Expression de la synchronisation dans les systèmes informatiques et conception d'architectures tolérant les pannes

Jacques Pulou

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
«Génie Informatique»

par

Jacques PULOU

EXPRESSION DE LA SYNCHRONISATION DANS LES SYSTEMES INFORMATIQUES ET CONCEPTION D'ARCHITECTURES TOLERANT LES PANNES.

Thèse soutenue le 25 septembre 1979 devant la commission d'examen

Monsieur BOLLIET — Président

Messieurs ALEONARD
COSTES — Examinateurs
KRAKOWIAK
Madame SAUCIER

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1977-1978

Président : M. Philippe TRAYNARD

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

PROFESSEURS TITULAIRES

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

* en congé pour études

PROFESSEURS SANS CHAIRE

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

.../...

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

MAITRES DE CONFERENCES

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

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

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

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

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

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

.../...

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

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

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

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

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

Je remercie

Monsieur L. BOLLIET, Professeur à l'Université de Grenoble qui m'a fait l'honneur de présider le jury de cette thèse,

Monsieur C. ALEONARD, Ingénieur à la Société CROUZET, qui a accepté de juger ce travail,

Monsieur A. COSTES, Maître de Conférences à l'Institut National Polytechnique de Toulouse, qui a bien voulu s'intéresser à ce travail et qui, par ces remarques, a contribué à son amélioration,

Monsieur S. KRAKOWIAK, Professeur à l'Université de Grenoble, qui a accepté de faire partie de ce jury,

Madame G. SAUCIER, Maître de Conférences à l'Institut National Polytechnique de Grenoble, qui m'a accueilli dans son équipe et m'a toujours témoigné sa plus grande confiance,

Je tiens également à remercier mes camarades de l'équipe "Conception et Sécurité des Systèmes en particulier

Monsieur P. CASPI pour notre collaboration amicale depuis maintenant plus d'une année,
Monsieur M. MOALLA pour l'amitié et l'attention qu'il m'a toujours témoignées,

Je voudrais également remercier tous ceux qui ont contribué au développement des idées contenues dans ce mémoire et en particulier Monsieur J. SIFAKIS, ainsi que tous mes amis qui m'ont soutenu durant la réalisation de ce travail, en particulier P. BROUQUET et J. SERBIELLE,

Je ne voudrais pas oublier Madame G. DUFFOURD qui a dactylographié ce mémoire, et le service de reprographie de l'IMAG qui en a assuré le tirage.

A mes parents
à ma soeur,
à tous mes amis ...

... et aux belles mouchet
que ce travail a sauvées

PRESENTATION

Le travail présenté dans ce mémoire comporte deux parties :

- Une étude de l'expression de la synchronisation dans les systèmes d'exploitation.
- Quelques éléments méthodologiques dans la conception de systèmes tolérant les pannes, illustrées par un exemple.

La première partie comprend une classification des outils de synchronisatic les plus courants et propose une description formelle de la synchronisatior ainsi qu'un outil facilement insérable dans un langage d'écriture de systè

La seconde partie propose quelques éléments méthodologiques pour la conception de systèmes tolérant les pannes construits en vue d'une application donnée.
Les progrès importants réalisés dans les technologies d'intégration ont amené la fabrication d'unités fonctionnelles monolithiques de taille importante. Ces unités fonctionnelles peuvent être facilement interconnectées pour former des architectures à la demande.

Cette aptitude permet d'envisager la réalisation d'architectures répondant aux contraintes d'applications précises. Cette approche devient une alternative à l'utilisation de machines d'usage général pour la conception d'architectures particulières.

Notre méthode vient compléter cette approche "par architectures spécialisée en permettant la prise en compte de contraintes de sûreté de fonctionnement au cours de la conception de tels systèmes.

Dans le cadre de cette méthode nous proposons une technique originale de redondance applicable à un type d'architectures spécialisées : applications fonctionnellement réparties sur un réseau de calculateurs.
Cette méthode et cette technique de redondance sont conjointement appliquées à la conception d'une centrale anémométrique embarquée.

- SOMMAIRE -

PARTIE A

SYNCHRONISATION

PARTIE A : SYNCHRONISATION

I - PARALLELISME ET SYNCHRONISATION

La modélisation apparaît aujourd'hui indispensable à la maîtrise des systèmes informatiques ou logiques complexes.

Les modèles seront tout particulièrement utiles pendant la conception des systèmes.
A chacune des étapes de cette dernière, le concepteur se trouve aux prises avec des problèmes susceptibles de recevoir plusieurs solutions; à chacune de ces étapes, l'utilisation d'un modèle adapté au problème à résoudre, facilitera la recherche de solutions correctes par une représentation directe et, voire même, par une évaluation qualitative ou quantitative de l'impact de ce choix sur les performances du système.

Mais l'usage de modèles se révèle également indispensable à l'explication et à la compréhension d'un système.
Chaque modèle propose un schéma, une représentation d'un système, permettant de mieux en appréhender un aspect.
Plusieurs modèles contribuent ainsi à l'acquisition d'une vision globale du fonctionnement du système étudié.

La définition de modèles adaptés, ainsi que des liaisons entre ces modèles, semble donc être aujourd'hui une activité d'importance primordiale.

En ce qui concerne les modélisations des machines informatiques, on en distingue classiquement plusieurs types, chacun correspondant à un niveau différent :

- niveau "transfert de registres" : CASSANDRE [ANC 69],...,
- niveau "interpréteur d'instructions": ISP [BELL 71],...,
- niveau "système d'exploitation" : modèles à files d'attente [POT 77
- .....

A la plupart de ces niveaux de description le système se présente comme le siège de plusieurs "activités" dont les exécutions se recouvrent dans le temps. Ces activités parallèles ne sont généralement pas indépendantes :

- elles se partagent les ressources du système;
- elles coopèrent en vue de la réalisation des fonctions de ce même système.

Ces liaisons entre activités impliquent certaines relations entre leurs exécutions; ce sont ces relations que nous appelerons contraintes de synchronisation.

Ce parallélisme dans la description peut simplement traduire la présence de parallélisme dans le système lui-même. Il a souvent l'efficacité comme cause première :

- soit pour une meilleure utilisation des ressources : chaque activité du système ne nécessitant pas simultanément toutes les ressources, une meilleure utilisation de ces dernières est atteinte grâce à l'exécution parallèle de plusieurs activités. Une telle technique sera souvent mise en oeuvre pour utiliser au maximum les ressources les plus coûteuses;
- soit pour améliorer les performances (architecture pipte-line des grosses machines).

Mais ce parallélisme peut être un outil de modélisation utile, voire indispensable à la maîtrise et à la compréhension des systèmes.

Cet aspect est particulièrement important en phase conception; comme exemples dans le domaine des systèmes d'exploitation, on peut citer la conception sur calculateur monoprocesseur de systèmes interactifs multiconsole ou de systèmes de commande en temps réel d'un procédé industriel découpé en unités partiellement indépendantes. La conception de tels systèmes sera souvent grandement facilitée par une décomposition préalable en activités parallèles,calquée sur celle du milieu extérieur. Un émulateur de ces activités parallèles sur le calculateur achèvera alors la réalisation projetée.

Nous nous intéresserons dans la suite au parallélisme apparaissant lors de la conception et de l'écriture des systèmes d'exploitation et, plus particulièrement, à l'expression de la synchronisation au sein de ces logiciels.

II - PARALLELISME DANS LES SYSTEMES D'EXPLOITATION

II - 1. Multiprogrammation - multitraitement

Le parallélisme apparaît clairement dans la conception des systèmes d'exploitation, à partir de l'introduction des calculateurs de la troisième génération.
Le travail le plus ancien couramment cité [DIJK 65] dans les bibliographies concernant ce sujet ([MUNTE 78],[CROCU 75],[ANDL 78],[MOSS 77]) est légèrement postérieur à l'introduction de la série IBM 360, célèbres représentants de cette génération, qui date des années 1963-64 [AMDHA 64].

Troisième génération

L'augmentation rapide de la puissance de l'unité centrale à la fin des années cinquante et durant les années soixante, du fait des progrès technologiques, a produit une différence de plus en plus grande entre la vitesse de traitement (centaines de milliers à quelques millions d'instructions à la seconde) et la vitesse des opérations d'entrées/sorties (lecture d'un enregistrement sur disque en quelques millisecondes, écriture d'une ligne sur terminal en plusieurs secondes).
On se devait donc "d'économiser" l'unité centrale,ressource coûteuse, en la réservant au seul traitement. L'introduction de dispositifs permettant une exécution simultanée des opérations de traitement et d'entrées/sorties était nécessaire.

La solution retenue consiste à introduire un nouveau type de processeur (le canal) exclusivement réservé aux opérations d'entrées/sorties et exécutant des programmes (programmes canaux) également implantés en mémoire centrale.
Le fonctionnement parallèle de ces processeurs découle de leurs accès simultanés en mémoire centrale. Cette simultanéité est obtenue par le découpage de cette dernière en blocs indépendants.
La baisse relative du coût des processeurs par rapport à celui de la mémoire a favorisé l'augmentation de leur nombre. On atteignait ainsi une meilleure utilisation de la ressource de coût élevé par la multiplication du nombre d'unités de coût plus faible.

A l'issue de ces évolutions successives on aboutit à des architectures schématisées par la figure II.1 pour les machines grosses ou moyennes, à partir du milieu des années soixante et durant les années soixante dix.

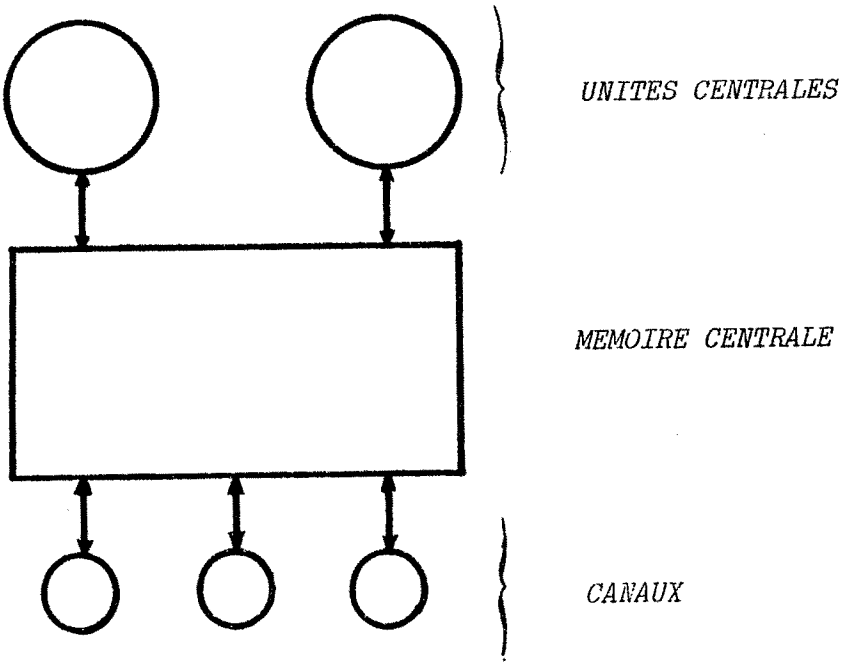


Figure II.1

Multiprogrammation

Une fois cette architexture définie,il reste donc a concevoir un système d'exploitation capable d'utiliser le parallélisme potentiel entre traitement(s) et opérations d'entrées/sorties offert par ces machines. Le partage des unités centrales entre plusieurs programmes ou multiprogrammation permet d'atteindre cet objectif.

Un dispositif logiciel, l'allocateur de processeurs, pilote ce multiplexage des unités centrales en allouant ces dernières à des programmes préalablement chargés en mémoire. Ce dispositif est activé par l'occurrence d'événements qui peuvent être soit synchrones à l'exécution des programmes (demandes d'entrée/sortie, déroutements etc),soit asynchrones (interruptions de fin d'entrée/sortie, interruptions horloge temps réels,...). A l'issue de ces événements, le système d'exploitation, après avoir éventuellement lancé une opération d'entrée/sortie, activera l'allocateur de proces seur qui attribuera alors l'unité centrale libérée à un des programmes

présents en mémoire. Cette technique permet ainsi l'utilisation parallèle des processeurs de traitement (unités centrales) et des processeurs d'entrées/sorties.
Si l'on observe alors la population des programmes introduits dans le système on s'aperçoit que plusieurs d'entre eux sont simultanément en cour d'exécution.
Dans le cas d'une unité centrale unique on parle alors d'exécution quasi parallèle des programmes. En effet, l'unité centrale étant unique il n'y a pas de traitement parallèle proprement dit. Le seul parallélisme réel étant l'exécution simultanée d'un programme par l'unité centrale et d'une ou plusieurs opérations d'entrée/sortie (programmes canaux) dans les processeurs d'entrée/sortie (canaux).
Le traitement parallèle réel appelé multitraitement n'est possible qu'avec plusieurs unités centrales.

II - 2. Parallélisme dans les systèmes d'exploitation : les processus

L'utilisation efficiente des machines de la troisième génération passait donc par l'écriture de systèmes d'exploitation multiprogrammés. La notion de processus séquentiel [DIJK 67] a permis de dégager une méthodologie de conception de tels systèmes.
Nous définissons,brièvement, un processus comme l'exécution d'un programme et nous renvoyons le lecteur à l'abondante littérature concernant ce sujet [HANS 70],[CROCU 75],[BELP 74] pour une présentation plus détaillée.

Un processus peut être concrètement représenté par l'union d'un programme et d'un contexte d'exécution ou état du processus. Un contexte renferme toutes les informations nécessaires à la reprise d'un processus, une fois celui-ci suspendu par le dispositif de multiplexage des unités centrales.
Une méthodologie classique [HANS 70],[CROCU 75],[DIJK 68]de conception d'un système multiprogrammé débute par l'écriture préalable d'un noyau K, extension logicielle de la machine nue H, transformant ces calculateurs en machines à processus .

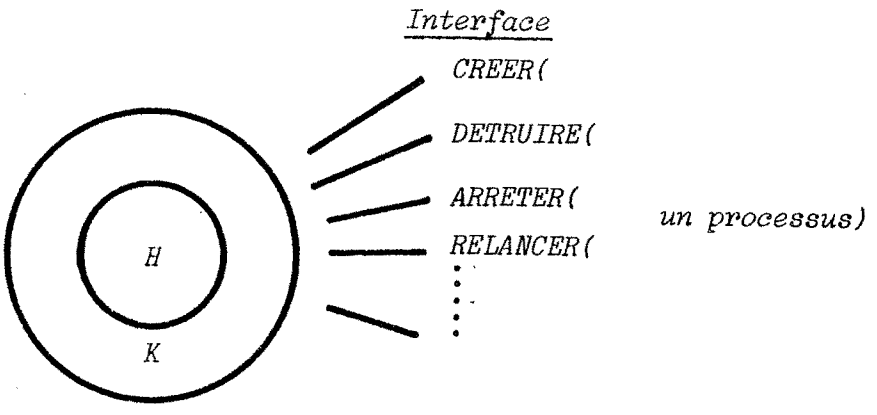


Figure II.2

Un tel noyau fournit alors les fonctions nécessaires à la gestion d'un ensemble de processus séquentiels s'exécutant parallèlement (multi) ou quasi-parallèlement (monoprocesseur) (Figure II.2).
Dans une seconde phase, le concepteur procèdera à la décomposition du système en un ensemble de processus séquentiels parallèles (THE:[DIJK 68 [HANS 70], ESOPE : [CROCU 75] & [MOSS 77]). Le noyau forme l'infrastructure de ces systèmes d'exploitations. Dans ces projets, le processus séquentiel s'est révélé en tant qu'outil de décomposition et de structuration des systèmes.

Cette exécution en parallèle de plusieurs processus utilisant des données communes amène certains problèmes :

- concurrence de l'accès à une donnée : problème de partage (exclusion mutuelle..) et d'allocation (priorité, stratégie..),
- coordination entre processus coopérant pour la réalisation d'une fonction globale du système.

Les règles qui vont régir ces interactions entre processus s'appellent règles de synchronisation.
Ces règles établissent, en effet, des relations (contraintes de synchronisation) entre les processus du système.

Nous avons vu précédemment que l'avancement des processus (i.e. l'allocatic des unités centrales aux processus) est en partie rythmée par l'occurrence d'événements asynchrones au déroulement des processus (interruptions d'entrée/sortie, horloge..) et par conséquent difficilement prévisibles lors de l'écriture du programme d'un processus. Cette difficulté a rendu nécessaire l'expression de la synchronisation indépendamment de la réalisation de l'allocation des unités centrales et, par conséquent, de l'occurrence de ces événements asynchrones.

Cette disposition amène deux avantages :

- Elle simplifie la tache du concepteur de systèmes en lui permettant de faire abstraction de la réalisation du noyau et de travailler à partir de spécifications externes simplificatrices :
  - . un processus n'attendant qu'une unité centrale se la verra attribuer au bout d'un temps fini : "tous les processus avancent à une vitesse non nulle";
  - . un processus peut être suspendu à "tout moment" : i.e à tous les points interruptibles de son programme.

  La synchronisation inter-processus devra donc être assurée sous ces seules hypothèses.
- L'indépendance entre réalisation du noyau et programmation du système proprement dite, permet de modifier une de ces deux parties indépendamment de l'autre : plusieurs systèmes peuvent ainsi être construits à partir du même noyau, et un même système peut être piloté par des noyaux différents proposant des stratégies d'allocation d'unité centrale distinctes.

Cette méthodologie de décomposition conduit donc à une architecture particulière du système d'exploitation qui implique notamment le respect de règles de synchronisation entre processus. Il convenait donc de rechercher des méthodes de programmation de ces synchronisations et, le cas échéant, de définir des outils adaptés à cette programmation.

N.B. La référence [DEN 71] donne une vue très complète sur ces systèmes.

II - 3. Présentation et classification d'outils de synchronisation classiques

L'abondance des travaux sur ce sujet (voir bibliographies de [ANDL 78], [MUNTE 78] & [MOSS 77]...)témoigne de l'intensité des recherches dans ce domaine durant les dix dernières années environ.
Dans la suite nous présenterons, dans un ordre chronologique, les outils les plus marquants exposés dans ces travaux. Au cours de cette revue nous tenterons de dégager pour chaque outil un ensemble de motivations justifiant leur introduction.
Pour chacun de ces outils nous donnerons la vision du problème de synchronisation induite par leur emploi : (nature des objets manipulés, type des interactions entre ces objets...), ainsi que quelques remarques portant sur la clarté de l'expression et sur l'efficacité résultant de leurs utilisation. Ces deux derniers critères correspondent à ceux retenus par d'autres travaux [ANDL 78], [MOSS 77], [PULOU 77], [MUNTE 78].
En conclusion, nous résumerons cette présentation par une classification à priori de ces outils en fonction des critères précédents.

Remarques préliminaires :

- Le déroulement des processus doit respecter certaines règles de synchronisation. Si à un instant donné, la poursuite d'un processus risquai de provoquer la violation d'une de ces règles, il conviendrait de stopper ce processus, pour le relancer ultérieurement, à un instant où ce risque aura disparu. En conséquence, si tous les outils de synchronisation permettent les opérations élémentaires "d'arrêt" et de "réveil" de processus, ils se distinguent dans l'expression de ces opérations et par les liaisons qu'ils proposent entre ces dernières et les programmes respectifs des processus.

- Nous ne considérons pas les primitives de sychronisation directes [CROCU 75].Celles-ci permettent à un processus d'arrêter ou de réveiller un autre processus de façon asynchrone à son déroulement (i.e sans que le programme du processus ainsi piloté le spécifie). A cette limitation nous donnons trois raisons :

. La mise en oeuvre effective de tels mécanismes n'évite pas la résolution d'un problème de synchronisation majeur que nous étudierons par la suite : l'exclusion mutuelle.

. La validité de synchronisation utilisant ces primitives risque de reposer, dans certains cas, sur des hypothèses concernant les vitesses relatives des processus, notamment lorsque le point d'arrêt d'un processus n'est pas indifférent.

. Ces primitives sont, à notre avis, adaptées à la mise en oeuvr de hiérarchie de processus (exemple : sous-système multiprogrammé : génération, contrôle et destruction d'un ensemble de processus fils par un seul processus père) et semblent en contradiction avec la méthode de décomposition des systèmes en processus parallèles concurrents, à priori sans prérogative les uns sur les autres.

II - 3.1. Exclusion mutuelle et sémaphores

Le problème d'exclusion mutuelle, [DIJK 65], outre son importance historique en tant que premier problème de synchronisation étudié, présente un intérêt théorique et pratique important : pratique, car les systèmes d'exploitation réels en offrent de nombreux exemples, et théorique en étant, comme nous le verrons, un problème "central" pour la solution des autres schémas de synchronisation entre processus.
Pour donner un énoncé réaliste de ce problème nous considèrerons deux processus P_1, P_2 (de programmes respectifs p_1 et p_2) accédant à une même ressource (périphérique, fichier, base de donnée) en exécutant deux sections de code, respectivement SC_1 et SC_2. Le maintien de la cohérence des informations contenues dans la ressource (fichiers, base de donnée..) ou des contraintes physiques (périphérique) imposent que, à tout instant, au plus un seul des deux processus (P_1, P_2) accède à la ressource.

Le problème se ramène donc à l'écriture, pour chaque section de code SC1, SC2, d'un prologue et d'un épilogue: <entrée 1> et <sortie 1> (respectivement <entrée2> et <sortie2>) qui satisfassent aux quatre conditions suivantes : [DIJK 67] (Figure II.3).

a) A tout instant un seul processus au plus exécute sa section critique (noté SC) : respectivement SC_1 et SC_2.

b) Si p_1 et p_2 sont bloqués en attente d'exécution de leur SC, l'un d'eux doit pouvoir y "entrer" au bout d'un temps fini.

c) Le blocage d'un des processus hors de la SC ne doit pas compromettre l'entrée de l'autre en SC.

d) Aucun processus ne doit jouer un rôle privilégié.

P_1 : p_1	P_2 : p_2
⋮	⋮
<entrée 1> ;	<entrée 2 > ;
$<SC_1>$;	$<SC_2>$;
<sortie 1>;	<sortie 2> ;

Figure II.3

REMARQUE

Toutes les machines fournissent des "exclusions mutuelles"élémentaires
- les instructions sont (en majorité) ininterruptibles:une interruptio n'est prise en compte par un processeur qu'à la fin de l'exécution d'une instruction,
- le mécanisme matériel de partage de la mémoire centrale entre processeurs garantit "l'exclusion mutuelle" entre les accès mémoire de ces derniers : un processeur ne peut donc "lire" ou "écrire" un mot mémoire pendant qu'un autre processeur le "lit" ou le "modifie".

Deux types de solutions sont, a priori, envisageables pour l'écriture des séquences de prologue et d'épilogue : celles qui ne font pas intervenir le noyau et celles qui le font intervenir.

- Solutions ne faisant pas intervenir le noyau : ces solutions consis à réaliser l'exclusion mutuelle en utilisant simplement le répertoi d'instructions de la machine nue.

  . Solution avec instructions de chargement et de lecture : "LOAD et "STORE". Une telle solution a été découverte et prouvée par Dijkstra [DIJK 65] .

  . Solution avec des instructions spéciales ininterruptibles perm tant une lecture et une écriture successives en mémoire (TEST and SET;IBM 360/67;ou échange registre-mémoire).

Toutes ces solutions ont en commun un même point faible : "l'attente active"; un processus en attente de l'entrée en section critique teste périodiquement un jeu de conditions. Il consomme donc du temps "CPU" sans effectuer aucun travail. Ces solutions ont donc un effet opposé à celui recherché par la multiprogrammation. Cela explique l'intérêt des solutions suivantes qui éviterons ces attentes actives au prix d'une intervention du noyau.

- Solutions faisant intervenir le noyau :

  . Séquences "masquées" : cette solution consiste à "bloquer" temporairement (durant l'exécution de la section critique) le mécanisme d'allocation des processeurs en l'empêchant de reprendre le contrôle grâce au masquage global des interruptions. Ce masquage est obtenu par un déroutement d'appel au superviseur (SVC) ou par un branchement direct dans l'entrée du noyau prévu à cet effet. Le principal inconvénient de cette solution réside dans le risque de "perte" d'interruptions et conduit donc à des sections critiques relativement brèves afin d'en abaisser la probabilité.

  . Sémaphores [DIJK 67] : Dijkstra a proposé de résoudre ce problème d'exclusion mutuelle en ajoutant au noyau deux primitives (ininterruptibles) P et V sur un nouveau type de variables : les sémaphores.
  Ces primitives seront réalisées technologiquement dans la machi à processus (directement câblées, ou extensions logicielles au sein du noyau).

Un sémaphore s est formé d'un entier e(s) initialisé à la valeur unité et d'une file d'attente de processus f(s) initialement vide.
Les primitives P et V s'écrivent alors (figure II.4)[DIJK 68]:

```
P(s) : début
si e(s) < 0 alors
        début
        "stocker le processus exécutant dans f(s)";
        ARRETER (le processus exécutant);
        fin

V(s) : début
       e(s) := e(s)+1;
       si e(s) ≤ 0 alors cø : il y a au moins 1 processus bloqué
                début
                "choisir un processus p dans la file f(s)";
                RELANCER (
                fin
       fin
```

Figure II.4

REMARQUE

La réalisation de ces primitives implique l'existence d'une exclusion mutuelle élémentaire. En particulier une implantation logicielle utilisera soit des instructions "spéciales" ("lit et positionne" ou "échange registre mémoire"), soit des séquences "masquées" (ininterruptibles).

A l'aide de ces primitives, le problème de l'exclusion mutuelle se résout comme indiqué sur la figure II.5.

```
   p1              p2
P(s);           P(s);
<sc1>;          <sc2>;
V(s);           V(s);
```

Figure II.5

Extensions des primitives P et V

L'utilisation des sémaphores permet de résoudre directement des problèmes différents de l'exclusion mutuelle :

- contrôle de l'accès d'une ressource à n points d'entrées,
- couplage de deux processus suivant le schéma du "producteur - consommateur",
- ....

Nous employons le terme "directement" dans le sens suggéré par les travaux suivants : [PARN 75], [PATIL 71]; et qui peut s'exprimer comme suit : La résolution d'un problème de synchronisation à l'aide des sémaphores est dite "directe" lorsqu'elle n'utilise que les seuls sémaphores, manipulés exclusivement à travers les primitives P et V classiques [DIJK 68].

Cette restriction implique qu'une résolution directe ne comprendra pas l'emploi de variables auxiliaires dont les valeurs conditionneraient l'exécution d'une primitive sémaphorique.
Pratiquement, si l'on se place dans le cas d'un langage type ALGOLW étendu par l'existence de sémaphores, la restriction précédente revient à ne pas utiliser les primitives P et V dans les expressions conditionnelles de ce langage.

Exemples : Les solutions au problème des "lecteurs et des rédacteurs" dans [COURT 71] sont des solutions non directes.

Avec cette restriction, de nombreux auteurs (outre ceux susmentionnés, on peut citer [KOSA 73]) ont découvert des schémas de synchronisation impossibles à réaliser avec les primitives P et V dans leur définition originale.

De nombreuses extensions des primitives P et V ont alors été introduites, soit pour répondre à des difficultés rencontrées dans l'utilisation pratique des sémaphores, soit pour éliminer les limitations indiquées précédemment. Parmi ces travaux, citons [PATIL 71], [VANTI 72], ou [BELP 74], dans ce dernier, les primitives P et V sont décomposées en primitives plus élémentaires qui sont alors recombinées pour aboutir à une généralisation des primitives sémaphoriques classiques.

La plupart de ces travaux tentaient de définir un ensemble de primitives capables de résoudre "directement", sinon tous les problèmes de synchronisation, du moins une partie significative d'entre eux, comprenant notamment les plus classiques. Nous conclurons en remarquant que, si ces travaux ont proposé des primitives d'emploi plus aisé dans certains cas, aucune primitive générale et universellement acceptée n'en est ressortie.

L'exclusion mutuelle : problème de synchronisation central

Considérons deux processus devant se synchroniser suivant un protocole donné. Lorsque l'un des deux processus désire interragir avec l'autre, suivant les modalités prévues par ce protocole, ce processus doit connaître l'état de ce protocole au moment de sa tentative d'interraction afin de savoir si cette interraction est conforme ou non à ce protocole. Dans la négative le processus devra s'interdire l'interraction envisagée,et par conséquent,se bloquer si l'on veut éviter le phénomène d'attente active. En conséquence, au cours d'une tentative d'interraction, chaque processus devra vérifier, dans le cas où l'autre processus serait bloqué, si, au vu de l'état du protocole dont il a connaissance, ce processus ne pourrait pas être relancé.
L'implémentation d'un tel mécanisme nécessite la concrétisation de l'état courant du protocole par les valeurs d'un ensemble d'objets du langage de programmation utilisé. Chaque processus, exécute d'une part, des lectures de ces valeurs (i.e leur stockage dans des variables propres au processus) et d'autre part des modifications de ces valeurs, conformément à l'avancement de l'état du protocole provoqué par le processus. La cohérence de ces "variables d'état" partagées par les deux processus doit donc être garanti par un accès en exclusion mutuelle à ces variables. Nous constatons donc qu'un mécanisme d'exclusion mutuelle,renforcé par des primitives de blocage du processus exécutant et de réveil des processus bloqués suffit à assurer la synchronisation de deux processus.

Le schéma de la figure II.6 symbolise cette approche.

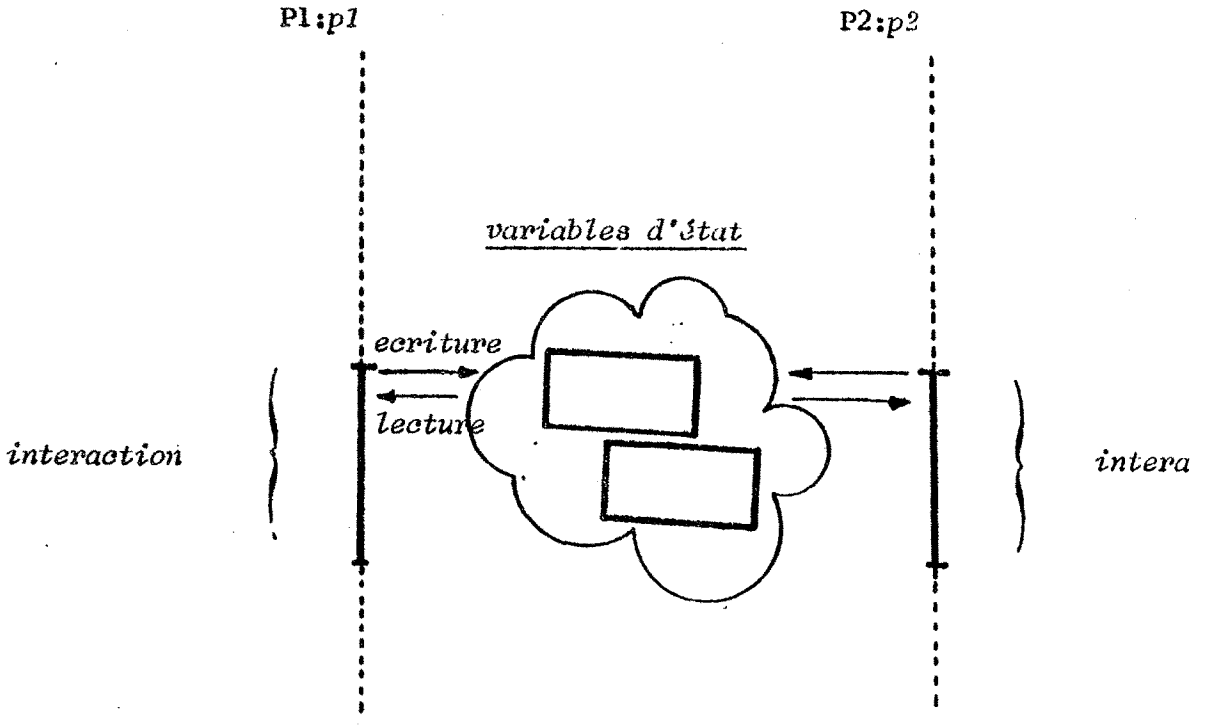


Figure II.6

Ce schéma de réalisation de synchronisation, dont nous avons tenté de donner une explication logique, est tout à fait analogue aux schémas d'utilisation des sémaphores (Figure II.7) remarqués par Dijkstra [DIJK ' 68] dans la programmation du système "THE". Dans cette construction Dijkstra met en évidence deux utilisations distinctes des sémaphores : exclusion mutuelle et privée, correspondant respectivement aux mécanismes d'exclusion mutuelle et de blocage/réveil signalés précédemment La généralité de cette structure nous semble garantie par son utilisation exclusive dans un logiciel de la taille d'un système d'exploitation.

P(mutex);

"Examen et modification des variables d'état comprenant éventuellemen
des opérations P et V sur les sémaphores privés d'autres processus ou
sur le sien propre";

V(mutex);

P(son propre sémaphore privé);

Figure II.7.

Ajoutons que la plupart des solutions aux problèmes de synchronisation classiques utilisent également ce même schéma : "lecteurs-rédacteurs" [COURT 71] ,"philosophes et spaghetti"dans DIJK 71 . On remarque donc la capacité du sémaphore à assurer ces deux mécanisme (exclusion mutuelle, et blocage/réveil).

L'existence d'une telle structure générale donne,selon nous,plus d'intérêt à la recherche d'une structuration dans l'utilisation des primitives sémaphoriques qu'à la recherche d'extensions plus ou moins intéressantes de ces primitives. Des travaux plus récents, que nous présenterons dans la suite, ont d'ailleurs adopté cette première approche.

Enfin, pour terminer, signalons que la définition des primitives P et V laisse indéterminée la politique de gestion de la file d'attente des sémaphores à l'exception d'une clause d'"équitabilité" assurant qu'aucun processus en attente ne risque d'être "oublié" par le système de gestion de la file.
Cette absence d'information sur cette politique impose l'écriture de solutions indépendantes de cette dernière. Si cette option laisse une grande liberté dans l'implantation des primitives P et V, elle conduit lorsque une politique particulière est exigée, à la programmation explicite de celle-ci. Une autre solution, consisterait à définir un jeu de primitives P et V pour chaque politique désirée; cette méthode n'est pas pratiquement applicable de par le nombre de ces politiques qui entraineraient un nombre prohibitif de primitives P et V différentes. De plus, la gestion de ces files faisant partie intégrante du système, leur programmation explicite nous semble préférable.

II - 3.2. Structuration : sections critiques et sections critiques conditionnelles

Les études évoquées lors du paragraphe précédent aboutissent à deux résultats essentiels :

- La définition du sémaphore .
- La mise en évidence d'un schéma d'utilisation standard des primitive: P et V.

Ces primitives P et V, tout à fait semblables à des instructions machine classiques, peuvent être disséminées sans restriction tout au long des programmes des processus. Une utilisation anarchique de ces primitives peut aboutir à des systèmes de processus peu compréhensibles et difficilement maîtrisables; souvent, même des énoncés simples conduisent à des solutions sémaphoriques obscures. Pour pallier ces inconvénients, la démarche prise peut être comparée à celle de la programmation structurée : imposer, par la syntaxe du langage de programmation lui-même, des structures d'utilisation des sémaphores facilitant la compréhension des programmes; structures devant évidemment tenir compte des résultats acquis antérieurement. Deux propositions ont été faites dans ce sens par [HOARE 71]

- Section critique
- Section critique conditionnelle.

. La première permet de rassembler tout un accès à une ressource critique dans une unité syntaxique unique ayant la forme suivante :

  with R do <SC>;

  dans laquelle R désigne la ressource critique, concrétisée dans le programme par un ensemble d'objets du langage,et <SC> symbolise la section de code que l'on désire exécuter en exclusion mutuelle. Cette construction est strictement équivalente au programme suivant :

  $P(S_R)$; <SC>; $V(S_R)$;

  dans lequel S_R désigne un sémaphore d'exclusion mutuelle protégeant la ressource R. Outre une meilleure lisibilité grâce à une localisation visuelle immédiate des sections critiques, la section critique permet d'éviter à l'écriture, ou de détecter à la compilation des erreurs telles que :

  - emploi incorrect des primitives P et V autour des sections critiques
  - accès à des objets de R en dehors des sections critiques,
  - mauvaise imbrication des sections critiques (suppression de certains interblocages [CROCU 75].

. La seconde proposition (section critique conditionnelle), comme le signale son auteur [HOARE 71], correspond au cas d'un processus impliqué dans un schéma de synchronisation, désirant exécuter une action dans le cadre de cette synchronisation et devant par conséquent attendre que l'état du protocole de synchronisation permette cette interaction. Cette construction a la forme suivante :

with R when <B> do <SC>;

R et <SC> ont le même sens que dans la section critique; <B> symbolise une expression booléenne portant sur des variables pouvant éventuellement appartenir à la ressource critique R.
Cette expression <B> est donc tout à fait adaptée à la concrétisation de la condition d'autorisation d'une interaction, tandis que R et <SC> peuvent être utilisées respectivement à matérialiser l'état du protocole et sa transformation consécutive à cette interaction. C'est précisément cette utilisation qui a été retenue par P.B. Hansen [HANS 72] dans la solution au problème de Lecteurs et des Rédacteurs [COURT 71]. La critique la plus généralement exprimée([MUNTE 78], [MOSS 77], [COURT 72]) à l'égard des sections critiques conditionnelles est l'inefficacité de leur implantation. La condition <B> associée à chaque processus en attente d'une ressource R doit être reévaluée chaque fois qu'un processus a terminé l'exécution d'une section critique <SC> associée à cette même ressource. Comme pour les sémaphores, l'ordre d'évaluation des conditions <B>, et donc l'ordre de libération effectif des processus en attente, n'est fixé que par l'implémentation. En contrepartie, de nombreux travaux([HANS 72], [HOARE 71]) même parmi les plus récents([MOSS 77]) considèrent que cette structure est une des plus agréables et des plus faciles à utiliser; nous constatons avec leurs auteurs la clarté et la rapidité des solutions aux problèmes classiques qu'elle propose. Notons enfin que si, comme nous l'avons signalé précédemment, les primitives sémaphoriques semblent tout indiquées pour la programmation en langage d'assemblage, les sections critiques conditionnelles présentent une forme syntaxique plus élevée, caractéristique d'un langage de plus haut niveau. Les sections critiques conditionnelles constituaient lors de leur introduction, une proposition d'"expression de contrôle du parallélisme" dans le cadre d'un thème de recherche encore ouvert aujourd'hui : langage de haut niveau pour l'écritur de systèmes d'exploitation.

II - 3.3. L'approche modulaire - Synchronisation par les ressources : Le Moniteur [HANS 78],[HOARE 74].

Cette approche repose sur l'hypothèse selon laquelle tout schéma de synchronisation entre des processus peut être associé à une ressource partagée entre ces processus : le schéma de synchronisation interprocessus s'identifie alors à la synchronisation entre les opérations d'accès à la ressource. Si dans certains cas (cf. exemple ci-dessous) l'introduction d'une telle ressource peut paraître quelque peu artificielle, on peut remarquer que cette ressource est au moins matérialisée par les variables renfermant l'état du protocole de synchronisation (Cf. II.3.1).

Exemple : "Rendez-vous" entre deux processus [BEKK 74] : deux processus doivent s'attendre l'un l'autre en deux points de leurs programmes respectifs. Ce problème conduit facilement aux programmes suivants (Figure II.8.A ; les variables entières des sémaphores S_1 et S_2 étant initialisées à la valeur unité).

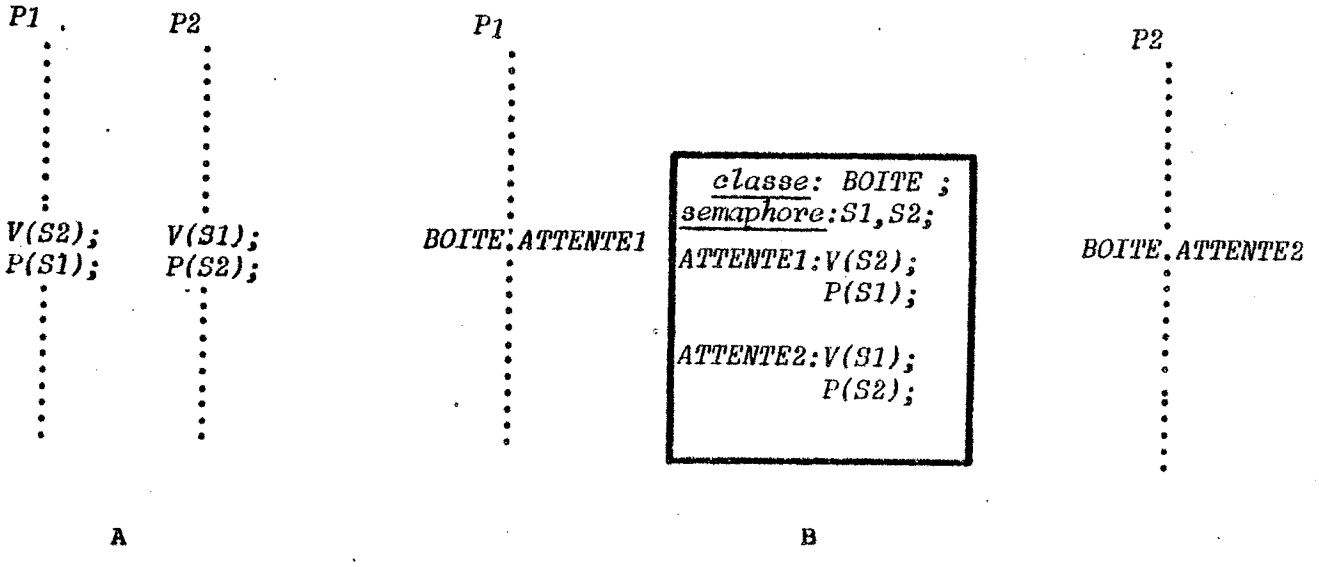


Figure II.8

Dans l'approche modulaire, la synchronisation interprocessus s'organise autour de l'accès à une ressource "boite" créée pour la circonstance (Figure II.8.B).

Dans ce cas, les primitives de synchronisation disparaissent du programme des processus et n'apparaissent plus que dans les fonctions d'accès aux ressources. Cette approche nécessite donc l'existence dans le langage de programmation d'une entité représentant la notion de ressource; c'est-à-dire le regroupement d'une structure de donnée et des procédures d'accès sépcifiques à cette structure de donnée.

Il ne semble pas que le besoin de structuration dans l'expression de la synchronisation ait provoqué l'apparition de l'entité "ressource". Le "secrétaire" de Dijkstra [DIJK 71] assez voisin de notre "ressource" n'a pas eu de prolongement immédiat.

De toute façon, il faut remonter au langage SIMULA [DAHL 70] pour reconnaître la première apparition de cette entité dans la notion de classe offerte par ce langage. De nombreux langages de haut niveau, postérieurs à SIMULA ont repris cette notion tant dans la programmation classique où elle favorise la décomposition par abstraction que pour l'écriture de systèmes.Dans ce dernier domaine elle fournit un puissant outil de décomposition connu par exemple sous le nom de module dans SESAME [MOSS 77] ou dans [DARON 78].Cette entité, rassemblant à la fois une structure de données et les seules procédures autorisées à la manipuler directement, (Figure II.9), procure un cadre syntaxique adapté à la représentation des objets rémanents nécessaires à la conception des systèmes; elle permet en outre l'expression de la protection de ces objets et de leurs liaisons mutuelles. Cette notion apparaît aujourd'hui indispensable à tout langage de haut niveau destiné à l'écriture de systèmes; d'où la nécessité d'adaptation à cet état de fait des recherches sur l'expression à un haut niveau de la synchronisation.

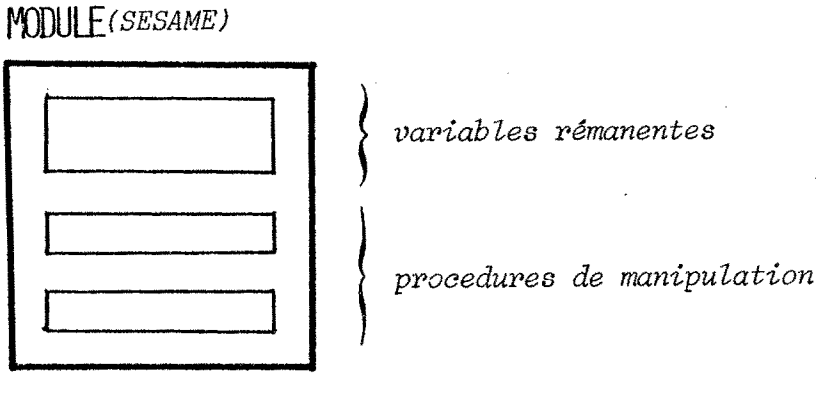


Figure II.9

Conformément à la proposition de Hoare,nous appelerons moniteur un module dont les exécutions des procédures internes doivent respecter des contraintes de synchronisation. Dans le moniteur deux mécanismes permettent l'expression de cette synchronisation :

- un mécanisme implicite; toutes les procédures du moniteur appartiennent à une sorte de section critique (Cf. II.3.1) : à tout moment un seul processus est actif dans le moniteur,
- un mécanisme explicite d'attente et de réveil de processus.

L'existence du seul mécanisme implicite conduirait simplement à l'exécutic des procédures dans une même section critique. Le second mécanisme permet à un processus de se mettre en attente d'un "signal" C interne au moniteur tout en relachant cette exclusion mutuelle,par exécution de la primitive C WAIT. Chaque "signal" C est représenté dans la proposition de Hoare par un variable de type condition, nouveau type introduit spécialement à cet effe Une file d'attente de processus, analogue à celle du sémaphore, concrétis chaque variable de ce type. Gâce à ce dispositif plusieurs exécutions de procédures du moniteur peuvent être simultanément en cours; l'exclusion mu le impose simplement qu'un seul processus soit actif dans le moniteur. Ce processus actif peut émettre des "signaux" de réveil par éxécution d'une pri notée C.SIGNAL pour le "signal" C; si aucun processus n'est en attente du "signal" C, cette exécution est sans effet (il n'y a donc pas,comme dans le sémaphore, de mémorisation des "signaux" de réveil); dans le cas contraire, u des processus en attente du "signal" C deviendra le processus actif du moni alors que le processus ayant exécuté la primitive C.SIGNAL sera mis en atte

A partir de l'instant où un processus tente d'entrer dans le moniteur en appelant une procédure de ce dernier et jusqu'à sa sortie, lors du retour de cette procédure, un processus peut se trouver dans quatre états différents :

1) Il attend d'entrer dans le moniteur.
2) Il est en attente d'un "signal" C postérieurement à l'exécution d'une primitive C.WAIT. Il est donc placé dans la file d'attente associée à la condition C.
3) Il est suspendu consécutivement à l'exécution d'une primitive SIGNA qui a réveillé un processus en attente.
4) Le processus est le seul processus actif du moniteur.

Dans la proposition de Hoare, les transitions entre ces états sont assurées grâce à un système de deux files d'attente (simulées par deux sémaphores d'exclusion mutuelle dans [HOARE 74]):

- Une file d'attente f_1 contenant les processus placés dans l'état 1,
- Une file d'attente f_3 contenant les processus placés dans l'état 3.

Lors de la libération de la section critique du moniteur (sortie du moniteur où exécution d'une primitive wait par le processus actif), on attribuera le contrôle du moniteur à un des processus en attente dans f_1 où f_3; la priorité est donnée au processus dans l'état 3 (un processus dans l'état 1 ne peut prendre la section critique que lorsque f_3 est vide). Si ces deux files sont simultanément vides, le moniteur est disponible pour le prochain processus qui tentera d'y pénétrer.

Le principal apport du moniteur semble résider dans son utilisation en tant qu'outil de décomposition de systèmes. Son apport sur le plan de la synchronisation indépendamment de la vision "ressource" qu'il impose nous semble finalement assez réduit ne serait-ce qu'en raison des trois points suivants :

1) Si l'exclusion mutuelle entre procédures du moniteur simplifie dans de nombreux cas l'écriture du moniteur, elle s'oppose parfois à l'efficacité de la collaboration entre processus utilisateurs du moniteur, en imposant des accès mutuellement exclusifs à la ressource représentée par le moniteur. Les solutions proposées à ce type de problème consistent scuvent en la décomposition de la ressource en deux parties :
   . un moniteur qui en contrôle l'accès,
   . un module représentant la ressource proprement dite.

   De telles solutions présentent l'inconvénient de multiplier le nombre de modules du système. Dans le même style, on trouve les problèmes des appels intermoniteurs.

2) L'emploi de primitives de synchronisation peu structurées produit souvent des solutions obscures pour des problèmes relativement peu complexes. De plus, ces primitives, très voisines de primitives de synchronisation par événements, risquent dans certains cas de présenter les mêmes inconvénients [CROCU75]: fonctionnement dépendant du temps.

3) Remarquons enfin la complexité des interactions (qui de plus sont implicites) entre les deux mécanismes de synchronisation. Une solution faisant intervenir des particularités (priorités entre les divers processus en attente du moniteur par exemple) de cette liaison risque d'être d'une compréhension délicate.

Conclusion

En guise de conclusion sur les moniteurs nous ferons trois remarques :

- Une méthodologie de programmation, proposée dans [HOARE74] permet de supprimer en partie les inconvénients du point 2 ci-dessus, en évitant un emploi anarchique des primitives WAIT et SIGNAL. Cette méthodologie consiste à :

  . associer à chaque variable condition un prédicat portant sur des variables locales du moniteur,
  . n'exécuter une primitive signal sur cette condition que lorsqu ce prédicat est vérifié. On aboutira à des programmes dans lesquels chaque appel à la primitive signal sera précédé par une vérification du prédicat associé.

  On notera que la validité de cette méthode repose sur la cohérence des variables locales au moniteur, qui est garantie par l'exclusion mutuelle.
  Nous tempèrerons ce résultat en disant qu'une méthode de programmation n'est vraiment réellement employée qu'à condition d'être imposée par la syntaxe [MOSS 77] ce qui est loin d'être le cas dans le moniteur.

- Les moniteurs laissent le programmeur libre d'exprimer explicitement le réveil et le blocage des processus. Cette disposition, conduisant à une meilleure efficacité, puisque adaptable à chaque problème conduit à l'adaptation du moniteur dans de nombreux langages d'écriture de systèmes (Exemples : SESAME [MOSS 77], MODULA [WIRTH 77] CONCURRENT PASCAL [HANS 78].

- Quelques améliorations ont été proposées pour augmenter la lisibilité du moniteur [SHRIV 76], [KESS 77]. Ce dernier travail impose une structuration de la synchronisation dans un moniteur et lie explicitement une condition avec un prédicat sur les variables internes au moniteur. Ce mécanisme libère le programmeur de l'écriture des évaluations de ces prédicats et des réveils explicites des processus en attente. Ces derniers seront libérés dès que la section critique du moniteur sera relachée et que le prédicat associé à la condition d'attente sera validé.

II - 3.4. Séparation traitement-synchronisation

Dans la programmation comme dans la description des systèmes lorsque l'on désire maîtriser ou étudier un certain aspect du système il est généralement intéressant de pouvoir isoler dans la représentation elle-même tout ce qui le concerne. Cette disposition focalisera l'attention de l'utilisateur sur cet aspect en lui permettant de faire abstraction de tous les détails inutiles. Nous avons supposé que le langage d'écriture de systèmes nous offrait l'objet module pour traduire la notion de ressource. Nous désirons spécifier et étudier la représentation de la synchronisation entre procédures du module; il est donc naturel de tenter d'extraire et d'isoler l'aspect synchronisation du corps des procédures du module. Partant d'une structure de module héritée de celle de CLASS dans Simula 67 [DAHL 70] nous arrivons à une structure dans laquelle à côté des parties structure de données et procédures, apparaît une partie synchronisation (Fig.II.10).

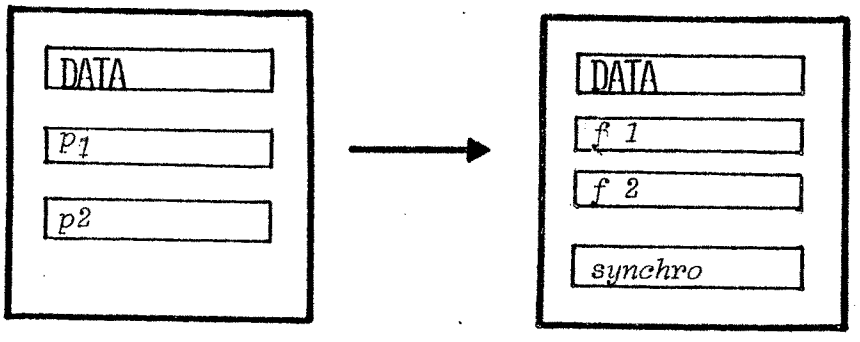


Figure II.10

Devant cette approche, plusieurs questions se posent :

- Peut-on exprimer tous les problèmes de synchronisation sous forme "séparée" ?
- Dans la négative : quels sont les problèmes dans lesquels cette "séparation" est possible ?
- Peut-on réellement programmer la partie synchronisation sans connaître la réalisation des autres parties ?
- Peut-on véritablement changer la programmation des procédures et la spécification de la synchronisation indépendamment l'une de l'autre ?
- ...

Nous essaierons de répondre à ces questions ultérieurement (§II.5). Les réponses dépendent étroitement de l'expression de la synchronisation et donc de l'outil choisi pour cette expression. Nous allons successivement présenter deux propositions de tels outils : les expressions de chemins [CAMBP 74] et les modules de contrôle [ROVER1 77].

Expressions de chemins (Path Expressions)

Les expressions de chemins proposent une description de la synchronisation par spécification de contraintes dans l'ordre des exécutions des procédures du module.
Cette spécification se présente comme un ensemble de contraintes indépendantes, chacune représentée par une "expression de chemin" distincte. Comme toute expression, une expression de chemin est une suite d'opérateurs et d'opérandes. Les opérandes sont les noms de procédures dont l'ordre des exécutions subit les contraintes spécifiées par l'expression.
Parmi les opérateurs on peut distinguer trois types :

- Les opérateurs "réguliers" (tout à fait semblables à ceux des expressions régulières):
  - opérateur d'exclusion ou de sélection (union dans les expressions régulières) noté + :
    Exemple : a+b; les procédures a et b doivent être exécutées en exclusion mutuelle;
  - opérateur de succession (concaténation dans les expressions régulières) noté multiplicativement :
    Exemple : a b; toute exécution de la procédure b devra être précédée d'une exécution de la procédure a;

. opérateur de répétition (fermeture de Kleene)
Exemple : a^*, indique la possibilité d'un nombre quelconque d'exécutions successives de la procédure a.

Ces opérateurs peuvent être utilisés récursivement, i.e chaque opérande peut être remplacé par une expression de chemin construite à l'aide des trois opérateurs ci-dessus. Cette extension comporte de nombreuses restrictions dans la proposition originale[CAMPB 74]. Ces restrictions sont consécutives non pas à des difficultés sémantiques, mais à des limitations imposées par la méthode d'implémentation de ces expressions à l'aide de sémaphores proposée dans ce même article. On peut trouver curieux les motifs de ces restrictions pour un outil de spécification de la synchronisation.

- Opérateurs "non réguliers" :
  . opérateurs d'exécutions simultanées [CAMPB 74].
  Exemples - {a} indiquant la possibilité d'exécutions simultanées de la procédure a par des processus distincts. La procédure a peut être remplacée par une expression régulière complète. Dans la proposition originale cette expression ne devait pas comporter d'opérateurs d'exécutions simultanées .

  - a{b c} a impose une exécution préalable de la procédure a, suivie par une suite d'exécutions de procédures b et c de telle sorte que le nombre d'exécutions de c soit inférieur ou égal au nombre d'exécutions de b ;à un instant où :
    + il n'y aura plus d'exécutions de b ou de c en cours
    + il y aura eu autant d'exécutions de b et de c;
    un processus pourra exécuter la procédure a.

  . opérateurs "parallèles" [CAMPB 76].
  Exemples - a ||b indique qu'une exécution de a et une exécution de b (éventuellement simultanées) doivent avoir lieu,
  - c(a||b)d impose une exécution de la procédure c suivi d'une exécution de chaque procédure a et b. Ces exécutions pouvant se recouvrir dans le temps.
    Une fois ces deux éxécutions terminées, un processus pourra éxécuter la procédure d.

. opérateur "ou exclusif" (noté ⊕).

Exemples - a ⊕ b qui peut se représenter à l'aide des opérateu déjà mentionnés,

- a ⊕ b <=> a+b+(a||b)
- (a ⊕ b)d <=> (a+b+(a||b))d

. opérateur de priorité [CAMPB 76],[HABER 75].

Cet opérateur noté >(ou <) peut remplacer l'opérateur + et donne une priorité d'exécution à un des deux opérandes, tout en maintenant l'exclusion mutuelle originale.

Exemple : d(a > b)c

Après une première exécution de d, si deux processus demandent l'un une exécution de a, l'autre une exécution de b, celui qui demande l'exécution de a aura la priorité.

. Expressionsde chemin "ouverte"[CAMPB 76]et expressions de chemin avec compteurs [FLON76]qui permettent de spécifier des contraintes entre les nombres d'exécutions de chaque procédure

Exemple : [ANDL 78].

$(f - g - \ldots - h)^n$

n est un entier positif : f,g,...,h sont des noms de procédures.

Cette expression est équivalente à l'union de deux contraintes :

-(f+g+ + h) : expression de chemin régulière (exclusion mutuelle)

$-(\#(f) \geq \#(g) \ldots \geq \#(h) \geq \#(f)-n$

$\#(i)$ désigne le nombre d'exécutions de la procédure i(y compris celle éventuellement en cours).

- <u>Opérateurs faisant intervenir la partie traitement</u>:

. Expressions de chemins conditionnelles[HABER75][CAMPB76].Soit u expression ne comportant que des opérateurs de sélection +;

Exemple : $p_1 + p_2 + .. + p_i + ... + p_q$

Il est possible d'associer à chaque procédure p_i une condition C_i (expression booléennes sur les variables rémanentes du modul telle que ces conditions soient disjointes deux à deux

$(\neg C_i \wedge C_j \quad \forall i,j \; i \neq j \in \{1,\ldots,q\})$ et complètes $(\bigvee_{i=1}^{q} C_i)$

On aboutit à la syntaxe suivante :

$C_1 : p_1 + C_2 : p_2 \ldots + C_i : p_i + \ldots + C_q : p_q$

à laquelle on donne le sens suivant : la procédure p_i ne peut être exécutée que lorsque la condition C_i est vérifiée.

. Expressions de chemin conditionnelles avec répétitions [CAMPB 7(

Dans la construction précédente certaines conditions peuvent être remplacées par l'expression

$\underline{\text{while}}\ C_i\ \underline{\text{do}}\ p_i$

La procédure p_i sera alors exécutable tant que la condition C_i restera validée.

Notons que l'emploi de ces extensions dans des expressions est souvent limité, soit du fait d'un manque d'interprétation (limitations sémantiques) soit par la méthode d'implantation choisie par les auteurs.

Conclusions

Après cette présentation des expressions de chemins plusieurs remarques s'imposent :

1) Les expressions de chemin limitées aux opérateurs réguliers ne permettent de résoudre que les problèmes de synchronisation dans lesquels :

. toutes contraintes de synchronisation peuvent s'exprimer par des contraintes sur l'ordre d'exécution des procédures. (toutes les procédures mentionnées dans une même expression régulière doivent être exécutées en exclusion mutuelle);

. il faut évidemment que les contraintes précédentes soient régulières notons que dans de nombreux cas classiques de non régularité des contraintes,celle-ci découle uniquement de l'absence (voulue) d'informationssur le nombre maximal de processus du système. Faute de ces informations de nombreux problèmes doivent être résolus pour un nombre de processus quelconque et non borné à priori. Dans le cas où une borne maximale du nombre de processus est donnée, la solution redevient régulière, mais présente souvent une expression lourde et pénible. Pour alléger cette représentation des auteurs [LAUER 77] proposent une"macro notation"des expressions de chemin. Les valeurs de paramètres de génération(entiers) conditionnent l'expansion de ces macro notations; paramètres qui peuvent être fonction de ce nombre maximal de processus. Les problèmes des lecteurs/rédacteurs [COURT 71] offrent de bons exemples de ce type de problèmes.

2) Les extensions "non régulières" manquent de cohérences. Les expressions utilisant de tels opérateurs ({ },||...) ne peuvent généralement pas être ramenées directement à un ensemble de contraintes dans l'ordre des exécutions de procédures.

3) Les extensions faisant intervenir les variables rémanentes du module remettent en cause le principe de séparation partie traitement/partie synchronisation.

En conclusion, l'étude des expressions de chemins laisse l'impression que les promoteurs de cet outil, devant ses faiblesses initiales,ont rajouté, aux gré des problèmes rencontrés, les extensions "had-hoc" sans se préoccuper ni de la généralité d'emploi de ces extensions ni de leur cohérence avec les primitives préexistantes.

Modules de contrôle [.ROVER1 77] & [ROVER2 77] .

Un module de contrôle offre une alternative pour la programmation de la partie synchronisation d'un module.

Description

Un module de contrôle comprend deux types d'objets :

- des objets renfermant des informations relatives à l'état du module vis à vis de la synchronisation : celles-ci se rangent sous deux catégories :
  - . Un ensemble de queues {Qi} (i = 1...n)
    A chaque queue Q_i correspond un ensemble E_i de procédures du module. Les ensembles E_i réalisent une partition de l'ensemble E des procédures du module.
    ($E_i \cap E_j = \emptyset \ \forall i \neq j \in (1 \ldots n)$ et $\bigcup_{i=1}^{n} E_i = E$)
    Chaque queue Q_i est concrétisée par une file d'attente de processus destinée à recevoir tous les processus demandant l'exécution d'une procédure appartenant à E_i et qui, par suite des règles de synchronisation, ne peuvent procéder à cette exécution.
  - . Un ensemble de compteurs d'état.
    A chaque procédure p on associe cinq compteurs d'état. Un compteur d'état est une variable rémanente de type entier dont la valeur à tout instant sera égale au nombre de processus dans un même état relativement à la procédure p.

Dénomination	*interprétation*
#*req.P*	*nombre de demandes d'exécutions de la procédure P depuis l'initialisation du système.*
#*aut.P*	*nombre d'autorisations d'exécution de la procédure P depuis l'initialisation du système.*
#*term.P*	*nombre d'exécutions de la procédure P terminées depuis l'initialisation du système.*
#*act.P* =#*aut.P*-#*term.P*	*nombre de processus en cours d'exécution de la procedure P.*
#*wait.P* =#*req.P*-#*aut.P*	*nombre de processus en attente d'exécution de la procedure P.*

Figure II.11

- Un jeu de conditions, chacune associée à une procédure distincte. Ces conditions portent sur les valeurs des compteurs d'état du module. Chaque condition est spécifiée par une expression booléenne comparable à celles des langages type ALGOL et ne faisant intervenir aucune variable autre que les seuls compteurs d'état. Ces expressions sont de plus limitées à une classe très restreinte, classe choisie de façon arbitraire par les auteurs qui désiraient valider une approche beaucoup plus qu'une réalisation pratique.

Fonctionnement

Lorsqu'un processus demande à exécuter une procédure p, il est placé dans la queue associée à la procédure p. Il ne pourra procéder à cette exécution que lorsque deux conditions seront simultanément remplies :

- il est en tête de la queue associée à la procédure p,
- la condition correspondant à la procédure p est vérifiée.

L'examen de ces deux conditions doit, a priori, être effectué chaque fois qu'un processus demande, commence, ou termine une exécution d'une des procédures du module. Notons que, comme pour les sémaphores, ni l'ordre d'examen des files d'attente, ni la gestion de ces files d'attentes ne sont spécifiés. La synchronisation désirée devra donc être assurée indépendamment de ces choix d'implémentation.

II - 3.5. Conclusion

En guise de conclusion nous présenterons deux schémas récapitulatifs :

- Le premier donnera une esquisse de classement des outils de synchronisation en fonction de critères dégagés précédemment (figure II.12).
- Le second présentera les filiations avouées ou attribuées entre ces divers outils, ainsi que les principales causes de leur introduction (figure II.14).
- Il faut prendre garde à ne pas confondre le niveau du langage hôte des outils de synchronisation avec le "niveau" propre de ces outils.

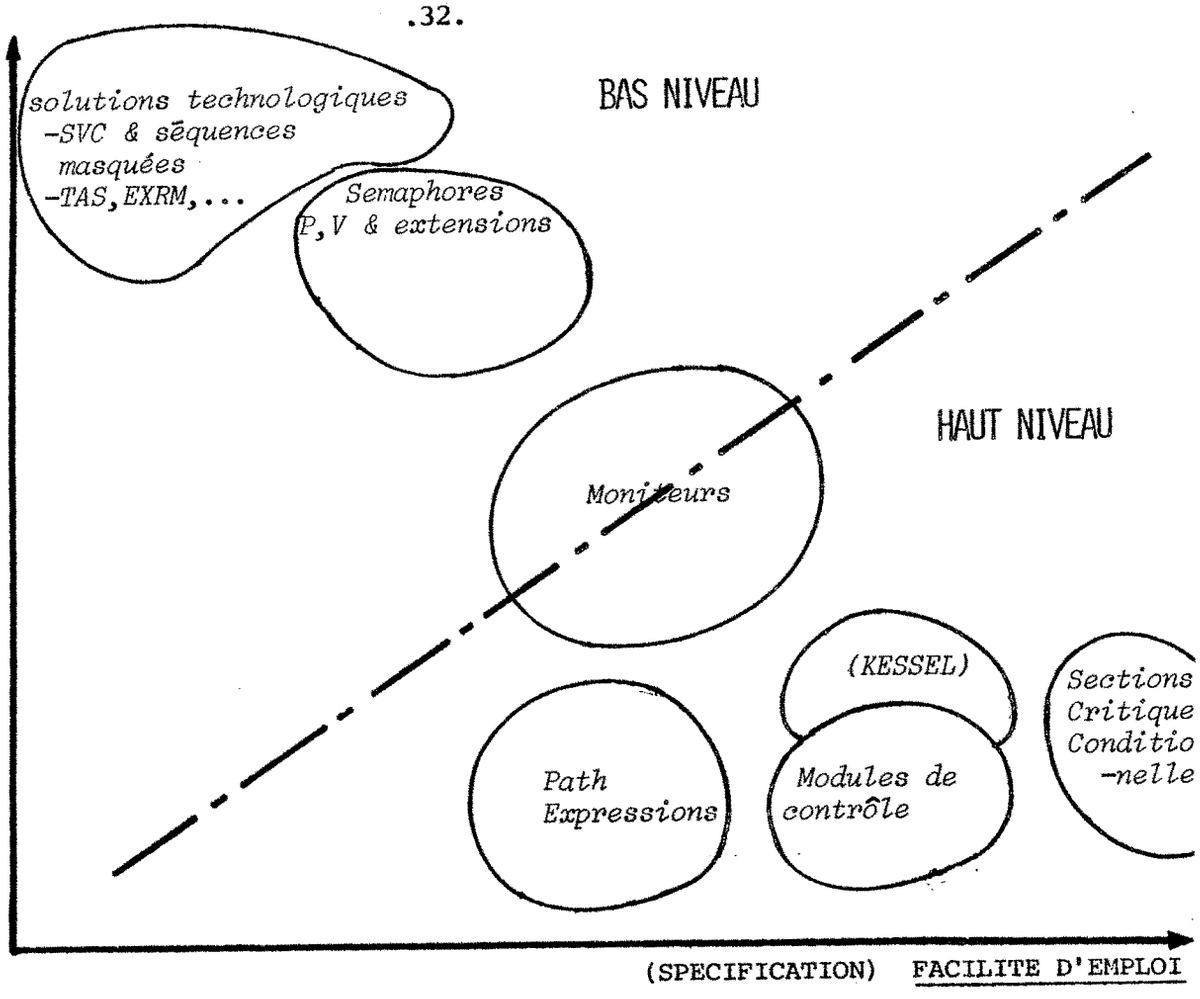


Figure II.12

Technologiques (orientées machines)	*Logiques (orientées problèmes)*			
	vision processus		*vision ressources*	
-SVC & séquences masquées	*bas niveau*	*haut niveau*	*bas niveau*	*haut niveau*
-LOAD & STORE	*P, V & extensions*	*Sections Critiques Conditionnel--les*	*Moniteurs*	*-Path -Compteurs -...*
-TEST & SET				

Figure II.13

Selon nous, le niveau d'un outil de synchronisation est lié à l'existence de primitives de réveil et de blocage explicites de processus.
Nous dirons qu'un outil est de bas niveau lorsque de telles primitives existent (P,V : sémaphores , C. wait, C. signal : Moniteurs loch, unloch...).
Tous ces dispositifs imposent en effet l'écriture explicite des programmes d'évaluation des conditions de blocage ou de réveil portant sur les variables mémorisant l'état de la synchronisation à assurer.

La synchronisation implicite semble amener une certaine inefficacité mais, en revanche, conduit le plus souvent à des spécifications plus rapides et plus claires des problèmes de synchronisation.

Toutes ces primitives peuvent être incluses dans des langages de haut niveau, néanmoins les sections critiques, les "path" , et les "modules de contrôle" dont l'utilisation suppose une syntaxe rigide, impose l'utilisation d'un compilateur (Figure II.13).

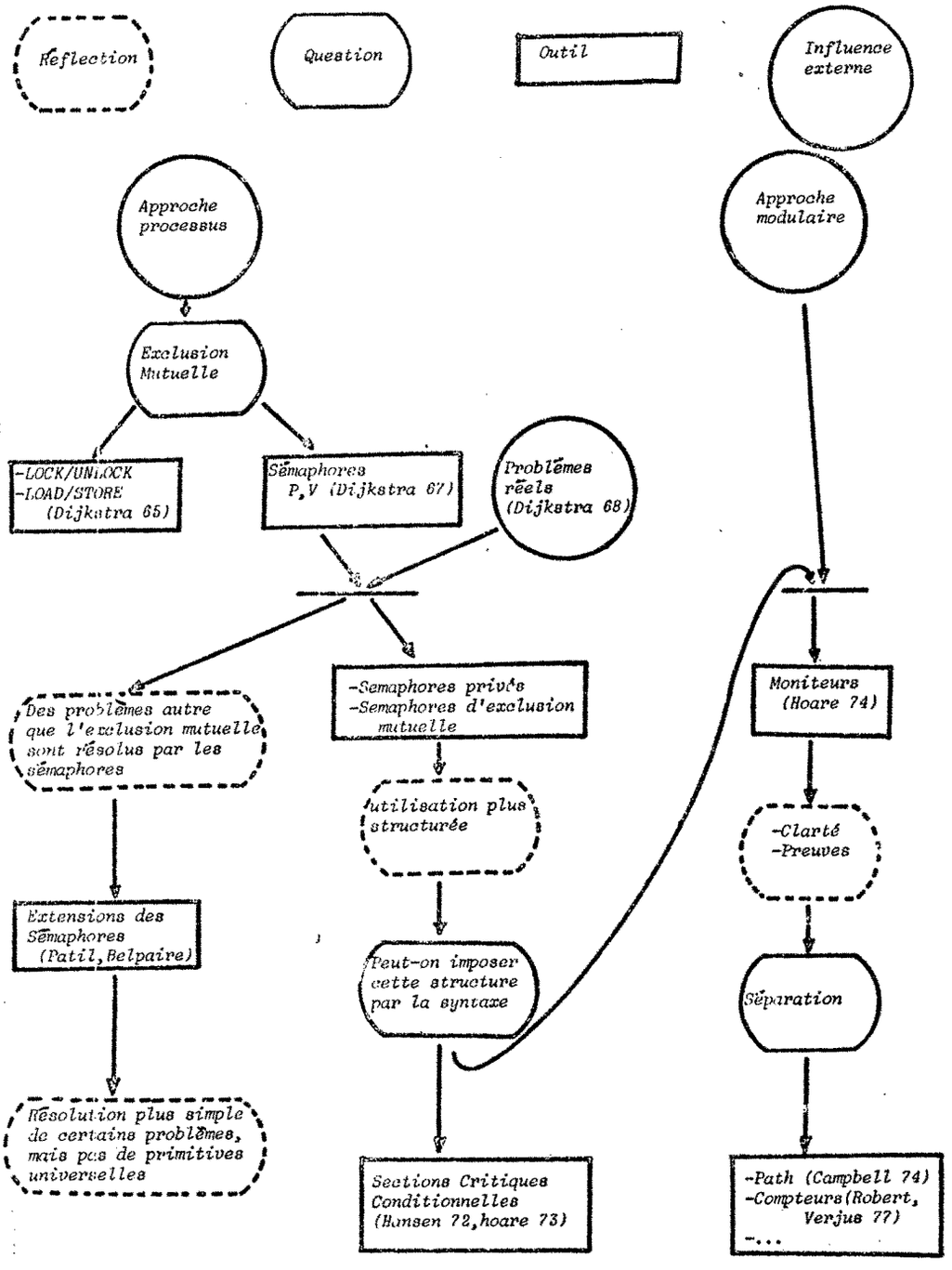

Réflection
Question
Outil
Influence externe
Approche processus
Approche modulaire
Exclusion Mutuelle
-LOCK/UNLOCK
-LOAD/STORE
(Dijkstra 65)
Sémaphores P,V (Dijkstra 67)
Problèmes réels (Dijkstra 68)
Des problèmes autre que l'exclusion mutuelle sont résolus par les sémaphores
-Semaphores privés
-Semaphores d'exclusion mutuelle
Moniteurs (Hoare 74)
utilisation plus structurée
-Clarté
-Preuves
Extensions des Sémaphores (Patil, Belpaire)
Peut-on imposer cette structure par la syntaxe
Séparation
Résolution plus simple de certains problèmes, mais pas de primitives universelles
Sections Critiques Conditionnelles (Hansen 72, hoare 73)
-Path (Campbell 74)
-Compteurs (Robert, Verjus 77)
-...

Figure II.14

II - 4. Modélisation - Formalisation

II - 4.1. Présentation et hypothèses

Le problème que nous nous proposons d'étudier est le problème de la spécification de la synchronisation inter-processus dans un système d'exploitation.

Spécification

La réalisation d'un logiciel met en jeu trois types d'entités distinctes:

- les intentions,
- les spécifications,
- le programme.

Le terme "intentions" recouvre tous les aspects pris par son projet (de produit logiciel) dans la conscience de son initiateur. Cet initiateur, au moins lorsqu'il se propose d'utiliser les services d'un programmeur, se doit de traduire objectivement cette conscience par un écrit servant de point de départ au travail du programmeur; travail qui aboutira à l'élaboration d'un programme. Cet écrit rassemble les spécifications du programme à réaliser. Outre cette fonction de communication imposée par la division du travail, les spécifications assurent un rôle de document de référence pour une certification ultérieure du programme.
D'une façon générale, l'objectif recherché dans l'écriture de spécifications consiste en l'obtention d'un texte non-ambigü décrivant le problème à résoudre. Cette exigence de non-ambigüité impose l'utilisation d'un mécanisme de description lui-même non ambigü, i.e. d'un langage formel muni d'une sémantique précise parfaitement assimilée par ses utilisateurs (pour atteindre une telle universalité, la sémantique sera bien souvent exprimée dans un langage accepté et bien compris par tous : le langage mathématique).
Par delà ces nécessités, la qualité d'un bon langage se juge en partie sur son aptitude "à formuler facilement des spécifications lisibles de n'importe quel problème afin de rendre la transition entre intentions et spécifications très sûre"[CAPL 78] . Nos buts, dans cette étude, tout en étant similaires, restent beaucoup plus modestes que ceux poursuivis dans[CAPL 78]. Notre problème, bien qu'étant un cas particulier de programmation, s'en distingue pourtant par les remarques suivantes.

1) Il se pose dans un cadre très particulier, mettant en jeu des notions déjà bien définies par les études précédentes (processus, ressources, synchronisations..).Les solutions éventuelles ne pourront qu'être très orientées par la vision du problème découlant de ces notions.

2) L'existence d'un (petit) nombre de problèmes classiques de ce type, chacun de ces problèmes étant "spécifié" par un ou plusieurs énoncés en langue naturelle.

3) L'existence préalable de "schémas", voire d'outils, conduisant à des solutions pour l'ensemble de ces problèmes (cf. § précédent).

Ces particularités suggèrent une approche en trois étapes :

a) Formaliser le problème de synchronisation en tenant compte des notions dégagées antérieurement et extraire de cette formalisation les caractéristiques d'une structure de synchronisation adaptée.

b) Donner à cette structure de synchronisation une forme propre à son utilisation par les programmeurs et tester cette aptitude sur des exemples classiques. Nous testerons ainsi la sûreté de transition entre intentions et spécifications par la simplicité et la rapidité du passage de l'énoncé en langue naturelle au texte de spécification.

c) Proposer des méthodes de réalisation en s'appuyant sur les outils précédemment définis.

Hypothèses

Au cours de cette étude nous nous placerons délibérément dans l'optique des travaux les plus récents sur ce sujet. Nous adopterons donc les principes suivants :

- Approche modulaire dont l'apport en tant qu'outil de décomposition et de structuration des systèmes est aujourd'hui reconnu et qui, par conséquent, ne peut être ignorée d'une étude actuelle sur la synchronisation.

- La séparation entre expressions dutraitement et de la synchronisatio au sein des modules. Les premières études engagées dans ce sens ont montré que les objectifs recherchés dans cette séparation étaient bien atteints :

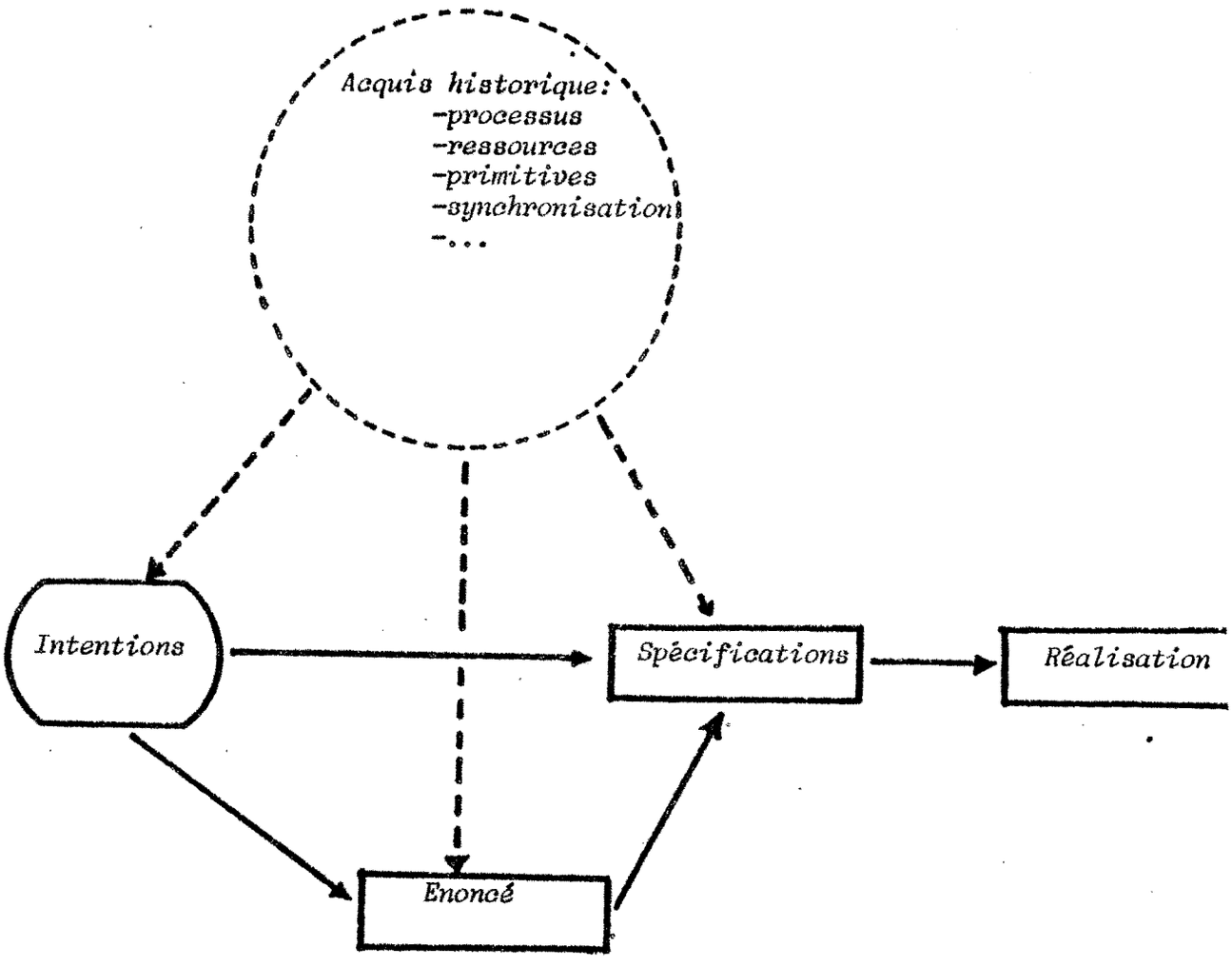


Figure II.15

. Malgré des limitations inhérentes à cette séparation, limitations que nous préciserons ultérieurement, un domaine significatif d'applications (contenant les plus classiques) peut être résolu.

. Au vu des exemples traités le gain en clarté et en rapidité d'expression paraît évident.

. L'interchangeabilité des parties traitement et synchronisation indépendamment l'une de l'autre semble assurée.

Remarque :

Avant d'aborder ce problème d'expression séparée de la synchronisation entre procédures d'un même module, il convient d'abord d'en prouver l'intérêt. Plusieurs auteurs relèvent en effet le petit nombre de problèmes de synchronisation effectivement rencontrés dans les systèmes réels. Si l'on adopte ce jugement, la spécification de la synchronisation au niveau des modules ne nécessite plus que l'adjonction au module à synchroniser d'un

attribut indiquant le type de synchronisation souhaité . Cet attribut pourrait être donné par un mot-clef désignant cette synchronisation suivi, éventuellement, de paramètres judicieux.

Exemple : spécification d'une politique producteur/consommateur [COURT 71] entre deux procédures d'un même module (respectivement : PRODUIRE,CONSOM) : figure II.16; prod/cons est le mot clef du langage d'écriture associé à la politique choisie.

```
module TAMPON sync : prod/cons (PRODUIRE,CONSOM );
    :
 procédure PRODUIRE(...);
    :
 procédure CONSOM  (...);
    :
fin module
```

Figure II.16

Une telle méthode de spécification serait certainement la plus précise (sous réserve de la bonne compréhension de chaque politique sélectée) et de toute évidence la plus rapide, la plus sûre et la plus claire; elle éliminerait parallèlement toute nécessité de preuve au niveau du module. Bien que n'ayant pas d'expérience pratique susceptible de nous appuyer, nous remarquons que, si la plupart des problèmes peuvent être effectivement rangés en un nombre restreint de types, chacun de ces types peut faire l'objet d'un très grand nombre de variantes. Ces variantes ne remettent ordinairement pas en cause les contraintes fondamentales d'utilisation de la ressource, elles concernent, par contre, très souvent la mise en oeuvre de stratégies d'accès plus sophistiquées, généralement adaptatives. De telles variantes ne nous paraissent pas entièrement académiques comme le montre certains exemples réels présentés dans [BBV 76].

En tout état de cause, seule l'écriture de systèmes réels s'appuyant sur un langage autorisant une spécification claire et directe de telles stratégies pourra définitivement trancher cette question.

II - 4.2. Modèle Formel

Point de synchronisation

D'une façon générale, "l'état" des processus relativement à une ressource caractérise l'état de cette ressource vis à vis du problème de synchronisation qui lui est associé. Les instants de changement de cet état sont prévus dans le programme de chaque processus sous la forme de modifications de la situation de ce dernier vis à vis de cette ressource. Imposer des règles de synchronisation revient donc à imposer des contraintes dans l'ordre de ces modifications. Ces instants particuliers correspondent à la présence de points de synchronisation (notés PDS) dans le programme du processus considéré. Cette présence se manifeste généralement par l'utilisation de primitives de synchronisation. A chacun de ces instants, deux éventualités peuvent se produire selon que les règles de synchronisation permettent la poursuite du processus ou non (on attend pour franchir le PDS que les règles le permettent).

Vocabulaire et langage de synchronisation

L'intégration de la synchronisation dans la description d'une ressource requiert l'insertion des PDS liés à cette ressource dans le corps des procédures qui la manipulent. Il n'y a donc qu'un nombre fini de PDS pour une ressource donnée. On associe à chacun de ces PDS un nom : élément d'un vocabulaire V propre à la ressource. Si l'on suppose que les instants de franchissement des PDS sont tous distincts, l'activité des processus vis à vis de cette ressource se traduira par une suite de franchissements de PDS, et donc également par une chaîne du monoïde libre V^*. La séparation entre partie traitement et partie contrôle impose que la condition de franchissement d'un PDS soit indépendante du processus demandeur. Cette condition ne dépend donc que du PDS lui-même et de l'état de la ressource vis à vis de son problème de synchronisation. Le contrôle de la synchronisation consiste à retarder (éventuellement) le franchissement de certains PDS par les processus afin de maintenir cette chaîne dans un langage L : traduction formelle des règles d'utilisation de la ressource. L est appelé langage de synchronisation associé à cette ressource. Notons que certains PDS sont toujours franchissables : leur condition de franchissement est toujours vérifiée. Les processus ne seront jamais retardés par leur franchissement.

Remarquons que tout préfixe d'une chaîne U, appartenant à un langage de synchronisation L, appartient aussi à L :
Si U traduit une utilisation permise de la ressource (U ∈ L) jusqu'à un instant donné, alors tout préfixe de U traduit une utilisation également permise de la ressource jusqu'à un instant antérieur.

Propriété de préfixe

$$\left\{\begin{array}{l} \forall\ Ls \text{ langage de synchronisation} \\ \quad Ls \subset V^* \end{array}\right. \quad \left.\begin{array}{l} \tau \in Ls\ \tau = \mu.\nu \\ \nu \in V^* \end{array}\right\} \Rightarrow \{\ \mu \in Ls\ \}$$

Exemples : Nous illustrons notre proposition par la modélisation de quelques exemples classiques. Pour chacun de ces exemples nous donnerons, après une rapide description, le jeu de points de synchronisation et le langage de synchronisation associés. La présentation de chaque exemple s'appuiera sur le squelette du module (au sens de SESAME [MOSS 77]) qui concrétiserait l'exemple étudié dans une réalisation effective.

Enoncé 1 : "Exclusion mutuelle"[DIJK 65] :

- deux procédures accèdent à une même ressource
- à tout instant une seule (au plus) des deux procédures est en cours d'exécution.

```
module MUTEX
    ext procédure A ;
    ext procédure B ;
fin
```

Figure II.17

Enoncé 2 : "Buffer infini"[HABER 72] :

- deux procédures accèdent à un même tampon
- à tout instant le nombre d'exécutions d'une des procédures (RETIRER) (N.B y compris celle éventuellement en cours) doit être inférieure au nombre d'exécutions (N.B : non compris celle éventuellement en cours) de l'autre procédure (DEPOSER)

```
module TAMPON
   :
ext procédure RETIRER (...) ;
   :
ext procédure DEPOSER (...) ;
   :
fin ;
```

Figure II.18

Remarque : On suppose donc implicitement que :

- soit un seul processus exécute RETIRER (resp. DEPOSER),
- soit que l'on impose par ailleurs l'accès aux "pointeurs" de manipulation du tampon en exclusion mutuelle.

Enoncé 3 : "Lecteur-Rédacteur" [COURT 71]:

- deux classes de processus (les lecteurs et les rédacteurs) se partagent une ressource (fichier, par exemple) à laquelle ils accèdent par deux procédures distinctes (resp. LIRE et ECRIRE). Un lecteur ne peut exécuter LIRE que si aucun rédacteur n'exécute ECRIRE, de même, un rédacteur ne peut exécuter ECRIRE que si aucun processus(lecteur rédacteur)n'accède à la ressource.

```
module FICHIER ;
   :
   ext procédure LIRE (...);
   :
   ext procédure ECRIRE (...);
   :
fin ;
```

Figure II.19

Remarques préliminaires

- Dans tous ces exemples "l'état d'occupation" de la ressource considérée ne change qu'au début ou à la fin de l'exécution de chaque procédure d'accès à la ressource (cas qui, nous le verrons, est très fréquent). En conséquence, les seuls points de synchronisation à considérer sont ces débuts et ces fins d'exécution de procédures.

Notation : Le nom associé au début (resp. à la fin) d'une procédure sera noté d(resp.f) suivi de l'initiale du nom de la procédure (le choix des noms mis en jeu éliminent tous risques d'erreurs).

- Les langages de synchronisation sont donnés sous forme de grammaires syntagmatiques.

modèle 1 : "exclusion mutuelle" : $L_s^1 = L(G_1)$ avec $G_1 = (V_T^1, V_N^1, S, R^1)$:

$V_T^1 = \{d_A, d_B, f_A, f_B\}$, $V_N^1 = \{S\}$,

$$R^1 = \left\{ \begin{array}{l} S \to d_A \mid d_B \mid \varepsilon \\ S \to d_A\ f_A\ S \mid d_B\ f_B\ S \end{array} \right\}$$

modèle 2 : "buffer infini." : $L_S^2 = L(G_2)$ avec $G_2 = (V_T^2, V_N^2, S, R^2)$:

$V_T^2 = \{f_d, d_r\}$, $V_N^2 = \{S\}$,

$R^2 = \{ S \to S\ f\ d \quad S \to S\ d_r\ A\ f_d \quad A \to d_r\ A\ f_d \mid \varepsilon \mid AA \}$

modèle 3 : "Lecteur-rédacteur" : $L_S^3 = L(G_3)$ avec $G_3 = (V_T^3, V_N^3, S, R^3)$:

$V_T^3 = \{d_1, f_1, d_e, f_e\}$, $V_N^3 = \{S, A, B, C, D\}$,

$$R^3 = \left[\begin{array}{l} S \quad A.B \\ A \to \varepsilon \mid d_e\ f_e \mid AA \mid C \\ C \to d_1\ C\ f_1 \mid CC \mid \varepsilon \\ B \to \varepsilon \mid d_e \mid Bd_1 \mid\ B\ d_1\ D\ f_1 \\ D \to\ d_1\ Df_1 \mid DD \mid \varepsilon \end{array}\right]$$

Remarque : Tous les langages de synchronisation étudiés ci-dessus ont pu être engendrés par des grammaires hors contextes (voire même régulières (cf.exemple 1)). D'une façon générale, tous les langages associés aux problèmes classiques (producteur-consommateur, contrôle d'une procédure à n points d'accès,...) ont des langages de synchronisation hors-contexte et deviennent même, dans la plupart des cas, réguliers (s'ils ne le sont pas déjà) lorsque l'on donne une borne maximale au nombre de processus du système.

Contrôleur de synchronisation

Présentation : on se propose de construire un organe capable de contrôler l'accès à une ressource. D'après le modèle précédent, un tel organe peut être entièrement représenté par un mécanisme capable de maintenir dans le langage de synchronisation, la chaîne $\tau(t)$ ($\in V^*$, V : vocabulaire de synchronisation) produite par l'activité des processus dans la ressource à partir de l'état initial et jusqu'à l'instant t. Cette modélisation déjà présentée dans le paragraphe précédent (vocabulaire et langage de synchronisation) suppose déjà l'existence d'une exclusion mutuelle élémentaire : "Les franchissements de points de synchronisation ne peuvent être simultanés" ou "étant donnés deux franchissementsde PDS on est toujours capable de décider de leur ordre d'arrivée respectif"; ce qui revient à considérer que les processus présentent successivement au mécanisme de contrôle leurs demandes de franchissement des PDS. L'autorisation de franchissement

d'un point de synchronisation p de nom v à l'instant t est soumise à l'appartenance de la chaîne $\tau(t)v$ au langage de synchronisation de la ressource. L'existence d'un contrôleur est donc équivalente à la récursivité du langage de synchronisation.

Modifications des contraintes imposées par le contrôleur :
Considérons un module M concrétisant une ressource; selon notre approche il comprend donc trois types d'objets : une structure de données, des procédures, un contrôleur de synchronisation. Des points de synchronisation ont été disséminés dans le code des procédures. Supprimons le contrôleur de synchronisation de cette structure et examinons le fonctionnement de ce nouveau module M'. Manifestement, les chaînes $\tau'(t)$ susceptibles d'être produites par M' ne vont pas décrire le monoïde V^* tout entier mais simplement un sous-ensemble de celui-ci : le langage L_p. Ces contraintes sont dues au contrôle interne des procédures du module. (Remarquons que L_p vérifie lui aussi la propriété de préfixe).

- Exemples : Dans de nombreux cas, expliciter ce langage L_p nécessite une analyse fine des procédures et des interactions entre exécutions à travers leurs variables rémanentes (variables internes du module).

  Notation : nous associons à chaque caractère v de V une fonction de V^* dans N (entiers naturels) notée $n_v|\tau|$ qui, à chaque phrase τ de V^*, associe le nombre d'occurrences de v dans τ.

- Exemple 1 : Une procédure a une structure purement séquentielle vis à vis de la synchronisation, si toutes ces exécutions sont formées d'une même suite d'exécutions d'actions séparées par le franchissement de PDS identiques (figure II.20).

```
procédure EX(...);
    début
        Action 0;
        Franchir V1;
        .
        .
        Action i-1;
        franchir Vi;
        .
        .
        Action q-1;
        franchir Vq;
        Action q;
    fin;
```

Figure II.20

Chaque action ne comportant pas de franchissements de PDS, la seule contrainte imposée par Ex s'écrit :

$$\{ \tau \in L_p \quad (i,j) \in \{1,2 \ldots q\} \quad i \leq j \} \Rightarrow \{ n_{v_j}|\tau| \geq n_{v_i}|\tau| \}$$

Dans le cas où toutes les procédures peuvent se mettre sous la forme indiquée par le figure II.20, on peut exhiber une condition nécessaire et suffisante d'appartenance à L_p portant sur les nombres d'occurrences de caractères dans les chaines de L_p .

- notons P_1, $P_2 \ldots P_n$ les procédures du module,
- à chaque procédure $P_i (1 \leq i \leq n)$ on associe la suite des P.D.S. $(v_1^i, v_2^i \ldots v_j^i \ldots v_{qi}^i)$ franchis dans cet ordre lors d'une exécution de P_i (P_i possède q_i P.D.S.).

Appelons P la propriété suivante :

$$\{P(\tau), \tau \in V^*\} \Leftrightarrow \{ \forall i\ 1 \leq i \leq n \quad \forall j,h\ 1 \leq j \leq h \leq q_i \ n_{v_j^i}|\tau| \geq n_{v_h^i}|\tau$$

La propriété caractéristique d'appartenance à L_p s'écrit :

$$\{ \tau \in L_p \} \Leftrightarrow \{\tau = \mu.\nu \Rightarrow P(\mu)\} \quad \mu,\nu \in V^*$$

- Exemple 2 : Considérons une structure comportant des instructions conditionnelles dont les "branches" conduisent à des suites distincte de franchissements de PDS (Figure II.21) .

```
procédure EX2(...);
   début
      Action 0
      franchir V1;
      si(...) alors Action 1; franchir V2;
      sinon         Action 2; franchir V3;
      Action 3;
      Franchir V4;
      fin;
```

Figure II.21

Une des contraintes imposées par EX2 s'écrit :

$$\{\tau \in L_p\} \Rightarrow \{n_{V_2}|\tau| + n_{V_3}|\tau| \leq n_{V_1}|\tau| \text{ et } n_{V_4}|\tau| \leq n_{V_2}|\tau| + n_{V_3}|\tau|\}$$

Ce résultat est facilement généralisable et on peut en déduire une condition nécessaire d'appartenance à L_p. Cette condition nécessaire est formée d'un ensemble d'inéquations linéaires portant sur les nombres d'occurrences de chaque caractère de V dans la chaîne considérée. Cette condition nécessaire peut être renforcée par la propriété de préfixe.

Compte tenu de cette remarque le contrôleur de synchronisation d'une ressource peut se borner à imposer l'appartenance de $\tau(t)$ non pas à L_s mais à un langage L_c tel que

$$L_c \cap L_p = L_s \quad (1)$$

Si l'on ne connaît qu'un majorant de L_p : L'_p (exemple 2), on construira un contrôleur qui se limitera à vérifier l'appartenance de τ à un minorant de L_c : L'_c de telle sorte que :

$$L'_c \cap L'_p = L_s \qquad L_s \subseteq L'_c \subseteq L_c$$

- Exemple 3:"exclusion mutuelle":

$$L_s = (d_A f_A + d_B f_B)^*(d_A + d_B + \epsilon)$$

$$L_p = \{\tau \in V^* \mid \tau = \mu.\nu \Rightarrow \nu \in V^* \Rightarrow \begin{cases} n_{f_A}|\mu| \leq n_{d_A}|\mu| \\ n_{f_B}|\mu| \leq n_{d_B}|\mu \end{cases}\}$$

On peut prendre pour langage L_c le langage suivant :

$$L_c = ((d_A + d_B)(f_A + f_B))^*(d_A + d_B + \epsilon)$$

* Le contrôleur peut être simplifié par réduction du vocabulaire de synchronisation. Cette réduction peut être obtenue grâce à la relation d'équivalence R_V définie sur V de la manière suivante :

$$\{v, v' \in V \quad vR_Vv'\} \overset{\text{def}}{\Leftrightarrow} \{\tau\, v\, \mu \in L_c \Leftrightarrow \tau v'\mu \in L_c\}$$

La relation R_V détermine sur V des noms de PDS "équivalents" vis à vis du langage L_c. Chaque PDS peut recevoir comme étiquette la classe d'équivalence (élément de V/R_{Vv}) à laquelle appartient son nom v. Parallèlement, L_c sera "réduit" en L_c/R_V par homomorphisme. Le nouveau contrôleur se bornera à maintenir la chaîne $\tau(t)/R_V$ dans L_c/R_V. Ce procédé de réduction pourra alors être appliqué à ce nouveau contrôleur et cela itérativement jusqu'à ce que plus aucune réduction ne soit possible.

Exemple : "exclusion mutuelle" ;
On a vu que $L_c = ((d_A+d_B)(f_A+f_B))^*(d_A+d_B+\varepsilon)$

Il vient de façon triviale : $d_A \equiv d_B$ et $f_A \equiv f_B$ (mod. R_V) ;

appelons d et f (resp.) les classes d'équivalences ainsi définies.

Le langage L_c se trouve alors réduit en :

$$L_c/R_V = (df)^*(d+\varepsilon)$$

N.B : Les inéquations linéaires établies dans les exemples 1, 2 et 3 précédents, supposent que tous les P.D.S. portent des noms distincts. Dans le cas où un ou plusieurs noms sont partagés par plusieurs P.D.S, des relations entre les occurrences de ces noms dans les phrases du langage de synchronisation peuvent être déduites des inéquations proposées précédemment par un procédé de "projection" :

Soit $P = (P_1, P_2 \ldots P_n)$ l'ensemble des points de synchronisation et L_s le langage de synchronisation obtenu en considérant P comme un vocabulaire.
A chaque chaîne τ de L_p on peut associer un vecteur $X(\tau)$ de N^n en attribuant à chaque composante $x_i(\tau)$ de X la valeur $n_{p_i}|\tau|$.

Les inéquations précédentes se traduisant sous forme matricielle par une relation de la forme :

$$AX \leqslant 0 \quad (1)$$

Considérons, maintenant, une application D de P dans $V=(V_1, V_2 .. V_q)$. Par l'extension naturelle de D notée D^* le langage L_p est transformé en un langage L_V sur V^*. Désignons par $Y(\mu)$ le vecteur associé à une chaîne μ de L_V ($y_i(\mu) = n_{V_i}|\mu|$). L'application D^* impose une relation linéaire entre $X(\tau)$ et $Y(D(\tau))$ indépendante de τ :

$$Y(D(\tau)) = B\ X(\tau) \qquad (2)$$

avec

$$B_{ij} = 1 \iff D(p_j) = v_i$$

$$B_{ij} = 0 \iff D(p_j) \neq v_i$$

Il est facile d'éliminer X entre les relations (1) et (2) afin d'obten une relation portant sur Y :

$$CY \leq 0 \qquad (3)$$

Cette élimination correspond à une projection du polyèdre (1) suivant les directions exprimées par (2).

* Conclusions : Nous tirerons deux conclusions des deux remarques. précédentes :
  - L'introduction de P.D.S. portant des noms identiques semble tout à fait naturelle.
  - Nous avons vu que le langage de synchronisation L_s est l'intersection de deux langages L_p et L_c. Dans le cas où l'on ne connaît directement qu'un majorant de L_p ($L_{p'}$), il se peut que certaines propriétés ne soient pas prouvables sur $L_{p'} \cap L_c$.

  Dans ce dernier cas,une analyse fine des procédures est nécessaire, ce qui compliquera cette preuve.

  Si l'on veut isoler, dans le contrôleur, le "maximum" d'informations concernant la synchronisation afin de faciliter son étude, il convient de minimiser l'influence des procédures de traitement de cette synchronisation.
  Une solution conforme aux buts recherchés dans la séparation traitement/synchronisation, consiste à se limiter, chaque fois que cela est possible aux distributions de P.D.S. répondant au schéma de la Figure II.20; ce schéma produisant des contraintes directement traduisibles en restrictions sur l'ordre des franchissements des P.D.S.
  Cette restriction amène évidemment des limitations dans l'utilisation des contrôleurs de synchronisation. Nous discuterons de ces limitations au paragraphe II.5.7. Remarquons toutefois que les expressions de chemins [CAMPB 74] et les modules de contrôle [ROVER 77] ont aussi cette limitation.

Caractéristique et construction d'un contrôleur

L'étude précédente a permis de dégager un aspect du contrôleur de synchronisation : l'aspect automate accepteur "incrémentiel" de langage (i.e si l'automate a déjà accepté τ, il est capable de décider de $\{\tau v \in L_s$ (où L_c) $\forall v \in V\}$, et donc d'accepter successivement une suite de caractères de V).

Dans une réalisation réelle un mécanisme capable d'arrêter et de relancer sélectivement les processus cohabitera avec cet automate incrémentiel. Ce mécanisme peut être réalisé grâce à des sémaphores privés [DIJK 68] . Une réalisation possible, programmée en langage type ALGOL, est présentée ci-après.

Soit une machine abstraite dans laquelle sont définis les objets suivants :

- des processus désignés chacun par un nombre entier,
- des sémaphores d'exclusion mutuelle ou privés. A chaque processus i on associe le sémaphore privé S(i),
- des ensembles d'entiers désignés eux-mêmes par un nombre entier et gérés par les procédures suivantes :

  Procédure AJOUTER (entier nprocess, nset);
  co : ajoute l'entier nprocess à l'ensemble nset;

  Procédure OTER (entier nprocess, nset);
  co : si l'ensemble nset n'est pas vide cette procédure supprime un ses éléments et affecte cette valeur à l'entier nprocess;
  Logique procédure VIDE (entier nset);
  co : vraie sss l'ensemble snet est vide;

Considérons une ressource R dont la synchronisation est exprimée par un langage L sur un vocabulaire V formé des N premiers nombres entiers.

$$V = \{1,2 \ldots N\}$$

Dans la réalisation proposée ci-après l'ensemble j d'entiers est utilisé pour mémoriser les processus en attente du franchissement d'un PDS de nom j. Il y a donc autant d'ensembles d'entiers que d'éléments dans V.

Soit une procédure ACCEPTE exécutable en exclusion mutuelle par chaque processus et permettant de décider si un point de synchronisation de nom v est franchissable.

En cas d'autorisation de franchissement d'un PDS de nom v, il y a libération d'un processus demandeur. On considèrera dans ce cas que la procédure ACCEPTE modifie les variables rémanentes pour traduire le passage de la chaîne $\tau(t)$ à $\tau(t)$ v. La cohérence de ces variables rémanentes est garantie par l'exécution de la procédure ACCEPTE en exclusion mutuelle.

Le franchissement d'un PDS de nom Ps par un processus Pr peut se programme selon le schéma de la figure II.22.
Remarquons que ce schéma est très voisin de ceux proposés par[DIJK 68], dont une version plus générale est donnée figure II.23.

```
P(mutex)
si ¬ ACCEPTE (p_s) alors AJOUTER (p_r,p_s)
sinon
    début
    logique STABLE;
    V(S(p_r)); STABLE := faux;
    tant que¬ STABLE faire
        début
        STABLE := vrai;
        pour i := 1 jusqu'à N faire
        si ¬ VIDE (i) et ACCEPTE (i) alors
           début
           entier n;
           STABLE := faux;                P(mutex)
           OTER (n,i);                    "Examen et modification des
           V(S(n));                       variables d'état comprenant
           fin                            des opérations conditionnell
        fin                               sur les sémaphores privés
    fin;                                  d'autres processus et éventu
V(mutex);                                 lement sur le sien propre".
P(S(p_r));                                V(mutex)
                                          P(son propre sémaphore privé
```

Figure II.22

Figure II.23

Nous n'avons donné ici qu'une réalisation parmi d'autres. D'autres structures de données peuvent être envisagées (diverses organisations de files d'attente), ainsi que différents algorithmes (diverses gestions de ces files). On peut, notamment pour des problèmes particuliers, construire des réalisations plus efficaces ne nécessitant pas l'examen de tous les PDS ayant des demandes de franchissement en attente après chaque libération de processus (cf. § II.5.7) .

Remarquons que le choix des processus libérés et, par suite, l'ordre dans lesquels les franchissements vont réellement se produire dépend de la réalisation choisie.

Nous considérons que l'utilisateur d'un tel dispositif ne connaît pas la réalisation effective. Toutes les règles de synchronisation devront être uniquement assurées par la procédure ACCEPTE. Cette procédure est alors la seule partie spécifique d'un problème et constitue donc la seule partie accessible à l'utilisateur.

Dans la suite, nous étudierons donc la spécification de cette procédure.

II - 4.3. Propriétés des langages de synchronisation

Grâce à la propriété caractéristique des langages de synchronisation, on peut facilement découvrir quelques propriétés de ces langages.

Notations : - V, V', V"... désignent des vocabulaires,
- L, L', $L"$... représentent des langages,
- ε est le mot vide,
- LS est l'ensemble des langages de synchronisation,
- $|\tau|$ est la longueur de la chaîne τ.

Rappel : propriété caractéristique :

$$L \subset V^*_{\{L \in LS\}} \iff \{ \tau \in L, \tau = \mu v, \quad v \in V^* \Rightarrow \mu \in L \}$$

A - Propriétés de fermeture

* Fermeture pour l'intersection :

Preuve :

Soient L' et $L"$ langages de synchronisation sur V

Soit $\tau \in L' \cap L"$ avec $\tau = \mu v$ et $v \in V$

$\tau \in L' \Rightarrow \mu \in L'$

$\tau \in L" \Rightarrow \mu \in L"$ $\Rightarrow \mu \in L' \cap L'$

et donc $L' \cap L" \in LS$.

* Fermeture pour la réunion :

Preuve :

Soient L', L'' langages de synchronisation sur V ;
Soit $\tau \in L' \cup L''$ avec $\tau = \mu\, v$ et $v \in V$ .On a
soit $\tau \in L' \Rightarrow \mu \in L'$,
soit $\tau \in L'' \Rightarrow \mu \in L''$

Il s'en suit que $\mu \in L' \cup L''$
et donc, $L' \cup L'' \in LS$.

* Fermeture pour la concaténation

Preuve :
Soient L', L'' langages de synchronisation sur V
Soit $\tau \in L'\ L''$: $(\exists\, \tau' \in L', \tau'' \in L'', \tau = \tau'\,\tau'')$
puisque $\varepsilon\varepsilon \in L' \cap L'' \Rightarrow L'\,L'' \supset L' \cup L''$

Supposons $\tau = \mu\, v, v \in V$:

Trois cas se présentent :

- $\tau'' = \varepsilon \Rightarrow \tau = \tau'$ et si $\tau' = \mu v \Rightarrow \mu \in L' \subset L'\ L''$
- $\tau'' = v \Rightarrow \tau = \tau' v \quad \tau' \in L' \Rightarrow \tau' \in L'\ L''$
- $\tau'' = \mu v \Rightarrow \tau = \tau'\mu v$ $\left.\begin{matrix}\tau' \in L' \\ \mu \in L''\end{matrix}\right\} \Rightarrow \tau'\mu \in L'\ L''$
  $\mu (\neq \varepsilon) \in L''$

Fermeture pour l'opération "" (concaténation dénombrable de Kleene)

Preuve : Soit L un langage de synchronisation sur V :

$\tau \in L^* \Rightarrow \{\, \tau \in V^* \mid \exists n \in N,\ \tau = \tau_1\, \tau_2 \cdots \tau_i \cdots \tau_n,\ \tau_i \in L\ \forall_i \in \{1 \ldots$

Soit $\tau \in L$, deux cas sont à considérer :

- $\tau = \varepsilon$ et donc $\tau = \mu v \Rightarrow \mu \in L$
- $\tau \neq \varepsilon$ et donc $\exists n \in N$ et $\tau_n \neq \varepsilon$, $\tau = \tau_1 \cdots \tau_n$

Posons $\tau \Rightarrow \mu v \Rightarrow \tau_n = \mu_n v$ avec $\mu_n \in L$
$\Rightarrow \mu = \tau_1 \cdots \tau_{n-1}\, \mu_n \in L^*$ ($\tau_i \in L$ et $\mu_n \in L$)

*Fermeture pour les substitutions

Preuve : Soit une substitution f ($V \xrightarrow{f}$ LS sur V').

Soit L un langage de synchronisation sur V.

Considérons $\tau' \in f(L)$ (f, par abus de langage, désigne également l'extension à V^* de f)

$$\tau' \in f(L) \Rightarrow \left\{ \begin{array}{l} \tau \in L,\ \tau = v_1 \ldots v_n,\ \tau' = a_1\, a_2 \ldots a_n \text{ et} \\ a_i \in f(v_i), \forall i \in \{1 \ldots n\} \end{array} \right\}$$

Deux cas se présentent :

- $\tau' = \varepsilon$ et donc $\{\tau' = \mu v, v \in V'\} \Rightarrow \mu \in f(L)$ $(\mu = \varepsilon)$
- $\tau' \neq \varepsilon$ supposons que $a_q \neq \varepsilon$ et $a_k = \varepsilon\ \forall k \geq n$; posons $a_q = \mu_q v$

$\mu_q \in f(v_n)$ car $f(v_n) \in$ LS donc $\tau' = a_1\, a_2 \ldots a_{n-1}\, \mu_q \in f(L)$

B - Propriétés liées à la synchronisation

Les définitions données dans ce paragraphe constituent, en quelque sorte, une première validation du formalisme proposé précédemment en montrant comment des propriétés jugées intéressantes (souhaitables ou non) pour des contrôleurs de synchronisation, se traduisent directement en des propriétés sur les langages de synchronisation associées.

Blocages

- Langage de synchronisation bloqué

  Un langage de synchronisation est dit bloqué si toutes ces chaînes sont de longueur bornée.

  L_s bloqué $\overset{\text{def}}{\Leftrightarrow} \{\exists\, n \in N,\ \forall\, \tau \in L_s \quad |\tau| \leq n\}$

- Impasse d'un langage de synchronisation L_s

  On appelle impasse d'un langage L_s toute chaîne de L_s qui n'est pas préfixe d'une autre chaîne de L_s.

  τ impasse $\overset{\text{def}}{\Leftrightarrow} \{\forall\, \mu \in V^+\ \tau\, \mu \notin L_s\}$

  soit encore (propriété de préfixe):

  τ impasse $\overset{\text{def}}{\Leftrightarrow} \{\forall\, v \in V\ \ \tau\, v \notin L_s\}$

- Un langage de synchronisation est dit blocable s'il contient au moins une impasse.

Les notions de langage bloqué (resp.blocable) formalisent le comportement de modules pour lesquels tous les processus utilisateurs sont obligatoirement (resp.potentiellement) bloqués dans le module sans espoir de déblocage.

Vivacités

- Un élément de v est dit n-vivant ($n \in \mathbb{N}$) pour une chaîne τ du langage L_s si et seulement si :

$$\forall \mu, \tau\,\mu \in L_s,\ \exists \nu, \tau\,\mu\,\nu \in L_s \Rightarrow n_v |\mu.\nu| \geq n$$

Si $\tau = \varepsilon$ (mot vide de V^*) v est dit simplement n-vivant.
Si v est n-vivant pour tout n, v est dit vivant.
Si tous les caractères de ν sont vivants, L_s est dit vivant.

Les notions de vivacités traduisent formellement le risque (ou l'absence de risque) d'étreinte mortelle ou "deadlock". Ces situations provoquent le blocage sélectif de certains processus utilisateurs du module, les autres processus utilisateurs restant actifs .

Coalition

- Un élément v de V est dit indéfiniment retardable pour une chaîne τ de L_s si pour tout entier n on peut trouver une chaîne w s'écrivant $\tau\mu$ avec $|\mu| > n$ et $n_v |\mu| = 0$.

$$\left\{\begin{array}{l}\text{v indéfiniment retardable}\\ \text{pour une chaîne } \tau \text{ de } L_s\end{array}\right\} \overset{\text{def}}{\Longleftrightarrow} \left\{\begin{array}{l}\forall n \in \mathbb{N},\ \exists \mu,\ \tau\mu \in L_s \text{ et } |\mu| > n\\ \qquad\qquad \text{et } n_v |\mu| = 0\end{array}\right.$$

Cette notion peut être rapprochée de la notion de famine ou de coalition [DIJK 71] et on peut énoncer :

$$\left\{\begin{array}{l}L_s \text{ est sans famine}\\ \qquad\text{ou}\\ \text{(sans coalition) pour une chaîne}\\ \tau \text{ de } L_s\end{array}\right\} \Longleftrightarrow \left\{\begin{array}{l}\forall v \in V,\ v \neg \text{ indéfiniment}\\ \text{retardable pour } \tau\end{array}\right.$$

La notion de non-coalition vient renforcer les notions de vivacité en rendant obligatoires les franchissements de certains P.D.S.,qui n'étaient que possibles dans le cas de la vivacité.

La propriété d'être sans famine pour un langage L_s permet de se prémunir contre des coalitions de processus ou des fonctionnements"aberrants"ou "inéquitables"du contrôleur de synchronisation lui-même.

En effet, un contrôleur de synchronisation se trouve souvent face à des situations dans lesquelles il doit choisir entre plusieurs processus libérables (non déterminisme). La grande liberté laissé dans la réalisation des contrôleurs, permet d'en imaginer certains qui favorisaient systématiquement la même classe de processus. On peut, dans ce cas, sans connaître la réalisation d'un contrôleur, supprimer ces fonctionnements aberrants potentiels en agissant directement au niveau du langage de synchronisation.

Remarques

- Les notions que nous avons formalisées dans le cadre des langages de synchronisation, sont à rapprocher de celles énoncées et approfondies par Sifakis [SIFAK 79]dans le cadre des systèmes de transitions.
- Les propriétés ci-dessus sont des propriétés propres au module étudié. Elles ne supposent rien sur l'utilisation faite de ces modules par les processus. Ces derniers ne devraient pas être ignorés dans l'examen des propriétés globales d'un système.

II - 4.4. Conclusion

Dans ce paragraphe, nous avons introduit une nouvelle formalisation de la synchronisation entre procédures d'un module.
Comme tous modèles formels, celui-ci permet d'appliquer les résultats de la théorie à laquelle il appartient,à la réalité qu'il représente. Plus précisément, en ce qui concerne notre approche, on peut songer à une classification des problèmes de synchronisation calquée sur celle des automates et des langages formels. Cette formalisation conduit notamment à remarquer que la plupart des problèmes de synchronisation classiques correspondent à des langages de synchronisation hors-contextes.

Par rapport à d'autres modèles, (par exemple[MUNTE 78]) notre propositio autorise des définitions formelles directes de propriétés jugées généralement importantes dans le cadre de la synchronisation (II-4.3).

II - 5. Outils et méthodologie de spécification de la synchronisation

II - 5.1. Introduction

Nous avons vu (§II.4) que la spécification d'un problème de synchronisation se ramène à la description d'un "automate accepteur incrémentiel" d'un langage possédant la propriété de préfixe.
Plusieurs solutions existantes semblent très proches de notre formalisme : les expressions de chemins[CAMPB 74],[HABER 75],[FLON 76],[CAMPB 76],[LAUER les mots de synchronisation[ROUC 78],[ROUC 79] .

Nous allons donc, tout d'abord, examiner ces deux outils à la lumière de la modélisation proposée au § II.4.
Le § II.5.4 sera consacré à la recherche d'un outil "naturel" de spécification de la synchronisation. Le caractère "naturel" correspond à l'idée intuitive d'une bonne adaptation à la gamme des problèmes visés; cette qualité subjective sera confirmée par la description d'une approche méthodologique de cette spécification, basée sur cet outil et par l'utilisation de ce dernier dans quelques exemples.

II - 5.2. Expressions de chemins

Au cours de sa présentation (cf. II.3.4) nous avions déjà émis des critiques dues au nombre pléthorique des "primitives" de cet outil ou à l'abandon du principe de séparation dans certaines extensions. Les critiques que nous formulerons ici, ont une origine plus profonde directement reliée à la nature même de la synchronisation, ou dues à des manques de cohérence internes.

- Les auteurs des expressions de chemin ont, du moins à l'origine, choisi d'exprimer la synchronisation par des contraintes sur l'ordre des exécutions des procédures du module. Ce choix est source de plusieurs inconvénients.
  - . Toutes les procédures mises en jeu dans une expression de chemin régulière doivent être exécutées en exclusion mutuelle puisque cette expression spécifie les séquences d'exécutions possibles. Cette limitation conduisant à une certaine inefficacité a pu être levée de deux façons différentes :

- soit par l'introduction de primitives supplémentaires qui rompent ce caractère séquentiel ({ } par exemple).
- soit par l'adjonction de nouvelles procédures dans le modul Ces procédures "vides",(i.e. sans corps ni paramètres),ne sont là que pour "simuler" des PDS manquants. Dans la plupart des exemples[BEKK 74] ces procédures jouent le rôle de PDS "encadrant" une procédure du module original.

Cette adjonction de procédures vides montre que le choix d'une politique de synchronisation rejaillit sur le découpage en procédures de la partie traitement; cela semble montrer que la séparation, partie synchronisation/partie traitement, n'est pas assurée et donc que la possibilité de modification d'une de ces parties indépendamment de l'autre reste douteuse.

- Un certain manque de cohérence car, bien que s'inspirant ostensiblement de la théorie des langages et des automates réguliers, il n'est pas possible d'associer un langage sur le vocabulaire choisi (i.e celui des noms des procédures à synchroniser) à de nombreuses constructions à base d'expressions de chemins :
  - opérateur d'exécutions simultanées "{ }" ,
  - plusieurs expressions dans un même module (expression multiple),
  - ...

* On peut également leur reprocher une certaine inefficacité" théoriq

En effet, si l'on se limite aux seuls opérateurs réguliers (cf.II.3.4) certains problèmes intrinsèquement d'état fini ne peuvent être directement résolus.

Exemple : Producteurs-consommateurs "buffer à N cases". On peut facilement montrer que ce problème est d'état fini soit en examinant le langage de synchronisation, soit en remarquant qu'il existe une solution utilisant directement des sémaphores dont les variables entières restent bornées[DIJK

Nous terminerons ce paragraphe sur des remarques inspirées par des exemples d'utilisation de ces expressions de chemin. Il nous semble que l'emploi de ces dernières conduisent dans de nombreux problèmes :

- Soit à des spécifications de synchronisation obscures, notamment lors de l'utilisation d'extensions nombreuses ou d'expressions multiples. Dans ces derniers cas les expressions de chemin se revèlent être à notre avis d'un emploi difficile de la part d'un utilisateur non averti.
- Soit à des solutions comportant la définition préalable d'objets voisins des sémaphores. Dans ce cas, les expressions de chemins ne reflètent que très partiellement le schéma de synchronisation réalisé et le gain en clarté par rapport aux solutions sémaphoriques s'avère pratiquement nul.

II - 5.3. Mots de synchronisation

Signalons immédiatement que l'approche ici proposée est tout à fait identique à la nôtre. Dans ce dispositif l'état de "synchronisation" de la ressource est conservé dans des variables "rémanentes" à valeurs dans un monoïde libre Σ^*. Σ représente un alphabet d'événements tout à fait semblable à notre vocabulaire de synchronisation. De même, l'analogie entre présence d'un PDS et présence d'une primitive ininterruptible de manipulation de mots est totale. Ce mode de spécification de la synchronisation en permet une expression structurée, en associant à chaque contrainte, un mot de synchronisation destiné à mémoriser toutes les informations nécessaires à son contrôle.

Nous ferons néanmoins deux réserves concernant cette approche :

- La première motivée par la part d'arbitraire dans le choix des primitives de base qui, malgré sa complétude (i.e puissance "identique" à celle d'une machine de Turing), semble très orientée vers la résolution des problèmes classiques donnés en exemple dans [ROUC 78] Cette opinion paraît confirmée par l'extension donnée à la primitive AJOUTE dans [ROUC 79] et, visiblement destinée à résoudre le problème des lecteurs-rédacteurs[COURT 71] dans la variante de Hoare [HOARE 74] .
- En second lieu il nous semble que, bien que plus "parlant"" que les expressions de chemin, notamment par la possibilité d'utiliser un vocabulaire aussi étendu que désiré, cet outil ne puisse pas être employé directement par un programmeur inexpérimenté. En un mot, son aspect "naturel" ne nous semble pas garanti, à cause notamment de l'utilisation imposée d'un nombre restreint de primitives, certaines n'ayant d'ailleurs pas une sémantique "transparente".

II - 5.4. Un outil pour la description de la synchronisation

Introduction

L'outil découle directement d'une méthode naturelle, c'est-à-dire adaptée au type d'énoncé des problèmes classiques. Une telle méthode présentée ci-dessous s'inspire de l'étude précédente et ne s'applique donc qu'aux problèmes où la séparation partie contrôle/partie traitement est possible. Ce qui exclut par exemple les problèmes où les règles de synchronisation font intervenir des paramètres des procédures à synchroniser.

La méthode comprend quatre phases :

Première phase

Cette phase repose sur l'hypothèse selon laquelle la liste des PDS nécessaires se déduit "naturellement" de l'énoncé même du problème étudié. Le programmeur pourra alors choisir les noms de ces PDS.

Deuxième phase

Au cours de cette phase le programmeur choisit les variables de synchronisation, c'est-à-dire leur nombre, leur type, leur structure, la signification de leur contenu (sémantique). Ce choix de variables est initialement guidé de façon intuitive par les contraintes mêmes de l'énoncé.

Troisième phase

Le programmeur dans cette phase essaiera de traduire les contraintes du texte en termes d'algorithmes de calculs de prédicats sur les variables précédemment définies. Une telle programmation permet la spécification formelle des conditions de franchissement des PDS définis lors de la première phase. Une tentative infructueuse pourra remettre en cause les choix effectués lors de la seconde phase. Cette itération sera poursuivie jusqu'à réalisation satisfaisante de cette troisième phase.

Quatrième phase

Cette phase consiste simplement en la programmation des transformations du contenu des variables de synchronisation qui seront effectuées à l'issue du franchissement de chaque PDS. Cette programmation est guidée par la signification de chaque variable telle qu'elle a été définie dans la deuxième phase.

Le contrôleur de synchronisation

Nous proposons dans la suite une description possible de cet outil à l'intérieur d'un langage de programmation de haut niveau classique modifié pour la circonstance.
Ce langage comprendra la notion de "type" ou de "module"[MOSS 77],[HABER 75]
Nous ne nous intéresserons pas ici à la déclaration de type à plusieurs niveaux ni au problème de paramétrage de ces divers types[MOSS 77],
La structure de ces types sera analogue à celle porposée par HABERMANN dans [HABER 75]. On trouvera donc successivement :

- Une déclaration de la structure de données associée à ce type décrit en terme de types déjà existants.
- La description des algorithmes de manipulation de cette structure. La synchronisation apparaît ici par l'intermédiaire de la primitive FRANCHIR suivie du nom de PDS que l'on désire franchir.
- La description de la synchronisation par l'écriture d'un contrôleur de synchronisation construit sur le vocabulaire de synchronisation défini dans la partie précédente.
  Cette description pourra être décomposée en trois parties :
  . Déclaration des variables de synchronisation qui sont du type du langage de base. On désigne par E le produit cartésien des domaines respectifs de chaque variable.
  . Spécification des valeurs initiales des variables de synchronisation.
  . Une liste de triplets :
  + nom de PDS,
  + condition de franchissement : prédicat sur E,
  + transformation : application de E dans E .

Ces différentes fonctions seront données par un algorithme de calcul décrit dans un langage de programmation classique (Algolw, PL1...).

Exemple : Lecteurs-rédacteurs
(Voir énoncé au paragraphe II.4.2).

L'état d'occupation de la ressource ne change qu'aux instants où un processus commence ou termine son accès à la ressource. On a donc quatre PDS :

- DLIRE, FLIRE : début et fin d'exécution de LIRE.
- DECR, FECR : début et fin de ECRIRE.

Appelons NREDAC une variable qui, à tout instant, contient le nombre de rédacteurs exécutant ECRIRE. La condition associée à DLIRE, équivalente à la première contrainte, s'écrit NREDAC = 0. Pour traduire la deuxième contrainte, on introduit une variable NLECT qui, à tout instant, contient le nombre de lecteurs qui exécutent LIRE. La condition de franchissement de DECR, équivalente à la deuxième contrainte, s'écrit : NREDAC = 0 et NLECT = 0.

Les deux autres points de synchronisation FLIRE, FECR sont toujours franchissables; leur condition associée est donc toujours vraie. Les transformations associées aux divers PDS maintiennent la cohérence des variables NREDAC et NLECT (Figure II.24).

```
type FICHIER - JOURNAL
    Structure
     fichier FILE
    fin structure
    opération
     procédure LIRE(...);
        début
        franchir DLIRE; <corps procédure LIRE>; franchir FLIRE;
        fin;
    procédure ECRIRE (...);
        début
        franchir DECR; <corps procédure ECRIRE>; franchir FECR;
        fin;
    fin opération
    synchronisation
        entier NLECT init 0, NREDAC init 0;
        DLIRE : condition  : NREDAC = 0;
                transformation : NLECT : = NLECT +1;
        FLIRE : condition : vrai;
                transformation : NLECT := NLECT -1;
        DECR : condition : NREDAC = 0 et NLECT = 0;
               transformation : NREDAC := NREDAC +1;
        FECR : condition : vrai;
               transformation : NREDAC := NREDAC -1;
    fin synchronisation
fin type FICHIER - JOURNAL ;
```

Figure II.24

Relations contrôleurs-modules de contrôle

Si l'on fait abstraction du mécanisme de queues, les modules de contrôle apparaissent comme des contrôleurs de synchronisation auxquels on apporterait les trois restrictions suivantes :

1) Choix des points de synchronisation

Chaque procédure comprend trois PDS et trois seulement :

- demande d'exécution de la procédure
- début d'exécution de la procédure
- fin d'exécution de la procédure.

Ce choix respecte la "pure séquentialité" des procédures vis à vis de la synchronisation. Cette restriction ne nous paraît que peu limitative Elle peut être facilement contournée par un simple découpage des procédures originales du module.

2) Choix des variables d'état ("compteurs d'état")

Ce choix impose le nombre des variables d'état et surtout l'interprétation de leur contenu respectif. Il est évident qu'en utilisant seulement ces variables on ne va pas pouvoir mémoriser toute la suite des franchissements de PDS déjà effectués au cours de la vie antérieure du système. Cette remarque montre les limites imposées par un tel choix des variables d'état et prouve que ce mécanisme ne pourra pas contrôler certains schémas de synchronisation, l'information nécessaire à ces contrôles ayant disparu .

Remarque : Cette "incomplétude" justifie l'introduction de la variable λ permettant de connaître la dernière procédure exécutée dans [ROVER1]

Les auteurs expliquent cette restriction en remarquant que ces variables suffisent à l'expression de la plupart des schémas de synchronisation classiques.

3) Choix des conditions associées aux PDS

Celles-ci sont limitées à une certaine classe de prédicats sur l'ensemble des valeurs des différents compteurs. Cette restriction peut s'expliquer par un argument analogue au précédent.

En résumé, on peut dire que malgré de nombreuses limitations, parfois arbitraires, la plupart des problèmes classiques sont aisément résolus et les solutions produites très claires.

Conventions simplificatives

A - Compte tenu des remarques précédentes, une bonne solution pour la réalisation d'un langage capable d'exprimer la synchronisation semble être la déclaration implicite, pour chaque procédure à synchroniser, des trois PDS utilisés dans les modules de contrôle. Dans les exemples suivants,ces points, pour une procédure P, sont désignés par :

- dem.P : demande d'exécution de P;
- deb.P : début d'exécution de P;
- fin.P : fin d'exécution de P.

De même, vue la généralité de leur emploi,on admet la déclaration implicite des compteurs d'état. Pour une procédure P, ces compteurs sont désignés par les noms suivants :

$\#\#$ req.P, $\#\#$ aut.P, $\#\#$ term.P, $\#\#$ wait.P, $\#\#$ act.P

Exemple : Lecteurs-Rédacteurs (voir section précédente). Grâce aux déclarations implicites proposées ci-dessus, cet exemple se réduit au schéma de la figure II.25.

```
type FICHIER-JOURNAL
    structure
        fichier FILE
    fin structure
    opération
        procédure LIRE (...);
            début
            <corps procédure LIRE>;
            fin;
        procédure ECRIRE (...);
            début
            <corps procédure ECRIRE>;
            fin;
    fin opération
    synchronisation
        deb. ECRIRE : condition : (## act. ECRIRE=0) et (## act. LIRE
        deb. LIRE : condition : (## act.ECRIRE=0);
    fin synchronisation
fin type FICHIER-JOURNAL
```

Figure II.25

Signalons enfin que ces compteurs permettent l'expression explicite et directe des contraintes issues de la "pure séquentialité" des procédures synchronisées.
Pour une procédure P cette propriété se traduit simplement par la relation:

$$\#\# \text{ req.P} \geq \#\# \text{ aut.P} \geq \#\# \text{ term.P}$$

Ces relations sont importantes pour les preuves formelles de programmes parallèles. La plupart des travaux dans ce domaine introduisent ces variable a postériori.

Exemples :

- variables auxiliaires:[OWIC 76] ,
- variables np, nv, nf :[HABER 72] ,
  (cf. II.5.5 : émulation de sémaphore).

Nous présenterons au paragraphe II.5.5 une émulation complète (y compris le mécanisme de queues) des modules de contrôles à l'aide de notre contrôle

B - Plus généralement on convient d'associer à chaque nom v de PDS, trois compteurs d'état notés ## req.v, ## wait.v, ## term.v, désignant respectivement :

- ## term.v : le nombre de franchissements de PDS de nom v enregistrés depuis l'initialisation du système ,
- ## wait.v : le nombre de processus en attente de franchissement d'un PDS de nom v,
- ## term.v : le nombre de demandes de franchissement d'un PDS de nom v enregistré depuis l'initialisation du système .

On conviendra également d'omettre les conditions "toujours vérifiées" (prédicats identiquement vrais) ainsi que les transformations "vides" (applications identitées).

C - Toutes les extensions proposées ici ne sont aucunement nécessaires mais elles conduisent souvent à un allègement notable de l'écriture. Elles seron utilisées dans les exemples qui vont suivre (II.5.5).

II - 5.5. Exemples d'utilisation

Nous avons classé les exemples présentés ici en deux parties correspondant à deux utilisations distinctes de notre outil.

Spécification de quelques primitives de synchronisation

*Sémaphores

Un sémaphore S peut être facilement simulé par un de nos modules :

```
module SEMAPHORE_S ;
    opération;
        procédure P;
        procédure V;
    fin opération ;
    synchronisation
        entier E init E0;
        dem.P transformation : E:=E-1;
        deb.P : condition : E ≥ 0;
        deb.V : transformation : E:=E+1;
    fin synchronisation;
fin module;
```

Figure II.26

On remarque que les trois points de synchronisation correspondent aux trois opérations élémentaires définies sur les sémaphores par [BELP 74] . Le théorème du sémaphore [HABER 72] s'exprime directement à l'aide des compteurs implicites et se démontre facilement en remarquant :

($\#\#$ aut.P = $\#\#$ term.P; $\#\#$ req.P $\geq$ $\#\#$ aut.P; $\#\#$ term.V = $\#\#$ req.V = $\#\#$ aut.V)

$\#\#$ wait.P = 0 $\Rightarrow$ E $\geq$ 0

$\#\#$ wait.P > 0 $\Rightarrow$ E+ $\#\#$ wait.P=0

Ce théorème s'écrit :

$\#\#$ term.P = min ($\#\#$ req.P, E0 + $\#\#$ term.V)

On remarque que np(S) = $\#\#$ req.P, nv(S) = $\#\#$ term.V et nf(S) = $\#\#$ term.P .

*Moniteurs

Dans ce paragraphe nous proposons une méthode systématique de traduction des moniteurs à l'aide de notre outil. Cette traduction consiste, dans un premier temps, à spécifier une distribution de PDS dans le corps des procédures du moniteur. Cette distribution s'effectue de la façon suivante :

- on place un PDS de nom ENTREE-MONITEUR (resp. SORTIE-MONITEUR) au début (resp. à la fin) de chaque procédure du moniteur.
- chaque utilisation de la primitive C. SIGNAL est remplacée par le franchissement de deux PDS de noms SIGNAL-C (propre à la condition C) et MONIT-WAIT communs à toutes les primitives .SIGNAL.
- chaque utilisation de la primitive C.WAIT est remplacée par le franchissement de deux P.D.S., l'un commun à toutes les primitives .WAIT: MONIT-FREE et l'autre propre à la condition C : WAIT-C (remarquer la symétrie entre .WAIT et .SIGNAL).

Cette spécification est complétée par la donnée d'un contrôleur de synchronisation :

```
Synchronisation  MONITEUR;
     logique MUTEX init vrai, STOP init faux,.., C init faux,..;
ENTREE-MONITEUR : condition : MUTEX
                  transformation : MUTEX := faux;
SORTIE-MONITEUR : transformation : if ## wait.MONIT-WAIT > 0 then STOP :
                                   STOP:= faux  else MUTEX := vrai;

C.WAIT  { MONIT-FREE : transformation if ## wait. MONIT.WAIT > 0 then STOP:=faux
                                   else MUTEX := vrai;
        { WAIT-C     : condition : C;
                       transformation C := faux;

C.SIGNAL{ SIGNAL-C   : transformation : if ## wait.WAIT-C > 0 then
                                   begin STOP:=vrai; C:=vrai; end;
        { MONIT-WAIT : condition :¬STOP ;
                       transformation : STOP := vrai ;
fin synchronisation;
```

Figure II.27

La variable MUTEX sert à assurer l'exclusion mutuelle du moniteur:

MUTEX<=>aucun processus n'est actif dans le moniteur.

La variable STOP sert à libérer un par un les processus mis en attente consécutivement à l'exécution d'un C.signal effectif :

STOP, associée aux PDS MONIT-WAIT et MONIT-FREE, permet la gestion de ce dispositif .

Avant de relâcher l'exclusion mutuelle, un processus vérifie s'il n'y a pas un processus en attente de franchissement de MONIT-WAIT. A chaque condition C est associée une variable logique C permettant de simuler le fonctionnement des attentes et des réveils explicites du moniteur.

*Modulesde contrôle

Dans la comparaison des modules de contrôle avec notre outil, nous avons fait abstraction du mécanisme de "queues". Nous allons maintenant simuler ce mécanisme à l'aide de notre outil. Pour cela nous associons à chaque queue Q un nom de PDS noté également Q et une variable logique GUICHET-Q. Grâce à ce dispositif nous pourrons examiner successivement les processus placés dans une même queue.

La traduction sera alors immédiate :

- en tête de la procédure P appartenant à la queue Q on placera les deux demandes de franchissement de PDS successives :
  franchir Q; franchir deb.P;
  Cette dernière venant remplacer la demande de franchissement implicite du PDS de nom deb.P placéepar défaut, en tête de la procédure P.

- la synchronisation est alors assurée par le contrôleur suivant :

```
synchronisation
pour chaque queue Q :
    logique .. GUICHET-Q init Vrai,...;
    Q : condition : GUICHET-Q;
        transformation : GUICHET-Q := faux;
pour chaque procédure P de la queue Q :
    deb.P : condition : identique à celle du module de contrôle ;
            transformation : GUICHET-Q := vrai;
fin synchronisation ;
```

Figure II.28

Utilisation de notre outil sur quelques exemples

Pour montrer l'intérêt de notre dispositif nous avons été conduit à choisir des exemples relativement complexes; les problèmes classiques étant résolus de façon plus ou moins heureuse par la plupart des primitives existantes. Ce qui, d'ailleurs, n'est pas surprenant puisque ces primitives ont souvent été définies en fonction des exemples qu'elles auraient à résoudre.

*Variantes du problème des Lecteurs-Rédacteurs

- Exemple 1 : Accès au fichier par "grappes"

  Aux contraintes de base exposées précédemment, (§II.5.5) nous désirons ajouter une contrainte pour établir une stratégie d'allocation de la ressource plus fine entre ces deux classes de processus.

  On dit qu'une classe de processus A a la "priorité" sur une autre classe B, si aucun processus de la classe B ne peut acquérir la ressource tant qu'il y a un processus de la classe A en attente de cette ressource.
  Nous supposerons qu'à l'instant initial la priorité est attribuée arbitrairement à un des deux classes (lecteurs, par exemple). Cette priorité devra être inversée chaque fois que les deux conditions suivantes seront vérifiées :

  . Un processus prioritaire au moins a pu commencer à travailler sur la ressource.
  . Aucun processus prioritaire n'est en attente de la ressource.

  On remarque que cette stratégie permet de découper en "grappes" de processus d'une même classe, le flux des processus accédant à la ressource. Les processus d'une classe profitant des "trous" dans l'arrivée des processus de l'autre classe pour accaparer la ressource

  Une variable logique appelée PRILECT indiquera la classe des processus prioritaires. Si PRILECT est vraie la priorité est donnée aux lecteurs, sinon elle est donnée aux rédacteurs (Fig.II.29).

  Si l'on souhaite que le basculement de priorité s'effectue non pas au début, mais à la fin de l'accès du dernier processus d'une grappe il suffit de modifier la partie synchronisation en ajoutant les sections entre guillemets de la figure II.29.

```
synchronisation
    logique PRTLECT init vrai;
    comment : initialement la priorité est donnée aux lecteurs;
    deb.LIRE : condition : (## act. ECRIRE = 0) et (PRTLECT ou
                           ## wait. ECRIRE = 0);
  "fin lire":: transformation : si PRTLECT et (# wait. LIRE = 0)
                                alors PRTLECT := faux;
  deb.ECRIRE : condition : (# act. ECRIRE = 0) et (# act.LIRE = 0.
                           et ¬PRTLECT ou # wait.LIRE = 0);
  "fin.ECRIRE" : transformation : si ¬PRTLECT et (# wait.ECRIRE = (
                                  alors PRTLECT : = vrai;
fin synchronisation
```

Figure II.29

- Exemple 2 : Variante de Hoare : problème de réservation aérienne [HOARE 74].

Cette variante est basée sur une solution donnant une priorité aux rédacteurs. Mais pour éviter une coalition de ces derniers, Hoare propose la règle suivante : les lecteurs en attente au moment où se termine l'accès d'un rédacteur ont priorité sur la prochaîne opération d'écriture.

De façon analogue au cas précédent, le flux des processus de lecture va être découpé en grappes par les opérations d'écriture. Ce découpage va nécessiter l'emploi de deux PDS successifs à l'entrée de la procédure LIRE (on utilisera les PDS implicites dem.LIRE et deb.LIRE) Les processus lecteurs pris dans une même grappe seront bloqués entre ces deux PDS. La partie synchronisation peut alors être facilement codée en remarquant qu'elle peut fonctionner sous deux modes : un mode "normal" dans lequel la partie synchronisation reste conform à la solution de base et un mode "forcé" dans lequel une grappe de lecteurs accapare la ressource, même si un rédacteur est en attente. Le mode sera indiqué par une variable MODE susceptible de prendre deux valeurs : NORMAL, ou FORCE (variable "SCALAR" de PASCAL).

```
synchronisation
    type TOTO = (FORCE,NORMAL);
    var MODE : TOTO init NORMAL;
  dem.LIRE : condition : MODE = NORMAL;
  deb.LIRE : condition (MODE = NORMAL) et (## act.ECRIRE = 0) et
                       (## wait.ECRIRE = 0)) ou (MODE = FORCE);
             transformation : si (MODE = FORCE) et (## wait-LIRE = 0)
                              alors MODE:= NORMAL;
  deb.ECRIRE : condition : ((MODE = NORMAL) et ((## act.ECRIRE = 0)
                           et (## act.LIRE = 0);
 "fin-ECRIRE":"transformation : si (## wait.LIRE¬ = 0)alors MODE:=FORCE
fin synchronisation ;
```

Figure II.30

Par suppression de l'expression entre guillemets, le découpage en grappes des lecteurs n'est plus rythmé par la fin mais par le début des écritures.

*Variantes au problème de l'exclusion mutuelle [DIJK 65] .

- Exemple 1 : Partage d'une ressource entre deux classes de processus de priorités différentes sans coalition; l'accès à la ressource se faisant en exclusion mutuelle.
  Comme pour l'exemple précédent, deux classes de processus, A et B accèdent à une ressource par l'intermédiaire de deux procédures, resp. ACCES-A et ACCES-B.
  On donne priorité aux processus de la classe A, mais, pour éviter une coalition des processus de cette dernière, qui empêcherait indéfiniment l'accès de la ressource aux processus de la classe B, on limite, en cas d'attente de processus de la classe B, à N (fixe) le nombre maximum de processus de la classe A pouvant accéder à la ressource entre deux accès des processus de la classe B. La variable COMPTE du contrôleur sert à compter le nombre de processus de la classe A accédant à la ressource, à partir de l'instant où un processus de la classe B est en attente. Cette variable est remise à zéro après chaque accès d'un processus de la classe B.

  On remarque dans cet exemple, l'utilisation d'une variable unique pour assurer l'exclusion mutuelle (MUTEX). Celle-ci aurait pu être assurée en utilisant l'invariant classique [ROVER1 77] :

  ## act.ACCES A + ## act.ACCES B <= 1

ce qui aurait permis l'élimination des références explicites aux points de synchronisation de noms fin. Accès-A et fin. Accès-B.

```
synchronisation
    entier COMPTE init 0
    logique MUTEX init vrai;
    deb. ACCES-A : condition : (# wait.Acces-B = 0 ou COMPTE < N)
                               et MUTEX;
                   transformation : si (# wait.ACCES-B > 0) alors
                                    COMPTE := COMPTE +1;
                                    MUTEX := faux;
    fin.ACCES-A : condition : vrai;
                  transformation : MUTEX := vrai;
    deb.ACCES-B : condition : (# wait.ACCES-A = 0 ou COMPTE = N)
                              et MUTEX;
                  transformation : COMPTE := 0; MUTEX := faux;
    fin.ACCES-B : condition : vrai;
                  transformation : MUTEX := vrai;
fin synchronisation
```

Figure II.31

Si l'on prend N égal à l'unité, la solution proposée ci-dessus correspond à la solution du problème du "ramonte-pente" exposée dans [VERJU 78].

II - 5.6. Implantation de contrôleurs de synchronisation

Implantation directe

L'implantation directe d'un contrôleur de synchronisation peut être faite en deux étapes :

1) Déclaration des variables de synchronisation.
2) Ecriture d'une procédure franchir ayant pour paramètres un identificateur de processus et un nom de PDS. Un appel à cette procédure remplacera systématiquement toutes les occurrences de la primitive franchir dans le corps des procédures de manipulation du module auquel appartient le contrôleur de synchronisation. Toutes les exécutions de la procédure franchir devront être mutuellement exclusives pour sauvegarder la cohérence des variables de synchronisation. A ces dernières on rajoute une structure de donnée, mémorisant, à chaque instant, l'ensemble des processus en attente.

Notations

- PDS(q) : Retourne le nom du PDS que désire franchir le processus q préalablement placé en attente.
- C(p), T(p) : Désignent respectivement la fonction procédure et la procédur calculant la condition de franchissement associée au nom de PDS p et effectuant la transformation des variables d'état relative au franchissement d'un PDS de nom p.
- LIBERER(n) : L'effet de cette procédure consiste à "relancer" le processus n et de le supprimer de l'ensemble des processus en attente.
- BLOQUER(n) : Procédure inverse de libérer; elle a pour effet de suspendre le processus n et de le placer dans l'ensemble des processus bloqués dans le module.
- nprocess : Désigne le type attribué aux identificateurs de processus.
- nPDS : Désigne le type des noms de PDS.
- ens-de-processus : Type générant des variables susceptibles de renfermer un ensemble d'identificateurs de processus.
- ELEMENT(W) : Procédure retournant un élément de l'ensemble représenté par W et le supprimant de W.

```
procédure FRANCHIR (nprocess n, nPDS p);
    CØ le processus n désire franchir un PDS de nom p;
    début
    si  ¬ C(p) alors BLOQUER (n);
    Sinon ACTIVER (p);
    fin FRANCHIR ;
```

Figure II.32

La procédure ACTIVER effectue la transformation sur les variables d'états correspondant au franchissement d'un PDS dont le nom p lui est passé en paramètre. Cette procédure, pouvant, le cas échéant, "libérer" des processus, et, donc provoquer de nouveaux franchissements, sera écrite récursivement.

```
procédure ACTIVER (n-PDS p);
    début
    ens-de-processus WAIT-L;
    T(p);
    CONSTRUIRE (WAIT-L, p); CØ W ensemble de processus en attente;
    Tantque ¬WAIT-L¬ = Ø faire
        début
        n-process q;
        q := ELEMENT (WAIT-L);
        si C (PDS(q)) alors
                      debut
                        LIBERER(q); ACTIVER(PDS(q));
                      fin
        fin;
        WAIT-L := Ø
    fin ACTIVER;
```

Figure II.33

L'appel de procédure CONSTRUIRE (WAIT-L,p); a pour effet de placer dans WAIT-L tous les processus susceptibles d'être "réveillés" consécutivement au franchissement d'un PDS de nom p.
Cette procédure utilise toutes les variables rémanentes du module et en particulier la structure de données, mémorisant l'ensemble des processus en attente.
Une réalisation immédiate et correcte de CONSTRUIRE consiste à placer tous les processus en attente dans WAIT-L.

Remarques : Le programme de la Figure II.33 est intéressant de par sa généralité : notre proposition (§II.4.2), aussi bien que celle de [SVL 76], n'en sont que des raffinements.

- Notre proposition avait pour but de présenter un exemple simple de ce que pouvait être un contrôleur de synchronisation.
  Dans cet exemple, tout se passe comme si, d'une part WAIT-L contenait tous les processus en attente et si, d'autre part, la fonction ELEMENT balayait WAIT-L en respectant un certain ordre dans les noms des PDS dont le franchissement retardait certains processus.

- Dans la proposition de [SVL 76], l'examen des demandes de franchissement pendantes est effectué en donnant, en quelque sorte, la priorité aux demandes les plus anciennes. Par cet ordre d'examen, on cherchait à éviter certains problèmes de famine imputables à des fonctionnements "pathologiques" du contrôleur.

Il nous semble qu'il serait dangereux de porter à la connaissance d'un programmeur les détails de l'implantation sur laquelle il travaille. Cette connaissance l'encouragerait à écrire des spécifications qui utiliseraient des particularités de cette implémentation. Le fonctionnement réel ne serait pas entièrement donné par le programme ce qui, de toute évidence, nuirait à la clarté de la spécification.
Remarquons enfin que la récursivité de la procédure ACTIVER peut facilement être supprimée en employant les règles proposées dans [VEIL 75].

Améliorations

L'amélioration de l'algorithme représenté par les procédures FRANCHIR et ACTIVER, sans en modifier le fonctionnement exact passe par la suppression de WAIT-L de tous processus n dont la valeur du prédicat associé C(PDS(n)) peut être connue sans nécessiter une évaluation.
Pour éviter des évaluations inutiles on peut "profiter" de toutes informat: recueillies au cours des exécutions de ces procédures. Ces informations peuvent être symbolisées par les assertions distribuées à travers le progr: me de ces procédures.

```
procédure FRANCHIR (nprocess n, nPDSp);
    début
    {po}
    si ¬C(p) alors  {p1} BLOQUER(n);
    sinon {p2}  ACTIVER(p);
    fin;

procédure ACTIVER (nPDSp);
    début
    ens-de-processus WAIT-L;
    {p3} T(p); { p4 } CONSTRUIRE (WAIT-L,P);
    Tantque WAIT_L ¬= ∅ faire
      début
      n process q;
      q := ELEMENT(W);
      {p5}
      si C(PDS(q)) alors
```

```
            debut
            {p6}  LIBERER(q); ACTIVER (PDS(q));
             fin;
        sinon {p7};
        fin;
    WAIT-L := ∅;
fin ACTIVER;
```

Figure II.34

Ces assertions ne sont évidemment pas indépendantes :

- {p1} ⊆ {p3},
- {p5} ⊆ {p3}
- etc .

Ces assertions portent sur les variables du programme. Elles peuvent être "renforcées" en utilisant des variables auxiliaires; dans notre cas on peut penser à des variables de synchronisation redondantes. Pour limiter les évaluations des conditions C on peut utiliser les assertions précédentes en remplaçant l'évaluation d'une condition C par celle d'une condition C' telle que :

C' ∧ {p5} <=> C (1)

Le remplacement de C par C' se paye du maintient de la cohérence des éventuelles variables redondantes. On peut se satisfaire de résultats partiels.

- $\overline{C}$ ∧ {p5} => C (2)
- $\underline{C}$ ∧ {p5} =>¬C (3)

Des techniques analogues peuvent être utilisées pour éliminer des processus inutiles de l'ensemble WAIT-L. Cette élimination pourra s'effectuer au moment de la construction de WAIT-L (relation (4)) ou chaque fois qu'un calcul de condition rend une valeur fausse (relation(5)):

- C' ∧ {p4} =>¬C (4)
- C" ∧ {p7} =>¬C (5)

Ces relations permettent l'élimination de WAIT-L de tous les processus en attente de franchissement d'un PDS de nom v tel que $C(v) \equiv C$

Ces résultats peuvent être facilités par la connaissance d'un invariant J portant sur les variables du contrôleur de synchronisation.
En effet, on peut ici ainsi remplacer la recherche d'une condition C' telle que :

$C' \wedge \{p\} \Rightarrow C$ (forme générale des relations 2,3,4,5)

par celle d'une condition $\tilde{C}$ telle que :

$J \wedge \tilde{C} \wedge \{p\} \Rightarrow C$

On conçoit que plus l'invariant J connu sera "fort", plus la recherche de $\tilde{C}$ en sera facilitée ($\tilde{C}$ pourra être d'autant plus faible donc généralement plus facile à découvrir et à calculer). De même l'assertion {p} pourra être parallèlement affaiblie.

Améliorations directes

Plusieurs améliorations peuvent être introduites sans aucune adjonction de variables auxiliaires :
On cherchera par exemple des relations du type :

$\{p = p_o\} \wedge \{p4\} \Rightarrow \neg C(k)$ (2')

ou $\{PDS(q) = p_o\} \wedge \{p7\} \Rightarrow \neg C(k)$ (3')

qui permettent "l'épuration" de la liste WAIT-L des processus n en attente dans le module et tels que {PDS(n) = k}.
Ces éliminations seront respectivement effectuées à la construction de WAIT (indiquée par {p4} sur la figure II.34) ou après une évaluation négative d'une condition C(PDS(q)) (signalée par {p7} sur la même figure).

De même, on peut rechercher des relations du type:

$\{PDS(q) = p_o\} \wedge \{p5\} \Rightarrow \neg C(p_o)$ (4')
$\{PDS(q) = p_o\} \wedge \{p5\} \Rightarrow C(p_o)$ (5')

qui permettent d'accélérer l'évaluation de C(PDS(q)) pour certains choix de q (i.e PDS(q) = p_o

L'intérêt de ces méthodes réside dans le fait qu'elles concentrent la plupart du travail dans la démonstration statique de certaines propriétés signalées plus haut, en ne rejetant dans la partie vérification dynamique que des tests très simples : PDS(q)=p_o , etc.

Ces démonstrations peuvent être effectuées en utilisant à la place des {pi} des assertions simplifiées (affaiblies) p'_i :{pi} =>{p'_i}

$$\{p'_4\} = \bigvee_{i=1}^{k} (\{p=pi\} \wedge Post_{T(pi)}(C(pi)))$$

$$\{p'_7\} = \{p'_4\} \wedge \bigvee_{j=1}^{k} \{PDS(q) = pj\} \wedge \neg C(pj)$$

$$\{p'_5\} = \{p'_4\}$$

Notation : - le vocabulaire de synchronisation est noté:
VS = {pl,... pk}

Les formules 2', 3', 4', 5' conduisent respectivement à la recherche des formules identiquement fausses contenues dans les formules suivantes en supposant connu un invariant J.

(2") : $J \wedge post_{T(pi)} (C(pi) \wedge J) \wedge C(pj)$ $\quad i,j \in \{1...k\}$

(3") : $J \wedge post_{T(pi)} (C(pi) \wedge J) \wedge \neg C(pl) \wedge C(pj)$ $\quad i,j,l \in \{1...k\}$

(4") : $J \wedge post_{T(pi)} (C(pi) \wedge J) \wedge \neg C(pj)$ $\quad i,j \in \{1... k\}$

(5") : Les processus éliminés de WAIT-L par la relation 5' auraient déjà été éliminés par la relation 2' car nous avons pris $\{p'_5\} = \{p'_4\}$.

Exemple d'utilisation : lecteurs-rédacteurs (Fig.II.24)

Synchronisation integer NREDAC init 0, NLECT init 0;

$C_{(dlire)}$: NREDAC = 0;

$T_{(dlire)}$: NLECT := NLECT+1;

$C_{(flire)}$: Vrai;

$T_{(flire)}$: NLECT := NLECT-1;

$Q_{(decr)}$: (NREDAC = 0) V(NLECT = 0);

$T_{(decr)}$: NREDAC := NREDAC+1;

$C_{fecr)}$: Vrai;

$T_{fecr)}$: NREDAC := NREDAC-1;

fin synchro

Figure II.25

En supposant connu un invariant I, d'ailleurs non directement inductif:

$I = (NREDAC \geq 0 \wedge NLECT \geq 0)$

$$I \wedge POST_{Tdlire}(Cdlire \wedge I) \supset NREDAC = 0 \wedge NLECT > 0 \supset \neg Cdecr$$

On saura donc éliminer les processus en attente d'écriture grâce aux relations 4'.

$I \wedge Post_{decr}(Cdecr \wedge I) \Rightarrow \{NREDAC = 1 \wedge NLECT = 0\} \Rightarrow (\neg Cdecr \wedge \neg Cdlire)$

On éliminera donc tous les processus en attente de lecture aussi bien que les processus en attente d'écriture de la liste WAIT_L qui sera alors vidée complètement).

Remarque : On peut toujours utiliser comme invariants des relations déduites de l'hypohtèse de "pure séquentialité" des primitives d'un module.

Cette hypothèse conduit à des contraintes linéaires entre quantités représentant des nombres de franchissement de PDS.

Dans l'exemple précédent il vient :

$$\#\# dem.dlire \geq \#\# term.dlire \geq \#\# dem.flire \geq \#\# term.flire \geq 0 \quad (1)$$

$$\#\# dem.decr \geq \#\# term.decr \geq \#\# dem.fecr \geq \#\# term.fecr \geq 0 \quad (2)$$

Initialement nous avons $NREDAC = NLECT = \#\# dem.dlire = \#\# term.dlire = 0$

Il est facile de montrer par induction que :

$$Nlect = \#\# term.dlire - \#\# term.flire$$

et donc par (1) il vient $NLECT \geq 0$

On montrerait de même que $NREDACT \geq 0$ grâce à (2). L'invariant précédent en serait donc déduit.

Améliorations indirectes

Ces améliorations sont obtenues en augmentant le nombre de variables de synchronisation. Ces variables supplémentaires permettrons de maintenir ces informations acquises en cours de fonctionnement. Ces informations en quelque sorte redondantes, pourrons intervenir dans des améliorations de l'implantation du contrôleur.

Avec les seules variables originales, on assiste à des pertes d'information

- Chaque fois qu'il y a destruction de la liste WAIT-L (appel récursif de franchir).
- Chaque fois qu'il y a évaluation d'une condition le résultat de cette évaluation est "oubliée" après son application immédiate (5').

Exemple :

On peut essayer de "propager" une partie de cette information par exemple à l'aide d'un tableau de variables logiques (de nom VAL) servant à mémoriser les résultats d'évaluations antérieures de conditions.

i.e $VAL(i) \Rightarrow \neg C(pi)$

On pourrait alors profiter de relations telles que :

$$post_{T(pi)}(C(pi) \wedge \neg C(pj) \wedge I) \wedge I \Rightarrow \neg C(pl) \quad (resp.C(pl)) \qquad (6)$$

voir plus généralement de :

$$post_{T(pi)}(C(pi) \wedge \bigvee_{pj \in S \subset Vs} \{\neg C(pj)\} \wedge I) \wedge I \Rightarrow \neg C(pl) \quad (resp.C(pl)) \qquad (7)$$

Ces relations, utilisées aux points étiquetés par les assertions {p4} {p7}(resp. {p5}), conduiront à une élimination de processus de la liste WAIT-L (resp. à une accélération de l'évaluation de la condition C(PDS(p))).

D'autres adjonctions sont aussi possibles; on peut notamment penser à transmettre des informations entre appels récursifs de FRANCHIR:

- transmettre des informations sur la chaîne de PDS antérieurement franchis ,
- utiliser les valeurs précédentes de VAL(i),
- etc... .

II - 5.7. Limitations et extensions

Limitations

La principale limitation de notre mode d'expression provient de l'impossibilité de faire dépendre les contraintes de synchronisation de la valeur des paramètres d'appel des procédures du module (i.e tous les processus demandant le franchissement d'un même PDS sont indistinguables).

Cette limitation peut se justifier par les deux arguments suivants :

- Malgré ces limitations la plupart des problèmes classiques peuvent être résolus.

Notons que, s'il existe quelques problèmes réels qui ne peuvent l'être directement, tous le seront indirectement (cf. ci-dessous "simulation de sémaphores").

- Nous avons défendu une vision "ressource" de la synchronisation (§II.3.3). Dans cette optique, une ressource est capable d'assurer un ensemble de services (procédures externe du module matérialisant la ressource); services qu'il convenait de synchroniser indépendamment des processus demandeurs et donc des éventuels paramètres d'appels.

Solutions de substitution

Dans cette classe, nous distinguerons deux méthodes, qui ne demandent aucune adjonction à notre proposition.

Succédannés par "projection"

Cette méthode consiste à "éclater" chacune des procédures dont les valeurs de certains paramètres interfèrent avec la synchronisation. Chaque procédure de ce type se voit donc remplacée par autant de procédures que de valeurs dans le produit cartésien des domaines de ces paramètres.

Cette "projection" permet l'élimination des paramètres délicats et par suite la mise en oeuvre de notre méthode de synchronisation entre les diverses procédures ainsi crées.Cette "élimination" n'est évidemment possible que si les domaines de chacun des paramètres "de synchronisation" sont finis. L'utilisation d'une telle solution peut être facilitée par des mécanismes d'expansions implantés au niveau du langage par l'utilisation d'un macro-générateur.
L'inconvénient de cette méthode réside dans l'augmentation du volume de code machine résultant de cette multiplication des primitives.

Simulation de sémaphores

Cette solution est généralement employée lorsque l'outil de synchronisation utilisé ne permet pas de résoudre le problème posé et que la méthode précédente reste inacceptable.

Le problème de "l'allocateur de ressources"[MOSS 77], [BEKK 74] dans le cas des modules de contrôle ou des expressions de chemins, fournit un bon exemple d'emploi de cette technique.

Cette méthode comprend deux étapes :

. Définition de modules simulant le fonctionnement de sémaphores.
. Résolution des problèmes à l'aide de ces objets. Les solutions obtenues sont calquées sur les solutions sémaphoriques.

Dans ce type de solution, le gain en clarté, obtenu par une expression de haut niveau de la synchronisation, reste limité.

Extensions

Introduction explicite de processus de contrôle

Le contrôleur serait un nouveau constituant de programme au même titre que les modules ou les processus. Ce contrôleur pourrait être essentiellement formé de trois types d'objets :

- une file de demandes de services,
- un ensemble de processus internes au contrôleur,
- un ensemble de variables internes du contrôleur.

(Cf. fig.II.36)

Les processus utilisateurs interagissent avec les processus internes en déposant dans la file tous les paramètres de la demande de service qu'il désire voir remplir par le contrôleur.

Les processus internes viendrons alors choisir dans cette file, la demande qu'ils se proposent d'accomplir. Ces processus internes pourront alors être synchronisés par n'importe quel mécanisme de synchronisation et,en particulier, le nôtre. On peut imaginer deux catégories de requêtes de la part desprocessus utilisateurs : synchrones et asynchrones, suivant que le processus demandeur attend ou n'attend pas une réponse à sa demande.

Une telle méthode serait favorisée par l'introduction dans le langage d'écriture d'objets spécialisés (contrôleur...).
La désignation des différentes demandes de services pourrait, par exemple, s'inspirer de la notion de "Manager" proposée par [SILB 77], ce qui conjointement à l'emploi d'un langage évolué, aboutirait à une bonne protection entre usagers (Figure II.37) .

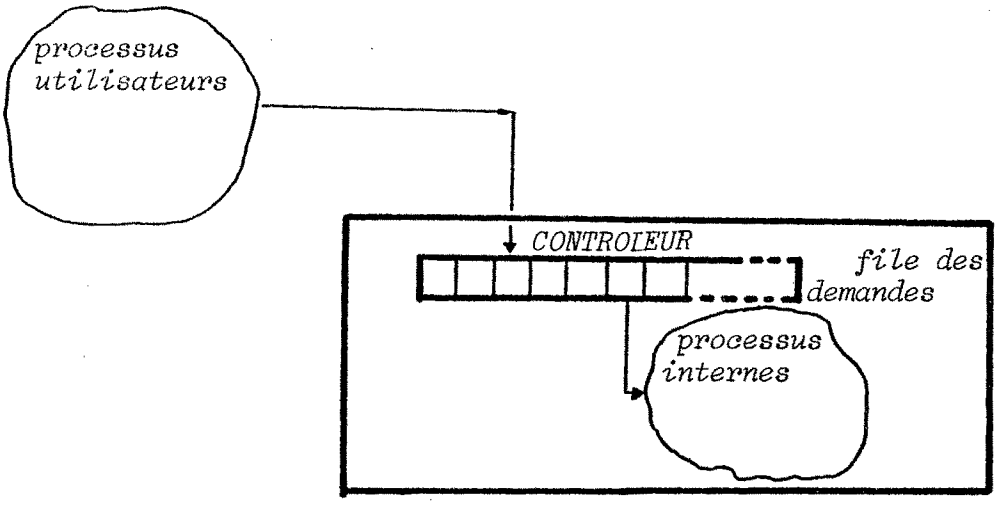


Figure II.36

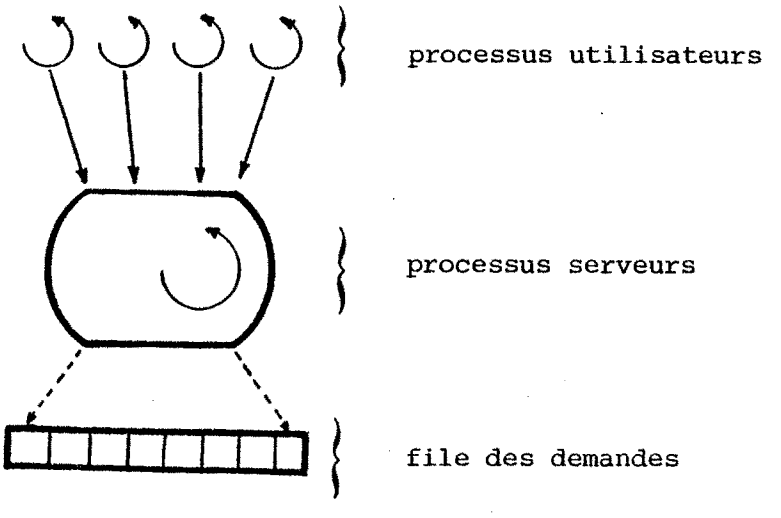


Figure II.3

Introduction explicite des paramètres des procédures dans la synchronisatic

Cette introduction est proposée dans [BDM 78] . Elle ne semble véritablement intéressante que lorsque chaque paramètre intervenant dans la synchronisation est passé par valeur, empêchant ainsi toutes interactions entre partie traitement et partie contrôle. Néanmoins cette méthode risque de se heurter à un problème de désignation de paramètres lorsque les contraintes de synchronisation dépendent non seulement du paramètre propre à une demande mais également de paramètres de demandes antérieures ou simultanées (Exemple : problème de priorité d'accès dépendant de dates critiques). La mémorisation d'information sur ces paramètres dans les variables de synchronisation du contrôleur constitue une solution à ce problème.

III - CONCLUSION

A l'issue d'une revue critique des principaux outils de synchronisation logiciels existants,nous proposons une formalisation de la synchronisation à l'intérieur d'un module.
Cette formalisation conduit à deux résultats distincts.

- D'une part une critique plus précise des outils modulaires d'expression de la synchronisation et de la méthode ayant pour certaines d'entre elles, présidée à leur genèse. Cette méthode consiste à rajouter la primitive ad-hoc chaque fois qu'apparaît un problème que l'on ne peut pas ou que l'on ne sait pas résoudre à l'aide des primitives de base. Ces auteurs ne se préoccupent pas de la nécessité ou de la généralité de l'emploi de ces primitives. Cette attitude a conduit à des outils boursouflés auxquels, périodiquement, on rajoute une primitive. Les avantages et les principes de l'outil original sont alors progressivement remis en question et finissent par disparaître complètement.

- D'autre part, devant les faiblesses des primitives classiques, cette étude propose un outil général d'expression de la synchronisation limité aux problèmes autorisant la séparation entre partie traitement et partie contrôle.
  Les modèles obtenus à l'aide de cet outil apparaissent comme une généralisation des modules de contrôle [ROVER2 77]. Mais, contrairement à ceux-ci, le programmeur peut traduire tout problème proposé à l'aide du nombre de variables qu'il désire, ayant la signification et le type qu'il souhaite et portant des noms qu'il a lui-même choisi. Cette liberté, laissée à l'utilisateur, conduit à une expression rapide et naturelle de la synchronisation, mais provoque en revanche la multiplication des solutions données à un même problème. Néanmoins cette liberté permet l'adaptation de la description d'une solution au raisonnement qui l'a amenée, la multiplicité des solutions ne faisant que traduire la multiplicité des raisonnements possibles pour la résolution d'un problème.

Un des buts avoués de nombreux travaux consiste en la recherche de "preuves des propriétés jugées souhaitables dans les schémas de synchronisation. Ces propriétés, brièvement mentionnées au paragraphe II.4.3 ne reçoivent de preuves pratiques que pour des exemples de taille réduite (comparables aux exemples présentés au paragraphe II.5.6).

Le but ultime, dans ce domaine, reste toujours la preuve de toute propriété concernant la synchronisation pour un système complet, problème de taille bien supérieure à celle des exemples ordinairement traités. Pour dépasser ces difficultés, une solution actuellement envisageable consiste à découvrir un "principe de récurrence" permettant de déduire une preu e globale (i.e au niveau du système) d'un ensemble de preuves locales (i.e. au niveau d'un module).
L'approche que nous suggérons en guise de conclusion se situe dans le cadre d'une conception modulaire (Fig.II.38). Chaque module M est géré suivant une politique de synchronisation S_M.

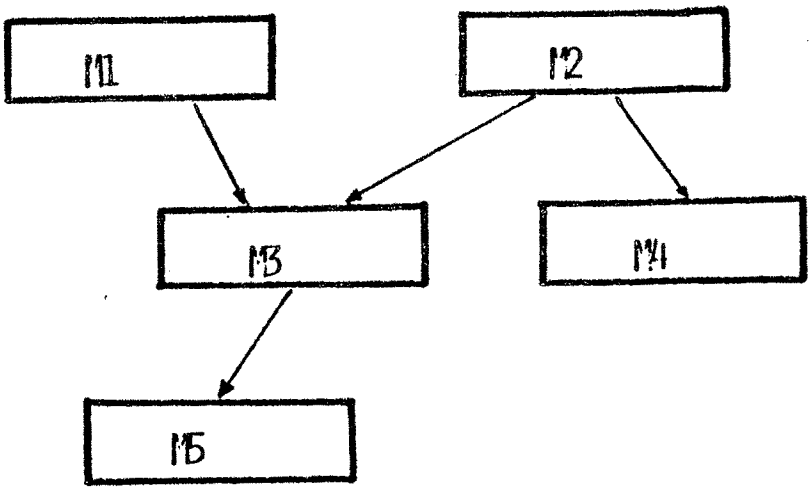


Figure II.38

La conception d'un système vérifiant une certaine propriété P se ramène à la résolution de la question suivante :

"Pour quels types de synchronisation locale, et pour quels schémas d'interconnexion P est-elle vérifiée globalement ?"

Cette question pourrait être abordée en deux étapes :

- Preuve globale en utilisant une "vision simplifiée" de chaque module au sein d'un modèle de l'interconnexion.
- Démonstration de la cohérence entre "vision simplifiée" et implantati réelle, complètée par une preuve locale.

La résolution des problèmes posés par cette approche pourrait s'appuyer sur la théorie des langages et des automates en profitant des deux remarques suivantes :

- Les langages de synchronisation ont la propriété de préfixe.
- Tous les schémas de synchronisation classiques correspondent à des langages hors contextes (voire même réguliers).

REFERENCES BIBLIOGRAPHIQUES : Ch. I

[ANC 69] F. ANCEAU, P. LIDDEL, J. MERMET & C. PAYAN
CASSANDRE: A Language to Describe Digital Systems, Application to logic design
Proceedings of the 3rd International Symposium on Computers and Information Science, Miami, F.10,Decembre 1969

[BELL 71] G. BELL & A. NEWELL
Computers Structures: Readings and Examples
McGraw-Hill Book Company, New-York, 1971.

[POT 77] D. POTIER
Modèles à files d'attente et gestion des ressources dans les systèmes informatiques
Thèse de Doctorat es Sciences Mathématiques, soutenue le 15 Janvier 1977, USMG-INP Grenoble.

REFERENCES BIBLIOGRAPHIQUES : § II.1

[AMDHA 64] G. M. AMDHAL, G. A. BLAAUW & F. P. BROOKS Jr
Architecture of the I.B.M. System 360
I.B.M. Journal, April 1964, pp 87-101.

[ANDL 78] S. ANDLER
Synchronization Primitives and the Verification of Concurrent Programs
Proceedings of the second International Symposium on Operating Systems, I. R. I. A.,Le Chesnay, Octobre 1978.

[CROCU 75] CROCUS
Systèmes d'exploitation des calculateurs
DUNOD Editeur, Paris, 1975.

[DIJK 65] E. W. DIJKSTRA
Solution of a problem in concurrent programming control
Com. A. C. M., (8,9), Septembre 1965, p 569.

[MOSS 77] J. MOSSIERE
Méthodes de conception des systèmes d'exploitation
Thèse de Doctorat d'Etat, USMG-INP Grenoble, Septembre 1977

[MUNTE 78] T. MUNTEAN
Expression de la synchronisation par contraintes
Thèse de Doctorat de 3° Cycle, USMG-INP Grenoble, 19 Juin 1978 .

REFERENCES BIBLIOGRAPHIQUES : § II.2

[BELP 74] G. BELPAIRE & J. P. WILMOTTE
Correctness of realisation of levels of abstraction in operating systems
Congrés sur les aspects théoriques et pratiques des systèmes d'exploitation, I.R.I.A.,Paris, Avril 1974 .

[CROCU 75] CROCUS
Systèmes d'exploitation des calculateurs
DUNOD Ed., Paris, 1975 .

[DEN 71] P. J. DENNING
Third generation computer systems
A.C.M. Computing Surveys, V 2,N°4,Decembre 1971, pp 175-216.

[DIJK 67] E. W. DIJKSTRA
Cooperating sequential processes
Dans "PROGRAMMING LANGUAGES", F. GENUYS Ed., Academic Press,19

[DIJK 68] E. W. DIJKSTRA
The structure of the "THE" Multiprogramming System
Com. A.C.M., (11,5), May 1968, pp 341-346.

[HANS 70] P. B. HANSEN
The nucleus of a multiprogramming system
Com. A.C.M., V 13, N°4, Avril 1970 .

[MOSS 77] J. MOSSIERE
Méthodes de conception des systèmes d'exploitation
Thèse de Doctorat d'Etat, USMG-INP Grenoble, Septembre 1977 .

REFERENCES BIBLIOGRAPHIQUES : § II.3

[ANDL 78] S. ANDLER
Synchronization Primitives and the Verification of Concurrent Programs
Proceedings of the second internationnal Symposium on Operating Systems, I.R.I.A., Le Chesnay, Octobre 1978 .

[BEKK 74] Y. BEKKERS
A comparison of two high level synchronizing concepts
Department of computer science, Technical Report, The Queen's University of Belfast, May 1974.

[BELP 74] G. BELPAIRE & J. P. WILMOTTE
Correctness of realisation of levels of abstraction in operating systems
Congrés sur les aspects theoriques et pratiques des systèmes d'exploitation, I.R.I.A.,Paris, Avril 1974.

[CAMPB 74] R. H. CAMPBELL & A. N. HABERMANN
The specification of process synchronization by Path Expression
Lecture notes in computer science, Editeurs G. GOOS & J. HARTMANIS, pp 89-102,V 16,Springer Verlag 1974.

[CAMPB 76] R. H. CAMPBELL
Path Expressions: a technique for specifying process synchronization
Ph. D. Thesis, Computing Laboratory, The University of Newcastle Upon Tyne, Newcastle Upon Tyne, England August 1976.

[COURT 71] P. J. COURTOIS, F. HEYMANS & D. L. PARNAS
Concurrent control with readers and writers
Com. A.C.M. (14,10), October 1971, pp 667-668.

REFERENCES BIBLIOGRAPHIQUES : § II.3 (suite)

[COURT 72] P. J. COURTOIS, F. HEYMANS & D. L. PARNAS
Comments on "A comparison of two synchronising concepts"
Acta Informatica 1, pp 190-199, 1972.

[CROCU 75] CROCUS
Systèmes d'exploitation des calculateurs
DUNOD Editeur, Paris, 1975.

[DAHL 70] O. J. DAHL, B. MYRHAUG & K. NYGAARD
SIMULA 67, The Common Base Language
Pub. S-22, Norvegian Computing Center, 1970.

[DARON 78] P. DARONDEAU
Types et objets dans un système multi-langages
Thèse de doctorat es Sciences Mathématiques
soutenue le 9 mars 1978, USMG-INP Grenoble.

[DIJK 65] E. W. DIJKSTRA
Solution of a problem in concurrent programming control
Com. A.C.M., (8,9),Septembre 1965, p 569.

[DIJK 67] E. W. DIJKSTRA
Cooperating Sequential Processes
Dans "PROGRAMMING LANGUAGES", F. GENUYS Ed,,
Academic Press, 1967.

[DIJK 68] E. W. DIJKSTRA
The structure of the "THE" Multiprogramming System
Com. A.C.M., (11,5), May 1968, pp 341-346.

[DIJK 71] E. W. DIJKSTRA
Hierarchical Ordering of Sequential processes
Acta Informatica 1,2, 1971

REFERENCES BIBLIOGRAPHIQUES : § II.3 (suite)

[FLON 76] L. FLON & A. N. HABERMANN
Toward the construction of verifiable software systems
Proceedings of A.C.M. Conference on Data, SIGPLAN Notices, March 1976, pp 141-148.

[HABER 75] A. N. HABERMANN
Path Expressions
Technical Report, Department of Computer Science, Carnegie-Mellon University,Pittsburgh, June 1975.

[HANS 72] P. BRINCH HANSEN
A comparison of two synchronizing concepts
Acta Informatica, 1, pp 190-199 .

[HANS 78] P. BRINCH HANSEN
The Architecture of Concurrent Programms
Prentice Hall, 1978 .

[HOARE 71] C. A. R. HOARE
Towards a theorie of parallel programming
Proceedings of the International Seminar on Operating Systems Techniques, Belfast, Northern Ireland, August-september 1971, Edited in "Operating System Techniques", by HOARE & PERROT,Academic Press, London, 1972, pp 61-71.

[HOARE 74] C. A. R. HOARE
Monitors: An operating system structuring concept
Com. A.C.M., V 17, N°10, october 1974, pp 549-557.

REFERENCES BIBLIOGRAPHIQUES : § II.3 (suite)

[KESS 77] J. L. W. KESSELS
An alternative to event queues for synchronization in Monitors
Com. A.C.M., V 20, N°7, July 1977, pp 500-503.

[KOSA 73] S. R. KOSARAJU
Limitations of DIJKSTRA'S Semaphore Primitives and Petri Nets
Proceedings of the A.C.M. SIGOPS Symposium on operating system principles, New-York, October 1973, pp 122-126.

[LAUER 77] P. E. LAUER & M. W. SHIELDS
Abstract specification of ressource accessing disciplines: Adequacy, Starvation, Priority and Interrupts
Comptes rendues des journées d'étude sur les méthodes de synchronisation et de programmation globales en temps réel, A.F.C.E.T.,Paris, Novembre 1977.

[MOSS 77] J. MOSSIERE
Méthodes de conception des systèmes d'exploitation
Thèse de Doctorat d'Etat, USMG-inp Grenoble, Septembre 1977

[MUNTE 78] T. MUNTEAN
Specification de la synchronisation par contraintes
Thèse de 3° Cycle, USMG-INP Grenoble, 19 Juin 1978.

[PARN 75] D. L. PARNAS
On a solution to the cigarette smokers' problem (without conditional statements)
Com. A.C.M., V 18, N°3, March 1975, pp181-183.

REFERENCES BIBLIOGRAPHIQUES : § II.3 (suite)

[PATIL 71] S. S. PATIL
Limitations and Capapilities of Dijkstra's Semaphore Primitives for Co-ordination amongst Processes
M.I.T., Cambridge, Mass., Project MAC, Computation Structures Group Memo 57, February 1971.

[PULOU 77] J. PULOU
Etude du problème de la synchronisation inter-processus
Rapport de D.E.A. Genie Informatique, INP Grenoble, Juin 77.

[ROVER1 77] P. ROBERT & J. P. VERJUS
Toward autonomous descriptions of synchronization modules
Rapport de recherche IMAG N°47, Octobre 76
Proceedings of the 1977 I.F.I.P. congress, Toronto, pp 981-986, North-Holland Pub. Comp..

[ROVER2 77] P. ROBERT & J. P. VERJUS
Expression autonome de la synchronisation des processus concurrents
Comptes rendus des journées d'etude sur les methodes de synchronisation et de programmation globale en temps réel, A.F.C.E.T., Paris , Novembre 1977.

[SHRIV 76] S. K. SHRIVASTAVA
Systematic programming of scheduling algorithms
Software Practice & Experience, V 6, pp 357-370,

[VANTI 72] H. VANTILBORGH & A. VAN LAMSVEERDE
On an extension of Dijkstra's semaphore primitives
Information Processing Letters, 1, pp 181-186,

REFERENCES BIBLIOGRAPHIQUES : § II.3 (fin)

[WIRTH 77] N. WIRTH
MODULA, a language for modular multiprogramming
Software - Practice & Experience, V 7, 1977, pp 3-35.

REFERENCES BIBLIOGRAPHIQUES : § II.4

[BBV 76] Y. BEKKERS, J. BRIAT & J. P. VERJUS
Expression et decomposition du contrôle du parallélisme dans un système
Rapport de Recherche I.M.A.G., RR-66, Grenoble, Janvier 1977.

[CAMPB 74] R. H. CAMPBELL & A. N. HABERMANN
The specification of process synchronization by Path Expressions
Lecture notes in Computer science, Editeurs G. GOOS & J. HARTMANIS, pp 89-102, V 16, Springer Verlag 1974 .

[CAPL 78] M. CAPLAIN
Langage de Spécifications
Thèse de Doctorat es Sciences soutenue le 20 Decembre 1978, INP Grenoble.

[COURT 71] P. J. COURTOIS, F. HEYMANS & D. L. PARNAS
Concurrent control with readers and writers
Com. A.C.M. (14,10), October 1971, pp 667-668.

[DIJK 65] E. W. DIJKSTRA
Solution of a problem in concurrent programming control
Com. A.C.M., (8,9), Septembre 1965, p 569.

[DIJK 68] E. W. DIJKSTRA
The structure of the "THE" Multiprogramming System
Com. A.C.M., (11,5), May 1968, PP 341-346.

[DIJK 71] E. W. DIJKSTRA
Hierarchical Ordering of Sequential Processes
Acta Informatica , 1,2, 1971.

[MOSS 77] J. MOSSIERE
Méthodes de conception des systèmes d'exploitation
Thèse de Doctorat d'Etat, USMG-INP Grenoble, Septembre 1977.

REFERENCES BIBLIOGRAPHIQUES : § II.4 (suite & fin)

[MUNTE 78] T. MUNTEAN
Spécification de la synchronisation par contraintes
Thèse de 3ième Cycle, USMG-INP Grenoble, 18 Juin 1978 .

[ROVER 77] P. ROBERT & J. P. VERJUS
Toward autonomous descriptions of synchronization modules
Rapport de recherches I.M.A.G.,RR N°47, Octobre 76 & Proceedings of the 1977 I.F.I.P. Congress, Toronto, pp 981-986, North-Holland Pub. Comp. .

[SIFAK 79] J. SIFAKIS
Le contrôle des systèmes asynchrones : concepts, propriétés, analyse statique
Thèse de Doctorat es Sciences Mathématiques, soutenue le 25 Juin 1979 , USMG-INP Grenoble .

REFERENCES BIBLIOGRAPHIQUES : § II.5

[BEKK 74] Y. BEKKERS
A comparison of two high level synchronizing concepts
Department of computer science, Technical Report, The Queen's University of Belfast, May 1974.

[BELP 74] G. BELPAIRE & J. P. WILMOTTE
Correctness of realisation of levels of abstraction in operating systems
Congrés sur les aspects théoriques et pratiques des système d'exploitation, I.R.I.A., Paris, Avril 1974.

[BBV 76] Y. BEKKERS, J. BRIAT & J. P. VERJUS
Expression et décomposition du contrôle du parallélisme dans un système
Rapport de recherche I.M.A.G., RR-66, Grenoble, Janvier 1977.

[BDM 78] Y. BEKKERS, P. DARONDEAU, T. MUNTEAN
Types de synchronisation et types abstraits concurrentiels
Rapport de Recherche I.M.A.G., RR-108, Grenoble, Janvier 1978.

[CAMPB 74] R. H. CAMPBELL & A. N. HABERMANN
The specification of process synchronization by Path Expressions
Lecture notes in computer science, Editeurs G. GOOS & J. HARTMANIS, pp 89-102, V 16, Spinger Verlag 1974.

[CAMPB 76] R. H. CAMPBELL
Path Expressions: a technique for specifying process synchronisation
Ph. D. Thesis, Computing Laboratory, The University of Newcastle Upon Tyne, Newcastle Upon Tyne, England, August 1976

REFERENCES BIBLIOGRAPHIQUES : § II.5 (suite)

[COURT 71] P. J. COURTOIS, F. HEYMANS & D. L. PARNAS
Concurrent control with readers and writers
Com. A.C.M., (14,10), October 1971, pp 667-668.

[DIJK 65] E. W. DIJKSTRA
Solution of a problem in concurrent programming
Com. A.C.M., (8,9), September 1965, p 569.

[DIJK 67] E. W. DIJKSTRA
Cooperating Sequential Processes
Dans "PROGRAMMING LANGUAGES", F. GENUYS Ed.,
Academic Press, 1967.

[FLON 76] L. FLON & A. N. HABERMANN
Toward the construction of verifiable software systems
Proceedings of A.C.M. Conference on Data, SIGPLAN Notices,
March 1976, pp 141-148.

[HABER 72] A. N. HABERMANN
Synchronisation of communicating processes
Com. A.C.M., (15,3), March 1972, pp117-184.

[HABER 75] A. N. HABERMANN
Path Expressions
Technical Report, Department of Computer Science, Carnegie-
Mellon University, Pittsburgh, June 1975.

[HOARE 74] C. A. R. HOARE
Monitors: An operating system structuring concept
Com. A.C.M., (17,10), October 1974, pp 549-557.

REFERENCES BIBLIOGRAPHIQUES : § II.5 (suite)

[LAUER 77] P. E. LAUER & M. W. SHIELDS
Abstract specification of ressource accessing disciplines: Adequacy, Starvation, Priority and Interrupts
Comptes rendues des journées d'étude sur les méthodes de synchronisation et de programmation globales en temps réel, A.F.C.E.T., Paris, Novembre 1977.

[MOSS 77] J. MOSSIERE
Méthode de conception des systèmes d'exploitation
Thèse de Doctorat d'Etat, USMG-INP Grenoble, Septembre 1977.

[OWIC 76] S. OWICKI & D. GRIES
An axiomatic proof technique for parallel programs
Acta Informatica V 16, pp 319-340, 1976.

[ROUC 78] G. ROUCAIROL
Mots de synchronisation
R.A.I.R.O. Serie bleue "Computer Science", A.F.C.E.T.-DUNOD Ed., V 12, N°4, Decembre 1978.

[ROUC 79] G. ROUCAIROL
Vers une caracterisation de la synchronisation de processus parallèles
Proceedings of the 1st European conference on parallel & distributed processing, Toulouse France, 14-16 Février 1979.

[ROVER1 77] P. ROBERT & J. P. VERJUS
Toward autonomous descriptions of synchronization modules
Rapport de recherche IMAG N°47, Octobre 1976,
Proceedings of the 1977 I.F.I.P. Congress, Toronto, North-Holland Pub. Comp., pp 981-986.

REFERENCES BIBLIOGRAPHIQUES : § II.5 (fin)

[ROVER2 77] P. ROBERT & J. P. VERJUS
Expression autonome de la synchronisation des processus concurrents
Comptes rendus des journées d'étude sur les méthodes de synchronisation et de programmation globales en temps réel, A.F.C.E.T., Paris, Novembre 1977.

[SILB 77] A. SILBERSCHATZ, R. B. KIEBURTZ & A. J. BERNSTEIN
Extending Concurrent PASCAL to allow dynamic ressource management
I.E.E.E. Transactions on Software Engineering ,V-SE-3,N°3, May 1977, pp 210-217.

[SVL 75] M. SINTZOFF & A VAN LANSVEERDE
Constructing correct and efficient concurrent programs
Proceedings of the International Conference on Reliable Software, SIGPLAN Notices(10,6),1975, pp 319-326.

[VEIL 75] G. VEILLON
Cours d'algorithmique non numerique
2iéme Année E.N.S.I.M.A.G. , Grenoble 1975.

[VERJU 78] J. P. VERJUS
Ordonnancement de processus par cheminement dans des files
Notes de travail I.R.I.S.A., Rennes,Février 1978.

PARTIE B

CONCEPTION D'ARCHITECTURES SPECIALISEES TOLERANT LES PANNES

PARTIE B : CONCEPTION D'ARCHITECTURES SPECIALISEES TOLERANT LES PANNES

INTRODUCTION

L'apparition récente de composants hautement intégrés (LSI) et la baisse des prix en résultant, ont conduit à l'utilisation de systèmes (micro) informatiques dans des domaines qui, jusque là, leur étaient interdits. Cette utilisation a multiplié le nombre des études et des réalisations de systèmes "à la demande". Dans de nombreux cas les solutions produites concurrencent directement, essentiellement pour des raisons de coût, l'emploi de calculateurs standards (mini par exemple) spécialisés à postériori par le logiciel où par l'équipement périphérique. L'élaboration de méthodes pour la conception de systèmes (micro) informatiques spécialisés, satisfaisant certaines contraintes de sûreté de fonctionnement, aurait donc deux conséquences :

- ouvrir aux concepteurs de systèmes (micro) informatiques "à la demande" le marché des systèmes à haute sûreté de fonctionnement. Ces systèmes couvriraient des applications jusque là peu automatisées ou réservées à des chaînes de traitements analogiques;
- concurrencer les systèmes universels (non spécialisés) à haute sûreté de fonctionnement.

I - ASPECTS METHODOLOGIQUES

I - 1. Nécessité d'une méthode

Malgré l'existence de nombreuses techniques de tolérance aux pannes, la conception de systèmes sûrs de fonctionnement reste un problème délicat. Cette difficulté ne fait que refléter l'absence de méthodologies globales de conception à partir d'un cahier des charges comportant à la fois des spécifications fonctionnelles et des contraintes de sûreté de fonctionnement. Outre de nombreuses études théoriques([BOUR 71],[COURT 76],[LAP 75] les seuls travaux susceptibles de donner les premiers éléments d'une telle méthodologie, sont issus de quelques projets de systèmes tolérant les pannes ayant dépassé le stade de la simple définition ([HOPK 78], [MERAU 79],[WENS 78]).

L'adjonction de critères économiques tels que la minimisation des coûts de fabrication ou d'exploitation concourt à rendre plus ardu ce problème de conception.

Si quelques principes importants peuvent d'ores et déjà être dégagés de diverses expériences antérieures, celles-ci semblent être encore trop partielles pour en déduire une approche globale.

Dans ce chapitre, nous présentons les grandes lignes d'une méthodologie de résolution du problème d'optimisation précédemment évoqué.

I - 2. Position du problème

Rappels et définitions

Nous nous intéressons tout spécialement à des systèmes non réparables pour lesquels de hautes fiabilités de missions sont exigées (systèmes embarqués, applications spatiales...).

Ces systèmes doivent généralement assurer un ensemble de missions distinctes pendant une durée T. Une contrainte de fiabilité au niveau du système global consiste en la donnée d'une quantité R(T) pour chacune des missions du système.

La contrainte en résultant peut être formulée comme suit :

La probabilité de bonne exécution d'une mission pendant la durée T doit être supérieure ou égale à la quantité R associée à cette mission.

Mais le cahier des charges de ces systèmes peut également comprendre des spécifications de sécurité.
La spécification de contraintes de sécurité consiste en la donnée d'une quantité S(T) pour chacune des missions du système.
La contrainte en résultant peut être exprimée comme suit :

La probabilité de bonne exécution d'une mission ou d'apparition d'un signal d'alarme doit être supérieure ou égale à la quantité S associée à cette mission.

Problème
Nous étudions le problème de la conception de systèmes non-réparables accomplissant une ou plusieurs missions pour lesquelles seule une même contrainte de fiabilité est exigée. Ces systèmes devront donc être capables de remplir cette mission aussi longtemps que possible, au risque de fonctionnements erronés non signalés.

I - 3. Approche proposée

Hiérarchisation des contraintes

Une approche globale, faisant intervenir toutes les contraintes, dès les premières étapes de la conception, paraît aujourd'hui impraticable. Nous proposons donc une hiérarchisation de ces contraintes en se fixant des objectifs intermédiaires.
Cette hiérarchie conduira à une approche en deux étapes :

Première étape : Conception d'une architecture "minimale" capable de remplir les spécifications fonctionnelles de l'application projetée.

Deuxième étape : Construction, à partir de l'architecture précédente, d'une structure éventuellement redondante, satisfaisant à la fois les spécifications fonctionnelles et les contraintes de sûreté du cahier des charges.

L'objectif de cette approche est double :

* Simplification grâce à une sérialisation des problèmes.
* Aboutir toujours à une solution qui, sans être optimale au sens des coûts sera tout de même acceptable au vu du cahier des charges.

Première étape

Nous n'expliciterons pas cette étape. Nous ne considérons pas, par là, cette première étape comme triviale, nous pensons simplement que les problèmes qu'elle pose sont, sans être résolus dans leur généralité, mieux connus que ceux rencontrés lors de la deuxième étape.
Signalons cependant, que l'architecture de base définie dans cette première étape devra être construite, sinon au moindre coût, du moins en n'utilisant que des composants nécessaires.

Dès cette première étape pourront intervenir des critères architecturaux préparant la deuxième étape :

- utilisation de circuits très intégrés (diminution du nombre de connexions, augmentation de la fiabilité intrinsèque);
- architecture "souple" : constituants interchangeables, facilement extensible, etc;
- minimisation du matériel utilisé.

Deuxième étape

Dans ce paragraphe, nous détaillons les différentes phases de cette étape. Tout au long de cette présentation nous désignons respectivement par T et R_{MIN} la durée de mission et la fiabilité minimale exigée pour le système .

Première phase

Cette première phase comprend l'évaluation des taux de panne de chacun des constituants de la structure de base. Elle se complète par le calcul, à partir de ces taux de panne , de la fiabilité R(T) de la structure de base.
Tous les composants de cette structure étant indispensables à l'accomplissement de la mission (cette architecture est "minimale"), la fiabilité R(T) n'est autre que le produit des fiabilités de chacun des composants, calculées pour une durée T.

Si R(T) se révèle supérieur à R_{MIN}, cette structure répond aux spécifications et constitue donc, sans modification aucune, une solution acceptable Dans le cas contraire on aura recours aux phases suivantes :

Deuxième phase

Dans cette deuxième phase on fixe un partitionnement du système ainsi qu'une stratégie de redondance pour chacune des partitions.
Ces deux choix ne sont évidemment pas indépendants l'un de l'autre. Le choix d'un partitionnement doit tenir compte des possibilités d'implantation de stratégies de redondances offertes par chaque type de composant de la structure :

- possibilités d'interconnexions nombreuses;
- unité munie de test où d'autotest à couverture connue. Souvent ces informations, lorsqu'elles seront connues, se révèleront d'emploi difficile de par leur imprécision; imprécision encore accentuée dans le cas de composants d'apparition récente et donc non encore expérimentés;
- possibilités de redondances partielles (codes détecteurs où correcteurs);
- caractéristiques fonctionnelles et de sûreté des organes pilotant ces stratégies (voteurs, comparateurs, commutateurs, calculateurs..)

Mais ce choix du partitionnement peut faire intervenir des critères tels que :

- "l'équilibrage" des taux de panne respectifs de chaque partition pour obtenir une certaine cohérence entre degrés de redondance de chaque partition. On rappelle que, pour un coût constant en composants et pour des stratégies de redondance identiques dans chaque partition, la fiabilité maximale correspond à des taux de pannes de partition égaux entre eux;
- le coût en essayant, dans le cas de grandes différences de prix entre composants, de limiter l'emploi des plus onéreux d'entre eux en les plaçant seuls dans une partition.

Le concepteur pourra également profiter de certaines caractéristiques fonctionnelles des constituants. Par exemple, il peut arriver que le dispositif (physique ou humain) utilisateur des signaux émis par ce constituant puisse s'accommoder de brèves périodes de défaillance (pannes transitoires de durée inférieure à une limite). On pourra alors adopter pour ce constituant des mécanismes de tolérance aux pannes moins prompts et également moins onéreux.

Ces techniques de redondances sont généralement rangées en deux classes :

- redondance partielle (code correcteur);
- redondance massive ou sélective:
  - . masquage : triplication, vote k parmi n
  - . détection/remplacement : remplacement d'unités, de paires d'unités.

Troisième phase

Cette troisième phase est consacrée au choix du degré de redondance pour chacune des partitions précédemment définies. Chaque partition est indispensable à la mission; elles sont donc placées en série du point de vue de la fiabilité.

Si $R_i(n_i)$ représente la fiabilité de la partition numéro i lorsque celle-ci est répliquée n_i fois, la fiabilité du système est donnée par la formule :

$$R = \prod_{i=1}^{n} R_i(n_i) \quad : \text{ n partitions numérotées de 1 à n.}$$

Le problème à résoudre peut alors s'exprimer sous la forme d'un problème d'optimisation combinatoire :

- Trouver un jeu d'entiers n_i (i=1..n) vérifiant la contrainte :

$$\prod_{i=1}^{n} R_i(n_i) \geq R_{MIN} \qquad (1)$$

pour un coût minimum.

Ce coût peut être exprimé par la fonction objective C :

$$C = \sum_{i=1}^{n} C_i(n_i) \qquad (2)$$

où $C_i(n_i)$ représente le coût de la partition numéro i lorsque celle-ci est répliquée n_i fois.
Dans le cas où l'on néglige le coût du matériel auxiliaire (voteurs, comparateurs etc) le coût C est donné par la formule :

$$C = \sum_{i=1}^{n} n_i \; c_i \qquad (2')$$

dans laquelle c_i représente le coût d'un exemplaire de la partition i.

Ce problème est difficile à résoudre pour des fonctions $C_i(n_i)$ et $R_i(n_i)$ arbitraires.
On trouve toutefois dans [KAUF 77] une solution à ce problème lorsque $R_i(n_i) = 1-(1-r_i)^{n_i}$ et $C_i(n_i) = n_i \; c_i$ (2'); r_i et c_i désignent respectivement la fiabilité et le coût d'un replicat de la partition i.

Cette formule correspond à la fiabilité résultant d'une stratégie de reconfiguration portant sur un "pool" de n_i unités identiques et capables d'assurer un service identique à celui d'une unité tant qu'une des unités du "pool" initial se trouve encore en état de marche. Cette stratégie peut être, par exemple, réalisée pour des unités possédant un test à couverture totale, si l'on suppose les organes de pilotage de cette stratégie sans défaillance.

L'examen des hypothèses précédentes montre que le problème n'est résolu que dans un cas idéal. Ce résultat est donc en pratique difficilement exploitable.

Remarque : La contrainte (1) induit une condition nécessaire sur les n_i :

$$\forall n_i \; i = \{1...n\} \quad R_i(n_i) \geq R_{MIN} \qquad (3)$$

Il est facile de calculer les n_i minimaux (notés $\hat{n}_i$) vérifiant (3). Les n_i étant entiers et les variations de $R_i(n_i)$ étant fortes (spécialement pour les faibles valeurs de n_i), les $\hat{n}_i$ vérifieront souvent aussi la condition (1) :

$$\prod_{i=1}^{n} R_i(\hat{n}_i) \geq R_{MIN}$$

dans ce cas favorable et fréquent, le problème d'optimisation sera aisément résolu.

Quatrième phase :

Une fois les n_i obtenus, on passe à la recherche d'une architecture permettant d'approcher le schéma de fiabilité série (i.e celui donnant une fiabilité R égale à $\prod_{i=1}^{n} R_i(n_i)$). Les caractéristiques propres des constituants peuvent rendre difficile, voire impossible, la conception d'une telle architecture. Dans ce cas, l'architecture retenue présentera une fiabilité R' inférieure à R. Cette fiabilité devra être évaluée (par exemple par une modélisation markovienne).

Si la valeur de fiabilité trouvée est supérieure à R_{MIN} l'architecture étudiée peut être retenue et on peut directement passer à la phase suivante.
Dans le cas contraire, on modifie la structure pour en améliorer la fiabilité :

- augmentation du degré de redondance dans certaines partitions;
- augmentation des liaisons entre éléments, pour se rapprocher d'un schéma de fiabilité série.

Exemple : passage de l'architecture exposée au §III.2.4 à celle proposée au §III.2.5.

REMARQUE

Le cas inverse peut se produire : la fiabilité trouvée peut être bien supérieure à celle spécifiée. On pourra, dans ce cas, faire des économies, non pas en réduisant le nombre d'éléments (il s'agit du nombre minimum) mais en supprimant des liaisons ou des possibilités de reconfigurations (simplification des programmes ou des organes) dirigeant la reconfiguration.

Exemple : passage de l'architecture exposée au §III.2.3 à celle du §III.2.4.

Cinquième phase :

Cette cinquième étape concerne la partie logicielle de l'application. Il s'agit de l'écriture, souvent délicate, des programmes de reconfiguration correspondants à la stratégie de reconfiguration choisie.

CONCLUSION

Nous conclurons en signalant, tout d'abord, que si la solution ainsi obtenue n'est pas optimale au sens de la minimisation des coûts, il semble néanmoins naturel de penser qu'une minimisation à chaque étape du matériel utilisé, garantira le bon compromis coût/performances de la solution finale.

Le chapitre suivant propose une méthode de redondance (basée sur une duplication massive), applicable à un réseau de calculateurs. Cette méthode pourra être intégrée à la deuxième phase de l'approche précédente, comme le montrera l'exemple réel présenté au chapitre trois.

II - RESEAUX DE MICROCALCULATEURS

II - 1. Introduction

L'étude présentée dans ce chapitre s'insère dans la méthodologie de conceptic de systèmes spécialisés esquissée au chapitre précédent. Cette étude repose sur le choix d'une architecture particulière : le réseau de calculateurs.

Par réseau de calculateurs nous désignons l'interconnexion d'un ensemble de calculateurs par des liaisons à faible débit (interface d'entrée/sortie à interface d'entrée/sortie). Ce type d'architecture est facilement réalisable avec le matériel standard actuel et peut être aisément adaptée à l'application visée.

Notre approche s'oppose donc à celle consistant à implanter l'application sur un système d'utilisation générale préexistant dont l'architecture est déjà largement fixée.
Ces architectures[MAZA 78] sont généralement assez différentes de l'architecture "réseau" proposée. Elles se présentent pour la plupart comme un ensemble "d'unités" interconnectées par l'intermédiaire d'une mémoire centrale (Exemple : C.mmp, MICRAL M, Multi-micro CII-HB, IRIS 80 CII-HB [MAZA78]) ou d'un réseau de voies de communication rapides (Exemple:"array processor"). Chaque unité peut être constituée d'un (IRIS 80 CII-HB) ou de plusieurs processeurs (Multimicro CII-HB, Cm*,[MAZA78]) et munis (MICRALM,Multimicro CII-HB[MAZA 78]) ou non (IRIS 80 CII-HB) d'une mémoire locale.
Contrairement à la solution réseau proposée, ces architectures imposent l'écriture d'un système d'exploitation important,ainsi que la réalisation de mécanismes délicats de communication (partage mémoire, cache, allocateur de bus...).

La faiblesse des vitesses d'échange d'informations entre unités vient contrebalancer ces avantages mais,de par son adaptation à l'utilisation de circuits hautement integrés standards, une telle solution mérite d'être considérée. Il est intéressant de remarquer qu'elle se situe parfaitement dans la tendance actuelle de la conception des circuits intégrés : la fabrication d'unités fonctionnelles de plus en plus importantes. Cette tendance est illustrée par l'apparition des calculateurs monolithiques qui paraissent particulièrement adaptés à l'approche réseaux (faible coût, compacité, interconnexion facile, faible consommation, bonne fiabilité).

Compte tenu de leur faible puissance et de leur manque d'aptitude aux techniques classiques de partage mémoire, l'architecture réseau reste la seule possibilité pour atteindre une puissance conséquente avec ces circuits.

Les cahiers des charges contiennent de plus en plus souvent, outre des spécifications fonctionnelles, un jeu de contraintes opérationnelles touchant à la sûreté de fonctionnement des applications projetées.

Cette étude suppose défini un réseau de calculateurs permettant la satisfaction des contraintes fonctionnelles,et ne s'intéresse qu'à l'obtention d'un réseau déduit du précédent, capable des mêmes performances, mais doué d'une meilleure fiabilité de mission.

Nous considérons comme donné ce réseau original. L'obtention de ce réseau demande une analyse approfondie de l'application conduisant à une décomposition fonctionnelle en processus présentant un faible taux d'échange mutuel d'informations et susceptibles, chacun, d'être implanté sur un seul calculateur.

Les réseaux ainsi conçus sont directement calqués sur une décomposition fonctionnelle de l'application projetée. Contrairement aux architectures multiprocesseurs classiques, ce sont des structures hautement spécialisées. Ce problème de décomposition ne sera pas abordé ici (voir, par exemple, [GRAHA 78],[SAUC 78]).

Lorsque l'on recherche une meilleure fiabilité (au sens durée de mission, comme par exemple dans les systèmes non réparables : calculateurs embarqués, applications spatiales...) deux approches complémentaires peuvent être prises [AVIZ 77] :

- Evitement des pannes (emploi de circuits plus fiables...).
- Tolérance aux pannes (utilisation de techniques de redondance).

L'utilisation systématique de circuits à haut degré d'intégration relève de la première approche. Mais les limites technologiques à la fiabilité des circuits ainsi que le prix élevé de tout accroissement de cette fiabilit rendent nécessaire le recours aux techniques de redondance. Ces dernières imposent la conception de structures facilement reconfigurables : i.e offrant de nombreuses possibilités d'affectation des processus aux processeurs et des voies de communication multiples entre composants. Tous ces

impératifs entrent largement en contradiction avec l'architecture réseaux qui présente des liaisons spécifiques entre calculateurs. Dans cette étude nous proposons une solution permettant de dépasser ces contradictions et d'offrir une fiabilité intéressante tout en conservant la structure réseaux

II - 2. Le problème

Type d'application

Comme indiqué dans l'introduction, l'application projetée est supposée décomposée en un ensemble de processus à faible taux d'échange mutuel d'information.
Nous ferons l'hypothèse (forte) suivante :

> *Ces processus sont cycliques et la bonne exécution d'un cycle ne dépend pas de celle des cycles précédents.*

Cette hypothèse permet d'éliminer les problèmes de constitution de points de reprise et de "retour arrière" après reconfiguration matérielle[MERAU 76]
Nous verrons au § II.5 que l'on pourra affaiblir quelque peu cette hypothèse.

Réseaux de calculateurs adjacents

Nous supposons donc donné un réseau de calculateurs capable d'assurer les spécifications fonctionnelles de la mission proposée. En raison de la proximité géographique des calculateurs ceux-ci peuvent être directement liés par des liaisons d'interfaces à interfaces [BELIM 77] .
Un tel réseau est représenté sur la figure II.1. Dans cet exemple certains calculateurs sont munis d'une interface d'entrée/sortie; ceci s'explique par le nombre restreint de ces interfaces possédés par un microcalculateur monolithique. Suivant les besoins d'interconnexion d'un tel calculateur avec le reste du réseau, on peut être conduit à adjoindre quelques interfaces supplémentaires au calculateur afin d'augmenter ces possibilités.

Nous symboliserons un tel réseau (figure I.1) par un graphe fonctionnel F (figure II.2), dont les sommets seront appelés indifféremment noeuds ou calculateurs.

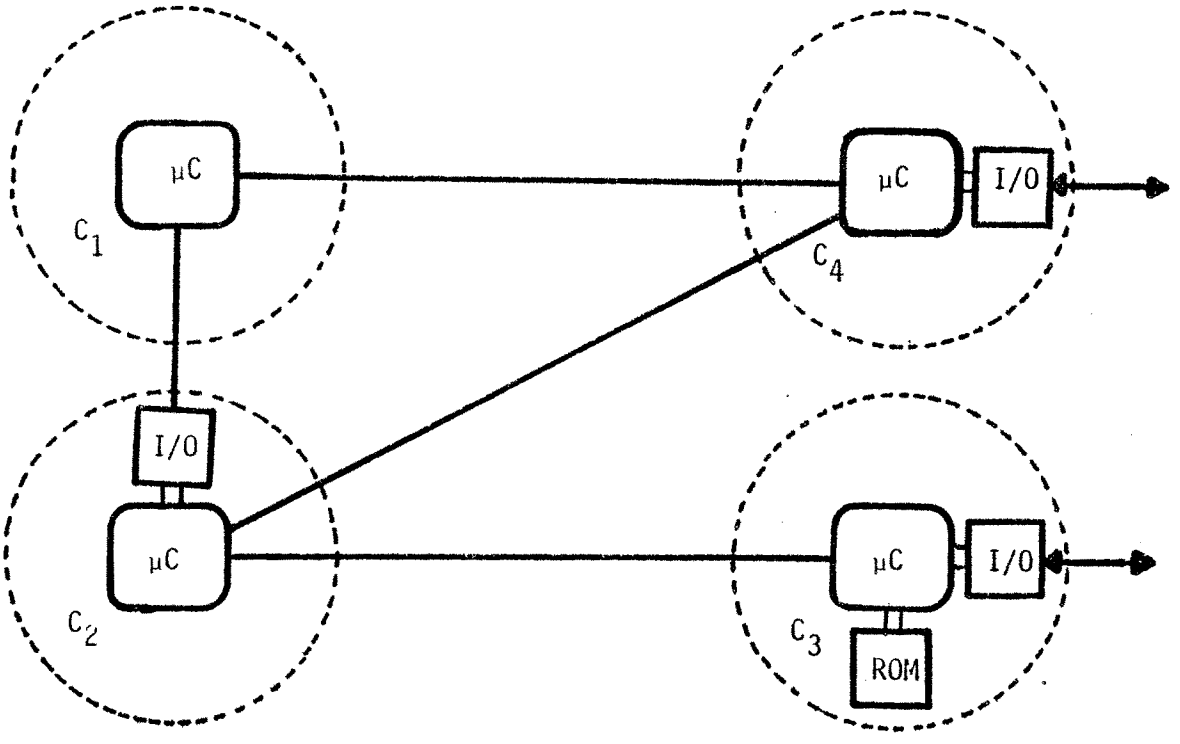


Figure II.1

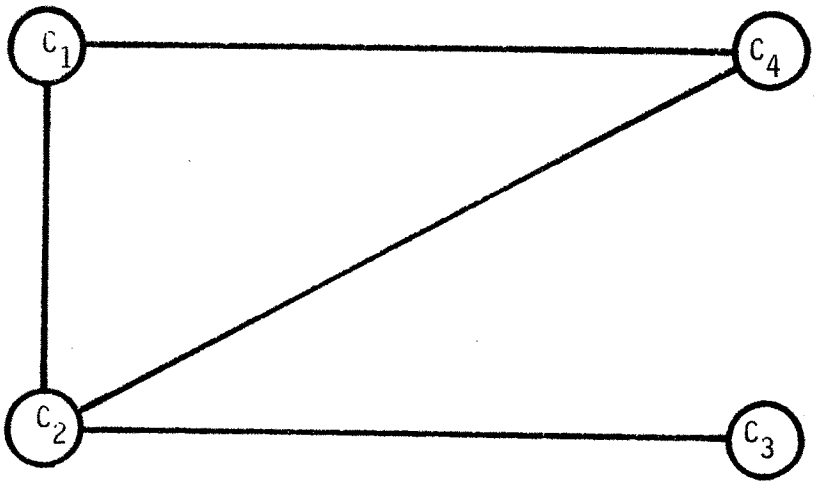


Figure II.2

Nous supposons également que cette architecture est, en un certain sens "minimale" : tous les calculateurs sont nécessaires : la suppression de l'un d'entre eux ne peut être compensée par un surcroît de travail des calculateurs restants.

Contraintes de fiabilité

Conformément au paragraphe précédent nous nous placerons dans le cadre d'un système non réparable, ne comportant aucune spécification de sécurité particulière et accomplissant des missions démunies de tout mode dégradé admissible. Le problème se résume donc à assurer ces missions durant un temps aussi long que possible.
Pratiquement une telle contrainte peut être exprimée par une fiabilité limite sur un temps de mission donné, pour l'ensemble des missions du réseau.

II - 3. Solution retenue

Les hautes fiabilités sont généralement obtenues par la succession de trois opérations : détection d'une panne, localisation de cette panne et reconfiguration. Après la détection et la localisation d'une panne le ou les composants suspects sont écartés. Trois éventualités sont alors possibles :

- "dégradation fonctionnelle" (ou dégradation progressive) les composants ne sont pas remplacés ou sont remplacés par des composants assurant des fonctions qui sont alors abandonnées (cannibalisation). Cette technique demande des missions multiples de fiabilités différentes, elle est donc inapplicable dans notre cas;
- "consommation" de composants oisifs placés en réserve ;
- "dégradation de sécurité" par utilisation de composants remplissant antérieurement des fonctions de test.

II - 3.1. Détection

Parmi toutes les méthodes de détection qui s'offraient à nous, nous avons choisi de doubler chaque calculateur par un calculateur identique. Les deux organes peuvent ainsi se contrôler mutuellement grâce à des exécutions simultanées de la même tâche et la comparaison des résultats. Plusieur raisons nous ont imposé ce choix :

1) Le taux de détection des pannes est un facteur déterminant dans l'obtention de bonnes performances de fiabilité [BOUR 71] . Nous recherchons donc une technique nous donnant un taux de détection (couverture) le plus proche possible de l'unité.

2) Les circuits L.S.I. autotestés, ou plutôt sûrs de fonctionnement restent encore à l'état de projets. On peut également se demander si le taux de détection résultant ne serait pas insuffisant pour des objectifs de haute fiabilité.

3) Les tests en ligne par vérification de vraisemblance ou par emploi de redondances logicielles sont incapables d'atteindre les taux recherchés. Pour ces méthodes, le prix du logiciel supplémentaire devrait également être considéré.

4) Enfin nous écartons le test périodique à cause de sa "tansparence" aux pannes fugitives qui forment pourtant une forte proportion des défaillances enregistrées dans les circuits intégrés [SIEW 78] .

Définitions : Chaque calculateur Ci, exécutant la tâche Ti dans le réseau initial est remplacé par une paire Pi de calculateurs : Pi = (Ci, $C_i^!$). Le calculateur Ci (resp $C_i^!$) est appelé calculateur actif (resp. calculateur passif) de la paire Pi.

Ci et C'i exécutent donc tous les deux la tâche Ti et comparent leurs résultats : ils travaillent en mode duplex.

Afin d'augmenter la souplesse de la structure ainsi dupliquée nous avons préféré une comparaison par voie logicielle à celle obtenue grâce à un comparateur matériel. Une telle méthode requiert l'existence d'une simple liaison "interface à interface" entre les deux calculateurs de la paire. Elle ne demande pas une stricte synchronisation des deux calculateurs et, est donc tout à fait cohérente avec le type de communication choisi pour le réseau.

On sait que l'efficacité d'une méthode de détection par duplex repose fortement sur l'indépendance des processus d'arrivée des pannes dans les deux unités formant le duplex.

Cette indépendance est favorisée dans tous les cas par l'emploi d'alimentations séparées pour chaque calculateur.
Dans le cas particulier d'une comparaison logicielle cette indépendance peut être accrue par l'utilisation d'horloges distinctes. On évite ainsi le difficile problème de resynchronisation dont la solution entraîne une augmentation du matériel partagé entre les deux unités (génération des signaux d'horloge communs [DAVI 78]).

On note également que la comparaison logicielle permet :

- d'une part l'utilisation d'une communication interface à interface qui assure l'isolation mutuelle des deux unités et donc l'indépendance entre les processus de défaillance,
- d'autre part, de placer dans chaque calculateur d'une même paire deux logiciels différents et donc de détecter les défaillances éventuelles de ce logiciel.

Mais le choix d'une comparaison logicielle pose une double question importante :

- quelles performances attendre d'un tel dispositif ?
- quelle est l'influence de la fréquence de comparaison sur ces performances ?

On peut remarquer que deux effets vont ici s'opposer :

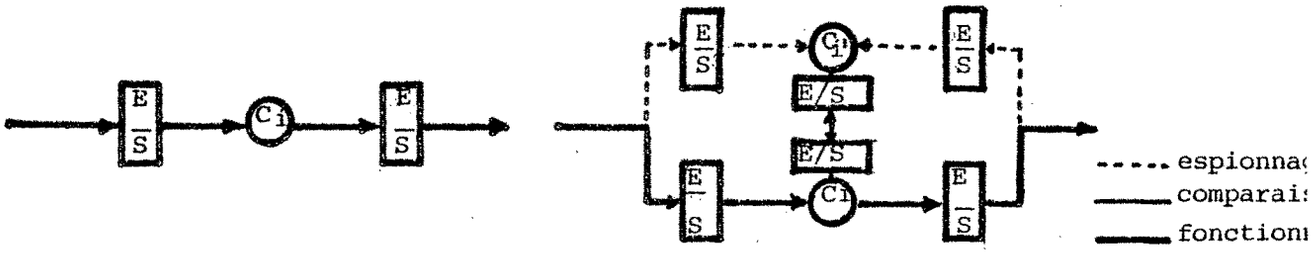
- des comparaisons trop fréquentes contribueront à l'augmentation globale de la durée d'une mission et donc, provoqueront une baisse de la fiabilité de cette mission;
- des comparaisons trop rares augmenteront le risque de pannes simultanées des deux unités entre deux comparaisons et diminueront parallèlement la couverture de détection.

L'étude présentée en Annexe I montrera l'efficience de la solution logicielle par rapport aux méthodes matérielles, ainsi que la vaste plage de fréquences de comparaisons susceptibles de produire ces bonnes performances. Cette étude s'intéressera tant à des stratégies utilisant une détection par duplication qu'à des stratégies basées sur une triplication des unités fonctionnelles.

On notera toutefois que les méthodes matérielles induisent parfois des baisses de performances appréciables provoquées par la resynchronisation entre unités :
Exemple : 12 à 16% de baisse de performances par rapport au "simplex" pour calculateur à vote majoritaire C.vmp [SIEW 78] .

Définition : La liaison entre calculateurs actif et passif d'une même paire est appelée liaison de comparaison.

En raison des forts taux de détection recherchés, il serait intéressant de disposer d'une architecture de paire capable de détecter toutes ses propres défaillances. Les défaillances des calculateurs proprement dits sont "couvertes" par la méthode de comparaison des résultats avancée plus haut; reste donc le problème des pannes dans les interfaces d'entrée/sortie. Ces pannes pourront être détectées si l'on ajoute dans la paire des liaisons "d'espionnage" (en lecture seule) permettant le contrôle par le calculateur passif des résultats et des signaux arrivant ou sortant de la paire (Figure II.3). Un tel schéma permet la détection par (au moins) un des deux calculateurs de la paire de toutes pannes (uniques) arrivant au sein de cette paire. Si, lors d'une détection de panne, chaque calculateur émet un signal de défaillance, l'union de ces deux signaux indiquera sûrement toutes pannes de la paire (à la fiabilité près du matériel, très restreint, utilisé spécifiquement pour l'émission de ces signaux) (Figure II.5).



Reseau initial Reseau dupliqué

Figure II.3

L'hypothèse de proximité géographique autorise des liaisons entre interfaces formées de simples fils. En conséquence, toute panne d'une liaison ne pourra être le fait que d'une défaillance d'une des interfaces de cette liaison et sera donc détectée par la seule paire à laquelle appartient cette interface.
La non observation de ce principe "d'auto-détection" des pannes au niveau d'une paire conduirait à un difficile problème de diagnosabilité [PREPA 67].

L'implantation de cette méthode de détection conduit donc à un réseau dupliqué qui peut être symbolisé par un graphe G (voir figure II.4 graphe G associé au graphe F de la figure II.1).

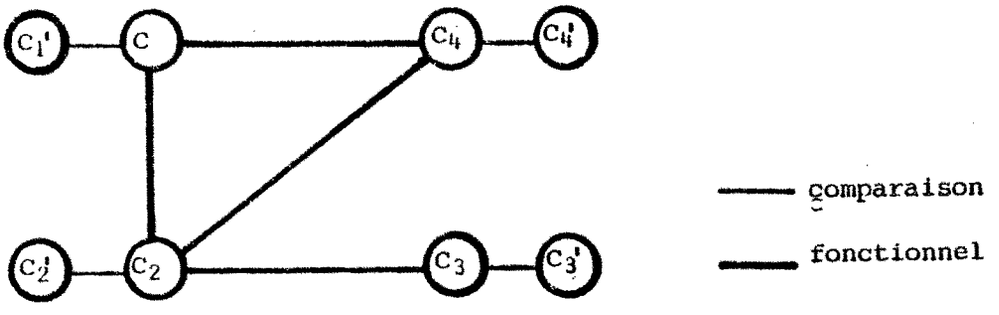


Figure II.4

II - 3.2. Politique de remplacement

Après avoir choisi une politique de détection nous nous occupons ici de problème de la reconfiguration. L'hypothèse de "minimalité" du réseau fonctionnel nous oblige à envisager des solutions par remplacement. Le réseau initial (graphe F) assurant des fonctions de mêmes fiabilités n'admettant aucun mode dégradé, deux solutions s'offrent à nous :

- *utilisation d'éléments oisifs en réserve :* deux solutions extrêmes sont alors envisageables :
  . calculateurs banalisés : chaque calculateur en réserve peut remplacer n'importe quel calculateur du réseau. Cette méthode, compte tenu des caractéristiques de l'architecture réseau, demande d'une part une grande mémoire pour contenir le logiciel de toutes les tâches du réseau ,et d'autre part un grand nombre de liaisons;

. calculateurs spécifiques : cette solution consiste à ajouter des calculateurs de réserves à chaque noeud du graphe G. Ces calculateurs ne possèderont que les programmes et les liaisons du noeud auquel ils sont associés. Dans le cas d'une réserve unique cela revient à une triplication au niveau de chaque noeud du graphe F . Plusieurs politiques sont alors applicables au niveau de chaque noeud [MATH 71]. Dans l'ordre des fiabilités strictement croissantes, on peut citer :

* Le classique TMR
* La stratégie DUPLEX-RESERVE : lors de l'occurrence d'une défaillance dans la paire initiale, la tâche abandonnée par cette dernière est reprise par le calculateur en réserve : meilleure fiabilité au prix d'une perte en sécurité par rapport au TMR.
* La stratégie TMR-SIMPLEX : cette stratégie débute comme le TMR mais l'occurrence de la première panne provoque l'éclatement du trio et la poursuite de la mission sur un des deux calculateurs encore valides après cette première panne.

Ces solutions correspondent, comme nous le verrons lors des évaluations des fiabilités (§II.4), à un degré de partitionnement égal à l'unité de l'architecture originale.

- Solution par dégradation de sécurité

La solution originale que nous proposons ici appartient à cette deuxième catégorie : une paire défaillante est remplacée par le calculateur passif d'une autre paire. Les deux noeuds mis en cause par l'occurrence d'une panne voient donc leurs tâches respectives assurées par des calculateurs travaillant en mode simplex.

Quelques méthodes de détection peuvent être appliquées aux calculateurs qui, par suite d'une reconfiguration, travaillent en mode simplex : tests périodiques, vérifications de vraisemblance (ces dernières nécessitant une connaissance précise de l'application assurée par le réseau : domaine et dynamique des grandeurs manipulées). Ces méthodes présentent une couverture qui, tout en étant à priori difficilement estimable, peut être supposée inférieure à celle obtenue par duplication. Nous avons choisi de ne prévoir aucune reconfiguration dans le cas de l'occurrence d'une panne dans une unité travaillant en mode simplex.

La mission sera donc :

- soit poursuivie, si aucun mécanisme de détection n'a joué, et fourni par conséquent des résultats erronés;
- soit suspendue, un signal d'arrêt apparaissant à l'extérieur.

Néanmoins, ces méthodes de détection à couverture partielle restent nécessaires notamment pour éviter l'émission de signaux dangereux et augmentent ainsi la sécurité de la structure.

La structure se reconfigure en subissant une perte de sécurité. Notre soluti est, en quelque sorte, une solution intermédiaire entre des réserves spécifiques et des réserves banalisées.

Par exemple, supposons qu'une panne soit détectée dans une paire (Ci,C'i), et,qu'il existe un calculateur passif C'j capable de reprendre la tâche Ti. La stratégie consiste alors à :

1) "Tuer" la paire défaillante sans tenter une localisation plus fine de la panne (par exemple en arrêtant les horloges ou en coupant les alimentations de la paire).
   Cette mesure n'est prise que pour isoler la paire endommagée du réseau. Grâce à l'emploi de liaisons interface à interface cette mesure peut se révéler superflue.

2) Provoquer la reprise de la tâche Ti par le calculateur C'j. La paire Pj est alors disjointe, et la tâche Tj reste assurée par le seul calculateur Cj.

   Cette reprise est facilitée par l'hypothèse de tâche cyclique sans mémorisation d'un cycle sur l'autre (§II.2).

3) Signaler aux noeuds adjacents à la paire Pi son remplacement par le calculateur C'j.

Remarque : cette stratégie oblige chaque calculateur passif à pouvoir assurer au moins deux tâches (non simultanément) : celle qu'il contrôle, et celle qu'il peut remplacer.

Le schéma ci-après (figure II.5) présente un mécanisme possible pour la conduite de cette stratégie au niveau d'une paire.

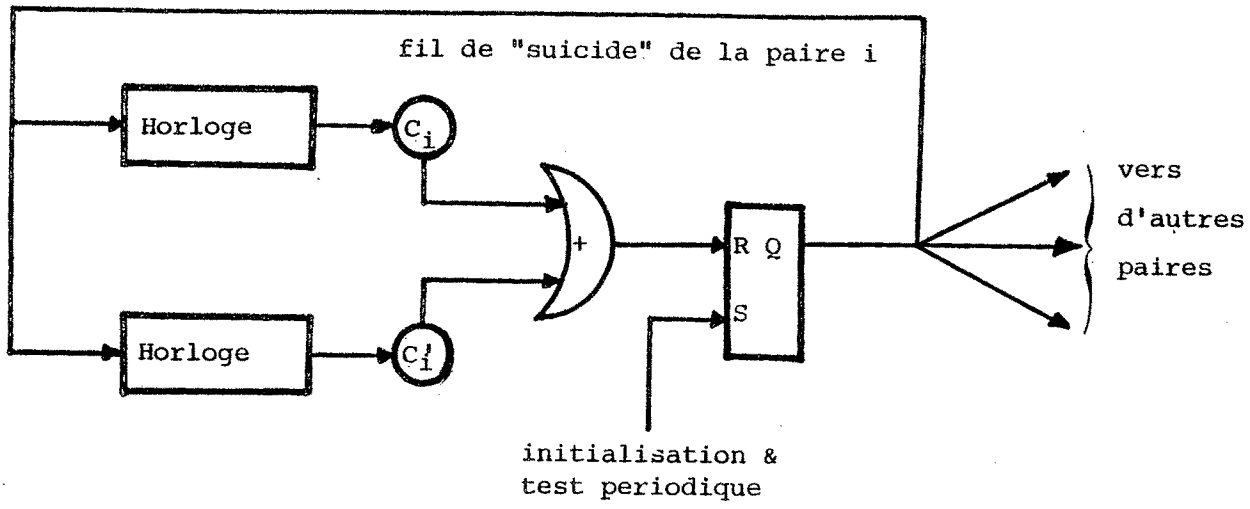


Figure II.5

Là encore, l'hypothèse de proximité géographique des noeuds facilite la conception de ce mécanisme : les signaux destinés aux autres calculateurs transiteront dans de simples fils conducteurs.

Ce mécanisme demandera l'adjonction d'un matériel auxiliaire (dispositif d'arrêt d'horloge, porte, bascule...) dont le taux de panne ne devra pas être négligé dans une évaluation fine de la fiabilité du réseau redondé. Remarquons toutefois que l'influence de ce matériel va être faible. Appelons R_c la fiabilité d'un des calculateurs et R_m la fiabilité du matériel auxiliaire. Une panne de ce dispositif n'est catastrophique que lorsqu'il y a simultanément une panne dans la paire de calculateurs; soit un événement ayant une probabilité très faible d'occurrence $((1-R_c^2)(1-R_m))$. En effet, une défaillance dans ce seul matériel auxiliaire, soit ne sera suivi d'aucun effet (pas de signal de panne), soit provoquera à tort l'émission d'un signal de panne. Le volume de ces circuits étant faible, leur fiabilité globale R_m pourra être intégrée à celle de la paire R_c sans la modifier de façon significative.

$$R_c^2 \approx R_c^2 R_m$$

Tout se passera donc au niveau de la fiabilité du réseau final (cf. II.4.2) comme si l'on utilisait des calculateurs légérement moins fiables.

II - 3.3. Stratégie de reconfiguration

Supposons qu'à un certain instant ce réseau assure sa mission. Consécutivement aux occurrences antérieures de pannes, ce réseau est dans un état dégradé et se compose :

- de n calculateurs travaillant en mode simplex. Ce sont des calculateurs secondaires de paires disjointes en réponse aux pannes précédentes;
- de 2 s calculateurs fonctionnant en mode duplex. Ces calculateurs forment les s paires restantes.

La mission étant encore assurée, il vient : $n + s = N$.

La définition d'une stratégie de dégradation S revient à définir une application appelée affectation qui, pour chaque état de dégradation atteignable de la structure, associera à chaque noeud i du graphe F un noeud j distinct de i, de ce même graphe.
S est défini en convenant que la disjonction (éventuelle) de la paire j:(Cj, C'j) pallie une panne de la paire i:(Ci, C'i) dans l'état de dégradation considéré.

Définition : L'affectation S sera une bonne affectation si, pour chaque état de dégradation atteignable, S fait correspondre à toute paire non disjointe i une paire non disjointe j.

Remarques

- La stratégie construite sur une bonne affectation est "optimale" par rapport à la méthode de détection choisie dans le sens où toutes pannes détectées (i.e arrivant dans une paire non disjointe) seront "recouvertes".
- Une telle affectation ne peut exister que dans le cas d'un graphe F à nombre pair de noeuds. En effet, chaque panne détectée provoque la disparition de deux paires. Dans le cas d'un nombre impair de noeuds on risque d'aboutir à un état dégradé ne comportant qu'une paire non encore disjointe, et pour lequel il sera par conséquent impossible de construire une bonne affectation.

Définition : Une affectation S est fixe si et seulement si elle est indépendante de l'état de dégradation de la structure. (S est une application des noeuds de F sur eux-mêmes).

Proposition : Une affectation S fixe est une bonne affectation si et seulement si S est une application telle que quel que soit le sommet i du graphe F, S(S(i)) = i : S est une application involutive.

Démonstration

1) Condition suffisante : Toute affectation fixe involutive est une bonne affectation.
S est fixe par définition. Considérons un noeud i; s étant une application, on peut noter j le noeud associé à i par S. Il vient :
j = S(i) et donc (S est involutive), i = S(j);
s étant une application, si un noeud k est distinct de i et de j alors s(k) sera distinct de i et de j. (sinon s(i) où s(j) serait identique à k).
A l'instant initial les paires i et j travaillent en duplex. Si a un instant ultérieur la paire i est disjointe cela ne peut être attribué qu'à l'occurrence d'une panne dans le noeud j.
Il s'en suit que la paire i travaille en duplex si et seulement si la paire j travaille également en duplex. S est donc une bonne affectation .

2) Condition nécessaire : Toute bonne affectation fixe est involutive.
Soit S une bonne affectation fixe. Supposons que S ne soit pas involutive. Il existe (au moins) trois noeuds i, j, k tous disctincts, tels que : S(i) = j ; S(j) = k .
Plaçons-nous à l'instant initial, pour lequel aucune des paires placées aux noeuds i, j, k ne sont encore disjointes. Considérons l'occurrence d'une panne dans la paire j. Cette panne provoque, conformément à S, la disjonction de la paire k. S associe donc à la paire i, non encore disjointe, la paire j déjà disjointe : S n'est donc pas une bonne affectation.

Remarques :

- En conséquence, dans le cas d'une bonne affectation fixe, un calculateur actif ne devra être capable d'exécuter que sa propre tâche, alors qu'un calculateur passif devra être susceptible d'exécuter aussi bien la tâche de sa paire, que celle de la paire qu'il pourra remplacer éventuellement.
  Cette caractéristique est parfaitement cohérente avec l'utilisation d microcalculateurs monolithiques dont l'emploi est généralement limité par l'exigüité de la mémoire interne.

- Lorsque le nombre de noeuds du graphe F est impair, on pourra prévoir une stratégie spécifique pour un des noeuds tandis que notre solution restera applicable au réseau restant (voir § suivant).

- Dans le cas d'un graphe F comportant un nombre pair de noeuds (N=2Q), la définition d'une bonne affectation S revient à partitionner ce graphe en Q couples disjoints. Les membres de chaque couple seront mutuellement associés par l'application involutive .

II - 3.4. Construction des liaisons redondantes et minimisation de leur nombre

Considérons à nouveau un graphe F comportant un nombre pair de noeuds. Le réseau dupliqué, symbolisé par le graphe G (Fig.II.3) a été découpé en Q couples de paires (Fig.II.6), ce qui définit complètement la stratégie de reconfiguration. Mais des liaisons supplémentaires doivent être ajoutées à ce réseau dupliqué pour permettre la reprise de chaque tâche par le calculateur passif associé. Le réseau résultant de ces adjonctions sera représenté par un graphe H. La construction de ce réseau H sera illustré par l'exemple de la figure II.6. Plusieurs partitions en Q couples des N paires sont envisageables (C_N^2). Nous recherchons maintenant les "meilleures" partitions : i.e celles qui minimisent le nombre de liaisons à rajouter (la minimisation de ce critère correspond pratiquement à la minimisation du nombre des liens physiques et des circuits d'interfaces). Pour cela nous allons, dans un premier temps, évaluer le nombre des liaisons (i.e des arrêtes du graphe H) résultant d'une partition donnée.

- Supposons que le graphe F possède N(=2Q) sommets et P arrêtes, et que chaque sommet C_i de ce graphe partage n_i arrêtes avec les autres noeuds; il vient :

$$\sum_{i=1}^{n} n_i = 2P$$

La partition du graphe F en Q couples, fait apparaître deux types de couples :

. Les couples adjacents : construits à partir de deux noeuds adjacents dans le graphe F (Exemple: (C_1, C'_1) et (C_2, C'_2) sur la figure II.6). Soit K le nombre de ces couples.

. Les autres couples, ne possédant pas cette propriété. Leur nombre est par conséquent Q-K (Exemple: (C_3, C'_3) et (C_4, C'_4) sur la figure II.6).

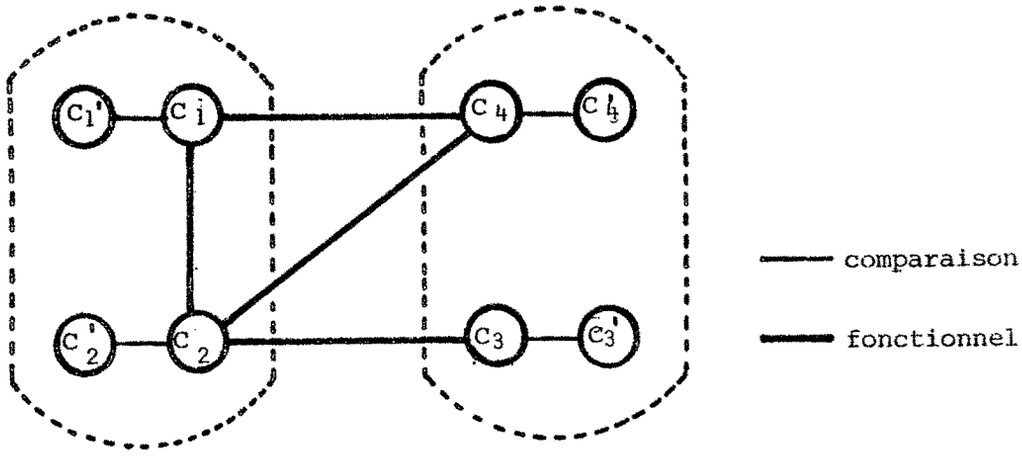


Figure II.6

Dans le graphe final H les deux conditions suivantes sont nécessaires et suffisantes pour assurer la stratégie proposée.

Condition 1 : Chaque calculateur actif C_i doit avoir les liaisons suivantes

1.1 - Une liaison avec son calculateur passif (liaison de comparaison).

1.2 - Une liaison avec chacun des calculateurs actifs qui lui sont adjacents dans le graphe fonctionnel F (liaisons fonctionnelles).

1.3 - Une liaison avec tous les calculateurs passifs susceptibles de remplacer un de ces calculateurs adjacents (liaisons redondantes)

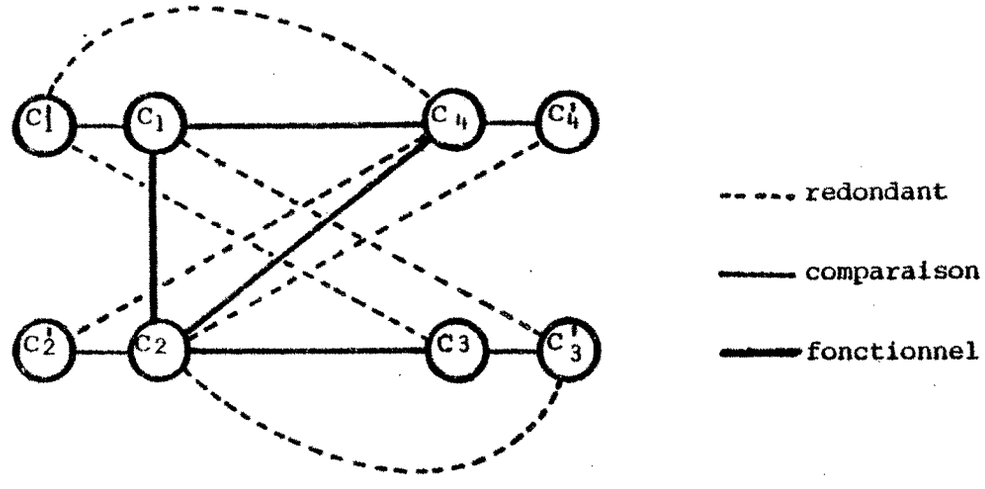


Figure II.7

On remarque qu'une liaison de comparaison peut également être une liaison redondante.

Exemple : (Figure II.7) liaison de comparaison C_1, C'_1. Cela provient de l'adjacence entre les deux calculateurs actifs d'un même couple (Exemple C_1, C_2).

Condition 2 : Cette condition porte sur les liaisons entre calculateurs passifs (liens bi-redondants).

Un calculateur passif C'_i susceptible de remplacer un calculateur actif C_j doit être connecté à tous calculateurs passifs C'_h auxquels C_j est connecté par une liaison redondante à l'exception de C'_j.

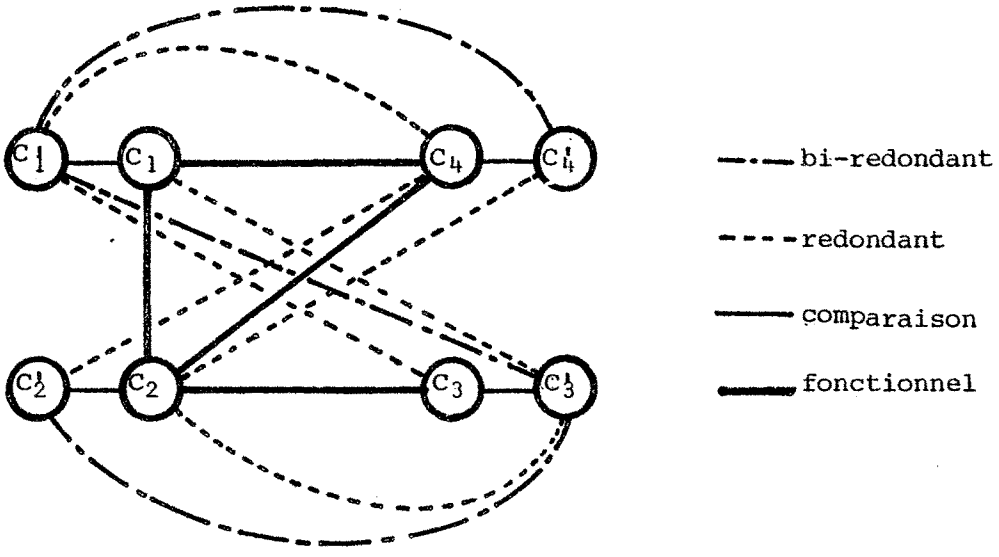


Figure II.8

On pourrait croire que le nombre des liaisons bi-redondantes est égal au nombre de liaisons redondantes qui ne sont pas aussi des liaisons de comparaison. En fait, on peut montrer que chaque liaison bi-redondante peut jouer le rôle de deux liaisons bi-redondantes distinctes.

Démonstration

Ces deux conditions sont évidemment nécessaires. Considérons maintenant un couple de paires ((C_i, C'_i) et (C_j, C'_j)) travaillant encore en mode duplex. Supposons qu'une défaillance non fatale (i.e d'une paire travaillant en duplex) surviennent à l'extérieur du couple. C_i, (resp. C_j) s'adaptera à la nouvelle situation puisqu'il possède (condition 1) des liaisons avec tous les remplaçants des calculateurs qui lui sont adjacents. (Ces liaisons ne seront utilisées que si la panne survient dans un de ces calculateurs). Si, maintenant on considère une panne au sein du couple, par exemple dans la paire (C_i, C'_i), C'_j est alors capable de reprendre la tâche T_i puisqu'il possède toutes les liaisons de C_i (à l'exception du lien de comparaison (C_i,C'_i) rendu inutile par la "mort" de (C_i,C'_i))(condition 2) .

Calculs du nombre de liaisons : considérons un couple (C_i, C'_j) et (C_j,$C'_$

Un noeud C_i possède : - 1 liaison de comparaison avec le calculateur C

- n_i liaisons fonctionnelles (1-2)

- n_i liaisons redondantes (1-3) si C_i appartient à un couple non adjacent et n_i-1 dans le cas contraire (la liaison de comparaison C'_i C_i jouant le rôle d'une liaison redondante). Le degré du sommet C_i dans le graphe H est donc égal, soit à $2n_i+1$, soit à $2n_i$.

Un noeud passif C'_i possède : - 1 liaison de comparaison avec le calculateur C_i (1-1)

- n_j ou n_j-1 liaisons redondantes (1-3)

- n_j ou n_j-1 liaisons bi-redondantes (2)

Le degré du sommet C'_i dans le graphe H est donc égal à : $2n_j$-1, si C'_i appartient à un couple adjacent, et $2n_j$+1 dans le cas contraire.

Le nombre total d'arrêtes P du graphe H est donné par la demi-somme des degrés des sommets de H, soit :

$$P' = \frac{1}{2}\left[\sum_{\substack{\text{couples}\\\text{adjacents}}} (2n_i+2n_j+2n_i-1+2n_j-1) + \sum_{\substack{\text{couples}\\\text{non adjacents}}} (2n_i+1+2n_j+1+2n_i+1+2n_j\right.$$

$$P' = \sum_{\substack{\text{tous les}\\\text{couples}}} \left[2n_i+2n_j\right] - K + 2(Q-K) \text{ soit :}$$

$$\boxed{P' = 4P + N-3K}$$

Cas d'un nombre impair de calculateurs dans le réseau original : N=2Q+1

Dans ce cas nous supposerons qu'un noeud a été laissé en dehors des reconfigurations précédentes. Plusieurs stratégies sont possibles indépendamment du reste du réseau. Appliquons-lui par exemple une stratégie DUPLEX-RESERVE (le noeud est donc dupliqué et muni d'un calculateur de réserve). Soient C_V le noeud ainsi isolé et n_V son degré dans le graphe fonctionnel F. La stratégie proposée ci-dessus appliquée au reste du réseau conduira à un graphe H possédant $4(P-n_V)+2Q-3K$ arrêtes. C_V possède n_V liaisons fonctionnelles, n_V liaisons redondantes et une liaison de comparaison avec son calculateur passif C'_V. Le calculateur en réserve sera pourvu de n_V liaisons redondantes et n_V liaisons bi-redondantes. Le nombre total des arrêtes du

réseau final sera donné par la même formule :

$$P' = 4(P-n_v)+2Q-3K+4n_v+1 = 4P+N-K$$

Dans tous les cas le minimum du nombre de liaisons du réseau redondant est atteint par toute partition des noeuds du réseau dupliqué, maximisant le nombre des couples adjacents.
Ce problème de minimisation se réduit donc à un simple problème de couplage maximum dans le graphe fonctionnel F [BERGE 70] .

Remarque : Un autre critère de minimisation aurait pu être retenu : la minimisation du nombre maximal de liaisons incidentes à un même calculateur. Ce critère pouvait être directement lié au nombre maximal d'interfaces susceptibles d'être ajoutées à un calculateur. Mais, ce critère n'est pas très intéressant car le degré d'un noeud ne peut varier que d'une ou deux unités suivant qu'il appartient ou non à un couple adjacent.

La figure II.9 montre quelques exemples d'applications de cette méthode.

II - 4. Evaluation de la fiabilité

Nous proposons ici une évaluation "grossière" de la fiabilité des structures redondantes évoquées ci-dessus.
Cette évaluation est largement simplifiée par les hypothèses suivantes :

- Elle suppose que tous les noeuds du réseau redondant bénéficient de la même fiabilité R(T).
- Elle néglige quelques événements "dangereux" ayant une très faible probabilité d'apparition. Citons, par exemple :
  - La défaillance des deux calculateurs d'une même paire dans l'intervalle séparant deux comparaisons.
  - La défaillance d'un calculateur survenant alors que le mécanisme d'"arrêt" (figure II.4) de la paire est déjà en panne, sans avoir provoqué la disparition de cette dernière.
  - ... .

On peut remarquer que, si chaque calculateur possède sa propre alimentation, la fiabilité de celle-ci peut être intégrée dans la fiabilité R(t) du noeud.

----redondant & bi-redondant

——comparaison

——fonctionnel

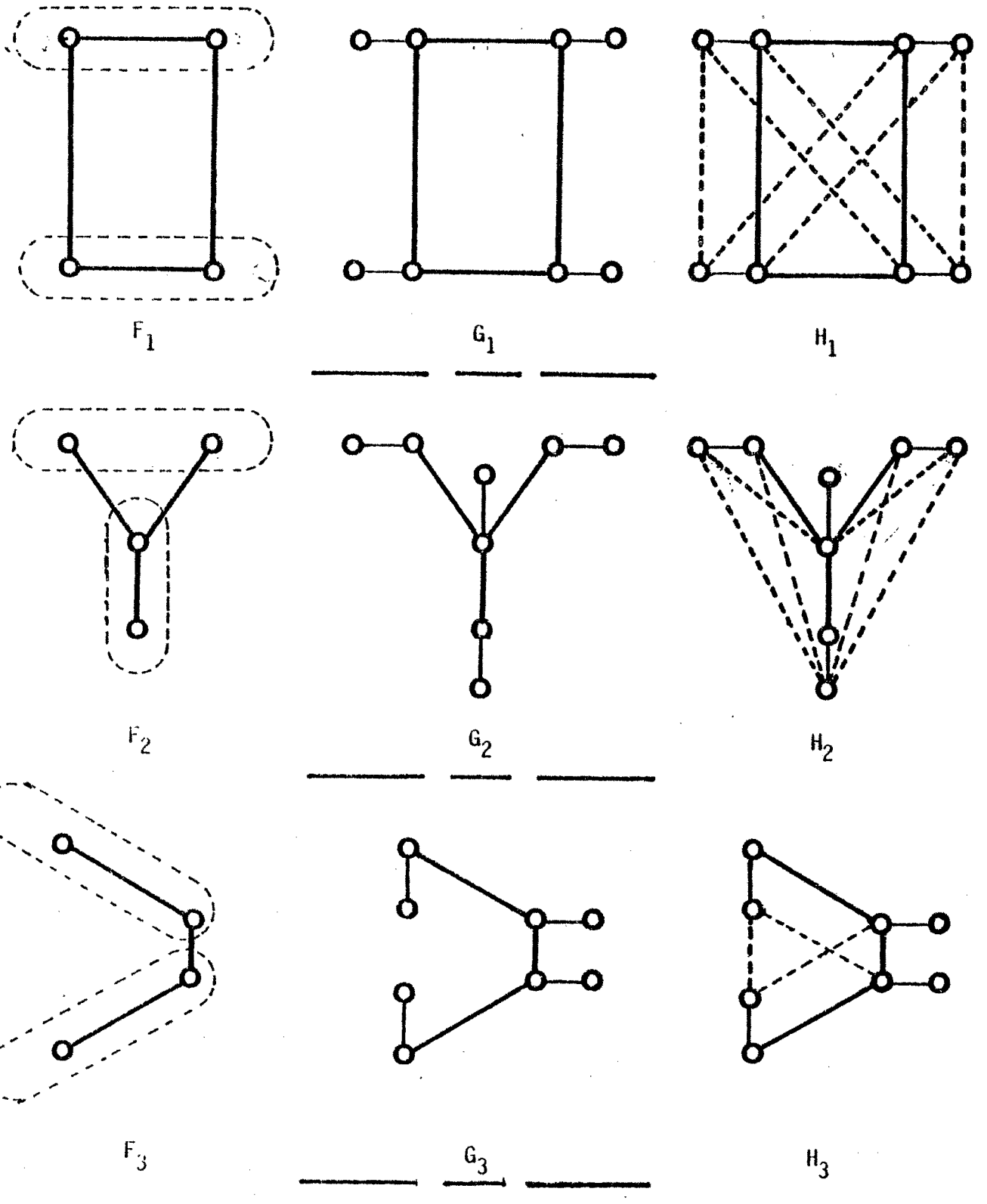


Figure II.9

Le but de cette évaluation est double :

- Donner une idée grossière des performances de notre méthode.
- Comparer ces performances avec celles des autres méthodes classiques.

II - 4.1. Evaluation

Nous nous placerons dans le cas d'un nombre pair de noeuds (N=2Q). Le réseau redondant, formé de Q couples, assure sa mission tant que chacun de ces Q couples assure les deux tâches qui lui sont confiées. Donc en appelant R_N la fiabilité recherchée et R_c la fiabilité de mission d'un couple il vient :

$$R_N = R_c^Q$$

Un couple assure sa mission lorsque, au moins, une de ces deux paires fonctionne. Si R_p désigne la fiabilité d'une paire il vient :

$$R_c = R_p^2 + 2R_p(1-R_p) = 2R_p - R_p^2$$

et par suite :

$$R_N = (2R_p - R_p^2)^Q$$

Une paire fonctionne si les deux calculateurs qui la composent sont exempts de pannes, soit :

$$R_p = R^2(t)$$

ce qui donne : $R_N = [2R^2(t) - R^4(t)]^Q$

II - 4.2. Comparaison avec des méthodes classiques

Il n'existe pas, à notre connaissance, de méthodes classiques à éléments de remplacement spécialisés comportant un degré de redondance inférieur à la triplication. Nous allons donc comparer notre méthode avec les classiques stratégies tripliquées suivantes :

- TMR,
- Duplex-Réserve,
- TMR-Simplex.

Ces solutions seront, bien entendu, appliquées à un réseau de complexité équivalente, soit formé de N noeuds fonctionnels.

D'une façon générale, ces stratégies peuvent être appliquées directement au réseau fonctionnel ou à chaque sous-ensemble obtenu après partitionnemen de ce même réseau. Pour faciliter cette évalution nous ne considérerons que des partitions régulières :
W sous-ensembles de N/W calculateurs chacun. Néanmoins, les valeurs non entières de N/W fournissent une estimation des performances obtenues par des partitions irrégulières. Nous n'envisagerons pas des degrés de partitionnement supérieurs à N parce qu'ils correspondent à des partitionnements internes aux noeuds et supposent donc une modification de l'architecture interne de ces noeuds :

$$1 \leq W \leq N$$

De façon analogue au paragraphe II.4.1 Nous considérons des estimations grossières (et optimistes) des fiabilités produites par les diverses stratégies précédemment évoquées [COURT 76] :

$$\text{TMR} : R_{TMR} = (3R^{2N/W} - 2R^{3N/W})^{W}$$

$$\text{Duplex-Réserve} : R_{D} = (R^{N/W} - R^{3N/W} + R^{2N/W})^{W}$$

$$\text{TMR-Simplex} : R_{TS} = (\frac{3}{2} R^{N/W} - \frac{1}{2} R^{3N/W})^{W}$$

Résultats :

En comparant les quantités ci-dessus avec l'estimation R_N de la fiabilité donnée par la solution proposée,on obtient les résultats suivants :

- quel que soit $R(0 \leq R \leq 1)$, quel que soit N et quel que soit $W(1 \leq W \leq N)$ notre solution est plus fiable que les solutions TMR et Duplex-Réserve.

- comparée à la solution TMR-Simplex notre solution apparaît
  - . plus fiable, quelsque soient R et N, pour $\frac{N}{W} \geq 4/3$
  - . moins fiable quels que soient R et N, pour $(1 \leq \frac{N}{W} \leq \frac{2\log 3/2}{\log 2}$ $(\simeq 1.17)$.
  - . pour les valeurs de N/W comprises entre 4/3 et $\frac{2\log 3/2}{\log 2}$ notre solution ne jouit d'une meilleure fiabilité que pour les forts temps de mission (c'est-à-dire lorsque la fiabilité de chaque calculateur devient faible).

La démonstration de ces résultats est détaillée dans l'ANNEXE 2.

Interprétation et discussion

Au vu des résultats précédents et compte tenu des diverses simplifications effectuées, notre stratégie, bien que dupliquée, n'est inférieure qu'à la meilleure (au sens fiabilité) des stratégies tripliquées (TMR-Simplex) et cela seulement si cette stratégie est appliquée au niveau de chaque noeud.

Notre méthode permet donc d'atteindre des fiabilités comparables, voire supérieures, à celles des méthodes classiques tout en permettant une économie significative sur le matériel.
Ce résultat semble confirmé par l'exemple réel traité au chapitre III.
En contrepartie on notera que certains de nos calculateurs sont plus complexes que ceux utilisés dans une stratégie tripliquée : les calculateurs passifs doivent pouvoir exécuter deux tâches contre une seule pour les calculateurs dans les autres stratégies.

II - 5. Extensions

Applications à contextes

La solution proposée peut être adaptée à des applications pour lesquelles des sauvegardes d'information (contexte) sont nécessaires à toute reprise. Dans ce cas, il conviendra de faire périodiquement transiter ces contextes des calculateurs actifs, vers les calculateurs passifs susceptibles de les remplacer.
Ces transferts à l'intérieur d'un couple pourront se faire :

- soit par le lien inter-paire s'il s'agit d'un couple adjacent,
- soit par une transmission à travers d'autres noeuds dans le cas contraire.

Dans cette dernière éventualité, on pourra rendre adjacent ce couple en ajoutant une liaison supplémentaire au graphe F. Une telle modification n'est pas très coûteuse car P' (cf. §III.3.4) ne varie, comme P, que d'une unité.

Application à forte disponibilité

Après occurrence d'une panne dans une paire travaillant en mode duplex, deux tâches seront assurées par deux calculateurs travaillant en mode simplex jusqu'à ce qu'une réparation de la paire défaillante intervienne.
Si ce temps de réparation est suffisamment bref pour qu'une nouvelle panne dans les deux calculateurs travaillant en simplex soit très improbable le système jouira d'une bonne disponibilité. On remarquera que ce temps de réparation peut être raccourci par un diagnostic utilisant les signaux de défaillance (Figure II.5) et par une implantation facilitant le remplacement d'une paire.

II - 5. Conclusion

La bonne fiabilité produite par notre solution est partiellement expliquée par sa simplicité matérielle et notamment par l'absence de "points durs"; goulots d'étranglement de la fiabilité. (pas de mécanismes de partage de mémoire, pas de bus communs, pas d'horloge ni d'alimentation commune).
Mais, cette simplicité n'est acquise qu'au prix d'un accroissement de la complexité du logiciel sur lequel repose désormais :

- Les comparaisons et "l'espionnage" au sein d'une paire.
- La synchronisation des échanges dans le réseau.
- La reconfiguration et la reprise après une défaillance.
- Et éventuellement l'auto-test des calculateurs évoluant en mode simplex pour améliorer la sécurité.

L'évaluation précédente ne prend pas en compte la fiabilité de ce logiciel. Celui-ci constituant, en quelque sorte, le seul "point dur" de la structure, des méthodes adaptées devront être employées pour rendre sa conception la plus fiable possible.

III - APPLICATION A UN EXEMPLE REEL : CONCEPTION D'UNE CENTRALE ANEMOMETRI

But

Ce chapitre n'a pas pour objet la simple présentation de la construction d'un système spécialisé à haute fiabilité. Cette présentation ne serait alors qu'un résumé du document complet [CERT 78]. Notre propos ici est double :

- Valider la méthodologie proposée au chapitre I en montrant son application à un problème réel.
- Illustrer par un exemple, l'utilisation et les avantages de la stratégie de redondance du Chapitre II.

Notons enfin, que nous insistons délibérément sur les points essentiels de méthode, au détriment de détails, importants dans une réalisation effective, mais secondaires pour la présente validation. Ces détails sont amplement développés dans le document [CERT 78].

III - 1. Présentation

III - 1.1. Rôle d'une centrale anémométrique

La mission d'une centrale anémométrique consiste en l'élaboration d'informations relatives à la trajectoire de l'appareil (vitesse, altitude..) à partir de données brutes fournies par un ensemble de capteurs. Ces informations sont alors transmises à l'organe chargé du pilotage de l'avion : pilote humain ou automatique. La centrale anémométrique s'inscrit donc dans la "boucle" de contrôle du vol de l'appareil.

III - 1.2. Cahier des charges

Le cahier des charges de cette centrale fournit :

- Les caractéristiques fonctionnelles des capteurs, tant statiques (précision, gamme de mesure..), que dynamiques (bande passante..).
- Les caractéristiques matérielles de ces mêmes capteurs (codage, mode d'accès...).
- Les relations fonctionnelles liant les différentes grandeurs d'entré (sorties capteurs) et de sortie.

- Les performances dynamiques de chacune des sorties de cette centrale
- Le protocole de connexion sur un bus standard de l'avion.

Il impose également l'utilisation d'un matériel particulier : les calculateurs monolithiques. A ces rubriques il convient de rajouter des contraintes de fiabilités sur lesquelles nous reviendrons au paragraphe C

Rappelons que les calculateurs monolithiques (INTEL 8748, MOSTEK 3870) sont des circuits d'apparition récente comprenant, intégrés sur le même "chip", une unité centrale, des blocs de mémoire de programme (ROM) et de données (RAM), quelques interfaces d'entrée/sortie digitales et parfois même la plus grande partie d'un circuit générateur d'impulsions pouvant jouer le rôle d'horloge.
Ce type de circuit peut donc, sans aucun dispositif externe, mis à part l'alimentation, supporter une application complète.
Il convient de remarquer toutefois que ces circuits ont encore des performances limitées (vitesse, taille mémoire, nombre d'interfaces d'entrée/sortie) et ne peuvent être utilisés tout seul que pour des applications de taille relativement modeste. Pour des applications plus délicates, comme celle qui nous intéresse, on sera obligé, d'une part de recourir à des solutions comportant plusieurs microcalculateurs et, d'autre part, d'augmenter les possibilités de ces derniers en leur adjoignant quelques circuits externes auxiliaires (extension mémoire, circuit d'interface supplémentaire...).

A - Fonctions de la centrale anémométrique

1 - La centrale anémométrique reçoit cinq entrées :
- Une mesure de pression statique brute (P_{si}). P_{si} représente la pression barométrique du milieu dans lequel évolue l'avion.
- Une mesure de pression totale brute (P_{ti}). P_{ti} représente la pression barométrique ambiante augmentée de la pression supplémentaire due à la vitesse de l'avion.
- Une mesure de température externe (T_i).
- Une mesure d'incidence dite incidence locale (α_L).

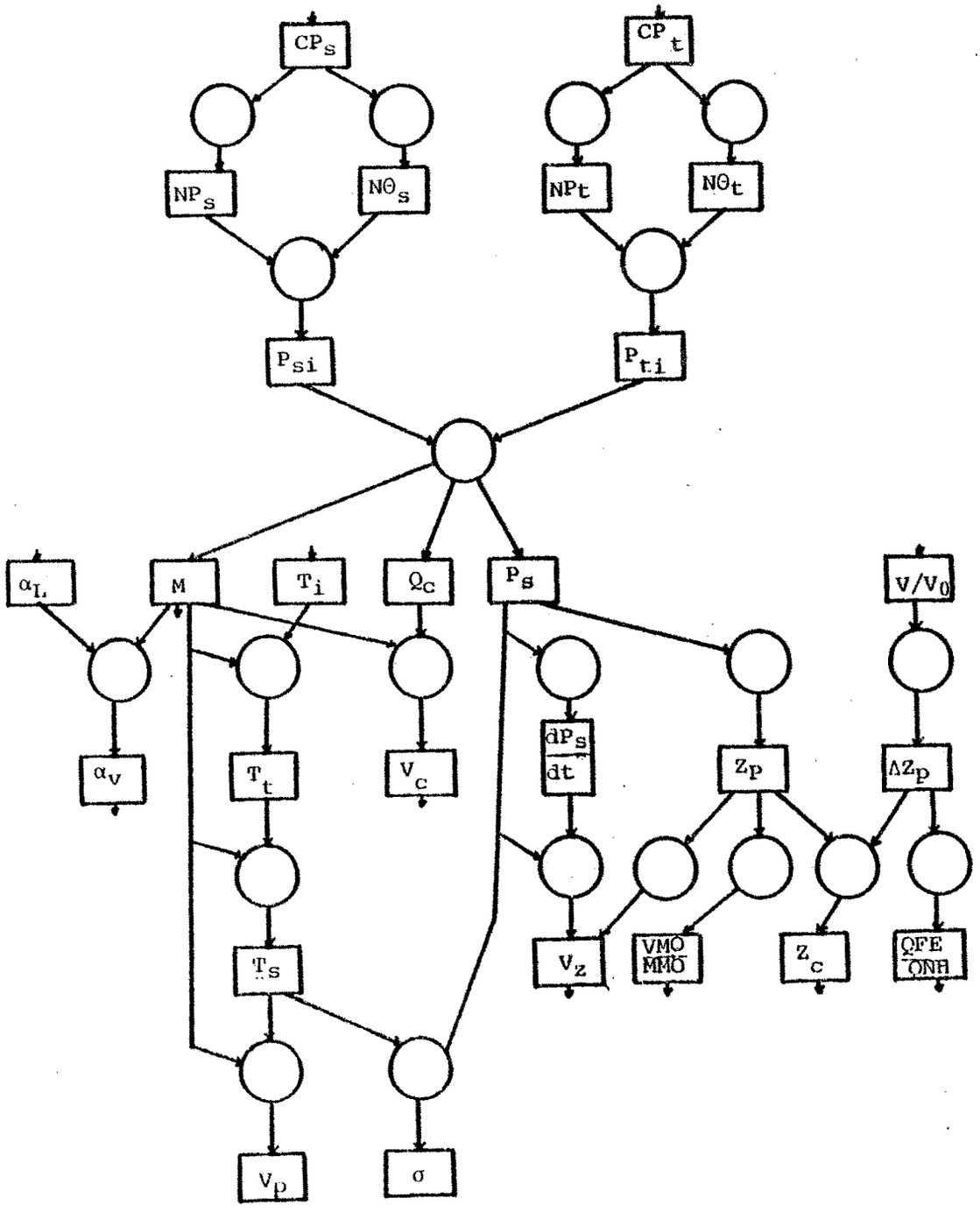
- Un paramètre externe modifiable donnant une information liée à la pression au sol (V/Vo). Ce paramètre permet le calcul de l'altitude réelle de l'avion à partir de la pression statique.

2 - La centrale doit fournir

- Diverses mesures concernant la position de l'avion :
  - une altitude dite altitude pression (Zp), obtenue directement à partir de Ps_i et Pt_i sans aucune correction,
  - une altitude dite altitude corrigée (Zc), obtenue en appliquant à Zp une correction due à la pression au sol (V/V0) ;
  - la correction en pression calculée à partir de (V/Vo) : (QNE / QNH),
  - l'incidence vraie (α_v) calculée à partir de l'incidence locale en tenant compte de la vitesse de l'avion (nombre de MACH : M).
- Des mesures concernant l'atmosphère ambiante :
  - la température totale (T_t) obtenu en appliquant à T_i (température externe) une correction due à la sonde de température. Cette correction est fonction du nombre de MACH(M),
  - la température statique (T_s). Cette mesure se déduit de T_t en tenant compte du nombre de MACH(M),
  - la pression statique (P_s). De façon analogue à la précédente, cette mesure se déduit de la pression statique brute (P_{si}) en tenant compte du nombre de MACH et de l'incidence locale (α_l
  - la densité réelle (σ) directement calculée à partir de P_s et T_i
- Des mesures de vitesse de l'avion :
  - nombre de MACH(M) obtenu à partir des deux mesures de pression brutes(P_{Si}, P_{Ti}),
  - vitesse conventionnelle (V_c),
  - vitesse propre (V_p) calculée à partir du nombre de MACH et de la température statique.
- Une valeur de vitesse limite (seuil de sécurité) à ne pas dépasser. Cette vitesse est fonction de l'altitude pression et des caractéristiques de l'avion. Ces caractéristiques pourront être stockées dans une mémoire interne de la centrale.

A ces fonctions doivent être ajoutés :

- d'une part le calcul des grandeurs P_{Si} et P_{Ti} à partir des signaux fournis par les capteurs : NP_s, $N\Theta_s$ et NP_t, $N\Theta_t$;
- d'autre part la sortie des résultats vers un bus externe. A ce niveau il convient de signaler que cette étude n'était qu'une étude de "faisabilité" et non de réalisation. Le cahier de charge ne spécifiait que le type de la liaison avec ce bus (liaison série) sans mentionner ni le mode (synchrone ou asynchrone), ni aucun format d'information.



B - Fréquence de calcul

Après confrontation du graphe de donnée et des bandes passantes respectives il a été possible de décomposer les calculs exigés en trois groupes de cadences différentes (périodes choisies : 39 ms, 312 ms, 1248 ms).

C - Spécifications de fiabilité

Chacun des signaux émis par la centrale peut être considéré comme une mission élémentaire. Pour chaque signal le cahier des charges prévoit une contrainte de fiabilité, donnée par un taux de panne équivalent sur une mission de durée fixée (dix heures). On constate que le cahier des charges n'exige aucune contrainte particulière de sécurité et ne propose aucun mode de fonctionnement dégradé .

Les différentes missions sont rangées en trois catégories suivant leur "taux de panne". On distingue, de la plus sévère à la moins sévère, les signaux dont la perte est :

- très dangereuse : taux de panne inférieur ou égal à 10^{-7} pannes par heure : V_c , Q_c ;
- dangereuse : taux de panne compris entre 10^{-7} et 10^{-5} pannes par heure : Z_p, M, P_s, α_v ;
- mineure : taux de panne pouvant être supérieur à 10^{-5} pannes par heure ; les autres signaux .

Nous avons directement traduit ces taux de panne en mesure de fiabilité en tenant compte de la durée d'exposition au risque (dix heures) ; ce qui donne des fiabilités limites respectives de $1 - 10^{-6}$ et de $1 - 10^{-4}$ pour des taux de panne de 10^{-7} et 10^{-5} pannes par heure.

Remarquons la "dureté" des contraintes de fiabilité exigées. Celles-ci sont, pour des missions de dix heures, tout à fait comparables à celles exigées dans le domaine spatial :

Exemple : [WENS 72] R = 99,5/100 pour une durée de $t = 5\ 10^4$ heures ($\sim$6ar

$$\lambda = \frac{-\text{Log } R}{t} = 1{,}0025\ 10^{-7}$$

III - 1.3. Architecture fonctionnelle

Le problème consiste à proposer un réseau de calculateurs capable de réaliser les fonctions d'une centrale anémométrique. Ce réseau doit respecter les spécifications fonctionnelles qui consistent essentiellement en des temps de réponse et des précisions minimales (III.1.2 B).
Les critères de choix de cette architecture ont été par ordre décroissant :

1) Minimisation du matériel nécessaire à la réalisation des diverses fonctions. Ce matériel a été fixé en examinant ces fonctions par ordre de fiabilité décroissant.

2) Minimisation des échanges à l'intérieur du réseau.

3) Equilibrage des charges et des volumes mémoire respectifs de chaque noeud.

Cette étude [CERT 78] a montré que :

- les missions de pertes très dangereuses ne pouvaient être remplies par un seul calculateur;
- deux calculateurs interconnectés suffisent à accomplir ces missions;
- cette structure à deux calculateurs peut remplir toutes les fonctions de la centrale anémométrique. Sur cette structure on a pu déterminer un ordonnancement et une répartition des tâches qui optimise les deux derniers critères .

Cette étude a abouti à l'architecture représentée sur le figure III.1.

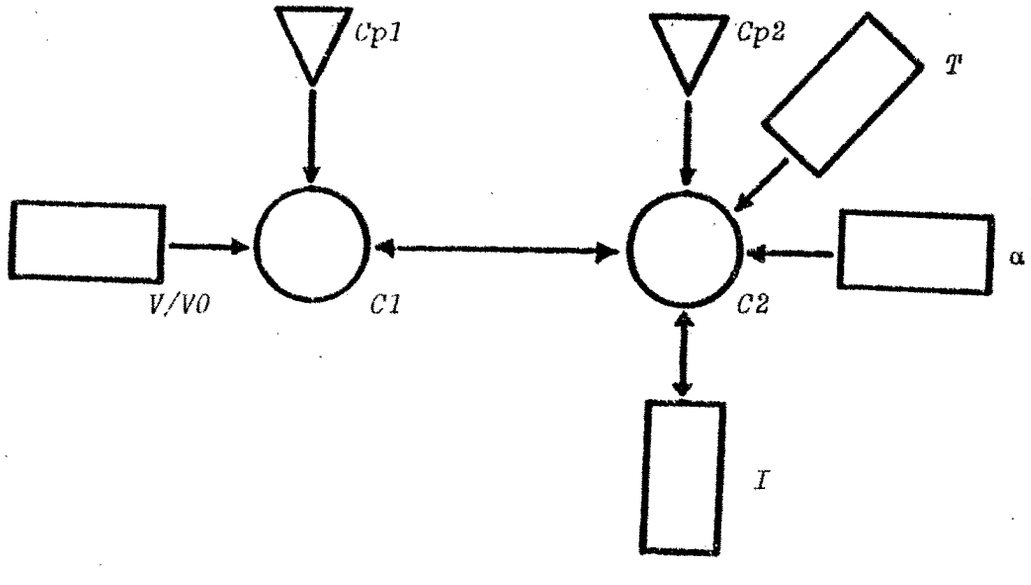


Figure III.1

Sur cette figure, les divers symboles ont la signification suivante :

: désigne un capteur. L'architecture nominale possède deux capteurs de type différent (l'un mesurant la pression totale P_{Ti} et l'autre la pression statique P_{Si}), notés C_{p1} et C_{p2} sur la figure III.1. Chaque capteur est directement connecté sur le bus du calculateur qui lui est adjacent (respectivement C_1 et C_2). Chacun de ces capteurs est vu par son calculateur sous la forme de deux zones mémoires :

- Une première zone, d'étendue restreinte, contient, codées en binaire, les valeurs délivrées par le capteur. Ces données résultent du passage des sorties fréquentielles du capteur à travers des échantillonneurs bloqueurs indépendants du calculateur. Ces données représentent une mesure de pression brute NP et une mesure de température NΘ .
- Une seconde zone, de volume plus grand, renferme un tableau de valeurs (codées en binaire) représentant la tabulation d'une fonction de deux variables donnant la valeur de la pression réelle P corrigée, à partir de la pression brute NP et de la température NΘ. L'exploitation de ce tableau par interpolation parabolique permet l'estimation de P. L'utilisation de cette méthode d'approximation coûteuse a permis de réduire considérablement la taille de cette mémoire qui reste pourtant de 1024 octets. Cette mémoire morte est une caractéristique propre du capteur; renfermant des valeurs établies expérimentalement, elle est fournie avec l'équipement.

: désigne un "groupe calculateur". Outre son calculateur monolithique le "groupe calculateur" comprend quelques circuits périphériques.

- Une extension mémoire pour pallier l'exigüité de la mémoire interne du microcalculateur choisi.
- Quelques circuits d'interfaces nécessaires à la connexion des capteurs de pression.

 : représente des circuits d'interfaces vers des capteurs distincts des capteurs pression décrits précédemment.

I designe l'interface de connexion avec le bus avion.

Remarques :

- Les capteurs de pression sont les seuls capteurs appartenant à la centrale anémométrique. L'étude et l'évaluation de fiabilité suivantes devront donc les prendre en compte.

- Les missions de perte très dangereuse n'ont pû être rassemblées sur un seul des deux calculateurs. Cette dispersion des ressources et des tâches nécessaires à la production des signaux les plus indispensables dans toute la structure, a deux conséquences :

  1) La plus grande partie de l'architecture devra être préservée avec une fiabilité supérieure ou égale à $1 - 10^{-6}$ pour une mission de 10 heures.

  2) Les missions dont la perte est la plus dangereuse, mettent en jeu les deux calculateurs de la structure. Les rôles de ces deux calculateurs devront être assurés avec une fiabilité au moins égale à $1 - 10^{-6}$. Il s'en suit que les autres missions, conduites elles-aussi par ces deux calculateurs, mais nécessitant les interfaces spécifiques α et T, pourront voir leurs fiabilités atteindre et même dépasser le chiffre de $1 - 10^{-4}$ exigé. Rien ne nous empêchera, en effet, de rajouter autant d'interfaces α et T que nécessaire, la détection et le remplacement des interfaces défaillantes étant assurés par les calculateurs eux-mêmes. Cette tâche peut être effectuée par un logiciel spécialisé.

En conséquence, nous devions donc assurer la survie pendant dix heures, avec une fiabilité meilleure que $1 - 10^{-6}$, de la structure donnée par la figure III.2.

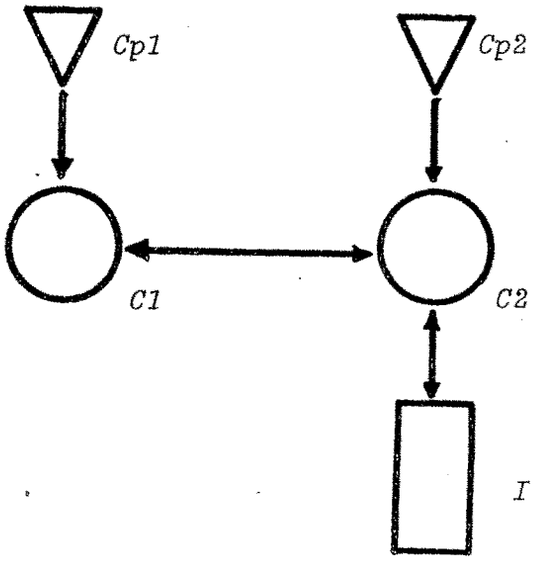


Figure III.2

- Alimentation de la centrale

Le cahier des charges indique que la distribution d'électricité à bord de l'avion est assurée par une barre d'alimentation unique sur laquelle doivent être branchés les régulateurs propres à chaque appareillage. Nous ne devions pas nous intéresser à cette alimentation mais simplement aux régulateurs propres à la centrale.
Au cours de cette étude il a été possible de définir un régulateur dupliqué de fiabilité (sur 10 heures) largement supérieur à $1 - 10^{-6}$
Ce dispositif, qui n'a pas été essayé, repose sur une détection et une commutation par diodes (standards et zeners); le régulateur défaillant étant ensuite éventuellement déconnecté par rupture de fusible. La simplicité du schéma et l'utilisation de composants surdimensionnés (en puissance et en tension de claquage) expliquent les bonnes performances obtenues.

III - 2. Etapes méthodologiques

III - 2.1. Evaluation des taux de pannes (1ière phase)

Toute méthode ou méthodologie de conception d'une architecture à haute fiabilité construite à partir d'un ensemble de composants doit nécessairement débuter par l'évaluation du taux de panne de chacun des composants choisis. En ce qui concerne les capteurs, cette évaluation était directement fournie par le cahier des charges. Dans le cas des circuits électroniques, cette étape a été rendue difficile par la nouveauté des produits utilisés. La plupart des modèles d'estimation des taux de panne de circuits intégrés ne correspondaient qu'à des technologies SSI ou MSI et étaient inopérants dans le cas des circuits LSI et VLSI largement utilisés dans cette étude.

Exemple : Une extrapolation des modèles du [MIL 74] ou du modèle du CNET [CNET 76](avant sa révision de 1979) conduit à des taux de panne d'une panne par heure pour des circuits possédant une complexité équivalente à 3000 portes logiques (mémoire 4K ou 16K aujourd'hui disponible). Ces circuits n'auraient donc même pas pu être construits.

En conséquence, pour l'évaluation de taux de panne des circuits LSI, nous avons utilisé un modèle encore expérimental [KASOUF 78],dont les estimations ont été recoupées avec des données constructeurs [MOTO 78] [INTEL 76]. Les circuits SSI et MSI ont été classiquement estimés grâce au modèle [CNET 76]. Ces estimations ont produit les résultats (résumés et simplifiés) suivants :

- Taux de panne du groupe calculateur : $\lambda c \leq 2{,}5 . 10^{-5}$ pannes par heure.
- Taux de panne d'un capteur : $\lambda cp \leq 2{,}8 . 10^{-5}$ pannes par heure.

Le groupe calculateur dont nous donnons ici le taux de panne comprend toutes les extensions mémoires où interfaces, y compris celles (éventuelles) avec le bus avion, à l'exclusion des interfaces V/Vo, α_v et T qui pourront être traitées ultérieurement suivant la remarque du § III.1.2.

Evaluation de l'architecture fonctionnelle

Tous les éléments de l'architecture présentée sur la figure II.2 sont indispensables et, donc, placés en "série" du point de vue de la fiabilité : leurs taux de pannes s'ajoutent pour donner le taux de panne de la structure globale λ_s. Il vient :

$$\lambda_s = 1{,}06\ 10^{-4}$$

Soit une fiabilité R_S (10h) pour une mission de dix heures égale à :

$$R_s\ (10h) = 0{,}99894$$

L'architecture fonctionnelle présente donc une fiabilité inférieure à celle requise par le cahier des charges:

$$R_s\ (10h) < 1 - 10^{-6}$$

Etude des stratégies de redondance classiques

Avant de passer à la méthode du chapitre I,on a vérifié que les stratégies tripliquées classiques, appliquées globalement à la centrale ne conviennent pas.

- TMR : Cette stratégie produirait une fiabilité R_{TMR} (10h) pour une mission de dix heures qui ne pourrait excéder la grandeur $R^{MAX}_{TMR}(10h) = 3\,R^2_s\,(10h) - 2R^3_s\,(10h)$. R^{MAX}_{TMR} représente la fiabilité produite par une stratégie TMR dans le cas idéal des voteurs parfaitement fiables.

  D'après le calcul précédent il vient :

  $$R^{MAX}_{TMR} \approx 1 - 3{,}36\ 10^{-6}$$

Soit $R_{TMR}\ (10h) < 1 - 3{,}36\ 10^{-6}$

et par suite : $R_{TMR}\ (10h) < 1 - 10^{-6}$

Une stratégie TMR, appliquée globalement à toute la centrale, ne peut donc pas satisfaire la contrainte de fiabilité exigée par le cahier des charges.

- TMR/Simplex

  Le même raisonnement conduit à une conclusion analogue en ce qui concerne la stratégie TMR/Simplex.

  On a en effet :

  $R^{MAX}_{TMR/simplex}$ (10h) $= \frac{3}{2} R_s$ (10h) $- \frac{1}{2} R_s^3$ (10h) $= 1 - 1{,}518\ 10^{-6}$

  qui est encore inférieur à $1 - 10^{-6}$.

Il est important de souligner l'influence de ces évaluations pour la suite du processus de conception. Ces taux de panne seront notamment utilisés lors du partitionnement du système.

III - 2.2. Partitionnement, stratégie et redondance (2 & 3ième phases)

Nous avons remarqué que ni les stratégies tripliquées globales (TMR, TMR/simplex), ni à plus forte raison, la solution nominale ne satisfaisaient aux contraintes de fiabilité requises.

Avant de songer à des solutions comportant un niveau de redondance plus élevé, et donc un coût plus fort, on peut tenter de recourir au partitionnement du système. Si l'un des critères de partitionnement peut être l'équilibrage des taux de panne des différentes partitions, un critère déterminant est la possibilité d'implantation de stratégies de redondance dans chaque partition . Cette "applicabilité" est fonction des possibilités de détection afférantes à chaque composants. Nous allons donc examiner successivement chaque composant de la structure dans cette optique.

III - 2.2.1 Méthodes de détection

1) Capteurs : nous ne disposions pour ces organes que d'analyses très incomplètes des modes de défaillances ne permettant d'élaborer qu'un test (vraisemblance + signaux de contrôle) à couverture relativement modérée (estimée par le constructeur à 70%). La faiblesse de ce taux de couverture par rapport à la fiabilité exigée impose, pour la détection, la comparaison des signaux générés par (au moins) deux capteurs identiques.

2) Groupes calculateurs : en ce qui concerne le groupe calculateur plusieurs solutions pourraient être envisagées pour les différents composants d'un groupe :

- les mémoires et les interfaces d'entrée/sortie digitales sont bien adaptées aux techniques de redondance partielle (codes détecteurs d'erreurs) ;
- les calculateurs eux-mêmes, vu les taux de couverture exigés (supérieurs à 99%) en raison de la haute fiabilité désirée, ne semblent redevables que d'une technique de duplication et d'une comparaison des résultats.

III - 2.2.2. Partitionnement et analyse de la redondance

Plusieurs autres éléments ont contribué au choix définitif de stratégies de redondance pour chaque composant :

- des critères d'équilibrage de taux de panne et de simplicité de connexion conduisent à ne pas partitionner le groupe calculateur. Les circuits monolithiques, imposés par le cahier des charges, se révèlent très mal adaptés aux techniques de partage de mémoires qui auraient été rendues nécessaires par un éventuel partitionnement du groupe calculateur;
- la fonction anémométrique elle-même est une fonction cyclique ne demandant aucune mémorisation d'information d'un cycle de calcul sur l'autre. Cette disposition permet d'envisager des méthodes de détection/remplacement par opposition aux solutions tripliquées (TMR) plus faciles à utiliser lorsque toute perte d'information est catastrophique. Notons également que le cahier des charges autorisait l'émission accidentelle d'échantillons erronés, ce qui facilitait le traitement des pannes transitoires dans le cas d'une solution à détection/remplacement.

1) Groupes calculateurs : le choix d'une détection, au niveau d'un groupe calculateur, par comparaison avec les signaux d'un groupe identique conduisait (au moins) à une triplication de chacun des groupes.

Pour tenter de diminuer la "masse" de matériel utilisée nous avons appliqué la méthode proposée au chapitre précédent. Le réseau de calculateur utilisé dans cet exemple est le réseau de la Figure III.3 qui conduit donc, par application de la méthode précédente, au réseau de la Figure III.4.

Figure III.3

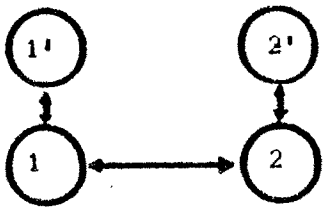


Figure III.4

Ce qui produit une fiabilité R_N estimée par la formule :

$$R_N = 2R_C^2\ (10) - R_C^4\ (10) \qquad \text{avec } R_C = e^{-2,5\,10^{-5}} \times 10 \approx 1-2,5\,10^{-4}$$

$$R_N = 1-2,5\,10^{-7}$$

R_N satisfait bien la condition nécessaire : $R_N > 1-10^{-6}$

2) Capteurs : Le choix de cette stratégie conduit donc à un partitionnement de degré trois de l'architecture initiale :

1 - capteur de type 1 (cp1),

2 - capteur de type 2 (cp2),

3 - groupe processeur .

En raison de la méthode de détection de panne choisie, chaque capteur doit être au minimum tripliqué. Parmi les trois stratégies tripliquées classiques (TMR, duplex-réserve, TMR-simplex) nous avons éliminé les deux stratégies comportant un vote majoritaire (TMR, TMR-simplex). Leur mise en oeuvre directe nécessitait la connexion simultanée de trois capteurs sur un même calculateur, lequel aurait alors assuré la procédure de vote majoritaire. Cette solution imposait un mécanisme complexe d'adressage en raison de l'étendue de la mémoire interne des capteurs et de l'exiguïté de l'espace d'adressage des microcalculateurs utilisés.
La stratégie retenue pour chaque capteur a donc été celle du duplex-réserve qui conduisait à une fiabilité Rd égale à :

$$R_d = R_{cp} - R_{cp}^3 + R_{cp}^2 \approx 1 - 12{,}5\ 10^{-8} \quad (R_{cp} \approx 1 - 2{,}8\ 10^{-5})$$

Notons que la fiabilité théorique totale R_S du système utilisant ces stratégies dépasse la fiabilité exigée :

D'après la formule (1) il vient :

$$R_S = R_d^2\ R_N \approx 1 - 3\ 10^{-7} \qquad (4)$$

Qui se trouve également vérifier la condition nécessaire $R_S > R_{MIN}$.

III - 2.3. Solution complète (4ième phase)

L'étude précédente indique qu'une architecture, susceptible de supporter à la fois la stratégie proposée pour les capteurs et la stratégie choisie pour les calculateurs, satisfera la contrainte de fiabilité donnée par le cahier des charges. La figure ci-après représente le schéma d'une telle architecture (Figure III.5).

N.B. Sur cette figure les fils de commande des commutateurs ne sont pas représentés.

On notera l'adjonction de circuits commutateurs ("SWITCH") supplémentaires Le taux de panne de ces circuits viendra évidemment diminuer la fiabilité de mission. Mais cette diminution restera tout de même très limitée en raison des deux facteurs suivants :

- taux de panne relativement faible du commutateur λSWITCH devant celle du capteur auquel il est associé.
  λSWITCH = $2{,}2\ 10^{-6}$ p/h
  λSWITCH est donc plus de dix fois plus faible que le taux de panne d'un capteur.
  La fiabilité du commutateur R_{SW} pour une mission de dix heures est donc égale à : $1 - 2{,}2\ 10^{-5}$,

- l'estimation de la fiabilité de mission associée à la structure schématisée sur la figure III.5, se déduit de la formule (4) en modifiant la fiabilité originale des capteurs pour tenir compte de la fiabilité du commutateur associé :

  $$R_{cp\ modifié} = R_{cp\ original} \times R_{SW} \qquad (5).$$

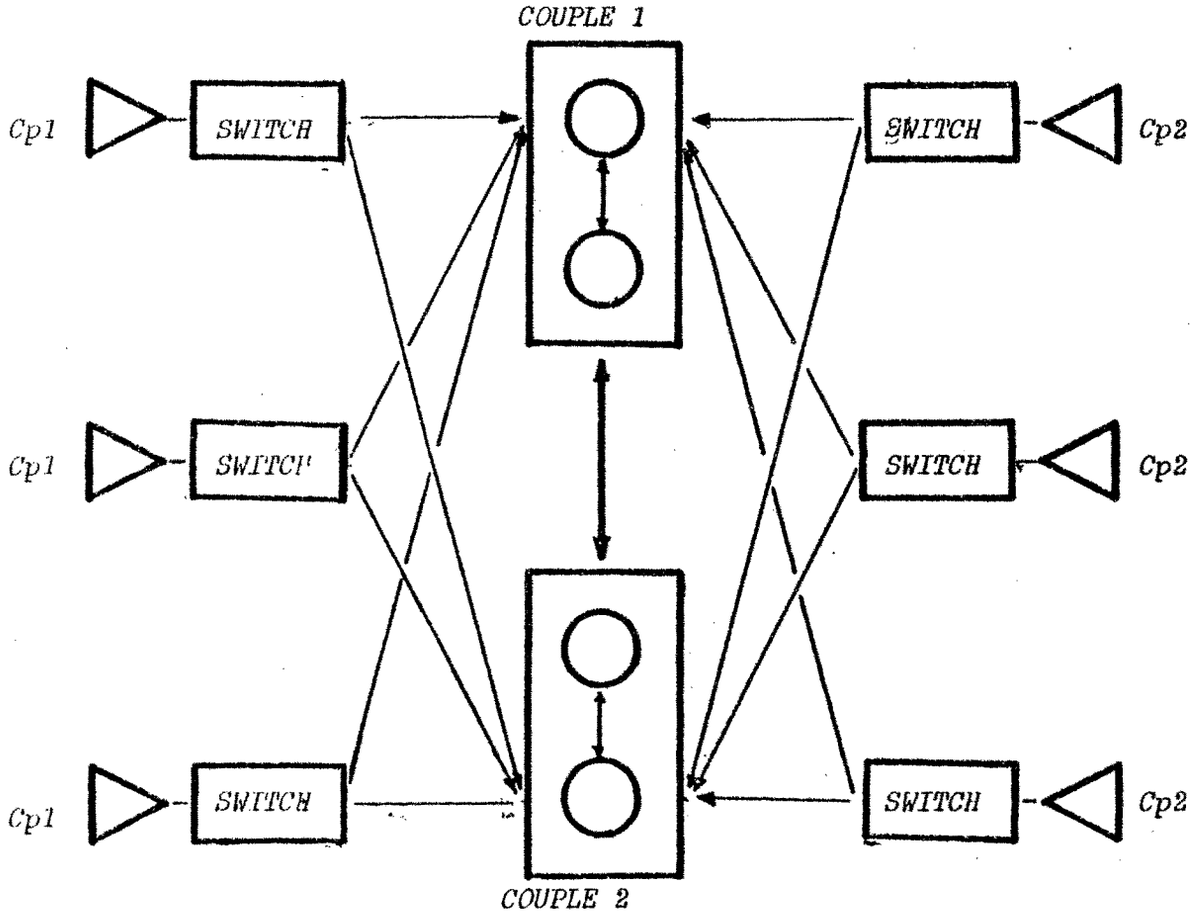

COUPLE 1
Cp1
SWITCH
SWITCH
Cp2
Cp1
SWITCH
SWITCH
Cp2
Cp1
SWITCH
SWITCH
Cp2
COUPLE 2

Figure III.5

La relation 4 (ou 4 modifiée en 5) montre que la solution proposée excède largement en fiabilité les exigences du cahier des charges. Une question se pose alors : ne peut-on pas diminuer la complexité de cette solution tout en produisant une fiabilité globale supérieure à $1-10^{-6}$? Les nombres d'exemplaires de chaque unité étant nécessaires (Cf. § I phase 3), il nous restait donc à tenter la suppression de certaines liaisons (diminuant parallèlement les possibilités de reconfiguration) tout en restant à des fiabilités situées au delà de $1-10^{-6}$. Remarquons que la solution "complète" de la figure III.5 aurait pû être écartée pour des raisons identiques à celles motivant l'élimination des stratégies tripliquées pour les capteurs.

III - 2.4. Solution par utilisation directe de la méthode du § II

Pour appliquer la méthode du deuxième paragraphe on intègre le capteur dans le groupe calculateur auquel il est connecté.

On est conduit à un schéma (Figure III.6) qui correspond bien à une simplification du schéma de la figure III.5.

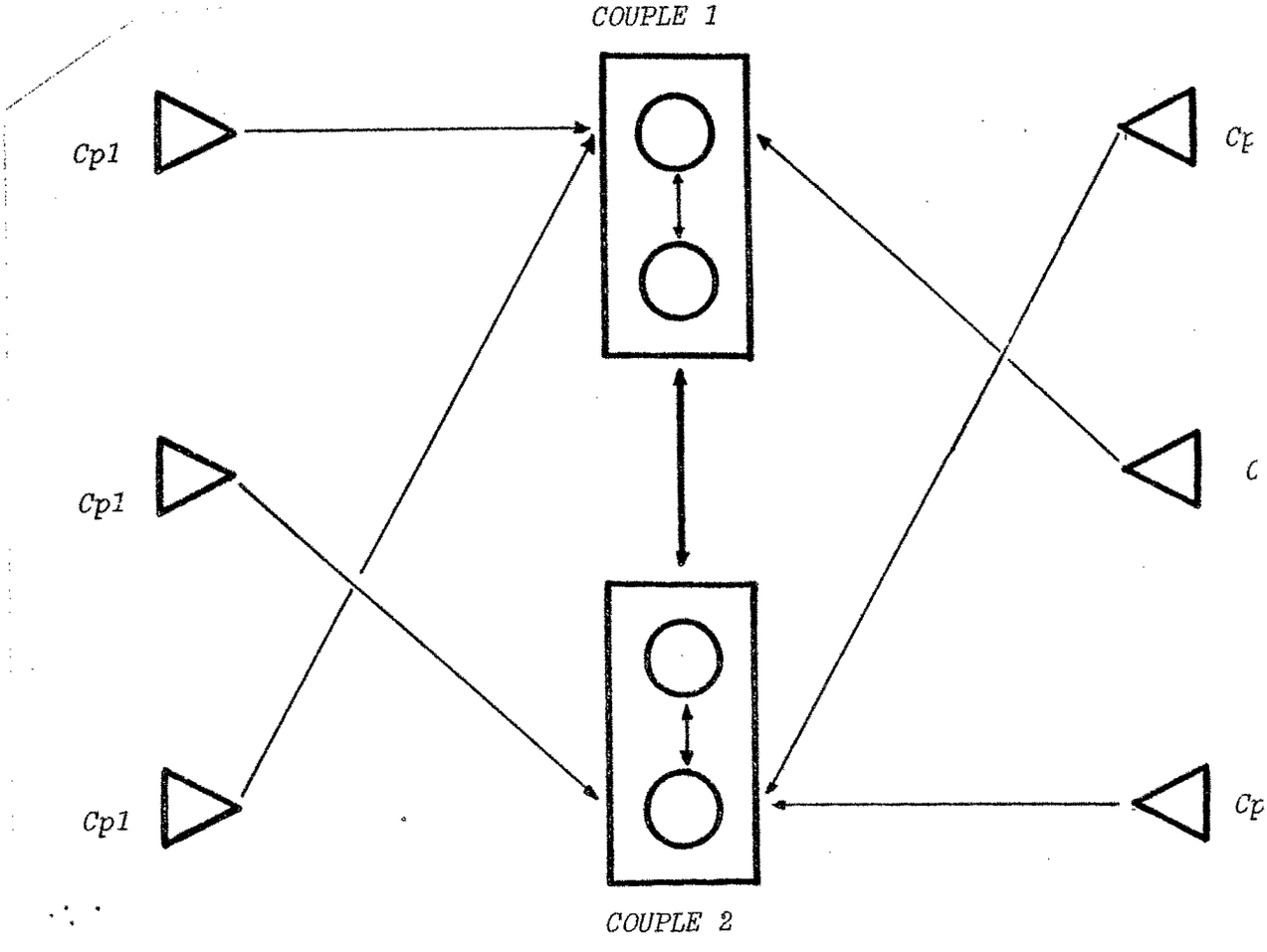


Figure III.6

Mais, une telle solution n'atteignait qu'une fiabilité n'excédant pas la valeur RT donnée par la formule :

$R_T = 2R_D - R_D^2$ (6) dans laquelle R_D désigne la fiabilité d'un ensemble formé d'une paire de groupes calculateurs et de trois capteurs. Nous avons $R_D = 1 - 1{,}34\ 10^{-3}$;

et donc par (6) il vient :

$R_T = 1 - 1{,}79\ 10^{-6}$

R_T est inférieure à $1 - 10^{-6}$ et donc la solution précédente ne peut être retenue : la réduction du nombre de liaisons a été trop forte (4ième phase, chapitre I).

III - 2.5. Solution proposée

La solution que nous proposons est, en quelque sorte, intermédiaire entre la solution représentée par la figure III.5 et celle de la figure III.6. Elle a pu être construite à partir de cette dernière en autorisant le partage des capteurs de réserve C^R_{P2} et C^R_{p1} entre les deux paires de calculateurs.

Cette architecture est représentée sur la figure III.7.

On remarque que la paire 1 (resp. la paire 2) possède deux capteurs non partageables du même typel(resp. type 2). L'accès aux capteurs partageables est fonction de l'état des paires de calculateurs de la structure :

- initialement le capteur partageable de type 1 (resp de type 2) est attribué à la paire de calculateurs possédant en propre deux capteurs du même type 1;
- lors de occurrence d'une panne au sein d'une paire, l'affectation du capteur partagé revient à l'autre paire qui est supposée encore valide.

La mise en oeuvre de cette stratégie est assurée par la commande directe des "circuits commutateurs" par les signaux de panne de chaque paire.

Après la présentation de cette architecture, nous décrivons une stratégie de reconfiguration adaptée à cette dernière (§2.5.1) avant d'en donner une évaluation de la fiabilité (§2.5.2).

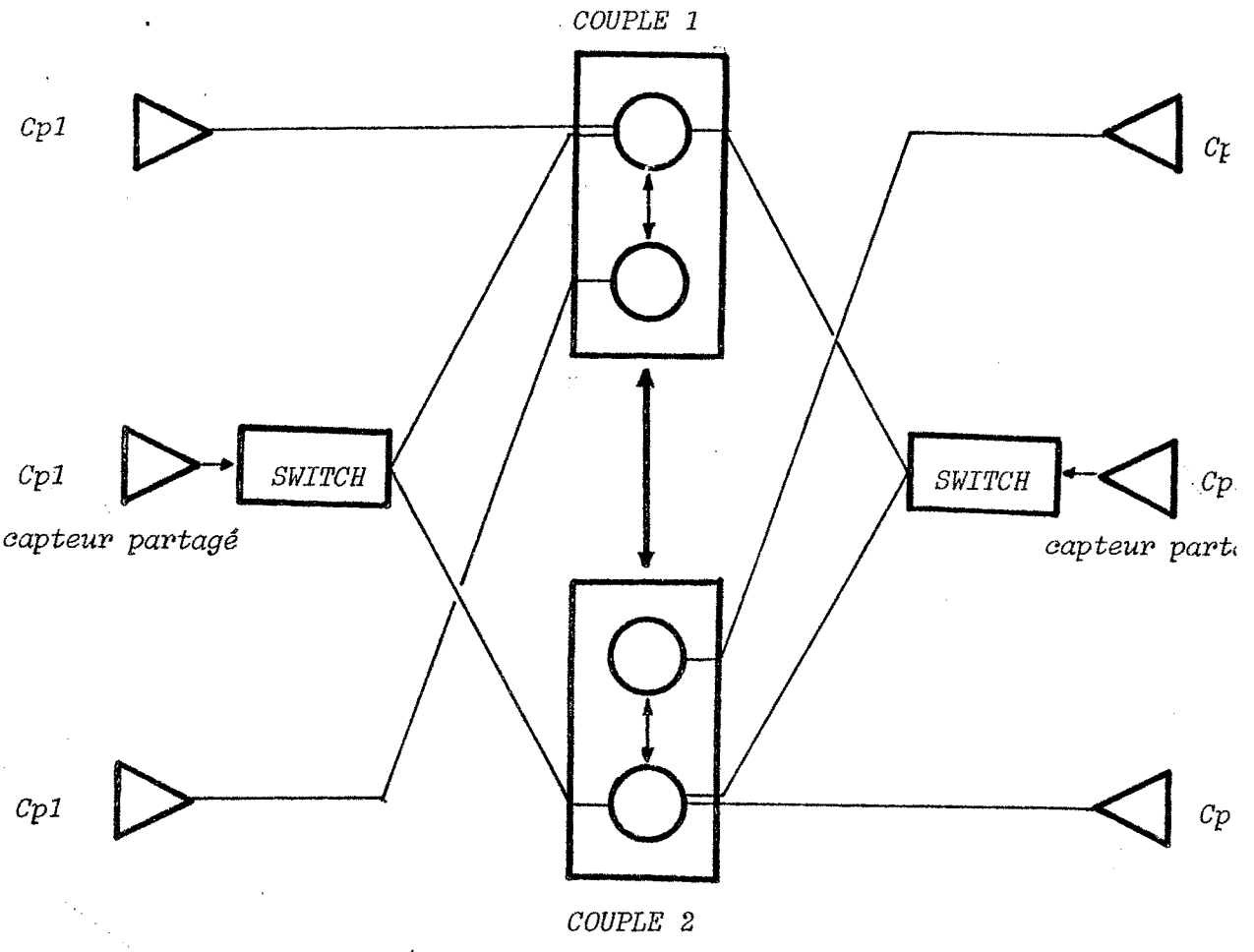


Figure III.7

III - 2.5.1. Stratégie de dégradation

Pour clarifier cette présentation, nous proposons une représentation symbolique des divers états de dégradation de la structure. Ce symbolisme comprendra trois éléments graphiques :

- l'ovoïde : une paire de processeurs,
- le triangle : un capteur non partagé,
- le carré : un capteur partagé.

Les arcs en pointillés (resp. en traits pleins) représenteront des liaisons possibles (resp. des liaisons effectivement utilisées). Les composants figurant sur la structure initiale (I) sans figurer sur une des autres configurations (II-VIII) sont des éléments défaillants, inutilisables ou définitivement inutilisés dans cette configuration. (Figure III.8).

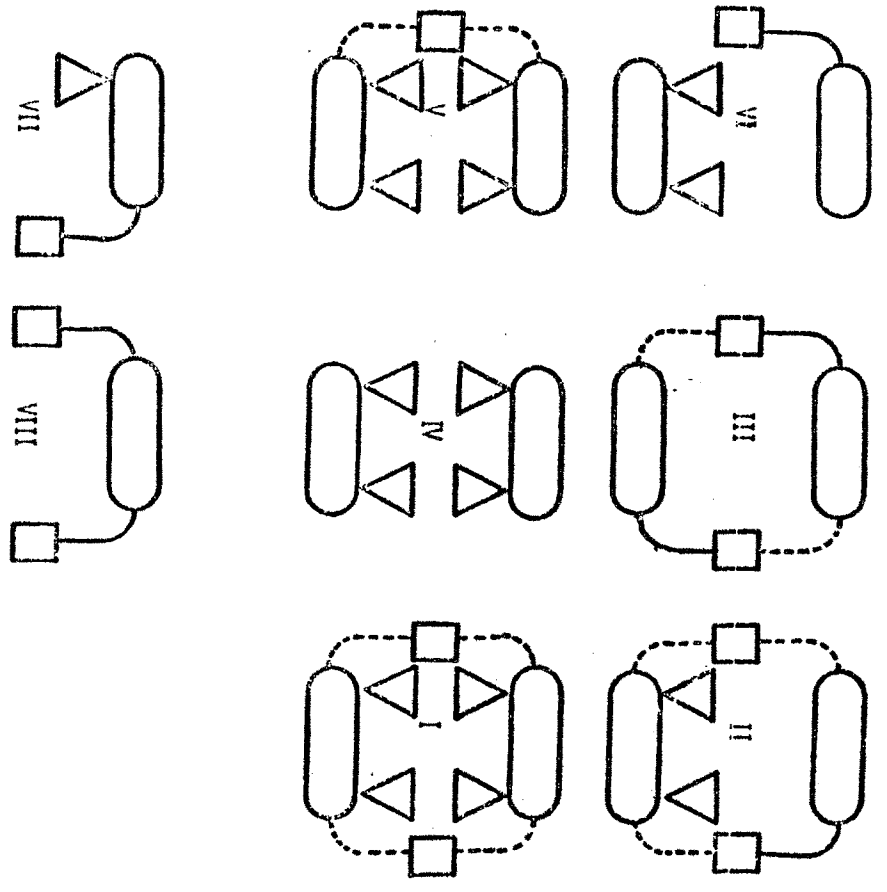


Figure III.8

N.B. : Chaque figure II, V, VI, VII, VIII symbolise deux configurations dégradées distinctes qui s'obtiennent par symétrie.

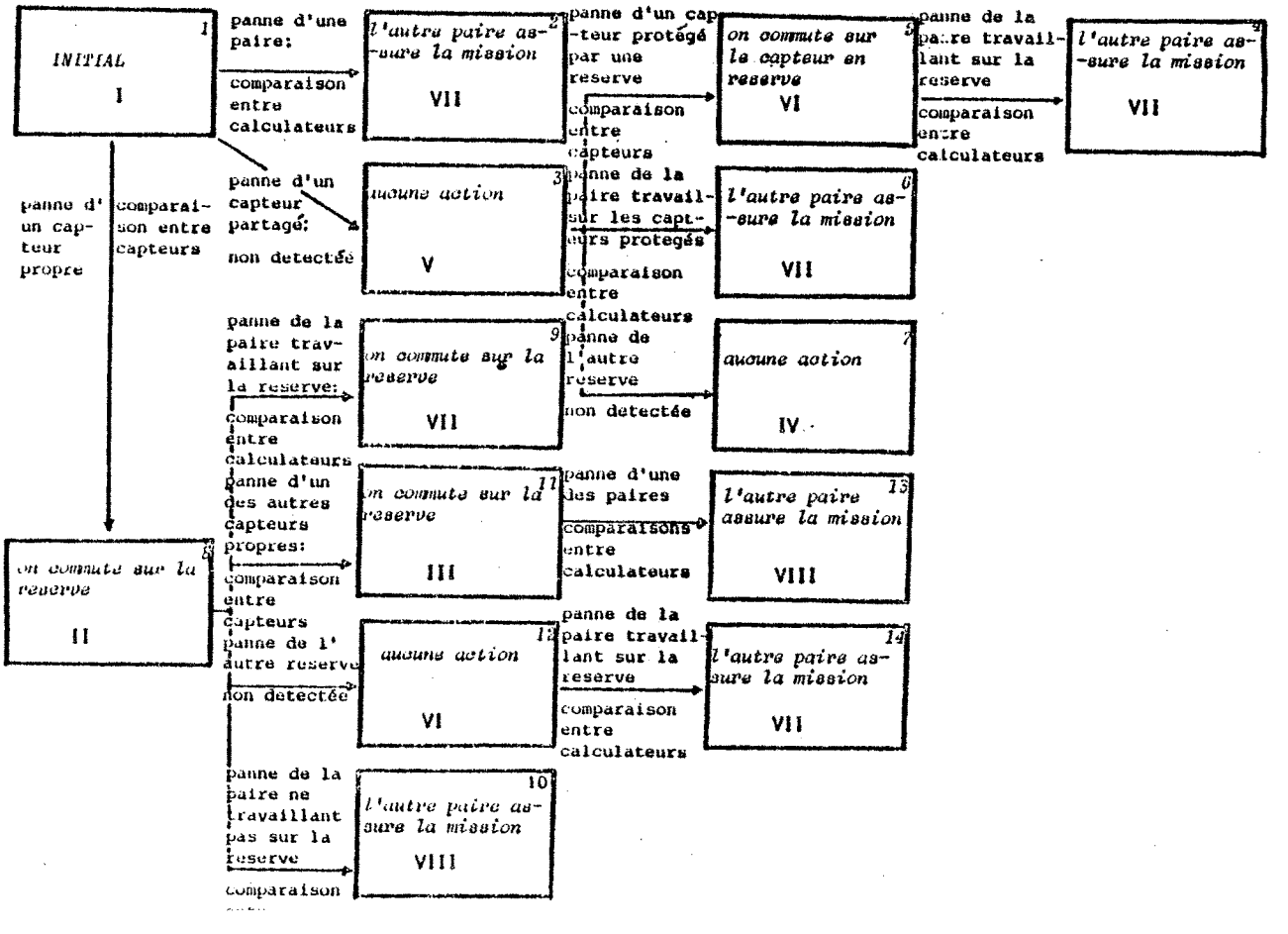


Figure III.9

La stratégie proposée est complètement représentée par la figure III.9. Dans ce schéma chaque rectangle présente un état atteignable au cours de cette stratégie. Chaque état porte trois informations distinctes :

- son numéro (1 à 14)
- le numéro de l'état de dégradation atteint (I à VIII)
- une brève indication des opérations de reconfiguration entreprises pour aboutir à cet état.

Chaque arc de ce schéma représente l'occurrence d'une panne ou d'un ensembl de pannes. Il porte les deux informations suivantes :

- une rapide description de cette panne ou de cet ensemble de pannes,
- la méthode de détection permettant le diagnostic de cette, ou de ces pannes.

Cette stratégie peut être facilement comprise en se plaçant au niveau d'une paire de calculateurs :

1) En l'absence de défaillance de l'autre paire, la procédure suivante est appliquée :

. Chaque calculateur de la paire travaille sur son capteur propre. Les résultats des deux calculateurs sont périodiquement comparés, notamment à l'émission de résultats vers l'extérieur.

. A l'apparition d'un désaccord persistant (non consécutif à une panne transitoire), on suppose qu'il s'agit d'une défaillance d'un capteur propre , et en conséquence on travaille sur le capteur de réserve, les deux calculateurs de la paire se contrôlant toujours mutuellement.

Cette dernière supposition ne résulte pas de leurs taux de pannes qui sont comparables (5.10^{-5}p/h pour les calculateurs et $5,6.10^{-5}$p/h pour les capte En supposant la panne d'un des deux capteurs nous choisissons simplement l'é tualité la moins préjudiciable à la structure : en effet, la supposition de la défaillance d'un calculateur conduirait également à la perte des deux capteurs propres. Remarquons enfin que si cette supposition de la défai lance capteur est fausse, le désaccord persistera et conduira au suicide de la paire de calculateurs (voir ci-après).

. Lors de l'occurrence d'un second désaccord persistant, on procède "au suicide" de la paire.

2) La défaillance de l'autre paire provoque le désaccouplement de la paire considérée. Celle-ci reprend alors toute la mission en s'emparant du capteur partagé appartenant primitivement à l'autre paire, et en travaillant soit avec un de ses capteurs propres, soit avec l'autre réserve, suivant l'état de ces deux capteurs propres, antérieurement à cette défaillance.

III - 2.5.2. Evaluation

Le but de ce paragraphe est d'établir une estimation de la fiabilité de mission fournie par la stratégie donnée ci-dessus appliquée à l'architecture de la figure III.7.

Cette estimation est effectuée en deux temps :

- Résolution d'un modèle Markovien approché : on néglige quelques événements de probabilité faible.
- Justification de ces simplifications.

Modélisation Markovienne

Ce modèle Markovien est directement issu du schéma de la figure III.9, après adjonction d'un état (puits) de défaillance (15) et regroupement des états {10, 13}, {2,4,6,9,14} et {12,5}.

Ce modèle Markovien est représenté sur la figure III.10.Chaque arc porte une formule de calcul de la somme des taux d'arrivées des pannes provoquant la transition entre les deux états connectés à cet arc.
Ces formules présentent trois variables P,S,C désignant respectivement :

P : taux de panne d'un capteur propre : $\lambda cp \approx 2{,}75 10^{-5}$ p /h

C : taux de panne d'une paire de calculateurs : $\lambda p \approx 4{,}89 10^{-5}$ p/h

S : taux de panne d'un capteur partagé : $\lambda^2 cp \approx 2{,}97 10^{-5}$ p/h.

L'évaluation de ces taux, plus précise que celle donnée précédemment est détaillée dans [CERT 78].
Dans la figure III.10,les numéros des états indiquent : soit le numéro de l'état correspondant dans la figure III.9, soit le numéro le plus faible des états de III.9 (regroupés) qu'ils représentent.

Ce modèle Markovien a été résolu par une méthode numérique classique (Runge-Kutta 4.4 avec un pas de 10^{-2} heures) pour une durée de 15 heures. Les résultats obtenus sont présentés sous forme de graphe sur la figure III.11. On remarque notamment que pour t = 10h on a $1-R(t) = 0,98\ 10^{-6}$; la fiabilité exigée est donc juste atteinte.

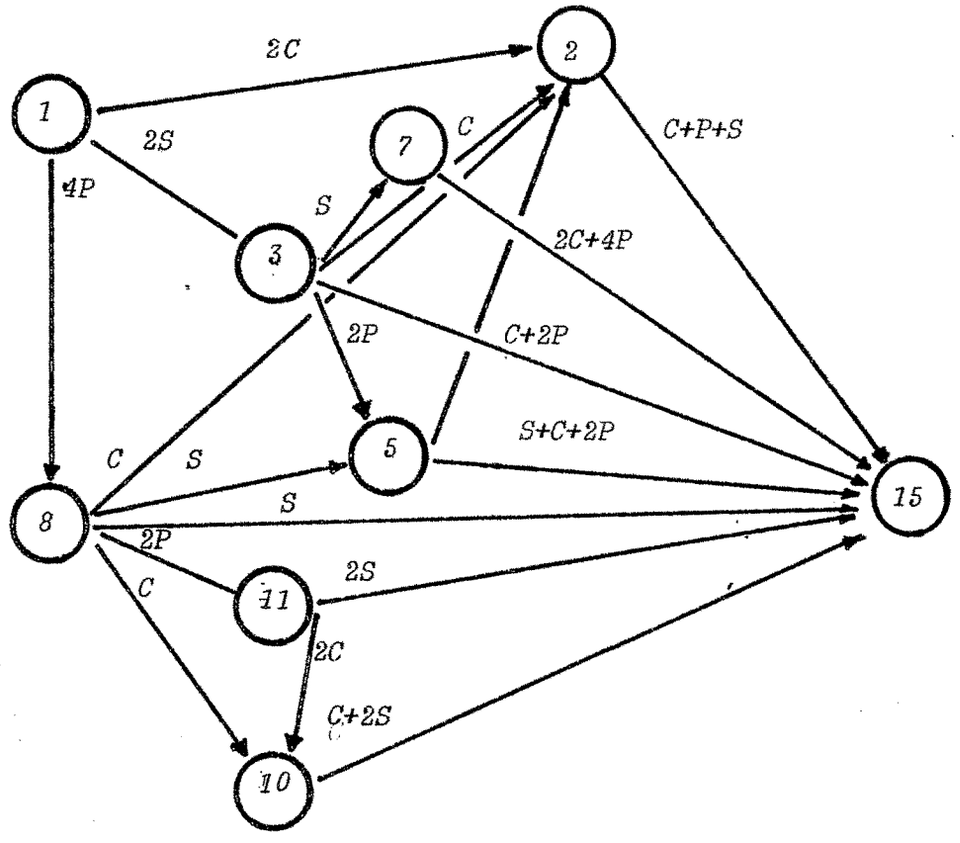


Figure III.10

Justification des simplifications

L'évaluation ci-dessus néglige quelques événements catastrophiques à très faible probabilité d'occurrence. Ces simplifications sont justifiées ci-dessous.

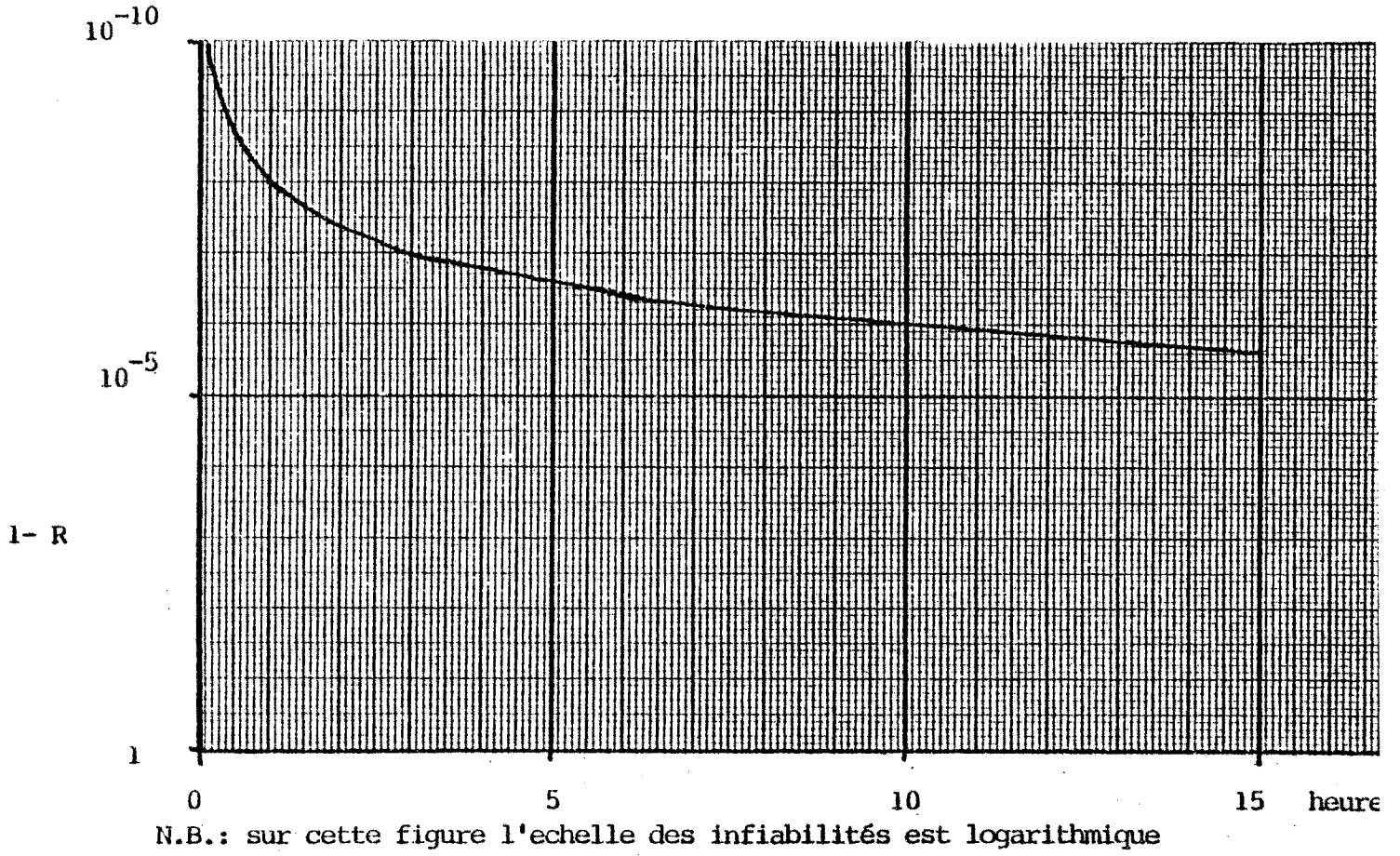
- Les résultats de l'annexe II permettent de ne pas considérer l'éventualité de pannes simultanées dans les deux processeurs d'une même paire.
- L'étude de l'alimentation conduit à une infiabilité de cette dernière égale à $6,8\ 10^{-8}$.

- Enfin, nous avons vu au chapitre II que le système mémorisateur de panne/"suicide" des calculateurs forme un "point dur" de cette architecture.
  En fait, cela ne va jouer que lorsqu'il y aura à la fois une panne dans la paire de calculateurs et une panne dans ce circuit; soit l'occurrence d'un événement ayant une probabilité très faible :
  ce circuit comporte (pour chaque paire) :
  . 1 bascule d'infiabilité $1-R_B = 3 \quad 10^{-6}$
  . 1 porte "ou" d'infiabilité $1-R_p = 3 \quad 10^{-6}$
  . 2 horloges externes d'infiabilité $1 - R_H = 3,7 \quad 10^{-6}$

Connaissant l'infiabilité d'un calculateur $1 - R_C = 2,5\ 10^{-4}$, il est facile de trouver un majorant M de cette probabilité

$$M = 2 \times 2,5\ 10^{-4} \times 9,7\ 10^{-6} \text{ ≃ } 5\ 10^{-9}$$

Soit pour deux paires, une probabilité de 10^{-8} qui ne vient donc pas remett en cause le résultat précédent.



N.B.: sur cette figure l'echelle des infiabilités est logarithmique

Figure III.11

III-5.3. Principes d'écriture du logiciel

Les méthodes de comparaison logicielle permettent d'éviter l'utilisation de voteurs ou de comparateurs, dont la fiabilité limiterait celle du système tout entier. En revanche, le logiciel redondant (i.e. non nécessaire fonctionnellement) devient, en quelque sorte le seul "point dur" de la structure, et par conséquent devra être très fiable.
Dans l'exemple de la centrale anémométrique, le logiciel assurera également le filtrage des pannes transitoires.

Introduction

Un des intérêts de la technique de détection/reconfiguration utilisée dans cette conception réside dans le fait qu'elle laisse chaque paire responsabl de ses propres détections, reconfigurations, et destruction. Remarquons que toutes les informations nécessaires à la conduite de ces actions sont entièrement disponibles au niveau de chacune des paires.

Stratégie de reconfiguration au niveau d'une paire

Chaque paire peut se trouver dans quatre états distincts de fonctionnement

- Etat initial (EI) tous les constituants de la paire sont valides.
- Etat dégradé (EP) par suite de la défaillance réelle ou supposée d'u des capteurs propres à la paire, cette dernière travaille maintenant à l'aide du capteur partagé avec l'autre paire.
- Etat de "mort" (EM) consécutif à une défaillance attribuée à un des calculateurs de la paire. Cet état se traduit par l'abandon des fonctions assurées par cette paire et l'émission d'un signal de "mor vers l'autre paire.
- Etat "découplé" (ED) la paire assure seule toute la mission. Cet état correspond à l'état de mort de l'autre paire.

Les transitions entre ces états sont illustrées sur le diagramme de la figure III.12. Chacun des arcs de cette figure porte une brève indication de l'évènement produisant la transition correspondante.

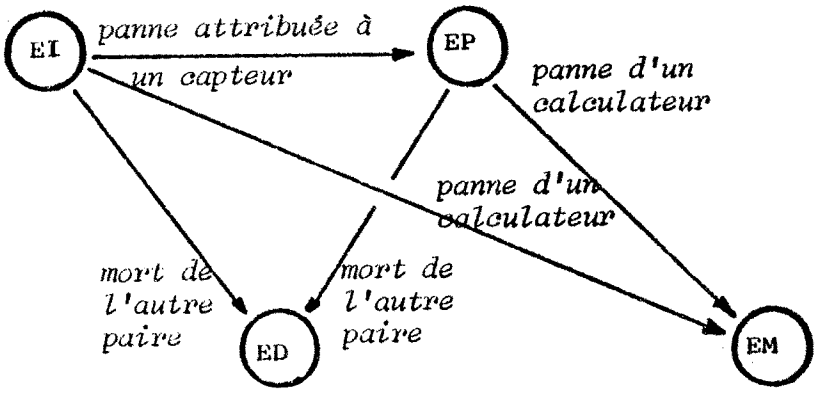


Figure III.12

Remarques

- L'état initial est état origine de deux transitions. Cette disposition est due au fait que, dans certains cas, les méthodes de détection utilisées ne pourront distinguer une erreur de capteur, d'une erreur de calculateur. Dans ces cas, la stratégie poursuivie consiste à faire l'hypothèse la moins préjudiciable à la vie de la paire : supposer que la panne détectée correspond à la défaillance d'un capteur. Lorsque cette supposition se révèle erronée; il s'agit en fait d'une panne de calculateur, celle-ci sera détectée ultérieurement et conduira à l'état E1.

- Le diagramme de la figure III.12 ne fait intervenir que des pannes considérées comme définitives. Les pannes transitoires feront l'objet d'un traitement particulier de la part du logiciel de détection.

- Les reconfigurations à entreprendre au niveau de chaque calculateur de la paire étant fonction de l'état de fonctionnement, cet état devra être conservé au niveau de chaque calculateur.

Décomposition du logiciel

Pour l'écriture du logiciel, nous adoptons une approche par "évitement de panne" en réservant l'utilisation de techniques de tolérances aux pannes pour une étape ultérieure.

Nous proposons pour cela une décomposition du logiciel en trois (ensembles de) modules ayant chacun une fonction précise :

. Le module Application comprend tout le logiciel destiné à assurer des fonctions du système, ainsi que les dispositifs de détection en ligne (dans la centrale : comparaison entre calculateurs, espionnage des sorties..).
Pour faciliter l'écriture de ce module, on peut penser à la réalisation préalable de procédures de services capables d'exécuter les fonctions élémentaires : comparaison , espionnage....

. Le module Analyse des défaillances qui sera chargé de centraliser toutes les informations concernant les défaillances détectées au sein du calculateur.
Suivant son propre diagnostic, il activera, soit à nouveau le module Application (défaillance bénigne ou transitoire), soit le module de reconfiguration, soit encore le "suicide" de la paire.

. Le module de reconfiguration. Ce module agit directement sur le module d'Application pour le reconfigurer selon les indications du module d'Analyse.
Cette interconnexion de modules est représentée sur la figure ci-après (Figure III.13).

Dans ce paragraphe, nous ne ferons qu'une rapide description des mécanismes à mettre en jeu pour le module d'Analyse.

Les modules d'application et de reconfiguration, qui sont fortement liés (ce dernier devant modifier la structure du premier), ont fait l'objet d'une étude séparée dont les résultats sont exposés en Annexe 3.

Nous dirons brièvement qu'à chaque état du graphe de la figure III.12 correspond un logiciel particulier. L'écriture globale de ce logiciel peut être facilitée par sa décomposition en modules utilisables dans plusieurs états; le passage d'un état à un autre se traduit alors par des modifications de liens entre les modules logiciels. La nature

de ces modifications est fonction du type de défaillance et de l'état de dégradation de la structure.

Nous rappelons que deux phases distinctes assurent toutes les détections :

- Emission de résultats vers l'extérieur. Le dispositif d'espionnage interne à la paire (Cf. II.3.1) permet la détection des pannes d'interface.
- Comparaison de résultats au sein de la paire : détection des pannes des calculateurs et des capteurs.

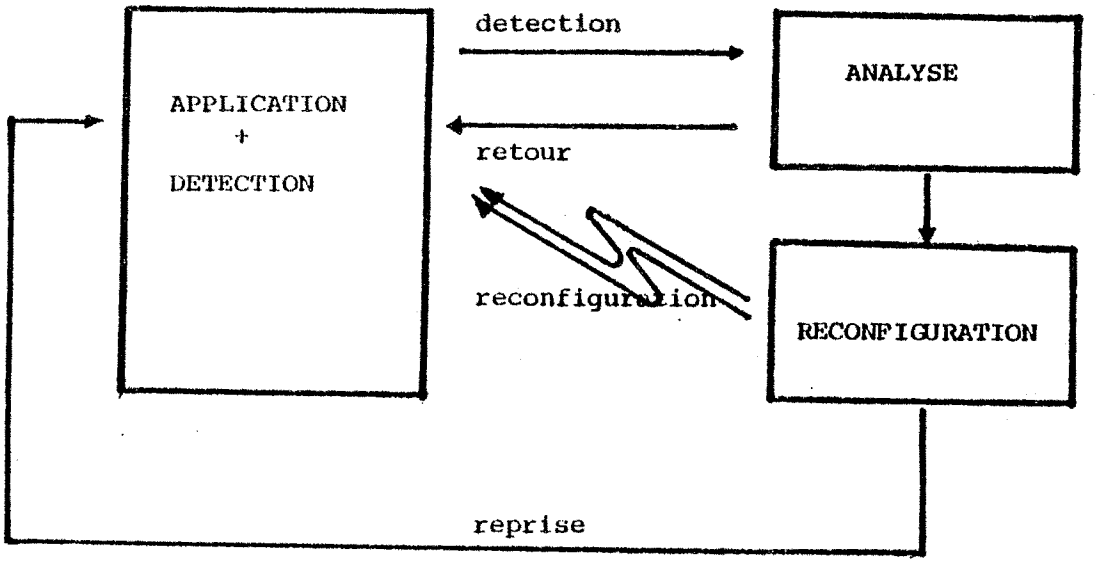


Figure III.13

On peut donc distinguer deux fonctions dans le module d'Analyse :

- Maintenir une variable indiquant les états de dégradation successifs de la structure selon le graphe de la figure II.12.
- Générer les signaux de pannes définitives pour faire évoluer cet état de dégradation.

Etant donnée la nature répétitive de l'application, nous proposons de ne considérer une défaillance comme définitive que lorsqu'elle a été détectée durant plusieurs cycles successifs.

Cette méthode de "filtrage" peut être mise en oeuvre en mémorisant dans chaque calculateur un "état de détection".

Les modifications de cet état de détection peuvent être décrites sous la forme d'un graphe d'état : la figure III.14.

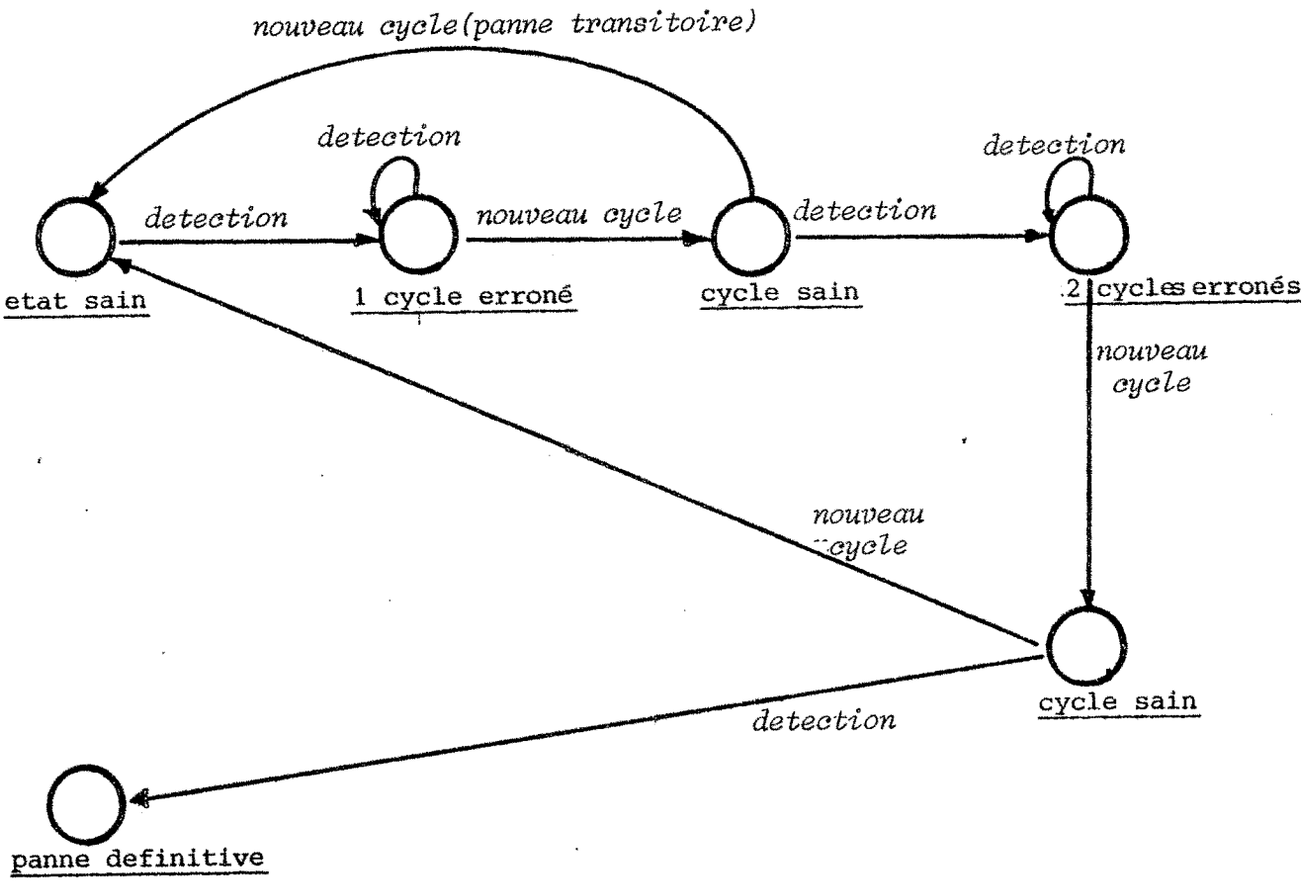


Figure III.14

Les transitions entre les états de graphe sont produites par les occurrences de deux types d'événements :

- détection : une erreur a été détectée; cet événement est généré par un des dispositifs de détection précédemment mentionné;
- nouveau cycle : cet événement est émis par le logiciel d'application à chaque début de cycle.

Remarques :

. Si l'on désire différencier plusieurs types de pannes, on pourra gérer, pour chaque panne, une variable d'état suivant un graphe analogue à celui de la figure III.14. Les transitions dans ce graphe seront déclanchées par les dispositifs assurant la détection des pannes du type considéré.

- Toutes les communications intercalculateurs sont assurées dans cette structure, par des échanges asynchrones. Des désynchronisations de tailles variables peuvent donc normalement apparaître (pannes transitoires, changement d'état de dégradation...). Mais des désynchronisations trop grandes mises en évidence par des "chiens de garde" sont des critères de défaillance qui doivent être pris en compte par les mécanismes de détection: Seules les désynchronisations entre les deux calculateurs d'une même paire déclencheront une reconfiguration par "suicide" de la paire. Les désynchronisations de longueur prohibitive entre les deux paires ne feront l'objet d'aucun traitement de reconfiguration directe, celui-ci n'étant déclenché que lorsqu'une des deux paires se détecte elle-même en panne.

Conclusion

Le logiciel jouant un rôle névralgique dans cette application, son écriture doit être entreprise de façon méthodique. Nous avons donné ici quelques éléments d'une telle méthode et nous avons défini quelques objets sur lesquels s'appuie cette méthode.

IV - CONCLUSION

Dans cette deuxième partie nous avons abordé le problème de la conception d'architectures spécialisées tolérant les pannes.
Pour cette conception nous proposons une approche en deux étapes :

- Recherche d'une architecture satisfaisant les contraintes fonctionnelles de l'application.
- Conception, à partir de cette dernière architecture, d'une structure susceptible de fournir les performances de sûreté de fonctionnement.

Nous nous sommes plus particulièrement intéressés à la deuxième partie pour laquelle nous présentons une esquisse de méthode (§I).

Cette méthode est limitée à des applications assurant une ou plusieurs missions réclamant une même fiabilité sans aucun mode dégradé prévu. En revanche, cette méthode conduira à une solution acceptable tant du point de vue fonctionnel que du point de vue sûreté de fonctionnement.

Cette méthode a été appliquée sur un exemple industriel (§III).
Dans cet exemple, nous employons un type d'architecture qui nous semble particulièrement intéressant : le réseau de microcalculateurs.

Pour ce type d'architecture nous avons développé une méthode permettant d'en accroître la fiabilité (§II).

Cette méthode est adaptée aux applications possédant un faible contexte (reprises faciles) et pour lesquelles seule une contrainte de fiabilité est exigée.
Cette méthode est donc limitée; mais, dans les cas où elle s'applique, elle fournit des résultats comparables à ceux de méthodes classiques plus onéreuses.
Ce résultat permet de penser que, pour d'autres problèmes caractérisés par d'autres types d'applications et d'autres contraintes de sûreté, des techniques analogues, fondées sur des réseaux de calculateurs pourront être développées.

ANNEXES :

ANNEXE 1. : Performances des systèmes à tolérance aux pannes assurée par le logiciel.

ANNEXE 2. : Comparaison entre les fiabilités de mission obtenues avec la stratégie proposée et celles obtenues à l'aide de stratégies tripliquées.

ANNEXE 3. : Dispositifs logiciels pour la tolérance aux pannes matérielles.

ANNEXE I

PERFORMANCES DES SYSTEMES A TOLERANCE AUX PANNES ASSUREE PAR LE LOGICIEL

But : Les méthodes logicielles de tolérance aux pannes sont basées sur des comparaisons mutuelles où sur des votes portant sur des résultats produits par les unités actives du système.
Le but de cette annexe est double :

- évaluation des performances de ces méthodes,
- influence de la fréquence de comparaisons ou votes sur ces performance

On remarque que deux effets vont ici jouer en sens contraire :

- des comparaisons trop fréquentes augmentent la durée d'une mission et produisent des pertes de puissance et de fiabilité de mission,
- des comparaisons trop rares accroissent le risque de pannes simultanées dans deux unités de la structure entre deux comparaisons.

Nous représenterons ces processus de comparaison par un couple de deux paramètres :

- θ : représentant la période des comparaisons (ou de vote),
- τ : désignant la durée de chaque comparaison ou vote.

Nous faisons donc deux hypothèses simplificatrices :

- régularité des comparaisons,
- constance du temps de comparaison.

Dans le cas où θ (resp. τ) ne serait pas constant, on pourrait utiliser en première approximation la valeur moyenne de cette grandeur .

Dans cette modélisation nous utilisons également une troisième hypothèse de nature différente :

On suppose que les échanges d'information entre unités sont suffisamment abondants pour exclure tous risques d'erreurs latentes (i.e. non détectées par la première comparaison suivant leur apparition).

Cette hypothèse est également implicite dans la plupart des travaux d'évaluation de stratégies assurées par le matériel.

Les seules causes d'échec de ces stratégies sont donc les occurrences de pannes simultanément dans plusieurs unités impliquées dans une même stratégie; ces occurrences étant comprises entre deux comparaisons successives.

Grandeurs étudiées

Pour définir des grandeurs caractéristiques de la sûreté de fonctionnement d'une structure à tolérance aux pannes assurée par le logiciel, il convient de tenir compte de deux particularités de ces dispositifs :

- Ces méthodes présentent un temps de réponse, après occurrence d'une erreur, plus long que les méthodes matérielles. Quelques résultats erronés pourrons être émis vers le milieu extérieur entre deux comparaisons, alors que la structure va détecter cette défaillance et se reconfigurer lors de la comparaison suivante.

- Contrairement aux méthodes matérielles, ces méthodes logicielles vont conduire, pour une même mission, à des durées différentes suivant les valeurs des paramètres τ et θ et des instants des occurrences des pannes éventuelles de la structure.
  Nous ne nous interesserons plus aux performances de sûreté pour un temps d'exposition aux risques fixé mais pour un travail fixé. Ce travail sera mesuré par un temps appelé temps de mission effectif (noté T_M).
  Nous définissons donc deux grandeurs que nous appelerons fiabilité et sécurité de mission pour un système à unités fonctionnelles possédant un taux de panne λ.

Définition 1

On appelle fiabilité de mission $R(\tau, \theta, \lambda, T_M)$ d'une structure à tolérance aux pannes assurée par logiciel, la probabilité d'effectuer, en un temps réel quelconque, une mission de temps de mission effectif T_M.

Définition 2

On appelle sécurité de mission $S(\tau, \theta, \lambda, T_M)$ d'une structure à tolérance aux pannes assurée par logiciel, la probabilité d'effectuer, en un temps réel quelconque, une mission de temps de mission effectif T_M ou de se placer dans un état de panne détectée.

Par abus de langage, et uniquement dans cette annexe, les fonctions R et S ci-dessus seront respectivement appelées simplement fiabilité et sécurité.

Modélisation - généralités

Tout au long de cette étude nous utilisons des modèles probabilistes appartenant à la famille des processus aléatoires à espace d'état et à base de temps discrets : les chaînes de Markov.
Les états de cette chaîne représenteront, comme pour la modélisation par processus de Markov [LAPRI75], les états dégradés de la structure. La discrétisation de la variable temps correspond à la périodicité des instants de détection, i.e de comparaison d'informations (figure A.1.1).

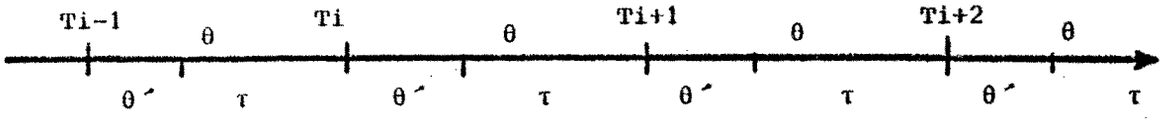


Figure A1.1

Cette technique sera explicitée dans le premier exemple traité puis directement appliquée pour les exemples suivants.

Résultats

Suivant une méthode analogue à celle suivie par [COURT 76], nous étudierons le comportement de quelques stratégies en fonction du taux de panne des unités, du temps de mission T_M et des caractéristiques du processus de comparaison.

I - SYSTEME DUPLEX

Le système duplex (schématisé sur la figure A1.2 est formé par une paire de calculateurs comparant périodiquement des résultats de leur calcul au moyen d'une liaison d'échange mutuel (par exemple d'interface d'entrée sortie à interface d'entrée/sortie).

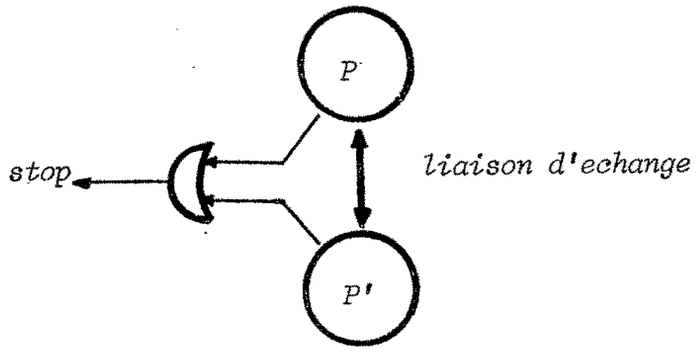


Figure A1.2

Le signal stop est simplement formé de l'union des signaux de détection de discordance émis par chaque processeur. On peut éventuellement prévoir la mémorisation de ce signal dans une bascule pour prévenir une disparition consécutive à l'occurrence d'une seconde panne postérieurement à la détection de la première défaillance.
Le système peut donc se trouver dans trois états distincts :

ETAT 1 : Etat de fonctionnement correct du système.

ETAT 2 : Une panne a été détectée, le système se place dans un état d'arrêt signalé par la monté du signal stop.

ETAT 3 : Les deux unités ont subies chacune une défaillance entre deux instants de comparaison.
Nous sommes alors conduit à adopter l'hypothèse la plus préjudiciable à la sûreté du système et donc de supposer que le système a échappé à tout contrôle.

Le diagramme de changement d'état est donné par le schéma de la figure A1.3. Les probabilités (conditionnelles) de passage entre les divers états (resp. 1 → 2 1 → 3) sont constants en fonction de l'hypothèse du taux de panne constant, retenu pour chaque unité.

Si p est la probabilité de non défaillance d'une unité pendant un intervalle θ, ces probabilités de passage sont respectivement égales à $2p(1-p)$ et $(1-p)^2$.

On est donc conduit à la formule de récurrence (1). Le vecteur P_n à, pour composantes, les probabilités de présence à l'instant t_n ($t_n = t_o + n.\theta$; t_o instant initial) dans les trois états précédemment définis.

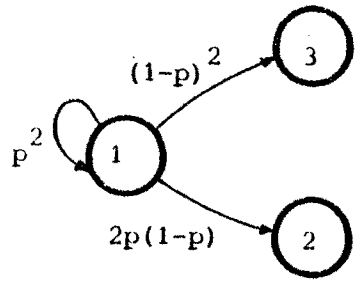


Figure A1.3

$$P^n = \begin{bmatrix} P_1^n \\ P_2^n \\ P_3^n \end{bmatrix} = \begin{bmatrix} p^2 & 0 & 0 \\ 2(1-p)p & 1 & 0 \\ (1-p)^2 & 0 & 1 \end{bmatrix} P^{n-1} \qquad (1)$$

avec P_i^n égal à la probabilité de présence dans l'état E_i après la $n^{ième}$ comparaison.

Par intégration du système (1) à partir de l'état initial $P_o = \begin{bmatrix} 1 \\ o \\ o \end{bmatrix}$ il vient

$$P_1 = p^{2n}$$

$$P_2 = 2p(1-p)\frac{1-p^{2n}}{1-p^2}$$

$$P_3 = (1-p)^2 \frac{1-p^{2n}}{1-p^2}$$

D'où l'estimation de la fiabilité R et de la sécurité S(au sens défini précédemment) d'une mission comportant n comparaisons :

$$R(n) = P_1 = p^{2n}$$

$$S(n) = P_1 + P_2 = 1-P_3 = 1-(1-p)^2 \frac{1-p^{2n}}{1-p^2} = 1- \frac{1-p}{1+p} [1-R]$$

Si l'on considère des unités présentant un taux de panne λ, p (probabilité de non panne d'une unité pendant une durée θ) peut s'écrire $p = e^{-\lambda\theta}$. De même n peut s'exprimer en fonction de θ, τ et du temps de mission effectif T_M

$$n = \frac{T_M}{\theta'} = \frac{T_M}{\theta-\tau}$$

Il vient alors :

$$R(\tau,\theta,\lambda,T_M) = e^{-2\lambda T_M \frac{\theta}{\theta-\tau}} \qquad (2)$$

$$S(\tau,\theta,\lambda,T_M) = 1 - \frac{1-e^{-\lambda\theta}}{1+e^{-\lambda\theta}} (1-R) \qquad (3)$$

Il est facile de montrer que la fiabilité R et la sécurité S sont respectivement des fonctions strictement croissante et décroissante de θ (Figure A1.4) .

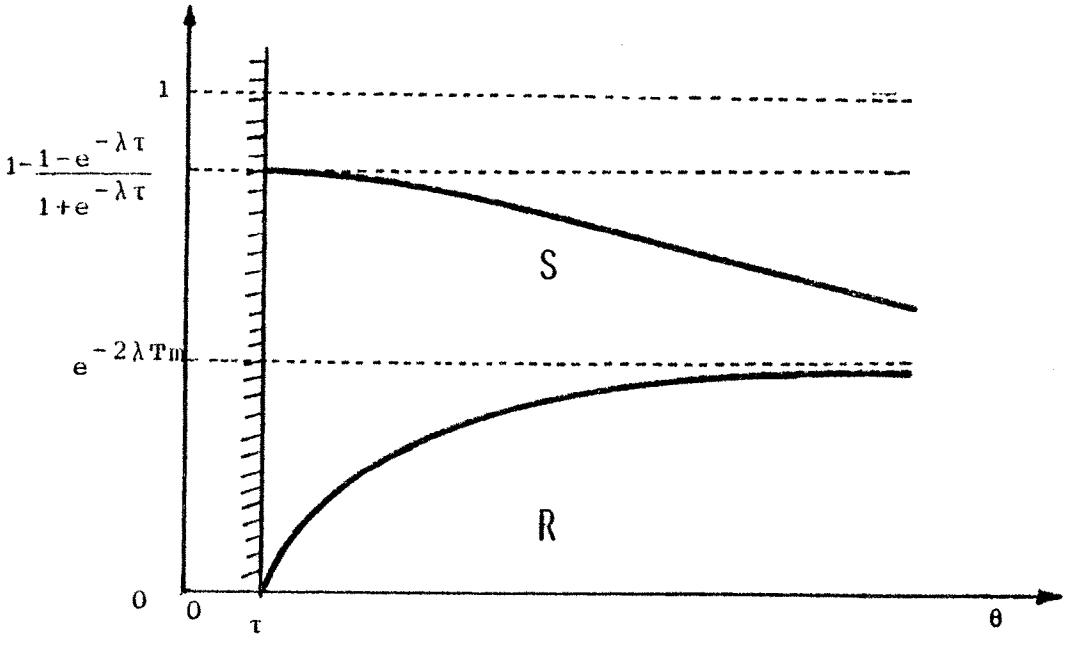


Figure A1.4

Remarques

Pour un domaine de valeurs réalistes :

- λ compris entre 10^{-3} et 10^{-5} pannes par heures
- T_M compris entre 100 et 10^4 heures
- τ compris entre 10 µs et 1 µs ($3{,}10^{-8}$ et $3{,}10^{-6}$ heures)
- θ compris entre 50 µs et 200 µs ($1{,}5.10^{-4}$ et $6{,}10^{-4}$ heures)

$\frac{\tau}{\theta}$ et $\lambda\theta$ apparaissent petits par rapport à l'unité :

$5.10^{-5} < \frac{\tau}{\theta} < 2.10^{-2}$ et $1{,}5.10^{-9} < \lambda\ \theta < 6.10^{-7}$

On remarque alors :

1. La faiblesse de la perte de fiabilité pour les courtes durées de mission effectives.

$$K = \frac{e^{-2\lambda T_M}}{R} = e^{\frac{+2\lambda\tau T_M}{\theta - \tau}}$$

$$K \leq e^{2\ 10^{-3}\ x\ 3\ 10^{-6}\ x\ T_M/1.50^{-4}} = e^{4.10^{-5}\ T_M}$$

2. La dépendance linéaire de S vis à vis de θ.

Démonstration informelle :

On cherche une approximation de S en considérant $\lambda\theta$ et $\frac{\tau}{\theta}$ comme infiniment petits.

$$1 - S = \frac{1 - e^{-\lambda\theta}}{1 + e^{-\lambda\theta}} \; [1 - R]$$

$$1 - S = [\frac{\lambda\theta}{2} - \frac{\lambda^3\theta^3}{8}] \; [1 - e^{-2\lambda T_M} \; [1 - 2\lambda T_M \; \frac{\tau}{\theta}]]$$

$$1 - S = 2\lambda^2 T_M \tau \; e^{-2\lambda T_M} + \frac{\lambda\theta}{2} \; [1 - e^{-2\lambda T_M}] - \frac{\lambda^4\theta^2}{4} \tau \, TM \, e^{-2\lambda T_M} - \frac{\lambda^3\theta^3}{8} \; [1 - e^{-2\lambda T_M}].$$

La faiblesse du coefficient de θ^2 dans la formule précédente traduit ce caractère linéaire. ($\frac{\lambda^4 \tau \, T_M}{4} \; e^{-2\lambda T_M} \leqslant 7{,}5 \; 10^{-15}$)

Comparaison avec un système matériel :

Considérons un système duplex matériel [COURT76]. Ce système est représenté par la figure A1.5.

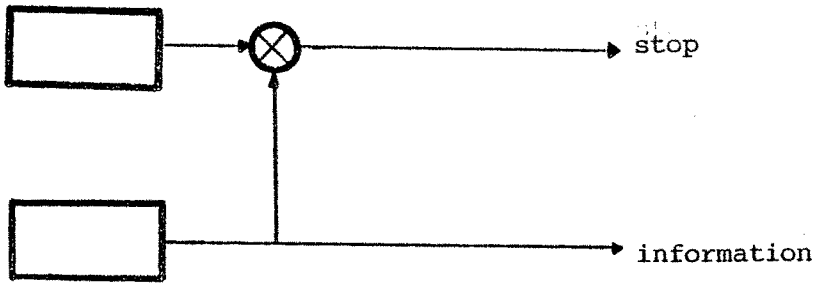


Figure A1.5

Si l'on appelle respectivement C et R les fiabilités du comparateur et d'une unité fonctionnelle, on obtient l'expression de la fiabilité et de la sécurité :

$$R = CR^2 \qquad (1)$$

$$S = C + (1-C)R \qquad (2)$$

On en déduit, par élimination de C entre (1) et (2), une relation (3) entre R, S et R.

$$\frac{R}{R^2} = \frac{S - R}{1 - R} \qquad (3)$$

Inversement, lorsque (3) est satisfaite par les valeurs R et S produites par un duplex à comparaison logicielle, celui-ci est alors équivalent à un duplex à comparaison matérielle. On peut associer à la stratégie logicielle un comparateur matériel équivalent. Le calcul suivant donne une estimation de la fiabilité de ce comparateur.

Si l'on pose $C' = e^{-2\lambda\tau}\frac{T_M}{\theta-\tau}$, $p = e^{-\lambda\theta}$, $R = e^{-2\lambda\frac{T_M\theta}{\theta\ \tau}}$, $R=r=e^{-\lambda T_M}$

l'équation (3) (qui admet toujours une solution unique) se transforme en :

$$C' = 1 - \frac{1-p}{1+p}\ [\frac{1-R^2C'}{1-R}] \qquad (4)$$

L'équation (4) peut être résolue par approximation :

Hypothèses

a) On suppose $\lambda\theta$ petit : $p \approx 1- \lambda\theta$
et donc $\frac{1-p}{1+p} \approx \frac{\lambda\theta}{2}$

b) On suppose $\frac{2\lambda\tau T_M}{\theta - \tau}$ petit et par suite $C' \approx 1 - \frac{2\lambda\tau T_M}{\theta - \tau}$

c) On suppose θ grand par rapport à τ et donc $C' \approx 1 - \frac{2\lambda\tau T_M}{\theta}$

L'équation (4) devient :

$$\frac{2\lambda\tau T_M}{\theta} = \frac{\lambda\theta}{2}\ [\frac{1-r^2}{1-r}(1- \frac{2\lambda\tau T_M}{\theta})] \quad (5)$$

Soit encore :

$$\frac{2\tau T_M}{\theta} = \frac{\theta}{2}\ [1 + r + \frac{r^2}{1-r}\ \frac{2\lambda\tau T_M}{\theta}] \qquad (6)$$

d'où l'équation équivalente ;

$\theta^2[\frac{1+r}{2}] + \theta[\frac{r^2\ \lambda\tau T_M}{1-r}] - 2\tau T_M = 0$; soit en prenant pour θ la racine positive :

$$\theta = \frac{-\frac{r^2\lambda\tau T_M}{1-r} + \sqrt{\frac{4\ \lambda^2\tau^2 T_M^2}{(1-r)^2} + 4(1+r)\ \tau T_M}}{1 + r} \qquad (7)$$

En raison de la faiblesse de τ et de λ cette formule (7) peut être simplifiée en :

$$\theta \approx 2\sqrt{\frac{\tau T_M}{1+r}} \quad \text{et} \quad C \approx e^{-2\lambda\sqrt{\tau T_M(1+r)}}$$

car $\lambda T_M r^2/1-r$ décroit strictement de 1 à 0 lorsque λT_M varie de 0 à $+\infty$.

Exemple :

$\lambda = 10^{-4}$ p/h, $T_M = 3\ 10^{3}$h, $\tau = 3\ 10^{-8}$ h,

$r = e^{-3\ 10^{-1}} = 7,\ 408\ 10^{-1}$, $1+r = 1,7408$

C=0,99999875

Le comparateur matériel équivalent a un taux de pannes égal dans cet exemple à $8,34.10^{-10}$.

La comparaison logicielle introduit donc un équivalent de comparateur matériel bien plus fiable que tous les circuits de comparaison que l'on peut construire à l'heure actuelle.

Remarques

- L'architecture duplex peut servir de "bloc" de base pour des architectures supportant des stratégies par détection/remplacement. La sécurité du duplex qui correspondra à la "couverture" de détection rejaillira sur la fiabilité de ces architectures comme nous le verrons lors du chapitre consacré à la stratégie duplex-réserve.

- Si le concepteur a choisi, dans un premier temps, une architecture duplex à comparaison logicielle pour assurer une mission, il devra dans un second temps calibrer la période θ des comparaisons pour atteindre les performances souhaitées.

  Exemple : Supposons une mission
  - de T_M heures (temps effectif),
  - devant être accomplie moins de T_{MAX} heures ($T_M < T_{MAX}$)
  - de temps de comparaison égal à τ heures,
  - devant être assurée avec une fiabilité supérieure à R_{MIN} et une sécurité supérieure à S_{MIN}.

  Le concepteur devra alors choisir un θ (s'il en existe) compatible avec le système d'inéquations suivant :

$$(1)\ \theta > \frac{T_M\,\tau}{T_{MAX}-T_M} + \tau\ ,\ \theta > \frac{\tau T_{MAX}}{T_{MAX}-T_M}$$

$$(2)\ \theta > \frac{\tau \operatorname{Log} R_{MIN}}{\operatorname{Log} R_{MIN} + 2T_M\lambda}$$

$$(3)\ \theta < \theta_{MAX} \text{ avec } S_{MIN} = 1 - \frac{1-e^{-\lambda\theta_{MAX}}}{1+e^{-\lambda\theta_{MAX}}}\left[1 - R(\theta_{MAX})\right]$$

II - STRATEGIE DUPLEX RESERVE

Description : La stratégie Duplex-Réserve est appliquée à un système constitué de trois unités (Figure A1.6). Deux de ces unités débutent la mission en mode duplex jusqu'à la détection d'une première panne. Cette détection provoque alors l'arrêt du duplex et la poursuite de la mission sur l'unité en réserve.

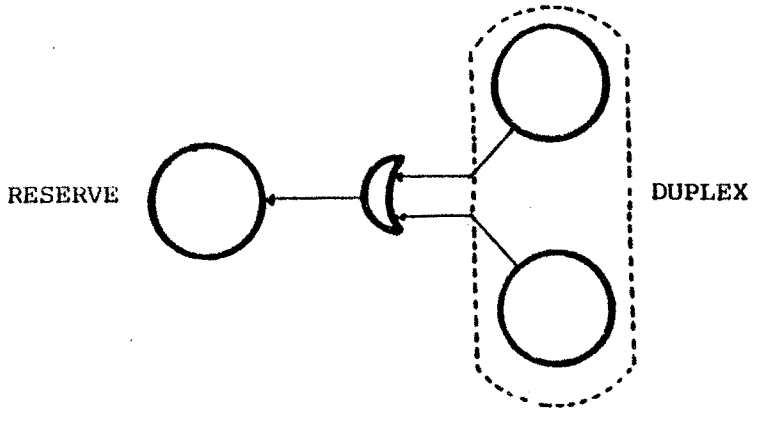


Figure A1.6

Evaluation : Contrairement à la stratégie précédente, cette stratégie prévoit un fonctionnement correct sans aucune comparaison d'information, lorsque la mission est assurée par l'unité en réserve. Nous considérons que cette réserve évolue alors en "temps continu" et nous ne modéliserons pas cette stratégie par une chaîne de Markov.
L'architecture de la Figure A1.6 étant démunie de tout signal d'arrêt nous n'avons à étudier que la fiabilité de ce système. Cette fiabilité R peut être évaluée de la façon suivante :

On peut poser : $R = P_1 + P_2$ (1)

avec :

P_1 = probabilité que le duplex marche pendant $\frac{T_M}{\theta-\tau}\,\theta$ unités de temps

P_2 = probabilité d'avoir une panne dans le duplex, que cette panne soit bien détectée, que le basculement sur la réserve soit correct et que cette réserve puisse achever la mission.

Remarques

- Conformément aux hypothèses exprimées dans le préambule nous considérons que cette reconfiguration est toujours réussie lorsque deux pannes ne surviennent pas dans les deux unités opérant en duplex durant une même période séparant deux comparaisons.
- La réserve devra être susceptible de reprendre la mission à tout instant. Il sera donc nécessaire de prévoir des mécanismes d'échange d'information entre le duplex et la réserve.

D'après l'étude précédente il vient :

$$P_1 = p^{2n} = e^{-2\lambda\left[T_M \frac{\theta}{\theta-\tau}\right]} = e^{-2\lambda\ T_M}\ e^{-2\lambda\left[T_M \frac{\tau}{\theta-\tau}\right]} \qquad (2)$$

Si n désigne le nombre de cycles de comparaison exigé par la mission :

$n = [\frac{T_M}{\theta-\tau}]$ ou [A] désigne le premier entier supérieur ou égal à A.

La probabilité P_2 peut se décomposer suivant le cycle d'occurrence de la panne. Il vient

$$P_2 = \sum_{k=1}^{k=n} \text{probabilité que} \left\{ \begin{array}{l} \text{- la panne arrive au } k^{\text{ième}} \text{ cycle et la} \\ \text{ reconfiguration est correcte} \\ \text{- la réserve achève la mission} \end{array} \right\}$$

La probabilité sous le signe Σ peut être mise sous la forme d'un produit en remarquant l'indépendance entre les événements qu'elle met en jeu. Il vient donc :

$$P_2 = \sum_{k=1}^{k=n} P(k).\ P'(k)$$

avec

$$P(k) = \text{Probabilité} \left\{ \begin{array}{l} \text{que la panne arrive au } k^{\text{ième}} \\ \text{cycle et que la reconfiguration} \\ \text{soit correcte} \end{array} \right\} = p^{2(k-1)}\ 2p(1-p)$$

$P'(k)$ = Probabilité {la mission soit achevée par la réserve}

Si la panne arrive au $k^{\text{ième}}$ cycle, k cycles ont déjà été correctement exécutés, il en reste donc L = n-k+1 à exécuter.

La mission ne sera donc accomplie que si la réserve reste en état de marche pendant une durée $T_M+(k-1)\tau+\theta$ comptée à partir de l'instant initial t_0, soit :

$$P'(k) = e^{-\lambda(T_M+(k-1)\tau+\theta)} .$$

D'où la formule

$$P_2 = \sum_{k=1}^{k=n} e^{-2\lambda(k-1)\theta}\ 2\, e^{-\lambda\theta}(1-e^{-\lambda\theta})\ e^{-\lambda(T_M+(k-1)\tau+\theta)}$$

soit

$$P_2 = 2e^{-2\lambda\theta}(1-e^{-\lambda\theta})\ e^{-\lambda T_M} \sum_{\ell=0}^{\ell=n-1} e^{-\ell(2\lambda\theta+\lambda\tau)}$$

et par suite

$$P_2 = 2e^{-2\lambda\theta}(1-e^{-\lambda\theta})\ e^{-\lambda T_M}\ \frac{1-e^{-\lambda(2\theta+\tau)n}}{1-e^{-\lambda(2\theta+\tau)}} .$$

En reportant dans la formule (1) après avoir remplacé n par sa valeur et posé $r = e^{-\lambda T_M}$ il vient :

$$R = \frac{2e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-\lambda(2\theta+\tau)}}\ r\left(1-r^{\frac{3\tau}{\theta-\tau}+2}\right)+ r^{2+\frac{2\tau}{\theta-\tau}} .$$

Il est évident que R ne pourra atteindre la fiabilité théorique R_D procurée par la stratégie Duplex-réserve

$$R_D = r(1-r^2)+r^2 .$$

Nous allons montrer que R admet un maximum en θ pour des valeurs ordinaires de λ, τ, T_M (voir le domaine de valeurs proposé au paragraphe consacré à la stratégie duplex) et que ce maximum est proche de cette valeur théorique.

Existence et localisation du maximum

Examinons P_2:

$$P_2 = \frac{2\, e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-2\lambda\theta}\, e^{-\lambda\tau}}\; r\left(1- r^{\frac{3\tau}{\theta-\tau}+2}\right) = f.g$$

avec $f = \frac{2e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-2\lambda\theta}e^{-\lambda\tau}}$ et $g = r\,(1-r^{\frac{3\tau}{\theta-\tau}+2})$

Posons $e^{-\lambda\theta} = x$ et $e^{-\lambda\tau} = a$ il vient $f(x) = \frac{2x^2(1-x)}{1-x^2a}$

et par dérivation :

$$\frac{df}{d\theta} = \frac{df}{dx}\,\frac{dx}{d\theta} = -\frac{(1-x^2a)(4x-6x^2)+2ax(2x^2-2x^3)}{(1-x^2a^2)}\,\lambda x = -\lambda x^2\,\frac{2a\,x^3-6x+4}{(1-x^2a)^2}$$

Posons $Q(x) = 2ax^3 - 6x + 4$. Etudions $Q(x)$ pour $a \geq x \geq 0$ $(\theta \in [\tau+\infty[)$

Etudions $P(a) = 2a^4-6a+4$

$\frac{dP(a)}{da} = 8\,a^3 - 6$; $\frac{dP(a)}{da} = 0$ et $a \geq 0 \Rightarrow a = \sqrt[3]{\frac{3}{4}}$

d'où le tableau suivant :

a	0	a_o	$\sqrt[3]{3/4}$	1
P(a)	4 ↘	0 ↘		↗ 0
$\frac{dP(a)}{da}$	-6	−	0	↗ 2 +

Les valeurs relatives de τ et de λ donnent une borne inférieure à a, a_{min}

$$a_{min} = e^{-10^{-3}} \times 310^{-7} \approx 1 - 310^{-10}$$

donc $a_{min} > \sqrt[3]{3/4}$ et par suite p(a) est toujours négatif pour toutes valeurs de a sur le domaine considéré.

On a $Q(0) = 4$ $Q(x)$ admet donc (au moins) une racine pour $\theta \in [\tau + \infty[$

$$\frac{d\,Q(x)}{dx} = 6\,ax^2 - 6 \qquad \frac{d\,Q(x)}{dx} = 0 \quad = \quad x = \pm\sqrt{\frac{I}{a}}$$

or $a < 1$ ce qui implique $\sqrt{\frac{I}{a}} > 1$ et par suite $\frac{d\,Q(x)}{dx} < 0$ pour $\theta \in [\tau + \infty[$

d'où le tableau suivant :

x	0	x_o	a
$\frac{d\,Q(x)}{dx}$		—	
$Q(x)$	4	0	(a)

d'où les variations de $f(\theta)$:

θ	τ	θ_o	$+\infty$
x	a	x_o	0
$Q(x)$	$p(a)$ —	0 +	4
$\frac{df}{d\theta}$	$\frac{\lambda a^2 p(a)}{(1-a^3)^2} > o$ +	0 —	0
f	$\frac{2a^2}{1+a+a^2} > o$		0

On peut facilement donner une estimation de θ_o en supposant $\lambda\theta_o$ petit devant l'unité.

$$a \overset{\sim}{\sim} 1 - \lambda\tau + \frac{\lambda^2\tau^2}{2}$$

$$x \overset{\sim}{\sim} 1 - \lambda\theta_o + \frac{\lambda^2\theta_o^2}{2} \quad \text{et} \quad x^3 \overset{\sim}{\sim} 1 - 3\lambda\theta_o + \frac{9\lambda^2\theta_o^2}{2}$$

L'équation $Q(x) = 0$ devient :

$$2(1-3\lambda\theta o + \frac{9\lambda^2 \theta o^2}{2})\ (1-\lambda\tau+\frac{\lambda^2\tau^2}{2}) - 6(1-\lambda\theta o + \frac{\lambda^2 \theta o^2}{2}) +4 = 0$$

Soit (en simplifiant par λ)

$$\theta o^2\ [6\lambda - 9\lambda^2\tau + \frac{9}{2}\lambda^3\tau^2] + \theta_o\ [6\lambda\tau - 3\lambda^2\tau^2] - 2\tau + \lambda\tau^2 = 0$$

La petitesse de λ devant l'unité conduit à l'équation

$$\theta_o^2\ [6\lambda] + \theta_o\ [6\lambda\tau] - 2\tau = 0$$

d'où la solution approchée :

$$\theta_o \approx \frac{3\lambda\tau + \sqrt{9\lambda^2\tau^2 + 12\lambda\tau}}{6\lambda}$$

$$\theta_o \approx \frac{1}{2}\tau + \sqrt{\frac{\tau^2}{4} + \frac{\tau}{3\lambda}}$$

Soit vu le domaine des valeurs de τ :

$$\theta_o \approx \sqrt{\frac{\tau}{3\lambda}}$$

- Nous pouvons estimer la valeur maximale de f en remarquant qu'autour de θ_o, $\lambda\theta_o$ reste négligeable devant l'unité et que τ est négligeable devant θ.

$$f(\theta_o) \approx \frac{2(1-2\lambda\theta)(\lambda\theta)}{1-(1-2\lambda\theta)(1-\lambda\tau)} \approx \frac{\theta}{2\theta+\tau}\ 2(1-2\lambda\theta)$$

$$f(\theta_o) \approx (1-\frac{\tau}{2\theta})(1-2\lambda\theta) \approx (1-2\lambda\theta - \frac{\tau}{2\theta})$$

$f(\theta_o)$ est donc proche de l'unité.

- Pour ces valeurs de θ fortes par rapport à τ, les fonctions $r^{\frac{3\tau}{\theta-\tau}+2}$ et $r^{\frac{2\tau}{\theta-\tau}+2}$ sont pratiquement égales à leur valeur maximale limite r^2. De plus, on a:

$$\lim_{\theta\to\tau} R = \frac{2}{3} e^{\lambda T_M} = \frac{2}{3} r$$

$$\lim_{\theta\to\infty} R = e^{-2\lambda T_M} = r^2$$

Or le majorant théorique de R, R_{DR}, qui est donc pratiquement atteint est largement supérieur à $\frac{2}{3} r$ et à r^2.

1) $R_{DR} - \frac{2}{3} r = r - r^3 + r^2 - \frac{2}{3} r$

$R_{DR} - \frac{2}{3} r > R_{DR} - r = r^2 - r^3 = r(1-r)$

Or pour r compris entre 0 et 1 on a évidemment $r(1-r) \geq 0$.

2) $R_{DR} - r^2 = r - r^3 = r\,[1-r^2]$

qui est toujours positif.

Ce qui prouve l'existence d'un maximum de R et que ce maximum est atteint pour des valeurs de θ telles que $\lambda\theta$ reste faible devant l'unité, et θ soit fort devant τ.

Estimation de la valeur $\hat{\theta}$ maximisant R

Ce calcul approché est rendu possible par la localisation de $\hat{\theta}$ effectué précédemment. Il est basé sur l'existence de deux "infiniment petits" $\lambda\theta$ et $\frac{\tau}{\theta}$ lorsque θ est situé au voisinage de $\hat{\theta}$.
Ces "deux infiniment petits" sont utilisés pour l'obtention d'une approximation de $R_{DR} - R$:

$R - R_{DR} = r(1-r^2) + r^2 - fg - P_2 = \Delta$;

grâce aux estimations suivantes :

$f(\theta) = (1-2\lambda\theta - \frac{\tau}{2\lambda\theta})$; $g(\theta) = r\,(1-r^2(1-3\lambda T_M \frac{\tau}{\theta}))$;

$P_2 = r^2(1-2\lambda T_M \frac{\tau}{\theta})$;

$\Delta = r(1-r^2) + r^2 - (1-2\lambda\theta - \frac{\tau}{2\lambda\theta})\, r(1-r^2(1-3\lambda T_M \frac{\tau}{\theta})) - r^2(1-2\lambda T_M \frac{\tau}{\theta})$;

$\Delta = (2\lambda\theta + \frac{\tau}{2\theta})\, r(1-r^2(1-3\lambda T_M \frac{\tau}{\theta})) - r^3\, 3\lambda T_M \frac{\tau}{\theta} + r^2(2\lambda T_M \frac{\tau}{\theta})$;

$\Delta = r(2\lambda\theta + \frac{\tau}{2\theta}) + r^2(2\lambda T_M \frac{\tau}{\theta}) - r^3\,[2\lambda\theta + \frac{\tau}{2\theta} + 3\lambda T_M \frac{\tau}{\theta} - 6\lambda^2 T_M \tau - \frac{3}{2}\lambda T_M \frac{\tau^2}{\theta^2}]$;

Si l'on néglige le terme $+r^3 \frac{3}{2} \lambda T_M \frac{\tau^2}{\theta^2}$ (en raison des résultats déjà obtenus) il vient :

$\Delta = r(2\lambda\theta + \frac{\tau}{2\theta}) + r^2(2\lambda T_M \frac{\tau}{\theta}) - r^3[2\lambda\theta + \frac{\tau}{2\theta} + 3\lambda T_M \frac{\tau}{\theta} - 6\lambda^2 T_M \tau]$

Δ est de la forme : $\Delta = a\,\theta + b + \frac{c}{\theta}$

avec : $a = 2\lambda r - 2\lambda r^3$

$b = +6\lambda^2 T_{M^\tau} r^3$

$$c = \frac{r_\tau}{2} + 2r^2\lambda T_{M^\tau} - \frac{r^3_\tau}{2} - 3r^3\lambda T_{M^\tau}$$

Vérifions que c est toujours positif !

Posons $\lambda T_M = x$ et $C = \frac{r_\tau}{2} P(x)$

Il vient $r = e^{-x}$ et

$$P(x) = 1-e^{-2x} + 2xe^{-x}(2-3e^{-x})$$

1) Si $e^{-x} \leqslant \frac{2}{3}$ il est évident que $P(x) > 0$

2) $e^{-x} \geqslant \frac{2}{3}$; $x < -\text{Log}\,\frac{2}{3} \simeq 0{,}405$

par dérivation il vient :

$$P'(x) = e^{-x}(-4x+4) - e^{-2x}(4-12x)$$

Posons $K(x) = 4\, e^{-x} P'(x)$.

Notons que $K(x)$ a le même signe que $P'(x)$.

Il vient :

$$K(x) = (1-x) - e^{-x}(1-3x)$$

en dérivant :

$$K'(x) = -1 - e^{-x}(-1+3x-3) = -1-e^{-x}(3x-4)$$

dérivons encore

$$K''(x) = -e^{-x}(-3x+7)$$

$0 < x < 0{,}405 \Rightarrow x < \frac{7}{3} \Rightarrow K''(x) < 0 \;\forall\; x \in [0, -\text{Log}\,\frac{2}{3}]$

D'où le tableau suivant :

x	$\alpha = -\log(2/3)$
e^{-x}	1 ↘ 2/3
K"(x)	-7 —
K'(x)	3 ↘ $-1-\frac{2}{3}(3\alpha-4)$
K(x)	0 + ↗
P(x)	0 + ↗

C reste donc toujours positif.

Revenons à Δ :

$$\Delta = a\theta + b + \frac{c}{\theta} \quad a, b, c > o$$

par dérivation il vient :

$$\frac{\partial \Delta}{\partial \theta} = a - \frac{c}{\theta 2} \quad \text{une seule racine positive } \theta_o = \sqrt{\frac{c}{a}}$$

pour cette valeur $\Delta(\theta_o) = 2\sqrt{ca} + b$ (remarquons $\Delta(\theta_o) > o$)

D'où le tableau récapitulatif :

θ	τ	$\sqrt{c/a}$	$+\infty$
$\frac{\partial \Delta}{\partial \theta}$	-	0	+
Δ	↘	$\Delta(\theta_0) = 2\sqrt{ca}+b$	↗ a

Cette formule n'est valide que pour les valeurs $\frac{\tau}{\theta}$ faibles.
Δ peut être représenté sur le graphe suivant (Figure A1.7).

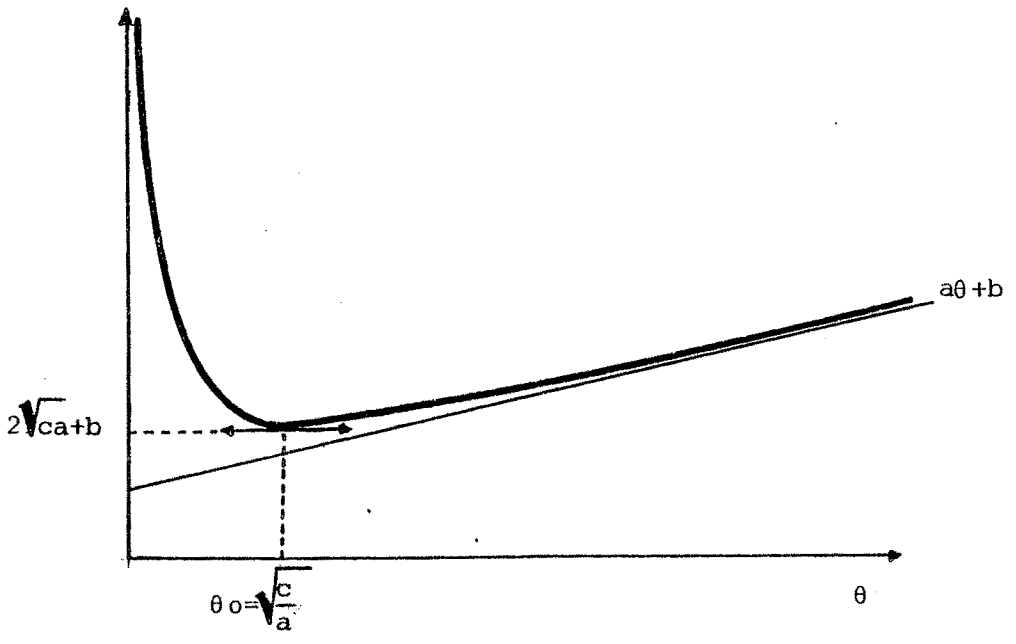


Figure A1.7

D'où l'expression de θ_o :

$$\theta_o = \sqrt{\frac{c}{a}} = \sqrt{\frac{\tau}{4\lambda}} \quad \sqrt{\frac{1 + 4\lambda T_M r - r^2 - 6r^2 T_M \lambda}{1 - r^2}} = \sqrt{\frac{\tau}{4\lambda}} \quad \sqrt{1 + \frac{2T_M \lambda r(2-3r)}{1 - r^2}}$$

Soit

$$\Delta(\theta_o) = 2\sqrt{ac} + b = 2r\sqrt{\lambda\tau}\sqrt{(1-r^2)\ [(1-r^2)+r\lambda T_M(4-6r)\ ]} + 6\lambda^2 T_M \tau r^3$$

Cas particulier des λT_M petits :

$$r = 1 - \lambda T_M$$

$$\theta_o \approx \sqrt{\frac{\tau}{4\lambda}} \sqrt{1 + 2\lambda T_M \frac{(1-\lambda T_M)(3\lambda T_M - 1)}{2\lambda T_M}}$$

$$\theta_o \approx \sqrt{\frac{\tau}{4\lambda}} \sqrt{4\lambda T_M}$$

$$\theta_o \approx \sqrt{\tau T_M}$$

et

$$\Delta(\theta_o) \approx 2(1-\lambda T_M)\sqrt{\lambda\tau}\sqrt{2\lambda T_M[2\lambda T_M + (1-\lambda T_M)\lambda T_M(6\lambda T_M - 2)] + 6\lambda^2 T_M\ (1-3\lambda T_M)}$$

$$\Delta(\theta_o) \approx 2(1-\lambda T_M)\sqrt{\lambda\tau}\sqrt{16\lambda^3 T_M^3} + 6\lambda^2 T_M \tau\ (1-3\lambda T_M)$$

$$\Delta(\theta_O) \underset{\sim}{\sim} 8\sqrt{\lambda^2\tau T_M}\,\lambda T_M + 6\lambda^2 T_M\tau = 2\lambda^2 T_M(4\sqrt{\tau T_M} + \tau)$$

en relatif

$$\frac{\Delta(\theta_O)}{1 - R_D} = \frac{8\lambda^2 T_M\sqrt{\tau T_M} + 12\lambda^2 T_M\tau}{2\lambda^2 T_M^2} = \frac{4\sqrt{\tau T_M} + \tau}{T_M}$$

si τ est faible

$$\frac{\Delta(\theta_O)}{1 - R_D} \sim \frac{4\sqrt{\tau}}{\sqrt{T_M}}$$

Exemple : $T_M = 10h, \tau = 10^{-7}$

Erreur relative $8\ 10^{-4}$.

Mais cette erreur relative reste faible même si θ varie fortement autour du maximum θ_O. En effet, posons $= \alpha\theta_O$
il vient $\Delta = (\alpha + \frac{1}{\alpha})\sqrt{ac} + b$

Soit encore

$$\frac{\Delta(\alpha)}{1-R_D} = \frac{(\alpha + \frac{1}{\alpha})\sqrt{ac} + b}{1 - R_D}$$

pour les petites valeurs de λT_M et en négligeant b ,il vient :

$$\frac{\Delta(\alpha)}{1-R_D} \quad 2(\alpha + \frac{1}{\alpha})\sqrt{\frac{\tau}{T_M}}$$

Exemple : $\alpha = 10^2$ ou 10^{-2} , $\tau = 3\ 10^{-8}$, $T_M = 3\ 10^{-2}$;

$$\frac{\Delta(\alpha)}{1-R_D} = 2\ 10^{-1}$$

III - SYSTEME T.M.R

Description

Le classique système T.M.R peut facilement être adapté aux techniques logicielles. Les solutions possibles sont toutes basées sur des échanges d'informations entre unités et des décisions de reconfiguration prises conjointement par plusieurs processeurs de la structure.

Solution A : "reconfiguration par décision majoritaire"

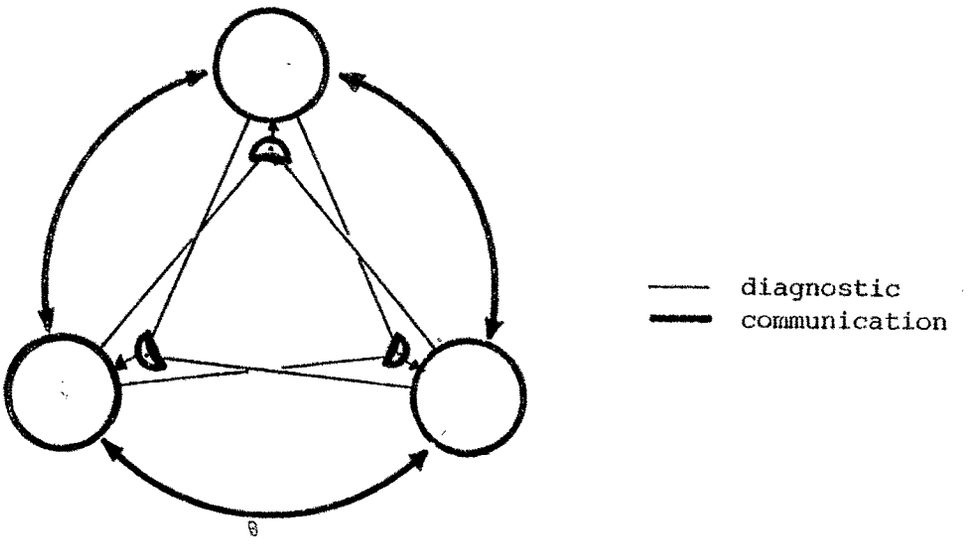


Figure A1.8

Chaque calculateur échange des informations avec les deux autres. Une décision de déconnexion n'est effectuéeque lorsque deux processeurs détectent simultanément une anomalie dans le troisième.

Solution B : "reconfiguration grâce à un graphe de diagnostic [PREPA 67]

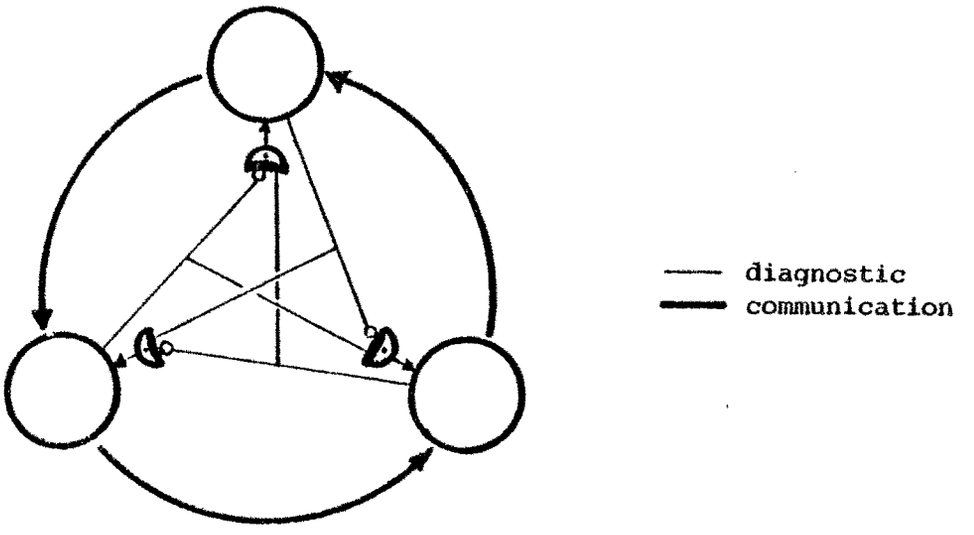


Figure A1.9

Chaque calculateur n'échange de l'information qu'avec un seul de ces voisins. La décision de déconnexion d'une unité n'est pratiquée que lorsque cette unité est reconnue en panne par une autre unité elle-même testée par la troisième. En présence d'une seule unité en panne, cette stratégie correspond bien à un T.M.R [PREPA 67] .

Remarque : Ces deux solutions peuvent recevoir un dispositif auxiliaire permettant l'application de la stratégie duplex après l'occurrence de la première panne. La modélisation proposée ci-dessous tient compte de cet ajout.

Modélisation

La modélisation s'effectue de façon analogue au cas duplex par une chaîne de Markov "à pas temporel" θ. Aux hypothèses simplificatrices déjà prises dans ce dernier cas vient s'ajouter l'hypothèse suivante :

- la période de comparaison reste inchangée lors du passage en mode duplex. Le système est alors représenté par la chaîne suivante (Figure A1.10).

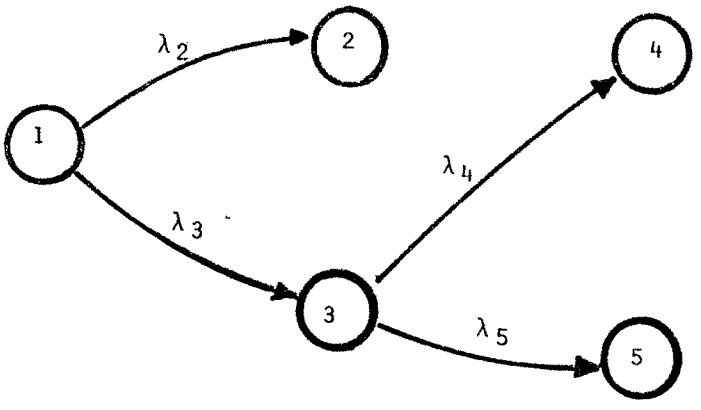


Figure A1.10

- l'état 1 est l'état de marche correcte;
- l'état 2 est l'état de panne non détectée. L'occurrence de panne multiple dans le triplet durant l'intervalle séparant deux comparaisons provoque la transition de l'état 1 à l'état 2;
- l'état 3 correspond à la détection et à la reconfiguration d'une panne (unique) du triplet initial;
- le sous graphe (3,4,5) correspond à la stratégie duplex.

En conséquence si nous posons $p = e^{-\lambda\theta}$ et $q = 1-e^{-\lambda\theta}$ nous avons :

- $\lambda_2 = q^3 + 3q^2p$
- $\lambda_3 = 3qp^2$
- $\lambda_4 = q^2$
- $\lambda_5 = 2qp$

ainsi que la représentation matricielle suivante :

$$P^{(n)} = \begin{bmatrix} p^3 & 0 & 0 & 0 & 0 \\ q^3+3q^2p & 1 & 0 & 0 & 0 \\ 3qp^2 & 0 & p^2 & 0 & 0 \\ 0 & 0 & q^2 & 1 & 0 \\ 0 & 0 & 2qp & 0 & 1 \end{bmatrix} \cdot P^{(n-1)} \quad \text{avec } P^{(0)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

et $P_i^{(n)}$ = Probabilité pour que le système soit dans l'état i au bout du pas n.

Résolution de proche en proche :

$$P_1^{(n)} = p^{3n}$$

$$P_2^{(n)} = [q^3 + 3q^2p]\, p^{3(n-1)} + P_2^{(n-1)} = [q^3 + 3q^2p] \sum_{i=o}^{n-1} p^{3i}$$

$$P_2^{(n)} = [q^3 + 3q^2p]\ [\frac{1-p^{3n}}{1-p^3}]$$

$$P_3^{(n)} = 3qp^2p^{3(n-1)} + p^2\, P_3^{(n-1)} = 3qp^2 \sum_{i=o}^{n-1} p^{3(i)}\, p^{-2(i-n+1)} = 3qp^{2n} \sum_{i=o}^{n-1} p^i$$

$$P_3^{(n)} = 3qp^{2n}\ \frac{1-p^n}{1-p} = 3[p^{2n} - p^{3n}]$$

$$P_4^{(n)} = q^2\, P_3^{(n-1)} + P_4^{(n-1)} = q^2 \sum_{i=o}^{n-1} P_3^{(n-1)} = 3q^2 [\sum_{i=o}^{n-1} p^{2n} - \sum_{i=o}^{n-1} p^{3n}]$$

$$P_4^{(n)} = 3q^2\ [\frac{1-p^{2n}}{1-p^2} - \frac{1-p^{3n}}{1-p^3}]$$

$$P_5^{(n)} = 2qp\, P_3^{(n-1)} + P_5^{(n-1)} = 2qp \sum_{i=o}^{n-1} P_3^{(n-1)}$$

$$P_5^{(n)} = 6qp\ [\frac{1-p^{2n}}{1-p^2} - \frac{1-p^{3n}}{1-p^3}]$$

D'où les expressions de la fiabilité et de la sécurité:

$$\boxed{R = P_3 + P_1 = 3p^{2n} - 2p^{3n}}$$

$$1 - S = P_2 + P_4 = 3q^2\ [\frac{1-p^{2n}}{1-p^2} - \frac{1-p^{3n}}{1-p^3}] + [q^3 + 3q^2p]\ [\frac{1-p^{3n}}{1-p^3}]$$

$$\boxed{1 - S = 3q^2\ [\frac{1-p^{2n}}{1-p^2}] - 2q^3\ [\frac{1-p^{3n}}{1-p^3}]}$$

Etude de la fiabilité

Il est facile de constater que R est une fonction strictement croissante de θ :

$$n = \frac{T_M}{\theta-\tau}$$

$$R = 3\, e^{-2\lambda\theta \frac{T_M}{\theta-\tau}} - 2\, e^{-3\lambda\theta\frac{T_M}{\theta-\tau}}$$

Soit en posant $x = e^{-\lambda\theta\frac{T_M}{\theta-\tau}} = p^n$ donc $x \in [0 \quad 1]$

$R(x) = 3x^2 - 2x^3$

On sait (fiabilité du T.M.R.) que R(x) est une fonction strictement croissante de x. On en déduit le tableau suivant :

θ	τ		$+\infty$
x	0		r
R	0	↗	$3r^2-2r^3$

On remarque que R atteint très vite sa valeur limite $R_{lim}=3r^2-2r^3$.

1) Supposons $\frac{\lambda T_M\theta}{\theta-\tau}$ petit ($< 10^{-2}$) il vient $R(\theta) \underset{\sim}{\simeq} 1 - \frac{3\lambda^2T_M^2\,\theta^2}{(\theta-\tau)^2}$

α et λT_M petits, il vient:

$$R_{lim} = 1 - 3\lambda^2 T_M$$

$$\Delta R = R_{lim} - R(\theta) = \frac{3\lambda^2T_M^2(2\theta\tau - \tau^2)}{(\theta-\tau)^2}$$

d'où une erreur relative sur l'infiabilité de :

$$\frac{\Delta R}{1-R_{lim}} = \frac{(2\theta\tau-\tau^2)}{(\theta-\tau)^2}$$

Exemple : $\theta = 10\tau$; $\frac{\Delta R}{1-R_{lim}} = 2\ 10^{-2}$

2) Supposons $\frac{\tau\lambda T_M}{\theta-\tau}$ petit ($< 10^{-2}$)

$$\Delta = 3r^2 - 2r^3 - 3\ r^2\ e^{-2\frac{\lambda T_M \tau}{\theta-\tau}} + 2\ r^3\ e^{-3\frac{\lambda T_M \tau}{\theta-\tau}}$$

$$\Delta = 3r^2 - 2r^3 - 3\ r^2\ (1-2\tfrac{\lambda T_M \tau}{\theta-\tau}) + 2\ r^3\ (1-3\ \tfrac{\lambda T_M \tau}{\theta-\tau})$$

$\Delta \underset{\sim}{\leq} 6\ \frac{\lambda T_M \tau}{\theta-\tau}\ (r^2 - r^3)$; on pose R majorant de la fiabilité du TMR matériel

$$\frac{\Delta}{1-R} \underset{\sim}{\leq} \frac{6\lambda T_M \tau r^2(1-r)}{(\theta-\tau)(1-3r^2+2r^3)} = \frac{6\lambda T_M \tau r^2(1-r)}{(\theta-\tau)(1-r)(1+r-2r^2)} = \frac{6\lambda T_M \tau r^2}{(\theta-\tau)(1+r-2r^2)}$$

r	0		1
F'(r)	0	↗	+∞

$$F(r) = \frac{r^2}{1+r-2r^3} \qquad F'(r) = \frac{2r+r^2}{(1+r-2r^2)^2}$$

Exemple : $r = 0,95$; $r^2 = 0,9025$; $1+r-2r^2 = 0,145$; $\frac{r^2}{1+r-2r^2} = 6,2$

$\lambda T_M = 0,0513$

$$\frac{\Delta}{1-R} \leq \frac{6 \times 5,2\ 10^{-2} \times \tau}{\theta - \tau} \times 6,23 \leq \frac{2\tau}{\theta-\tau}$$

$\theta = 101\tau \Rightarrow \frac{\Delta}{1-R} \leq 2\ 10^{-2}$. pour r plus proche de 1 et, donc λT_M plus peti
on se reportera à l'étude précédente.

Etude de la sécurité

Comme dans le cas de la stratégie duplex on peut montrer que pour des valeurs suffisamment élevées de θ la désécurité croît linéairement avec θ.

Supposons $\frac{\lambda T_M \tau}{\theta-\tau}$ petit ($< 10^{-2}$) et $\lambda\theta$ petit devant l'unité :

$$1-S \underset{\sim}{\leq} 3\lambda^2\theta^2\ (\frac{1-p^{2n}}{2\lambda\theta}) - 2\lambda^3\theta^3\ (\frac{1-p^{3n}}{3\lambda\theta})\ ;$$

Puisque $\frac{\lambda T_M \theta}{\theta-\tau}$ petit il vient:

$$1 - S \approx \frac{3}{2} \lambda\theta \left[1-r^2 \left[1- \frac{2\lambda T_M \tau}{\theta-\tau}\right]\right] - \frac{2}{3} \lambda^2\theta^2 \left[1-r^3\left(1- \frac{3\lambda T_M \theta}{\theta-\tau}\right.\right.$$

$$1 - S \approx \frac{3}{2} \lambda\theta\left[1-r^2 + r^2 \frac{2\lambda T_M \tau}{\theta-\tau}\right] - \frac{2}{3} \lambda^2\theta^2 \left[1-r^3 + r^2 \frac{2\lambda T_M \tau}{\theta-\tau}\right]$$

On peut négliger le terme du 2ième ordre en θ (coefficient λ^2) et en négligeant le terme en $\frac{\tau}{\theta-\tau}$ il vient

$$1 - S \approx \frac{3}{2} \lambda\theta \ (1-r^2)$$

Exemple : $\lambda = 3{,}6 \ 10^{-4}$ $\qquad 1-r^2 = 0{,}9985$

$1 - S \approx 5{,}392 \ 10^{-4}$ soit pour $\theta = 210^{-5}$ heures

$1 - S \approx 1{,}078 \ 10^{-8}$

IV - T.M.R. SIMPLEX

Une structure capable d'assurer cette stratégie peut être facilement dérivée des structures susceptibles de supporter la stratégie T.M.R.. Nous ne nous étendrons pas sur leurs architectures et nous nous bornerons à donner une évaluation rapide de cette stratégie.
Les conventions prises dans la suite sont analogues à celles adoptées lors de l'étude de la stratégie Duplex-Réserve.
La méthode de calcul de la fiabilité reste la même :

$R = P_r$ {mission assurée malgré une panne} + P_r {mission assurée sans panne}

En décomposant suivant le cycle d'arrivée de la panne il vient :

$$R = P_1 + P_2$$

$$P_1 = \sum_{k=1}^{k=n} e^{-3(k-1)\lambda\theta} \cdot 3e^{-2\lambda\theta}(1-e^{-\lambda\theta}) \cdot e^{-\lambda[T_M-(k-1)(\theta-\tau)]}$$

$$P_2 = e^{-3\lambda\theta\frac{T_M}{\theta-\tau}}$$

Soit encore (en posant $r = e^{-\lambda T_M}$)

$$P_1 = 3e^{-2\lambda\theta}(1-e^{-\lambda\theta}) \cdot e^{-\lambda T_M} \sum_{k=1}^{k=n} e^{(-2\theta-\tau)\lambda(k-1)}$$

$$P_1 = 3\, e^{-2\lambda\theta}(1-e^{-\lambda\theta})\; r\left[\frac{1 - e^{-\lambda(2\theta+\tau)\frac{T_M}{\theta-\tau}}}{1 - e^{-2\lambda\theta}\, e^{-\lambda\tau}}\right]$$

$$P_1 = \frac{3\, e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-2\lambda\theta}\, e^{-\lambda\tau}}\, r\left[1 - e^{-2\lambda T_M} - \frac{3\lambda T_M \tau}{\theta-\tau}\right]$$

$$P_1 = \frac{3\, e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-2\lambda\theta}\, e^{-\lambda\tau}} \cdot r\left[1 - r^2\, r^{\frac{3\tau}{\theta-\tau}}\right]$$

$$P_2 = r^3\, r^{\frac{3\tau}{\theta-\tau}}$$

$$R = \frac{3e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-2\lambda\theta}\, e^{-\lambda\tau}} \cdot \left(r\left[1-r^2\, r^{\frac{3\tau}{\theta-\tau}}\right]\right) + r^3\, r^{\frac{3\tau}{\theta-\tau}}$$

Comme dans le cas de la stratégie duplex-réserve le maximum peut être facilement situé en étudiant la fonction :

$$\frac{3e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1 - e^{-2\lambda\theta}e^{-\lambda\tau}}$$

Nous étudierons directement la différence en R et son majorant théorique $R_{TS} = \frac{3}{2} r - \frac{1}{2} r^3$ pour des valeurs faibles de $\frac{\tau}{\lambda}$ et $\lambda\theta$

$$\Delta = \frac{3}{2} r (1-r^2) + r^3 - R$$

Approximons les termes de R

$$\frac{3e^{-2\lambda\theta}(1-e^{-\lambda\theta})}{1-e^{-2\lambda\theta}e^{-\lambda\tau}} \approx \frac{3(1-2\lambda\theta(\lambda\theta)}{2\lambda\theta+\tau\lambda} \approx (1-2\lambda\theta) \frac{3\theta}{2\theta+\tau} = \frac{3}{2}(1-\frac{\tau}{2\theta})(1-2\lambda\theta)$$

$$r^{\frac{3\tau}{\theta-\tau}} = 1 - 3\lambda T_M \frac{\tau}{\theta} \qquad \approx \frac{3}{2}(1-\frac{\tau}{2\theta}-2\lambda\theta)$$

D'où l'expression de Δ

$$\Delta = \frac{3}{2} r(1-r^2) + r^3 - \frac{3}{2}(1-\frac{\tau}{2\theta} - 2\lambda\theta)\, r\, (1-r^2(1-3\lambda\, T_M \frac{\tau}{\theta})) - r^3(1-3\lambda T_M \frac{\tau}{\theta})$$

$$\Delta = r^3\, 3\lambda\, T_M \frac{\tau}{\theta} - \frac{9}{2} r^3 \lambda\, T_M \frac{\tau}{\theta} + \frac{3}{2}(\frac{\tau}{2\theta} + 2\lambda\theta)\, r(1-r^2(1-3\lambda T_M \frac{\tau}{\theta}))$$

$$\Delta = -\frac{3}{2} r^3\, \lambda T_M \frac{\tau}{\theta} + \frac{3}{2} r \frac{\tau}{2\theta} - \frac{3}{2}\frac{\tau}{2\theta} r^3 + \frac{9}{4}\frac{\tau^2}{\theta^2} r^3\, \lambda T_M + 3\lambda\theta r - 3\lambda\theta r^3 + 9\lambda^2 T_M$$

Qui comme pour le duplex-réserve (II) à la forme : (en négligeant le terme en $\frac{\tau^2}{\theta^2}$) :

$$\Delta = a\theta + b + \frac{c}{\theta}$$

avec : $a = 3\lambda r(1-r^2)$

$$b = 9\lambda^2 T_M \tau r^3$$

$$c = \frac{3}{2}\tau\, r\, [\frac{1}{2}(1-r^2) - r^2 \lambda T_M]$$

On peut montrer que c est toujours positif :

Posons $x = 2\lambda T_M$ $(x > 0)$

c a le signe de k(x) ($c = \frac{3}{2} \tau r.K(x)$)

$$k(x) = \frac{1}{2}(1-e^{-x}) - \frac{e^{-x}}{2} x$$

$$2k(x) = 1 - e^{-x} - e^{-x} x = 1 - (1+x)e^{-x}$$

$2k'(x) = x\, e^{-x}$ donc c croissant avec λT_M de 0 à $3\tau r$.

D'où le minimum pour

$$\theta_{MAX} = \sqrt{\frac{c}{a}} = \sqrt{\frac{1\tau}{4\lambda}} \sqrt{1 - \frac{2r^2\lambda T_M}{1-r^2}}$$

Et la valeur de ce minimum

$$\Delta_{MAX} = 2\sqrt{ca} + b = 2\sqrt{\frac{9}{4} \lambda\tau r^2 \left[1-r^2\right]\left[1-r^2-2r^2\lambda T_M\right]} + 9\lambda^2 T_M \tau r^3$$

$$\Delta_{MAX} = 3r\sqrt{\lambda\tau}\ (1-r^2)\sqrt{1 - \frac{2r^2\lambda T_M}{1-r^2}} + 9\lambda^2 T_M \tau r^3$$

Cas particulier des petits λT_M :

$$\theta_{MAX} = \sqrt{\frac{1}{4} \frac{\tau}{\lambda}} \sqrt{2\lambda T_M} = \sqrt{\frac{\tau T_M}{2}}$$

$$\Delta_{MAX} = 6\sqrt{\lambda\tau}\ \lambda T_M \sqrt{2\lambda T_M}$$

$$\Delta_{MAX} = 6\lambda^2 T_M \sqrt{2\tau\ T_M}$$

On peut facilement montrer que les variations de θ autour du maximum n'ont que peu d'influence sur l'incertitude $\frac{\Delta}{1-R_{TS}}$ en raisonnant comme lors de l'étude du duplex-réserve.

V - QUELQUES RESULTATS

Pour ces résultats nous avons fixé la durée τ de l'échange d'information :

$$\tau = 3\ 10^{-8} \text{ heures} \approx 10\ \mu s$$

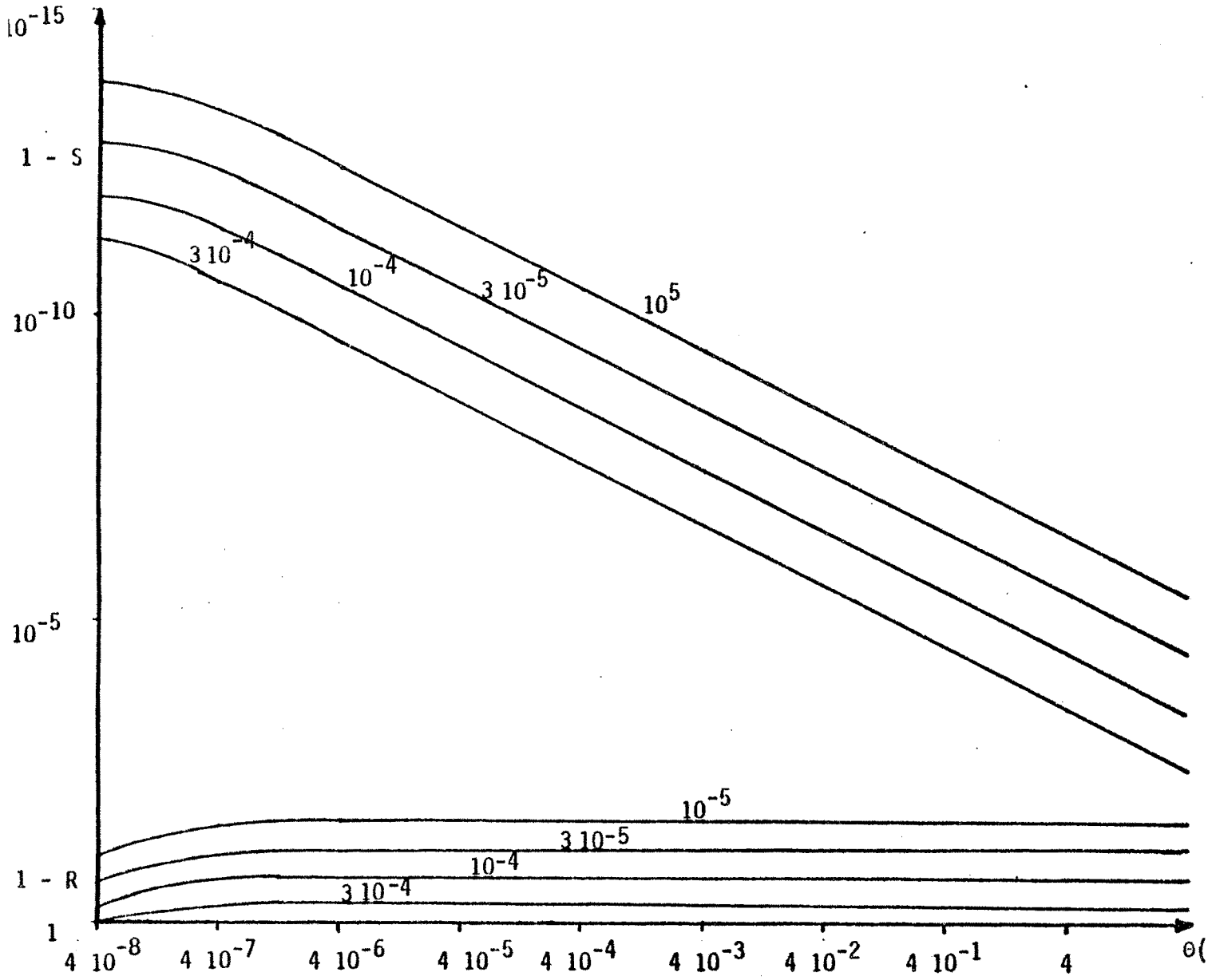
Dans ces résultats nous donnons soit l'"infiabilité" & l'"insécurité"(DUPLE T.M.R.), soit,simplement, l'infiabilité. Ces grandeurs seront portées sur une echelle logarithmique$_{\theta}$.

Influence du taux de panne des unités fonctionnelles

Nous opérons à temps de mission effectif constant (TM=1000 h). Les résulta présentés sur les graphiques I, II & III correspondent respectivement à des taux de panne égaux à : 10^{-5}, $3\ 10^{-5}$, 10^{-4}, $3\ 10^{-4}$ pannes par heures .

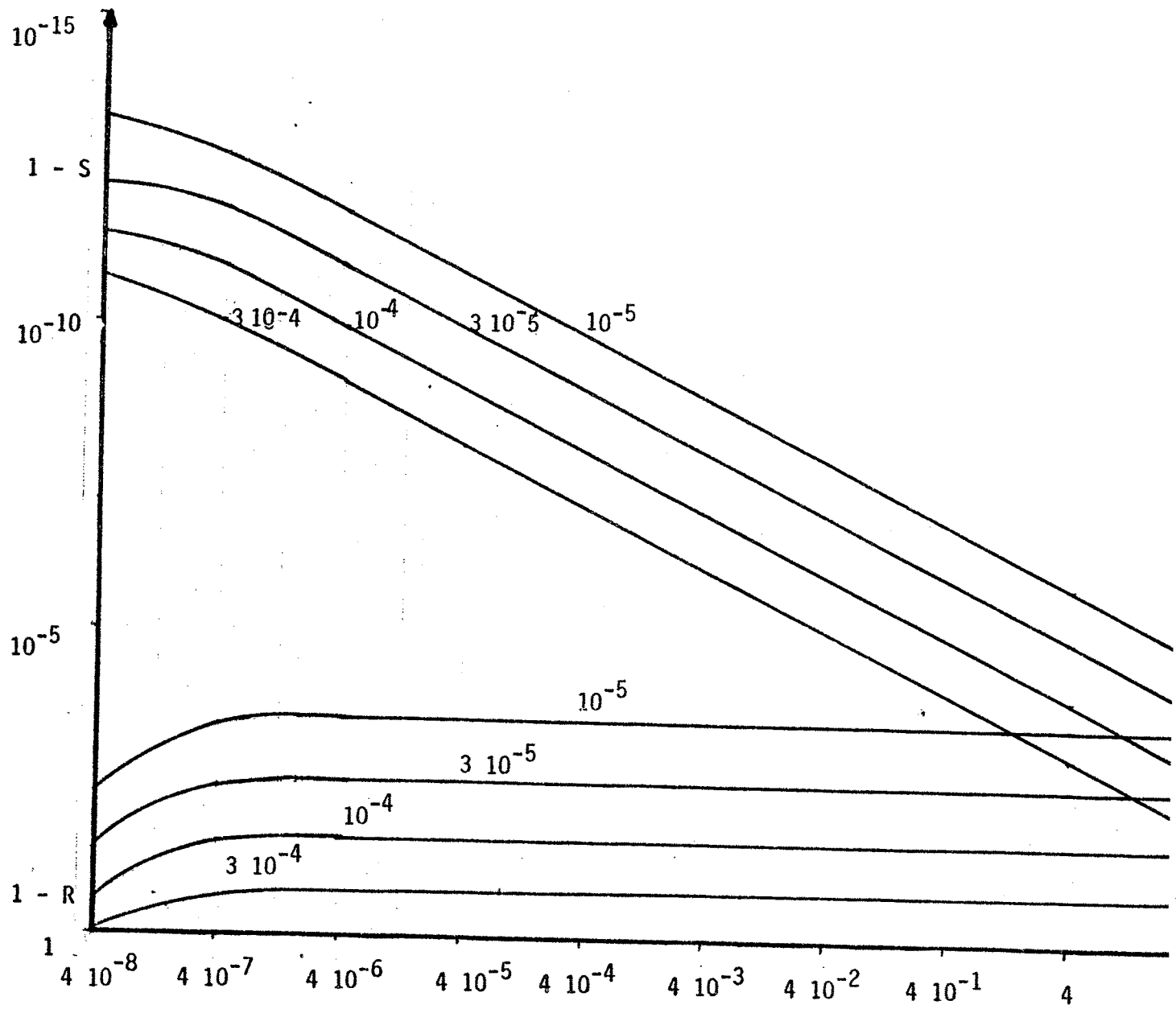
Influence du temps de mission effectif

Nous opérons à taux de panne constant ($\lambda=10^{-4}$p/h). Les résultats présentés sur les graphiquesIV, V & VI correspondent respectivement à des temps de mission égaux à : 10 h, 100 h, 1000 h, 10000 h .



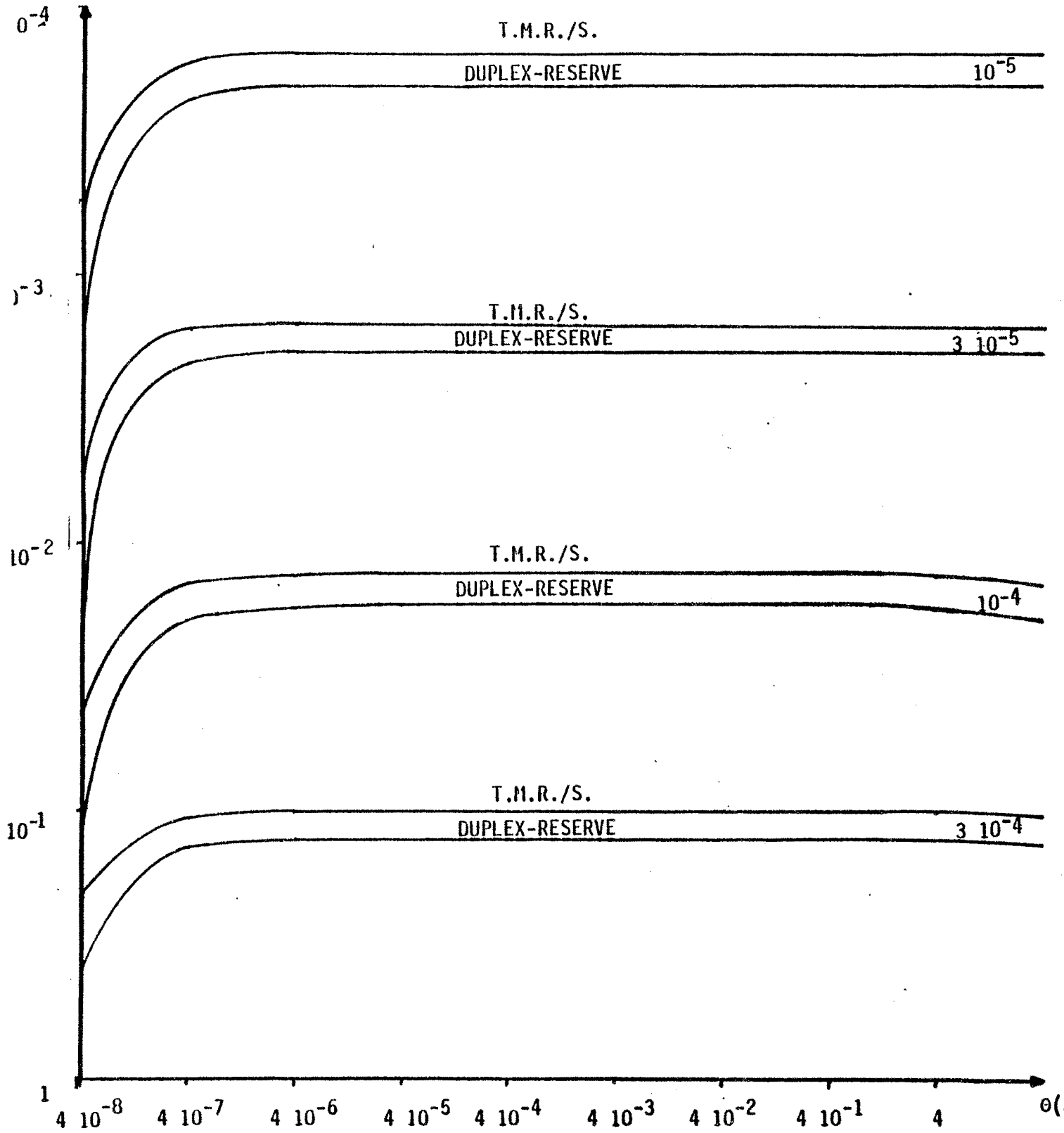
GRAPHIQUE I DUPLEX : - TM=1000 h

- $\tau = 3\ 10^{-8}$ h



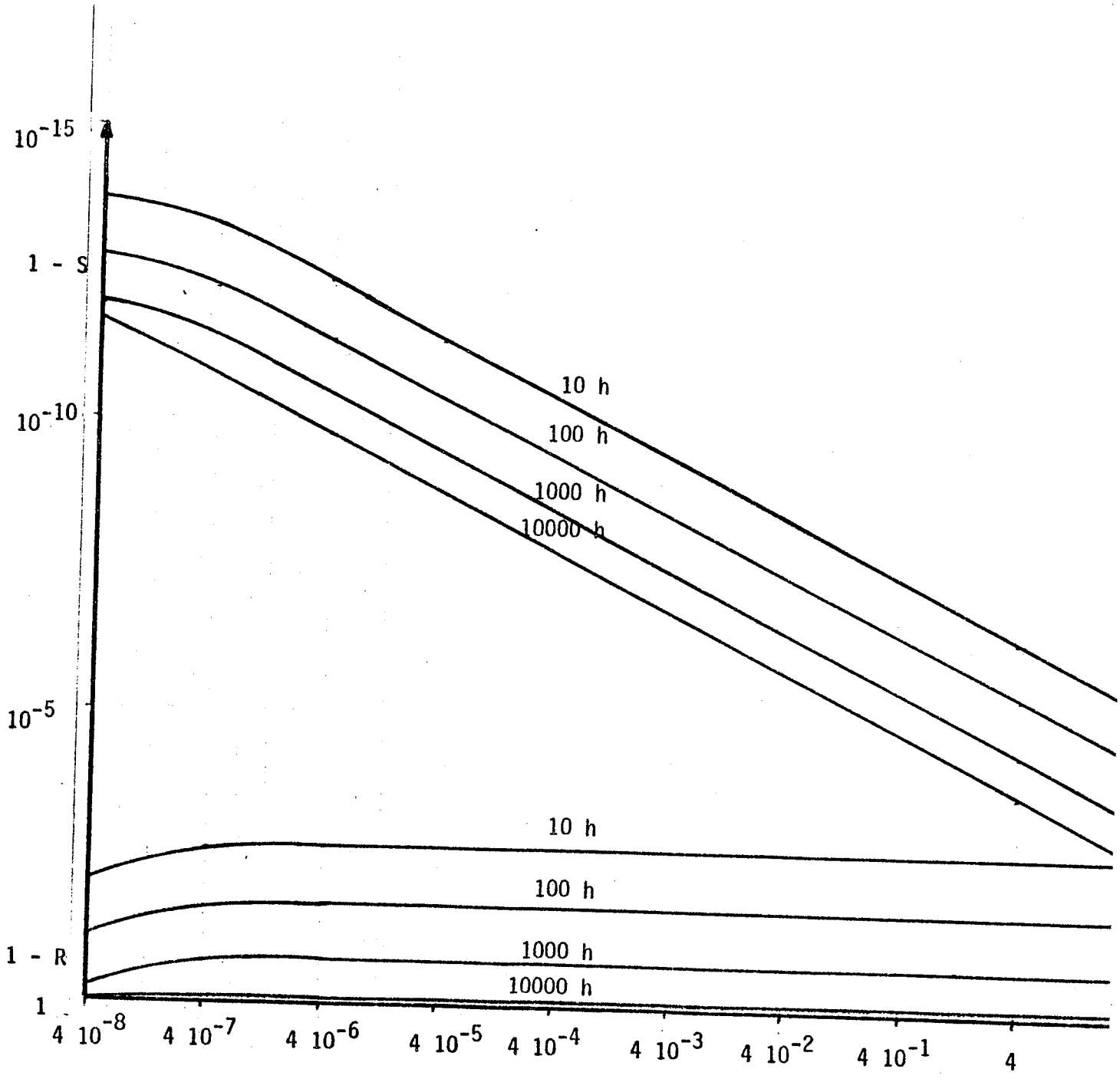
GRAPHIQUE II T.M.R. : $-TM=1000$ h

$-\tau = 3\ 10^{-8}$ h



GRAPHIQUE III T.M.R./S. & DUPLEX-RESERVE :

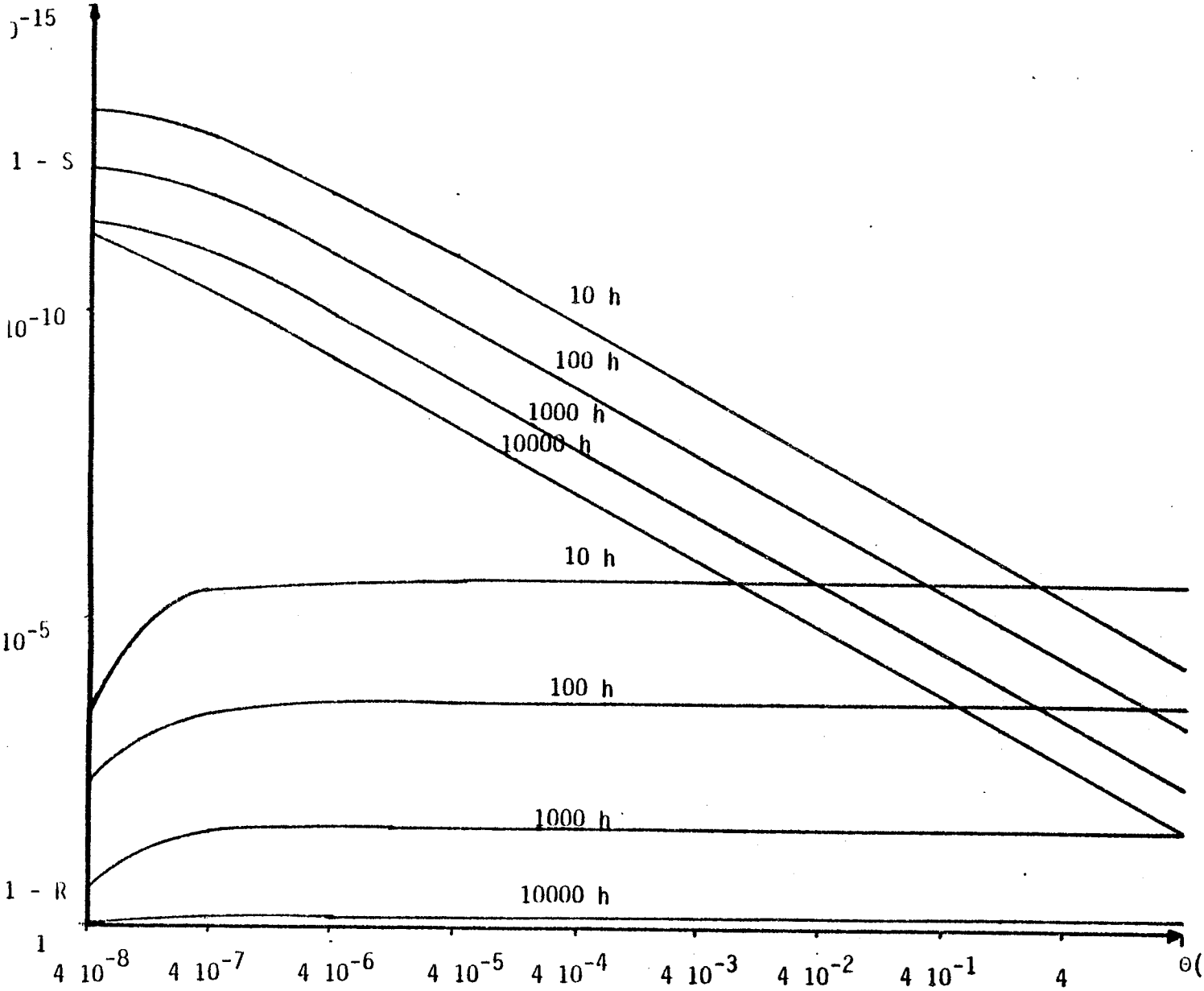
- TM=1000 h
- $\tau = 3\ 10^{-8}$ h



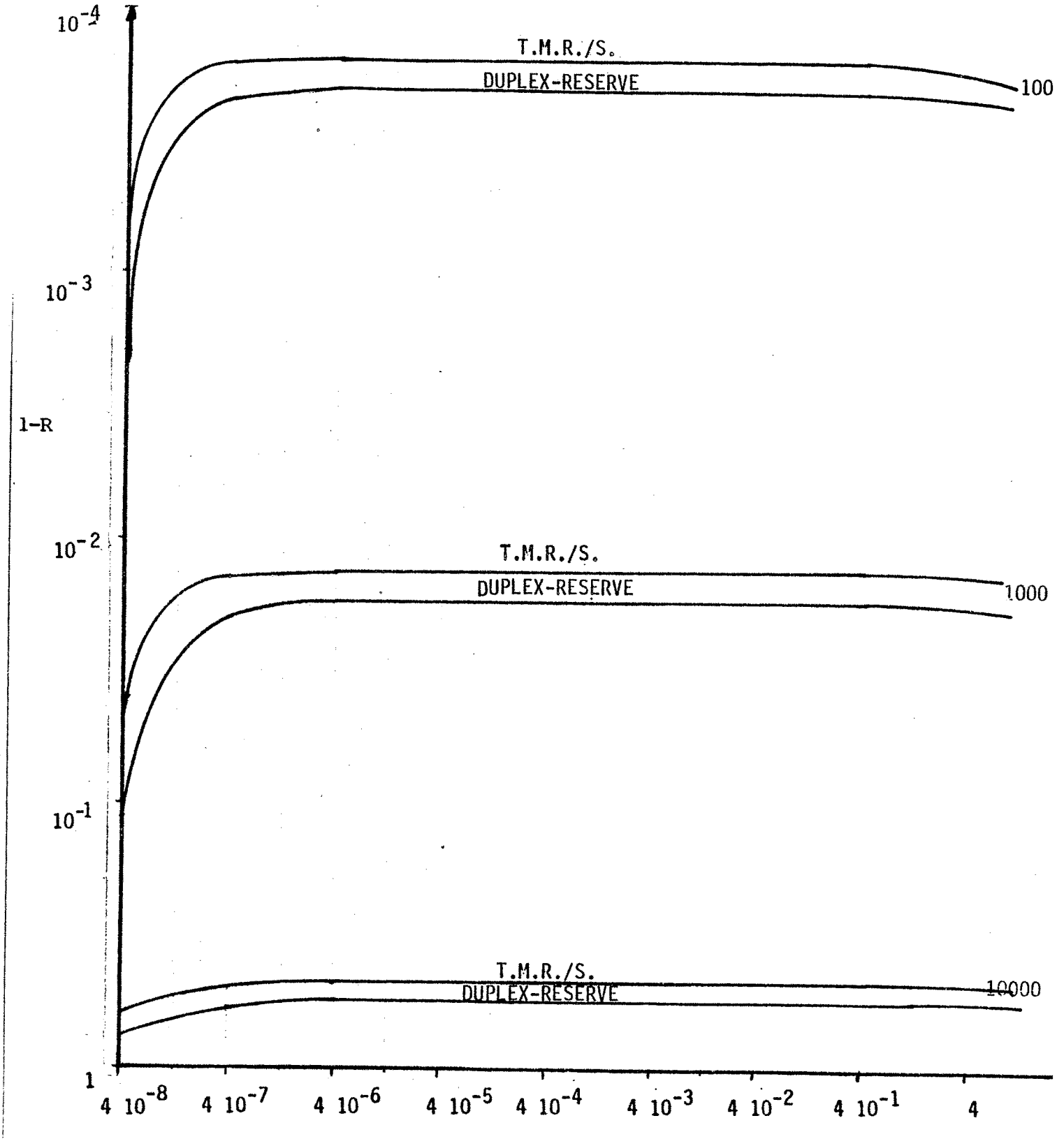
GRAPHIQUE IV

DUPLEX : $-\lambda = 10^{-4}$ p/h

$-\tau = 3\ 10^{-8}$ h



GRAPHIQUE V T.M.R. :-$\lambda=10^{-4}$ p/h
-$\tau=3\ 10^{-8}$ h



GRAPHIQUE VI T.M.R./S. & DUPLEX-RESERVE :

$-\lambda = 10^{-4}$ p/h

$-\tau = 3\ 10^{-8}$ h

VI - CONCLUSION

Pour chacune des architectures étudiées, les résultats précédents doivent être interprétés en fonction des caractéristiques de l'application implantée sur ces architectures.

Parmi les caractéristiques à prendre en compte on peut citer :

- La disposition temporelle des plages d'oisiveté des organes actifs ; ces plages pourront être utilisées par les échanges et les comparaisons de résultats.
- Les fréquences et volumes des sorties de résultats.
- Les performances de sûreté de fonctionnement exigées :
  * Fiabilité - Disponibilité.
  * Sécurité - Crédibilité.

Les résultats obtenus sont récapitulés ci-dessous :

* Missions à haute disponibilité :
  - Faible influence de la fréquence des comparaisons
  - Très bonnes performances : fiabilités pratiquement égales aux chiffres théoriques obtenus en considérant des voteurs et des comparateurs parfaits [BOUR 71].

* Missions à haute crédibilité : (on utilisera des architectures orientées vers ce type de missions : Duplex, TMR)
  - Forte influence de la période de comparaison sur la crédibilité (cf. valeurs de ce que nous avons appelé sécurité) qui reste toutefois proche de l'unité.

* Les principales difficultés ne vont survenir que pour des applications exigeant à la fois une forte disponibilité et une bonne crédibilité et n'offrant que peu de temps d'oisiveté en regard de leur débit d'information vers l'extérieur.
  Ces types d'applications,auxquelles les méthodes logicielles semblent inadaptées, constituent le domaine propre aux méthodes matérielles.
  On peut se demander si la perte de fiabilité ou de sécurité due à l'accroissement du logiciel ne viendra pas remettre en cause les résultats précédents.

  En fait, le volume de ce logiciel de comparaison est vraissemblablement faible par rapport à celui du logiciel d'application et, donc, ce logiciel ne pèsera que faiblement sur le terme de fiabilité du

logiciel par lequel on doit, dans tous les cas (comparaison logicielle ou matérielle), multiplier la fiabilité de la structure matérielle redondante pour obtenir la valeur de la fiabilité (ou de la sécurité) globale. De plus, on peut penser que ce logiciel est relativement simple à écrire.

En ce qui concerne la sécurité, de la même façon que pour le logiciel d'application, le logiciel de tolérance aux pannes pourra être implanté sur chaque calculateur dans des versions différentes [AVIZ 77].

ANNEXE 2

COMPARAISON ENTRE LES FIABILITES DE MISSION DE NOTRE STRATEGIE ET CELLES OBTENUES A L'AIDE DE STRATEGIES TRIPLIQUEES CLASSIQUES

Nous avons donc à comparer $R_N(Q,R)$ et $R_{TMR}(Q,R,W)$, $R_D(Q,R,W)$

ou $R_{TS}(Q,R,W)$ pour $Q \geq 1$; $1 \geq R \geq 0$; $2Q \geq W \geq 1$

avec $R_N(Q,R) = [2R^2-R^4]^Q$

$$R_{TMR}(Q,R,W) = [3R^{2Q/W} - 2R^{6Q/W}]^W$$

$$R_D(Q,R,W) = [R^{2Q/W} - R^{6Q/W} + R^{4Q/W}]^W$$

$$R_{TS}(Q,R,W) = [\frac{3}{2} R^{2Q/W} - \frac{1}{2} R^{6Q/N}]^W$$

On remarque facilement que pour le domaine considéré :

$$R_{TMR}(Q,R,W) < R_D(Q,R,W) < R_{TS}(Q,R,W) \quad (1)$$

en effet :

(1) $(r - r^3 + r^2) - (3r^2 - 2r^3) = r - 2r^2 + r^3 = (\sqrt{r} - r\sqrt{r})^2$

qui est toujours strictement positif quel que soit r positif,

(2) $(\frac{3}{2}r - \frac{1}{2} r^3) - (r - r^3 + r^2) = \frac{1}{2} r - r^2 + \frac{1}{2} r^3 = (\sqrt{\frac{r}{2}} - r\sqrt{\frac{r}{2}})^2$

qui est également toujours positif quel que soit r positif.

Nous comparerons donc successivement R_N avec R_{TS} puis R_D et enfin avec R_{TMR} s'il y a lieu.

I - COMPARAISON AVEC LA STRATEGIE TMR-SIMPLEX PARTITIONNEE

Considérons le rapport $A = R_N / R_{TS}$

$$A = \frac{[2R^2 - R^4]^Q}{[\frac{3}{2} R^{2Q/W} - \frac{1}{2} R^{6Q/W}]^W} = \frac{[2 - R^2]^Q}{[\frac{3}{2} - \frac{1}{2} R^{4Q/W}]^W}$$

posons $B = \text{Log } A$ il vient :

$$B = Q \text{ Log}[2 - R^2] - W \text{ Log}[\frac{3}{2} - \frac{1}{2} R^{4Q/W}].$$

Nous désirons comparer A à l'unité, il s'en suit que seul le signe de B nous intéresse.

Posons $C = \frac{2B}{W}$ et $\alpha = \frac{2Q}{W}$ $(1 \leq \alpha \leq 2Q)$

$$C(\alpha,R) = \alpha \text{Log}[2-R^2] - 2\text{Log}[\frac{3}{2} - \frac{1}{2} R^{4Q/W}]$$

$$\frac{\partial C}{\partial R} = 2\alpha R \left[\frac{-3+4R^{2\alpha-2} - R^{2\alpha}}{(2-R^2)(2-R^{2\alpha})} \right]$$

Appelons $F(\alpha,R)$ le polynôme $-3 + 4R^{2\alpha-2} - R^{2\alpha}$. R étant compris entre 0 et 1 le dénominateur de $\frac{\partial C}{\partial R}$ est toujours positif et $\frac{\partial C}{\partial R}$ a le même signe que $F(\alpha,R)$.

$$\frac{\partial F}{\partial R} = 2R^{2\alpha-3} [2(2\alpha-2) - \alpha R^2].$$

Deux cas se présentent suivant la position de la racine positive de $\frac{\partial F}{\partial R}$ $(R_o = 2\sqrt{\frac{\alpha-1}{\alpha}})$ par rapport au domaine de R $(0 \leq R \leq 1)$:

$-1 \leq \alpha < \frac{4}{3}$ $\qquad 0 \leq R_o \leq 1$

$-\alpha \geq \frac{4}{3}$ $\qquad R_o \geq 1$

1 : $\alpha \geq \frac{4}{3}$

Les variations des fonctions précédentes sont résumées dans le tableau suivant :

R	0		1
$\frac{\partial F}{\partial R}$	0	+	$6\alpha-8(\geq 0)$
F	-3	↗	0
$\frac{\partial C}{\partial R}$	0	−	0
C	$\alpha Log2-2Log\frac{3}{2}$	↘	0

Dans ce cas notre solution est toujours supérieure à la solution TMR-Simplex.

2 : $1 \leq \alpha \leq \frac{4}{3}$

Nous sommes alors conduits au tableau suivant :

R	0		R_1		R_0		1
$\frac{\partial F}{\partial R}$	0	+		+	0	−	$6\alpha-8$
F	-3	↗	0	↗		↘	0
$\frac{\partial C}{\partial R}$	0	−	0	+		+	0
C	$\alpha Log2-2Log\frac{3}{2}$	↘		↗			0

Deux éventualités se présentent suivant le signe de $K = \alpha Log2 - 2Log\frac{3}{2}$

2-1 $K \leq 0$: $1 \leq \alpha \leq \frac{2Log\frac{3}{2}}{Log2} \approx 1,17$

C est toujours négative. Notre solution est toujours inférieure à la solution TMR-Simplex.

2-2 $K > 0$: $\frac{2Log\frac{3}{2}}{Log2} < \alpha < \frac{4}{3}$

C admet une racine $R_2 (0 < R_2 < R_1)$. Pour les courtes missions $(R_2 \leq R < 1)$ notre solution est moins fiable que la solution TMR-Simplex; elle ne devient meilleure que lorsque le temps de mission augmente $(0 < R \leq R_2)$.

NB: On rappelle que R est une fonction décroissante du temps de mission.

II - COMPARAISON AVEC LA STRATEGIE DUPLEX-RESERVE

Considérons le rapport $A = R_N/R_D$

$$A = \frac{[2R^2 - R^4]^Q}{[R^{2Q/W} - R^{6Q/W} + R^{4Q/W}]^W} = \frac{[2 - R^2]^Q}{[1 - R^{4Q/W} + R^{2Q/W}]^W}$$

Comme précédemment nous étudierons le signe du logarithme de A que nous désignons par B.

$$B = \text{Log}A = Q \text{ Log } [2-R^2] - W \text{ Log}[1-R^{4Q/W} + R^{2Q/W}]$$

Nous poserons également $C = \frac{2B}{W}$ et $\alpha = \frac{2Q}{W}$ $(1 \leq \alpha \leq 2Q)$

Il vient donc :

$$C(\alpha,R) = \alpha \text{Log}[2-R^2] - 2\text{Log}[1-R^{2\alpha}+R^{\alpha}]$$

et donc par dérivation

$$\frac{\partial C}{\partial R} = -2\alpha R \frac{[1+2R^{\alpha-2}-4R^{2\alpha-2}+R^{2\alpha}]}{[2-R^2][1-R^{2\alpha}+R^{\alpha}]}$$

On remarque que $(1-r^2 + r)$ et $(2-r^2)$ sont positifs pour r compris entre 0 et 1 (NB : $r^2 \leq r$, $\forall r \in [0,1]$).

$\frac{\partial C}{\partial R}$ a donc le même signe que le polynôme $F(\alpha,R)$ avec:

$$F(\alpha,R) = -1 - 2R^{\alpha-2} + 4R^{2\alpha-2} - R^{2\alpha}$$

par dérivation il vient :

$$\frac{\partial F}{\partial R} = 2R^{\alpha-3} \left[-(\alpha-2) + 2(2\alpha-2)R^{\alpha} - \alpha R^{\alpha+2}\right]$$

Posons $G(\alpha,R) = -(\alpha-2) + (4\alpha-4)R^{\alpha} - \alpha R^{\alpha+2}$. G a le même signe que $\frac{\partial F}{\partial R}$.

Par dérivation il vient :

$$\frac{\partial G}{\partial R} = R^{\alpha-1}\ [(4\alpha-4)\alpha\ - \alpha(\alpha+2)R^2] = \alpha R^{\alpha-1}[\ (4\alpha-4)\ -\ (\alpha+2)R^2]$$

On remarque alors que la racine positive R_o ($R_o = \sqrt{\frac{4\alpha-4}{\alpha+2}}$) peut se trouver à l'intérieur du domaine de R ($0 \leqslant R \leqslant 1$)

$$\{\frac{4\alpha-4}{\alpha+2} \leqslant 1\} \Rightarrow \{\alpha \leqslant 2\}$$

Deux cas sont donc à étudier

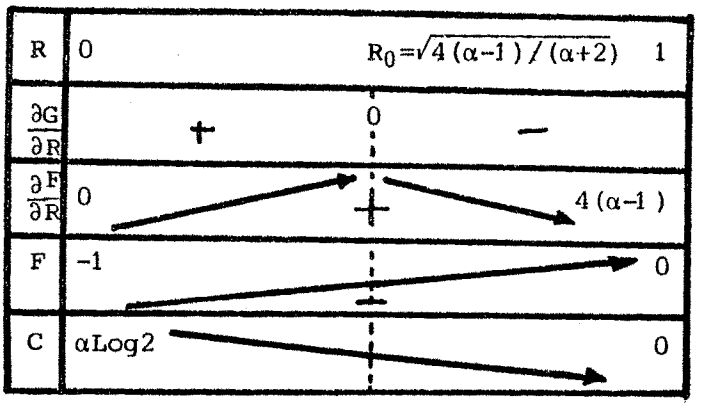
1) $\alpha > 2$: $\frac{\partial G}{\partial R}$ est donc toujours positif.

Le tableau récapitulatif suivant peut donc être tracé :

R	0		1
$\frac{\partial G}{\partial R}$		+	
$\frac{\partial F}{\partial R}$	0	+ ↗	$4(\alpha-1)$
F	-1	↗ − ↘	0
C	αLog2	↘	0

2) $\alpha \leqslant 2$: $\frac{\partial G}{\partial R}$ admet donc une racine R_o comprise entre 0 et 1.

Le tableau récapitulatif suivant est alors obtenu :



αLog2 étant toujours positif notre solution offre donc une fiabilité toujou supérieure à la solution duplex-réserve et ce, quel que soit le degré de partitionnement W.

III - COMPARAISON AVEC LA STRATEGIE TMR

Au vu du résultat précédent et compte tenu de la relation (1), nous pouvons conclure à la supériorité de notre solution quel que soit le degré de partitionnement W.

ANNEXE 3

DISPOSITIFS LOGICIELS POUR LA TOLERANCE AUX PANNES MATERIELLES

I - POSITION DU PROBLEME - FONCTIONS DU LOGICIEL REDONDANT

I - 1. Relations logiciel-matériel

Cette étude concerne l'exploitation de redondances matérielles grâce au logiciel. Il importe tout d'abord d'examiner la vision de ce matériel proposée au programmeur par le logiciel.
D'une façon générale, on considère que tout logiciel peut être décomposé en un groupe de programmes manipulant un ensemble de données. Cette constatation induit deux remarques :

1) Un système de programmation quel qu'il soit, doit nécessairement permettre la manipulation (directe ou indirecte) de deux types élémentaires d'objets.
   - des objets programmes (procédures, sous-programmes, fonctions... objets exécutables par un processeur),
   - des objets-données, désignés par des noms (ou adresses) et renfermant des valeurs manipulées et modifiées par les programmes.

2) Toutes les ressources du système adressables par le processeur, et donc manipulables au niveau du logiciel, apparaissent donc comme des objets donnés.

I - 2. Fonctions du logiciel redondant

Ces fonctions peuvent être classiquement partagées en trois groupes [AVIZ 77].

- Détection.
- Localisation.
- Reconfiguration.

Nous assimilons les techniques de masquage appliquées au niveau logiciel à des reconfigurations localisées au bloc logiciel ayant assuré la détection.

I - 2.1. Détection

L'obtention d'une bonne détection est cruciale pour atteindre de bonnes performances en sûreté de fonctionnement. Dans cette étude nous ne nous intéresserons qu'à la détection continue par opposition à la détection discontinue [BELLO 77]. Par détection discontinue nous entendons une détection menée par des programmes de test des ressources du système indépendamment de l'utilisation de ces dernières par les programmes d'application.

Ces méthodes tentent une vérification systématique du matériel en essayant de forcer la manifestation des pannes éventuelles [MOAL 76],[ROBA 75] . Elles ne s'adressent qu'aux pannes définitives. Par contre une détection continue possède les caractéristiques suivantes :

- Elle s'exerce au cours de l'exécution normale de l'application.
- Elle concerne la vérification de certaines valeurs au moment même de leur acquisition. Elle permet donc d'éviter en partie la propagation des erreurs ainsi que de se prémunir contre les pannes transitoires

Cette détection s'effectue par des vérifications de propriétés invariantes des objets du système.

Exemples : - Exécutions simultanées ou successives de plusieurs procédures décrivant un même algorithme. L'invariant à vérifier est alors l'identité des résultats.
- Tests de vraisemblance sur les valeurs de variables. L'invariant est alors simplement l'appartenance à un certain domaine de valeurs.

Les méthodes de programmation structurée (raffinements successifs, décomposition modulaire,...) prévoient la constitution de tout logiciel par une interconnexion d'objets. Chacun de ces objets est défini par une interface (domaine de valeurs conservées, fonctions d'accès...). Ces définitions décrivent certaines propriétés invariantes des objets inhérents à l'abstraction qu'ils représentent. L'utilisation d'un langage de programmation proposant un tel système d'abstraction suggère donc le placement de vérification d'invariant au niveau même des interfaces, i.e. des déclarations d'objets abstraits. De telles solutions sont généralement retenues pour la détection d'erreurs dans le logiciel [CRIST 79] (Figure A3.1).
Considérons maintenant un système construit à l'aide d'un langage utilisant explicitement un mécanisme d'abstraction (i.e. types abstraits). Les diverses liaisons entre couches logicielles sont alors clairement mentionnées. Si le niveau primitif d'abstraction est le niveau matériel lui-même, les liaisons matériel-logiciel seront propagées,grâce au mécanisme d'abstraction jusqu'aux niveaux élevés. Cette disposition nous semble tout particulièrement intéressante pour notre problème de contrôle par le logiciel de pannes matérielles.
Ces mécanismes d'abstraction permettent également la spécification de redondances dans l'implantation des objets. Un niveau d'abstraction pourra alors inclure des dispositifs de détection ou de masquage d'erreurs affectant

un niveau sous-jacent. Si ce dernier niveau est le matériel lui-même la correspondance erreurs détectées/pannes matérielles sera immédiate.

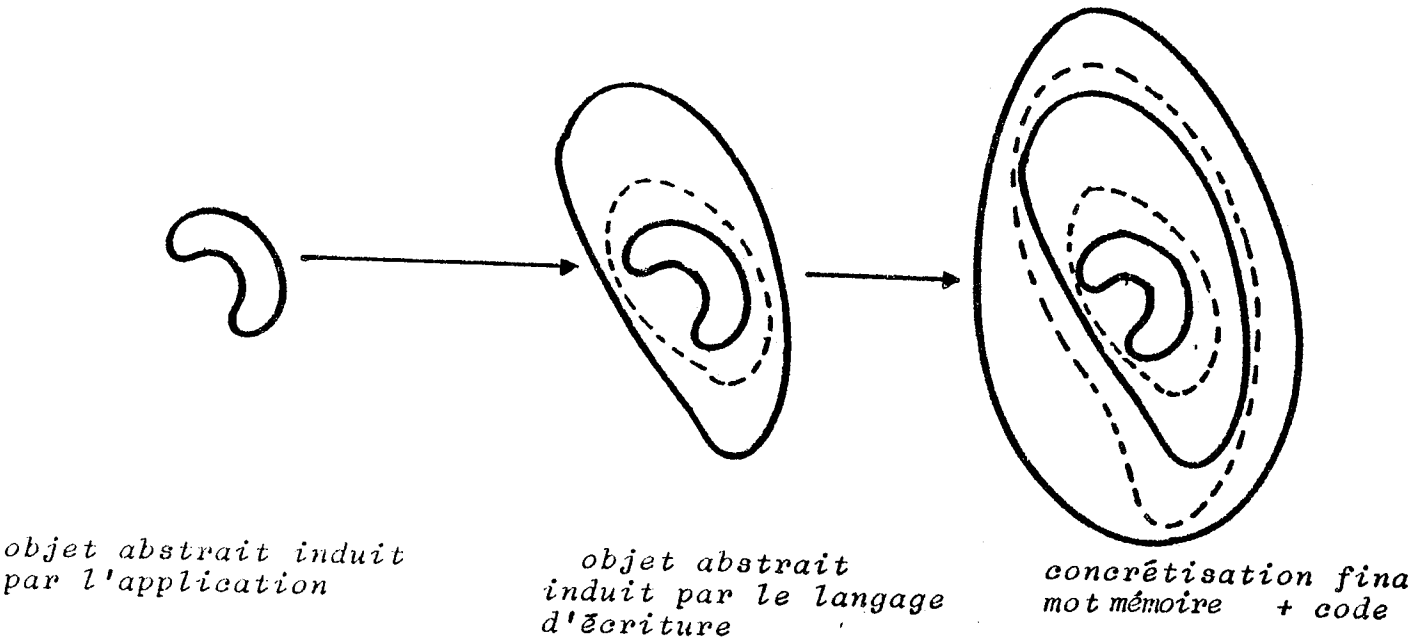


Figure A3.1

I - 2.2. Localisation

La localisation d'une panne est entreprise après détection d'une erreur. Elle a pour but la détermination d'un ensemble d'éléments "incriminables" le plus restreint possible. Ces éléments suspects seront alors définitivement inutilisés ou isolés après reconfiguration du système. La finesse de cette localisation influence fortement les performances opérationnelles (notamment durée de vie) du système.

D'une façon générale, cette localisation par voie logicielle est conduite par l'exécution de procédures de tests définies lors de la conception du système. Pour ce qui concerne les erreurs matérielles, l'existence de niveaux d'abstraction successifs permettra de placer ces procédures de localisation dans un interface de niveau compatible avec la finesse de diagnostic du matériel recherchée.

I - 2.3. Reconfiguration

Si l'activation par le logiciel, de dispositifs matériels de reconfiguratio (commutateur, interrupteur, "aiguillage"..) ne présente pas de difficultés particulières (il suffira en effet de plonger ces objets dans l'espace d'adressage des processeurs de la structure) il n'en va pas de même pour l'écriture de programmes destinés à modifier le logiciel lui-même pour le conformer à une nouvelle structure matérielle.
Ces difficultés proviennent essentiellement de la mauvaise adaptation des langages de programmation à l'expression de ces reconfigurations.

I - 2.3.1. Modifications du logiciel

Ces modifications sont consécutives à l'occurrence de pannes. Ces pannes lorsqu'elles sont définitives se traduisent par la perte de constituants de la structure, soit par suite de leur propre défaillance, soit par la défaillance de certaines liaisons. Si l'on reste au niveau de l'unité fonctionnelle, ces pertes, vues du logiciel, pourront être regroupées selon trois catégories :

- Perte d'emplacements mémoire ,
- Perte de périphériques ,
- Perte de processeurs.

Le logiciel après reconfiguration n'utilisera plus que le matériel encore indemne. Cette diminution du matériel disponible conduira, lorsque le logiciel assure lui-même sa reconfiguration, à des modifications souvent profondes :

- Modifications de la mission à assurer et donc modifications des procédures à activer ou de leurs liaisons mutuelles.
- Modification de l'implantation mémoire des variables manipulées par les programmes.
- Modifications dans l'adressage des ressources; une ressource en remplaçant une autre par suite de la défaillance de cette dernière.

I - 2.3.2. Expression des modifications du logiciel

Compte tenu des modifications désirées, cette expression se trouve soumise à des contraintes difficilement conciliables. Quel que soit le niveau de programmation choisi, cette expression augmentera la complexité globale du logiciel et donc le risque d'erreur, ce qui paraît peu souhaitable pour des applications orientées vers une haute sûreté de fonctionnement.

Si une programmation de haut niveau a été choisie, l'expression de ces modifications ne pourra être effectuée que sur la base des objets définis par la programmation. Ce qui pose à nouveau les problèmes de relations entre logiciel et matériel évoqués au §I.2.1 .
Remarquons enfin que le principe même de reconfiguration logicielle en cours de fonctionnement est profondément contradictoire avec les méthodes de programmation de haut-niveau dans lesquelles les descriptions de liaisons statiques sont en général préférées à tout mode de liaisons dynamiques (plus difficilement maîtrisables et donc sources d'erreurs potentielles).

I - 3. Conclusion

Si l'on se place dans le cadre d'une programmation de haut niveau, la production de logiciels orientés vers la tolérance aux pannes induit deux caractéristiques sur le langage d'écriture :

- Une liaison étroite entre logiciel et matériel (i.e. correspondance entre objet, logiciel et matériel support de l'objet). Cette liaison pouvant être assurée par un mécanisme d'abstraction inclus dans le langage de programmation lui-même.
- Une aptitude à l'expression de reconfigurations (i.e. remplacement d'un objet par un autre objet). Cette expression de la reconfiguration devra respecter au mieux les avantages de la programmation de haut-niveau (clarté d'expression, lisibilité, facilités d'entretien..).

II - PROPOSITIONS

II - 1. Principe de reconfiguration

Pour la mise en oeuvre d'algorithmes de reconfiguration, deux solutions semblent possibles :

- Spécification de la reconfiguration au sein même du programme de l'application.
- Utiliser un langage distinct du langage de programmation permettant de spécifier les modifications (consécutives à une reconfiguration) exercées sur les objets définis au cours de l'écriture de l'application.

* Le principal défaut de la première solution réside dans le mélange entre plusieurs concepts qui semblent logiquement indépendants :

- La programmation de la partie fonctionnelle.
- La programmation de la stratégie de reconfiguration.

Ce problème risque d'alourdir considérablement la programmation et cela, particulièrement dans le cas de langages de haut niveau (gestion d'une table de "sauts" supplémentaires ou remplacement de chaque appel de procédure par une suite de branchements conditionnés par l'état de dégradation de la structure).

* La deuxième solution peut être shématisée par l'interconnexion de deux programmes (Figure A3.2).

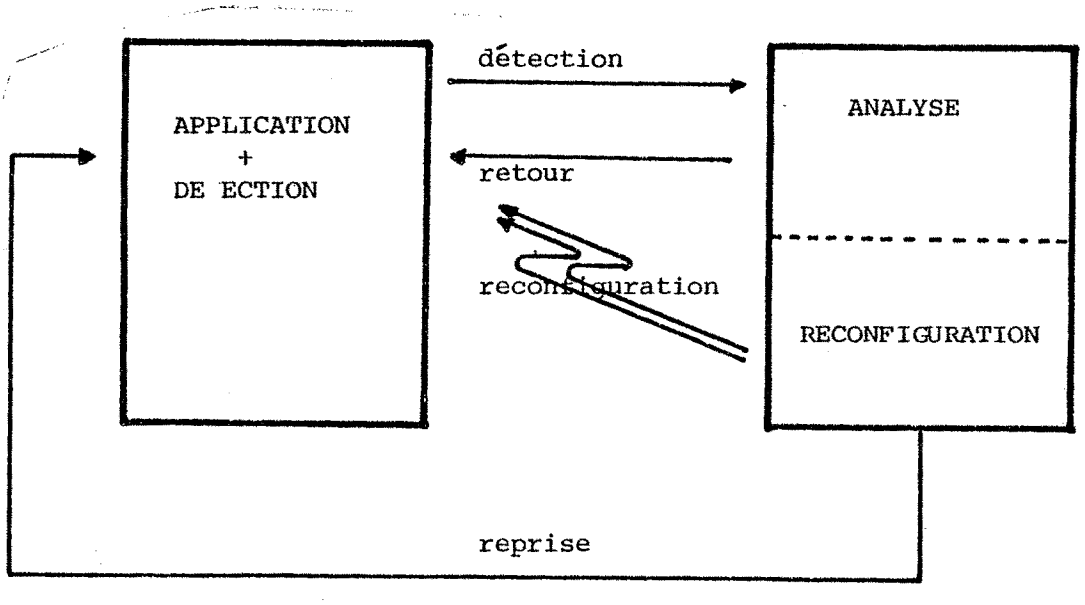


Figure A3.2

Lors de la détection d'une erreur nécessitant reconfiguration par le programme d'application, celui-ci fait appel au programme d'analyse et de reconfiguration. Ce dernier pourra alors entreprendre les modifications requises avant de redonner, le cas échéant, le contrôle au programme d'application.
On notera que cette solution permet de traiter indistinctement les erreurs détectées par le logiciel d'application et celles qui pourraient être signalées au moyen d'interruptions par des dispostififs matériels auxiliaires.
Cette solution conduit également à la séparation entre logiciel d'application (partie fonctionnelle et détection en ligne) et logiciel d'analyse de reconfiguration. De cette séparation, les avantages suivants sont attendus :

- Meilleure clarté et meilleure lisibilité des programmes d'où mise au point et maintenance facilitées.
- Isolation et faible interconnexion entre aspect fonctionnel et aspect reconfiguration, ce qui devrait permettre des modifications d'une de ces parties indépendamment de l'autre :
  - . modification de la stratégie de reconfiguration pour son adaptation à de nouveaux objectifs opérationnels,
  - . modification du logiciel fonctionnel :
    + introduction d'un nouveau périphérique ,
    + changement d'une ou de plusieurs fonctions;
    + etc.

Cette solution impose la décomposition du logiciel d'application en "blocs" sur la base desquels sera exécutée la reconfiguration. Ce qui n'offrira aucune difficulté si ces blocs correspondent aux unités de décomposition du langage d'écriture (I.2.3.2).
On remarque que cette opération de reconfiguration est très analogue à une "édition de liens" classique. Cette édition de lien n'est ni une édition de lien avant exécution (statique : à la compilation ou dans une phase de traduction prévue à cet effet), ni une édition de liens "dynamique" (genre MULTICS [CROCU 75]),mais d'une édition de liens "en cours de fonctionnement", l'exécution du programme d'application étant temporairement suspendue .
Une telle solution semble intéressante dans des cas autres que les systèmes tolérant les pannes; elle pourrait,en effet, avoir des applications, par exemple dans la conduite de procédés industriels qui réclame souvent des

modifications en ligne[DESH 77] ,ou même être incluse dans un langage d'écriture de systèmes[MOSS 77] .
De plus, cette solution reste applicable à des architectures modestes pour lesquelles une édition de liens globale suivie d'un chargement total de tout le programme d'application se serait révélée impossible (absence de mémoire secondaire, manque de place en mémoire principale .
Cette technique aurait pu être écartée pour des motifs de coût, de temps d'exécution ou de fiabilité (matériel supplémentaire d'où causes de panne supplémentaires) .

Structure à l'exécution

Nous sommes donc conduits à pratiquer cette édition de liens directement sur le code de l'application en mémoire centrale. Il est donc nécessaire que ce code possède une structure à l'exécution (liaisons normalisées entre "morceaux" de logiciels).

Deux solutions s'affrontent : adressage direct (figure A3.3); adressage indirect (figure A3.4).

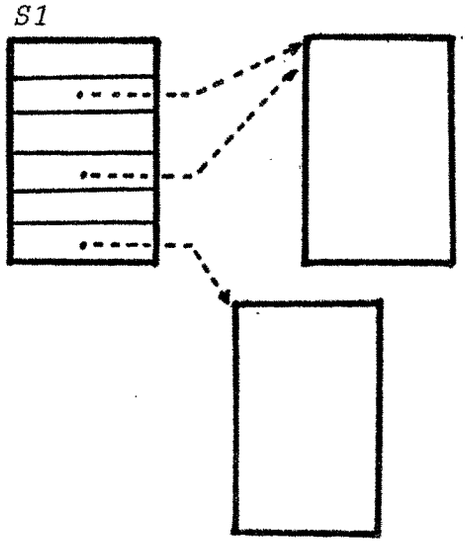


Figure A3.3

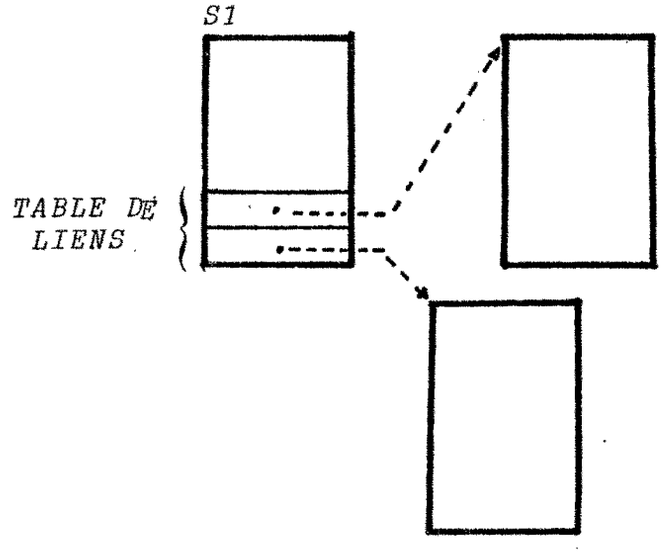


Figure A3.4

L'adressage direct est plus rapide à l'exécution, mais,par contre,augmente le nombre de modifications en cas de reconfiguration . A ces contraintes d'efficacité s'ajoute une contrainte propre à la technologie même des systèmes actuels à microprocesseurs : la séparation entre mémoire morte (ROM) et mémoire vive (RAM); les "pointeurs" de lien, du fait de leur modification éventuelle par suite d'une reconfiguration, doivent être implantés en mémoire vive et donc dans une zone disjointe du "morceau logiciel" duquel ils assurent les liaisons. Ce dernier sera souvent ("morceau de programme") implanté en memoire morte. Cette contrainte nous impose l'abandon de la solution "adressage direct (figure A3.3) et nous conduit à modifier la solution "adressage indirect" (figure A3.4) selon le schéma indiqué par la figure A3.5 .

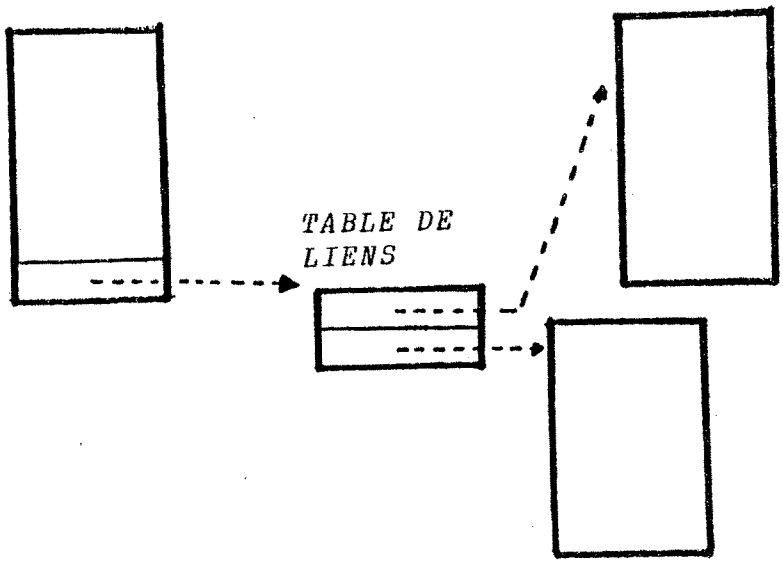


Figure A3.5

Programme d'analyse/reconfiguration

La structure du programme d'analyse/reconfiguration et ses liaisons avec le programme d'application peuvent être précisées :

Analyse

Le programme d'analyse/reconfiguration est appelé lors d'une détection d'erreur provenant soit du programme d'application, soit d'un détecteur matériel (voteur, comparateur, détecteur de code). Il est donc nécessaire de disposer au sein du langage d'écriture de l'application d'une primitive permettant l'activation du programme de reconfigura-

ration en lui mentionnant une indication sur le type de l'erreur commise. Plusieurs techniques d'implantation sont, a priori, possibles (déroutement, appel de sous-programmes, branchement..).
Le traitement d'une erreur par le programme de reconfiguration est conditionné par l'état de dégradation de la structure (i.e. "catalogu des ressources encore disponibles). Cet état devra être conservé dans des variables rémanentes (de durée de vie supérieure à l'exécution d'une reconfiguration) adressables par le programme d'analyse/reconfiguration.

Reconfiguration

La reconfiguration proprement dite comprend deux phases distinctes :

- Modification des variables rémanentes consécutivement aux nouvelles dégradations et modification des liens inter-modules.
- Retour du contrôle au programme d'application par activation d'une procédure de reprise prévue à cet effet.

Conclusion

Nous sommes donc confrontés à un double problème de définition (Figure A3.

- d'un mécanisme permettant de spécifier la correspondance matériel-logiciel dans un langage de haut niveau;
- d'une structure à l'exécution permettant une traduction aisée tant du programme d'application que du programme de reconfiguration.

Nous nous sommes délibérément placés dans le cadre d'un langage extensible (i.e comportant la notion de type abstrait). Ce choix se justifie d'une part par la facilité et la sécurité d'emploi offertes par ces langages et d'autre part par leur intérêt dans le domaine de la tolérance aux pannes (II.2).
L'approche suivie ici consiste à rapprocher au maximum le niveau primitif du langage d'application de la structure d'exécution. Ce qui nous conduit à nous pencher d'abord sur ce deuxième problème, avant de nous attaquer au problème de liaison matériel logiciel.

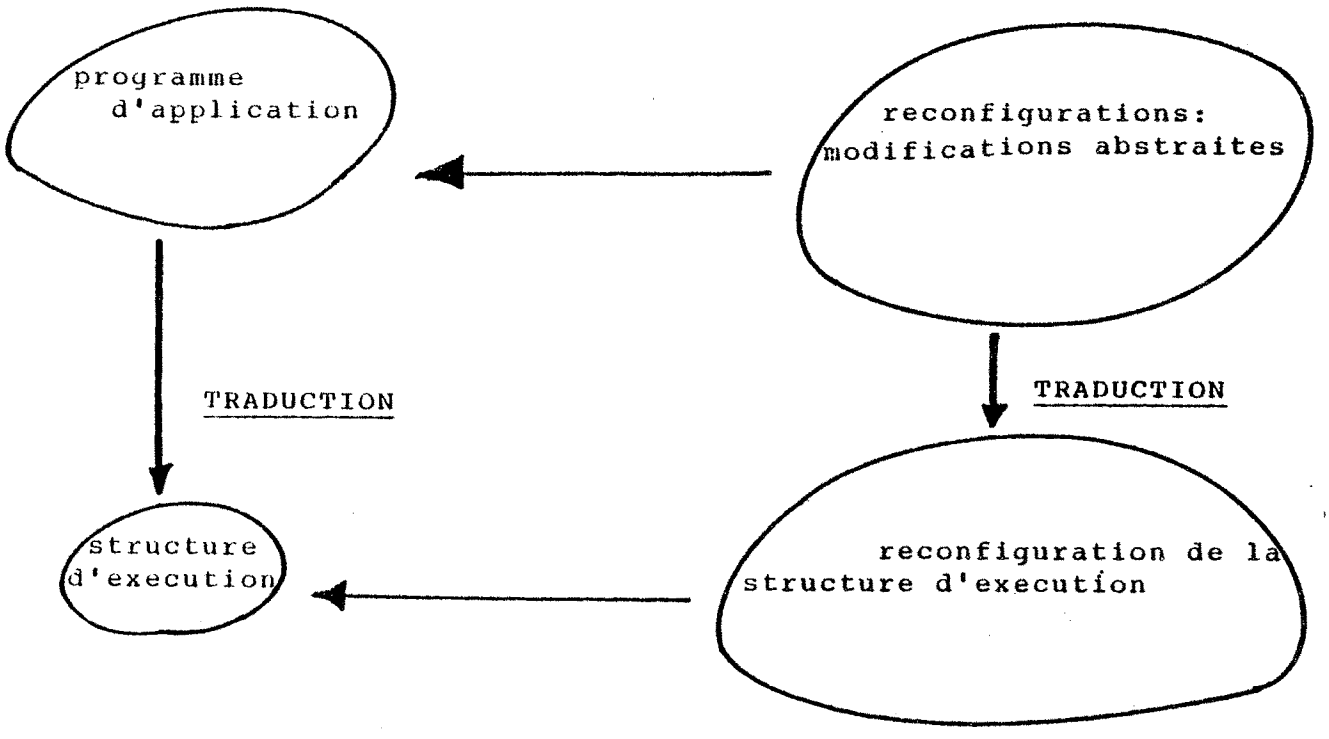


Figure A3.6

II - 2. Structure à l'exécution

Nous avons choisi une structure formée de l'interconnexion de plusieurs objets appelés segments. Un segment correspond à une suite d'emplacements (mots d'adresses physiques consécutives). Un segment possède donc une adresse d'origine ou base et une longueur (nombre d'emplacements du segment). Il est caractérisé par un type précisant la nature des informations qu'il contient. On distingue classiquement deux types distincts (conformément au II.1) :

- segment procédure renfermant du code exécutable
- segment donnée renfermant des valeurs variables ou constantes .

Cette décomposition des programmes est largement répandue notamment dans les langages de bas niveau (PL360 [WIRTH 67]et Assembleur OS/360 [IBM] : "CSECT, DSECT"). Cette structure est très adaptée aux adressages "basés" ou "indexés" largement répandus sur toutes les machines et en particulier sur les microprocesseurs. Cette solution ne nous semble pas imposer de restrictions notables sur le langage de programmation de l'application. Du point de vue de la reconfiguration cette solution impose le segment comme unité "remplaçable élémentaire" au niveau de cette structure d'exécution (et non au niveau du langage d'application).

Les liaisons entre segments sont assurées comme sur la figure A3.5 avec la restriction suivante :

- seuls les segments procédure possèdent une table de liaison chacune modifiant une des entrées de la table de liens : "placer l'adresse du segment i dans l'emplacement numéro j de la table de liens du segment procédure k".

II - 3. Langage d'application - Liaison matériel/logiciel

Le but de ce paragraphe n'est ni la définition exhaustive d'un langage de programmation, ni une proposition de traduction d'objets définis dans les langages de programmation en terme de la structure d'exécution choisie. En effet, si aucun consensus n'existe encore pour un langage d'écriture de système, les études actuelles (SESAME [MOSS 77], concurrent PASCAL [HANS 78].) présentent de nombreux points communs. Le mécanisme de traduction, lorsque celui-ci est décrit, fait souvent intervenir une structure d'exécution voisine de la nôtre.
Ce paragraphe a pour objet de proposer quelques solutions acceptables pour la spécification des liaisons logiciel/matériel.

II - 3.1. Objets renfermant du code exécutable

La nature même de l'information contenue dans ces objets facilite la correspondance matériel/logiciel : l'exécution séquentielle du code machine impose la contiguïté physique de cette information. Les langages de programmation proposent généralement des objets élémentaires de décomposition des programmes (sous-programme , procédure, fonction). La nécessaire contiguïté des instructions machine formant la traduction de ces objets conduit à une représentation naturelle des liaisons matériel/logiciel concernant ces objets. Il suffira pour cela de faire correspondre la traduction de chacun de ses objets à un segment-procédure de la structure d'exécution. Ce choix reste cohérent avec notre désir d'expression des reconfigurations au niveau du langage de programmation : l'unité de décomposition de programme est identique à l'unité de remplacement élémentaire. Par abus de langage nous adopterons la même dénomination (segment procédure pour ces deux unités.

II - 3.2. Objets données

A quelques exceptions près ([LEST 76]) le problème de liaisons entre objets-données etmatériel d'implantation n'est traité que dans des langages de bas niveau (PL360 [WIRTH 67], ASS OS/360[IBM]). Dans ces langages, l'espace mémoire attribué aux données est formé d'un ensemble de segments (analogues àceux de notre structure d'exécution). Chaque segment est décomposé en une suite d'objets "typés" (cette notion de type est très différente, notamment en ce qui concerne les protections d'accès de la notion classique de type). Cette méthode de spécification est très rigide et impose un mode d'implantation très strict des objets manipulés dans le programme; elle ne permet pas la constitution d'objets implantés sur plusieurs segments distincts.
Notre proposition essaie donc de dépasser ces limitations.
Nous supposons que chaque objet élémentaire de décomposition des programmes comprend la spécification de l'espace données sur lequel il devra travailler. Cette spécification sera fournie par une liste de noms. Chaque nom correspondra à une zone mémoire réservée à l'implantation de données. Chacune de ces zones correspondra à un segment donnéde la structure d'exécution. Ces zones, objets définis dans le langage d'application, seront désormais désignées sous le même nom que leurs représentations dans la structure d'exécution : segment-données. Ces objets sont les seuls à recevoir une adresse physique explicite, mais ne sont pas directement manipulés par les instructions du programme. Ce programme ne connaît qu'un ensemble d'objets dont l'implantation est spécifiée à leur création par l'intermédiaire du type auquel ils appartiennent.

Remarques

- L'ensemble des objets abstraits exprime donc la "vue" de l'espace de données propre au programme. Rien n'empêche, à priori, deux programmes distincts de proposer deux images distinctes des mêmes segments données.
- Le mécanisme décrit ci-dessus ne fait aucune hypothèse sur le mode (statique ou dynamique) de création des objets abstraits.

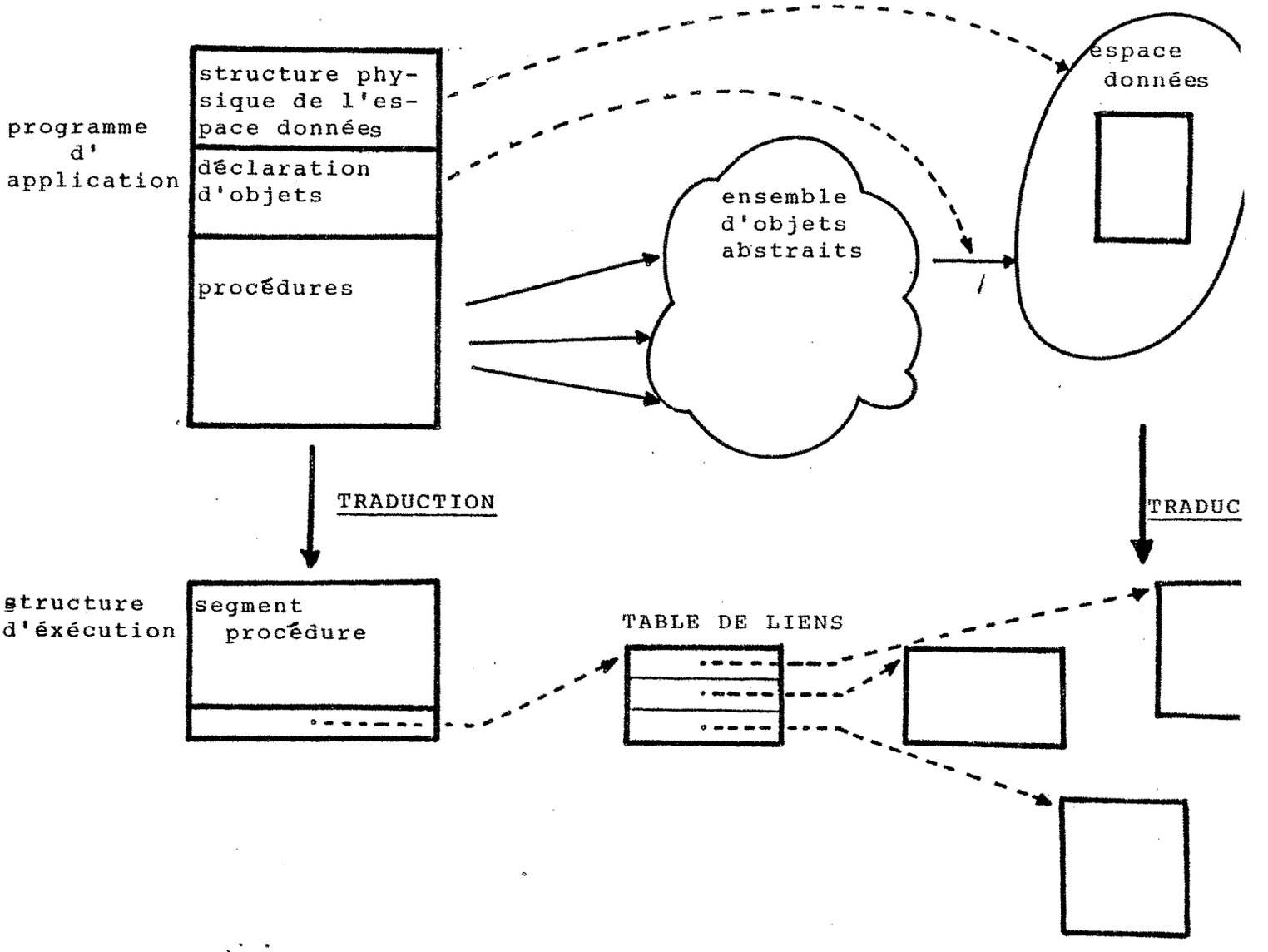


Figure A3.7

Proposition d'implantation à travers la notion de type

Dans notre proposition nous considérons que chaque type sera représenté en mémoire par un ensemble de zones mémoires. Chaque zone mémoire pouvant éventuellement appartenir à des segments différents. Cette implantation sera spécifiée au moment de la déclaration de l'ensemble des champs de ce type.

N.B. On peut supposer que les types primitifs du langage d'application ne possèdent qu'une zone de mémoire unique bien que cette contrainte ne soit aucunement nécessaire.

Exemple : Type T(S1, S2, S3);
Chaque objet du type T sera concrétisé par trois zones désignées respectivement par S1, S2, S3.

Lors de la déclaration d'un objet dans un segment procédure, on affecte à chacune des zones du type un nom de segment appartenant à l'espace donné du programme considéré.

Exemple :

dcl A of type T(SA = S2, SB = S1, SC = S3);

On crée un objet A de type T dont les trois zones S1, S2, S3 appartiennent respectivement aux segments données SB, SA, SC.

N.B. Notons que l'ordre de ces spécifications est indifférent.
Ce qui conduit à l'implantation (Figure A3.8).

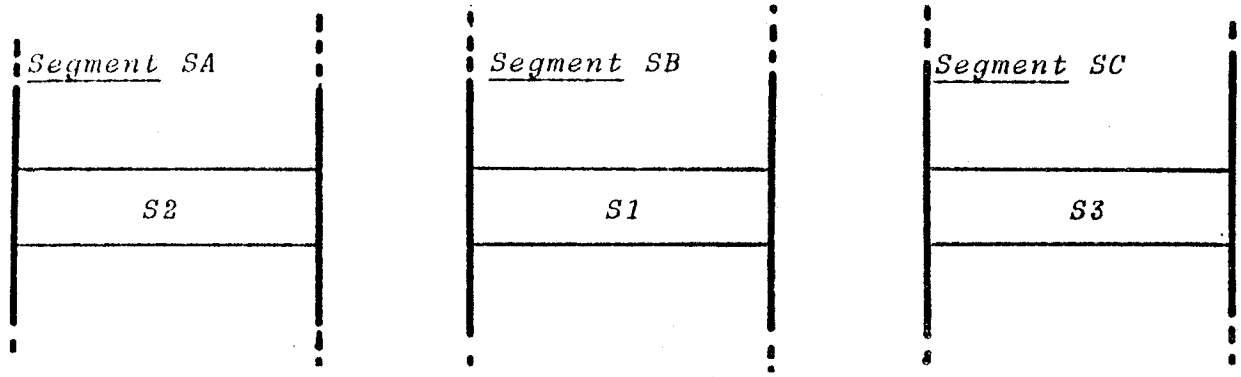


Figure A3.8

Il est possible d'implanter deux zones d'un même objet dans des emplacements contigüs d'un même segment.
Exemple :dcl A of type T (SA = (S1,S2), SC = S3);
qui conduit à l'implantation suivante : (Figure A3.9).

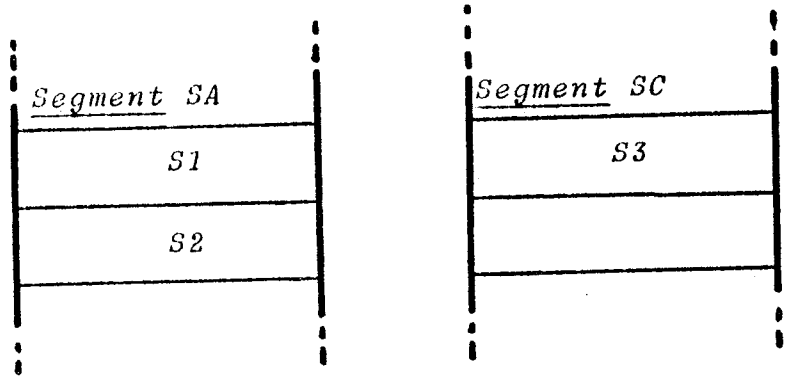


Figure A3.9

L'ordre d'apparition (S1,S2) correspond à l'ordre d'implantation :
Exemple : dcl A of type T (SA = (S2,S1), SB = S3);

qui produit : (Figure A3.10).

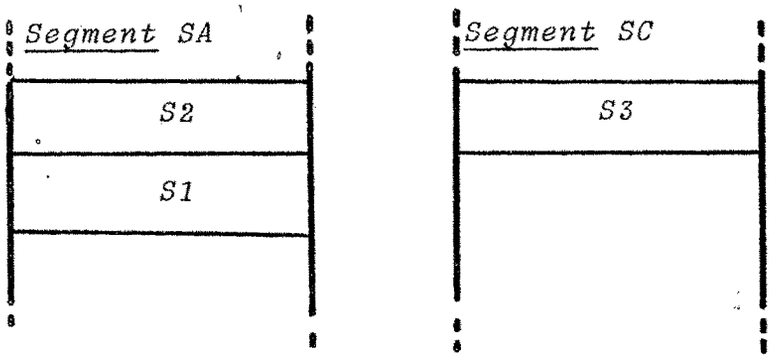


Figure A3.10

La structure de l'espace donnéesd'un segment procédure résulte d'une suite de déclaration d'objets appartenant chacun à des types définis antérieurement. L'implantation mémoire de ces objets se fera successivement dans l'ordre de leur déclaration.

Remarque : deux zones appartenant à des objets distincts, implantées dans un même segment, seront situées dans ce segment suivant un ordre identique à celui des déclarations des objets auxquels ils appartiennent.
Les types peuvent être construits à partie d'autres types antérieurement définis. Nous avons donc à répondre au problème de la "transmission" des schémas d'implantation précédemment définis à travers cette "hiérarchie" de types.
Nous proposons un mécanisme analogue à celui employé pour l'appel de procédure P dans une déclaration de procédure *P* ; la procédure P ayant des paramètres effectifs qui sont simultanément paramètres formels de *P* (Figure A3.11).

```
type T( D1,D2,D3,D4)
      dcl A of type T (D1 = (S1,S2), D4 = S3);
      dcl B of type T (D1 = S1, D2 = S2, D3 = S3);
end type
```

Figure A3.11

La déclaration ci-dessous

```
dcl C of type T (SA =D1 , SB =D2 , SC =D3 , S D =D4 );
```

conduit à l'implantation suivante (Figure A3.12).

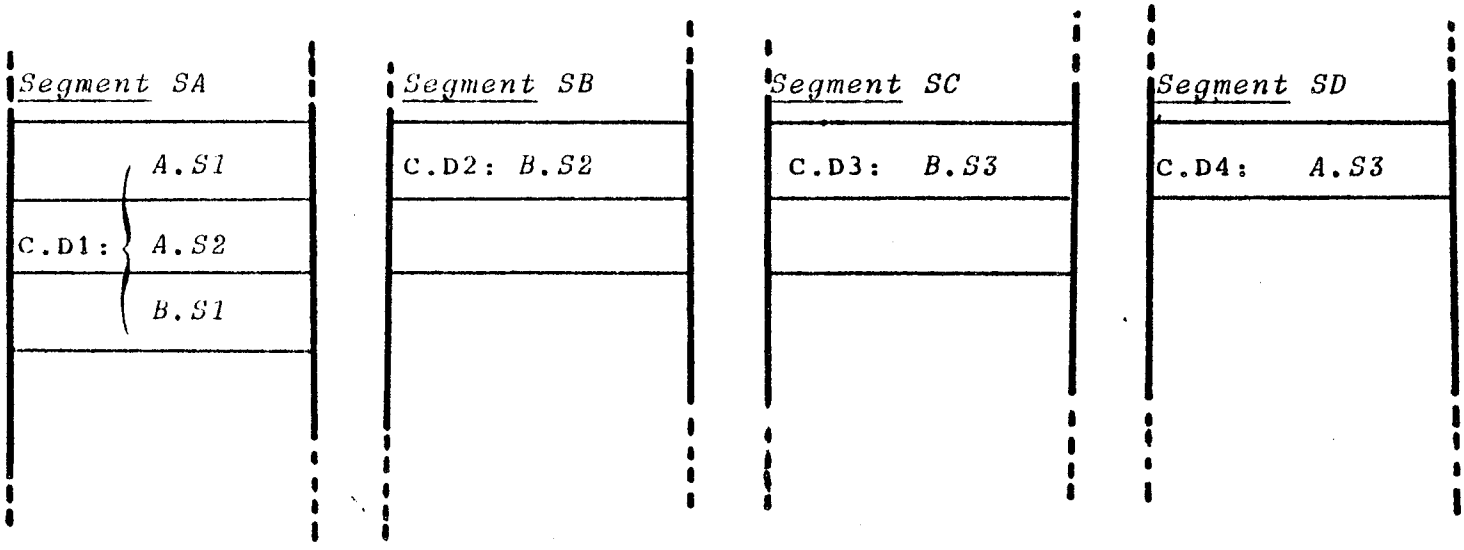


Figure A3.12

Alors que la déclaration ci-dessous

dcl C of type T (SA = (D2,D3), SB = (D1,D4))

mène à l'implantation indiquée par la figure A3.13.

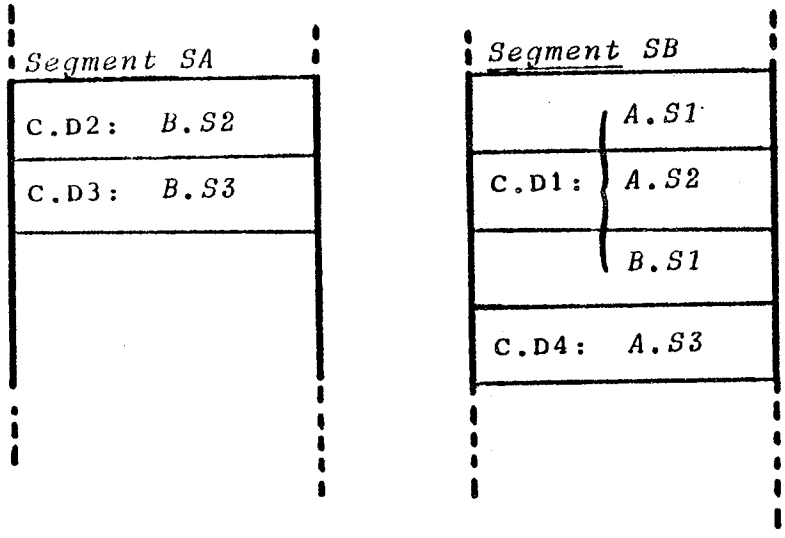


Figure A3.13

Remarques

1) Passage en paramètre d'une procédure d'un objet de type construit

Dans le cas de l'appel par référence (seul appel raisonnable pour des types suffisamment complexes) il suffira de passer une adresse par zone d'implantation du type.

Exemple

```
CALL TOTO (A); CØ  A est de type T ;
sera"traduit" par :
CALL TOTO (A.S1, A.S2, A.S3);
```

2) Imbrication des implantations d'objets abstraits

Nous avons vu que l'ordre des zones d'objets abstraits dans un segment est celui de la déclaration de ces objets.
La rupture de cet ordre dans notre proposition (i.e l'imbrication des implantations) peut être résolue par la déclaration d'un type supplémentaire.

Exemple : Soit à "imbriquer" les implantations de deux objets A et B appartenant au même type T.
Nous déclarons un type supplémentaire SUP de la façon suivante (Figure A3.

```
type SUP (Z1, Z2, Z3, Z4, Z5, Z6)
     dcl A of type T (Z1 = S1, Z2 = S2, Z3 = S3);
     dcl B of type T (Z4 = S1, Z5 = S2, Z6 = S3);
end type
```

Figure A3.14

La déclaration d'un objet SUP peut permettre toutes les imbrications entre les implantations de deux objets A et B de type T.

Dans le cas (classique) où les valeurs mémorisées dans un objet typé ne peuvent être manipulées que par les procédures d'accès propresà ce type, cette solution présente un point faible : on ne peut plus accéder au champ A et B d'un objet de type SUP par les procédures propres au type T,mais seulement par l'intermédiaire de procédures propres au type SUP; procédures qu'il conviendra alors de définir. Ce problème, que nous n'aborderons pas en détail ici,pourrait recevoir selon nous deux solutions :

A) Définition d'une nouvelle entité (structure par exemple) ayant toutes les caractéristiques d'un type, excepté son "opacité" (analogue à une CLASS de SIMULA [DAHL 70].) .

B) Affubler chacun des champs du type d'un attribut d'opacité. Cet attribut peut être plus ou moins complexe :
   - simplement indiquer si le champ est "caché" ou "visible" de l'extérieur,
   - ou spécifier pour chaque champ la liste des fonctions propres à son type qui pourront y accéder.

3) Allocation dynamique

L'approche proposée reste tout à fait compatible avec une allocation dynamique. L'adaptation à une allocation dynamique implicite (structure de bloc, attribut automatic PL/1 [VEILL]) est immédiate.: seule change la fonction de l'instruction de déclaration. Une allocation explicite (attribut controlled PL/1 [VEILL]) nécessite l'introduction d'une instruction de désallocation explicite.

4) Variables indicées : tableau

Dans un cadre d'allocateur statique ou dynamique, la "taille" d'une variable indicée est connue à sa déclaration. On peut donc traduire cette déclaration par une suite de déclarations d'objets de même type, le compilateur se chargeant de générer le code nécessaire pour l'accès à cette structure.

II - 4. Reconfigurations dans le langage d'application

Nous avons choisi d'exprimer les reconfigurations du logiciel sur la base même des objets du langage d'application (§II.2.3.2). L'expression de ces reconfigurations portant originellement sur des objets de base, réclamera parfois la définition d'objets nouveaux dans le langage d'application. Le présent paragraphe est consacré au problème du choix de ces objets, de l'expression des reconfigurations et de leur réalisation.

II - 4.1. Niveau primitif : Les segments

Si nous choisissons de faire apparaître explicitement les objets mêmes de la structure d'exécution au niveau du langage d'application, les modifications possibles dans le schéma d'interconnexion entre ces segments pourront être immédiatement traduites par une séquence d'instructions élémentaires de modifications de la structure d'exécution (voir §III.2).

II - 4.2. Introduction de la notion de module

La notion de module au sens de [MOSS 77]ou[DARON 78]peut facilement être construite à l'aide des segments précédemment définis et de règles de "visibilités" :
Un module est un regroupement de segments données et de segments procédure ces derniers étant les seuls autorisés à manipuler les premiers à l'exclusion de tous les autres segments procédures de l'application. Ces règles d'accès seront imposées par le système de programmation lui-même (compilateur ou éditeur de lien). A l'exécution un module sera donc représenté par un ensemble de segments procédures et de segments de données interconnectés (Figure A3.15).

Dans le cas du module, notre proposition de reconfiguration par modification de liens entre segments semble tout à fait adaptée aux transformations susceptibles d'être appliquées à un module :

- Remplacement d'une procédure d'accès,
- Remplacement d'une zone de donnée,
- Remplacement du module lui-même.

Toutes ces modifications peuvent être, en effet, aisément traduites en une succession de modifications de liens au sein de la structure d'exécution : seules opérations possibles dans notre proposition.

Remarques :

- On peut imaginer des langages d'application dans lesquels coëxistent la notion de module et la notion de segment . Ces segments externes aux modules présenteraient un accès non protégé de la part des segments procédure de tout le système.
- Si l'on suppose que tous les constituants d'un module sont compilés en même temps, il est facile d'imposer à toutes les procédures d'accès du module une même "vision" des segments de données du module. Cette option revient à une "mise en facteur", entre les

diverses procédures d'accès,du découpage des segments données en objets abstraits.

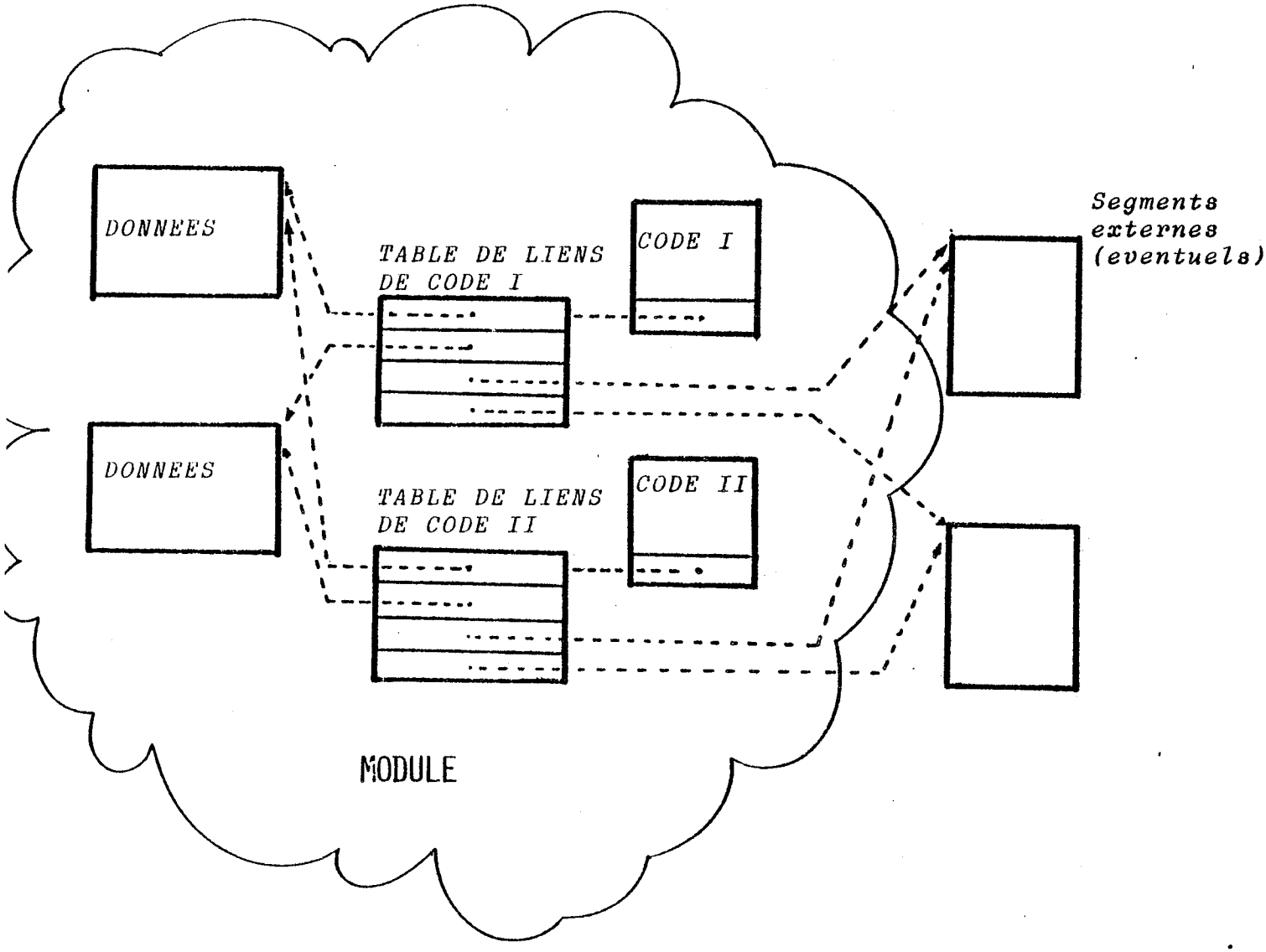


Figure A3.15

II - 5. Langage de reconfiguration

Le langage de reconfiguration doit comprendre un type d'instruction qui lui est propre : les instructions de reconfiguration qui manipulent directement les liens entre objets du langage d'application . Ces instructions sont trés voisines d'instruction de connexions [SESAM 76].

II - 5.1. Instructions de reconfiguration

L'instruction élémentaire de ce type consiste à changer la valeur d'un lien. Grâce au choix d'une méthode de compilation séparée sur la base des modules les liens entre procédures sont explicitement mentionnés sous forme de références externes. Les liens entre procédures et données sont explicitement spécifiés par la liste de segments placée en tête de module, conformément à la proposition du §II.2.
Les liens entre procédures et fonctions d'accès à un objet typé reçoivent une affectation dès la compilation. Leur désignation individuelle ne peut être réalisée que sous le nom de cette première affectation.
L'instruction de reconfiguration élémentaire aura donc la forme suivante (tout à fait identique à une instruction de connexion de SESAME).

<nom de lien> := <nom d'objet>

Compte tenu de la présentation précédent, on distingue deux noms de liens distincts :

- référence à un segment de donnée interne à un module.

Chacun de ces liens sera désigné par un couple nom de module, nom (fictif) de segment de donnée à l'intérieur du module, avec la syntaxe suivante (notation pointée) :

<nom de module> . <nom de segment fictif>

Le nom d'objet situé en partie droite d'une telle instruction devra obligatoirement être un nom de segment donnée.

- référence à une procédure externe et donc fournie par un module distinct. Même chose que précédemment et pour reprendre la terminologie de SESAME on aboutira à la syntaxe :

<nom de module> . <nom de procédure fictive>

Comme pour une instruction de connexion de SESAME le nom situé en partie droite sera un nom de procédure i.e. un nom de module suivi d'un nom de procédure interne à ce module.

II - 5.2. Liaison avec l'éditeur de lien (connecteur)

Un programme de reconfiguration, séquence d'instruction de reconfiguration, sera donc analogue à la phase de connexion de l'édition de lien d'un programme SESAME. On pourra notamment y inclure les primitives foreach [MOSS 77]ou with [MONT 77].

La phase de déclaration et de création des modules devra également comporter la spécification de l'implantation physique des différents segments (segments données ou procédures internes aux modules).

Cette spécification peut être faite en indiquant simplement pour chacun des segments une adresse physique d'implantation. La traduction de toutes les instructions de reconfigurations pourra donc être pratiquée au moment de l'édition de lien. Le regroupement de ces instructions permet également des vérifications à l'édition de lien (pas de partage de segments de données entre modules distincts, cohérence de l'implantation des segments compte tenu de leur taille respective..).

REFERENCES BIBLIOGRAPHIQUES : Ch. I

[BOUR 71] W. G. BOURICIUS & al.
Reliability modeling for fault-tolerant computers
I.E.E.E. Transactions on Computers, Vol C-20, N°11, Novembre 1971, pp 1306-1311.

[COURT 76] B. COURTOIS
Etude d'un calculateur tolérant des pannes: ses fiabilité, sécurité, performance et coût
Thèse de Docteur-ingenieur, 18 Decembre 1976, INP Grenoble

[HOPK 78] A. L. HOPKINS Jr, T. B. SMITH III,& J. H. LALA
A Highly Reliable Fault-Tolerant Computer for Aircraft Control
Proceedings of the I.E.E.E., V 66 N°10,Octobre 1978,pp 1221-123

[KAUF 77] A. KAUFMANN, D. GROUCHKO & R. CRUON
Mathematical Models for the Study of the Reliability of Systems
V 124 in Mathematics in Science & Enginering, R. BELLMAN Ed., ACADEMIC PRESS , New-york ,1977.

[LAP 75] J. C. LAPRIE
Prévision de la sureté de fonctionnement et architectures de structures temps réel réparables
Thèse de Doctorat d'Etat, 16 Juin 1975, UPS, Toulouse.

[MERAU 79] C. MERAUD, F. BROWAEYS, J. P. QUEILLE,& G. GERMAIN
COPRA: An Ultra-Reliable, Reconfigurable Computer for Aerospace Applications
Proceedings of the 1979 Fault Tolerant Computing Symposium, Madison (Wisc.), 20-22 Juin 1979.

[WENS 78] J. H. WENSLEY and al.
S.I.F.T.: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control
Proceedings of the I.E.E.E., V 66 N°10, Octobre 1978, pp 1240-1254.

REFERENCES BIBLIOGRAPHIQUES : Ch. II

[AVIZ 77] A. AVIZIENIS
Fault Tolerant Computing-progress, problems and prospects
Proceedings of the 1977 I.F.I.P. Congress, Toronto, North-Holland Publishing Company, pp 405-420.

[BELLM 77] H. BELLM & A. SAUER
Methods of data exchange between computers
Proceedings of the 1977 EUROMICRO Symposium, Amsterdam, North-Holland Publishing Company, pp 16-22.

[BERGE 70] C. BERGE
Graphes et hypergraphes
DUNOD Editeur, 1970 Paris.

[BOUR 71] W. G. BOURICIUS & al.
Reliability modeling for fault-tolerant computers
I.E.E.E. Transactions on Computers, V.C-20, N°11, pp 1306-1311, Novembre 1971.

[COURT 76] B. COURTOIS
Etude d'un calculateur tolerant des pannes: ses fiabilité, securité, performance et cöût
Thèse de Docteur-Ingenieur soutenue le 18 Décembre 1976, INP Grenoble.

[DAVI 78] D. DAVIES
Reliable Synchronization of Redondant Systems
Proceedings of the 5th Annual Symposium on Computer Architecture, Palo-Alto (Ca.), 3-4 Avril 1978, pp 236-237.

REFERENCES BIBLIOGRAPHIQUES : Ch II (suite)

[GRAHA 78] R. L. GRAHAM
The combinatorial Mathematics of scheduling
Scientific American, V.238, N°3, pp 124-132,
Mars 1978.

[MATH 71] F. P. MATHUR
On Reliability Modeling and Analysis of Ultrareliable Fault Tolerant Digital Systems
I.E.E.E. Transactions on Computers, C-20, N°11,
pp 1376-1382, Novembre 1971.

[MAZA 78] G. MAZARE
Structures multi-microprocesseurs, problèmes de parallélisme, définition et évaluation d'un système particulier
Thèse d'Etat soutenue le 19 Juin 1978, USMG-INP Grenoble.

[MERAU 76] C. MERAUD, F. BROWAEYS & G. GERMAIN
Automatic Rollback Techniques of the COPRA computer
Proceedings of the 1976 International Symposium on Fault-Tolerant Computing, I.E.E.E. N°76CH1094-2C, pp 23-29.

[PREPA 67] F. P. PREPARATA, G. METZE, & R. T. CHIEN
On the connection assignement problem of diagnosable systems
I.E.E.E. Transactions on Computers, V EC-16, N°6,
pp 848-854, Décembre 1967.

[SIEW 78] D. P. SIEWIOREK & S. McCONNEL
C.vmp : The implementation, Performance and Reliability of a Fault Tolerant Multiprocessor
Departments of Electrical Engineering and Computer Science, Carnegie-Mellon University, CMU-CS-78-108,
March 1978.

REFERENCES BIBLIOGRAPHIQUES : Ch II (fin)

[SAUC 78] G. SAUCIER
Design of high safety systems on microprocessors
Proceedings of EUROMICRO 78 Symposium, North-Holland Publishing Company, pp 160-166.

REFERENCES BIBLIOGRAPHIQUES : Ch III

[CERT 78] P. CASPI, J. GILLON, R. MAMPEY, J. PULOU
Etudes critiques de faisabilité d'architectures multiprocesseurs en avionique
Rapport principal & rapport annexe ,N°1/7179 & N°2/7179, CERT-DERA, Juillet 1978.

[CNET 76] CNET
Recueil de données de fiabilité du CNET, EDITION 76
CNET, Departement CPM/FMI, Route de Tregastel 22301, Lannion.

[INTEL 76] B. PASCOE
Reliability Reports RR-9,10,11, (MOS Static RAM 2107A, 2107B , 8080, 8080A microcomputer)
INTEL CORP.,Mars 1976 .

[KASOU 78] G. KASOUF & S. MERCURIO
Evaluation of LSI/MSI Reliability Models
Proceedings of the 1978 Annual Reliability and Maintainability Symposium, I.E.E.E. N°77CH1308-6R pp 443-447, Los Angeles (Ca.), 17-19 January 1978.

[MOTO 78] E. DOMANGUE
Motorola Reliability Report
Report Number 7750 Microcomponents MC 68XX Family April 1978.

[MIL 74] Dept. of Defence
Military standard.handbook : Reliability Prediction of Electronic Equipment
Septembre 74.

[WENS 72] J. H. WENSLEY
S.I.F.T. : Software Implemented Fault Tolerance
Proceedings of the 1972 Fall Joint Computer Conference A.F.I.P.S., V 41 Part 1 pp 243-253

REFERENCES BIBLIOGRAPHIQUES : ANNEXE I

[BOUR 71] W. G. BOURICIUS & al.
Reliability modeling for fault-tolerant computers
IEEE Transactions on computers, Vol C-20, n°11, pp1306-1311, novembre 1971.

[CHEN 78] L. CHEN & A. AVIZIENIS
N-version programming: a fault-tolerance approach to reliability of software operation
Proceedings of The Eighth Annual International Conference on Fault-Tolerant Computing, Toulouse, France, juin 1978.

[COURT 76] B. COURTOIS
Etude d'un calculateur tolérant des pannes: ses fiabilité, sécurité, performance et côut
Thèse de docteur-ingenieur, 18 decembre 1976, INP Grenoble.

[LAPRI 75] J. C. LAPRIE
Prevision de la sureté de fonctionnement et architectures de structures numeriques temps réel réparables
Thèse de Doctorat d'Etat soutenue le 16 Juin 1975 à l'Université Paul Sabatier Toulouse .

[PREP 67] F. P. PREPARATA, G. METZE & R. T. CHIEN
On the connection assignement problem of diagnosable systems
IEEE Transactions on Computers, Vol EC-16, n°6, pp848-854, decembre 1967.

REFERENCES BIBLIOGRAPHIQUES : ANNEXE 3

[AVIZ 77] A. AVIZIENIS
Fault Tolerant Computing - progress, problems and prospects
Proceedings of the 1977 I.F.I.P. Congress, Toronto
North-Holland Publishing Company, pp 405-420.

[BELLO 77] C. BELLON
Etude de la dégradation progressive dans les systèmes répartis
Thèse de 3^{iéme} Cycle, INP Grenoble, 14 Septembre 1977.

[CRIST 79] F. CRISTIAN
A recovery mechanism for modular software
Rapport de recherche I.M.A.G., RR-146, Janvier 1979.

[CROCU 75] CROCUS
Systèmes d'exploitation des calculateur
DUNOD Ed., Paris 1975.

[DAHL 70] O. J. DAHL, B. MYRHAUG & K. NYGAARD
SIMULA 67, The Common Base Language
Pub. S-22, Norvegian Computing Center 1970.

[DARON 78] P. DARONDEAU
Types et objets dans un système multi-langages
Thèse de Doctorat ès Sciences Mathématiques soutenue le 9 Mars 1978, USMG-INP Grenoble.

[DESH 77] P. DESHIZEAUX & P. LADET
Programmation en temps réel des synchronisations par gestion des liens événements-processus
Comptes rendus des journées d'étude sur les méthodes de synchronisation et de programmation globales en temps réel
A.F.C.E.T., Paris, Novembre 1977.

REFERENCES BIBLIOGRAPHIQUES : ANNEXE 3 (suite)

[HANS 78] P. B. HANSEN
The Architecture of Concurrent Programms
Prentice Hall, Englewood Cliffs, New Jersey, 1978.

[IBM] I.B.M. Corp.
OS/360 Assembler F Programmer's guide

[LEST 76] G. BETOURNE, L. FERRAUD, J. JOULIA & J. M. RIGAUD
Le langage d'écriture de systèmes "LEST"
Congrés AFCET, Paris 1976.

[MOAL 76] M. MOALLA
L'approche fonctionnelle dans la vérification des systèmes informatiques. Proposition de quelques méthodologies
Thèse de Docteur-Ingenieur, Decembre 1976, INP Grenoble.

[MONT 77] J. MONTUELLE & M. LUCAS
Conception modulaire des systèmes d'exploitation
Thèse de Docteur-ingenieur et de 3ième Cycle, 27 Juin 1977, USMG-INP Grenoble .

[MOSS 77] J. MOSSIERE
Méthodes de conception des systèmes d'exploitation
Thèse de Doctorat d'Etat, USMG-INP Grenoble, Septembre 77.

[ROBA 75] C. ROBACH
Méthodologie de test de processus: impact sur la conception
Thèse de Docteur-Ingenieur, Juin 75, USMG-INP Grenoble.

[SESAM 76] J. L. CHEVAL, F. CRISTIAN, S. KRAKOWIAK, M. LUCAS, J. MONTUELLE & J. MOSSIERE
Un système d'aide à l'écriture des systèmes d'exploitation
Congrés AFCET, Paris, 1976.

REFERENCES BIBLIOGRAPHIQUES : ANNEXE 3 (fin)

[VEILL] F. VEILLON & J. M. CAGNAT
Cours de programmation en langage PL/1
Collection U Serie Informatique,Armand Colin Ed.

[WIRTH 67] N. WIRTH
A programming language for the 360 Computer
Computer Sciences Department, School of Humanities and Sciences, Standford University, Technical Report N° CS53
20 December 1966, Revised 20 June 1967.

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 :

- Madame G. SAUCIER, Maître de Conférences à l'Institut National Polytechnique de GRENOBLE

- Monsieur A. COSTES, Maître de Conférences à l'Institut National Polytechnique de TOULOUSE

Monsieur Jacques P U L O U

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

Grenoble, le 12 Septembre 1979

Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique