Un point de vue sur le contrôle des accès parallèles à des sources partageables

José Graça Martins

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR INGENIEUR
«informatique»

par

José GRAÇA MARTINS

UN POINT DE VUE SUR LE CONTROLE DES ACCES PARALLELES A DES RESSOURCES PARTAGEABLES

Thèse soutenue le 3 avril 1980 devant la commission d'examen

Président: L. BOLLIET
Examinateurs: J.S. BANINO
N.X. DANG
C. DELOBEL
J.P. VERJUS
H. ZIMMERMANN

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

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

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioélectricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

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

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

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

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

MM.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges — Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

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

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

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

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

E.N.S.I.M.A.G.

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

à mon Père

à Génita

La soutenance d'une thèse par un étudiant étranger, ne peut pas être considérée comme un simple point d'une trajectoire qui trouve sa continuité-passée dans les schémas traditionnels d'une structure d'enseignement, et sa continuité-future dans le tissu socio-culturel d'un pays.

Ce type d'expérience fait partie d'une dynamique sociale (au sens large), qui doit apporter des bénéfices à toutes les parties. Néanmoins, elle représente pour l'individu qui la vit un point de discontinuité, soit d'un point de vue professionnel, soit d'un point de vue humain. L'expérience est passionante puisque l'effort d'intégration, à tous les niveaux, exige de l'individu l'application prolongée de toutes ses facultés, en lui donnant une meilleure connaissance de lui-même et donc des autres. Aussi, son résultat dépend en grande partie de l'environnement dans lequel il est plongé.

Le manuscrit de thèse n'étant simplement que l'expression d'une froide réalité technique, les remerciements sont le seul endroit où il est possible de faire le compte-rendu de cette expérience.

J'ai été reçu dans l'équipe "Réseaux et Télétraitement" par M. Guy SERGEANT (responsable de cette équipe) et M. Michel DANG, à une époque difficile pour cette équipe à la suite de la récente disparition de son créateur et responsable Monsieur DU MASLE. Dans de telles circonstances, je ne peux pas oublier le soutien et l'orientation de Michel DANG, pendant la préparation du DEA qui a été axé sur la conception d'un protocole de communication entre les processus du système SIGOR. Ses encouragements ont été un facteur décisif pour la poursuite de mon travail en direction d'une thèse.

C'est alors qu'a débuté une période importante de mon séjour en France. Jusqu'en Décembre 1978 j'ai participé à la mise en oeuvre du système SIGOR, et j'ai eu l'occasion de travailler de près avec Guy SERGEANT. Sa compétence alliée à ses caractéristiques humaines ont fait de cette période des moments de travail intense, à la fois calmes et très riches d'enseignements. Je lui témoigne ma profonde estime et mon admiration.

Néanmoins, pour des motifs divers et indépendants du travail technique de l'équipe, le projet a été arrêté en cours de route, et j'ai voulu choisir un thème de thèse sans trop de rapport avec SIGOR. Un stage à l'IRIA pendant les mois de Janvier à Mars 1979, m'a donné un temps de réflexion important pour la prise de décisions concernant l'avenir.

Je veux remercier Monsieur ZIMMERMANN de m'avoir accepté en stage dans son équipe de recherche sur les systèmes répartis. Le dynamisme et l'imagination que j'ai pû observer dans les réunions de travail qu'il dirigeait, m'ont beaucoup marqué et seront toujours une référence pour moi. Je tiens aussi à le remercier d'avoir bien voulu participer au jury de cette thèse.

Je veux également exprimer ma gratitude à toute l'équipe CHORUS pour l'amitié qu'elle m'a témoignée. En particulier je veux remercier Monsieur Jean-Serge BANINO par son soutien critique qui m'a beaucoup aidé pendant la rédaction de cette thèse, et qui a bien voulu accepter d'en être le rapporteur extérieur.

Revenu à Grenoble, Michel DANG a accepté de me guider dans le développement du thème qui a conduit à ce travail de thèse. Le chemin n'a pas toujours été sans encombre, mais après toutes nos discussions j'ai pu déterminer clairement ce qui "allait" et ce qui "n'allait pas". Pour son amitié, ses critiques, sa ténacité et les innombrables heures de travail passées ensemble (ou passées sur des versions successives du texte), je lui présente mes vifs remerciements et ma grande estime.

La rédaction d'une thèse dans une langue étrangère est naturellement une difficulté de plus à surmonter. Je veux donc remercier Guy SERGEANT qui, lui aussi, n'a jamais refusé son aide pour rendre plus lisibles les morceaux de textes que je produisais, en ajoutant toujours ses critiques qui m'ont été d'une aide précieuse.

Il n'est pas possible d'énumérer ici les noms de tous ceux qui d'une façon ou d'une autre ont contribué à l'exécution de ce travail. Je n'en citerai qu'un, dont l'amitié, les "footings" au bord de l'Isère et les discussions sur les aspects langage de ce travail, ont beaucoup contribué à sa concrétisation. Merci Michel DELAUNAY. En particulier, je tiens à remercier Michel DANG et Michel DELAUNAY pour la peine qu'ils se sont donnés à transcrire mes algorithmes sous une forme proche du Pascal, tels qu'ils sont présentés dans le texte.

Je remercie M. BOLLIET, professeur à l'Université de Grenoble, qui m'a fait l'honneur de présider le jury de cette thèse.

Je remercie aussi M. DELOBEL, professeur à l'Université de Grenoble, qui a bien voulu accepter le rôle de rapporteur interne.

En Septembre 1979, j'ai dû faire un petit exposé sur le thème de cette thèse à M. VERJUS, professeur à l'Université de Rennes. Ses encouragements à poursuivre ont été une grande aide pour moi. Je tiens aussi à le remercier d'avoir accepté de prendre part au jury. Ces remarques sur le texte final ont permis encore plusieurs modifications qui ont beaucoup amélioré ce travail.

Je remercie Mme TREVISAN et Mme ROCHE pour tout le mal qu'elles se sont donné dans la frappe, en deux jours, de cette thèse.

Je remercie enfin tout le personnel du service de reprographie pour la réalisation matérielle de cet ouvrage.

I - INTRODUCTION

L'informatique offre un moyen pour modéliser un domaine de la réalité. Un des effets du développement de l'Informatique, est l'élargissement du domaine de la réalité modélisable à travers cette technique. Quand les systèmes informatiques répartis et l'utilisation massive des micro-processeurs deviendront une réalité, nous devons nous attendre à observer une extension de ce phénomène.

En particulier, le problème de la gestion des ressources partagées, prendra une nouvelle dimension. Jusqu'ici il était presque exclusif du domaine des systèmes d'exploitation, mais maintenant nous croyons qu'il va devenir aussi un problème de l'utilisateur.

Les ressources définies par l'utilisateur auront une sémantique proche de celle des objets physiques correspondants, qui peut être assez différente de celle des objets informatiques gérés habituellement par les systèmes d'exploitation (imprimantes, disques, mémoire, ...). Ceci lève des problèmes nouveaux, ou peu habituels jusqu'ici.

Ce fait amène à des caractéristiques pour les algorithmes de gestion des ressources, qui justifient des propositions nouvelles pour plusieurs mécanismes, notamment pour les techniques de gestion du parallélisme.

Nous allons décrire à travers un exemple simple un problème de gestion des ressources décrites par l'utilisateur. L'exemple nous permet de mettre en évidence deux caractéristiques qui nous semblent intéressantes, et peu habituelles dans les problèmes de gestion des ressources qui sont habituellement posés et résolus aujourd'hui. Nous montrons que les problèmes qui ont ces caractéristiques ne sont pas résolus par une classe de propositions, dont les propriétés sont cependant particulièrement intéressantes (expressions de chemin [CAM 74], compteurs [ROB 77], ...).

Nous proposons une technique de contrôle du parallélisme, qui tient compte des caractéristiques citées. Cette proposition est développée d'abord dans un environnement où une seule ressource est accédée en concurrence par plusieurs utilisateurs.

Cette proposition est généralisée par la suite, au cas des accès concurrents à un système de ressources, sans que cette généralisation tienne compte des phénomènes d'interblocage. Néanmoins, notre proposition peut faciliter la tâche des programmeurs qui doivent éviter ce type de phénomène ; ceci est dû au fait que les structures algorithmiques (appelées plus tard services) qui spécifient les accès concurrents sont bien isolées dans l'algorithme global de l'application, et peuvent être compilées et cataloguées séparément de celui-ci. En plus, le "graphe de synchronisation" (formalisme graphique pour l'expression des règles de synchronisation des accès à une ressource-cf. 3.3.2.) nous semble une aide intéressante pour la spécification des services.

Finalement, nous considérons un environnement réparti (plusieurs sites), où les accès à une ressource peuvent être demandés à partir des sites autres que celui où la ressource est implémentée. Nous décrivons le fonctionnement de notre proposition, dans cet environnement, et nous décrivons aussi un ensemble d'outils suffisants pour la mettre en oeuvre.

Pour faciliter la lecture du texte et sa compréhension, le choix est fait d'illustrer le développement de la proposition par le développement simultanée de l'exemple proposé au début.

2 - EXEMPLE INTRODUCTIF

2.1. Les comptes bancaires

Une institution bancaire doit, entre autres, faire la gestion des comptes bancaires, partagées par plusieurs personnes (exemple : les conjoints, une société), que nous appelons les usagers des comptes.

Un compte bancaire est décrit par son solde, et il est soumis aux actions (ou accès) suivants :

- Addition
- Soustraction
- Consultation
- Suspendre
- Libérer

N.B. - Ici, les termes suspendre et libérer n'ont aucun rapport avec les homonymes informatiques - ils sont pris au sens bancaire. La banque peut suspendre un compte, par exemple, pour des motifs judiciaires (un des titulaires vient à mourir, faillite de l'entreprise, ...). Libérer est l'action inverse de l'action suspendre. Le mécanisme de protection doit garantir que les accès suspendre et libérer sont exécutés seulement par des utilisateurs privilégiés.

L'exécution des actions précédentes peut être demandée en concurrence par des usagers différents, d'où le besoin d'établir des règles de synchronisation entre les accès, qui tendent à garantir l'intégrité de chaque compte :

a) Un compte peut être soumis à l'action libérer, si et seulement si, il est suspendu. Réciproquement un compte peut être soumis à l'action suspendre, si et seulement si il est non-suspendu.

b) Un compte peut être consulté, si et seulement si il est non-suspendu. Il peut être consulté par plusieurs usagers simultanément. Le compte redevient non-suspendu quand il n'y a plus d'usagers qui font des consultations.

c) Le solde d'un compte peut être modifié (actions addition ou soustraction), si et seulement si il est non-bloqué. Il ne peut être modifié que par un seul usager à la fois. Quand chaque modification se termine, le compte revient à l'état non-suspendu.

Nous pouvons concevoir un niveau à l'intérieur de l'organisation de la banque pour lequel cette description est suffisante (*) : le "niveau administrateur' ou "niveau 0". Dans ce niveau sont définis les accès et les règles de synchronisation des accès à la ressource; règles de synchronisation permettant d'assurer l'intégrité de la ressource.

Sur l'ensemble des comptes bancaires gérés par la banque, sont définis des services plus élaborés, à la demande des usagers. Un service peut être défini globalement comme une séquence ordonnée d'accès, ayant une sémantique propre. Par exemple, de tels services sont :

(*) - Exemple : niveau de la comptabilité.

1) Service "apport d'argent"

Soit un compte de dépôt à vue, partagé par plusieurs usagers. Selon la consigne laissée par ces usagers, toute addition d'argent doit donner lieu à un achat immédiat d'or si telle somme est atteinte dans le compte de dépôt ; ce qui revient à dire que l'accès d'addition dans le compte de dépôt peut être dynamiquement suivi par un accès de soustraction et dans ce cas, le solde du compte ne doit pas changer entre les deux accès.

2) Service "retrait d'argent"

Soit un compte d'épargne-logement partagé par les mêmes usagers du compte de dépôt à vue. Ces usagers peuvent demander que lors d'un retrait d'argent sur le compte de dépôt, si celui-ci ne comporte pas assez d'argent et s'il y en a assez dans le compte d'épargne-logement (ce qui sous-entend un accès en lecture au solde du compte épargne-logement) on retire de l'argent dans ce compte pour faire l'appoint au compte de dépôt (accès d'addition) avant de faire le retrait d'argent (accès de soustraction).

Les services sont décrits par un "niveau 1" ou "niveau des services", qui utilise l'interface d'accès à la ressource définie par le niveau 0.

Dans cet exemple, nous voyons que les séquences d'accès qui constituent les services dépendent de la valeur du solde de chaque compte pendant l'exécution du service. En effet, un processus exécutant le service apport d'argent pour le compte d'un usager exécute l'achat d'or (accès soustraction) seulement si le solde est supérieur à une certaine valeur (après l'accès addition). De même dans le service retrait d'argent, le retrait sur le compte d'épargne-logement est exécuté seulement si le solde du compte de dépôt est inférieur à la valeur du retrait.

Nous voyons donc, dans le cas de cet exemple, que la valeur de la partie de la représentation de la ressource qui est partageable par les usagers (le solde), ne doit pas changer entre deux accès consécutifs d'un service exécuté pour compte d'un usager.

Nous faisons la convention de désigner la valeur de la partie de la représentation de la ressource qui est partageable par les usagers, simplement par valeur partageable de la ressource, dans le texte qui suit. La valeur partageable d'un compte bancaire est son solde.

Cet exemple des comptes bancaires illustre une situation possédant deux caractéristiques qui nous semblent particulièrement intéressantes, que nous décrivons ci-après. Nous pensons qu'un tel type de situation sera courant dans les problèmes de gestion de ressources partagées des utilisateurs à venir.

2.2. Caractéristiques intéressantes

2.2.1. Première caractéristique

Nous avons structuré l'exemple des comptes bancaires en deux niveaux d'abstraction :

a) Niveau administrateur ou niveau 0, où les ressources sont décrites et les règles de synchronisation énoncées.

b) Niveau des services ou niveau 1, où sont définies les séquences d'accès constituant les services.

Remarquons comme première caractéristique intéressante de cet exemple, que cette structuration du modèle informatique n'a pas comme but principal, de simplifier la solution du problème, mais plutôt qu'elle a été imposée comme reflet d'une structuration déjà existante au niveau du modèle physique.

En effet, un service intéressant aujourd'hui pour un usager, peut ne plus l'être demain, ou peut devoir être modifié pour répondre à des nouvelles sollicitations de l'usager. Alors que la description des ressources faite par l'administrateur a un caractère plus stable, car liée à la nature physique des ressources.

Il est donc intéressant que cette structuration soit prise en compte, par une technique de contrôle du parallélisme, et par le moyen d'expression associé. Des modifications, des insertions ou des suppressions de services pourront être effectuées, sans toucher ni à la description des ressources ni aux règles de synchronisation.

2.2.2. Deuxième caractéristique

Aussi, nous défendons comme [ROB 77] "that in designing a parallelism control technique, there are numerous advantages to the separation of the description of the process to be synchronized of the description of the synchronizing rules". C'est le cas des expressions de chemin [CAM 74] (cf. Annexe B-I) et des compteurs [ROB 77] (cf. Annexe B-II), par exemple.

La description des règles de synchronisation correspondant au niveau 0, peut être faite à travers le langage de contrôle des propositions auxquelles on a fait référence. Par contre, c'est la technique de contrôle du parallélisme de ces mêmes propositions, qui est incompatible avec une autre caractéristique de notre problème.

Comme nous l'avons fait remarquer, la séquence des accès qui constituent un service dépend de la valeur de la ressource pendant l'exécution du service ; cette deuxième caractéristique est particulièrement intéressante.

En général, les résultats d'un accès A_i (cf. figure 1) peuvent être des informations sur la valeur de la ressource, ou sur l'état de la ressource (cf. 3.2.). Ces informations seront utilisées au niveau du service pour déterminer l'accès suivant (A_j).

La sémantique de la détermination dynamique des accès, doit donc être exprimée au niveau des services (niveaux 1), dans le type de problèmes que nous voulons caractériser (et résoudre).

Ce fait exige que associé à la technique de contrôle du parallélisme, un ensemble de règles définissent des conditions (au moins suffisantes) pour que la valeur de la ressource ou son état, ne soient pas modifiées entre deux accès appelés consécutivement à partir de l'exécution d'un service. Sinon la sémantique du service ne sera plus satisfaite.

Dans les bases de données réparties, ce problème est résolu en verrouillant les accès à une partie de la base pendant l'exécution d'une transaction par un utilisateur [ROS 78].

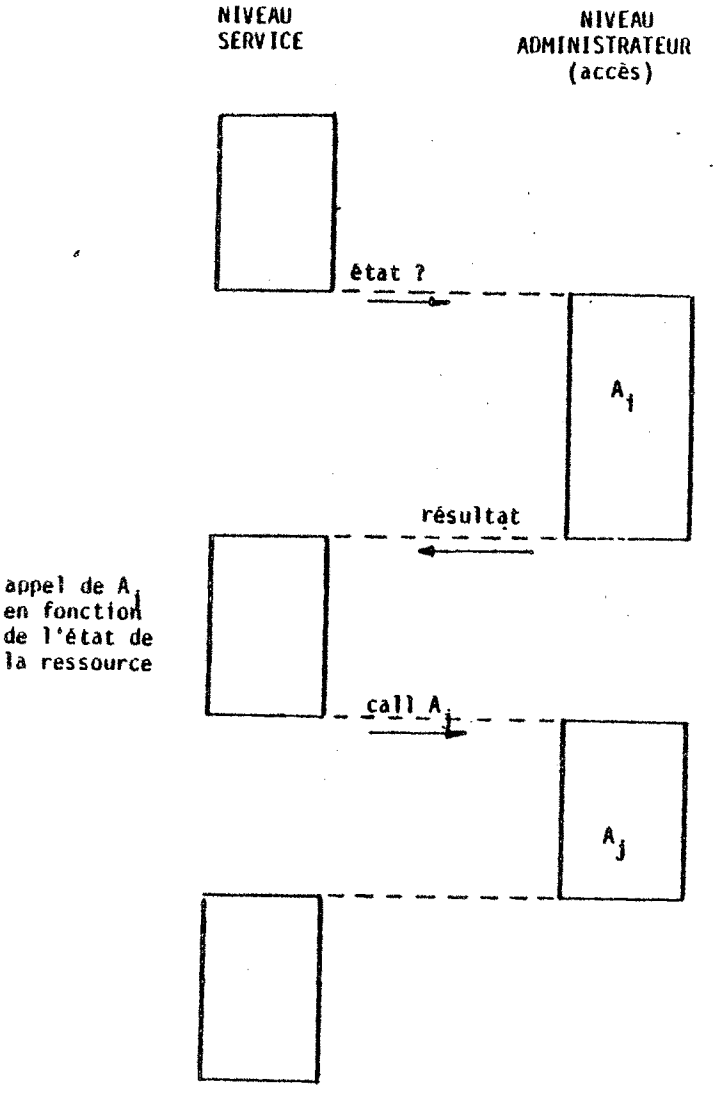


FIGURE 1

Les propositions telles que les expressions de chemin et les compteurs, ne peuvent pas assurer ce type de contrainte. En effet, ces techniques de contrôle du parallélisme se contentent de (*) :

(*) CAM 74 :"Each path expression is implemented by a controller ... The controller mechanism could operate as follows: Each procedure commences with a prologue and finishes with an epilogue. A process executing the prologue of a synchronized procedure enquires of the controller whether it may proceed. The controller, using the synchronization specification, may decide either to delay execution or to allow it to continue. Finally, when the process executes the epilogue of the procedure, it notifies the controller which may be able to release other delayed processes".

- Faire précéder tout accès à chaque ressource, d'un prologue de contrôle, où le processus demandeur de l'accès peut être mis en attente sur la vérification d'une condition déterminée, sur l'état d'accessibilité de la ressource.

- Faire suivre tout accès à chaque ressource, d'un épilogue de contrôle, servant à redéfinir l'état d'accessibilité de la ressource (si nécessaire).

Dans ces propositions, les conditions manipulées dans les prologues et épilogues de contrôle, ne tiennent compte que de la nature de l'accès demandé. Elles sont indépendantes de l'identification du processus demandeur.

Dans ces conditions, si à travers les techniques de contrôle citées il faut assurer que, entre deux accès consécutifs à une ressource, exécutés dans un service, la valeur de la ressource ne se modifie pas, il est nécessaire que ce fait soit pris en compte par la sémantique des prologues et des épilogues de contrôle des accès (qui devrait concerner seulement les règles de synchronisation - niveau administrateur). D'autre part, ces mêmes techniques de contrôle ne permettent pas d'assurer que c'est le même processus qui exécute les deux accès.

Les règles de synchronisation de l'exemple du compte bancaire, peuvent être exprimées par l'expression de chemin :

path addition , soustration , {consultation} ,(suspendre ; libérer) end

Deux séquences d'accès à la ressource se déroulant en parallèle et respectant l'expression de chemin énoncée peuvent se terminer par une incohérence (le solde du compte bancaire devient débiteur par exemple.

Nous proposons par la suite, une technique de contrôle du parallélisme qui résoud les problèmes mis en évidence par les caractéristiques décrites ci-dessus.

Pour faciliter la lecture, la présentation graduelle de la description de l'exemple des comptes bancaires servira à illustrer le texte.

Dans le chapitre 3 nous considérons d'abord le cas des accès concurrents à une seule ressource, qui est illustré avec la description d'un compte bancaire, et du service apport d'argent. Dans la partie finale du chapitre 3 nous faisons

la généralisation à un système de plusieurs ressources, illustré avec la description d'un système de comptes : compte de dépôt à vue et compte d'épargne-logement, et du service retrait d'argent.

3. PROPOSITION

3.1. Introduction

Une façon de séparer la description des règles de synchronisation, de la description des processus à synchroniser (selon le principe énoncé au § 2.2.2.), c'est faire le contrôle du parallélisme à travers des routines prologues et épilogues de contrôle des accès.

La classe de propositions qui respectent le principe énoncé ont, à notre avis des propriétés particulièrement intéressantes pour la vérification des programmes, sa lisibilité, la facilité dans leur modification.

Notre proposition se situe donc dans la même classe. Nous adoptons la technique d'utilisation des routines prologues et épilogues de contrôle. Mais nous proposons un nouveau contrôle d'exécution de ces routines, assurant qu'entre deux accès consécutifs à une même ressource, par un processus exécutant un même service, la valeur partageable de la ressource et/ou la valeur de sa variable de contrôle (définie dans 3.2.) ne changent pas, dès que les conditions énoncées dans 3.4.2.B et C sont vérifiées.

D'abord nous précisons quelques idées sur les routines prologues et épilogues de contrôle.

3.2. Prologue et épilogue de contrôle. Variable de contrôle

La classe de techniques de contrôle du parallélisme que nous venons de citer, peut être mise en oeuvre par un contrôleur - entité qui spécifie exactement les règles de synchronisation des accès à la ressource partagée. Les fonctions d'accès à la structure de données d'un contrôleur sont exécutées en exclusion mutuelle par chaque processus à synchroniser (ou sont exécutées par un processus dédié, selon le type de mise en oeuvre).

Ces techniques peuvent être décrites comme suit :

Avant d'exécuter un accès F (cf. figure 2) à une ressource partagée, un processus utilisateur P doit exécuter une routine de contrôle (ou demander son exécution, selon la mise en oeuvre du contrôleur).

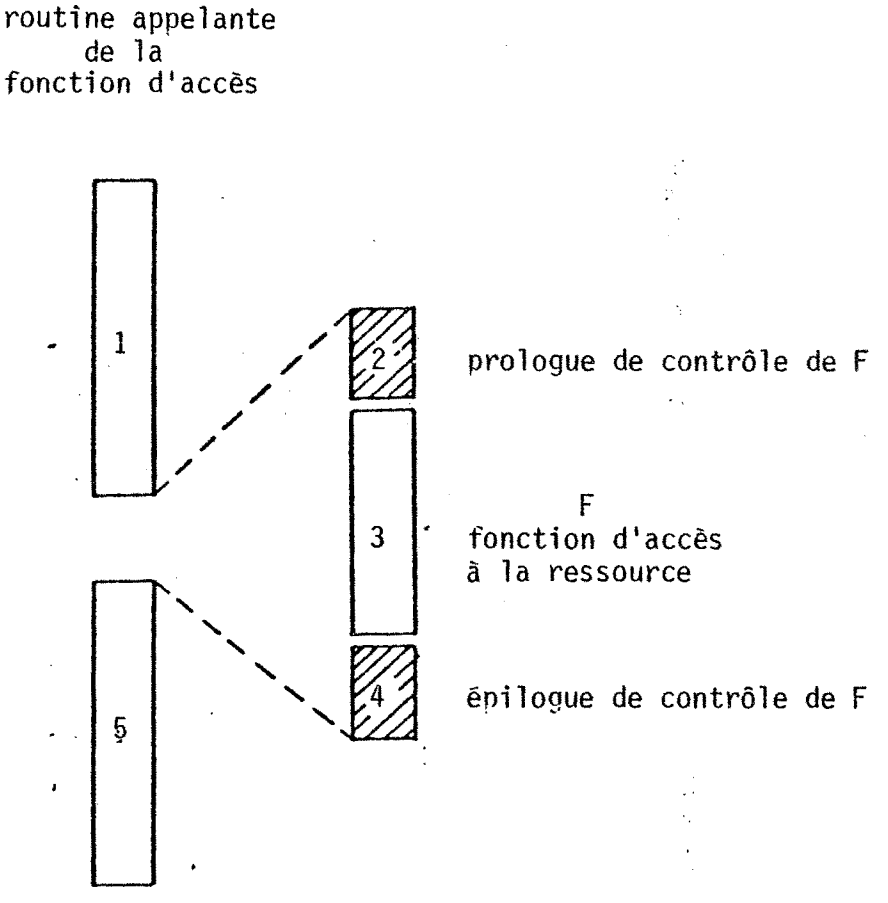


FIGURE 2

Pendant l'exécution de cette routine prologue de contrôle, le processus peut être mis en attente, jusqu'au moment où les contraintes de synchronisation (invariants du problème de synchronisation impliquant un certain nombre d'objets rémanents du contrôleur) sont satisfaites, permettant l'exécution de l'accès.

Ces objets rémanents du contrôleur représentent l'état d'accessibilité de l'objet à gérer. Ils peuvent être vus comme un objet composé - la variable de contrôle.

C'est pour garantir l'intégrité de la variable de contrôle, que les fonctions d'accès à la variable de contrôle - les routines de contrôle - doivent être exécutées en exclusion mutuelle.

L'exécution de l'accès F à l'objet partagé peut obliger à un changement de l'état d'accessibilité de celui-ci. D'où, à la fin de l'exécution d'un accès à un objet partagé, un processus doit exécuter une routine (épilogue) de contrôle (ou demander son exécution), qui aura comme effet la redéfinition de la valeur de la variable de contrôle.

La variable de contrôle d'un contrôleur ne doit être modifiée, qu'à partir des routines de contrôle (prologues et épilogues) de ce contrôleur. En effet :

a) Si la variable de contrôle était modifiable à partir de la routine appelante de F (1 ou 5 sur la figure 2), ceci impliquerait qu'un utilisateur peut changer la politique de gestion des objets partagés, ce qui ne doit pas être permis.

b) Si la variable de contrôle était modifiable à partir de la fonction d'accès (3 sur la figure 2) à l'objet partagé, ceci serait contradictoire avec le principe énoncé. C'est-à-dire, à l'intérieur de la fonction d'accès à l'objet partagé, il y aurait des spécifications des règles de synchronisation qui contrôlent l'accès à l'objet.

La remarque b) inclus le cas où la fonction d'accès à l'objet partagé spécifie des accès à d'autres objets partagés. Dans le cas d'appels imbriqués, les routines de contrôle (prologues et épilogues) de chaque niveau d'imbrication, doivent accéder à des variables de contrôle différentes, pour que le principe énoncé au début ne soit pas violé.

3.3. Description d'une ressource

3.3.1. Introduction

Dans ce paragraphe nous allons décrire l'ensemble de données, et les fonctions d'accès à ces données, qui décrivent la ressource partagée telle qu'elle doit

être vue par un programmeur des services.

Celui-ci peut accéder seulement à une partie de cet ensemble, à partir des fonctions d'accès à la ressource. Même si les programmeurs des services ne peuvent pas accéder directement à la variable de contrôle, ils doivent avoir connaissance des règles de synchronisation des accès à la ressource. Les règles de synchronisation constituent aussi une partie de la description.

Nous décrivons les données et les fonctions d'accès à travers une forme d'expression proche du langage Pascal (*).

Ce choix a été dicté par un souci de clarté, compte tenu du but global de ce texte - décrire un mécanisme de contrôle du parallélisme dans le domaine des systèmes d'exploitation. Il ne doit pas être vu comme une proposition d'outil d'expression à offrir au programmeur du niveau administrateur.

La proposition d'un tel outil est en dehors des buts de ce travail (même si elle fait partie de nos préoccupations actuelles). Ici, nous admettons que le programmeur du niveau administrateur dispose d'un outil pour exprimer la synchronisation entre les accès à la ressource, qui permet la génération automatique des données et fonctions d'accès décrites dans ce paragraphe. Il peut être, par exemple :

a) Identique aux expressions de chemin, même si cette forme d'expression offre peu de souplesse pour la spécification des routines de contrôle.

b) Identique aux compteurs. Cette proposition permet une expression plus fine des routines de contrôle. Le prix est, à notre avis, une lisibilité plus difficile.

3.3.2. La description

Une ressource partagée est décrite par :

i) Les fonctions d'accès à la partie partageable de la ressource, que nous

---

(*) Dans la description algorithmique, ce qui est propre à l'exemple du compte bancaire est écrit en italique ; le reste est écrit en caractères d'imprimerie courant.

appelons fonctions d'accès à la ressource.

ii) Les règles de synchronisation des fonctions d'accès à la ressource.

Pour pouvoir préciser ce que nous entendons par description d'une ressource, nous allons détailler l'exemple de la ressource compte bancaire.

A) Les fonctions d'accès à la ressource

Un compte bancaire est représenté par un nombre réel, auquel nous associons l'identificateur SOLDE, et par les fonctions d'accès décrites par l'illustration 3, page 14.

Le SOLDE constitue la partie partageable de la ressource. Sa valeur est la valeur partageable du compte bancaire (cf. 2.1.).

B) Les règles de synchronisation

L'état d'accessibilité d'une ressource est décrit par la variable de contrôle. Celle-ci peut être modifiée seulement par les routines de contrôle. Ces routines sont exécutées comme prologues ou comme épilogues de contrôle de chaque fonction d'accès à la ressource.

Les règles de synchronisation peuvent être vues comme une relation entre l'ensemble des fonctions d'accès à la ressource et les valeurs de la variable de contrôle de la ressource. Cette relation détermine pour chaque valeur de la variable de contrôle, quelles sont les routines prologue et épilogue de contrôle de chaque fonction d'accès à la ressource. Elle est représentée dans une première étape par un tableau (cf. figure 4a, page 15) qui indique si telle routine de contrôle est exécutable dans tel état courant.

ILLUSTRATION 3

```
§ fonctions d'accès à la ressource compte-bancaire §
type résultat_retrait =
   record
      x : real ;
      y : bool
   end ;
var solde : real ;

function soustraction (retrait : real) : résultat_retrait ;
var z : résultat_retrait ;
§ le résultat d'un retrait comporte le nouveau solde et un booléen §
begin
   if solde < retrait then z.y := false else
   begin
      solde := solde - retrait ;
      z.y := true ;
   end ;
   z.x := solde ;
   soustraction := z ;
end §soustraction§ ;

function addition (apport : real) : real ;
begin
   solde := solde + apport ;
   addition := solde ;
end §addition§ ;

function consultation : real ;
begin
   consultation := solde ;
end §consultation § ;

proc suspendre ;
begin
   § cette procédure a comme seul but d'obliger à l'exécution
     d'une routine épilogue de contrôle qui va changer l'état
     de la ressource §
end §suspendre§

proc libérer ;
begin
   § cette procédure a comme seul but d'obliger à l'exécution
     d'une routine épilogue de contrôle qui va changer l'état
     de la ressource §
end §libérer§
```

fonctions d'accès / valeurs de la variable de contrôle	*suspendre*		*libérer*		*addition*		*soustraction*		*consultation*	
	prologue	*épilogue*	*prologue*	*épilogue*	*prologue*	*épilogue*	*prologue*	*épilogue*	*prologue*	*épilogue*
suspendu			*permis*							
suspension_en_cours		*permis*								
libération_en_cours				*permis*						
NON_suspendu	*permis*				*permis*		*permis*		*permis*	
addition_en_cours						*permis*				
soustraction_en_cours								*permis*		
consultation									*permis*	*permis*

FIGURE 4a

La description d'un compte bancaire, fait au chapitre 2, nous permet de décider que les états d'accessibilité de cette ressource peuvent être :

- suspendu
- non-suspendu
- consultation

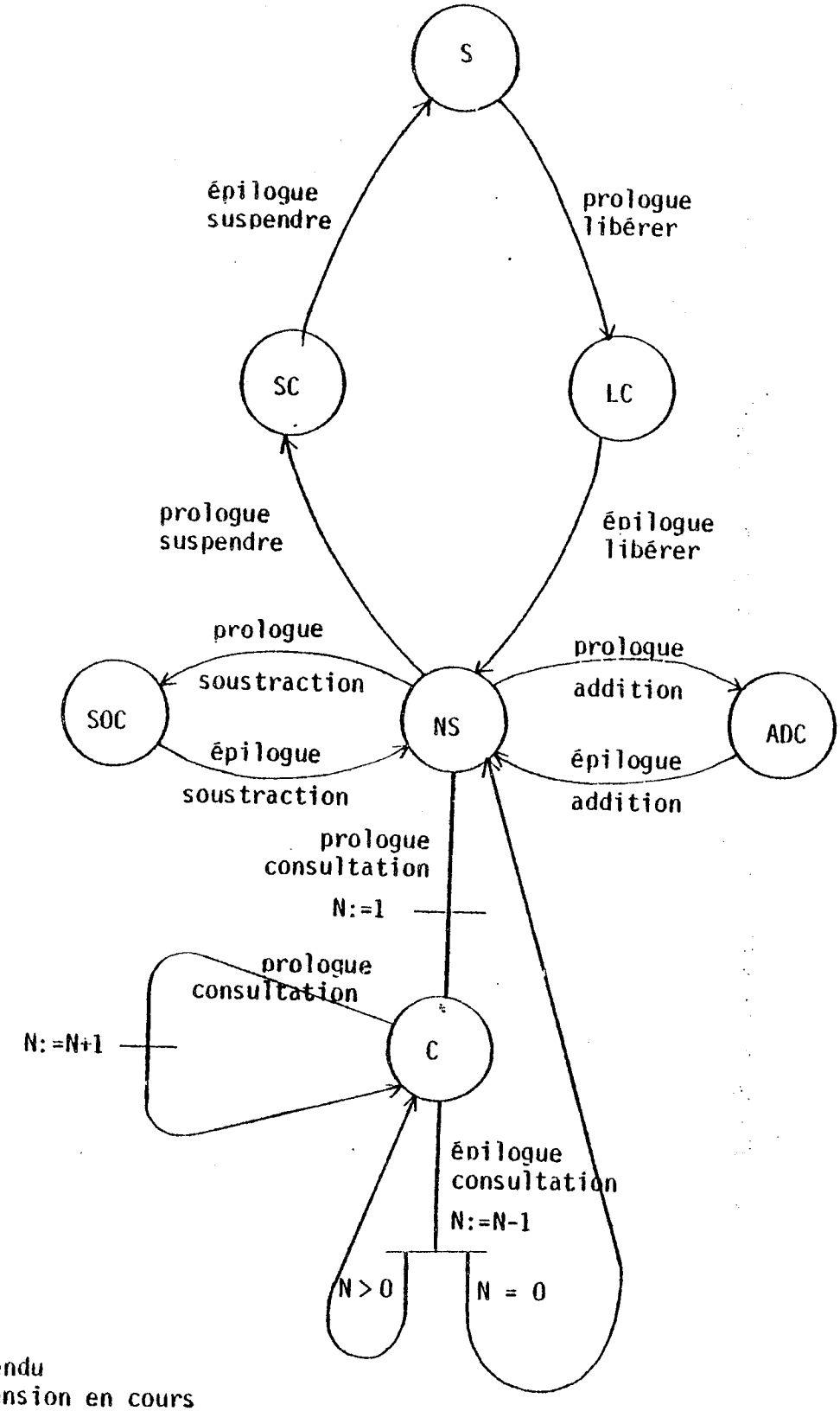
et encore :

- addition en cours
- soustraction en cours
- suspension en cours
- libération en cours

ILLUSTRATION 4b

```
type
   état_accès = (suspendu, suspension_en_cours, non_suspendu, libération_en_cours,
                 addition_en_cours, soustraction_en_cours, état_consultation) ;

   nature_accès = (suspendre, libérer, addition, soustraction, consultation) ;

   nature_prologue = (prologue_suspendre, prologue_libérer, prologue_addition,
                      prologue_soustraction, prologue_consultation, impossible) ;

   nature_épilogue = (épilogue_suspendre, épilogue_libérer, épilogue_addition,
                      épilogue_soustraction, épilogue_consultation, impossible) ;

var table_prologues : array [état_accès, nature_accès] : nature_prologue ;
    table_épilogues : array [état_accès, nature_accès] : nature_épilogue ;

§ initialisation des tableaux table_prologues et table_épilogues §

var i : état_accès ;
    j : nature_accès ;

for i:= suspendu to état_consultation do
for j:= suspendre to consultation do
begin
   table_prologues [i,j] := impossible ;
   table_épilogues [i,j] := impossible
end ;
table_prologues [suspendu, libérer] := prologue_libérer ;
table_prologues [non_suspendu, suspendre] := prologue_suspendre ;
table_prologues [non_suspendu, addition] := prologue_addition ;
table_prologues [non_suspendu, soustraction] := prologue_soustraction ;
table_prologues [non_suspendu, consultation] := prologue_consultation_un ;
table_prologues [état_consultation, consultation] := prologue_consultation_deux ;
table_épilogues [suspension_en_cours, suspendre] := épilogue_suspendre ;
table_épilogues [libération_en_cours, libérer] := épilogue_libérer ;
table_épilogues [addition_en_cours, addition] := épilogue_addition ;
table_épilogues [soustraction_en_cours, soustraction] := épilogue_soustraction ;
table_épilogues [état_consultation, consultation] := épilogue_consultation ;

§ fin initialisation §
```

Le tableau de la figure4a montre pour chaque état quel évènement (initialisation ou terminaison de l'exécution d'une fonction d'accès à la ressource) peut être traité. Pour chaque cas il faut spécifier quel est le traitement correspondant c'-à-dire quelle est la routine de contrôle à exécuter. Cette spécification peut être faite à l'aide d'un formalisme proche des diagrammes de Nutt [NUT 72]. Nous l'avons fait à la figure 5, à laquelle nous ajoutons des commentaires.



Légende :

S - suspendu
SC - suspension en cours
LC - libération en cours
NS - non suspendu
ADC - addition en cours
SOC - soustraction en cours
C - consultation

FIGURE 5a

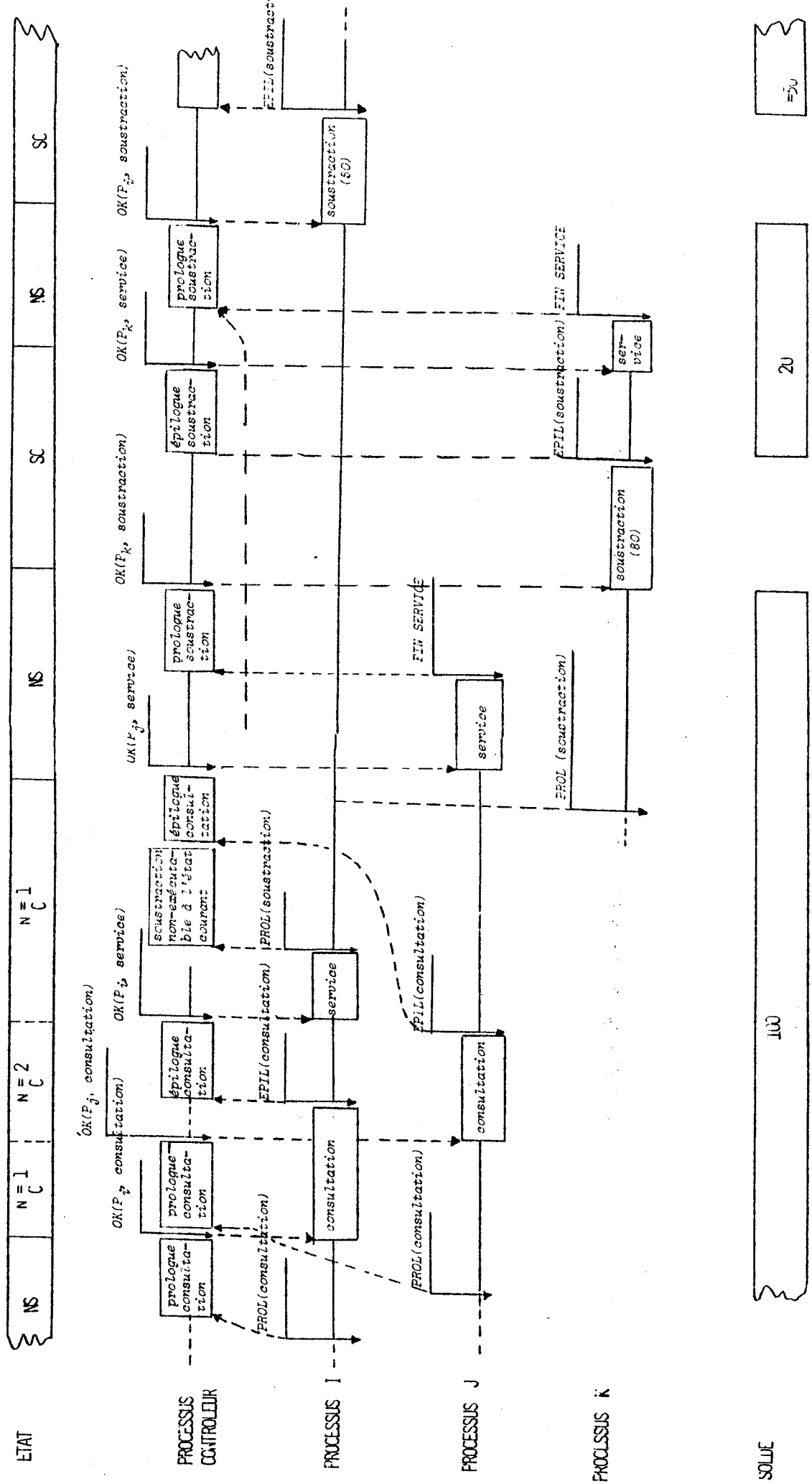

ETAT
NS
$N = 1_C$
$N = 2_C$
$N = 1_C$
NS
SC
NS
SC
PROCESSUS CONTROLEUR
PROCESSUS 1
PROCESSUS J
PROCESSUS K
SOLDE
prologue consultation
prologue consultation
prologue consultation
consultation
consultation
consultation
épilogue consultation
épilogue consultation
soustraction non-exécutable à l'état courant
service
service
service
épilogue consultation
prologue soustraction
prologue soustraction
prologue soustraction
soustraction (50)
soustraction (80)
épilogue soustraction
épilogue soustraction
service
service
$OK(P_2$, consultation)
$OK(P_j$, consultation)
$OK(P_2$, service)
$OK(P_j$, service)
$OK(P_2$, soustraction)
$OK(P_k$, soustraction)
$OK(P_j$, service)
$OK(P_k$, service)
PROL(consultation)
PROL(consultation)
PROL(consultation)
EPIL(consultation)
EPIL(consultation)
EPIL(consultation)
PROL(soustraction)
PROL (soustraction)
EPIL(soustraction)
EPIL(soustraction)
FIN SERVICE
FIN SERVICE
2U
3U
=5U

FIGURE 5b

La variable de contrôle, dans cet exemple, est composée :

i) D'un objet auquel nous associons l'identificateur ETAT-COURANT (variable d'état) et dont les valeurs sont 'suspendu' 'non-suspendu', 'consultation', 'addition en cours', 'soustraction en cours', 'suspension en cours' et 'libération en cours'.

ii) D'un nombre entier auquel nous associons l'identificateur N, et qui sert à contrôler le nombre de processus utilisateurs qui exécutent des fonctions d'accès consultation simultanément.

Dans le formalisme graphique présenté dans la figure 5a :

a) Les noeuds représentent les valeurs de la variable d'état (les états)

b) Les arcs sortant de chaque noeud représentent les évènements qui peuvent être traités quand l'état courant de la ressource correspond à celui du noeud.

c) Le traitement de chaque évènement est représenté par un arc, et constitue une routine de contrôle. Le noeud où l'arc se termine spécifie la nouvelle valeur de la variable d'état. Les modifications des composants de la variable de contrôle, autres que la variable d'état, sont spécifiés à côté d'un trait perpendiculaire à l'arc. Le même évènement peut être traité différemment selon la valeur de l'état courant.

La même description peut être faite algorithmiquement (*). Une forme possible est illustrée, sur le même exemple :

- L'illustration 6 décrit la variable de contrôle, et les routines de contrôle.

(*) Nous n'ajoutons aucune information à celle qui existe déjà sur le graphe. Donc, nous conseillons le lecteur de sauter cette description algorithmique, dans une première lecture.

ILLUSTRATION 6

§ déclaration de la variable de contrôle §

```
§ rappel
type état_accès = (suspendu, non_suspendu, suspension_en_cours, libération_en_cours,
                   addition_en_cours, soustraction_en_cours, état_consultation) ;
fin rappel §

§ déclaration de la variable de contrôle qui comporte l'état courant et un entier n §

var état_courant := état_accès ;
    n : int ;
```

§ spécification de toutes les routines de contrôle §

```
proc e_contrôle_libérer ; § épilogue de contrôle des fonctions d'accès
                            addition, soustraction et libérer §
begin
   état_courant := non_suspendu ;
end §e_contrôle_libérer§ ;

proc e_contrôle_suspendre ; § épilogue de contrôle de la fonction d'accès suspendre §
begin
   état_courant := suspendu ;
end §e_contrôle_suspendre§ ;

proc p_contrôle_addition ; §prologue de contrôle de la fonction d'accès addition §
begin
   état_courant := addition_en_cours ;
end  §p_contrôle_addition§  ;

proc p_contrôle_soustraction ; § prologue de contrôle de la fonction d'accès soustrac
begin                                                                          tion
   état_courant := soustraction_en_cours ;
end §p_contrôle_soustraction§ ;

proc p_contrôle_consultation_un ; § un des prologues de contrôle de la fonction
                                    d'accès consultation §
begin
   n := 1 ;
   état_courant := état_consultation ;
end §p_contrôle_consultation_un§ ;

proc p_contrôle_consultation_deux ; § un des prologues de contrôle de la fonction
                                      d'accès consultation §
begin
   n := n + 1 ;
   état_courant := état_consultation ;
end §p_contrôle_consultation_deux§ ;

proc e_contrôle_consultation ; § épilogue de contrôle de la fonction d'accès consulte
begin                                                                          tior
   n := n - 1 ;
   if n = 0 then état_courant := état_consultation else état_courant := non_bloqué ;
end §e_contrôle_consultation§ ;

proc p_contrôle_suspendre ; § prologue de contrôle de la fonction d'accès suspendre §
begin
   état_courant := suspension_en_cours ;
end §p_contrôle_suspendre§ ;

proc p_contrôle_libérer ; § prologue de contrôle de la fonction d'accès libérer §
begin   état_courant := libération_en_cours ;
end §p_contrôle_libérer§ ;
```

- la figure 7 est un tableau identique à celui de la figure 4a, où sont spécifiés les routines de contrôle qui traitent chaque évènement. Il décrit donc les règles de synchronisation.

fonctions d'accès de la valeurs ressource de la variable de contrôle	*suspendre*		*libérer*		*addition*		*soustraction*		*consultation*	
	prologue	*épilogue*	*prologue*	*épilogue*	*prologue*	*épilogue*	*prologue*	*épilogue*	*prologue*	*épilogue*
suspendu			p_contrôle_libérer							
suspension_en_cours		e_contrôle_suspendu								
libération_en_cours				e_contrôle_libérer						
non_suspendu	p_contrôle_suspendu				p_contrôle_addition		p_contrôle_soustraction		p_contrôle_consultation_un	
addition_en_cours						e_contrôle_libérer				
soustraction_en_cours								e_contrôle_libérer		
consultation									p_contrôle_consultation_deux	e_contrôle_consultation

FIGURE 7

- L'illustration 8 décrit algorithmiquement la procédure qui permet d'exécuter une routine de contrôle à partir de la connaissance du tableau de la figure 7.

ILLUSTRATION 8

```
§ rappel
var table_prologues : array [état_accès,nature_accès] : nature_prologue ;
    table_épilogues : array [état_accès,nature_accès] : nature_épilogue ;
fin rappel §

type t_nature_évènement = (prologue,épilogue) ;

type nom_routine =
   record
      g : t_nature_évènement ; § informations nécéssaires pour accéder à §
      i : état_accès ;         § l'élément (i,j) de la table_prologues   §
      j : nature_accès         § ou de la table_épilogues                §
   end ;


proc exécution_de_la_routine_de_contrôle (routine_de_contrôle : nom_routine) ;
begin
   if routine_de_contrôle.g = prologue then
      case table_prologues [i,j] of
         prologue_libérer : §call§ p_contrôle_libérer ;
         prologue_suspendre : §call§ p_contrôle_suspendre ;
         prologue_addition : §call§ p_contrôle_addition ;
         prologue_soustraction := §call§ p_contrôle_soustraction ;
         prologue_consultation_un := §call§ p_contrôle_consultation_un ;
         prologue_consultation_deux := §call§ p_contrôle_consultation_deux
      end §case§ ;
   else
      § routine_de_contrôle.g = épilogue §
      case table_épilogues [i,j] of
        épilogue_suspendre : §call§ e_contrôle_suspendre ;
        épilogue_libérer : §call§ e_contrôle_libérer ;
        épilogue_addition : §call§ e_contrôle_libérer ;
        épilogue_soustraction : §call§ e_contrôle_libérer ;
        épilogue_consultation : §call§ e_contrôle_consultation
      end §case§ ;
end §exécution_de_la_routine_de_contrôle§ ;
```

3.4. Accès concurrents à une ressource partagée

3.4.1. Introduction

Dans ce paragraphe nous allons proposer une technique de contrôle d'exécution des fonctions d'accès à la ressource et des routines de contrôle que décrivent une ressource partagée, présentées dans le paragraphe antérieur.

Ce contrôle d'exécution doit assurer que les règles de synchronisation ne sont pas violés, et que si les règles énoncées dans 3.4.2-B et C sont respectées, la valeur de la variable de contrôle de la ressource et/ou sa valeur partageable ne changent pas (restent invariantes) entre deux accès d'un même utilisateur.

Dans 3.4.3. nous décrivons fonctionnellement une mise-en-oeuvre de notre proposition.

L'ensemble des données et des routines présentées dans 3.4.3., et l'ensemble des données, et les fonctions d'accès à ces données, qui décrivent une ressource partagée, présentées dans 3.3.2., constituent une représentation de la ressource, étudiée dans 3.6.

Dans le sous-chapitre 3.5., nous faisons la généralisation pour un système de ressources, de la proposition faite dans 3.4. concernant des accès concurrents à une seule ressource.

3.4.2. La technique de contrôle du parallélisme

Dans la partie A de ce paragraphe, nous allons décrire la technique de contrôle du parallélisme que nous voulons proposer.

Dans le § 2.2.2., nous avons exigé que, associé à la technique de contrôle du parallélisme, un ensemble de règles définissent des conditions (au moins suffisantes), pour que l'état de la ressource ou sa valeur ne soient pas modifiés entre deux accès appelés consécutivement à partir de l'exécution d'un service. Ces règles sont établies dans les parties B et C de ce paragraphe.

A) DESCRIPTION

1 - Toutes les routines prologues et épilogues de contrôle de toutes les fonctions d'accès à une ressource partagée, sont exécutées par un seul processus (le processus contrôleur de la ressource), à la demande des processus utilisateurs*.

2 - Après le dépôt d'une demande d'exécution d'une routine de contrôle (prologue ou épilogue), le processus utilisateur attend d'être réveillé par le processus contrôleur.

3 - Ce réveil a lieu quand le processus contrôleur termine l'exécution du contrôle demandé. Pour que le contrôle demandé soit exécuté, il est nécessaire que l'état courant de la ressource le permette, selon les spécifications des règles de synchronisation.

4 - A la fin de l'exécution d'une routine épilogue de contrôle, le processus contrôleur réveille le processus demandeur P, et se met lui-même en attente d'être réveillé par le même processus P. Le réveil aura lieu quand le processus P demande à exécuter un autre accès à la ressource, ou quand il termine l'exécution du service.

5 - Dans les conditions décrites dans 4, quand le processus utilisateur P demande à exécuter un autre accès à la ressource, il réveille le processus contrôleur. Celui-ci vérifie d'abord si l'accès demandé est exécutable selon l'état courant de la ressource, et :

   a) si l'accès est exécutable, il va exécuter la routine prologue de contrôle correspondante ;

   b) sinon, selon l'état courant de la ressource, le processus contrôleur peut traiter une demande d'un autre processus dont la routine de contrôle est exécutable. L'accès demandé par P reste "stocké", et sera autorisé quand l'état courant de la ressource sera "convenable", selon les spécifications des règles de synchronisation.

(*) L'alinéa 1 assure que les fonctions d'accès à la variable de contrôle (les routines de contrôle) ne sont pas exécutées en parallèle, en accord avec ce que nous avons exigé au paragraphe 3.2.

6 - A la fin de l'exécution d'une routine prologue de contrôle, le processus contrôleur réveille le processus demandeur, qui est ainsi autorisé à exécuter l'accès demandé à la ressource. Simultanément le processus contrôleur peut traiter une demande d'un autre processus dont la routine de contrôle (prologue ou épilogue) est exécutable selon l'état courant de la ressource. Les accès parallèles à une ressource sont ainsi autorisés, contrôlés à travers l'évolution de la valeur de la variable de contrôle de la ressource (*).

B - ETAT CONSTANT

Soient A_i et A_j deux fonctions d'accès appelées consécutivement à partir de l'exécution d'un service par un processus P (cf. figure 9). Si entre la fin de l'exécution de l'épilogue de contrôle de A_i et le début de l'exécution de la fonction d'accès A_j, aucune demande d'exécution d'une autre routine de contrôle, faite par un processus différent de P, n'est exécutée, alors il est évident que l'état de la ressource sera invariant entre les exécutions des accès A_i et A_j.

L'alinéa 4 de la description assure que entre la fin de l'exécution de l'épilogue de contrôle de A_i et la prochaine demande d'exécution d'un accès (A_j), faite par le processus P, aucune routine de contrôle sera exécutée, puisque le processus contrôleur sera dans l'état d'attente.

Si l'accès demandé (A_j) est exécutable selon l'état courant de la ressource, alors les alinéas 2, 3 et 5a de la description assurent qu'aucune routine de contrôle demandée par un processus différent de P, sera exécutée, jusqu'à l'exécution de l'accès A_j.

Sinon, (si l'accès demandé (A_j) n'est pas exécutable selon l'état courant de la ressource) l'alinéa 5b de la description autorise l'exécution d'une routine de contrôle, demandée par un processus différent de P.

(*) L'alinéa 6 rend potentiellement possibles les accès parallèles à une ressource partagée.

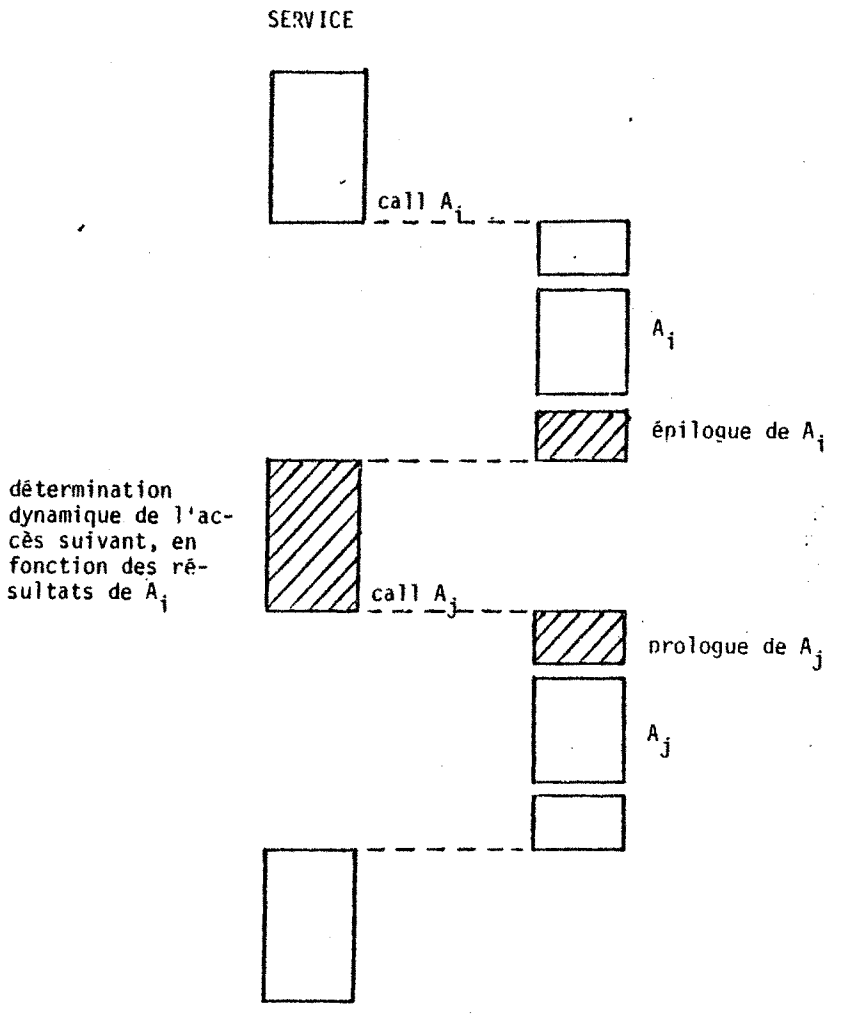


FIGURE 9

Nous pouvons donc énoncer la règle suivante :

REGLE I :

Si A_i et A_j sont deux accès à une ressource, qui peuvent être exécutés consécutivement par un processus P exécutant un service, alors :
pour que l'état d'accessibilité (valeur de la variable de contrôle) de la ressource ne change pas entre les deux accès, (due à l'intervention d'un processus différent de P), il est suffisant que :

le prologue de contrôle de A_j soit exécutable dans tous les états auxquels peut conduire l'exécution d'un épilogue de contrôle de A_i.

Cette règle est vérifiable par inspection du graphe de synchronisation de la ressource.

Sans faire des hypothèses sur la nature des accès A_i et A_j, nous ne sommes pas arrivé à interpréter le fait que les deux accès satisfont ou non la Règle I.(*)

C - VALEUR PARTAGEABLE CONSTANTE

Si entre la fin de l'exécution de l'accès A_i et le début de l'exécution de l'accès A_j par un processus P, aucun autre processus accède à la ressource, alors :

il est évident que la valeur partageable de la ressource sera invariante entre les exécutions des accès A_i et A_j.

L'alinéa 6 de la description rend potentiellement possibles les accès parallèles à la ressource. Il faut donc assurer que quand se termine l'exécution de l'accès A_i, nul autre processus n'exécute déjà une fonction d'accès. En particulier, il faut assurer que pendant l'exécution de l'accès A_i nul autre processus ne sera autorisé à exécuter un accès - l'épilogue de contrôle de A_i doit être la seule routine de contrôle exécutable selon l'état courant de la ressource.

D'autre part, il est évident que si les accès A_i et A_j ne satisfont pas la Règle I, alors entre les deux accès, d'autres peuvent être exécutées.

Nous pouvons donc énoncer la règle suivante :

(*) Dans l'article [ROS 78] (cf. Annexe B-III) est étudié le problème du contrôle de concurrence des processus qui partagent l'accès à une base de données répartie. Les auteurs considèrent que tous les objets (entités) sont accessibles seulement en lecture et en écriture, par conséquence les règles de synchronisation des accès sont bien définies et immuables. Pour ce scénario est défini un critère pour que deux accès (requêtes) soient "en conflit". Sous ces hypothèses, le fait que deux accès ne satisfont la Règle I est suffisant pour qu'elles soient "en conflit".

REGLE II :

Si A_i et A_j sont deux accès à une ressource R, qui peuvent être exécutés consécutivement par un processus P exécutant un même service, alors

pour que la valeur partageable de la ressource ne change pas entre les deux accès, il est suffisant que :

a) Au moment où s'initie l'exécution de la fonction d'accès A_i, aucun autre processus n'exécute une fonction d'accès.

b) Dans tous les états auxquels peut conduire l'exécution d'un prologue de contrôle de A_i, seulement est exécutable l'épilogue de contrôle de A_i.

c) La Règle I soit satisfaite.

Les alinéas b et c de cette règle sont vérifiables par inspection du graphe de synchronisation de la ressource.

Voyons un exemple de programmation d'un service où les conditions énoncées dans la Règle II ne sont pas satisfaites. Dans cet exemple la sémantique du service exige que la valeur partageable de la ressource ne change pas, d'où les conséquences peuvent être désastreuses, comme nous allons le voir.

Prenons l'exemple du compte bancaire et considérons le service apport d'argent (cf. page 4). Ce service pourrait être (mal) programmé, de telle façon que la séquence d'accès définie, serait :

1. Addition (d'une quantité d'argent X) ;
2. Consultation (donne comme résultat le solde du compte) ;

Si le solde est supérieur à Y, alors

3. Soustraction (d'une quantité d'argent Z, pour acheter de l'or) ;

Il est indispensable qu'entre l'accès consultation et l'accès soustraction, la valeur partageable de la ressource (le solde) ne change pas. Mais, si le graphe de synchronisation du compte bancaire est celui représenté dans la figure 5a, alors l'alinéa b de la Règle II n'est pas satisfaite par rapport à la séquence des accès consultation-soustraction.

En effet, (cf. figure 5a) l'exécution d'une routine prologue de contrôle d'un accès consultation conduit toujours à l'état "consultation", et à cet état est exécutable non seulement l'épilogue de contrôle de l'accès "consultation", mais aussi son prologue de contrôle.

La figure 5b décrit une situation où, dû au fait que nous venons de signaler :

- l'exécution du service apport d'argent, par un processus I (avec Z = 80);
- l'exécution, par un processus J, d'un service qui se termine par un accès consultation ;
- l'exécution, par un processus K, d'un service qui se termine par un accès soustraction ;

conduit à un solde débiteur.

Dans la situation décrite pour cette figure, le fait que les processus I et J exécutent l'accès consultation en parallèle, et que l'épilogue de contrôle correspondant est exécuté d'abord pour le processus I, fait que :

Quand le processus I demande à exécuter l'accès soustraction, le prologue de contrôle correspondant ne soit pas exécutable (l'état courant du compte bancaire est "consultation"). Dans ces conditions, si l'accès demandé par le processus I reste "stocké" (cf. page 24), le processus K peut être autorisé à exécuter l'accès soustraction avant le processus I, ce qui peut conduire à un solde débiteur.

Des situations comme celle que nous venons de décrire ont été prévues dans la spécification des fonctions d'accès à la ressource compte bancaire (cf. illustration 3). Les accès addition et soustraction ont été déclarés comme des fonctions, qui retournent à la routine appelant la valeur actualisée du solde.

Le service apport d'argent pourrait être (bien) programmé, de telle façon que la séquence d'accès définie, serait :

1. Addition (d'une quantité d'argent X), cette fonction retourne la nouvelle valeur du solde ;

Si le solde est supérieur à Y, alors

2. Soustraction (d'une quantité d'argent Z, pour acheter de l'or) ;

Les accès addition et soustraction satisfont la Règle II, et donc le solde ne change pas entre les accès.

La vérification statique du fait que si les conditions énoncées (Règle I et Règle II) sont satisfaites, dépend du langage d'expression des services et du langage de description des ressources.

Dans les cas où cette facilité ne peut pas être offerte, la vérification peut être faite par le programmeur des services. L'utilisation du formalisme graphique qui constitue le graphe de synchronisation (cf. figure 5), facilite cette tâche, puisqu'il permet une visualisation de la succession des états par rapport à l'exécution des routines de contrôle, et des routines qui sont exécutables à chaque état.

3.4.3. Une mise en oeuvre de la technique de contrôle du parallélisme (description fonctionnelle)

A) Hypothèse de départ

Nous décrivons maintenant un ensemble de données et de routines suffisant pour une mise en oeuvre de notre proposition.

Nous décrivons aussi un mécanisme de communication entre un processus contrôleur d'une ressource partagée et les processus utilisateurs. Nous utilisons le sémaphore avec message [CROCUS pages 44 et45] et l'exclusion mutuelle, respectivement comme outil et comme concepts élémentaires d'expression des interactions entre processus. Nous aurions pu utiliser d'autres outils d'expression (moniteurs [HOA 74], régions critiques conditionnelles [BRI 72(b)] ,...) ou d'autres concepts (ordonnancement de messages [BAN 79], ...). Nous avons voulu un outil de "très bas niveau" et un concept "établi".

B) Le service

Un service est l'expression algorithmique d'une séquence d'accès à des ressources partagées, dont la sémantique est telle qu'entre deux accès consécutifs à une même ressource, l'état d'accessibilité et/ou la valeur partageable de celle-ci peuvent être des invariants.

L'exécution d'une routine déclarée comme étant un service, est toujours précédée par l'exécution d'un prologue de service et suivie par l'exécution d'un épilogue de service.

Nous avons déjà dit (cf. 3.4.2.A.4), qu'à la fin de l'exécution d'une routine épilogue de contrôle, le processus contrôleur d'une ressource réveille le processus demandeur, et se met lui même en attente. Il sera réveillé par le même processus demandeur quand ce processus :

1ère situation : demande à exécuter le prochain accès à la ressource,

ou, si le processus ne demande aucun autre accès :

2ème situation : quand il termine l'exécution du service.

Prologue d'un service :

La 1ère situation équivaut à dire que le processus contrôleur sera systématiquement réveillé par un processus utilisateur demandant à exécuter un accès à la ressource, sauf si c'est la première demande de ce type pendant l'exécution du service. Ce réveil est exécuté par un prologue d'accès à la ressource (cf. 3.4.3.C).

Si le langage d'expression des services par le programmeur est tel qu'il permet de déterminer statiquement l'accès qui sera exécuté le premier, alors à cet accès peut être associé statiquement un prologue où le réveil du processus contrôleur ne sera pas spécifié.

Le cas échéant, il faut contrôler dynamiquement l'exécution de ce réveil. Une façon de le faire consiste à définir un booléen local à l'exécution du service (auquel nous associons l'identificateur REVEILLER-CONTROLEUR), qui est initialisé avec la valeur faux par le prologue-du-service, et est systématiquement affecté de la valeur vrai par l'épilogue de chaque accès à la ressource (cf. 3.4.3.C). De cette façon sa valeur sera faux seulement lors du premier accès à la ressource.

La routine prologue-service est décrite par l'illustration 10.

ILLUSTRATION 10

```
var réveiller_contrôleur : bool ;

proc prologue_service ;
begin
   réveiller_contrôleur := false ;
end §prologue_service§ ;
```

Epilogue d'un service

La deuxième situation (cf. page 31) exige que le processus contrôleur soit réveillé par un processus utilisateur, quand celui-ci termine l'exécution d'un service où il a au moins exécuté un accès à la ressource. Ce réveil est spécifié dans la routine épilogue du service.

Le booléen REVILLER-CONTROLEUR peut être utilisé dans cette routine épilogue, pour déterminer si le réveil du processus contrôleur doit être exécuté.

La routine épilogue-service est décrite par l'illustration 11.

ILLUSTRATION 11

```
var sémaphore_contrôleur : sémaphore_à_message ;

proc épilogue_service ;
begin
   if réveiller_contrôleur = true then
      v (sémaphore_contrôleur, 'c est une fin service') ;
end §épilogue_service§ ;
```

Ces deux routines (prologue-service et épilogue-service) sont les mêmes pour tous les services de toutes les ressources. Elles peuvent être des routines standard, cataloguées dans les bibliothèques du système, et sont exécutées systématiquement sans que le programmeur ait à le spécifier.

C) L'accès à une ressource

L'exécution d'une routine déclarée comme étant une fonction d'accès à une ressource partagée, sera toujours précédée par l'exécution d'un prologue et suivie par l'exécution d'un épilogue :

Prologue d'un accès

C'est à travers l'exécution de la routine prologue de toute fonction d'accès à une ressource partagée, que les processus utilisateurs soumettent au processus contrôleur la demande d'exécution de l'accès à la ressource.

Pour ce faire nous admettons l'existence d'une structure de données associée au processus contrôleur (que nous désignons par file-des-demandes). Nous détaillons cette structure et ses fonctions d'accès, au paragraphe 3.4.3.E.

Une demande est constituée des informations suivantes :

i) Nom du sémaphore associé au processus utilisateur (*).
   Le processus contrôleur exécute la primitive V sur ce sémaphore, quand il va autoriser le processus utilisateur à exécuter l'accès demandé.
ii) Identificateur de l'accès demandé.
iii) La valeur 'prologue', qui signale au processus contrôleur que le processus utilisateur veut être autorisé à commencer l'exécution de l'accès.

(*) Le nom du sémaphore associé au processus utilisateur sert aussi, par commodité, à identifier le processus utilisateur.

L'illustration 12 montre l'algorithme de la routine prologue-accès.

ILLUSTRATION 12

```
type nature_demande
   record
      sema_utilisateur : sémaphore ;
      accès            : nature_accès ;
      genre            : t_nature_évènement
   end ;

proc prologue_accès (sémaphore_utilisateur : sémaphore,
                     accès                 : nature_accès,
                     nature_évènement      : t_nature_évènement) ;
   § le processus utilisateur passe ces paramètres à la procédure prologue_accès
     exemple : (sem uti,addition,prologue) §
var demande : nature_demande ;
begin
   demande.sema_utilisateur := sémaphore_utilisateur ;
   demande.accès            := accès ;
   demande.genre            := nature_évènement ;
   §call§ déposer_demande (demande) ; § cette procédure est non bloquante §
   if réveiller_contrôleur = true then
      v (sémaphore_contrôleur, 'c est un accès') ;
   else
      réveiller_contrôleur := true ;
   p (sémaphore_utilisateur) ; §attente d'être réveiller par le processus contrôleur
end §prologue_accès§ ;
```

Le processus contrôleur examine périodiquement la file-des-demandes et selon un mécanisme de sélection décrit au paragraphe 3.4.3.E, va élire le processus utilisateur. Ensuite, il va exécuter la routine prologue de contrôle correspondante à cette sélection, qui a pour but de redéfinir la valeur de la variable de contrôle. A ce point le processus contrôleur exécute la primitive V sur le sémaphore du processus élu, pour l'autoriser à exécuter l'accès à la ressource.

Epilogue d'un accès

C'est à travers l'exécution de la routine épilogue de toute fonction d'accès à une ressource partagée, que les processus utilisateurs soumettent au processus contrôleur un message signalant la fin de l'exécution de l'accès à la ressource, et demandant l'exécution d'une routine épilogue de contrôle.

La demande est identique à celle du prologue d'accès. Elle est constituée des informations suivantes :

i) Le nom du sémaphore associé au processus utilisateur
ii) L'identificateur de l'accès demandé
iii) La valeur 'épilogue'.

Quand un processus utilisateur termine l'exécution d'une fonction d'accès à une ressource, il dépose la demande décrite dans la file-des-demandes du processus contrôleur de la ressource et se met en attente d'être réveillé par celui-ci.

Quand cette demande du processus utilisateur est élue par le processus contrôleur, celui-ci va exécuter une routine épilogue de contrôle (qui a comme but la redéfinition de la valeur de la variable de contrôle), suivi par l'exécution de la primitive V sur le sémaphore du processus utilisateur, pour l'autoriser à continuer l'exécution du service. Le processus contrôleur se met en attente, pour assurer que la valeur de la variable de contrôle et la valeur partageable de la ressource ne changent pas, jusqu'au prochain prologue-accès du même utilisateur.

Le séquencement des exécutions des routines prologue et épilogue d'accès, et prologue et épilogue de contrôle est mis en évidence par la figure 13.

L'illustration 14 montre l'algorithme de la routine épilogue-accès.

D) Le contrôleur

Comme nous avons déjà décrit dans le paragraphe précédent, le processus contrôleur d'une ressource partagée, examine périodiquement la file-des-demandes pour sélectionner les demandes faites par les processus utilisateurs pendant l'exécution des routines prologue-accès et épilogue-accès. Ces demandes sont sélectionnées et traitées une par une, et pour qu'une demande soit sélectionnée il est nécessaire qu'elle soit traitable selon l'état courant de la ressource, selon les spécifications des règles de synchronisation de la ressource.

La structure des données file-des-demandes est accessible au processus contrôleur à travers les fonctions d'accès "retirer-demande" et "choisir-demande".

La fonction "retirer-demande" sélectionne une demande déposée par un processus utilisateur sur la file-des-demandes, enlève la demande de cette structure (pour qu'elle ne soit pas traitée une deuxième fois), et délivre comme résultats :

- Le nom du sémaphore du processus utilisateur qui a fait la demande.

- Le nom de la routine de contrôle à exécuter, avant d'autoriser le processus utilisateur à poursuivre son exécution.

- La valeur 'prologue' ou 'épilogue', selon que la routine de contrôle est un prologue ou un épilogue.

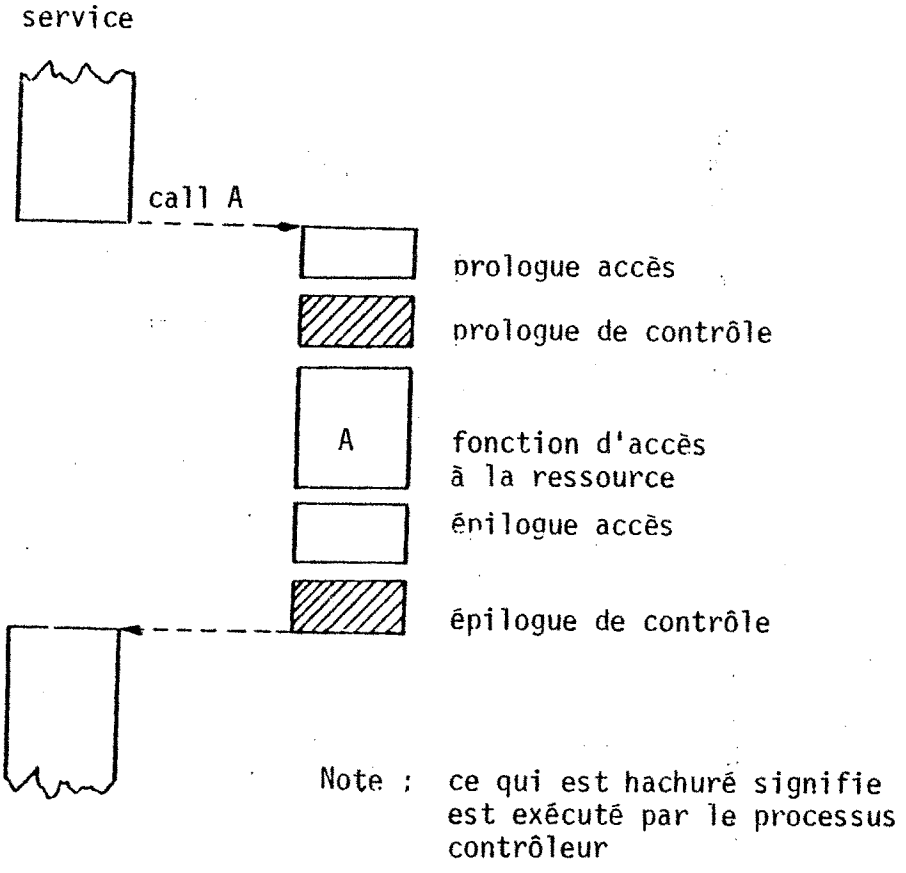


FIGURE 13

ILLUSTRATION 14

```
proc épilogue_accès (sémaphore_utilisateur : sémaphore,
                     accès                 : nature_accès,
                     nature_évènement      : t_nature_évènement) ;
   § le processus utilisateur passe ces paramètres à la procédure épilogue_accès
     exemple : (sem_uti,soustraction,épilogue) §
var demande : nature_demande ;
begin
   demande.séma_utilisateur := sémaphore_utilisateur ;
   demande.accès            := accès ;
   demande.genre            := nature_évènement ;
   §call§ déposer_demande (demande) ;
   p (sémaphore_utilisateur) ; §attente d'être réveiller par le processus contrôleur
end §épilogue_accès§ ;
```

La fonction "choisir-demande" sélectionne une demande déposée par un processus utilisateur (identifié par un paramètre : nom du sémaphore associé au processus)sur la file-des-demandes, enlève la demande de cette structure de données, et délivre comme résultats :

- La valeur 'exécutable' ou 'non-exécutable', selon qu'il y avait ou non une demande du processus spécifié, traitable à l'état courant de la ressource.
- Le nom de la routine de contrôle à exécuter, avant d'autoriser le processus utilisateur à poursuivre son exécution.

Une fois que la demande est sélectionnée, le processus contrôleur exécute la routine de contrôle correspondante et réveille le processus demandeur. Ce processus est de cette façon autorisé à exécuter l'accès à la ressource ou à continuer l'exécution du service, selon que la demande concerne un prologue ou un épilogue de contrôle.

Nous détaillons ce que le processus contrôleur fait dans les deux cas, avant de sélectionner une autre demande :

Après un épilogue de contrôle

Nous avons déjà dit (cf. 3.4.2.A.4) qu'à la fin de l'exécution d'une routine épilogue de contrôle, le processus contrôleur d'une ressource réveille le processus demandeur, et se met lui même en attente. Il sera réveillé par le même processus demandeur quand ce processus :

i - Demande à exécuter le prochain accès à la ressource.

Dans ce cas le processus contrôleur doit traiter cette demande en priorité pour que l'état de la ressource et/ou sa valeur partageable ne soient pas modifiés. Cependant, si la demande n'est pas traitable à l'état courant de la ressource, le processus contrôleur va sélectionner une autre demande. De cette façon la ressource partagée ne reste pas bloquée indéfiniment par un de ses utilisateurs.

ii - Quand le processus demandeur termine l'exécution du service.

Dans ce cas, le processus contrôleur va sélectionner une autre demande et va la traiter.

Après un prologue de contrôle

A la fin de l'exécution d'une routine prologue de contrôle, le processus contrôleur autorise le processus demandeur P à exécuter la fonction d'accès (F) à la ressource, sélectionne une autre demande et fait son traitement. Le fait que la

fonction F puisse être exécutée par plusieurs processus utilisateurs simultanément, ou par un seul processus est pris en compte par les règles de synchronisation.

La routine exécutée cycliquement par le processus contrôleur est décrite par l'illustration 15.

E) La communication entre le processus contrôleur et les processus utilisateurs

Le processus contrôleur d'une ressource, a comme fonction de contrôler l'exécution des accès concurrents des processus utilisateurs à la ressource et de régler les éventuels conflits.

Le mécanisme de communication entre le processus contrôleur et les processus utilisateurs est composé de :

a) Un sémaphore associé à chaque processus. Il est tel que :

i - La primitive P peut être exécutée seulement par son "propriétaire".

ii - La primitive V sur le sémaphore d'un processus contrôleur peut être exécutée par un processus utilisateur, et réciproquement la primitive V sur le sémaphore d'un processus utilisateur peut être exécutée par un processus contrôleur.

b) Une structure de données file-des-demandes dont les fonctions d'accès sont :

DEPOSER-DEMANDE exécutée par les processus utilisateurs
RETIRER-DEMANDE et CHOISIR-DEMANDE exécutées par le processus contrôleur.

Quand un processus utilisateur candidate à l'exécution d'un accès à une ressource, l'état de celle-ci est imprévisible. C'est le processus contrôleur qui doit "décider" quand le processus utilisateur sera autorisé à exécuter l'accès prétendu. La structure de données file-des-demandes permet au processus utilisateur de "déposer" sa demande, avant de se mettre en attente d'être réveillé pour exécuter l'accès demandé. L'illustration 16 montre une description algorithmique de la structure file-des-demandes, et de ses fonctions d'accès.

Le processus contrôleur exécute régulièrement la fonction RETIRER-DEMANDE. Quand il n'y a aucune demande exécutable à l'état courant de la ressource, une solution possible est d'obliger le processus contrôleur à faire une attente active. C'est-à-dire, qu'il doit continuer à exécuter régulièrement cette fonction jusqu'à ce qu'un processus utilisateur y dépose une demande exécutable à l'état courant.

ILLUSTRATION 15

```
§ rappel
type t_nature_évènement = (prologue,épilogue) ;
type t_nature_exécution = (exécutable, non_exécutable) ;
type nom_routine =
   record
      g : t_nature_évènement ;
      i : état_accès ;          §§ indice de ligne et indice de colonne          §§
      j : nature_accès ;        §§ dans la lecture du tableau table_prologues ou §§
   end ;                        §§ table_épilogues §§
type sémaphore = ... ;
type sémaphore_à_message = ... ; § le message est de type chaîne de caractères §
type chaîne_de_caractères = ... ;
fin rappel §


§ algorithme exécuté par le processus contrôleur §

var état_courant : état_accès ;
    n : int ;

proc contrôleur ;
var
   dédié_à_un_utilisateur : bool ;
   sémaphore_utilisateur  : sémaphore ;
   routine_de_contrôle    : nom_routine ;
   message_réveil         : chaîne_de_caractères ;
   sémaphore_contrôleur   : sémaphore_à_message ;
   nature_exécution       : t_nature_exécution ;
begin
   état_courant :=non_suspendu; § initialisation de état_courant et de n, liée §
   n := 0 ;                     § au problème des comptes bancaires            §
   repeat
      §call§ retirer_demande (sémaphore_utilisateur, routine_de_contrôle) ;
            § retirer_demande peut se mettre en attente ; au retour de retirer_
              demande, sémaphore_utilisateur contient le sémaphore privé de
              l'utilisateur sur lequel il a fait p et routine_de_contrôle contient
              les informations nécéssaires pour exécuter la routine de contrôle
              correspondantes ; la demande retirée est exécutable à l'état courant
              de la ressource §
      dédié_à_un_utilisateur := true ;
      while dédié_à_un_utilisateur do
      begin
         §call§ exécution_de_la_routine_de_contrôle (routine_de_contrôle) ;
         v (sémaphore_utilisateur) ;  § réveiller l'utilisateur §
         if routine_de_contrôle.g = prologue then dédié_à_un_utilisateur := false
         else begin
                 p (sémaphore_contrôleur, message_réveil) ; § le contrôleur se met
                 en attente ; quand il sera réveillé, le message sera contenu dans
                 message_réveil §
                 if message_réveil = 'c est une fin service' then
                    dédié_à_un_utilisateur := false
                 else begin
                    §call§ choisir_demande (sémaphore_utilisateur, routine_de_contrôle,
                                            nature_exécution) ;
                    if nature_exécution = non_exécutable then
                       dédié_à_un_utilisateur := false ;
                    end ; § if message_réveil = .. §
              end ; § if routine_de_contrôle.g = .. §
      end ; § while §
   until for_ever ;
end §contrôleur§ ;
```

Pour éviter l'attente active, nous avons compliqué un peut la description faite dans l'illustration 16, de la structure file-des-demandes. Nous avons ajouté un booléen (CONTROLEUR-EN-ATTENTE) et un sémaphore (ATTENTE) qui permettent :

1) Au processus contrôleur de se mettre en attente (dans la fonction RETIRER-DEMANDE), quand il n'y a aucune demande exécutable à l'état courant, et

2) Aux processus utilisateurs de réveiller (dans la routine DEPOSER-DEMANDE) le processus contrôleur, si celui-ci est en attente.

Nous avons déjà affirmé qu'il est nécessaire qu'une demande soit traitable selon l'état courant de la ressource (*), pour qu'elle puisse être sélectionnée par le processus contrôleur. Cette sélection est décrite par la routine RETIRER-DEMANDE, en particulier par la routine auxiliaire TESTER. Etant donnée que notre proposition énonce seulement une condition nécessaire pour qu'une demande soit sélectionnée, en réalité il peut avoir plusieurs demandes sélectionnables. Une politique de priorités peut être donc mise-en-place par la routine TESTER. Dans le cas de l'illustration 16, la routine TESTER se résume à sélectionner la demande exécutable d'indice le plus petit dans la file-des-demandes.

Une politique de priorités peut, par exemple, être définie à partir du graphe de synchronisation (cf. figure 5). Ce formalisme explicite quels sont les évènements (prologues et épilogues des fonctions d'accès) traitables à chaque état de la ressource. Alors ces évènements peuvent être ordonnés, pour chaque état, par ordre de priorité de traitement. Par exemple, si nous considérons le graphe de synchronisation d'un compte bancaire (cf. figure 5), nous pouvons exiger que à l'état NON-SUSPENDU soient traités avec des priorités décroissantes les événements :

PROLOGUE- SUSPENDRE
PROLOGUE-ADDITION
PROLOGUE-SOUSTRACTION
PROLOGUE-CONSULTATION

(*) Selon les spécifications des règles de synchronisation.

ILLUSTRATION 16

```
§ rappel
type nature_accès = (suspendre, libérer, addition, soustraction, consultation) ;

type état_accès = (suspendu, suspension_en_cours, non_suspendu, libération_en_cours,
                   addition_en_cours, soustraction_en_cours, état_consultation) ;

type nature_évènement = (prologue, épilogue) ;

fin rappel §


type nature_demande =
   record
      séma_utilisateur : sémaphore ;          § la demande de l'utilisateur doit     §
      accès            : nature_accès ;       § comporter les informations ci-contre §
      genre            : t_nature_évènement   § elle sera stocker dans la            §
   end ; § nature_demande §                   § file_des_demandes                    §

§ structure de données file_des_demandes §

const max_file_demandes = ... ; § taille maximum de la file des demandes §

var file_des_demandes : array (1..max_fil_demandes) of nature_demande ;

    indice_courant : int ; § initialisé à 1 §

    contrôleur_en_attente : bool ; § initialisé à false §

    attente, exmut : sémaphore ; § hauteur initiale à 0 §

indice_courant := 1 ;

contrôleur_en_attente := false ;

§ initialiser les sémaphores attente et exmut à la hauteur 0 §
```

```
proc déposer_demande (demande : nature_demande) ;
   § le processus utilisateur dépose une demande dans la file des demandes
     en utilisant cette procédure §
begin
   p (exmut) ; § acquérir l'exclusion mutuelle sur la file des demandes §
   file_de_demandes [indice_courant] := demande ;
      § mettre la demande dans la file des demandes ;
        voir dans illustration 12 comment le processus utilisateur
        a préparé sa demande §
   indice_courant := indice_courant + 1 ;
   § tester le débordement dans la file des demandes §
   if contrôleur_en_attente := true then
   begin
      v (attente) ; § réveiller le processus contrôleur qui a fait
                      p (attente) dans la procédure retirer_demande §
      contrôleur_en_attente := false ;
   end ;
   v (exmut) ; § libérer l'exclusion mutuelle sur la file des demandes §
end §déposer_demande§ ;
```

```
proc choisir_demande  (sémaphore_attendu : sémaphore,
                       routine_de_contrôle : nom_routine,    § valeur de retour §
                       nature_exécution : t_nature_exécution § valeur de retour §) ;
   § but de la procédure choisir_demande :
     chercher la demande correspondante d'un processus utilisateur donné (identifié
     en l'occurrence par un sémaphore privé) et ramener comme valeurs de retour la
     routine de contrôle à exécuter et l'information : exécutable ou non §
var indice_de_travail : int ;
begin
   indice_de_travail := 1 ;
   p (exmut) ; § acquérir l'exclusion mutuelle sur la file des demandes §
   repeat
      if sémaphore_attendu = file_des_demandes [indice_de_travail].séma_utilisateur
      then § on a trouvé la demande dont le sémaphore est le sémaphore attendu ;
              i.e. on a trouvé le processus attendu §
      begin
         §call§ tester (indice_de_travail,
                        indice_de_travail,
                        indice_de_travail, § valeur de retour §
                        routine_de_contrôle, § valeur de retour §
                        nature_exécution §valeur de retour §) ;
         §call§ enlever (indice_de_travail) ; § enlever la demande trouvé §
      end
      else indice_de_travail := indice_de_travail + 1 ;
   until (indice_de_travail = indice_courant) ;
end §choisir_demande§ ;
```

```
proc retirer_demande (sémaphore_utilisateur : sémaphore,
                      routine_de_contrôle : nom_routine) ;
   § but de la procédure retirer_demande :
     le processus contrôleur retire une demande dans la file des demandes  en appelant
     cette procédure; s'il n'y a pas de demande exécutable à l'état courant de la
     ressource, il se met en attente sur le sémaphore attente et le booléen contrôleur
     _en_attente est positionné à vrai; s'il y en a, cette procédure ramène comme
     résultats le sémaphore privé du processus utilisateur sur lequel il a fait p et
     les informations nécéssaires à l'exécution de la routine de contrôle correspon-
     dantes à la demande §
var indice_de_la_demande_exécutable,
    indice_de_travail : int ;
    nature_exécution : t_nature_exécution ; § (exécutable, non_exécutable) §
begin
   indice_de_travail := 1 ;
   repeat
      p (exmut) ; § acquérir l'exclusion mutuelle sur la file_des_demandes §
      §call§ tester (
                § entrée : indice inférieur = § indice_de_travail,
                § entrée : indice supérieur = § indice_courant,
                § résultat :                  § indice_de_la_demande_exécutable,
                § résultat : la routine à
                             exécuter est   = § routine_de_contrôle,
                § résultat : exécutable ou
                             non exécutable   § nature_exécution) ;
      if nature_exécution = non_exécutable then
      begin
         contrôleur_en_attente := true ;
         indice_de_travail := indice_courant ;
         v (exmut) ; § libérer l'exclusion mutuelle sur la file des demandes §
         p (attente) ; § le processus contrôleur se met en attente d'être réveillé
                         par un (ou des) processus utilisateur qui a déposé une
                         demande; l'indice courant de la file des demandes aura
                         changé, l'indice de travail reste le même §
      end ; § nature_exécution est une variable locale à la procédure retirer_demande §
   until (nature_exécution = exécutable) ;
   § on a trouvé une demande exécutable §
   sémaphore_utilisateur := file_des_demandes [indice_de_la_demande_exécutable].séma_
                                                                        utilisateur ;
   §call§ enlever (indice_de_la_demande_exécutable) ;
            § enlever la demande trouvée dans la file des demandes §
   v (exmut) ; § libérer l'exclusion mutuelle sur la file_des_demandes §
end §retirer_demande§ ;



proc enlever (indice_demande_à_enlever : int);
   § enlever une demande dans la file des demandes §
var indice_de_travail : int ;
begin
   indice_courant := indice_courant - 1 ;
   indice_de_travail := indice_demande_à_enlever ;
   while indice_de_travail < indice_courant do
   begin
      file_des_demandes [indice_de_travail] := file_des_demandes [indice_de_travail+1] ;
      indice_de_travail := indice_de_travail + 1 ;
   end ;
end §enlever§ ;
```

```
proc tester (indice_inférieur : int,
             indice_supérieur : int,
             indice_résultat  : int,  § valeur de retour §
             routine_de_contrôle : nom_routine, § valeur de retour §
             nature_exécution : t_nature_exécution § valeur de retour §) ;
   § but de la procédure tester :
     chercher s'il y a une demande exécutable à l'état courant de la ressource
     parmi les demandes comprises entre l'indice_inférieur et l'indice_supérieur
     de la file des demandes;
     s'il n'y en a pas, elle ramène non_exécutable dans nature_exécution;
     sinon, elle ramène les résultats suivants :
             . indice de la demande exécutable dans indice_résultat,
             . les informations nécéssaires pour exécuter la routine de contrôle
               correspondante à cette demande,
             . exécutable dans nature_exécution §
var indice_de_travail : int ;
begin
   indice_de_travail := indice_inférieur ;
   nature_exécution := non_exécutable ;
   while (nature_exécution = non_exécutable) and (indice_de_travail < indice_supérieur
   do
   begin
      if file_des_demandes [indice_de_travail].genre = prologue then
      begin
         if table_prologues [état_courant,
                             file_des_demandes [indice_de_travail].accès] = impossible
         then nature_exécution := non_exécutable
         else begin
                 nature_exécution := exécutable ;
                 routine_de_contrôle.g := prologue ;
                 routine_de_contrôle.i := état_courant ;
                 routine_de_contrôle.j := file_des_demandes [indice_de_travail].accès
              end ;
      end ;
      else
      begin
         if table_épilogues [état_courant,
                             file_des_demandes [indice_de_travail].accès] = impossible
         then nature_exécution := non_exécutable
         else begin
                 nature_exécution := exécutable ;
                 routine_de_contrôle.g := épilogue ;
                 routine_de_contrôle.i := état_courant ;
                 routine_de_contrôle.j := file_des_demandes [indice_de_travail].accès
              end ;
      end ;
      indice_de_travail := indice_de_travail + 1 ;
   end ;
   indice_résultat := indice_de_travail - 1 ;
end §tester§ ;
```

3.5. Accès concurrents à un système de ressources partagées

Dans le paragraphe précédent, nous avons considéré que l'algorithme d'un service spécifie des accès à une seule ressource partagée.

Dans ce paragraphe nous considérons le cas général, où chaque service peut spécifier des accès à plusieurs ressources partagées :

Soit le système de ressources partagées composé d'un compte bancaire et d'un compte épargne-logement. Le service "retrait d'argent" (cf. Chapitre 2) fonctionne de la manière suivante : s'il n'y a pas assez d'argent dans le compte bancaire et s'il y en a assez dans le compte épargne-logement, on retire de l'argent au compte épargne-logement pour faire l'appoint au compte bancaire avant de faire le retrait sur ce compte. Le déroulement de cette séquence d'accès successifs dépend de la valeur des comptes.

Du point de vue de chaque ressource, la technique de contrôle du parallélisme ne change pas. Entre deux accès consécutifs à une même ressource, son processus contrôleur reste bloqué par le processus utilisateur qui exécute l'accès. La seule différence est que maintenant entre ces deux accès à la même ressource, le processus utilisateur peut exécuter des accès à d'autres ressources partagées.

Evidemment, ce point de vue ne tient pas compte des phénomènes d'interblocage, tâche difficile si nous ne pouvons pas faire des hypothèses sur la nature des accès comme c'est le cas pour les bases de données (cf. [ROS 78] et Annexe B-III).

Néanmoins, dans notre proposition :

a) Les structures algorithmiques où les accès concurrents sont spécifiés (services), sont bien isolées dans l'algorithme global d'une application.

b) Les services peuvent être pré-compilés et catalogués, et donc offerts aux programmeurs des applications comme des primitives d'un langage de plus haut niveau.

c) Pour spécifier les services on s'appuie sur le "graphe de synchronisation" - formalisme graphique qui offre au programmeur des services, une vue globale des règles de synchronisation des accès à la ressource.

Ces caractéristiques peuvent éventuellement aider les programmeurs à éviter les phénomènes cités.

Dans ces conditions, l'unique modification à apporter à la proposition telle qu'elle a été faite jusqu'ici, concerne les algorithmes du prologue et de l'épilogue d'un service. Le booléen REVEILLER-CONTROLEUR utilisé dans les algorithmes proposés au chapitre 3.4.4.B, sert à contrôler dynamiquement si le processus contrôleur de la ressource accédée au cours du service, doit être réveillé avant chaque accès et à la fin du service. Dans le cas d'un système de ressources il faut définir un tel booléen pour chaque ressource potentiellement accessible du service.

Une mise en oeuvre possible de cette modification, est présentée par la suite.

Le langage d'expression des services doit permettre de déterminer statiquement une liste de toutes les ressources potentiellement accessibles à partir du service. Les ressources sont identifiées par son rang dans cette liste.

Le prologue d'un service doit donc initialiser, avec la valeur faux les booléens REVEILLER-CONTROLEUR correspondants au nombre de ressources (NB-RESSOURCES) accessibles à partir du service. L'illustration 17 décrit la nouvelle version de la routine PROLOGUE-SERVICE.

ILLUSTRATION 17

```
const nb_ressources = .. ;

var réveiller_contrôleur : array [1..nb_ressources] of bool ;

proc prologue_service ;
var i : int ;
begin
   for i := 1 to nb_ressources do
   begin
      réveiller_contrôleur [i] := false ;
   end ;
end §prologue_service§ ;
```

L'épilogue d'un service doit réveiller les processus contrôleurs des ressources qui ont été effectivement accédées au cours du service. L'illustration 18 décrit la nouvelle version de la routine EPILOGUE-SERVICE.

ILLUSTRATION 18

```
var tableau_sémaphore_contrôleur :
   array  1..nb_ressources  of sémaphore ;

proc épilogue_service ;
var i : int ;
begin
   for i := 1 to nb_ressources do
   begin
      if réveiller_contrôleur (i) = true then
         v (tableau_sémaphore_contrôleur i , 'c est une fin service') ;
   end ;
end §épilogue_service§ ;
```

3.6. "Représentation" d'une ressource

La représentation d'une ressource partagée peut être vue comme un ensemble de données et de routines, qui ont été décrites dans les trois paragraphes précédents (3.3., 3.4. et 3.5.). Ici nous présentons un point de vue d'ensemble, qui tient compte de la description d'une mise-en-oeuvre faite au paragraphe 3.4.4.

3.6.1. Point de vue d'ensemble

La figure 19 montre la représentation de la ressource compte bancaire, structurée en quatre parties.

I) Celle qui est accessible aux processus utilisateurs. Elle est constituée par les routines prologue et épilogue d'accès, par la valeur partageable de la ressource (solde) et par les fonctions d'accès à la ressource (addition, soustraction,...)

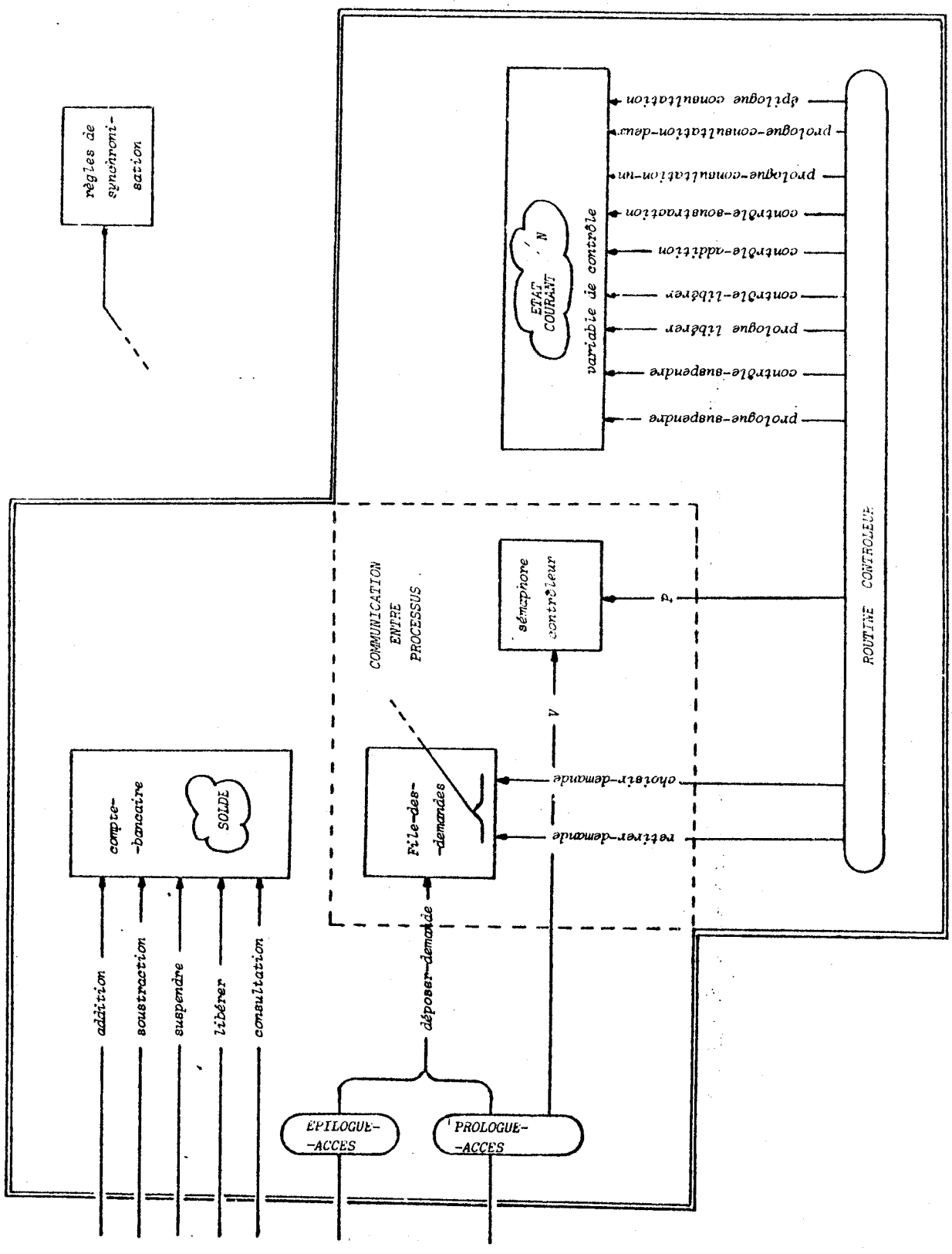

règles de synchronisation
variable de contrôle
ETAT COURANT
N
épilogue consultation
prologue-consultation-deux
prologue-consultation-un
contrôle-soustraction
contrôle-addition
contrôle-libérer
prologue libérer
contrôle-suspendre
prologue-suspendre
ROUTINE CONTROLEUR
COMMUNICATION ENTRE PROCESSUS
sémaphore contrôleur
P
V
File-des-demandes
compte-bancaire
SOLDE
obtenir-demande
retirer-demande
addition
soustraction
suspendre
libérer
consultation
déposer-demande
EPILOGUE-ACCES
PROLOGUE-ACCES

FIGURE 19

II) Celle qui est accessible exclusivement au processus contrôleur et qui représente l'état d'accessibilité de la ressource. Elle est constituée par la variable de contrôle (entier N et état courant de la ressource), par les routines de contrôle et par la routine contrôleur.

III) Celle qui représente le mécanisme de communication entre le processus contrôleur de la ressource et les processus utilisateurs. Elle est constituée par :

- le sémaphore contrôleur auquel le processus contrôleur accède à travers la fonction d'accès V, et les processus utilisateurs à travers la fonction d'accès P,
- par la file-des-demandes à laquelle le processus contrôleur accède à travers les fonctions d'accès RETIRER-DEMANDE et CHOISIR-DEMANDE, et les processus utilisateurs à travers la fonction d'accès DEPOSER-DEMANDE.

IV) Celle qui spécifie les règles de synchronisation. Elle est constituée par une structure de données qui permet de mettre en relation les fonctions d'accès à la ressource, les routines de contrôle et la valeur de l'état courant (cf. 3.3.2.B). Cette structure de données est accessible seulement à partir des routines RETIRER-DEMANDE et CHOISIR-DEMANDE ; donc accédée seulement par le processus contrôleur.

3.6.2. Remarque

Si nous continuons à considérer l'ensemble de données et routines qui constituent la représentation d'une ressource partagée (un compte bancaire dans le cas de l'exemple), nous pouvons aussi classer ces éléments selon au autre critère :

I) Données qui sont propres à une représentation d'un certain type de ressource (le compte bancaire de M. DUPONT, par exemple) :
   - valeur partageable de la ressource
   - variable de contrôle
   - file-des-demandes
   - sémaphore contrôleur.

II) Eléments qui sont propres à un type de ressource (compte bancaire, par exemple), et donc communs à toutes les représentations de ressources du type considéré :

- fonctions d'accès à la ressource
- routines de contrôle
- structure de données qui spécifie les règles de synchronisation.

III) Routines qui sont communes à n'importe quelle représentation de tous les types de ressources :

- routines DEPOSER-DEMANDE, RETIRER-DEMANDE et CHOISIR-DEMANDE
- routine CONTROLEUR
- routines PROLOGUE-ACCES et EPILOGUE-ACCES

Les éléments de l'ensemble qui constitue la représentation de la ressource, qui appartiennent au groupe III sont catalogués d'une manière standard dans les bibliothèques du système.

Ceux qui appartiennent au groupe II, sont catalogués comme résultat de la traduction de la description de la ressource faite au niveau administrateur (niveau 0).

Ceux qui appartiennent au groupe I, sont générés à chaque déclaration d'une ressource.

4. ACCES REPARTIS A UNE RESSOURCE PARTAGEE

4.1. Introduction

Dans ce chapitre nous allons considérer le cas où les processus utilisateurs peuvent s'exécuter sur des sites autres que celui où est définie la représentation de la ressource partagée.

Nous faisons d'abord l'hypothèse de l'existence d'un mécanisme très simple de transfert d'information entre les sites du réseau, qui nous permet d'éten-

dre aux processus du réseau le mécanisme de communication entre les processus, utilisé dans notre proposition.

Sur cette hypothèse nous proposons une décomposition de la représentation d'une ressource entre les sites où s'exécutent les processus utilisateurs et celui où s'exécute le processus contrôleur de la ressource.

Cette décomposition permet l'exécution des accès répartis à la ressource. C'est-à-dire, elle permet aux processus utilisateurs de soumettre au processus contrôleur de chaque ressource, des demandes d'exécution des accès de façon à ce que ces accès soient contrôlés selon les spécifications des règles de synchronisation de la ressource.

Nous admettons qu'à l'identificateur de la ressource peut être localement associé le nom réseau du processus contrôleur de la ressource, sans discuter comment cette association est faite. Cette attitude et le mécanisme de communication entre sites proposé par hypothèse, nous permet de présenter le fonctionnement de notre proposition dans un environnement réseau, indépendament du mécanisme d'établissement des chaînes d'accès [CROCUS, page 68] aux ressources.

Cette approche est due au fait qu'à notre avis, un mécanisme d'adressage (qui permet la définition des chaînes d'accès aux objets) et un mécanisme de synchronisation entre processus, doivent être conceptuellement indépendants dans la description d'un système d'exploitation. C'est-à-dire que nous ne voulons pas que notre proposition soit figée par un mécanisme d'adressage donné.

4.2. Proposition

4.2.1. Hypothèses de départ

Avant d'initier la description du mécanisme permettant la répartition des accès, nous faisons l'hypothèse suivante :

a) A chaque site du réseau est associé un processus local au site, que nous appelons "le processus-du-site", et

b) il existe un mécanisme permettant à ces processus d'échanger des messages.

4.2.2. La répartition des accès

Dans le paragraphe 3 nous avons proposé une technique de contrôle du parallélisme, selon laquelle l'exécution d'une fonction d'accès à une ressource partagée, peut être vue comme l'exécution successive des routines représentées dans la figure 13.

De toutes ces routines, seulement la PROLOGUE-ACCES spécifie des accès simultanément à un objet local au service (le booléen REVEILLER-CONTROLEUR) et à un élément de la représentation de la ressource (sémaphore-contrôleur).

Ces objets ont leur représentation sur des sites différents dans le cas où le processus utilisateur s'exécute sur un site (X) différent de celui où la ressource est représentée (site Y). La routine PROLOGUE-ACCES doit donc être décomposée en deux :

PROLOGUE-LOCAL-ACCES et
PROLOGUE-DISTANT-ACCES

Un processus utilisateur (P) qui veut exécuter un accès (A) à une ressource partagée (R), représentée sur un site distant (Y), exécute d'abord la routine PROLOGUE-LOCAL-ACCES et ensuite se met en attente. Il sera réveillé quand le processus du site reçoit un message signalant la fin de l'exécution de l'accès, et contenant les résultats de l'accès.

Le processus-du-site Y reçoit le message envoyé par le processus utilisateur pendant l'exécution de la routine PROLOGUE-LOCAL-ACCES, constitué par :

- la valeur 'prologue' qui permet au processus-du-site Y de déterminer le traitement à effectuer (celui que nous décrivons maintenant).

- le nom du processus utilisateur demandeur de l'accès.

- le nom du site où s'exécute le processus utilisateur (site X).

- la valeur du booléen REVEILLER-CONTROLEUR associé au service exécuté par le processus utilisateur demandeur de l'accès.

- l'identificateur de la ressource et de la fonction d'accès à exécuter.

Dans ces conditions, le processus-du-site Y va activer un processus local à Y, qui sera le représentant du processus utilisateur distant.

Alors, le processus représentant de l'utilisateur une fois créé exécute la routine PROLOGUE-DISTANT-ACCES qui lui conduit à avoir avec le processus contrôleur de la ressource à accéder, les mêmes interactions que le processus utilisateur, si celui-ci était un processus local à Y.

Selon le mécanisme proposé dans le chapitre 3, le processus contrôleur termine la séquence d'interactions citée, par le réveil du processus utilisateur (quand le processus contrôleur termine l'exécution de la routine épilogue de contrôle). Dans le cas des accès répartis c'est le processus représentant de l'utilisateur qui sera réveillé.

Celui-ci exécute alors la routine RETOURNER-RESULTATS, et se tue. Cette routine envoie au processus-du-site X un message avec les résultats de l'accès. Ce message est constitué par :

- la valeur 'fin-accès' qui permet au processus-du-site X de déterminer le traitement à effectuer (celui que nous décrivons maintenant).
- le nom du processus utilisateur
- les résultats de l'accès.

Dans ces conditions, le processus-du-site X va réveiller le processus utilisateur et va lui passer les résultats de l'accès.

Un service se termine par l'exécution de la routine EPILOGUE-SERVICE (cf. illustration 18), où le processus utilisateur réveille les processus contrôleurs éventuellement en attente. Dans le cas des accès répartis le processus utilisateur envoie au processus-du-site de chaque contrôleur un message signalant que l'exécution du service est terminée et identifiant la ressource accédée effectivement au cours du service. C'est chacun de ces processus qui va réveiller les processus contrôleurs respectifs.

5 - CONCLUSION

Le partage de ressources, exige la définition des règles de synchronisation entre accès (concurrents) exécutés par les processus utilisateurs. Ces règles expriment à la fois :

(i) une sémantique propre à la nature des ressources, qui tendent à garantir leur intégrité ;
(ii) une sémantique propre à l'usage que chaque processus fait des ressources, dont la conséquence est le "scheduling" de ces processus. "Scheduling" que nous voulons harmonieux, dans la mesure où il permet à chaque processus d'accomplir sa fonction.

Le travail exposé a abordé les problèmes liés à cette harmonie.

Si nous supposons un ensemble de programmeurs "fiables" et travaillant dans un environnement "confiant" (c'est la position des programmeurs d'un système d'exploitation), nous pouvons leur permettre de bloquer et débloquer chaque ressource partagée. De cette façon le "scheduling" des processus utilisateurs est contrôlé par l'exécution de ces primitives, et il est harmonieux (fonction des hypothèses que nous avons faites). Dans ces conditions il semble suffisant que les techniques de contrôle du parallélisme et les langages de contrôle associés ne tiennent compte que du premier niveau de synchronisation (i) - celui qui tend à garantir l'intégrité des ressources.

Ce scénario, très souvent vérifié jusqu'à aujourd'hui, risque (à notre avis) de perdre sa position majoritaire dans le futur, quand les algorithmes parallèles seront à la portée des utilisateurs. L'informatique répartie et la banalisation de la micro-informatique invitent les utilisateurs à ce type de programmation.

Cette perspective a motivé ce travail. La technique de contrôle du parallélisme proposée tient compte des deux niveaux de synchronisation défini (i et ii).

Pour le premier niveau (i), notre proposition exige que les descriptions des règles de synchronisation et des fonctions d'accès aux ressources, soient indépendantes.

Pour le deuxième niveau (ii), nous sommes arrivés à un mécanisme permettant à chaque utilisateur de décrire des séquences d'accès sémantiquement dépendantes sans pour autant bloquer nécessairement la ressource. Pour que les accès parallèles soient permis, il est nécessaire qu'ils ne mettent pas en cause l'intégrité des ressources prises individuellement.

La structuration de la mise-en-oeuvre décrite dans le paragraphe 3.4.4., sépare nettement les deux niveaux de synchronisation. En particulier le "scheduling" des processus utilisateurs est défini par la routine CONTROLEUR (et aussi par les prologues et épilogues SERVICE et ACCES). De cette façon la description de la politique de "scheduling" est indépendante de la description des ressources, ce qui peut permettre d'adapter cette politique aux conditions d'exploitation de chaque système de ressources.

Dans notre étude, nous n'avons pas abordé deux problèmes importants :

a) Prévention d'étreintes fatales et occurrences similaires ;

b) Garantie de la cohérence globale d'un exemple d'objets partagés par des processus concurrents.

Ces types de problèmes ont été étudiés dans le domaine des bases de données réparties, notamment par Rosenkrantz, Stearns et Lewis [ROS 76] [ROS 78]. Pour le scénario que nous proposons (nous ne faisons pas d'hypothèses sur la nature des fonctions d'accès aux ressources), et à notre connaissance, ces problèmes ne sont pas résolus, et sont donc encore ouverts (cf. Annexe B-III).

Cependant, nous remarquons que les problèmes cités se posent pendant l'exécution d'une structure algorithmique propre à notre proposition - le service. Le programmeur des applications "voit" la ressource à travers l'ensemble de services catalogués dans les bibliothèques. Ce fait nous permet de penser qu'il est

possible de définir un langage pour l'expression des services, tel que (en l'absence de moyens de contrôle statique ou dynamique) la fiabilité du code produit puisse être vérifiée par l'analyste et par le programmeur. Ceci est une voie ouverte.

Le travail que nous avons présenté fait suit à notre participation à la conception des mécanismes de communication entre les processus du système SIGOR [DAN 78(a)][GMA 77], et à l'implémentation de ce système [DAN 78(b)] (cf. Annexe A).

Dans SIGOR, le mécanisme de synchronisation entre processus a été imposé par le support de multiprogrammation utilisé : SYNCOP [DAN 78(a), page 96]. Dès le début, nous étions déjà convaincus de la nécessité de substituer au mécanisme élémentaire de synchronisation par événements un mécanisme plus évolué ; cette substitution devant être une des premières extensions à étudier.

Une autre extension envisagée concernait la définition d'un langage externe, qui permette par exemple à l'utilisateur la définition d'objets de modes construits sur les modes de base du langage interprété ; c'est-à-dire envisager l'existence d'entités dont les fonctions d'accès seraient différentes de (ou plus complexes que) lire ou écrire.

Dans notre travail, nous avons aussi essayé de tenir compte du contrôle de concurrence des processus d'une application.

Dans la phase actuelle de développement du travail présenté ici, il semble que la suite logique devrait être une implémentation de la technique de contrôle proposée, qui permettrait sa validation expérimentale et son évaluation rigoureuse.

La programmation d'exemples réels est nécessaire pour analyser jusqu'où notre modèle informatique peut s'y adapter convenablement. De tels exemples pouvaient être :

a) Un protocole de maintien de la cohérence d'un fichier à copies multiples. Ces protocoles peuvent être décrits par des graphes [WIL 79] où chaque type de message peut être vu comme un accès au système de gestion des copies.

b) Un système de contrôle de processus industriels. De tels systèmes présentent des objets dont les fonctions d'accès sont les plus diverses (convoyeurs, machines à souder, ...). Pour cette phase expérimentale, il serait nécessaire de choisir un exemple sans grandes contraintes dans ce qui concerne les temps de réponse. La chaîne de montage Peugeot [KA78] serait un exemple possible.

Cette démarche pourrait être un point de départ pour la définition des langages de description du niveau administrateur et du niveau des services.

ANNEXE A

COMMUNICATION ENTRE LES PROCESSUS DU SYSTEME SIGOR : UNE IMPLEMENTATION

Plan :

1. INTRODUCTION

1.1 APPLICATION SIGOR. PROCESSUS SIGOR.

"La Machine Réseau Logique (CHU 77) SIGOR (Système d'Interprétation Généralisée Orienté Réseau) propose un langage unique, transportable, de type procédural, pour l'expression des applications réparties.

Elle se compose, sur chaque site participant, d'un sous-système interpréteur et de mécanismes d'aide à la répartition ; l'ensemble étant géré par un (sous) système de multiprogrammation de type SYNCOP" (DAN 79a).

Une application SIGOR est décrite par un programme qui peut être spécifié à travers le langage LI (DAN 78b).

Ce dernier est un langage de type procédural, parmi les primitives duquel nous trouvons l'appel synchrone de procédure (CALL conventionel), et l'appel asynchrone de procédure (ACTIVE).

Une procédure LI peut appeler (CALL ou ACTIVE) un nombre quelconque d'autres procédures LI. Inversement, pour chaque procédure LI, il y a une seule procédure LI appelante. De cette façon, les procédures qui constituent une application SIGOR, peuvent être représentées par les noeuds d'un arbre, dont les arcs représentent les relations CALL ou ACTIVE entre des couples de procédures.

L'arbre des procédures d'une application SIGOR se construit de manière dynamique. Il varie avec l'exécution des appels et des retours de procédures (synchrones et asynchrones). La figure A-1 représente un exemple d'un tel arbre à un instant t.

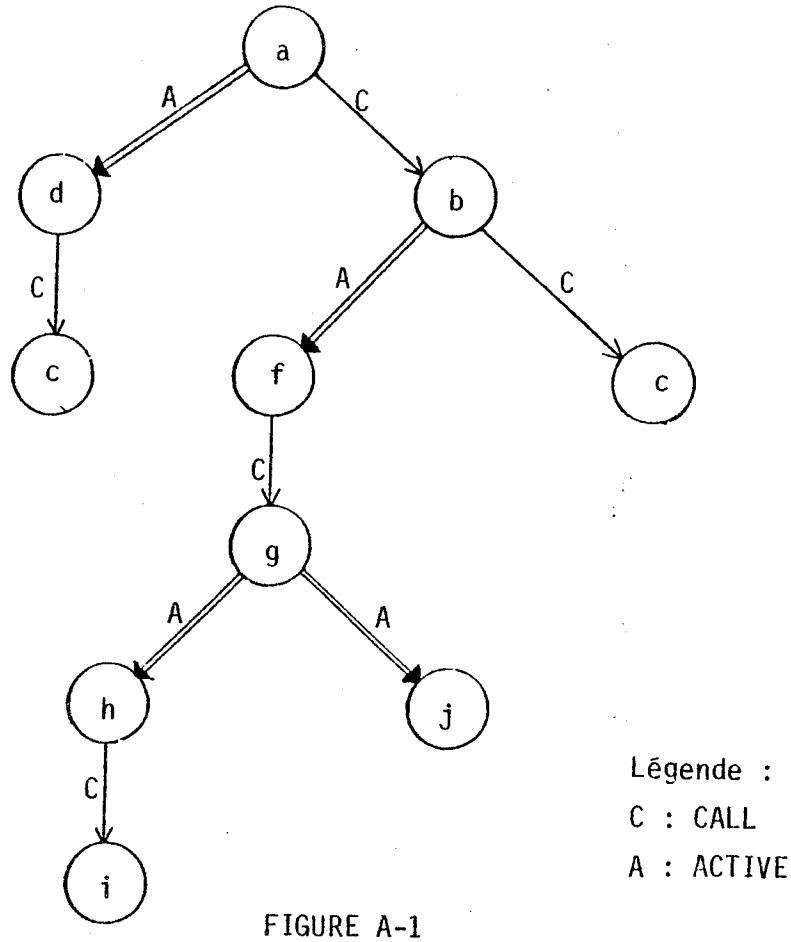


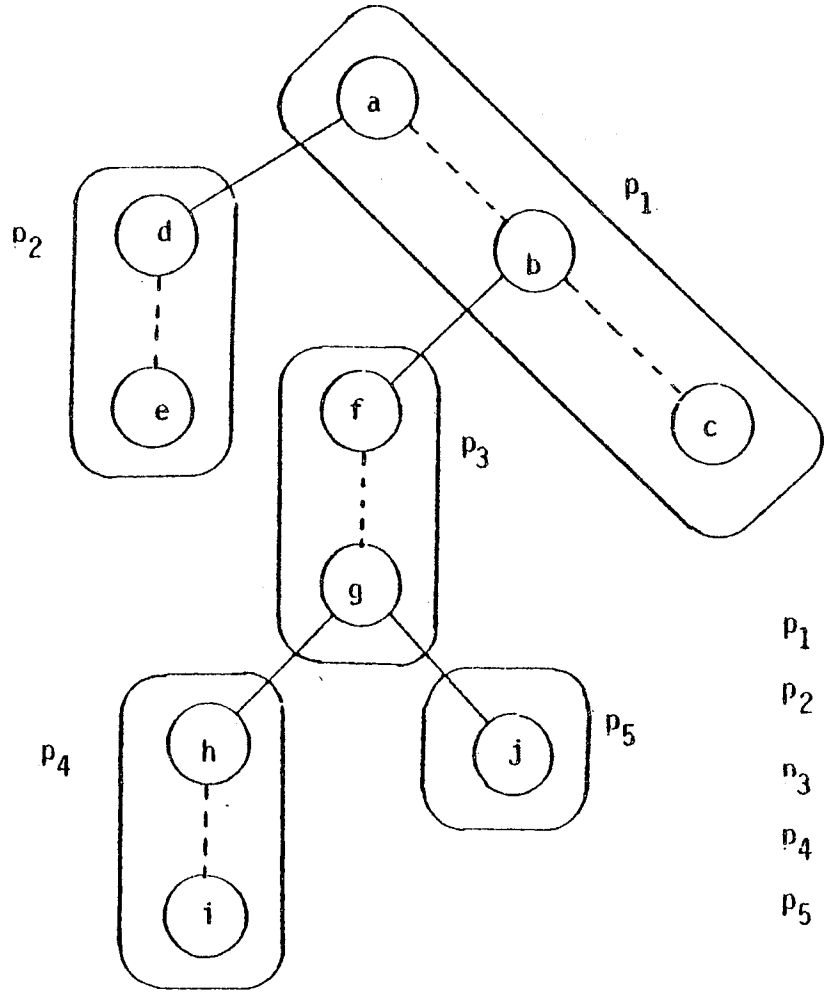
FIGURE A-1

Il y a toujours une façon unique de découper l'arbre des procédures d'une application SIGOR, en sous-ensembles tels que :

a) Si "y" est une procédure appelée de manière synchrone à partir d'une procédure "x", alors "y" et "x" appartiennent au même sous-ensemble.

b) Si "y" est une procédure appelée de manière asynchrone à partir d'une procédure "x", alors "y" et "x" n'appartiennent pas au même sous-ensemble.

A partir de l'arbre de la figure A-1, nous pouvons faire le découpage représenté sur la figure A-2.



$p_1 = \{a,b,c\}$
$p_2 = \{d,e\}$
$p_3 = \{f,g\}$
$p_4 = \{h,i\}$
$p_5 = \{j\}$

FIGURE A-2

Chaque sous-ensemble de procédures ainsi défini, est exécuté par un seul processus SIGOR, sur un site bien déterminé. Une application SIGOR peut être représentée par un arbre dynamique de processus SIGOR ; dans le cas de l'application représentée par les figures A-1 et A-2, cet arbre est :

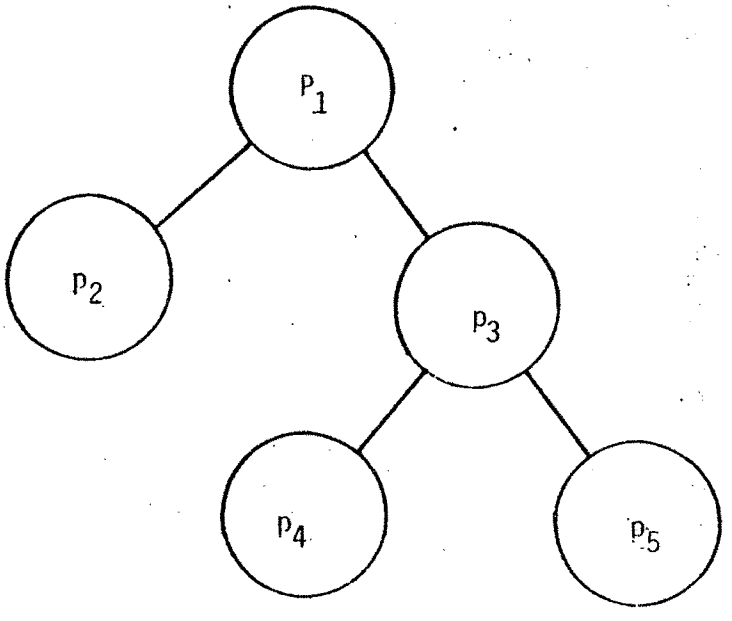


FIGURE A-3

Un processus SIGOR (P_y) est toujours activé à partir d'un autre processus SIGOR (P_x). Nous disons que P_x est le processus père de P_y, et que P_y est le processus fils de P_x (DAN 78a, page 23).

Une application SIGOR peut maintenant être définie comme un ensemble de processus SIGOR, entre lesquels est définie une relation de filiation. Ces processus doivent coopérer dans l'exécution d'une activité commune - l'application. Les mécanismes de communication entre processus du système SIGOR, offrent un moyen de contrôle de cette coopération.

1.2 ESPACE D'ADRESSAGE D'UN PROCESSUS SIGOR

Dans le système SIGOR, l'association d'un espace d'adressage à un processus, n'est pas fixe. Pendant l'exécution d'une procédure l'espace d'adressage est défini par un descriptif (CRO 75, § 3.222) appelé DMIA (Dynamic Interpretation Area). Le processus "navigue" de DMIA en DMIA, et cette "navigation" est régie par les appels et les retours de procédures. On garantit que le chemin de retour est l'inverse du chemin d'appel, par la présence d'une pile d'exécution (pile DISPLAY du contexte du processus).

Un processus n'a accès qu'aux objets dont le descripteur figure sur sa DMIA "courante". Le nom d'un objet, est son rang dans la DMIA "courante" du processus.

Un descripteur contient (entre autres) trois types d'informations sur l'objet correspondant :

- type : peut être "local", "param-val" (paramètre par valeur), "param-ref" (paramètre par référence).

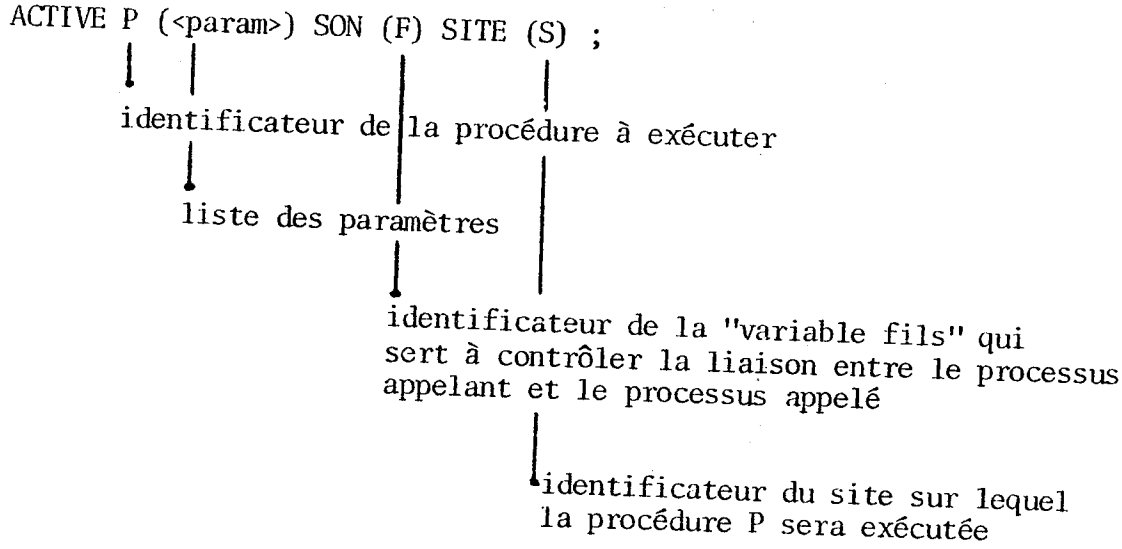
- mode : logique, entier, adresse, bloc réveil, chaîne de caractères et chaîne d'octets (DAN 78a, §2.4.1).

- repère : si le type est "local" ou "param-val" alors le repère est l'adresse de la représentation de la valeur de l'objet ; si le type est "param-ref" alors le repère est le nom du descripteur de l'objet, dans la DMIA de la procédure appelante.

La chaîne d'accès à un objet (CRO 75, § 3.111) est constituée par une séquence d'indirections sur les descripteurs de l'objet, de type "param-ref"
- séquence qui se termine par un descripteur de type "local "ou "param-val", qui repère directement la valeur de l'objet.

1.3 APPEL ASYNCHRONE DE PROCEDURE (ACTIVE). LIAISON IMPLICITE.

Un appel asynchrone de procédure se fait à travers la primitive ACTIVE :



L'interprétation d'une instruction ACTIVE, provoque :

- la création d'un processus, sur le site spécifié en paramètre de l'instruction ACTIVE - c'est le processus fils.

- l'établissement d'une voie de communication entre les deux processus (père et fils), appelée "liaison implicite" (DAN 78a, page 64).

Sur la liaison implicite, les processus père et fils échangent des messages dans les deux sens. De chaque côté de la liaison, ces messages sont traités par un processus du sous-système de multiprogrammation (SYNCOP, qui supporte le système SIGOR), le processus communication (DAN 79b). Le processus communication a accès à la station de transpat du site, interface avec le réseau, et au contexte d'exécution de son processus SIGOR. Le processus communication et le processus SIGOR correspondant s'exécutent en exclusion mutuelle et se communiquent des informations à travers un tampon défini sur le contexte du processus SIGOR.

La structure des messages échangés sur la liaison implicite, est la suivante :

{ repère-origine ; repère-destination ; fonction ; paramètres }

où :

repère-origine - c'est le nom d'un objet lié à la DMIA du processus SIGOR (père ou fils) qui a envoyé le message et à qui éventuellement peut être adressé un message "réponse".

repère-destination - c'est le nom d'un objet lié à une mémoire virtuelle (DMIA) du processus SIGOR (père ou fils) à qui le message est adressé.

fonction - identifie l'action devant être exécutée par le processus communication sur l'objet repéré (repère-destination).

paramètres - contient la valeur des paramètres devant être utilisés pour exécuter l'action référée.

Une liaison implicite met donc en correspondance deux espaces d'adressage (DMIA s) :

- Du côté du fils,il est toujours bien défini ; c'est la DMIA de la procédure racine du fils (première procédure exécutée après l'activation du processus).

- Du côté du père, c'est la DMIA de la procédure dans laquelle a été spécifé l'activation du processus fils.

Le repère-destination défini sur un message circulant dans le sens fils-père, est donc insuffisant pour que le processus communication puisse établir la chaîne d'accès vers l'objet. Il lui faut un repère vers la DMIA du père, par rapport à laquelle le nom transporté par le message doit être interprété. Ce repère est défini sur un objet associé à la liaison, du côté du père, et donc accessible au processus communication. Cet objet fait aussi partie de l'espace d'adressage du processus SIGOR, pour permettre à l'utilisateur de contrôler la liaison implicite. Nous désignons cet objet par "variable fils" - il a le mode de base "fils", du langage LI (DAN 78a, page 58).

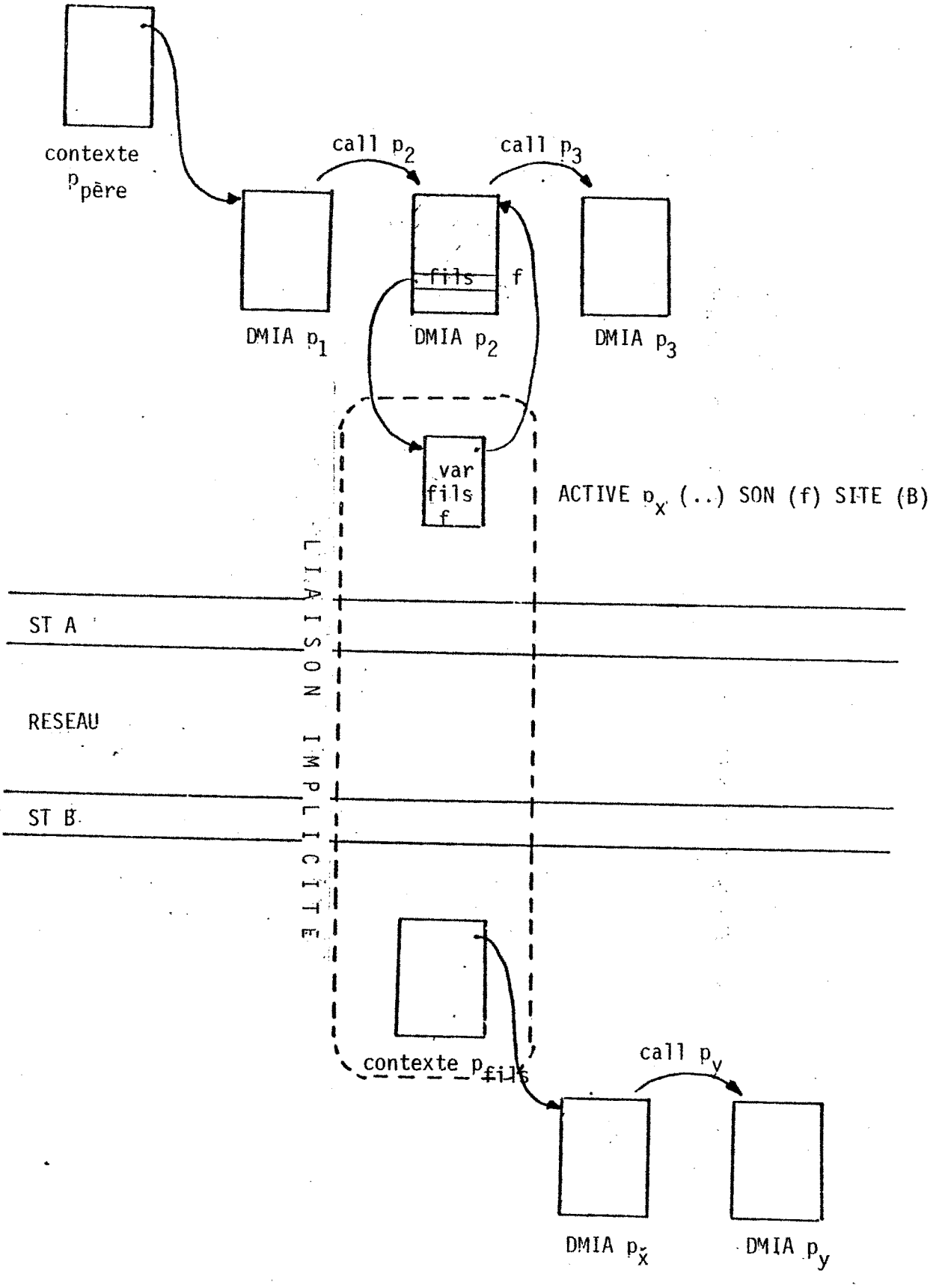

contexte
$p_{père}$
call p_2
call p_3
fils
f
DMIA p_1
DMIA p_2
DMIA p_3
var
fils
f
ACTIVE p_x (..) SON (f) SITE (B)
LIAISON IMPLICITE
ST A
RESEAU
ST B
contexte p_{fils}
call p_y
DMIA p_x
DMIA p_y

FIGURE A-4

2. LA COMMUNICATION IMPLICITE

2.1 CHAINE D'ACCES A UN OBJET D'UNE APPLICATION SIGOR

Un appel asynchrone de procédure peut aussi s'effectuer avec passage de paramètres par valeur et/ou par référence.

Le passage de paramètres par valeur ne pose pas de problèmes.

Le passage de paramètres par référence offre un moyen de communication entre les processus d'une application SIGOR, puisque ces processus peuvent partager dynamiquement l'accès à des objets non-dupliqués (DANS 79b). Ce moyen de communication est réalisé par le mécanisme de communication implicite. Ce mécanisme offre au programmeur traditionnel, un outil d'expression très souple, puisque le concept de "passage de paramètres par référence" est bien connu et maîtrisé par ce type de programmeur (DAN 79a).

Les outils que nous venons de décrire (liaison implicite, variable fils, processus communication, échange de messages formatés ...) nous permettent la définition d'un mécanisme d'adressage, sur l'arbre des procédures d'une application SIGOR, qui est la généralisation de celui déjà défini pour les objets d'un processus SIGOR (GMA 77).

La chaîne d'accès à un objet d'une application SIGOR, est constituée par une séquence de descripteurs de l'objet, qui se repèrent successivement :

$$d_0 \leftarrow d_1 \leftarrow d_2 \leftarrow \cdots - d_{i-1} \leftarrow d_i \leftarrow \cdots - d_m$$

Pour i > o, d_i est un descripteur de type "param-ref". Il repère le descripteur d_{i-1} en spécifiant "son" nom (cf. figure A-5)

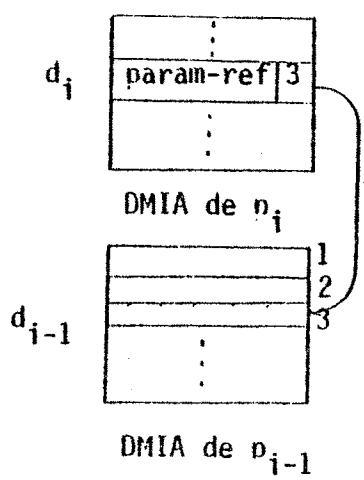


FIGURE A-5

d_o repère directement la valeur de l'objet, donc son type est "local" ou "param-val".

a) Si deux descripteurs successifs de la chaîne d'accès (d_i et d_{i-1}) correspondent à l'exécution de deux procédures (p_i et p_{i-1}) par un même processus *, alors entre les espaces d'adressage (DMIA s) correspondants il y a une "liaison locale", réalisée par la pile DISPLAY du contexte du processus (cf. figure A-6). Cette liaison permet la détermination de l'espace d'adressage à qui d_{i-1} appartient, et par rapport auquel doit être interprété le nom spécifié sur d_i.

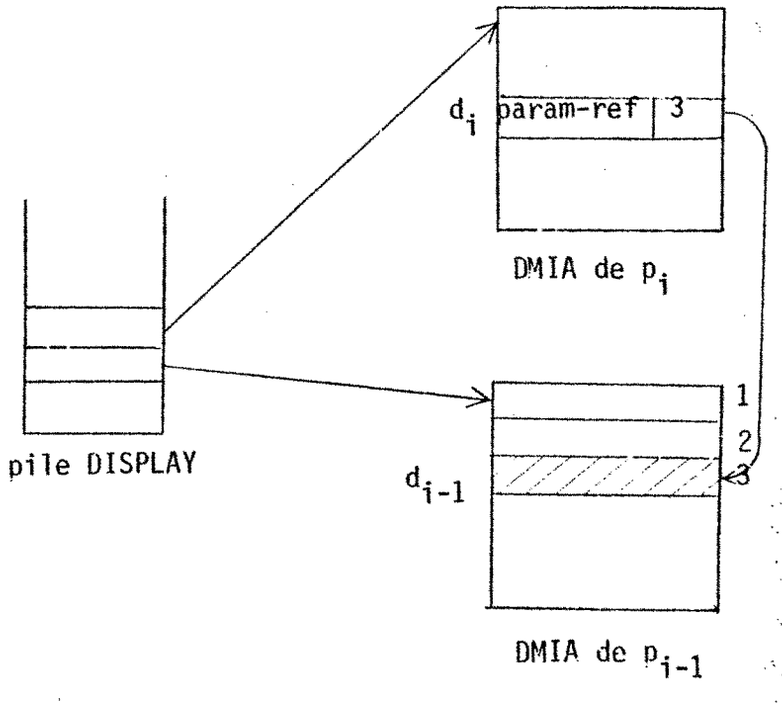


FIGURE A-6

(*) - p_{i-1} est la procédure appelante (CALL) de p_i.

b) Le cas échéant, c'est-à-dire si d_i et d_{i-1} correspondent à l'exécution de deux procédures p_i et p_{i-1} par deux processus différents (le processus fils P_i et le processus père P_{i-1}), alors entre les espaces d'adressage correspondants il y a une "liaison réseau" réalisée par l'exécution d'un protocole entre les processus communication qui partagent (un de chaque côté) la liaison implicite (cf. figure A-4) entre les processus père et fils - c'est le protocole de communication implicite.

Le champ "repère" du descripteur d'un paramètre reçu par référence dans une instruction ACTIVE, contient le nom du paramètre effectif dans la DMIA appelante.

EXEMPLE :

```
A : procédure ;
     dcl FILS1 son ;
     dcl XA integer ;
     B : procédure (XB) ;
          dcl FILS2 son ;
          dcl XB integer by name ;
          C : procédure (XC) ;
               dcl XC integer by name ;
               XC := XC + 1 ;
          end C ;
          active C(XC) son(FILS2) site(Y) ;
     end B ;
     active B(XA) son(FILS1) site(Z) ;
end A ;
```

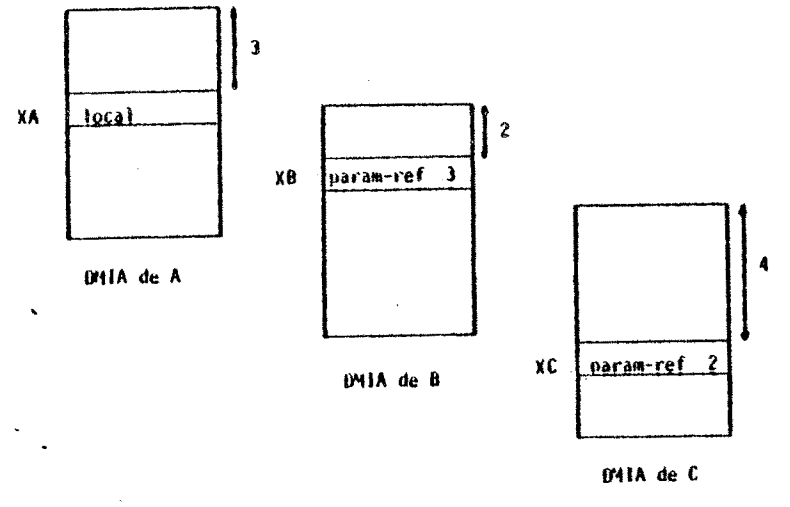


FIGURE A-7

2.2 FILE DES ADRESSES DE RETOUR (description simplifiée du protocole de communication implicite)

Nous allons considérer deux processus (père et fils) qui partagent l'accès (lecture et écriture) à un objet non-dupliqué, passé comme paramètre par référence, à l'activation du fils.

a) Pour affecter une valeur au paramètre effectif, le processus fils envoie un message au processus père, sur la liaison implicite, tel que :

repère-destination := nom de l'objet dans la DMIA du père (champ "repère" du descripteur local de l'objet) ;
fonction := écriture ;
paramètres := valeur à affecter ;
repère-origine - n'est pas nécessaire, puisqu'il n'y aura aucun message de réponse.

C'est le processus communication qui traite ce message, de l'autre côté de la liaison implicite (côté père). C'est lui qui réalise l'affectation. Si le processus père avait déjà reçu l'objet comme paramètre par référence, ce procédé est répété, soit sur la liaison locale, soit sur la liaison réseau, jusqu'à ce que la chaîne d'accès soit complètement établie.

b) Pour lire la valeur du paramètre effectif, le processus fils envoie un message au processus père, sur la liaison implicite, tel que :

repère-origine := nom de l'objet dans la DMIA du fils ;
repère-destination := nom de l'objet dans la DMIA du père (champ "repère" du descripteur local de l'objet) ;
fonction := lire ;
paramètres - champ vide.

De l'autre côté de la liaison implicite (côté du père), le processus communication peut traiter ce message de deux manières distinctes :

i - Si l'objet a le type local ou param-val, le processus communication envoie au processus fils un message "réponse" :

repère-destination := nom de l'objet dans la DMIA du fils
(contenu du champ repère-origine, dans le message lire) ;
fonction := réponse ;
paramètres := valeur de l'objet ;
repère-origine - n'est pas nécessaire ;

ii - Si l'objet a le type param-ref, le processus communication envoie à son processus père, un message lire. Quand la réponse arrive, il doit la faire transiter vers son fils qui lui avait envoyé la demande de lecture. Il faut donc garder "l'adresse de retour" des messages réponse.

A cet effet, nous ajoutons au descripteur d'un objet, une file dont chaque élément est constitué par deux noms :

{ FILS ; REPERE }

tels que :

FILS - nom de la variable fils associé à la liaison implicite sur laquelle est arrivé le message lire.

REPERE - le repère-origine du message lire.

Avec ces informations, le processus communication peut faire transiter le message réponse (quand il arrivera) sur la "bonne" liaison, vers le "bon" descripteur.

Une variable fils identifie toujours une liaison implicite, alors qu'il peut être aussi nécessaire de faire transiter les réponses sur une liaison locale. Puisqu'il n'existe aucun objet de nom < 0 >, nous réservons ce nom pour une variable fils fictive associée à la liaison locale (si elle existe). De cette façon, quand le processus communication doit faire transiter un message réponse, si l'adresse de retour est (0,N), alors :

- s'il existe une liaison locale, le nom N est interprété par rapport à la DMIA associée à la liaison locale ;

- sinon, N est le nom, dans la DMIA courante, de l'objet concerné par la réponse.

2.3 PROTOCOLE DE COMMUNICATION IMPLICITE

Nous avons soulevé quelques problèmes posés par la définition du protocole de communication implicite. Ce protocole peut être complètement décrit comme un interpréteur des messages qui arrivent sur la liaison implicite, et exécuté par chaque processus communication.

L'action exécutée sur un objet liée à cette interprétation, dépend du contenu du message interprété et, si l'objet est du type param-ref, dépend aussi de la nature des actions antérieurement exécutées sur l'objet *. Cette dépendance peut être décrite par un automate associé à chaque descripteur de type param-ref.

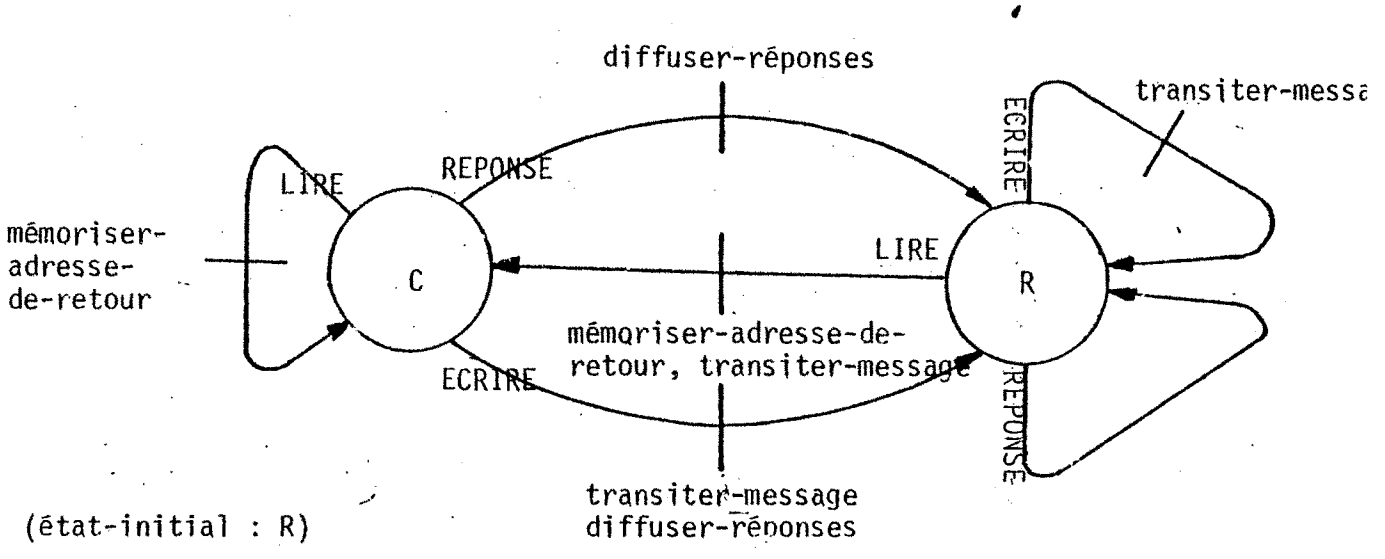


FIGURE A-8

(*) - Puisque l'exécution de ces actions est demandée par plusieurs entités (processus) asynchrones, il peut avoir des conflits entre ces demandes.

L'état de l'automate est défini sur un nouveau champ du descripteur de l'objet, que nous désignons par ETAT.

Le traitement d'un message par le processus communication, est le suivant :

1. Recevoir un message (M) sur une liaison (locale ou réseau) du processus SIGOR ;

2. Si le descripteur repéré par M est de type param-ref, alors :

    Cas fonction de M est :

    1. LIRE :
        1. Mémoriser adresse de retour (fils ; repère) :
            1. fils := nom de la variable fils associée à la liaison, sur laquelle est arrivé le message M ;
            2. repère := repère-origine du message M ;
        2. Si état du descripteur est R, alors :
            1. faire transiter un message M' vers la procédure appelante (soit à travers une liaison locale, soit à travers une liaison réseau) :
                1. repère-origine de M' := repère-destination de M ;
                2. repère-destination de M' := repère défini sur le descripteur ;
                3. fonction et paramètres de M' := fonction et paramètres de M ;
            2. état du descripteur := C ;

    2. ECRIRE :
        1. Faire transiter un message M' vers la procédure appelante (=2.1.2.1) ;
        2. Si état du descripteur est C (il y a au moins une réponse attendue), alors :
            1. fonction de M := réponse ;
            2. diffuser réponses :
                Pour tous les adresses de retour mémorisés sur le descripteur.
                1. Faire transiter un message M' vers la procédure appelée, correspondante à l'adresse de retour (soit sur une liaison locale, soit sur une liaison réseau) :

1. repère-destination de M' := repère défini par l'adresse de retour ;
2. fonction et paramètres de M' := fonction et paramètres de M ;
3. état du descripteur := R ;

3. REPONSE :
   1. Si état du descripteur est C (il y a au moins une réponse attendue), alors :
      1. Diffuser réponses (≡2.2.2.2) ;
      2. état du descripteur := R ;

3. Sinon (descripteur repéré par M, de type local ou param-val)

Cas fonction de M est :

1. LIRE :
   1. Est constitué un message M' (réponse) :
      1. repère-destination de M' := repère-origine de M ;
      2. fonction de M' := réponse ;
      3. paramètres de M' := valeur de l'objet ;
   2. Le message M' est envoyé sur la liaison sur laquelle la demande de lecture avait été reçue (liaison locale, ou liaison réseau) ;

2. ECRIRE :
   1. La nouvelle valeur (paramètres de M) est affectée à l'objet ;
   2. Réveiller le processus SIGOR ;

2.4 SCHEDULING DES PROCESSUS SIGOR ET COMMUNICATION

a) Le processus SIGOR et son processus communication, s'exécutent en exclusion mutuelle.

b) Quand un processus SIGOR exécute une affectation ou une lecture d'un objet de type param-ref, il envoie la demande correspondante à son processus communication (à cet effet il y a un tampon sur le contexte du processus SIGOR), et réveille ce processus.

c) "Périodiquement" le processus SIGOR réveille son processus communication (même en dehors des conditions énoncées dans b), pour permettre le traitement des demandes de ses fils.

d) Quand un processus communication a traité toutes les demandes adressées à un processus SIGOR, il le réveille (si le processus SIGOR n'est pas en attente d'une réponse).

Si les processus SIGOR et communication sont exécutés par des processeurs différents (ce qui n'était pas le cas dans notre implémentation), on peut améliorer les performances du système, à travers d'autres règles de synchronisation, où les processus peuvent s'exécuter en parallèle. Cette solution compliquera bien entendu le mécanisme de communication entre les deux processus.

3. LA COMMUNICATION EXPLICITE

3.1 INTRODUCTION. LIAISON EXPLICITE

L'exécution d'une affectation où d'une lecture sur un paramètre passé par référence dans un appel asynchrone, oblige le système à un "overhead" appréciable, par rapport à l'exécution des même actions sur un objet local. Le mécanisme de communication explicite permet à deux processus SIGOR d'échanger des informations de manière plus efficace.

Le protocole de communication explicite permet :

- L'ouverture d'une voie de communication entre deux processus SIGOR - c'est la "liaison explicite" (DAN 78a, page 87).

- L'échange de messages entre les deux processus, à travers la liaison ouverte.

3.2 UTILISATION DE LA LIAISON EXPLICITE

Chaque processus voit la liaison explicite comme un fichier virtuel, sur lequel il peut "lire" ou "écrire" des informations.

La liaison explicite est accessible au processus SIGOR, à travers un objet de mode de base "liaison" (du langage LI) lié à son espace d'adressage - c'est la variable liaison (DAN 78a, § 4.2.4). Un processus SIGOR "lit" sur sa variable liaison, une valeur que son interlocuteur a "écrit" sur sa variable liaison (et réciproquement). Les instructions respectives sont :

LIRE (nom de la variable liaison, nom du tampon qui reçoit l'information)

ECRIRE (nom de la variable liaison, texte)

Comme la variable fils de la communication implicite, la variable liaison est associée à la voie de communication ouverte entre les deux processus SIGOR, et est accessible au processus communication chargé de faire la gestion de cette voie et le traitement des messages qui y circulent.

3.3 ETABLISSEMENT ET RUPTURE D'UNE LIAISON EXPLICITE

Pour ouvrir une liaison explicite, deux processus SIGOR exécutent un protocole d'ouverture de liaison. Pour ce dialogue, les deux processus utilisent les moyens offerts par la communication implicite :

Soient p_x et p_y les procédures exécutées respectivement par les processus P_a et P_b, qui veulent ouvrir une liaison explicite. Etant donné que P_a et P_b sont deux processus d'une même application SIGOR, alors il y a toujours au moins une procédure (p_t) ancêtre commune de p_x et p_y, sur l'arbre des procédures de l'application. Sur l'arbre représenté dans la figure A-1, les procédures h et c, ont comme ancêtres communes les procédures b et a.

Pour que les processus P_a et P_b puissent ouvrir une liaison explicite, il est nécessaire qu'il partagent l'accès à un objet de mode de base "rendez-vous", local à une procédure ancêtre des procédures exécutées par P_a et P_b (DAN 78a, § 4.2.4). C'est-à-dire qu'il faut que cet objet (variable rendez-vous) ait été passé comme paramètre par référence, dans les appels (synchrones et asynchrones) successifs de l'arbre des procédures, qui conduisent de p_t à p_x, et de p_t à p_y.

Pour ouvrir une liaison explicite, chaque processus exécute un accès "ouverture" * à la variable rendez-vous commune. Mais la même variable rendez-vous peut être passé comme paramètre à un ensemble S de plusieurs processus fils **. Dans la variable rendez-vous, une demande d'ouverture émise par un processus P_i, sera associée à une demande identique d'un processus P_j. Mais, dû à la nature asynchrone des processus d'une application SIGOR, nous ne pouvons pas faire d'hypothèses sur l'identité de P_i et P_j, autres que :

$$\begin{matrix} P_i \in S \\ P_j \in S \end{matrix} \qquad (i \neq j)$$

(*) - A travers le mécanisme de communication implicite.

(**)- Néanmoins, à chaque instant, la variable rendez-vous supporte une seule liaison explicite (entre deux processus de S).

Pour avoir un contrôle sur ces identités, nous voulons que sur l'ensemble S, on puisse définir des sous-ensembles F_i (familles de processus), de telle manière que si un processus $P_k \in F_i$ demande une ouverture de liaison, elle ne sera établie qu'avec un processus $P_j \in F_i$. En particulier, P_j et P_k peuvent être les seuls processus de la famille F_i. Une famille est caractérisée par une valeur, que nous appelerons la "clef" de cette famille *. Un processus peut bien entendu appartenir à plus d'une famille (en utilisant plusieurs variables liaisons).

L'utilisation d'une liaison explicite, est précédée et suivie d'un dialogue entre les deux interlocuteurs (processus SIGOR), à travers la variable rendez-vous commune.

Dialogue initial

En ce qui concerne l'ouverture de la liaison explicite, les deux processus (P_a et P_b) n'occupent pas des positions symétriques - il y en a un qui prend l'initiative. Mais l'asynchronisme des deux processus ne permet pas de faire des hypothèses sur sa vitesse relative, et donc P_a et P_b doivent entamer un dialogue symétrique. L'assymétrie référée est gérée au niveau de la variable rendez-vous, comme nous allons voir.

Ce dialogue a pour but de :

a) Contrôler l'identité des deux interlocuteurs (les deux doivent appartenir à la même famille). A cet effet, chaque processus doit "réserver" la liaison. La variable rendez-vous sera assignée à deux processus de la même famille, de telle façon que, si seulement un des processus "libère" la liaison, un troisième processus (de la même famille) peut prendre sa place (à travers une réservation), et ouvrir une liaison explicite avec le premier processus.

Sur le descripteur d'un objet de mode rendez-vous et de type local il y a un champ avec la clef de la famille de processus à qui la variable rendez-vous est assignée. Nous désignons ce champ par "clef" du descripteur.

RESERVER (nom de la variable de mode liaison, clef) VIA (nom du paramètre de mode rendez-vous)

(*) - Dans notre mise-en-oeuvre la clef était un objet de mode chaîne-de-caractères.

b) Contrôler l'ouverture de la liaison explicite entre les deux interlocuteurs (qui ont déjà réservé la liaison). A cet effet, chaque processus doit envoyer (sur la variable rendez-vous) une demande d'"ouverture" de la liaison - sur cette demande est défini le nom réseau du processus demandeur.

La réponse à la première de ces deux demandes d'ouverture à être traitée, informe le processus demandeur qu'il doit attendre une demande d'ouverture d'une voie de communication (par exemple, un "flot" dans le réseau Cyclades), adressée par son interlocuteur.

La réponse à l'autre demande d'ouverture de liaison explicite, transmet au processus demandeur le nom réseau de son interlocuteur * pour qu'il puisse ouvrir une voie de communication avec celui-ci.

OUVRIR (nom de la variable de mode liaison)

Dialogue final

Ce dialogue est symétrique du dialogue initial. Chaque processus doit "fermer" la liaison explicite, et la "libérer".

FERMER (nom de la variable de mode liaison)

LIBERER (nom de la variable de mode liaison)

Quand une demande de libération est exécutée par le processus communication, celui-ci examine les demandes de réservation en attente d'être servies, et en élit une relative à un processus de la même famille que celui qui vient de libérer la liaison. Ce mécanisme permet à une famille de processus d'utiliser sans interruption une variable rendez-vous, comme nous avons déjà fait remarquer.

NOTE : Les instructions ouvrir, fermer et libérer spécifient seulement le nom de la variable de mode liaison, alors qu'elles impliquent des accès à la variable de mode rendez-vous. Le lien nécessaire, est établi par l'exécution de l'instruction réserver : sur le descripteur de la variable liaison est inscrit le nom de la variable rendez-vous, et réciproquement le descripteur de la variable rendez-vous repère la variable liaison.

(*) - Défini sur le champ "nom-interlocuteur" du descripteur de la variable rendez-vous.

3.4 NOUVEAUX TYPES DE MESSAGES. ADRESSE DE RETOUR

Les deux dialogues, prologue et épilogue, de la communication explicite, obligent à l'interprétation de nouveaux types de messages par le mécanisme de communication implicite :

- Dans le sens ascendant de l'arbre des procédures (sens fils-père) :

  réserver
  ouvrir
  fermer
  libérer

- Dans le sens descendant de l'arbre des procédures :

  réservation-exécutée - réponse à la demande réserver
  attendre-ouverture - première réponse à un interlocuteur qui a demandé l'ouverture
  vous-êtes-demandeur - deuxième réponse à un interlocuteur qui a demandé à l'ouverture
  fermeture-exécutée - réponse à la demande fermer
  libération-exécutée - réponse à la demande libérer

Dans le mécanisme de communication implicite, les messages qui circulent dans le sens descendant de l'arbre des procédures (messages de type réponse), sont diffusés à chaque noeud de l'arbre, pour toutes les adresses de retour qui y sont mémorisés.

Au contraire, les nouveaux types de messages descendants, ne doivent pas être diffusés. Ils doivent être transmis seulement vers la procédure "fille" qui l'attend - c'est-à-dire, qu'une seule des adresses de retour doit être servie.

Pour résoudre ce problème, nous associons à chaque descripteur d'un objet de mode rendez-vous et type param-ref, un générateur de références *. Quand un processus (père) reçoit un message ascendant (d'un fils), qui doit encore transiter (vers le processus grand-père) :

(*) - Par exemple, un générateur de nombres entiers séquentiels, module N suffisamment grand.

- Une référence (R) est générée, qui repère l'adresse de retour du message reçu (du fils), mémorisé sur la file du descripteur local (père) ;

- Le repère-origine du message transité (vers le grand-père), se compose de :

  . repère-destination du message reçu (du fils) ;
  . R ;

- R fait aussi partie de l'adresse de retour mémorisé dans le de descripteur (du grand-père) à qui s'adresse le message transité.

3.5 PROTOCOLE DE COMMUNICATION EXPLICITE

Quand un processus SIGOR interprète les instructions réserver, ouvrir, fermer et libérer, il délivre à son processus communication un message du type :

repère-origine := 0 (signale au processus communication que la demande est locale) ;
repère-destination := nom de la variable rendez-vous ;
fonction := l'identificateur de l'instruction interprétée (respectivement réserver, ouvrir, fermer, libérer) ;

si instruction interprétée est réserver, alors :

paramètres := clef ;

si instruction interprétée est ouvrir, alors :

paramètres := nom réseau du processus ;

Quand un processus SIGOR interprète les instructions lire ou écrire, il délivre à son processus communication un message tel que :

repère-origine := 0 ;
repère-destination := nom de la variable liaison, identifiée par l'instruction ;

si l'instruction interprétée est lire, alors :

fonction := lire ;
paramètres := nom du tampon récepteur ;

sinon (instruction écrire)

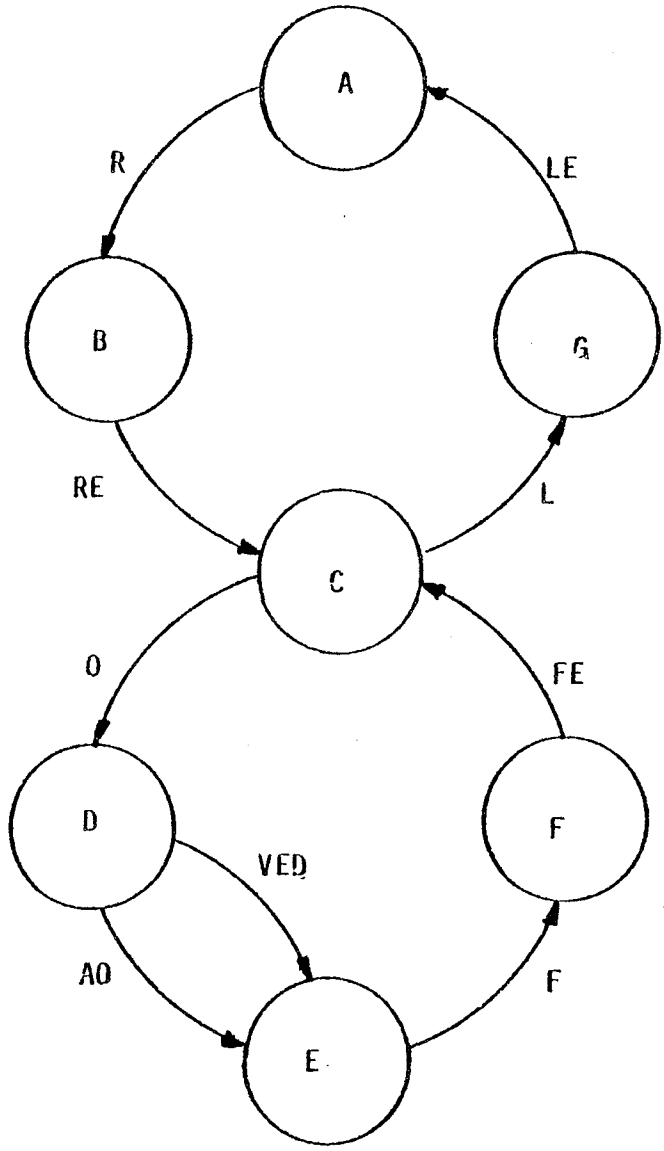
fonction := écrire ;
paramètres := texte ;

Le protocole de communication explicite peut être décrit par trois automates représentés par des graphes d'états finis) :

a) Deux automates "liaison" identiques, exécutés chacun par le processus communication de chaque processus SIGOR interlocuteur. L'état de cet automate est inscrit sur un champ du descripteur de la variable liaison correspondante, que nous désignons par "état-liaison".

b) Un automate "rendez-vous", exécuté par le processus communication du processus SIGOR pour qui la variable rendez-vous est locale. L'état de cet automate est inscrit sur un champ du descripteur de la variable rendez-vous correspondante, que nous désignons par "état-rendez-vous".

AUTOMATE LIAISON



Légende :

R - Réserver

RE - Réservation-exécutée

O - Ouvrir

AO - Attendre-ouverture

VED - Vous-êtes-demandeur

F - Fermer

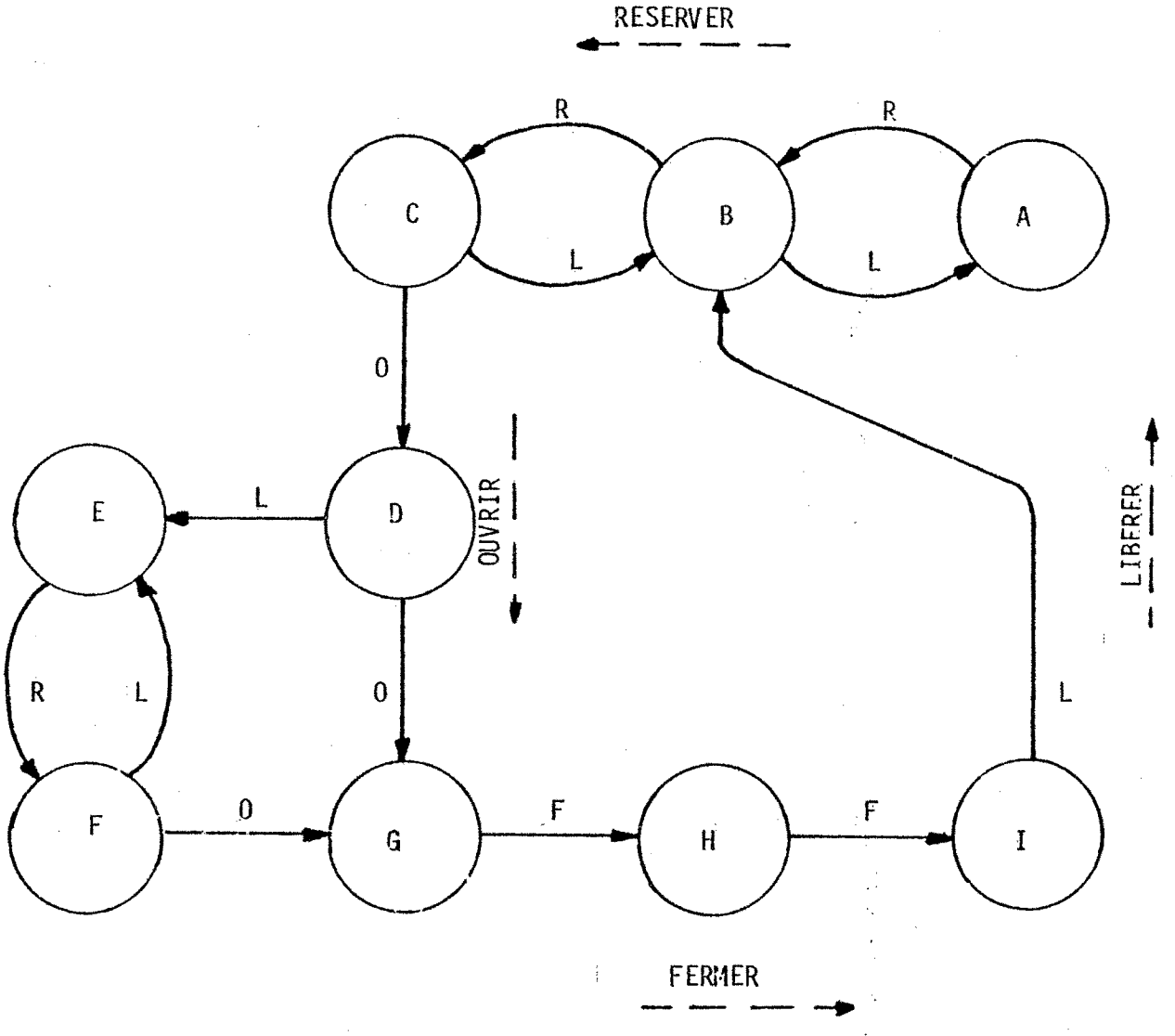
FE - Fermeture-exécutée

L - Libérer

LE - Libération-exécutée

FIGURE A-9

AUTOMATE RENDEZ-VOUS



Légende :

R - Réserver
O - Ouvrir
F - Fermer
L - Libérer

FIGURE A-10

Nous pouvons maintenant compléter l'algorithme de traitement d'un message par le processus communication, par rapport à la version présentée dans les pages A.15 et A.16 :

1. Recevoir un message (M) sur une liaison (locale ou réseau) du processus SIGOR ;

2. Si le descripteur repéré par M est de type param-ref, alors :

   Cas fonction de M est :

   1. LIRE : ...
   2. ECRIRE : ...
   3. REPONSE : ...
   4. RESERVER, OUVRIR, FERMER ou LIBERER :
      1. Si repère-origine de M = 0 (demande locale), alors :
         1. Contrôler graphe liaison :

            Cas (fonction de M ; état-liaison) est

            1. (réserver ; A) : état-liaison := B ;
            2. (ouvrir ; C) : état-liaison := D ;
            3. (fermer ; E) : état-liaison := F ;
            4. (libérer ; C) : état-liaison := G ;
            5. autre :

               Erreur :

               1. Signaler incohérence ;
               2. Avorter application SIGOR ;

      2. Mémoriser adresse de retour (fils ; repère ; ref-retour), sur le descripteur de la variable rendez-vous :
         1. fils := nom de la variable fils associée à la liaison implicite, sur laquelle est arrivé le message M ;
         2. repère := repère-origine du message M ;
         3. ref-retour := défini par le générateur de références du descripteur de la variable rendez-vous ;
      3. Faire transiter un message M' vers la procédure appelante (soit à travers une liaison locale, soit à travers une liaison réseau) :

1. repère-origine de M' := {repère-destination de M ; ref-retour de l'adresse de retour (cf. 2.4.2)} ;
2. repère-destination de M' := repère défini sur le descripteur de la variable rendez-vous ;
3. fonction et paramètres de M' := fonction et paramètres de M ;

5. RESERVATION-EXECUTEE, FERMETURE-EXECUTEE, LIBERATION-EXECUTEE, ATTENDRE-OUVERTURE ou VOUS-ETES-DEMANDEUR :
   1. Si repère-origine de M ≠ 0, alors
      1. Retirer de la file du descripteur, l'adresse de retour correspondant à la ref-retour du repère-destination de M
      2. Transmettre un message M' vers la procédure appelée, correspondant à l'adresse de retour (soit sur une liaison locale, soit sur une liaison réseau) :
         1. repère-destination de M' := {repère ; ref-retour} de l'adresse de retour ;
         2. fonction et paramètres de M' := fonction et paramètres de M ;
   2. Sinon (M = 0, et donc le message est arrivé à sa destination)
      1. Contrôler le graphe liaison :
         Cas {fonction de M ; état-liaison} est
         1. {réservation-exécutée ; B} : état-liaison := C ;
         2. {fermeture-exécutée ; F} : état-liaison := C ;
         3. {libération-exécutée ; G} : état-liaison := A ;
         4. {attendre-ouverture ou vous-êtes-demandeur ; D} : état-liaison := E ;
         5. autre : Erreur (≡2.4.1.1.5) ;
      2. Traiter demandes :
         Cas fonction de M est
         1. réservation-exécutée, ou fermeture-exécutée, ou libération-exécutée :
            1. réveiller processus SIGOR ;
         2. attendre-ouverture :
            1. processus communication (qui a accès à la ST), attend une demande d'ouverture d'une voie de communication (demandée par son interlocuteur) ;

2. Quand cette demande arrive, il établit la voie de communication ;
3. Réveiller le processus SIGOR ;

3. vous-êtes-demandeur :
1. Le processus communication (qui a accès à la ST), demande l'ouverture d'une voie de communication avec son interlocuteur, dont le nom réseau est paramètres de M ;

6. autre : Erreur (≡2.4.1.1.5) ;

3. Sinon (descripteur repéré par M, de type local ou param-val)

Cas fonction de M est

1. LIRE : ...

2. ECRIRE : ...

3. RESERVER :
Cas état-rendez-vous est
1. A :
1. état-rendez-vous := B ;
2. clef du descripteur := paramètres de M ;
3. acquitter demande, avec fct := réservation-exécutée et param := nil :
1. Construire message M' :
1. repère-destination de M' := repère-origine de M ;
2. fonction de M' := fct ;
3. paramètres de M' := param ;
2. Envoyer le message M' sur la liaison sur laquelle la demande initiale avait été reçue (liaison locale ou liaison réseau) ;

2. B (respectivement E) :
1. Si clef du descripteur ≠ paramètres de M, alors
1. Remettre M dans la file des messages reçus, pour traitement postérieur ;
2. Sinon
1. état-rendez-vous := C (respectivement F) ;

2. acquitter demande, avec fct := réservation-exécutée
et param := nil (≡3.3.1.3) ;

3. autre : Erreur (≡2.4.1.1.5) ;

4. OUVRIR
Cas état-rendez-vous est
1. C :
1. état- rendez-vous := D ;
2. nom-processus du descripteur := paramètres de M ;
3. acquitter demande, avec fct := attendre-ouverture et
param := nil (≡3.3.1.3) ;
2. D ou F :
1. état-rendez-vous := G ;
2. acquitter demande, avec fct := vous-êtes-demandeur et
param := nom-processus du descripteur
(≡3.3.1.3) ;
3. autre : Erreur (≡2.4.1.1.5) ;

5. FERMER :
Cas état-rendez-vous est
1. C (respectivement H) :
1. état-rendez-vous := H (respectivement I) ;
2. acquitter demande, avec fct := fermeture-exécutée et
param := nil (≡3.3.1.3) ;
2. autre : Erreur (≡2.4.1.1.5) ;

6. LIBERER :
Cas état-rendez-vous est
1. B (respectivement (C ou I), (D ou F)) :
1. état-rendez-vous := A (respectivement B, E) ;
2. acquitter demande, avec fct := libération-exécutée et
param := nil (≡3.3.1.3) ;
2. autre : Erreur (≡2.4.1.1.5) ;

7. autre : Erreur (≡2.4.1.1.5) ;

Il reste à spécifier le traitement des instructions LIRE et ECRIRE, correspondantes à l'utilisation de la liaison explicite.

Pour contrôler le flux à travers la liaison explicite, une demande d'écriture peut être servie seulement si l'interlocuteur a envoyé une demande de lecture. A cet effet (contrôle de flux) il y a un champ du descripteur de la variable liaison, désigné par "écritures", qui permet l'asservissement de l'écrivain au lecteur.

1. Recevoir un message (M) sur une liaison du processus SIGOR ;

2. Si le descripteur repéré par M est de type param-ref, alors

   ...

3. Sinon (descripteur repéré par M de type local ou param-val)

   Cas fonction de M est
   1. LIRE : ...
   2. ECRIRE : ...
   3. RESERVER : ...
   4. OUVRIR : ...
   5. FERMER : ...
   6. LIBERER : ...
   7. LIRE :
      1. Si repère-origine de M = 0 (donc le message est local), alors
         1. Envoyer un message M' vers l'interlocuteur (à travers la liaison réseau associée à la variable liaison identifiée par repère-destination de M) :
            1. repère-origine de M' := repère-destination de M ;
            2. fonction de M' := lire ;
      2. Sinon (M ≠ 0, donc le message est distant)
         1. le champ écritures de la variable liaison est incrémentée d'une unité ;

   8. ECRIRE :
      1. Si repère-origine de M = 0, alors
         1. Si le champ écritures de la variable liaison > 0, alors
            1. Envoyer un message M' vers l'interlocuteur :
               1. repère-origine de M' := repère-destination de M ;

2. fonction de M' := écrire ;
3. paramètres de M' := paramètres de M ;

2. le champ écritures de la variable liaison est décrémenté d'une unité;

2. Sinon
1. Remettre M sur la file des messages reçus, pour traitement postérieur ;

2. Sinon (repère-origine de M ≠ 0)
1. tampon récepteur (de l'instruction lire antérieur) := paramètres de M ;
2. Réveiller processus SIGOR ;

9. autre : Erreur (≡2.4.1.1.5) ;

ANNEXE B

NOTES BIBLIOGRAPHIQUES

Plan :

I) LES EXPRESSIONS DE CHEMIN

Référence bibliographique:

The specification of process synchronization by path expressions
R.H. CAMPBELL ; A.N. HABERMANN
Lecture Notes in Computer Science, Vol. 16, Springer Verlag, 1974

Les expressions de chemin constituent une méthode d'expression des règles de synchronisation, incorporée dans la définition du type décrivant les objets partagés par un ensemble de processus asynchrones.

Les deux structures fondamentales de cette méthode sont la séquence d'actions (une action est l'exécution d'une procédure par un processus) et la sélection d'une action parmi un ensemble d'actions.

SEQUENCE D'ACTIONS:

Supposons trois procédures p, q et r dont les exécutions doivent être séquentielles. Alors

p ; q ; r

c'est un exemple d'une expression de chemin qui exprime que les procédures p, q et r doivent être exécutéesl'une aprèsl'autre dans la séquence donnée. Les procédures peuvent être appelées par des processus, dans une ordre quelconque et à des instants quelconques.

SELECTION D'UNE ACTION:

Prenons l'exemple de trois procédures p, q et r dont l'exécution d'une seule peut être autorisée à un instant quelconque. Ceci s'exprime par

p , q , r

Ces deux structures (séquence et sélection), peuvent être combinées dans des expressions de chemin plus complexes du type:

p ; (q , r) ; s

Deux autres concepts de synchronisation, sont utils à représenter - répétition et exécution simultanée.

REPETITION D'UNE EXPRESSION DE CHEMIN:

Pour représenter qu'une expression de chemin doit être répétée, une fois terminée, nous la entourons par les mots clés path et end.

L'expression de chemin

path P end

spécifie l'exécution répétée de la procédure P par un ou plusieurs processus.

L'EXECUTION SIMULTANEE

L'exécution simultanée permet à un ensemble de processus, d'exécuter en parallèle un ensemble de procédures. L'exécution simultanée se représente en entourant la séquence de procédures spécifiée d'un pair de crochets ouvrant et fermant. L'expression de chemins

{ P }

autorise l'exécution de la procédure P par plusieurs processus, simultanément.

II) LES COMPTEURS

Référence bibliographique:

Towards autonomous descriptions of synchronization modules
P. ROBERT ; J.P. VERJUS
B. Gilchrist (ed), IFIP, North-Holland (1977)

Le module de contrôle pour synchroniser un ensemble de procédures P_1, P_2, ... , P_n, proposé par (ROB 77), contient deux types d'informations:

a) Un ensemble de files, où sont sauvegardées les requêtes d'exécution des procédures. Chaque procédure est assignée à une seule file, mais à une file peuvent être assignées plusieurs procédures.

Exemple:

queue: P_1, P_2, P_7 / P_3, P_5 / P_4, P_6 (ROB 77)

Toutes les requêtes pour P_3 et P_5 seront sauvegardées sur la même file.

A chaque instant un seul des éléments d'une file (non vide), est reconnu comme la tête de cette file.

b) Un ensemble de conditions qui expriment les règles de synchronisation. A chaque procédure est associée une condition.

Une requête d'exécution d'une procédure est autorisée si:

i - elle est la tête de la file correspondante à la procédure, et

ii - la condition associée à la procédure est vraie.

Une condition est une relation arithmétique d'un compteur composé d'un côté, et d'un entier (ou un module d'entier) de l'autre côté.

Un compteur composé est une combinaison linéaire des compteurs d'états avec des coeficients égaux à +1 ou -1.

Les auteurs proposent 5 variables qui sont les compteurs d'états:

$\rho(P)$: nombre total de requêtes (pour l'exécution de la procédure P) sauvegardées dès l'initialisation du module de contrôle;

$\alpha(P)$: nombre total d'autorisations d'exécution données dès l'initialisation;

$\lambda(P)$: nombre total de terminaisons d'exécution sauvegardées dès l'initialisation.

Les deux autres compteurs d'états, servent à faciliter la description des règles de synchronisation, et sont dépendants des précédents:

$\nu(P)$: nombre de processus qui exécutent actuellement la procédure P

$$\nu(P) = \alpha(P) - \lambda(P)$$

$\Phi(P)$: nombre de requêtes sauvegardées et non encore autorisées

$$\Phi(P) = \rho(P) - \alpha(P)$$

Exemple retiré de (ROB 77):

Considérons le problème d'un producteur P et d'un consommateur C, qui partagent un tampon de dimension 1. Les règles de synchronisation sont:

(i) le consommateur ne peut pas consommer plus que le producteur a produit, i.e. $\alpha(C) =< \lambda(P)$;

(ii) le nombre de productions autorisées ne peut pas dépasser le nombre de consommations terminées, i.e. $\alpha(P) =< \lambda(C) + 1$;

De (i), nous avons que la condition pour autoriser une consommation est $\alpha(C)<\lambda(P)$; de (ii) la condition pour autoriser une production est $\alpha(P)<\lambda(C)+1$.

Le module de contrôle est définit par:

```
PRODCON: control module
begin P, C: procedure;
    queue: P / C;
    condition (C): λ(P)-α(C)>0;
    condition (P): α(P)-λ(C)<1;
end;
```

III) CONTROLE DE CONCURRENCE DANS LES BASES DE DONNEES REPARTIES

Référence bibliographique:

System level concurrency control for distributed database systems
D.J. ROSENKRANTZ ; R.E. STEARNS ; P.M. LEWIS II
ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, Pages 178-198

La prévention d'étreintes fatales ou de phénomènes analogues, et la cohérence globale d'un ensemble d'objets partagés par des processus concurrents, sont des problèmes fondamentaux dans les systèmes répartis.

Ces types de problèmes ont été étudiés dans le domaine des bases de données réparties,notamment par Rosenkrantz, Stearns et Lewis (ROS 76) (ROS 78) .

Pour cet étude les auteurs ont considéré que les fonctions d'accès aux entités d'une base de données sont "lire" et "écrire", et donc les règles de synchronisation des accès sont bien définies et toujours les mêmes:

path {lire} , écrire end

Dans ces conditions ont été définis:

(i) un mécanisme de contrôle du parallélisme entre les processus concurrents (1), et

(ii) des conditions dans lesquelles deux demandes d'accès (requêtes de lecture ou d'écriture), faites par des processus différents, provoquent un conflit (2),

tels que si une requête provoque un conflit,le mécanisme de contrôle de concurrence (3) peut déterminer un "retour-arrière" dans l'évolution du système. C'est-à-dire, qu'un processus peut être reinitialisé, après que toutes les actions qu'ila exercées surle système (base de données) aient été "annulées".

Pour ce scénario sont définies les conditions (4) à satisfaire par le mécanisme de contrôle de concurrence, pour résoudre (partiellement ou totalement) (5) les deux problèmes considérés.

Notre proposition concerne des objets informatiques éventuellement autres que des entités des bases de données, c'est-à-dire dont les fonctions d'accès peuvent être différentes de "lire" ou "écrire". Il serait intéressant de généraliser l'étude citée (ROS 78), en ajoutant ce degré de complexité.

Il faudrait donc définir, indépendamment de la nature des accès:

a) Un mécanisme de contrôle du parallélisme. Nous pouvons adopter celui proposé dans (ROS 78) (1), si on considère qu'une fonction d'accès est

- de "type écriture" si elle spécifie au moins une affectation à la "valeur partageable" de la ressource;
- de "type lecture", le cas échéant.

b) Une condition de conflit entre les requêtes. Nous remarquons que le critère énoncé dans (ROS 78) (2) est intrinsèquement lié à la nature des accès "lire" et "écrire". Nous avons essayé de définir de tels critères à partir des caractéristiques du graphe de synchronisation. Par exemple, un critère pouvait être: une requête provoque un conflit si son prologue de contrôle n'est pas exécutable à l'état courant (cf. Règle I, page 26) - ce qui est une condition suffisante pour le critère de (ROS 78). Néanmoins nous avons trouvé des situations ou le prologue de contrôle d'une requête n'est pas exécutable à l'état courant, sans que cela représente logiquement aucun conflit. Nous n'avons donc pas plus développé cette voie (même si nous pensons revenir là dessus).

Une autre difficulté que se pose dans la généralisation de l'étude de (ROS 78), est la suivante:

Dans la proposition citée, si une requête provoque un conflit, le mécanisme de contrôle de concurrence peut déterminer que l'un des processus en conflit doit être reinitialisé, après avoir annulés toutes les mises à jour de la base de données réalisées par le processus. Cette action est exécutée à partir des copies de la valeur des entités de la base de données sauvegardées avant les modifications. En fonction de la simplicité du graphe de synchronisation des

accès"lire" et "écrire", et du critère de conflit, le problème de savoir sur quel état d'accessibilité revient l'entité, ne se pose pas.

Mais, dans le scénario que nous proposons ce sera un problème à résoudre. Nous pouvons le faire de deux manières:

a) Sauvegarder la "trajectoire" des accès sur le graphe de synchronisation, dans une pile associée à chaque ressource (identique à la pile d'exécution d'un processus, qui permet le contrôle des retours de procédures). Cette solution soulève d'autres problèmes (6) et surtout peut conduire à des systèmes encore plus lourds.

b) Sur le graphe de synchronisation doivent être spécifiées les routines de contrôle qui correspondent aux différents actions de "rollback" possibles. C'est une solution qui peut compliquer le graphe de synchronisation et le rendre inutilisable en tant qu'outil de spécification des règles de synchronisation.

Les deux problèmes:

- prévention d'étreintes fatales et phénomènes analogues, et
- garantie de la cohérence globale du système de ressources,

restent donc ouverts, et à notre connaissance ne sont pas résolus dans le scénario que nous proposons.

---

(1) - (ROS 78, page 183):
"We assume that the first request (whether a read or a write request) on an entity and the first write request (if the previous request were read request) on the entity by any given process invoke the concurrency control, which must select a response. We assume that any subsequent read request on the same entity by the process is automatically granted and results in the process seeing the same version of the entity as the last request. Any subsequent write request by the process on the entity is automatically granted and overwrites the version of the entity produced by the preceding write request".

(ROS 78, pages 181 et 182):
"The activity of a process can be stopped in two ways, either by termination or abortion.
Aborting a process consists of first stopping the running of the process at its active site, and then undoing all changes to the database made by the process. We use the term rollback to refer to the undoing of all the changes made by a process at a given site ... We assume that this rollback is caused by a "rollback" message sent from the active site to all visited sites.

Terminating a process consists of first stopping the running of the process at its active site and then making its changes to the database available to subsequent processes. ... We assume that this modification is made by a "terminate" message sent from the active site to all visited sites".

(2) - (ROS 78, page 184):
"We say that two requests by different processes on a given entity are in conflict if one of the requests is a write request and the site containing the entity has not received a termination message or rollback message for either process".

(3) - (ROS 78, page 179):
"The concurrency control is the portion of the system that is concerned with deciding what actions should be taken in response to requests by the individual processes to read and write into the database".

(4) - (ROS 78, page 184):
"We say that a concurrency control is strict if it is designed so that a request is never granted at a time when the request of another with a granted request of another process.

(ROS 78, page 186):
"Theorem 3. Suppose a concurrency control has the property that a request is not granted unless a termination message or rollback message has been received for each of the processes it caused a conflict with. Then the concurrency control is strict".

(ROS 78, page 188):
"We say that a concurrency control is superstrict if
(1) a request is not granted unless a termination message or rollback message has been received for each of the processes it caused a conflict with, and
(2) a process is only made to wait for processes that it causes a conflict with.

(5) - (ROS 78, page 184):
"Theorem 1. For a strict concurrency control, the current database is obtainable by a linearization of the terminated processes, ordered by their time of termination ... (linearization is a sufficient condition for a concurrency control to preserve consistency)".

(ROS 78, page 185):
"Theorem 2. For a strict concurrency control, no process can run forever".

(ROS 78, page 189):
"Theorem 4. If a superstrict concurrency control is not subject to deadlock, then no scenario can contain a process that waits forever".

(ROS 78, page 191):
"Theorem 5. If a superstrict concurrency control is not subject to deadlock and has restart based on a valid numbering method, then every process terminates".

(6) - La "trajectoire" des accès d'une transaction (service) qui se termine, peut être effacée de la file. Mais que se passe-t-il si cette "trajectoire" a des intersections non vides avec d'autres "trajectoires" d'autres services pouvant être encore en exécution. Comment et quand cette "trajectoire" pourra-t-elle effectivement être effacée?

BIBLIOGRAPHIE

AKK 78 — Conditions for the equivalence of synchronous and asynchronous systems
Eralp A. AKKOYUNLU ; Arthur J. BERNSTEIN ; Fred B. SCHNEIDER ; Abraham SILBERSCHATZ
IEEE Transactions on Software Engineering
Vol. SE-4, No. 6, Nov 78

AND 78 — Contribution à l'expression des interactions entre processus
F. ANDRE ; J.P. BANATRE ; D. HERMAN ; M. RAYNAL ; J.P. VERJUS
IRISA - Publication Interne n° 110, Dec 1978

BAN 78 — Contrôle de l'accès aux objets partagés dans les systèmes informatiques
Jean-Serge BANINO ; Jean FERRIE ; Claude KAISER ; Didier LANCIAUX
Monographies d'Informatique de l'AFCET
Editions Hommes et Techniques - 1978

BAN 79 — Synchronization for distributed systems using a single broadcast channel
J.S. BANINO ; C. KAISER ; H. ZIMMERMANN
First International Conference on Distributed Computing Systems, Alabama, 1979

BEK 75 — A comparison of two high level synchronizing concepts
Y. BEKKERS
Dept. of Computer Science - The Queen's University of Belfast

BEK 77 — Expression et décomposition du contrôle du parallélisme dans un système
Y. BEKKERS ; J. BRIAT ; J.P. VERJUS
INPG, Rapport de Recherche n° 66, Jan 1977

BRI 72a — Structured multiprogramming
Per Brinch Hansen
Comm ACM 15, 7 (Jul 1972), 574-578

BRI 72b — A comparison of two synchronizing concepts
Per BRINCH HANSEN
Acta Informatica, 1, 3 (1972), 190-199

BRI 78 — Distributed processes: A concurrent programming concept
Per BRINCH HANSEN
Comm ACM, 21, 11 (Nov 1978), 934-941

CAM 74 — The specification of process synchronization by path expressions
R.H. CAMPBELL ; A.N. HABERMANN
Lecture Notes in Computer Science, Vol. 16, Springer Verlag

CHE 79 — Un projet de système distribué: SORTILEGE
J.L. CHEVAL ; C. CLEMENCET ; J.M. GUILLOT ; P. LAFORGUE ; J. MOSSIERE ; J. RAYMOND ; S. ROUVEYROL
INPG, Rapport de Recherche n° 150, Jan 1979

CHU 77 — Répartition d'applications et de bases de données sur un réseau général d'ordinateurs
J.C. CHUPIN
Thèse de doctorat d'état. USMG. INPG. Grenoble

CRO 75 — Systèmes d'exploitation des ordinateurs
CROCUS
Dunod - Série Informatique, 1975

DAN 78a — Système et langage portable pour le traitement des applications réparties
Ng.X. DANG
Thèse Docteur 3ème Cycle, INPG

DAN 78b — Système d'interprétation généralisée orientée réseau: SIGOR
Spécifications de définition
Ng.X. DANG ; J.G. MARTINS ; G. SERGEANT ; P. WILMS
Rapport Interne (I.M.A.G.) - 1978

DAN 78c — Le macro langage d'utilisation de SIGOR: LM
Spécifications de définition
Ng.X. DANG ; J.G. MARTINS ; G. SERGEANT ; P. WILMS
Rapport Interne (I.M.A.G.) - 1978

DAN 79a — SIGOR, système d'interprétation généralisée orientée réseau, un outil pour l'écriture et la mise au point d'applications réparties
Ng.X. DANG ; J.G. MARTINS ; G. SERGEANT
Journées BIGRE - Nancy - 1979

DAN 79b — Communication between distributed processes by sharing non-duplicated objects
Ng.X. DANG ; M. DELAUNAY ; J.G. MARTINS
Euro IFIP 79, P.A. Samet (ed)
North-Holland Publ. Company, 1979

DIJ 68 — Cooperating sequential processes
Edsger W. DIJKSTRA
Programming Languages, F. Genuys (ed),
Academic Press, New York, 1968

DIJ 72 — Hierarchical ordering of sequential processes
Edsger W. DIJKSTRA
Operating System Techniques, Hoare and Perrot (ed),
Academic Press, London, 1972

DIJ 75 — Guarded commands, nondeterminacy and formal derivation of programs
Edsger W. DIJKSTRA
Comm ACM, 18, 8 (Aug 1975), 453-457

GER — Process synchronization by counter variables
A.J. GERBER
Dept. of Computer Science, University of Sidney

GMA 77 Communication implicite entre des processus répartis sur un réseau hétérogène
J.G. MARTINS
DEA Génie Infonmatique - INPG - Grenoble (Sept 77)

GRA 79 Access control in parallel programs
James R. McGRAW ; Gregory R. ANDREWS
IEEE Transactions on Software Engineering
Vol. SE-5, No. 1, Jan 1979

GUI 79a A synchronizing mechanism for distributed applications
J.M. GUILLOT
INPG, Rapport de Recherche n° 158, Fev 1979

GUI 79b Proposition pour un langage d'écriture de programmes répartis
Expression du contrôle et de la communication entre processus distribués
J.M. GUILLOT
Thèse 3ème Cycle - USMG - Grenoble (Dec 1979)

HAB 72 Synchronization of communicating processes
A.N. HABERMANN
Comm ACM, 15, 3 (Mar 1972), 171-176

HEW 79 Specification and proof techniques for serializers
C.E. HEWITT ; R.R. ATKINSON
IEEE Transactions on Software Engineering
Vol. SE-5, No° 1, Jan 1979

HOA 72 Towards a theory of parallel programming
C.A.R. HOARE
Operating System Techniques, Hoare and Perrot (ed),
Academic Press, London, 1972

HOA 74 Monitors: An operating system structuring concept
C.A.R. HOARE
Comm ACM, 17, 10 (Oct 1974), 549-557

HOA 78 Communicating sequential processes
C.A.R. HOARE
Comm ACM, 21, 8 (Aug 1978), 666-677

KAI 78 Système informatique répartie pour la conduite d'un atelier de mécanique
C. KAISER; M. KRONENTAL; J. LANGET; S. NATKIN; S. PALASSIN
Congrès de l'AFCET
Gif-sur-Yvette, 13-15 Nov. 1978

KEL 76 Formal verification of parallel programs
R.M. KELLER
Comm ACM, 19, 7 (Jul 1976), 371-384

KES 77 An alternative to event queus for synchronization in monitors
J.L.W. KESSELS
Comm ACM, 20, 7 (Jul 77), 500-503

LAM 79 Experience with processes and monitors in Mesa
Butler W. LAMPSON ; David D. REDELL
Xerox - Palo Alto Research Center, Apr 1979

LAU 78 On the duality of operating system structures
Hugh C. LAUER ; Roger M. NEEDHAM
Xerox - Palo Alto Research Center, 1978

PAR 72 On the criteria to be used in decomposing systems into modules
D.L. PARNAS
Comm ACM, 15, 12 (Dec 1972), 1053-1058

PUL 78 Un outil pour la spécification de la synchronisation dans les langages de haut-niveau
J. POLOU
INPG, Rapport de Recherche n° 107, Jan 1978

ROB 77 Toward autonomous descriptions of synchronization modules
P. ROBERT ; J.P. VERJUS
B. Gilchrist (ed), IFIP, North-Holland (1977)

ROS 76 Concurrency controls for database systems
R.E. STEARNS ; P.M. LEWIS ; D.J. ROSENKRANTZ
Proc. 17th Annual Symp. on Foundations of. Comptr. Sci.
Tex., Oct 1976, pp. 19-32

ROS 78 System level concurrency control for distributed database systems
D.J. ROSENKRANTZ ; R.E. STEARNS ; P.M. LEWIS
ACM Transactions on Database Systems, Vol.3, No°2, June 1978

STE Synchronization primitives and the verification of concurrent programs
STEN ANDLER
Dept. of Computer Science, Carnegie-Mellon University

STE 78 Predicate path expressions
STEN ANDLER
Thesis Proposal
Dept. of Computer Science, Carnegie-Mellon University
Jul 1978

WIL 79 Etude et comparaison d'algorithmes de maintien de la cohérence dans les bases de données réparties
Paul WILMS
Thèse Docteur-Ingénieur - INPG - Grenoble (Nov 1979)

WIR 71 The programming laguage Pascal
N. WIRTH
Acta Informatica, 1, 35-63 (1971) Springer-Verlag

WIR 73 An axiomatic definition of the programming language Pascal
N. WIRTH
Acta Informatica, 2, 335-355 (1973) Springer-Verlag

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU les rapports de présentation de Messieurs :

- J.S. BANINO, responsable permanent du projet "Analyse et Conception de Systèmes Répartis"à l'INRIA - LE CHESNAY -

Cl. DELOBEL, Professeur à l'Université Scientifique et Médicale de GRENOBLE

NGUYEN XUAN DANG, Assistant à l'Université des Sciences Sociales de GRENOBLE

Monsieur José DA GRAÇA MARTINS

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR-INGENIEUR, spécialité "Génie Informatique".

Grenoble, le 19 Mars 1980

Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique