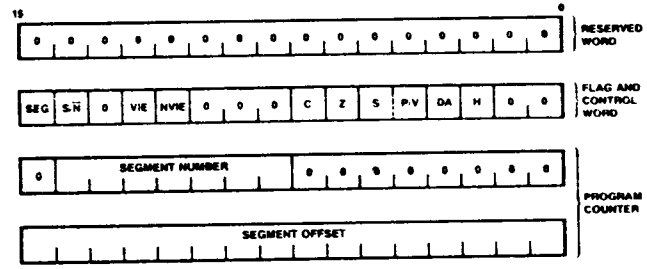


Figure 2. AmZ8001 Processor Status.

MOS-243

PROCESSOR STATUS

The CPU status consists of the 16-bit flag and control word (FCW) register, and the 23-bit program counter (see Figure 2). A reserved word is also included for future expansion. The following is a brief description of the FCW bits.

SEG: Segmented Non-Segmented Bit. Indicates whether the AmZ8001 is running in segmented or non-segmented mode. 1 indicates segmented, 0 indicates non-segmented. See the section on non-segmented mode, elsewhere in this document.

S/$\overline{N}$: System Normal – 1 indicates System Mode and 0 indicates Normal Mode.

VIE: Vectored Interrupt Enable – 1 indicates that Vectored Interrupt requests will be honored.

NVIE: Non-Vectored Interrupt Enable – 1 indicates that Non-vectored interrupt requests will be honored.

C: Carry – 1 indicates that a carry has occurred from the most significant bit position when performing arithmetic operations.

Z: Zero – 1 indicates that the result of an operation is zero.

S: Sign – 1 indicates that the result of an operation is negative (i.e., most significant bit is one.

P/V: Parity/Overflow – 1 indicates that there was an overflow during arithmetic operations. For byte logical operations this bit indicates parity of the result.

DA: Decimal Adjust – Records byte arithmetic operations.

H: Half Carry – 1 indicates that there was a carry from the most significant bit of the lower digit during byte arithmetic.

DATA TYPES

The AmZ8001 instructions operate on bits, digits (4 bits), bytes (8 bits), words (16 bits), long words (32 bits), byte strings and word strings type operands. Bits can be set, reset or tested. Digits are used to facilitate BCD arithmetic operations. Bytes are used for characters and small integers. Words are used for integer values and addresses while long words are used for large integer values and addresses. All operands except strings can reside either in memory or general registers. Strings can reside in memory only.

INTERRUPT AND TRAP STRUCTURE

Interrupt is defined as an external asynchronous event requiring program interruption. For example, interruption is caused by a peripheral needing service. Traps are synchronous events resulting from execution of certain instructions under some defined circumstances. Both interrupts and traps are handled in a similar manner by the AmZ8001.

The AmZ8001 supports three types of interrupts in order of descending priority – non-maskable, vectored and non-vectored. The vectored and non-vectored interrupts can be disabled by appropriate control bits in the FCW. The AmZ8001 has four traps – system call, segment trap, unimplemented opcode and privileged instruction. The traps have higher priority than interrupts.

When an interrupt or trap occurs, the current program status is automatically pushed on to the system stack. The program status consists of processor status (i.e., PC and FCW) plus a 16-bit identifier. The identifier contains the reason, source and other coded information relating to the interrupt or trap.

After saving the current program status, the new processor status is automatically loaded from the new program status area located in the memory. This area is designated by the New Program Status Area Pointer (NPSAP) register. See AM-PUB086 publication for further details.

SEGMENTED ADDRESSING

The AmZ8001 can directly address up to 8MB of memory space, using a 23-bit segmented address. The memory space is divided up into 128 segments, each up to 64KB in size. The upper seven bits of address designate the segment number, and are available on the SN0-SN6 outputs during a memory transaction. See the section on memory transactions for details.

The lower sixteen bits of address designate an offset within the segment, relative to the start of the segment, and are available on AD0-AD15 during part of the memory transaction. See the section on memory transactions for details.

The segmented address may be stored as a long word in memory, or in a register pair. The segment number and offset can be manipulated separately or together, by suitable use of the instruction set.

When the segmented address is contained in code space, a short offset format may be adopted. The segmented address is stored as one word, seven bits of segment number and eight bits of offset. Figure 3 shows the format for segmented addresses.

ADDRESSING MODES

Information contained in the AmZ8001 instructions consists of the operation to be performed, the operand type and the location of the operands. Operand locations are designated by general register addresses, memory addresses or I/O addresses. The addressing mode of a given instruction defines the address space referenced and the method to compute the operand address. Addressing modes are explicitly specified or implied in an instruction. Figure 4 illustrates the eight explicit addressing modes: Register (R), Immediate (IM), Indirect Register (IR), Direct Address (DA), Indexed (X), Relative Address (RA), Base Address (BA) and Base Indexed (BX).

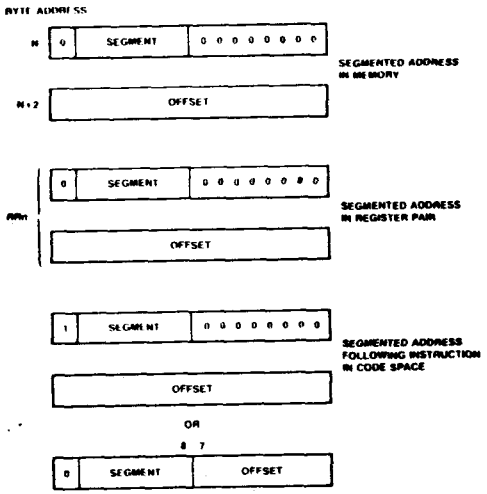


Figure 3. Segmented Address Formats.

MOS-244

When an effective segmented address is being computed according to the designated addressing mode, the segment number is not affected by any carry from the 16-bit offset.

NON-SEGMENTED MODE ON THE AmZ8001

The AmZ8001 can execute code designed to run on the non-segmented AmZ8002. This is achieved by changing the mode of execution of the AmZ8001 from segmented to non-segmented by writing a 0 to the SEG bit in the FCW. (See the section on processor status.) The change to non-segmented mode sets up a suitable environment for running non-segmented code. However, this environment only exists within the code segment that caused the change of mode from segmented to non-segmented.

SN0-SN6 will continue to indicate the code segment until a reset, interruption or return to segmented mode is encountered.

The effects of the non-segmented mode of operation on the AmZ8001 are described below.

a) The AmZ8001 will interpret instruction length as if it was a non-segmented AmZ8002.
b) The AmZ8001 will implement address computation in an identical manner to the AmZ8002.

Other CPU functions, such as interrupt and trap handling, reset and stack pointer manipulation are unaltered. These functions are characterized by the type of CPU, not by the state of the SEG bit in the FCW.

INPUT/OUTPUT

A set of I/O instructions are provided to accomplish byte or word transfers between the AmZ8001 and I/O devices. I/O devices are addressed using 16-bit I/O port addresses and I/O address space is not a part of the memory address space. Two types of I/O instructions are provided; each with its own 16-bit address space. I/O instructions include a comprehensive set of In, Out and Block transfers.

Mode	Operand Addressing			Operand Value
	In the Instruction	In a Register	In Memory	
Register		REGISTER ADDRESS → OPERAND		The content of the register
Immediate	OPERAND			In the instruction
Indirect *Register*		REGISTER ADDRESS → ADDRESS	→ OPERAND	The content of the location whose address is in the register.
Direct *Address*	ADDRESS		→ OPERAND	The content of the location whose address is in the instruction.
Index		REGISTER ADDRESS + DISPLACEMENT; BASE ADDRESS	→ OPERAND	The content of the location whose address is the address in the instruction, offset by the content of the working register.
Relative Address		PC VALUE; DISPLACEMENT	→ OPERAND	The content of the location whose address is the content of the program counter, offset by the displacement in the instruction.
Base Address		REGISTER ADDRESS → BASE ADDRESS; DISPLACEMENT	→ OPERAND	The content of the location whose address is the address in the register, offset by the displacement in the instruction.
Base Index		REGISTER ADDRESS → BASE ADDRESS; REGISTER ADDRESS → DISPLACEMENT	→ OPERAND	The content of the location whose address is the address in the register, offset by the displacement in the register.

Figure 4. Addressing Modes.

MOS-245

BIUS LILLE

AmZ8001 INSTRUCTION SET

LOAD AND EXCHANGE

Mnemonics	Operands	Addr. Modes	Operation
CLR CLRB	dst	R IR DA X	Clear dst ← 0
EX EXB	R, src	R IR DA X	Exchange R ↔ src
LD LDB LDL	R, src	R IM IM IR DA X BA BX	Load into Register R ← src
LD LDB LDL	dst, R	IR DA X BA BX	Load into Memory (Store) dst ← R
LD LDB	dst, IM	IR DA X	Load Immediate into Memory dst ← IM
LDA	R, src	DA X BA BX	Load Address R ← source address
LDAR	R, src	RA	Load Address Relative R ← source address
LDK	R, src	IM	Load Constant R ← n (n = 0 . . . 15)
LDM	R, src, n	IR DA X	Load Multiple R ← src (n consecutive words) (n = 1 . . . 16)
LDM	dst, R, n	IR DA X	Load Multiple (Store Multiple) dst ← R (n consecutive words) (n = 1 . . . 16)
LDR LDRB LDRL	R, src	RA	Load Relative R ← src (range −32768 . . . +32767)
LDR LDRB LDRL	dst, R	RA	Load Relative (Store Relative) dst ← R (range −32768 . . . +32767)
POP POPL	dst, R	R IR DA X	Pop dst ← IR Autoincrement contents of R
PUSH PUSHL	IR, src	R IM IR DA X	Push Autodecrement contents of R IR ← src

ARITHMETIC

Mnemonics	Operands	Addr. Modes	Operation
ADC ADCB	R, src	R	Add with Carry R ← R + src + carry
ADD ADDB ADDL	R, src	R IM IR DA X	Add R ← R + src
CP CPB CPL	R, src	R IM IR DA X	Compare with Register R − src
CP CPB	dst, IM	IR DA X	Compare with Immediate dst − IM
DAB	dst	R	Decimal Adjust
DEC DECB	dst, n	R IR DA X	Decrement by n dst ← dst − n (n = 1 . . . 16)
DIV DIVL	R, src	R IM IR DA X	Divide (signed) Word: $R_{n+1} \leftarrow R_{n,n+1} \div src$ $R_n \leftarrow$ remainder Long Word: $R_{n+2,n+3} \leftarrow R_{n\ldots n+3} \div src$ $R_{n,n+1} \leftarrow$ remainder
EXTS EXTSB EXTSL	dst	R	Extend Sign Extend sign of low order half of st through high order half of dst
INC INCB	dst, n	R IR DA X	Increment by n dst ← dst + n (n = 1 . . . 16)
MULT MULTL	R, src	R IM IR DA X	Multiply (signed) Word: $R_{n,n+1} \leftarrow R_{n+1} * src$ Long Word: $R_{n\ldots n+3} \leftarrow R_{n+2,n+3} * src$ *Plus seven cycles for each 1 in the multiplicand
NEG NEGB	dst	R IR DA X	Negate dst ← 0 − dst
SBC SBCB	R, src	R	Subtract with Carry R ← R − src − carry
SUB SUBB SUBL	R,src	R IM IR DA X	Subtract R ← R − src

LOGICAL

Mnemonics	Operands	Addr. Modes	Operation
AND ANDB	R, src	R IM IR DA X	AND R ← R AND src
COM COMB	dst	R IM IR DA X	Complement dst ← NOT dst
OR ORB	R, src	R IM IR DA X	OR R ← R OR src
TEST TESTB TESTL	dst	R IR DA X	TEST dst OR 0
TCC TCCB	cc, dst	R	Test Condition Code Set LSB if cc is true
XOR XORB	R, src	R IM IR DA X	Exclusive OR R ← R XOR src

PROGRAM CONTROL

Mnemonics	Operands	Addr. Modes	Operation
CALL	dst	IR DA X	Call Subroutine Autodecrement SP @ SP ← PC PC ← dst
CALR	dst	RA	Call Relative Autodecrement SP @ SP ← PC PC ← PC + dst (range −4094 to +4096)
DJNZ DBJNZ	R, dst	RA	Decrement and Jump if Non-Zero R ← R − 1 IF R = 0: PC ← PC + dst (range −254 to 0)
IRET*			Interrupt Return PS ← @ SP Autoincrement SP
JP	cc, dst	IR IR DA X	Jump Conditional If cc is true: PC ← dst
JR	cc, dst	RA	Jump Conditional Relative If cc is true: PC ← PC + dst (range −256 to +254)
RET	cc		Return Conditional If cc is true: PC ← @ SP Autodecrement SP
SC	src	IM	System Call Autodecrement SP @ SP ← old PS Push instruction PS ← System Call PS

*Privileged instructions. Executed in system mode only.

BIT MANIPULATION

Mnemonics	Operand	Addr. Modes	Operation
BIT BITB	dst, b	R IR DA X	Test Bit Static Z flag ← NOT dst bit specified by b
BIT BITB	dst, R	R	Test Bit Dynamic Z flag ← NOT dst bit specified by contents of R
RES RESB	dst, b	R IR DA X	Reset Bit Static Reset dst bit specified by b
RES RESB	dst, R	R	Reset Bit Dynamic Reset dst bit specified by contents of R
SET SETB	dst, b	R IR DA X	Set Bit Static Set dst bit specified by b
SET SETB	dst, R	R	Set Bit Dynamic Set dst bit specified by contents of R
TSET TSETB	dst	R IR DA X	Test and Set S flag ← MSB of dst dst ← all 1s

ROTATE AND SHIFT

Mnemonics	Operand	Addr. Modes	Operation
RLDB	R, src	R	Rotate Digit Left
RRDB	R, src	R	Rotate Digit Right
RL RLB	dst, n	R R	Rotate Left by n bits (n = 1, 2)
RLC RLCB	dst, n	R R	Rotate Left through Carry by n bits (n = 1, 2)
RR RRB	dst, n	R R	Rotate Right by n bits (n = 1, 2)
RRC RRCB	dst, n	R R	Rotate Right through Carry by n bits (n = 1, 2)
SDA SDAB SDAL	dst, R	R	Shift Dynamic Arithmetic Shift dst left or right by contents of R
SDL SDLB SDLL	dst, R	R	Shift Dynamic Logical Shift dst left or right by contents of R
SLA SLAB SLAL	dst, n	R	Shift Left Arithmetic by n bits
SLL SLLB SLLL	dst, n	R	Shift Left Logical by n bits
SRA SRAB SRAL	dst, n	R	Shift Right Arithmetic by n bits
SRL SRLB SRLL	dst, n	R	Shift Right Logical by n bits

BLOCK TRANSFER AND STRING MANIPULATION

Mnemonics	Operands	Addr. Modes	Operation
CPD CPDB	R_X, src, R_Y, cc	IR	Compare and Decrement R_X − src Autodecrement src address $R_Y \leftarrow R_Y - 1$
CPDR CPDRB	R_X, src, R_Y, cc	IR	Compare, Decrement and Repeat R_X − src Autodecrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
CPI CPIB	R_X, src, R_Y, cc	IR	Compare and Increment R_X − src Autoincrement src address $R_Y \leftarrow R_Y - 1$
CPIR CPIRB	R_X, src, R_Y, cc	IR	Compare, Increment and Repeat R_X − src Autoincrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
CPSD CPSDB	dst, src, R, cc	IR	Compare String and Decrement dst − src Autodecrement dst and src addresses $R \leftarrow R - 1$
CPSDR CPSDRB	dst, src, R, cc	IR	Compare String, Decr. and Repeat dst − src Autodecrement dst and src addresses $R \leftarrow R - 1$ Repeat until cc is true or R = 0
CPSI CPSIB	dst, src, R, cc	IR	Compare String and Increment dst − src Autoincrement dst and src addresses $R \leftarrow R - 1$
CPSIR CPSIRB	dst, src, R, cc	IR	Compare String, Incr. and Repeat dst − src Autoincrement dst and src addresses $R \leftarrow R - 1$ Repeat until cc is true or R = 0
LDD LDDB	dst, src, R	IR	*Load and Decrement* dst ← src Autodecrement dst and src addresses $R \leftarrow R - 1$
LDDR LDDRB	dst, src, R	IR	Load, Decrement and Repeat dst ← src Autodecrement dst and src addresses $R \leftarrow R - 1$ Repeat until R = 0

BLOCK TRANSFER AND STRING MANIPULATION (Cont.)

Mnemonics	Operands	Addr. Modes	Operation
LDI LDIB	dst, src, R	IR	Load and Increment dst ← src Autoincrement dst and src addresses R ← R + 1
LDIR LDIRB	dst, src, R	IR	Load, Increment and Repeat dst ← src Autoincrement dst and src addresses R ← R + 1 Repeat until R = 0
TRDB	dst, src, R	IR	Translate and Decrement dst ← src (dst) Autodecrement dst address R ← R − 1
TRDRB	dst, src, R	IR	Translate, Decrement and Repeat dst ← src (dst) Autodecrement dst address R ← R − 1 Repeat until R = 0
TRIB	dst, src, R	IR	Translate and Increment dst ← src (dst) Autoincrement dst address R ← R + 1
TRIRB	dst, src, R	IR	Translate, Increment and Repeat dst ← src (dst) Autoincrement dst address R ← R + 1 Repeat until R = 0
TRTDB	src 1, src 2, R	IR	Translate and Test, Decrement RH1 ← src 2 (src 1) Autodecrement src 1 address R ← R − 1
TRTDRB	src 1, src 2, R	IR	Translate and Test, Decrement and Repeat RH1 ← src 2 (src 1) Autodecrement src 1 address R ← R − 1 Repeat until R = 0 or RH1 = 0
TRTIB	src 1, src 2, R	IR	Translate and Test, Increment RH1 ← src 2 (src 1) Autoincrement src 1 address R ← R + 1
TRTIRB	src 1, src 2, R	IR	Translate and Test, Increment and Repeat RH1 ← src 2 (src 1) Autoincrement src 1 address R ← R + 1 Repeat until R = 0 or RH1 = 0

INPUT/OUTPUT

Mnemonics	Operands	Addr. Modes	Operation
IN* INB*	R, src	IR DA	Input R ← src
IND* INDB*	dst, src, R	IR	Input and Decrement dst ← src Autodecrement dst address R ← R − 1
INDR* INDRB*	dst, src, R	IR	Input, Decrement and Repeat dst ← src Autodecrement dst address R ← R − 1 Repeat until R = 0
INI* INIB*	dst, src, R	IR	Input and Increment dst ← src Autoincrement dst address R ← R + 1
INIR* INIRB*	dst, src, R	IR	Input, Increment and Repeat dst ← src Autoincrement dst address R ← R + 1 Repeat until R = 0
*OUT** OUTB*	dst, R	*IR* DA	*Output* dst ← R
OUTD* OUTDB*	dst, src, R	IR	Output and Decrement dst ← src Autodecrement src address R ← R − 1
OTDR* OTDRB*	dst, src, R	IR	Output, Decrement and Repeat dst ← src Autodecrement src address R ← R − 1 Repeat until R = 0
OUTI* OUTIB*	dst, src, R	IR	Output and Increment dst ← src *Autoincrement src address* R ← R − 1
OTIR* OTIRB*	dst, src, R	IR	Ouput, Increment and Repeat dst ← src Autoincrement src address R ← R + 1 Repeat until R = 0
SIN* SINB*	R, src	DA	Special Input R ← src
SIND* SINDB*	dst, src, R	IR	Special Input and Decrement dst ← src Autodecrement dst address R ← R − 1
SINDR* SINDRB*	dst, src, R	IR	Special Input, Decr. and Repeat dst ← src Autodecrement dst address R ← R − 1 Repeat until R = 0
SINI* SINIB*	dst, src, R	IR	Special Input and Increment dst ← src Autoincrement dst address R ← R + 1
SINIR* SINIRB*	dst, src, R	IR	Special Input, Incr. and Repeat dst ← src Autoincrement dst address R ← R + 1 *Repeat until R = 0*

INPUT/OUTPUT (Cont.)

Mnemonics	Operands	Addr. Modes	Operation
SOUT* SOUTB*	dst, src	DA	Special Output dst ← src
SOUTD* SOUTDB*	dst, src, R	IR	Special Output and Decrement dst ← src Autodecrement src address R ← R − 1
SOTDR* SOTDRB*	dst, src, R	IR	Special Output, Decr. and Repeat dst ← src Autodecrement src address R ← R − 1 Repeat until R = 0
SOUTI* SOUTIB*	dst, src, R	IR	Special Output and Increment dst ← src Autoincrement src address R ← R + 1
SOTIR* SOTIRB*	dst, src, R	R	Special Output, Incr. and Repeat dst ← src Autoincrement src address R ← R + 1 Repeat until R = 0

CPU CONTROL

Mnemonics	Operands	Addr. Modes	Operation
COMFLG	flags	–	Complement Flag (Any combination of C, Z, S, P V)
DI*	int	–	Disable Interrupt (Any combination of NVI, VI)
EI*	int	–	Enable Interrupt (Any combination of NVI, VI)
HALT*	–	–	HALT
LDCTL*	CTLR, src	R	Load into Control Register CTLR ← src
LDCTL*	dst, CTLR	R	Load from Control Register dst ← CTLR
LDCTLB	FLGR, src	R	Load into Flag Byte Register FLGR ← src
LDCTLB	dst, FLGR	R	Load from Flag Byte Register dst ← FLGR
LDPS*	src	IR DA X	Load Program Status PS ← src
MBIT*	–	–	Test Multi-Micro Bit Set S if μI is Low, reset S if μI is High
MREQ*	dst	R	Multi-Micro Request
MRES*	–	–	Multi-Micro Reset
MSET*	–	–	Multi-Micro Set
NOP			No Operation
RESFLG	flag	–	Reset Flag (Any combination of C, Z, S, P V)
SETFLG	flag	–	Set Flag (Any combination of C, Z, S, P V)

*Privileged instructions. Executed in system mode only

ANNEXE 3

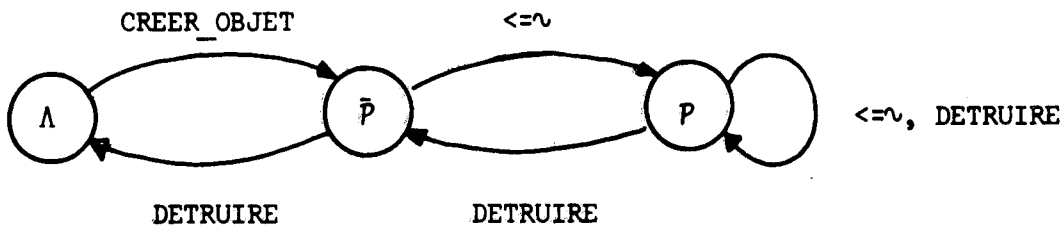
GESTION DES OBJETS

GRAPHES DES PROPRIETES TEMPORAIRES DES OBJETS

Notations : Λ état inexistant
P propriété temporaire de partage
I propriété temporaire d'immobilisation
A propriété temporaire d'activité
C propriété temporaire de catalogage
S propriété temporaire de désignation par un nom spatial

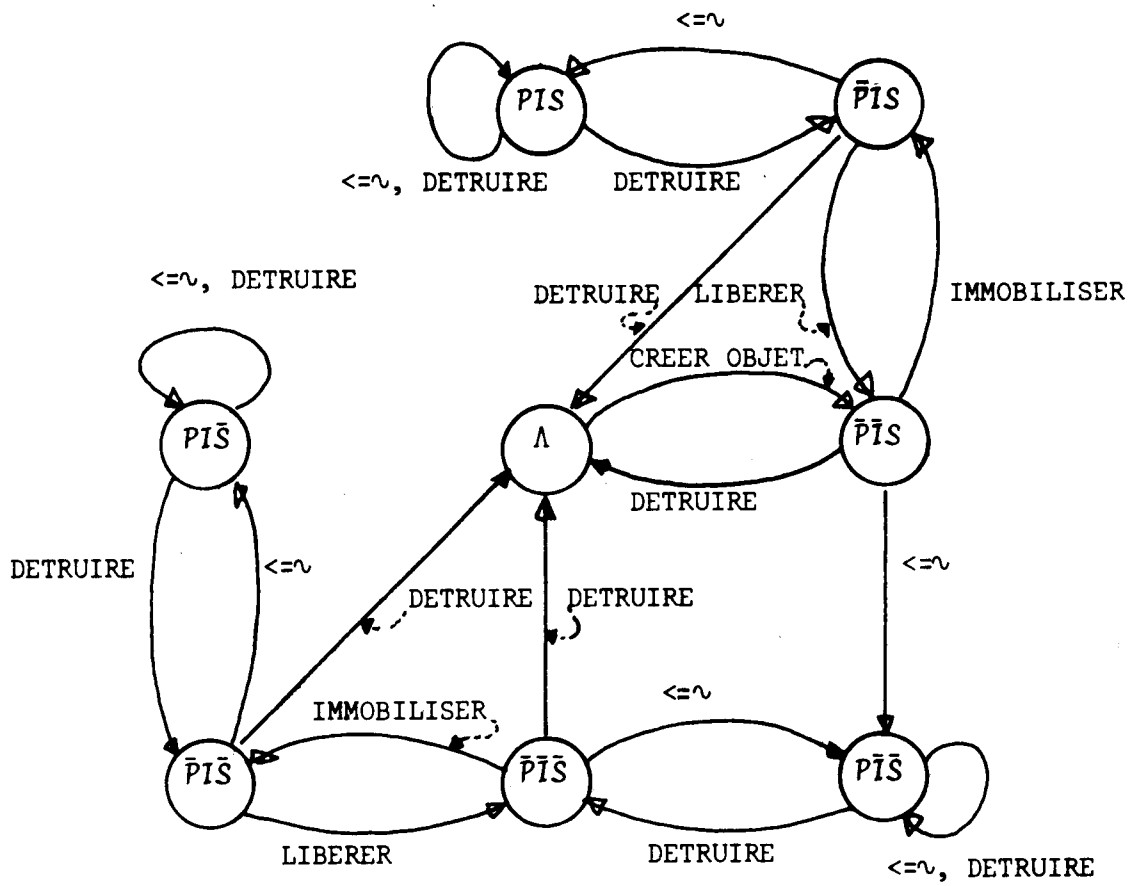
* ***Objets passagers et immobiles***

- $C/\bar{C}$ et $I/\bar{I}$ n'ont pas de sens.
- A et S sont vérifiées dans tous les cas.



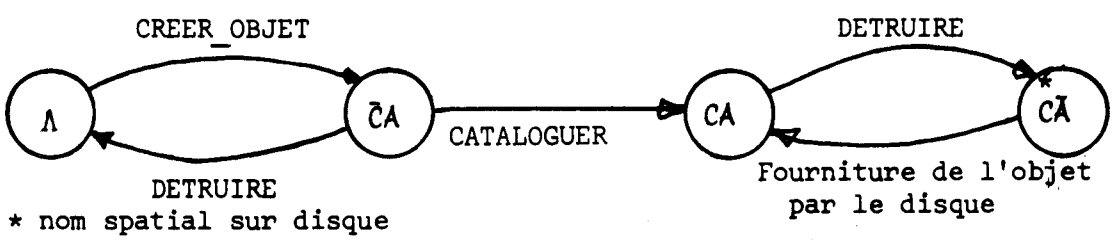
* ***Objets passagers et mobiles***

- $C/\bar{C}$ n'a pas de sens
- A est vérifié dans tous les cas



* ***Objets catalogables, non partageables et immobiles***

- $P/\bar{P}$ et $I/\bar{I}$ n'ont pas de sens
- S est vérifié dans tous les cas

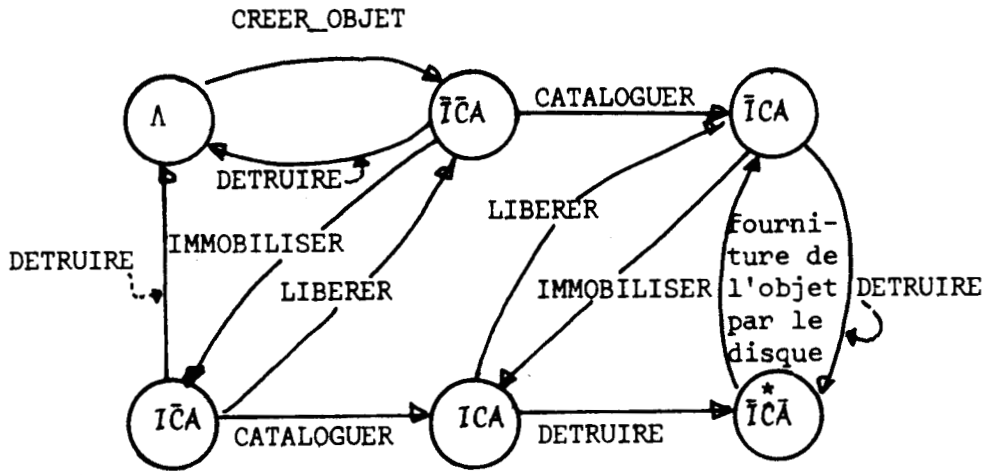


* nom spatial sur disque

BUS LILLE

* ***Objets catalogables, non partageables et mobiles***

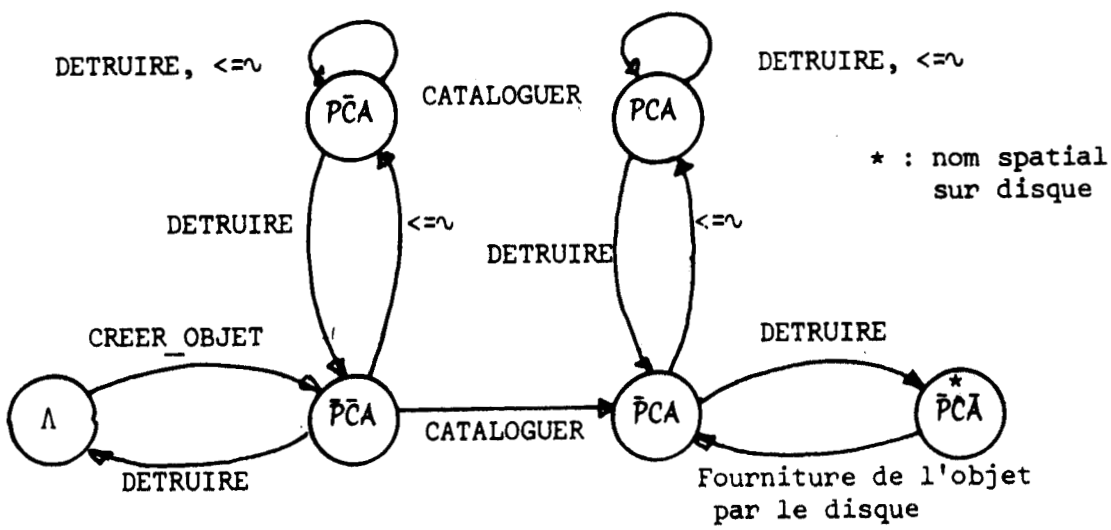
- $P/\bar{P}$ n'a pas de sens.
- S est vérifiée dans tous les cas.



* nom spatial sur disque

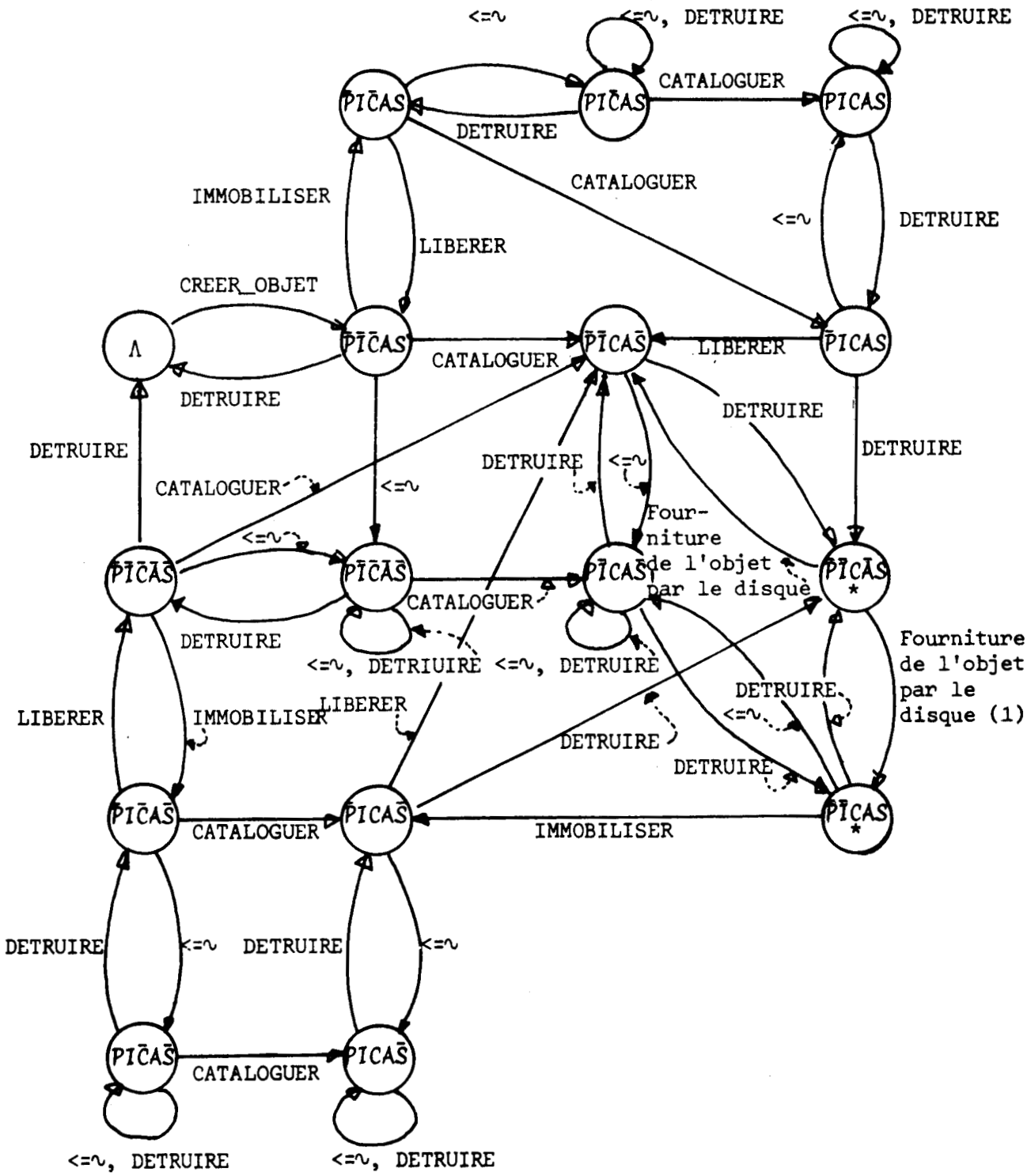
* ***Objets catalogables, partageables et immobiles***

- $I/\bar{I}$ n'a pas de sens
- S est vérifié dans tous les cas



BUS CILLE

* ***Objets catalogables, partageables et mobiles***

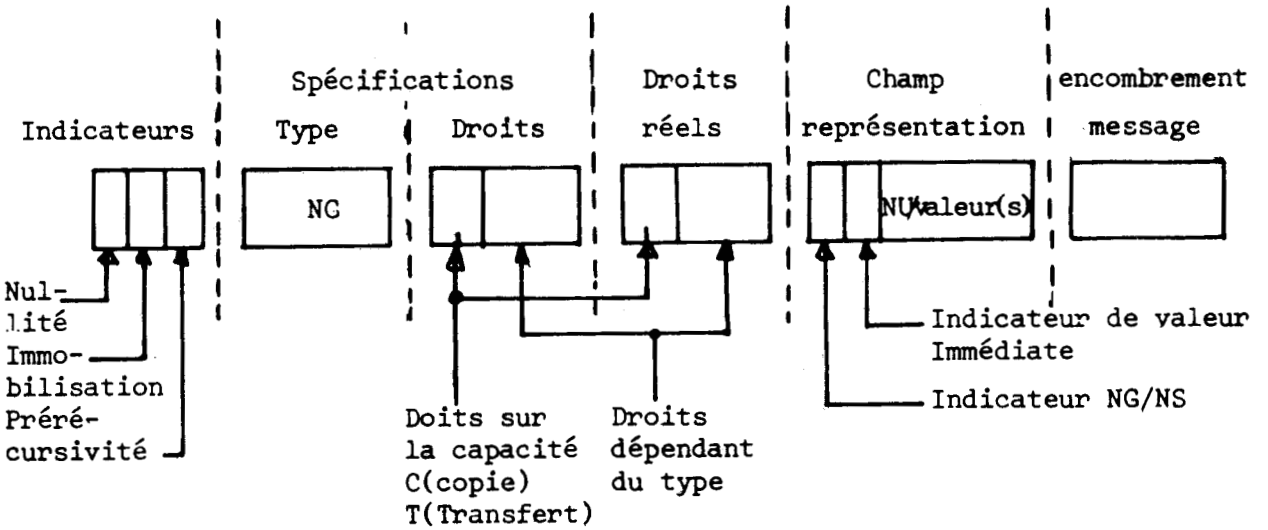


* : nom spatial sur disque

(1) : cas d'une exclusion mutuelle sur l'accès via le catalogue

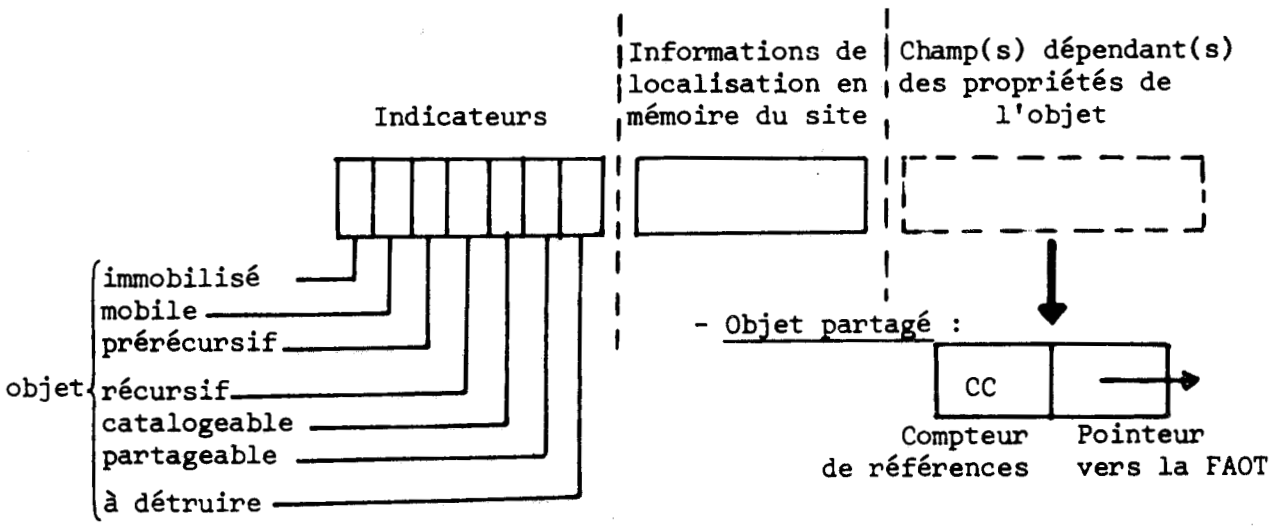
BUS LILLE

Format d'une capacité

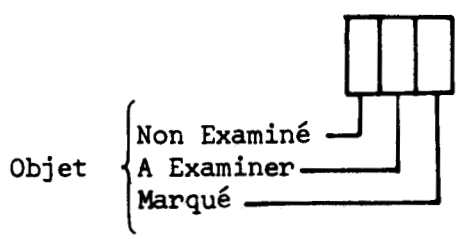


NG : Nom global, NS : Nom spatial

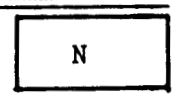
Format d'un descripteur



- Objet récursif :



- Objet catalogeable et non partageable ou immobilisé



nom dans la TGD

BUS LILLE

Liste des opérations nucléaires

Opérations	Notations	Droits repris
TYPES PRIMITIFS :		
Segment |SEG|		
- Opérations effectuées en partie pour le matériel		
Lecture dans un segment	-	L(Lecture)
Ecriture dans un segment	-	E(Ecriture)
Exécution	-	X(Exécuter)
- Autres opérations		
Lecture immédiate	V := LIRE (s, n)	L
Ecriture immédiate	ECRIRE (s, n) := V	E
Création	c <- CREER_SEGMENT (ℓ)	-
C-Liste |CL|		
Copie de capacité	d <= s	C(Copie),Ecr(Ecriture),Lec(Lecture)
Transfert de capacité	d <- s	T(Transfert), Ecr, Lec
Création	c <- CREER_C-LISTE (n)	-
Restriction des droits	c. Droits <- M	-
Maquette |MAQ|		
Interprétation	c <-∿ INTERPRETER_MAQUETTE (T,n,p)	-
TYPES NUCLEAIRES CONSTRUITS		
Objet-type |TYPE|		
Création d'objet	c <-∿ CREER_OBJET (t, p)	Cr(Création)
Création de domaine	c <-∿ CREER_DOMAINE (o, i, p)	-
Composition	0 <= COMPOSER (c, t)	Comp (Composition)
Décomposition	c <= DECOMPOSER (o, t, Dr)	Dec (Décomposition)
Domaine |DOM|		
Copie de capacité	d.e <=∿ c	C
Transfert de capacité	d.e <-∿ c	T
Transfert de paramètres	d <-∿ TRANSFERER_PARAMETRES (p_1,..,p_q)	-
Retour de paramètres	RETOURNER_PARAMETRES	-
Activation de domaines	ACTIVER d	A(Activation)
Réactivation de domaine	REACTIVER [d]	A[R(Réactivation)]
Création d'objets locaux	d <-∿ CREER_LOCAUX (m,c,p_1,..,p_q)	-
Suspension d'un domaine	ATTENDRE d[,c]	-
Fin d'exécution {	QUITTER	-
	ARRETER	-
Sémaphore |SEM|		
Blocage sur sémaphore	BLOQUER s	P
Déblocage du sémaphore	DEBLOQUER s	V
Portion de temps |TPS|		
Division	P2 <- DIVISER (p_1, q)	-
Fusion	P2 <- FUSIONNER p_1)	-
Allocation	ALLOUER_TEMPS (d, t)	-
Portion de mémoire |MEM|		
Division, Fusion : cf. TPS		
Allocation	ALLOUER_MEMOIRE (d, m)	-
OPERATIONS INDEPENDANTES D'UN TYPE		
Destruction de capacité	DETRUIRE c	
Immobilisation d'objet	IMMOBILISER (c, n)	
Libération d'objet	LIBERER c	
Catalogage d'un objet	CATALOGUER c	

ANNEXE 4

CAHIER DES CHARGES DU SOUS-SYSTÈME DE COMMUNICATION

- Le Sous-Système de Communication (SSC) est chargé d'acheminer des messages entre un site expéditeur et un ou plusieurs sites destinataires.

- L'identité du ou des sites destinataires est déterminée comme suit :

. un message peut être destiné à tous les sites : il est, alors, diffusé à tous les sites.

. un message peut être destiné à un site particulier : le message, alors destiné

"au site qui possède la représentation de l'objet de nom unique NU"

Si le nom unique est un nom spatial (n, d), n est le numéro du site destinataire. Le message est alors transmis au site n.

Si le nom NU est un nom global (n, v), le numéro du site destinataire est inconnu. Le message est diffusé à tous les sites. Un seul site l'accepte après recherche associative dans sa TOG (ou sa transposition matérielle - voir § 5.2).

- Format des messages. Un message contient au moins les informations suivantes

. le numéro du site expéditeur
. un indicateur de transmission/diffusion
. dans le cas d'une transmission à un site particulier, le nom NU.
. un code de fonction.
. des informations dépendant du code de fonction

- Liste des messages. La liste des messages est présentée comme indiqué page suivante. MX est une mnémonique. Un certain nombre de messages sont regroupés sous des mnémoniques MSn - Messages de Service numéro n -. Ce sont des messages courts en général dépourvus d'informations dépendant du code de fonction.

M X	<intitulé du message>
	<liste des informations dépendant du code de fonction>
	op = <liste des opérations générant le message>

* ***Création des objets***

M D I M	Message-Demande d'Interpréation de Maquette
	<numéro de la maquette à interpréter> <C-liste des paramètres d'interpretation> <nom unique de l'objet destinataire> <nom local dans l'objet destinataire>
	op = INTERPRETER_MAQUETE, CREER_LOCAUX
M D C O T	Message_Demande de Copie de l'Objet-Type
	<numéro de la maquette à interpréter> <C-liste des paramètres d'interprétation> <compteur de demandes> <nom unique de l'objet destinataire> <nom local dans l'objet destinataire>
	op = INTERPRETER_MAQUETTE, CREER_LOCAUX
M O C L	Message_Ordre de Création des locaux
	<nom local de la maquette à interpréter> <C-liste des paramètres d'interprétation> <nom local de la capacité à créer>
	op = CREER_LOCAUX

* ***Destruction des objets (* : message diffusé à tous les sites)***

M S 1	Ordre de décrémentation du compteur de références associé à NU
	op = DETRUIRE
M S 2	Restitution du nom global (n, v) au générateur du site n
	op = DETRUIRE
M S 3	Départ du processus de marquage*
	op = Processus de marquage
M D E	Message_Demande d'Examen
	op = Processus de marquage

BUS LILLE

* ***Transfert d'objet***

MOT	Message-Ordre de transfert
	<nom unique du domaine destinataire> <nom local dans le domaine destinataire>
	op = IMMOBILISER, INTERPRETER_MAQUETTE, ACTIVER, REACTIVER, DEBLOQUER
MTO	Message de Transfert d'Objet
	<nom local dans le domaine destinataire> <nom unique de l'objet> <descripteur de l'objet> <C-liste de l'objet>
	op = IMMOBILISER, INTERPRETER_MAQUETTE, ACTIVER, REACTIVER, DEBLOQUER

* ***Transfert de capacité***

MTC	Message de Transfert de Capacité
	<nom local dans l'objet destinataire> <capacité à transférer>
	op = <=∿, <-∿, INTERPRETER_MAQUETTE, ALLOUER_TEMPS, ALLOUER_MEMOIRE, ATTENDRE, QUITTER, ARRETER.

* ***Imobilisation/Libération***

MS10	Ordre d'Immobilisation
	op = IMMOBILISER
MS11	Ordre de Libération
	op = LIBERER

* ***Transfert de contrôle***

MDCSA	Message-Demande de choix de Site d'Activation
	<Capacité pour le domaine à activer> <nom local dans le domaine appelant>
	op = ACTIVER
MDA	Message-Demande d'Activation
	<Capacité pour le domaine à activer> <nom local dans le domaine appelant> <C-liste du domaine à activer> <compteur de demandes>
	op = ACTIVER

BU LILLE

* *Transfert de contrôle (suite)*

M O R	Message-Ordre de Réactivation
	<Capacité pour le domaine courant> <nom local dans le domaine appelant>
	op = REACTIVER
M T P	Message de Transfert de Paramètres
	<C-liste des paramètres> <sens du transfert appelant <-> appelé> <nom local dans le domaine appelant>
	op = TRANSFERER_PARAMETRES, RETOURNER_PARAMETRES
M S 12	Ordre de bloquage sur sémaphore
	op = BLOQUER
M S 13	Ordre de déblocage d'un domaine bloqué sur sémaphore
	op = DEBLOQUER

* *Gestion des noms (1) : message destiné au site disque*

M S 5	Ordre d'incrémentation du compteur de références associé à NU
	op = <=∿, COMPOSER, DECOMPOSER
M S 6	Substitution du nom global (n, v) au nom spatial (n, d)
	<nom global (n, v)>
	op = LIBERER, CATALOGUER, <=∿
M S 7	Liaison du nom global (n, v) à un descripteur en TGD (1)
	op = CATALOGUER
M S 8	Ordre de génération d'un nom dans la TGD (1)
	<nom de l'objet destinataire>
	op = CATALOGUER
M S 9	Retour de nom dans la TGD
	<nom dans la TGD>
	op = CATALOGUER

BUS LILLE

* *Gestion des noms (suite)*

M S 4	Actualisation de la liaison nom en TGD - nom spatial
	<nom en TGD> <nom global>
	op = LIBERER

- Le fonctionnement du SSC ne doit pas réduire la disponibilité du système.
- Le SSC doit contrôler l'intégrité des capacités contenues dans certains messages.
- Le SSC est chargé d'acheminer les informations de charge des sites.
- Les interfaces site-SSC doivent être capables de mesurer la charge du SSC.
- Certaines opérations du SSC doivent être indivisibles :

. Transmission de tous les MOT concernant un même domaine (§ 5.3)

. Test de terminaison du processus de marquage ou de son initialisation (§ 5.7.3)

. Transmission d'un nom global de la TOG d'un site à la TOG d'un autre site (§ 5.6.2).

BIBLIOGRAPHIE

[1] Advanced Micro Devices
"Am Z80001/2 Processor Instruction Set"
1979

[2] C. BETOURNE et S. KRAKOWIAK
"Mesures dans un système conversationnel"
RAIRO, B-2, Juillet 1975, 5-15.

[3] A.D. BIRELL and R.M. NEEDHAM
"An asynchronous garbage collector for the CAP filing system"
Operating Systems Review, Vol 12, N° 2, April 1978, 31-33.

[4] L.M. CASEY
"Computer structures for distributed systems"
Thesis, Univ. of Edinburgh, December 1977.

[5] L.M. CASEY, N. SHELNESS
"A domain structure for distributed computer systems"
Proc. of the sixth ACM Symp. on Operating Systems Principles, November 1977, 101-108.

[6] D.C. COSSERAT
"A data model based on the capability protection mechanism"
in "Protection in Operating Systems", IRIA, 1974, 35-53.

[7] CROCUS
"Systems d'exploitation des ordinateurs"
Dunod - Seconde édition - 1975.

[8] P. DARONDEAU
"Outils de protection statique dans les systèmes multi-langages"
in "Second colloque international sur les systèmes d'exploitation", IRIA, octobre 1978.

[9] P.J. DENNING
"Fault tolerant operating systems"
Computing surveys, Vol 8, N° 4, december 1976, 359-389.

[10] E.W. DIJKSTRA
"The structure of the "THE" Multiprogramming system"
Communications of the ACM, Vol 11, N° 5, may 1968, 341-345.

[11] D.M. ENGLAND
"Capability concept mechanisms and structure in System 250"
in "Protection in operating systems", IRIA, Août 1974.

[12] P.H. ENSLOW
"Multiprocessors and parallel processing"
COMTRE Coorporation.

[13] R.S. FABRY
"Capability-based addressing"
Communications of the ACM, Vol 17, n° 7, july 1974, 403-412.

[14] J. FERRIE
"Contrôle de l'accès aux objets dans les systèmes informatiques"
Thèse Université de Paris VI - Septembre 1975.

[15] FERRIE, KAISER, LANCIAUX, MARTIN
"An extensible structure for protected systems design"
in "Protection in operating systems" IRIA, Aout 1974, 74-83.

[16] E.A. FEUSTEL
"On the advantages of tagged architecture"
IEEE Transactions on computers, Vol C22, N° 7, July 1973, 644-656.

[17] S.H. FULLER, J.K. OUSTERHOUT, L. RASKIN, P.I. RUBINFELD, P.J. SINDHU, R.J. SWAN
"Multi-microprocessors : our everview and working exemple"
Proc of the IEEE, Vol 66, n° 2, February 1978, 216-288.

[18] A.N. HABERMANN, L. FLON, L. COOPRIDER
"Modularization and hierarchy in a family of operating systems"
Communications of the ACOR, Vol 9, n° 5, May 1976, 266-272.

[19] B.K. HADDAN
"Nested monitor calls"
ACM-SIGOPS-Operating Systems Review, Vol 12, n° 1, janv 78, 12-14.

[20] B. HANSEN
"The nucleus of a multiprogramming system"
Communications of the ACM, Vol 13, n° 4, December 1970, 238-242.

[21] B. HANSEN
"A programming methodology for operating system design"
IFIP Congress, 1974, 394 - 397.

[22] A.J. HERBERT
"A new protection architecture for the Cambridge Capability Computer"
ACM-SIGOPS- Operating Systems Review, Vol 12, N° 1, jan 1978, 24-28.

[23] A.J. HERBERT
"A hardware-supported protection architecture"
in "Second colloque international sur les systèmes d'exploitation"
IRIA, octobre 1978.

[24] C.A.R. HOARE
"Monitors : an operating system structuring concept"
Communications of the ACM, vol 17, n° 10, october 1974, 549-557.

[25] S. HOENER and W. ROEHDER
"Modular multi-microprocessor architecture with virtual memory"
Second Symposium on micro-architecture, EUROMICRO, 1976, 99-108.

[26] INTEL
"MCS - 86 users manuel"
Juillet 1978

[27] A.K. JONES, R.J. CHANSLER, I. DURHAM, P. FEILER, K. SCHWANS.
"Software management of Cm - A distributed multiprocessor"*
AFIPS NCC, Vol 46, 1977, 657-663.

[28] A.K. JONES and B. LISKOV
"A langage extension for expressing constraints on data access"
Communications of the ACM, Vol 21, n° 5, May 1978, 358-367.

[29] M. JOSEPH and V.R. PRASAD
"More on nested monitor calls"
ACM-SIGOPS, Operating System Review, vol 12, n° 2, Ap. 1978, 21-25.

[30] R.B. KIEBURTZ and A. SILBERSCHATZ
"Capability managers"
IEEE Transactions on Software Engineering, vol SE 4, n° 6, nov 1978, 467-477.

[31] B.W. Lampson
"Dynamic protection structures"
AFIPS FJCC, vol 35, 1969, 27-38.

[32] B.W. LAMPSON and H.E. STURGIS
"Reflections on an operating system design"
Communications of the ACM, Vol 19, n° 5, may 1976, 251-265.

[33] D. LANCIAUX et B. MARTIN
"Définition d'un adressage pour une machine à domaines"

[34] D. LANCIAUX
"Mécanismes de structuration des systèmes à capacités"
Thèse, Université de Lille, octobre 1978

[35] H.C. LAUER
"Protection and hierarchial adressing structures"
in "Protection in operating system", IRIA, 1974, 137-148

[36] B. et J. LEGUY
"Mémoire active limitée aux besoins de l'utilisateur"
Université de Lille, Publication n° 114, Août 1978.

[37] LEROUDIER
"Une analyse de système"
Thèse 3ème cycle, Grenoble, 1973

[38] T.A. LINDEN
"Operating system structures to support security and reliable software"
"Computing surveys, vol 8, n° 4, december 1976, 409-445.

[39] B. LISKOV
"An introduction to CLU"
in "New directions in algorithmic languages", IRIA, 1957, 139-156.

[40] B. LISKOV, A. SNYDER, R. ATKINSON, C. SCHAFFERT
"Abstractions mechanisms in CLU"
Communications of the ACM, Vol 20, n° 8, August 1977, 564-576.

[41] A. LISTER
"The problem of nested monitor calls"
ACM-SIGOPS, Vol 11, n° 3, july 1977, 5-7.

[42] M. LUCAS
"Conception modulaire des systèmes d'exploitation - Outils pour la programmation modulaire"
Thèse - Université de Grenoble, juin 1977

[43] M. LUCAS et J. MONTUELLE
"Conception modulaire des systèmes d'exploitation - Présentation du projet SESAME"
Thèse - Université de Grenoble - Juin 1977.

[44] I. LE MAIR
"Complex systems are simple to design"
Electronic Design 18, September 1, 1978, 100 - 107.

[45] M. MESSI N'JANGUE
"OMPHALE : le sous-système de communication"
Université de Lille, publication interne, septembre 1979.

[46] D. MICHAEL
"Etude des langages typés pour l'écriture des systèmes"
Mémoire de DEA, Université de Lille, septembre 1979.

[47] J. MONTUELLE
"Conception modulaire des systèmes d'exploitation - Méthode et exemple d'application"
Thèse - Université de Grenoble - Juin 1977.

[48] R.M. NEEDHAM
"Protection systems and protection implementations"
AFIPS FJCC, vol 41, 1972, 571-578.

[49] R.M. NEEDHAM and R.D.H. WALKER
"Protection and process management in the "CAP" computer"
in "Protection in operating systems", IRIA, 1974, 155-180

[50] R.M. NEEDHAM and R.D.H. WALKER
"The Cambridge CAP Computer and its protection system"
Proceeding of the sixth ACM symposium on operating systems principles"
Nov 1977, 1-10.

[51] R.M. NEEDHAM and M.V. WILDES
"Domains of protection and management of process"
The computer journal, vol 17, n° 2, 117-120.

[52] P.G. NEUMANN, R.S. FABRY, K.N. LEVITT, L. ROBINSON, J.H. WENSLEY
"On the design of a provably secure operating system"
in "Protection in operating systems", IRIA, 1974, 166-175.

[53] D.L. PARNAS
"On the criteria to be used in decomposing systems into modules"
Communications of the ACM, vol 15, n° 12, Decembre 1972, 1053-1058.

[54] D.L. PARNAS
"On a "buzzword" : hierarchical structure"
IFIP Congress, 1974, 336-339.

[55] D.L. PARNAS
"The non-problem of nested monitor calls"
ACM SIGOPS, Operating Systems Review, Vol 12, n° 1 (Jan. 78) 12-14.

[56] R.V.S. PRASAD, N. NATARAJAN and M.K. SINHA
"Physical and logical abstraction in a kernel"
Second colloque international sur les systèmes d'exploitation, IRIA, octobre 78.

[57] C.V. RAMAMOORTHY, T. KRISHNARRO, P. JAHANIAN
"Hardware software issues in multiprocessor computer architectures"
1st annual Rocky Mountain Symposium on microcomputers, 1977, 73-79.

[58] D.D. REDELL
"Nanning and protection in extensible operating systems"
M.I.T., Project MAC, TR-140, 1974.

[59] S.E. SCRUPSKI
"Minis, look out : here comes a powerful family of 16 bits C chips"
Electronic design 12, June 7, 1978, 54-58.

[60] M. SHIMA
"Two versions of 16 bit chips span microprocessor, minicomputer needs"
Electronics, december 21, 1978, 81-88.

[61] U.A. SHOLL and A.D. ELDRIDGE
"Acomparative analysis of selected system architectures using multiple micro/mini computers"
Micro 77 computer conference record - Oklahoma city - april 1977, 186-190.

[62] M.J. SPIER
"A model implementation for protective domains"
Int. Journal of Computer and Information Science, Vol 2, n° 3, 1973, 201-229.

[63] M.J. SPIER, T.N. HASTINGS and D.N. CUTLER
"A storage mapping technique for the implementation of protective domains"
Software practice and experience, Vol 4, 1974, 215-230.

[64] R.J. SWAN, H. FULLER, D.P. SIEWIOREK
"The structure and architecture of Cm : a modular multi-micro-processor"*
Computer Science Review, CMU, december 76, 25-47.

[65] R.J. SWAN, A. BECHTOLSHEIM, K.W. LAI, J.K. OUSTERHOUT
"The implementation of the Cm*multi-microprocessor"
AFIPS NCC, Vol 46, 1977, 645-655.

[66] H. WETTSTEIN
"The problem of nested monitor calls revisited"
ACM SIGOPS, Operating System Review, vol 12, n° 1, janv 78, 19-23.

[67] W. WULF
"HYDRA : the kernel of a multiprocessor operating system"
Communications of the ACM, Vol 17, n° 6, June 1974, 337-345.

[68] W.A. WULF, R.L. LONDON, M. SHAW
"Abstraction and verification in ALPHARD"
in "New directions in algorithmic languages", IRIA, 1975, 217-295.

[69] W.A. WULF and C.G. BELL
"Cmmp - A multi-mini-processor"
AFIPS FJCC, 1972, 765-777.

[70] W.A. WULF
"Reliable hardware - software architecture"
ACM, SIGPLAM Notices, Vol 10, N° 6,1975, 122.

[71] ZILOG
"Z8001 CPM - Z8002 - Product specification"
1978

[72] ZILOG
"The Z8000 family of microcomputer components"
1978.