MAS : réalisation d'un langage d'aide à la description et à la conception des systèmes logiques

Marianthi Zachariades

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE
Génie informatique

Marianthi ZACHARIADES

MAS : REALISATION D'UN LANGAGE D'AIDE A LA DESCRIPTION ET A LA CONCEPTION DES SYSTEMES LOGIQUES.

Thèse soutenue le 14 septembre 1977 devant la Commission d'Examen ;

Président : L. BOLLIET

Examinateurs : M. BLANCHARD
F. BOERI
M. CAPLAIN
G. SAUCIER

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Monsieur Philippe TRAYNARD : Président

Monsieur Pierre-Jean LAURENT : Vice Président

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BLOCH Daniel	Physique du solide
	BONNETAIN Lucien	Chimie minérale
	BONNIER Etienne	Electrochimie et électrométallurgie
	BOUDOURIS Georges	Radioélectricité
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	DURAND Francis	Métallurgie
	FELICI Noël	Electrostatique
	FOULARD Claude	Automatique
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie-Physique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	POLOUJADOFF Michel	Electrotechnique
	SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M.	ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electrotechnique
	LACOUME Jean-Louis	Géophysique
	LANCIA Roland	Electronique
	ROBERT François	Analyse Numérique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Mathématiques appliquées
	CHARTIER Germain	Electronique
	GUYOT Pierre	Chimie minérale
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du solide
	MORET Roger	Electrotechnique nucléaire
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme.	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan	Automatique

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

MM.	FRUCHART Robert	Directeur de Recherche
	ANSARA Ibrahim	Maître de Recherche
	CARRE René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Je tiens à remercier :

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

Madame G. SAUCIER, Maître de Conférences à l'Institut National Polytechnique de Grenoble, qui m'a accueillie dans son équipe et a dirigé mon travail avec beaucoup de bienveillance et d'attention,

Monsieur le Professeur F. BOERI, du Laboratoire de Signaux et Systèmes de l'Université de Nice, et
Monsieur M. BLANCHARD, Directeur de Recherche au Département d'Etude et de Recherche en Automatique du CERT, qui ont bien voulu accepter de faire partie du jury,

Monsieur M. CAPLAIN, Ingénieur C.E.A. qui m'a beaucoup aidée de ses conseils, tant pour le développement de mon travail que pour la rédaction.

Je voudrais remercier aussi tous les membres de l'équipe "Conception et Sécurité des Systèmes Logiques" qui ont participé, par de nombreuses discussions, à l'amélioration de ce travail. En particulier, Messieurs M. MOALLA et J. SIFAKIS qui sont, pour une grande part, à l'origine de ce travail.

Enfin je remercie Madame G. DUFFOURD qui a assuré la dactylographie de ce mémoire avec dévouement, ainsi que le Service de Reprographie pour le soin qu'il a apporté à sa réalisation matérielle.

SOMMAIRE

C H A P I T R E I

MODELES DE CONCEPTION

INTRODUCTION

La conception de systèmes logiques (informatiques ou automatiques) de plus en plus complexes, pose des problèmes bien connus aux concepteurs, à savoir :

- difficulté d'obtenir un énoncé clair, non ambigü, complet, du problème initial, i.e. problème de rédaction d'un cahier des charges,

- difficulté de communication entre les différentes équipes réalisant le système,

- nécessité d'une description progressive des spécifications fonctionnelles,

- possibilité, à chaque étape, d'une vérification de conformité fonctionnelle aux spécifications initiales,

- possibilité de mettre en place des moyens de protection fonctionnelle contre certains incidents,

- choix rationnel et pouvant être justifié par des ressources matérielles lors de l'implémentation.

Nous proposons donc un outil pouvant aider le concepteur face à ces problèmes. Il s'agit d'un outil de description fonctionnelle permettant une description rigoureuse de spécification fonctionnelle. L'accent étant mis sur :

- la possibilité d'une description progressive,
- la possibilité de décrire des systèmes complexes répartis et à fonctionnement parallèle (entre les différents sous-systèmes et à l'intérieur d'un sous-système),
- la possibilité de vérification fonctionnelle (blocages, conflit, etc).

La simulation d'un système, ainsi décrit, permettra en outre une première évaluation de performances (description prenant en compte le facteur temps) et aidera le concepteur dans ses choix d'implémentation matérielle.

Notre travail a donc pour objet la réalisation d'un langage de simulation considéré comme un outil d'aide à la description et à la conception. Ce langage utilise la décomposition du système en Partie Contrôle et en Partie Opérative. On définira le modèle (RCPP) qui représente la Partie Contrôle (Chapitre I).

Dans le chapitre II on définira le langage; on présentera sa syntaxe, ses primitives, son implémentation.

On étudiera l'évaluation et l'amélioration des performances du langage ainsi que son utilisation en temps réel (Chapitre III).

On appliquera ce langage (Chap.IV) à :

- la modélisation d'un système téléphonique, le E10, et la simulation de son fonctionnement,
- la modélisation d'un système multiprocesseur travaillant en pipe-line.

On présente, en annexe un résumé des langages d'aide à la conception.

I - METHODOLOGIE DE CONCEPTION

La conception d'un nouveau système consiste, pour l'essentiel, à étudier les différentes possibilités de le réaliser en satisfaisant les spécifications fonctionnelles et les contraintes entre coût-performance-fiabilité-sécurité du système [1], [2].

On peut distinguer quatre phases principales dans la démarche du concepteur :

1) *Le choix d'une décomposition fonctionnelle du système.*

2) *Le choix d'une structure et du type de connexion entre les modules de cette structure.*

3) *La définition des protocoles de synchronisation et d'échange entre modules.*

4) *La conception du fonctionnement logique de chaque module ainsi que le choix d'une technologie pour leur réalisation matérielle.*

Chacune de ces phases exige une validation fonctionnelle et la justification des choix adoptés.

Dans la première phase, le concepteur doit vérifier la cohérence des algorithmes issus de cette décomposition, en ce qui concerne l'exécution des fonctions globales du système.

Dans la deuxième phase on devra justifier le choix de la structure compte-tenu des possibilités du parallélisme, mises en évidence lors de la première phase, des critères de sécurité et de performances. Le concepteur peut associer un temps approximatif d'exécution aux opérations effectuées par les différents modules et évaluer ainsi l'influence de la structure sur les performances globales du système. On évaluera, donc, non seulement l'influence de l'architecture du système mais aussi celle des algorithmes de priorité aux interfaces.

A la troisième phase, à la suite du choix des protocoles d'échanges, le concepteur peut évaluer, plus précisément, le comportement global du système, ainsi que l'effet de ce choix sur le fonctionnement de chaque module. Il peut également mettre en évidence des situations critiques, telles que conflits, "deadlocks", et "thrashing".

Enfin, dans la dernière phase, il doit valider les schémas logiques des modules du système et justifier le choix de la technologie par une évaluation précise du coût, de la fiabilité, des performances ...

Vu la complexité du problème de la conception, cette approche ne peut être entièrement "linéaire". Le concepteur devra décrire à nouveau certaines étapes afin de revenir sur des choix effectués antérieurement. Cette "itération" permettra l'ajustement de certaines caractéristiques en vue de l'obtention d'un meilleur compromis. La même méthode peut être appliquée, non seulement à la conception d'un système global à partir d'un ensemble de modules, mais aussi à chacun de ces modules qui sera alors défini comme une interconnexion de modules plus fins.

En raison de la grande complexité de cette tâche, le concepteur doit disposer des outils appropriés de modélisation et de simulation. Ces outils doivent lui permettre la description d'un système à des niveaux de détails différents afin d'assurer la cohérence de ses décisions.

Ces outils doivent posséder les caractéristiques suivantes :

- *Possibilité d'une description fonctionnelle* d'un système à des niveaux de détails différents. Ceci permettra au concepteur de préciser progressivement son système en utilisant le même outil dès sa définition globale jusqu'à sa réalisation matérielle.

- *Possibilité d'une description modulaire* et d'une représentation de la synchronisation et du parallélisme à l'intérieur des modules et entre modules. Pour cela le facteur temps doit être explicitement introduit.

- *Possibilité d'analyser le comportement du système*, de détecter des conflits et des blocages, de déterminer la sensibilité des performances aux variations de tel ou tel facteur.

Le système M.A.S. a été conçu comme un tel outil. Il permet la modélisation et la simulation des systèmes complexes répartis, c'est-à-dire des systèmes réalisés par l'interconnexion de modules évoluant en parallèle et communiquant entre eux de manière synchrone ou asynchrone.

Dans le but de satisfaire aux caractéristiques énoncées et pour répondre notamment au troisième objectif, la définition de M.A.S. a été basée sur un modèle formel de description des systèmes.

Un grand nombre d'études ont été amorcées dans le domaine des modèles et surtout des modèles de calcul parallèle.

Deux modèles développés indépendamment ont constitué la base d'un grand nombre d'études : les schémas de programmes parallèles [3] et les réseaux de Petri [4].

- *Le système LOGOS* [5], dérivé des schémas de programmes parallèles, est défini par un graphe de contrôle et un graphe de données. Il est présenté en tant que modèle pour la conception de systèmes digitaux.

- *Les graphes UCLA* [6] sont inspirés de plusieurs travaux antérieurs, notamment d'une extension des schémas de programmes parallèles, en ce qui concerne la représentation graphique, et des réseaux de Petri, en ce qui concerne l'utilisation des marques pour représenter l'état de contrôle. Ce modèle est proposé pour la modélisation des systèmes d'exploitation et peut servir de noyau à un langage de programmation parallèle.

- *Les schémas à réseau de Petri* [7] sont formés d'un graphe de commande, d'un graphe de données et d'une interprétation, le graphe de commande étant un réseau de Petri étiqueté. Ce modèle est proposé dans le cadre d'une étude pour la représentation, l'analyse et la validation des systèmes parallèles de commande de procédés industriels.

Tous ces modèles ont beaucoup de liens entre eux mais chacun est un peu plus adapté à un type de problème. On peut retenir que les deux modèles, proposés pour la description de systèmes logiques, utilisent la décomposition d'un système en partie opérative (P.O) et partie contrôle (P.C), décomposition qui sera, aussi, utilisée par notre modèle.

II - DECOMPOSITION D'UN SYSTEME : PARTIE OPERATIVE - PARTIE CONTROLE

Un système est défini par un triplet (D, S, T) où :

D est un ensemble d'objets appelés données

S est un ensemble d'objets appelés sorties

T : D → S est une application définissant le comportement du système, c'est-à-dire qu'elle permet d'associer à une donnée d, d ∈ D, une sortie T(d) (Fig.1).

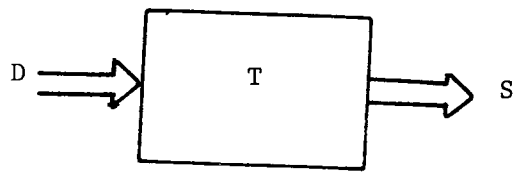


Figure 1.

Dans le cas où un système (D, S, T) est représenté à partir d'un ensemble d'opérateurs OP = $\{op_1, \ldots op_n\}$, l'application T est exprimée comme la composition des opérateurs de l'ensemble OP. Nous admettons que pour les systèmes considérés, T peut être décrite par un algorithme, à partir des éléments de l'ensemble OP.

Il est démontré que tout système, dont le fonctionnement est décrit sous forme d'algorithme, est décomposable en deux parties coopérantes (Fig. 2) [8].

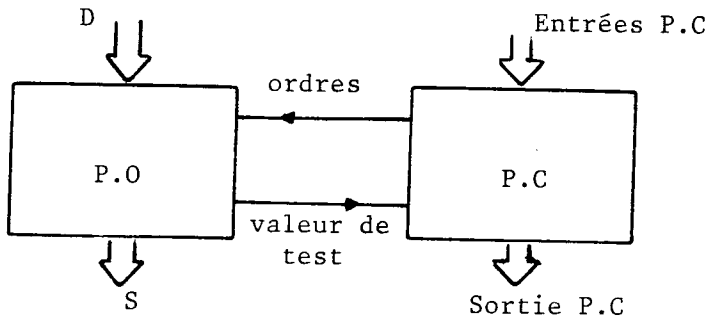


Figure 2.

- *La partie opérative (PO)* contient les données et les opérateurs qui sont utilisés, soit pour calculer des valeurs de test sur les données, soit pour les transformer.

- *La partie contrôle (PC)* est un automate qui gère les activations des opérations, dans la partie opérative, pour calculer l'application T. Quand l'algorithme représentant le système est calculable, alors la PC est un automate ayant un nombre fini d'états. A chaque état de la PC est associé un symbole de sortie qui est interprété par la PO comme l'ordre d'exécuter une action suivant un opérateur existant. La PC évolue d'un état à un autre état, en fonction des changements de ses entrées qui sont, soit des valeurs de test calculées dans la PO, soit des entrées externes provenant d'un opérateur humain ou d'un autre système.
  Un automatisme logique est un système discret dont la PO représente un processus physique.

La décomposition, en une PO et une PC, est assez facile à obtenir à partir d'une description algorithmique du fonctionnement du système.

III - MODELES FORMELS POUR LA DESCRIPTION D'UN SYSTEME

III - 1. Organigramme séquentiel

Un organigramme séquentiel est un réseau ayant 5 types de noeuds (Fig.3a)

- *Noeud initial* : il indique le point de départ unique à partir duquel on commence à exécuter l'algorithme représenté par l'organigramme.
- *Noeud opérateur* : quand on atteint l'arc entrant, on transforme les données suivant un opérateur op_i. Avec la fin de l'opération on atteint l'arc sortant. On suppose qu'une opération dure un temps non nul.
- *Noeud test* : les arcs sortants correspondent à des conditions portant sur les données. Ces conditions sont telles qu'une seule est vérifiée chaque fois. Quand on se présente à l'arc entrant, on atteint simultanément l'arc sortant (durée du test nulle).
- *Noeud disjonction* : quand on se présente à un arc entrant on atteint l'arc sortant avec un retard nul.
- *Noeud final* : il indique la fin de l'exécution de l'algorithme représenté par l'organigramme.

Les noeuds sont interconnectés de telle manière que :

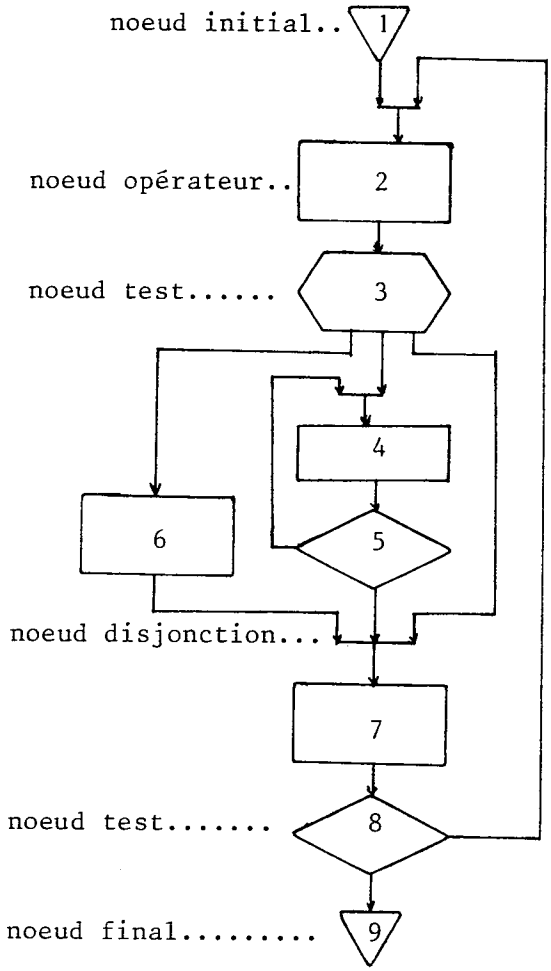
. chaque arc entrant soit connecté à un seul arc sortant et inversement,

. tout noeud puisse être atteint à partir du noeud initial.

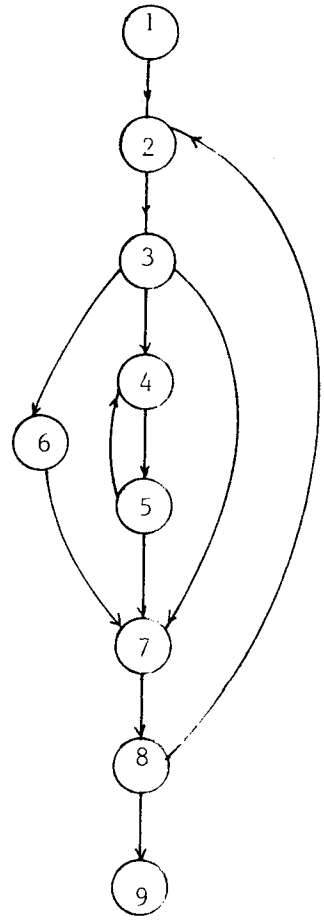
Obtention de la PC

Quand l'algorithme décrivant un système est donné sous forme d'organigramme séquentiel on peut obtenir, de façon directe, la PC sous forme de graphe d'état de la façon suivante (Fig. 3b) :

- A chaque noeud de l'organigramme du type initial, final, opérateur ou test, on fait correspondre un état de l'automate.
- Les sorties de l'automate sont associées aux états correspondant aux noeuds opérateurs de l'organigramme.
- On quitte un état, correspondant à un opérateur op_i, pour atteindre l'état correspondant au successeur unique de ce noeud, quand la PO signale l'accomplissement de l'opération op_i.



Organigramme séquentiel



Graphe d'Etat

(a) Figure 3. (b)

L'organigramme séquentiel est un réseau déterministe : on se trouve avec un seul noeud à un moment donné. Cette propriété est due au fait que l'on quitte toujours un noeud par un seul arc sortant et qu'il existe un seul noeud initial.

Ce modèle impose une description séquentielle des processus; il ne peut donc décrire de façon explicite le parallélisme possible lors de l'exécution d'un algorithme.

III - 2. Organigramme parallèle

Un organigramme parallèle est un réseau ayant 7 types de noeuds possibles.

- Les 5 types de noeuds de l'organigramme séquentiel.
- *Noeud convergence (JOINT)*: on atteint avec un retard nul l'arc sortant ssi on est présent à tout arc entrant (Fig.4).

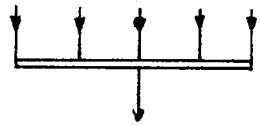


Figure 4.

- *Noeud divergence (FORK)* : on atteint tous les arcs sortants dès que l'on est présent à l'arc entrant (Fig.5).

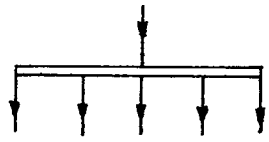


Figure 5.

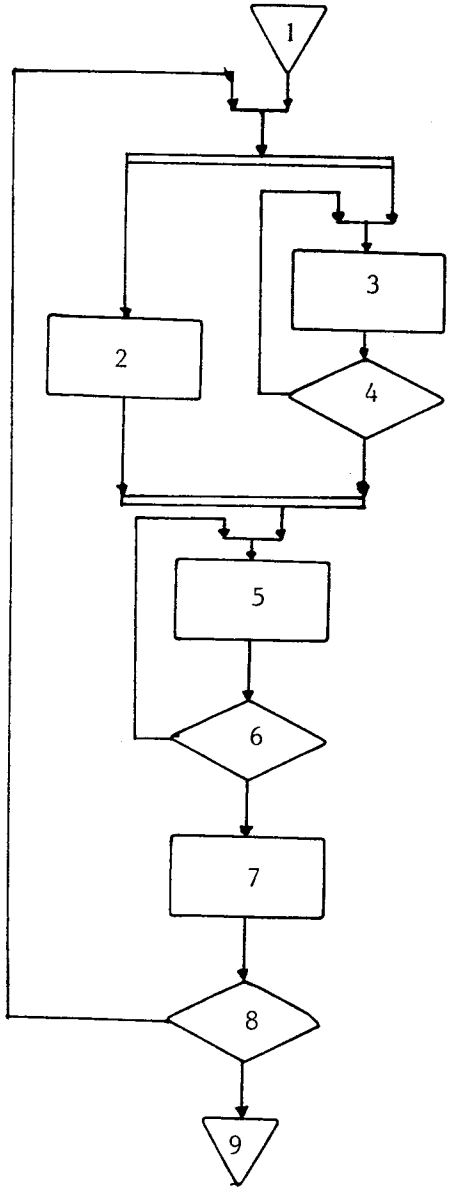
L'interconnexion des noeuds s'effectue de la même manière que pour l'organigramme séquentiel. Mais l'organigramme parallèle peut décrire grâce à ses deux noeuds supplémentaires, le parallélisme :

- *Le noeud divergence* à n arcs sortants permet le déclenchement de n-processus parallèles.
- *Le noeud convergence* peut être utilisé pour leur synchronisation.

Obtention de la PC

Les états de l'automate, représentant la PC d'un système décrit par un organigramme parallèle, ne correspondent plus aux noeuds de cet organigramme. Il faut rechercher toutes les configurations possibles des noeuds qui sont atteints à partir d'un noeud initial.

Organigramme parallèle



Réseau de Petri

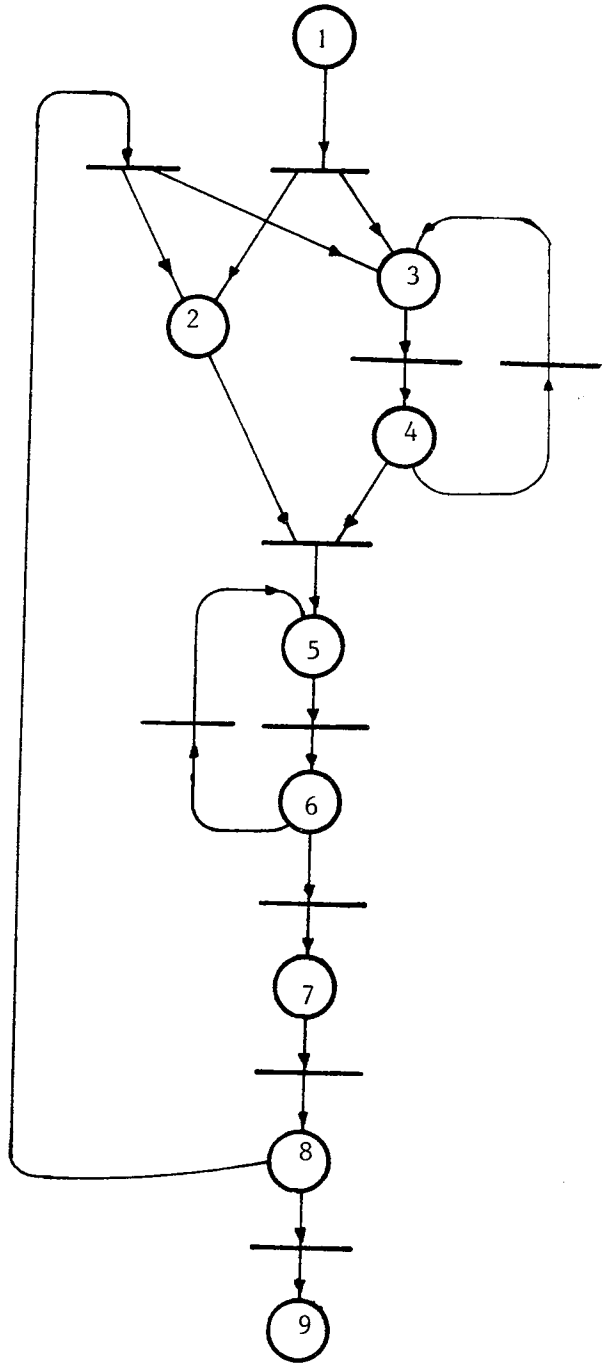


Figure 6.

La PC peut être obtenue directement sous forme de Réseaux de Petri (qui seront définis au §III.3), de la façon suivante (Fig.6) :

- A chaque noeud de l'organigramme du type initial, final, opérateur ou test, on associe une place du réseau de Petri.
- A chaque noeud divergence est associée une transition du même type dans le réseau de Petri.
- A chaque noeud convergence est associée une transition du même type dans le réseau de Petri.
- A un noeud disjonction on associe autant de transitions que d'arcs entrants(Fig.7).
- Les sorties de la PC sont associées aux places correspondant aux noeuds opérateurs.

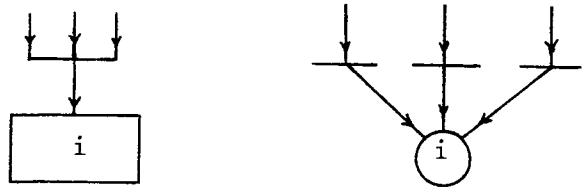


Figure 7.

III - 3. Le Réseau de Petri

Un Réseau de Petri (RdP) est un graphe orienté défini par un quadruplet {T, P, B, P°} avec :

$T = \{t_1, \ldots t_m\}$ un ensemble fini de transitions

$P = \{p_1, \ldots p_n\}$ un ensemble fini de places

$B = \{b_1, \ldots b_r\}$ un ensemble fini de branches (orientées) qui connectent soit une place à une transition, soit une transition à une place.

P° : ($p^\circ \subseteq p$) la distribution initiale de marques ; p° est le sous-ensemble des places qui contiennent initialement une marque.

Les transitions représentées par une barre — , et les places représentées par un cercle ◯, sont les noeuds du graphe. On définit, en plus, des objets appelés marques pouvant se déplacer dans le réseau. Une marque est représentée par un point à l'intérieur d'une place.

Une place peut contenir des marques. On l'appelle pleine si elle en a au moins une, et vide si elle n'en a pas.

Une place est une place d'entrée d'une transition si il existe un arc orienté de la place vers la transition. De façon analogue on définit une place de sortie d'une transition si il existe un arc orienté de la transition vers la place.

Dans un tel réseau chaque place représente une condition et chaque transition représente un évènement.

Règle d'évolution des marques

- Une transition est validée ssi toutes ses places d'entrée sont pleines.
- Toute transition validée est mise à feu. Ceci consiste à enlever une marque de chaque place d'entrée et à mettre une marque à chaque place de sortie (Fig. 8).

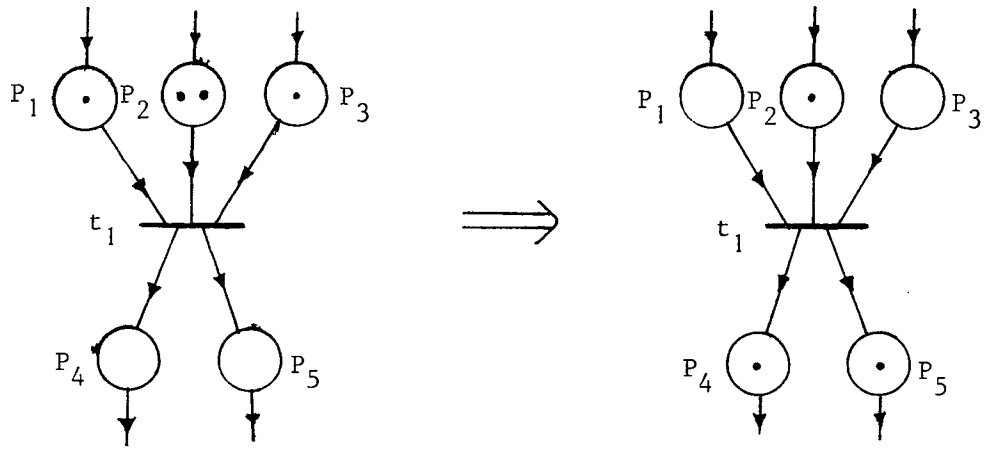


Figure 8.

Un conflit se produit quand deux ou plusieurs transitions validées se partagent la même place d'entrée (Fig.9).

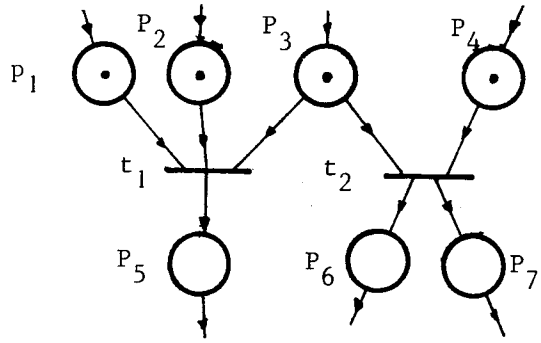


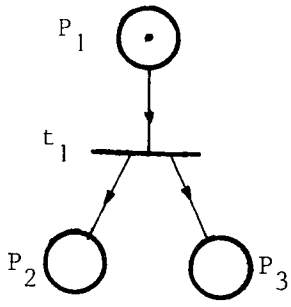
Figure 9.

La règle d'évolution dans un RdP (décrit plus haut) ne permet que la mise à feu d'une des transitions, les autres étant invalidées.

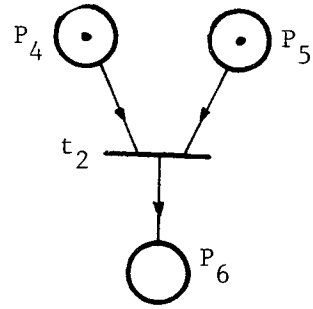
Un RdP est sauf si à aucun moment une place ne peut contenir plus qu'une marque.

Les RdP, par leur définition permettent de représenter très facilement le parallélisme et la synchronisation des évènements concurrentiels.

Le noeud divergence de l'organigramme parallèle est décrit par la transition de la figure 10a et le noeud convergence par celle de la figure 10b



FORK



JOINT

(a) Figure 10. (b)

Avantages et inconvénients des RdP

Les RdP, par rapport aux graphes d'états, offrent des avantages évidents. De plus, les RdP, contrairement aux organigrammes parallèles, permettent de représenter de manière graphique un sémaphore. Chaque sémaphore sera représenté alors par une place du réseau et sa valeur par le contenu de la place.

Ainsi, les RdP permettent :

. de traduire des évolutions parallèles,
. de représenter de façon aisée la synchronisation entre plusieurs processus,
. de détecter les conflits entre processus qui se partagent la même ressource,
. d'étudier les conditions de déterminisme et de sécurité d'un système.

Mais ce formalisme peut devenir un peu lourd par la traduction de certains mécanismes de synchronisation.

L'idée d'introduire aux transitions des conditions, qui seront fonctions du marquage et du non marquage des places, a conduit à la définition d'autres modèles tels que l'organiphase [9] et les réseaux de contrôle de processus paralleles [10].

Enfin, le RdP général défini ainsi ne peut décrire que des processus autonomes, c.a.d. des processus dont l'évolution ne dépend pas de l'environnement extérieur.

Certaines extensions de ce modèle permettent de mieux adapter son utilisation à la modélisation des systèmes réels :

. Prise en compte des évènements externes,
. Prise en compte du facteur temps,
. Définition d'une interprétation.

Exemple : Nous présentons, sur un exemple, l'utilisation des RdP. La Fig.11 montre le RdP correspondant à la synchronisation de deux processus producteur-consommateur avec un tampon unique.

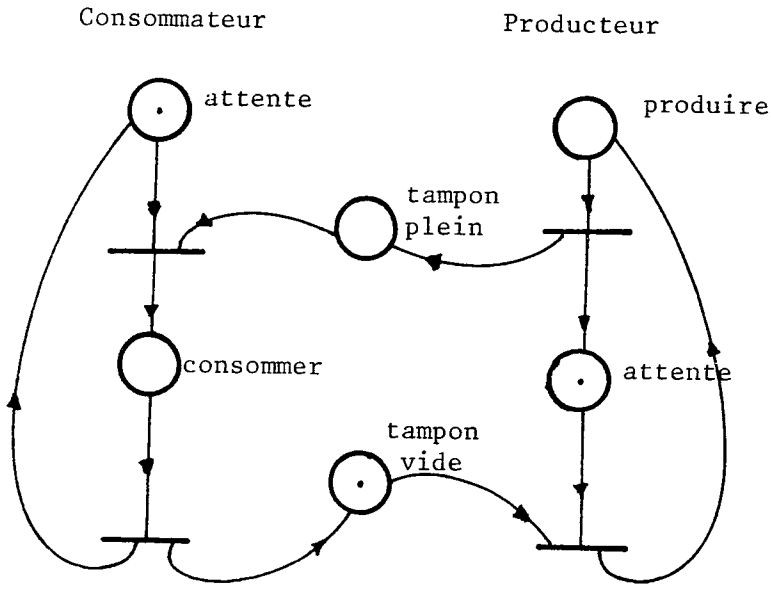


Figure 11.

III - 3.1. Réseau de Petri étiqueté - temporisé

L'influence de l'environnement extérieur sur l'évolution d'un processus peut être décrite par un *RdP étiqueté* Un RdP étiqueté est représenté par un RdP dont chaque transition t_i porte un évènement $\mu(t_i)$ pris dans un ensemble d'évènements externes.

Dans un tel réseau, la mise à feu, d'une transition validée, ne peut être réalisée qu'au moment de l'occurrence de l'évènement associé.

Les évènements externes peuvent représenter des variations des variables d'entrées qui arrivent, soit sur les impulsions d'une horloge (évolution synchrone), soit à des intervalles variables (évolution asynchrone). Un RdP étiqueté est capable de décrire un automate.

Un *RdP temporisé* est un RdP où à chaque place p_i est associé un temps pendant lequel une marque qui arrive reste *indisponible* pour toute évolution. Dans le cas général, ce temps n'est pas une constante mais une fonction de la place p_i et du temps d'arrivée de la marque.

Partant de ces deux extensions de RdP on peut définir un RdP étiqueté - temporisé. Ce modèle permettra, d'une part la prise en compte des évènements externes et, d'autre part, la temporisation de son évolution.

III - 3.2. Réseau de Petri interprété (RdPI)

Enfin, on définit un modèle apte à décrire le fonctionnement de tout système. On l'appelle RdPI et est défini par un RdP étiqueté -temporisé, une P.O et des applications qui associent les éléments de la P.O aux places et aux transitions du réseau.

Ainsi,

- *a chaque place est associé un opérateur dans la P.O qui s'active par l'arrivée d'une marque dans cette place,*
- *à chaque transition est associée une condition sur les valeurs des données de la P.O.*

III - 4. Les Réseaux de Contrôle des Processus Parallèles (RCPP)

Un Réseau de Contrôle de Processus Parallèles (RCPP) est défini par la donnée d'un quintuplet (X, P, Q, f, P_o) avec :

$X = \{x_1, x_2, \ldots x_n\}$ un ensemble fini des variables booléennes appelées variables d'entrées

$P = \{p_1, \ldots p_m\}$ un ensemble fini d'objets appelés phases

$Q = \{q_1, \ldots q_m\}$ un ensemble fini des variables booléennes en bijection avec les phases

f : une application de $P \times P \rightarrow F(Q, X)$, où $F(Q, X)$ est l'ensemble des fonctions booléennes simples de Q et de X telles que $f(p, p) = 0$

P_o : un ensemble de phases initiales contenu dans $P (P_o \subseteq P)$

A un RCPP peut être associé, de façon biunivoque, un réseau dont les noeuds s'identifient avec les phases et où la branche (p_i, p_j) porte la fonction $f(p_i, p_j)$, que l'on désignera par f_{ij}.

Règles d'évolution (Fig.12)

- Une phase peut se trouver dans l'un des deux états : *actif* ou *inactif*. Toutes les phases de l'ensemble P_o sont initialement actives. Une phase p_i est active à l'instant t si et seulement si $q_i(t) = 1$.

- Une phase p_j est activable ssi il existe une phase au moins p_i active et telle que $f_{ij} = 1$.

- Si à un instant t une phase p_j est activable, elle est activée et au même instant toutes les phases p_i telles que $f_{ij}(Q(t), X(t)) = 1$ sont désactivées.

- Toutes les activations et désactivations possibles à un instant t se font simultanément.

- Quand une phase active se désactive et s'active simultanément, elle reste active.

Une phase est représentée par un cercle. Un point à l'intérieur du cercle signifie que la phase est active.

Une phase *source* est une phase qui est toujours active. Elle est représentée par un carré (Fig.13 a).

(a) (b)

Figure 13 .

Règles d'évolution

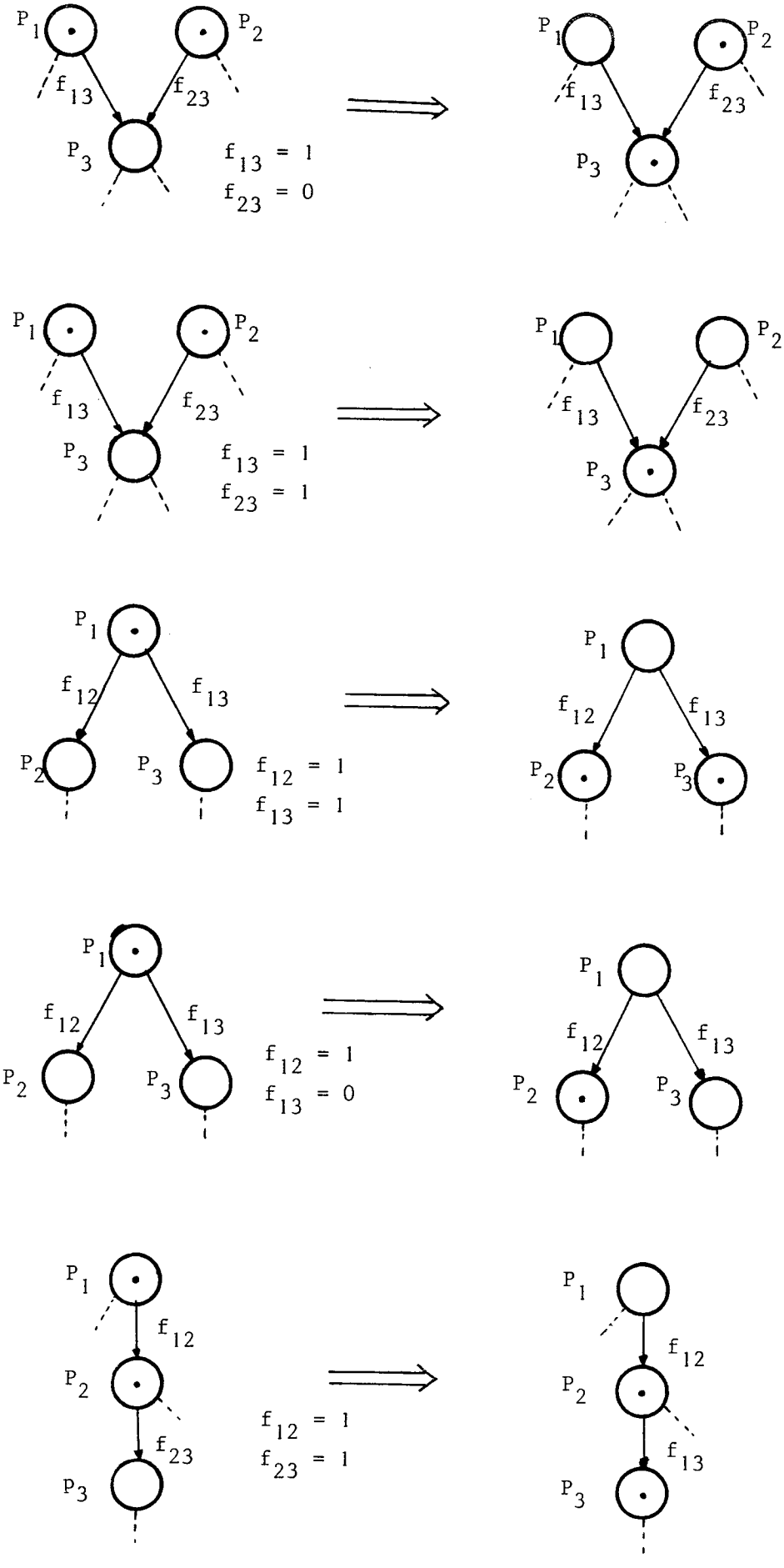


Figure 12.

De même, il est souvent utile de pouvoir exprimer des conditions de désactivation définies à l'aide d'une fonction combinatoire. C'est pourquoi il est permis, dans la représentation graphique, d'utiliser des arcs n'aboutissant à aucune phase (Fig. 13 b). La validation d'une transition de ce type a comme conséquence la désactivation de la phase d'entrée.

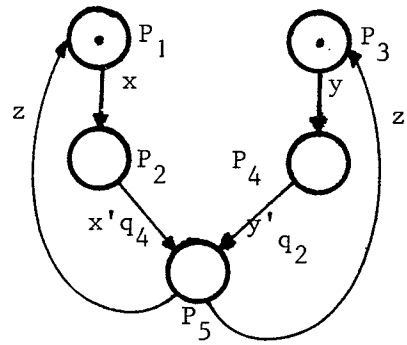
Exemple : La Fig.14b montre un exemple d'un RCPP défini par :

$X = \{x, y, z\}$, $P = \{p_1, p_2, p_3, p_4, p_5\}$, $Q = \{q_1, q_2, q_3, q_4, q_5\}$,

f, $P_o = \{p_1, p_3\}$ où f est donné au tableau de la Fig. 14 a.

i \ j	p_1	p_2	p_3	p_4	p_5
p_1	0	x	0	0	0
p_2	0	0	0	0	$x'q_4$
p_3	0	0	0	y	0
p_4	0	0	0	0	$y'q_2$
p_5	z	0	z	0	0

(a)



(b)

Figure 14.

Tout RdP étiqueté *sauf* est représentable par un RCPP en associant une phase à chaque place (une phase est active ssi la place correspondante a une marque) et en explicitant les conditions imposées par le fonctionnement.

En réalité, un RdP étiqueté sauf est un RCPP qui a comme étiquette sur les transitions, des fonctions de la forme $\mu(t_j) \prod_{j \in J} q_j$, où J est un sous-ensemble de {1, 2... m .}

Toutes les possibilités offertes par les RdP y sont inclues comme le montre la Fig.15.

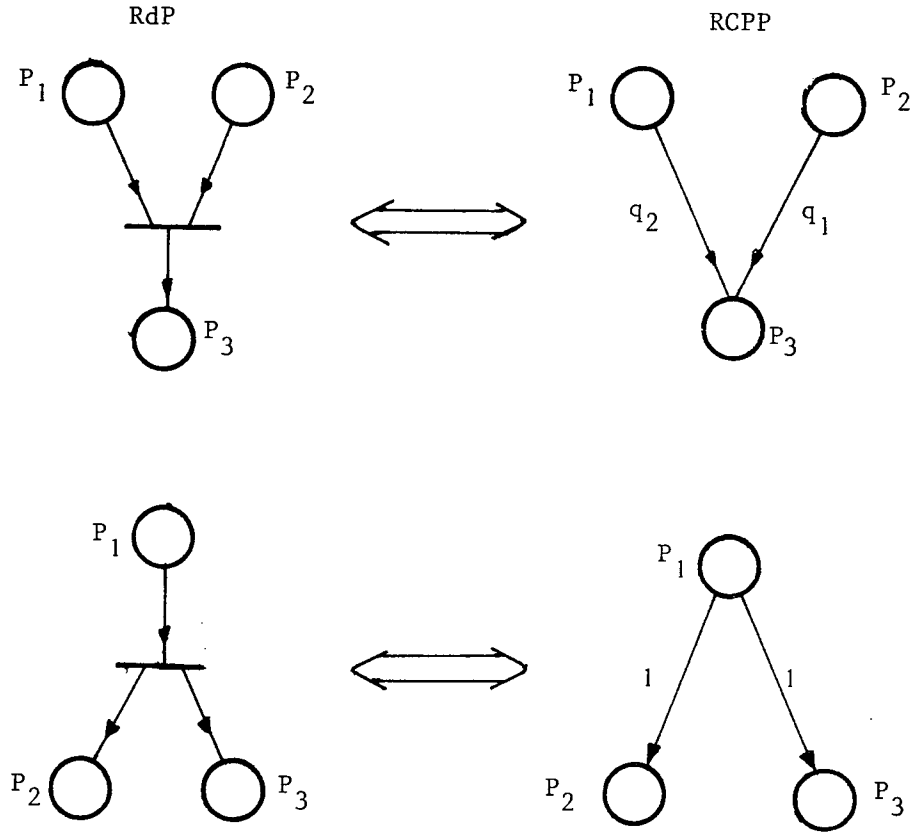


Figure 15.

Avantages et inconvénients des RCPP

Les avantages de RCPP par rapport aux graphes d'état sont les mêmes que ceux des RdP.

Les RCPP permettent, en plus, une description moins contraignante et éventuellement plus concise que les RdP. Ceci peut être illustré par un exemple simple :

Soit à représenter l'évolution suivante : "Une marque arrive à une place p_1, si une marque se trouvait à l'instant précédent en p_2 et p_3 mais pas en p_1".

Pour décrire cette évolution par un RdP, il faut créer une place p_1' "complémentaire" de la place p_1; p_1' sera une place qui aura une marque ssi

p_1 n'en a pas. Ceci conduit à l'augmentation du nombre de places par rapport au nombre de phases du RCPP qui représente la même évolution (Fig.16). De plus, la place p'_1 doit être une place d'entrée de toute transition menant à p_1 et une place de sortie de toute transition ayant p_1 comme place d'entrée.

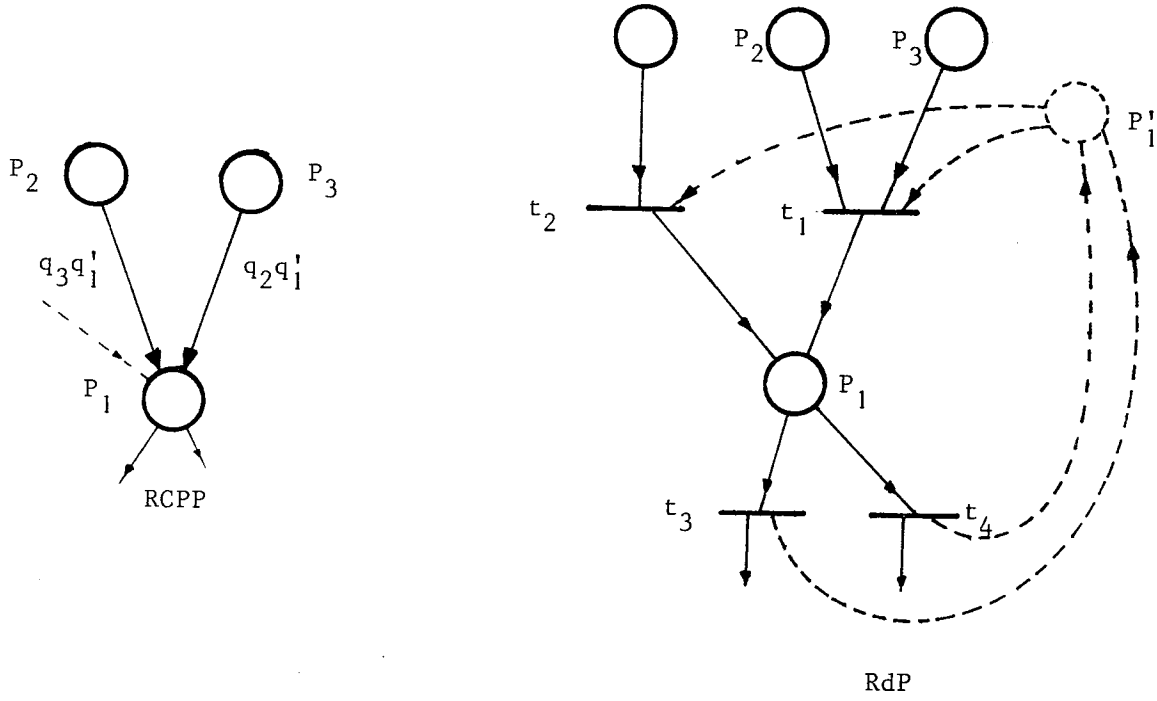


Figure 16.

C'est la même chose pour la lecture du contenu d'une place sans modification de son contenu (Fig.17).

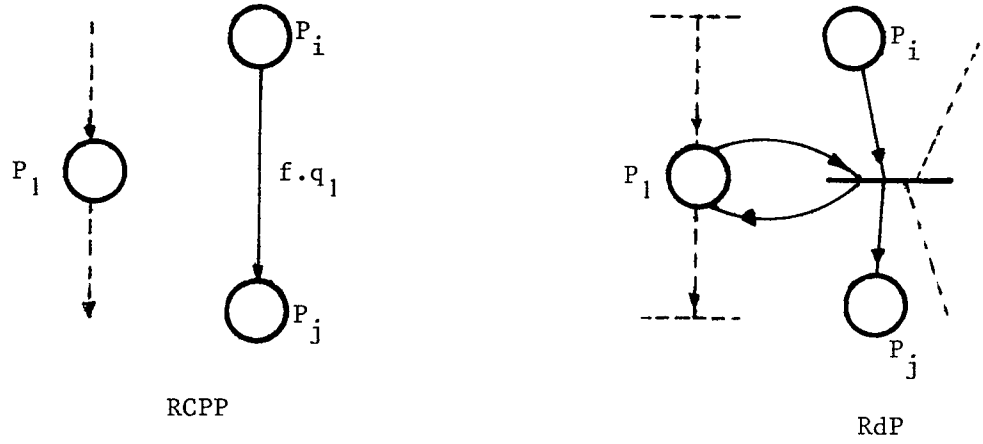


Figure 17.

Par contre, un inconvénient certain des RCPP est que des situations critiques, telles que les blocages et les conflits, sont plus difficiles à localiser que dans un RdP (Fig. 18). Le masquage des conflits est possible par l'utilisation des fonctions f_1 et f_2 qui peuvent être assez complexes.

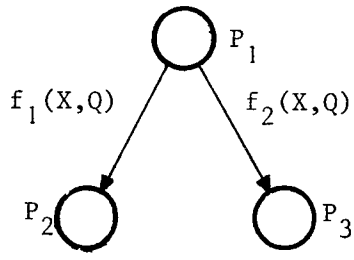


Figure 18.

III - 4.1. Représentation mathématique d'un RCPP

Etant donné que chaque phase correspond à un élément mémoire où on écrit un "1" si :

$$\sum_{j=1}^{n} f_{ji}\, q_j = 1 \qquad \text{(Fig. 19)}$$

et on l'efface si :

$$q_i \left(\sum_{k=1}^{n} f_{ik} \right) = 1 \quad \text{et} \quad \sum_{j=1}^{n} f_{ji}\, q_j = 0$$

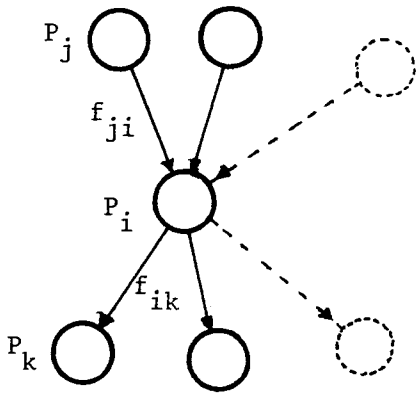


Figure 19

On peut représenter un RCPP par un système de n equations :

$$\{Q_i = \left(\sum_{i=1}^{n} f_{ji}\, q_j \right) + q_i \left(\overline{\sum_{k=1}^{n} f_{ik}} \right) \}_{i=1}^{n}$$

où $Q_i = q_i(t + \Delta t)$

et n : le nombre de phases dans le réseau.

A partir de ce système on peut arriver à une synthèse automatique par des méthodes classiques ou au contraire à une synthèse synchrone directe.

III - 4.2. Description d'un système global à l'aide d'un RCPP

Les RCPP définis ainsi sont suffisamment généraux pour permettre une description naturelle d'un automate de contrôle de processus. Partant de la définition initiale des RCPP nous introduirons certaines extensions, d'une façon analogue aux RdP [2], qui permettront de définir un outil pour la description de tout système logique décomposé en PO-PC.

III - 4.2.1. Introduction de la notion de temps - RCPP temporisé

Définition

Un RCPP temporisé est défini par :

- un RCPP (X, P, Q, f, p_o)
- un ensemble T ($T \subseteq R$, R ensemble de réels) complètement ordonné, appelé base de temps
- une application $\nu : P \times T \to T$ telle que si $\nu(p, t_i) = t_j$ alors $t_j \geq t_i$

Un RCPP temporisé est représenté par le RCPP associé, en indiquant en plus sur chaque phase p l'application $\nu(p, t)$ (Fig.20).

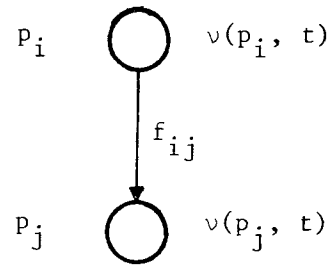


Figure 20.

Règles d'évolution

Aux règles d'évolution données pour un RCPP il faut ajouter en plus la suivante :

- une phase p reste bloquée à l'état actif durant l'intervalle de temps entre l'instant de son activation t et l'instant $\nu(p, t)$.

Ceci signifie que la phase p n'est pas prise en compte, pendant ce temps dans la recherche des phases susceptibles d'évolution.

L'association de la notion de temps aux phases du réseau, suppose qu'aucune durée n'est associée aux opérations d'activation ou de désactivation.

III - 4.2.2. RCPP Interprétés

Définition

Un RCPP interprété est défini par :

- une partie opérative (D, OP, C) telle que :
  - $D = \{d_1, d_2, \ldots d_u\}$ est un ensemble fini de variables de données
  - $OP = \{op_1, op_2, \ldots op_v\}$ est un ensemble fini d'opérateurs qui effectuent des transformations sur les données
  - $C = \{c_1, c_2, \ldots c_r\}$ est un ensemble de conditions (prédicats définis dans {0,1}) sur les valeurs des variables de données.
- Un RCPP temporisé (X, P, Q, e, P_0, T, ν) où l'on remplace la fonction f par :
  - e : une application de $P \times P \rightarrow F(Q, X, C)$, F étant l'ensemble des fonctions booléennes simples de Q, de X et de C.
- une application $\mu : P \rightarrow OP$

Un RCPP interprété est représenté par le RCPP temporisé associé, en indiquant, en plus, sur chaque phase p_i l'opérateur $\mu(p_i)$ et sur chaque branche (p_i, p_j) la fonction $e(p_i, p_j)$, désignée par e_{ij} (Fig. 21).

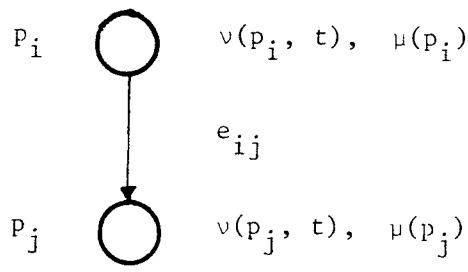


Figure 21.

Règles d'évolution

L'interaction entre les deux parties PC-PO provoque des modifications aux règles d'évolution définies au § III.4.

- une phase p_j est activable ssi il existe une phase au moins p_i active et non bloquée (Cf. III-4.2.1) telle que $e_{ij} = 1$,
- à l'instant t de l'activation d'une phase p, on active l'opérateur $\mu(p)$. La phase reste bloquée à l'état actif jusqu'à l'instant $\nu(p, t)$,
- si à un instant t une phase p_j est activable elle est activée et au même instant toutes les phases p_i actives et non bloquées, telles que $e_{ij} = 1$ sont désactivées.

Nous pouvons décrire un système global à l'aide d'un RCPP interprété. Le système sera constitué par une Partie Opérative et une Partie Contrôle; la communication entre les deux parties sera assurée par les deux applications μ et e associées respectivement aux phases et aux branches du RCPP.

Dans une telle description :

- le passage d'une phase à l'état actif déclenche l'exécution de l'opération associée dans la P.O,
- la fonction $\nu(p,t)-t$ représente, soit la durée de l'opération associée à la phase p, soit un délai simple (sans opération),
- pour les phases p_i de synchronisation, auxquelles ne correspond aucune action dans la P.O, l'opérateur $\mu(p_i)$ correspond à l'opérateur identité (son action n'a aucun effet sur les variables de la P.O) et la valeur de la fonction $\nu(p_i,t)$ est égale à t, c'est-à-dire la durée de l'activation est nulle,
- une phase p_i active peut provoquer une évolution après l'instant $\nu(p_i, t)$ (l'exécution de l'opération associée est terminée) et quand une transition au moins (p_i, p_j) porte une condition e_{ij} vraie,

- à une phase source (Cf. §III.4), on associe l'opérateur identité et un temps nul. Une telle phase peut influencer le comportement du système dès qu'une condition est vérifiée, quel que soit le niveau d'évolution du système. C'est le cas, par exemple, pour tester l'arrivée des entrées externes : demande d'une ressource, déclenchement des alarmes etc..

REMARQUE

On constate que les règles d'évolution dans un RCPP simple, et par conséquent dans un RCPP interprété, sont completes et sans ambiguité; ainsi, tous les cas de bon fonctionnement sont prévus, et l'unicité des configurations dans un RCPP est assurée (pour une configuration initiale et pour un système donné).

Par simulation des RCPP interprétés nous pouvons détecter des erreurs de conception, et le fonctionnement erroné d'un système.

Description multiniveau

En utilisant les RCPP interprétés on peut faire une description progressive d'un système.

A un niveau assez haut, des détails inutiles pour le fonctionnement global du système peuvent être omis de la PC, et transmis dans la P.O où les opérateurs effectueront des opérations complexes.

En descendant progressivement aux niveaux les plus bas, chaque phase pourra être éclatée à un sous-réseau qui décrira l'opérateur initial. Au niveau le plus bas on associe aux phases du réseau de contrôle des opérateurs élémentaires (transfert de registres).

La figure 22 montre le passage d'un niveau à un autre niveau inférieur par le remplacement de la phase P_3

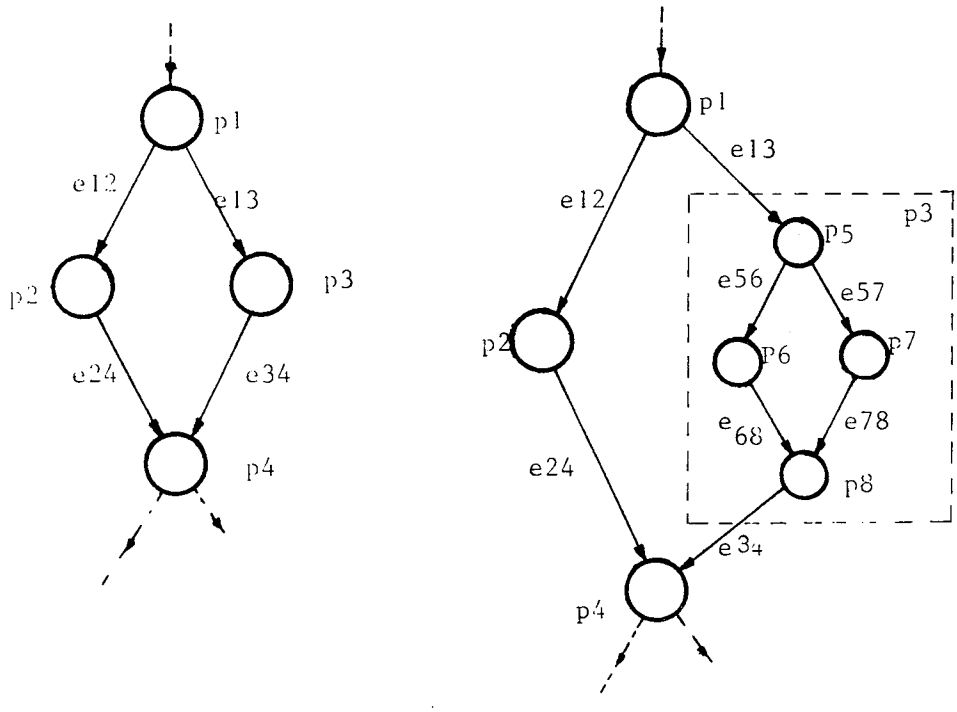


Figure 22.

Choix du modèle

Nous avons adopté les RCPP interprétés comme modèles de représentation d'un système, et ils constituent la base pour la définition du langage M.A.S.

Ce choix a été orienté par le fait que le RCPP semble plus facile à simuler que les RdP. En effet, dans une description avec un RdP on doit détailler tous les points de synchronisation. Dans une description qui utilise les RCPP beaucoup de problèmes délicats sont absorbés dans les expressions portées sur les branches.

Les RdP restent un meilleur outil pour une description graphique.

CHAPITRE II

LE LANGAGE MAS

I - ASPECTS METHODOLOGIQUES

Dans l'optique de la méthodologie de conception présentée dans le 1er chapitre, nous proposons un outil logiciel M.A.S.* destiné à faciliter la description, la simulation et la validation fonctionnelle d'un système logique. Ce langage permettra, en particulier, de décrire les structures de contrôle par des primitives très simples et rendra plus claire la démarche du concepteur, [11], [12], [13], [14].

Pour répondre aux objectifs, fixés au précédent chapitre, le langage possède les caractéristiques suivantes :

a) *Modularité* : La démarche adoptée est la description d'un système, en le décomposant en plusieurs modules fonctionnels. Des primitives du langage permettent :

- la définition des modules-types (DCLMOD) qui constitueront une bibliothèque de modules,
- la création, à partir de cette bibliothèque, d'un nombre arbitraire d'exemplaires d'un même module type (CREE) et,
- l'interconnexion des modules composant un système au moyen de variables d'interface (LIE).

b) *Séparation Partie Contrôle (PC) - Partie Opérative (PO)* : Chaque module du système est un RCPP interprété. La P.C est décrite par un langage simple non-procéduriel (Cf. Annexe) et la P.O est décrite par un ensemble de procédures, chacune représentant le fonctionnement d'un opérateur op_i (Cf. Chap.1). APL a été choisi comme le langage algorithmique pour la description des opérateurs [15]. Ce choix est justifié par :

. La richesse de ses opérateurs
. Sa notation très concise
. La facilité de son utilisation pour une description multiniveau

---

* Multilevel Assisted Design System

c) *Description fonctionnelle multiniveau* : Etant donné que le langage est basé sur les RCPP interprétés, l'utilisateur a la possibilité de décrire le fonctionnement d'un système à des niveaux de détail très divers. Ainsi, selon le niveau de la description, les procédures de la P.O peuvent représenter une action simple (une affectation, un délai..) ou un programme entier.

d) *Prise en compte du temps* : Dans la définition des RCPP interprétés, la notion du temps a été attribuée aux phases du réseau par l'application ν (Cf. Chap. I). Dans le langage, la fonction ν est spécifiée dans chaque procédure de la P.O (primitive TEMP), et elle est considérée comme la durée de l'opérateur correspondant. De plus, pour une meilleure description des systèmes réels, une temporisation est également autorisée dans la P.C. Cette temporisation est exrpimée par des prédicats temporels associés aux branches du réseau (alarmes, temps critique etc..).

Ayant défini nos objectifs principaux, la démarche adoptée dans la conception de ce langage paraîtra clairement dans le paragraphe suivant.

II - PRESENTATION INFORMELLE DU LANGAGE

Dans les paragraphes suivants nous présentons les primitives et les instructions essentielles (caractéristiques) du langage. Une définition complète et stricte de sa syntaxe est donnée par la suite.

II - 1. Définition des modules types

Comme nous l'avons déjà dit, chaque programme doit contenir la définition d'un certain nombre de modules qui décrivent le système.

La déclaration d'un module-type se fait au moyen de l'instruction :

DCLMOD nomodule (varint1, varint2 (dim),...)

La primitive DCLMOD a comme paramètre le nom attribué au module, désigné par *nomodule* et la liste des variables qui constituent l'interface du module, soit avec d'autres modules, soit avec l'environnement externe du système. Les variables inclues entre parenthèses et séparées par des virgules sont, soit des variables simples, soit des variables indicées.

La définition d'un module-type contient : (Fig.1)

- La description de sa Partie Opérative
- La description de sa Partie Contrôle.

Toutes les variables d'un module sont locales à ce module, sauf les variables d'interfaces et les entrées externes de sa P.C.

Le mot-clé FINMOD indique la fin de la définition du module.

II - 2.1. Partie Opérative

La P.O est décrite par un ensemble de procédures; chaque procédure étant associée à une phase du réseau de contrôle et activée par la P.C.

Une procédure est composée par une séquence d'instructions, constituant le corps de la procédure, précédée par un en-tête de déclaration et suivie d'une instruction de clôture.

a) *L'en-tête de déclaration a la forme générale :*

DCLPROC nomproc (par1, par2,..)
TEMP <expression APL>

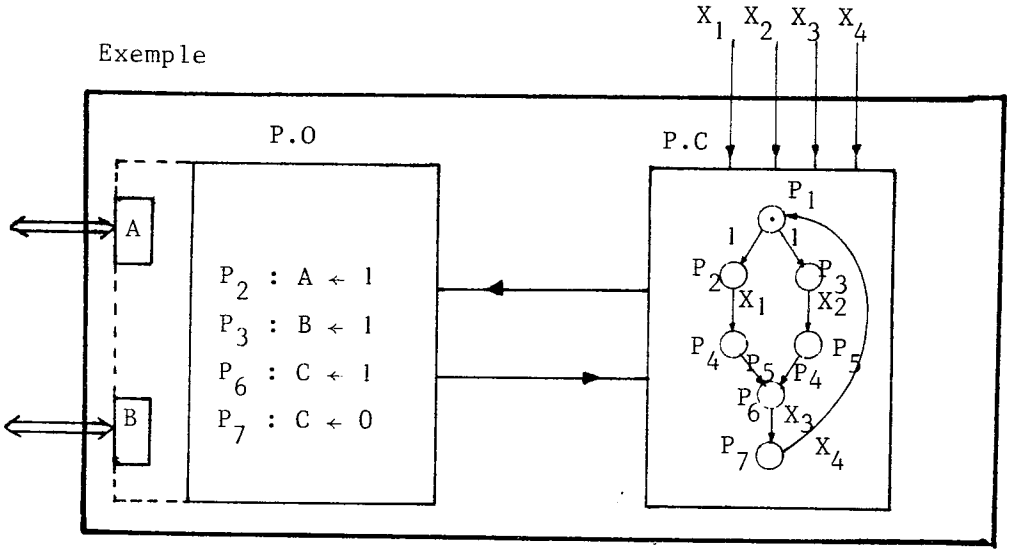
Le nom attribué à une procédure (*nomproc*) doit être forcément représenté par le même identificateur que la phase associée de la P.C.

La liste des paramètres est facultative et les variables *par1*, *par2*... sont des variables locales à la procédure, prenant leurs valeurs au moment de l'activation de la procédure.

TEMP est une primitive de temporisation. L'expression qui lui sert de paramètre est évaluée au moment de l'activation de la procédure et sa valeur indique le temps de simulation associé à l'exécution de la procédure.

b) *Le corps de la procédure est décrit en langage APL.* APL permet de décrire tout algorithme à partir des opérateurs primitifs. La structure et la syntaxe du corps d'une procédure sont identiques à celles utilisées pour décrire le corps d'une fonction définie en APL.

c) *Le mot-clé FIN signale la fin de la procédure.*




```
DCLMOD EXEMPLE (A, B)
DCLPROC P2
   TEMP 2
   A ← 1
FIN

DCLPROC P3
   TEMP 3
   B ← 1
FIN

DCLPROC P6
   TEMP 1
   C ← 1
FIN

DCLPROC P7
   TEMP 1
   C ← 0
FIN

DCLCTRL ASYN
ENTREES X1, X2, X3, X4
DEBUT
P1  si  1   ALORS  P2, P3
P2  si  X1  ALORS  P4
P3  si  X2  ALORS  P5
P4  si  P5  ALORS  P6
P5  si  P6  ALORS  P6
P6  si  X3  ALORS  P7
P7  si  X4  ALORS  P1

FINMOD
```

Figure 1.

II - 1.2. Partie Contrôle

Cette partie contient la description du RCPP correspondant. Un en-tête spécifie le mode de fonctionnement de la P.C pour la simulation ainsi que les entrées et les sorties éventuelles de la P.C. Le mot-clé DCLCTRL entame la définition de la P.C, SYN ou ASYN sont utilisés respectivement pour un mode de fonctionnement synchrone ou asynchrone.

DCLCTRL mode fonction
ENTREES X1, X2,...
SORTIES Y1, Y2,Y3...

X1, X2,.. et Y1, Y2,.. sont les variables utilisées pour désigner les entrées et les sorties de la P.C.

La déclaration des variables d'entrée, porte sur deux types de signaux différents : à niveau ou impulsionnels. On distingue les entrées à niveau, au moment de leur déclaration, par un caractère spécial qui suit le nom de la variable (astérisque).

par exemple : *X1, X2, X3*, X4*,...

La description proprement dite du RCPP (réseau de contrôle) est une séquence d'instructions, précédée par DEBUT et suivie par FINMOD qui désigne aussi la fin de la définition du module-type (voir Fig.1).

Instructions P.C

Un RCPP est un ensemble de phases liées entre elles par des arcs portant des expressions logiques (Fig.2) (voir Chap.I). La description d'un RCPP par un moyen autre que graphique est un ensemble de propositions de la forme :

"Si pi est active et non bloquée à un instant t et eij=1 alors pi est désactivée et pj est activée".

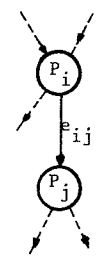


Figure 2.

Cette proposition est représentée par l'instruction de base [16].

pi Si eij alors pj

L'identificateur pi représente dans le programme la phase pi et aussi la variable booléenne qi associée.

L'expression eij peut être toute expression APL portant sur des variables définies soit en P.O, soit en P.C (des E/S ou des variables booléennes pi) et dont l'évaluation donne une valeur booléenne (0 ou 1).

Les cas d'une phase source (Fig.3a) et d'une simple désactivation avec des arcs n'aboutissant à aucune phase (Fig. 3(b), sont décrits par les deux instructions suivantes :

l si eli alors pi, pour une phase source

et pi si ei0 alors ∿pi, pour la désactivation d'une phase pi. ∿ est l'opérateur de négation d'APL appliqué sur une variable booléenne.

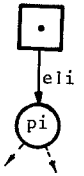


Figure 3 (a)

Figure 3 (b)

Pour faire des programmes plus compacts, la syntaxe du langage permet la description de plusieurs transitions dans une même instruction quand toutes ont la même phase d'entrée (Fig.4).

pi si eij1, eij2,... alors pj1, Pj2,...

Dans le cas où les fonctions d'activation eijk sont les mêmes, par exemple, cas d'un FORK, l'instruction peut être mise sous la forme :

pi si eij alors pj1, pj2,...

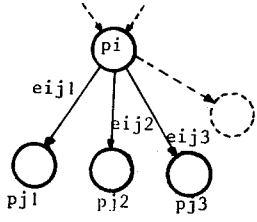


Figure 4.

Un autre type de syntaxe a été prévu pour décrire des situations plus particulières :

pi si eij1; eij2; eij3... alors pj1; pj2; pj3...

Cette instruction donne une priorité à une transition par rapport à une autre. La priorité est définie par l'ordre d'écriture, les premières transitions ayant une priorité plus grande que celles qui suivent. La transition

(pi, pjk) est validée si eijk est vraie et si toutes les conditions eijs pour S < K sont fausses.

Enfin, si des opérateurs associés à des phases différentes peuvent être décrits par le même algorithme, et par conséquent par une procédure paramétrée, les variables booléennes associées à ces phases sont des variables indicées.

Exemples :

1) L'instruction qui exprime les transitions multiples de la Fig.4, peut s'écrire :

pi si eij1, eij2,... alors pj(K)

La valeur de l'indice K correspond au rang de la première condition vérifiée (en comptant à partir de 1). Dans ce cas, une condition s'impose : les expressions eijk doivent être disjointes.

2) De même, une variable indicée pourrait remplacer les variables pik dans la Fig.5.

pi(K) si eij1, eij2,... alors pj

La valeur de K est déterminée par la phase pik, de l'ensemble de phases pi, qui sera active.

La possibilité d'associer des prédicats temporels aux transitions est exprimée dans MAS par l'instruction :

pi si !tc alors pj

Ceci signifie que la transition (pi, pj) ne sera effectuée que ssi la phase pi reste active et non bloquée pendant un temps critique égal à tc.

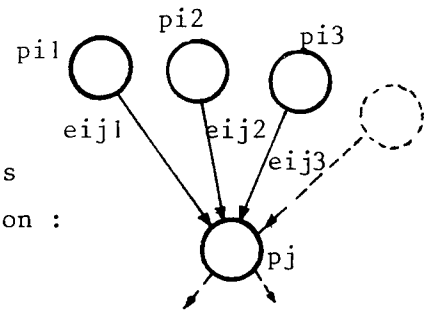


Figure 5.

II - 2. Création d'une configuration du système à partir des modules types prédéfinis

Ayant défini les types des modules qui constitueront le système décrit, il nous faut créer à partir de ces modules une configuration précise : choisir le nombre d'exemplaires de chaque module-type que le système utilisera et définir les interfaces entre les modules.

La construction du système se fait au moyen de 2 primitives CREE et LIE, dont l'utilisation doit être précédée par le mot-clé DCLCONF et suivie par FIN (Fig.6).

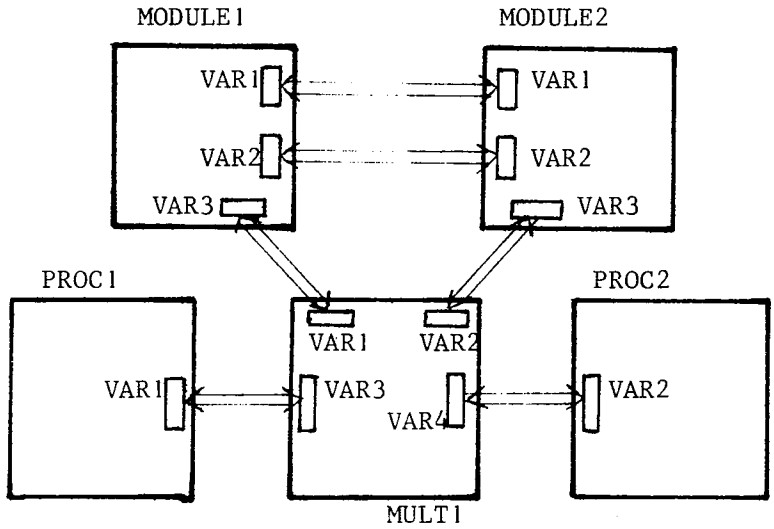
II - 2.1. Primitive CREE

La primitive CREE permet la création d'un ou plusieurs modules à partir d'un module type prédéfini

CREE modulei = nomodule

où modulei est le nouveau module créé à l'image du module type nomodule.

L'identificateur nomodule doit figurer dans le programme comme paramètre d'une primitive DCLMOD.




```
DCLCONF
  CREE    MODULE1, MODULE2 = MODULE
          PROC1, PROC2 = PROC
          MULT1 = MULT1

  LIE     MODULE1. VAR1 = MODULE2. VAR1
          MODULE1. VAR2 = MODULE2. VAR2
          MODULE1. VAR3 = MULT1. VAR1
          MODULE2. VAR3 = MULT1. VAR2
          PROC1. VAR1 = MULT1. VAR3
          PROC2. VAR1 = MULT1. VAR4
FIN
```

Figure 6.

On peut créer plusieurs modules sans répéter le mot CREE; soit des modules identiques au même module type, soit à des modules types différents :

CREE module1, module2... = nom1
module3... = nom2

Pour chaque nouveau module créé, il y aura duplication de l'espace mémoire associé et mise à jour des pointeurs appropriés (Cf.§ IV.2).

II - 2.2. Primitive LIE

La primitive LIE réalise l'interconnexion des modules créés auparavant.

Les paramètres de cette primitive sont deux variables d'interface des deux modules différents. Pour distinguer les variables qui ont le même nom mais appartiennent à des modules différents, on utilisera la notation "génétive": un nom composé constitué du nom du module et du nom de la variable, séparés par un point.

Ainsi, nous pouvons écrire :

LIE module1. varint1 = module2. varint2

L'effet de cette instruction est que les deux identificateurs *varint1* et *varint2* des modules *module1* et *module2*, désignent respectivement le même emplacement en mémoire. A partir de ce moment, tout changement dans le programme effectué sur la valeur de l'une des deux variables liées, se répercute sur l'autre.

II - 3. Données pour la simulation du fonctionnement du système

Jusqu'à présent, l'utilisateur a défini complètement son système. Il lui faut maintenant introduire des données pour simuler son fonctionnement.

- Définir une configuration initiale pour chaque module,
- Générer des entrées à des temps précis, dans le cas où il existe une communication avec l'environnement externe.

II - 3.1. Initialisation

La primitive INIT permet d'attribuer des valeurs initiales aux différentes variables du système :

INIT module1. var1 = val1
module1. var2, module2. var1 = val2
module2. var2 = val3

où *val1*, *val2*, *val3* sont des valeurs numériques. Toutes les variables sont initialisées, par défaut, à la valeur zéro.

II - 3.2. Génération des données temporisées

Le mot-clé DONNEES entame l'introduction des données pour la simulation. Elles sont fournies sous la forme de listes d'affectation, chaque liste étant caractérisée par un numéro qui définit le temps de son arrivée. Les variables affectées dans une telle liste doivent être déclarées :

- soit comme variables d'interface d'un module,
- soit comme variables d'entrée d'une partie contrôle.

L'introduction des données définit la simulation de l'environnement externe du système. Toutefois, par l'affectation des variables d'interface, autres qu'externes, l'utilisateur peut éventuellement intervenir sur le déroulement du programme par le changement de certaines conditions. Par contre, les variables qui sont locales à un module ne sont pas acceptées dans une liste de données, pour empêcher une intervention non souhaitée de l'utilisateur.

Le format d'une liste est le suivant :

ti module1. var1 = val1 module1. var2, module2. var3 = val2;

Ceci signifie qu'à l'instant ti les variables *var1* du module1 et *var2*, *var3* des modules 1 et 2 prendront les valeurs *val1* et *val2* respectivement.

Pour la génération des données, selon une distribution mathématique, il existe une autre possibilité; on définit un module qui génère les éléments de cette distribution. Ce module sera lié par des interfaces standards aux autres modules.

On peut également définir des entrées périodiques d'une période tp. Il suffit d'écrire dans une liste :

tk +tp (modulei. vari = vali modulej. varj = valj)

Les variables *vari* et *varj*, des modules correspondant, prendront les valeurs *vali* et *valj* respectivement pour la première fois au temps tk puis périodiquement après chaque intervalle tp.

Le mot-clé FIN termine un programme MAS. La simulation s'arrête soit par la fin de l'exécution du programme, soit par la primitive :

STOP ti

ce qui provoque l'arrêt de la simulation à l'instant ti.

III - DEFINITION FORMELLE DU LANGAGE

Dans ce qui suit, nous présentons les règles syntaxiques du langage en utilisant le meta-langage BNF (Backus Normal Form). Ce formalisme utilise un nombre de symboles d'une signification spéciale :

<X>	signifie	"l'objet nommé X"
::=	"	"est défini par"
|	"	"ou" (ou exclusive)
ε	"	"la chaîne vide"

La juxtaposition des noms ou des objets signifie qu'ils se suivent dans la construction des phrases du langage.

Il faut préciser, encore, que la grammaire est présentée d'une façon descendante, où les entités syntaxiques, de plus bas niveau, apparaissent les dernières.

LA GRAMMAIRE DU LANGAGE

A - STRUCTURE DU PROGRAMME

<programme> ::= <partie définition> <partie configuration>
<partie données>

<partie définition> ::= <liste des modules-types>
<partie configuration> ::= DCLCONF <liste instructions configuration> FIN
<partie données> ::= INIT <séquence d'affectation>
DONNEES <données pour simulation> FIN

<liste des modules-types> ::= <description module-type>|
<description module-type> <liste des modules-types>

B - DEFINITION D'UN MODULE-TYPE

description module-type ::= DCLMOD <entête module> <partie opérative>
<partie contrôle> FINMOD

<entête module> ::= <identificateur> (<liste interface>)
<liste interface> ::= <variable interface> | <variable interface>, <liste interface>
<variable interface> ::= <identificateur> | <identificateur> (<borne interface>)
<borne interface> ::= <constante>|<constante>, <borne interface>)

1) DESCRIPTION P.O

<partie opérative> ::= <procédure>|<procédure><partie opérative>
<procédure> ::= DCLPROC <entête procédure>|<liste instructions P.O> FIN
<entête procédure> ::= <identificateur>|<identificateur>(<liste variables>)
<liste instructions P.O> ::= <instruction P.O>|<instruction P.O>
<liste instructions P.O>
<instruction P.O> ::= TEMP <temps d'exécution>|<instruction APL>
<temps d'exécution> ::= <constante>|<expression APL>

2) DESCRIPTION P.C

<partie contrôle> ::= DCLCTRL <mode évolution> <déclaration entrées>
<déclaration sorties> DEBUT <liste instructions contrôle>
<mode évolution> ::= ASYN | SYN
<déclaration entrées> ::= ε|ENTREES <liste varaible d'E/S>
<déclaration sorties> ::= ε|SORTIES <liste variable d'E/S>
<liste variables d'E/S> ::= <identificateur> <type d'E/S>|
<identificateur> <type d'E/S>, <liste variables d'E/S>
<type d'E/S> ::= ε|*
<liste instructions contrôle> ::= <instruction contrôle>|
<instruction contrôle> <liste instruction contrôle>
<instruction contrôle> ::= <phase initiale> SI <fonction de transition>
ALORS <phase finale>
<phase initiale> ::= 1|<variable>
<fonction de transition> ::= <condition>|<condition> <délimiteur>
<fonction de transition>
<condition> ::= <expression logique APL> |!<expression APL>
<délimiteur> ::= ,|;
<phase finale> ::= <complément><variable> |
<complément><variable><délimiteur><phase finale>
<complément> ::= ε|∿

C - DEFINITION D'UNE CONFIGURATION

<liste d'instructions configuration> ::= <instruction configuration> |
<instruction configuration> <liste d'instruction configuration>
<instruction configuration> ::= CREE <création modules>
LIE <interconnexion modules>
<création modules> ::= <liste variable> = <identificateur>|
<liste variable> = <identificateur> <création modules>
<interconnexion modules> ::= <variable> = <variable> |
<variable> = <variable><interconnexion modules>

D - ENTREES DE DONNEES

<séquence d'affectation> ::= <liste variable> = <constante>|
<liste variable> = <constante><séquence d'affectation>
<données pour simulation> ::= <liste d'entrées>|
<liste d'entrées>; <données pour simulation>|
<liste d'entrées> ::= <constante><entrées affectées>|
STOP <constante>
<entréés affectées> ::= <séquence d'affectation>|
+ <constante> (<séquence d'affectation>)

E - VARIABLES

<liste de variables> ::= <variable>|
<variable>, <liste de variables>
<variable> ::= <variable indicée>|<identificateur> . <variable indicée>
<variable indicée> ::= <identificateur>|
<identificateur> (<liste d'indices>)
<liste d'indices> ::= <indice>|<indice>, <liste d'indices>
<indice> ::= <constante> | <variable indicée>
<identificateur> ::= <lettre> <suite de caractères αnumériques>
<suite de caractères αnumériques> ::= ε|<lettre><suite de caractères αnumériques>|
<chiffre><suite de caractères αnumériques>
<lettre> ::= A|B|C|.... |Z
<chiffre> ::= 0| 1| 2|... |9
<constante> ::= <chiffre>|<chiffre> <constante>

Les entités syntaxiques qui font référence au langage APL ne sont pas définies dans la grammaire du langage MAS :

<instruction APL> peut prendre la forme syntaxique d'une instruction APL avec la même sémantique.

<expression APL> et <expression logique APL> désignent toute expression APL dont les valeurs seront un entier positif et une valeur booléenne respectivement.

L'ordre d'évaluation des instructions et expressions APL est, ici aussi, de droite à gauche.

Les mots qui sont utilisés comme symboles de base à la définition de la syntaxe, sont des mots-clés réservés à l'interpréteur; ils ne peuvent pas être utilisés comme identificateurs.

On utilise en tout 19 mots clés :

DCLMOD	*DCLCTRL*	*SI*	*INIT*
FINMOD	*DEBUT*	*ALORS*	*DONNEES*
DCLPROC	*ASYN*	*DCLCONF*	*STOP*
FIN	*SYN*	*CREE*	
TEMP	*ENTREES*	*LIE*	
	SORTIES		

Les caractères spéciaux utilisés par le langage sont, d'une part ceux qui apparaissent comme symboles de base dans la grammaire (,; ( ) ⋆ ∿ !=) et d'autre part, ceux qui sont permis par APL.

Par exemple : ← → ≥≤∧∨≠ ⍳ ↑↓ etc...

Le blanc est utilisé pour séparer deux entités syntaxiques qui se suivent et aucune des deux n'est un caractère spécial.

IV - IMPLEMENTATION DU LANGAGE MAS

Une première version du langage est actuellement opérationnelle en conversationnel sous CP/CMS (IBM 360/67).

L'interpréteur de cette version du langage, qui aura un usage expérimental est écrit en APL pour deux raisons principales :

a) Assurer facilement l'interface entre l'interpréteur du langage MAS et celui d'APL, utilisé pour les parties du programme qui sont écrites en APL.

b) Prouver, par l'interpréteur même, la richesse du langage APL et son application à des algorithmes divers.

L'interpréteur réalise les fonctions suivantes (Fig.7) :

- Les analyses lexicographique et syntaxique du programme avec signalisation des erreurs syntaxiques.
- La génération du code objet.
- Simulation du fonctionnement du système décrit par le programme.

IV - 1. Analyse syntaxique

L'interpréteur travaille en un seul passage. Les phases d'analyse lexicographique et syntaxique sont classiques. La grammaire LL(1) est traitée par le transformateur de MM. Peltier et Griffiths [17], mais le code ainsi généré n'est pas utilisé directement. Il est retraité pour générer une chaîne d'entiers qui est interprétée en APL lors de l'analyse syntaxique. Celle-ci provoque l'appel des fonctions sémantiques qui constituent le corps même de l'interpréteur.

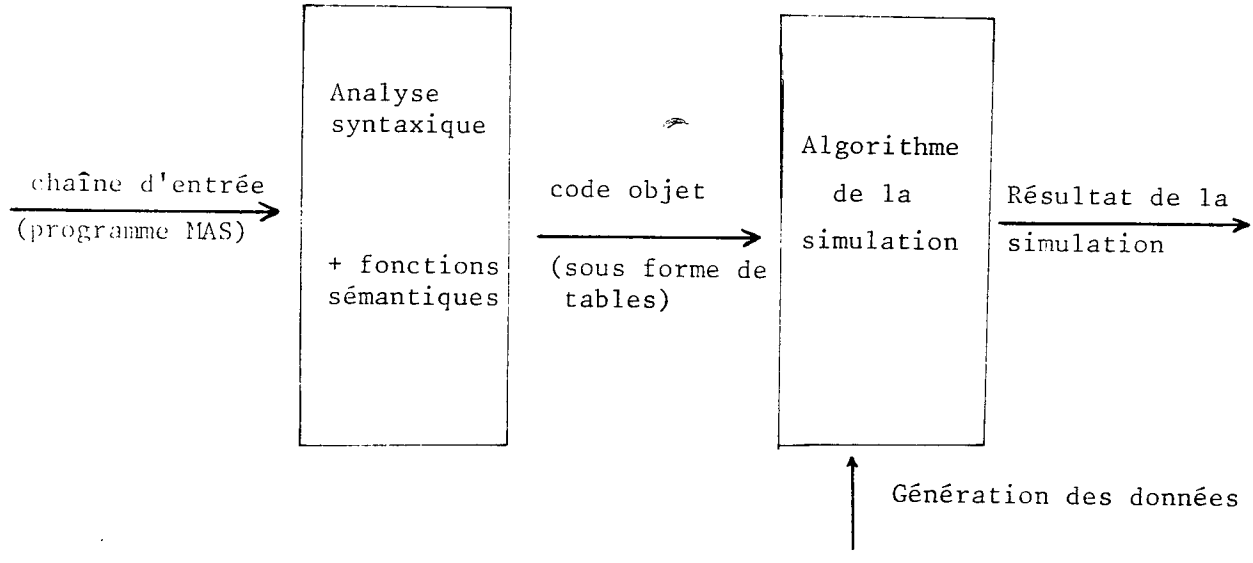


Figure 7.

IV - 2. Génération du code objet

L'interpréteur du langage par les fonctions sémantiques génère le code objet sous forme de table (Fig.8.)

Une variable est représentée par son déplacement dans l'espace mémoire du module auquel elle appartient. L'adresse réelle d'une variable est calculée en ajoutant à son déplacement une base qui indique pour chaque module le début de son propre espace mémoire.

Cet espace mémoire est divisé en 3 sous-espaces :

. Un pour les variables arithmétiques de la P.O.

. Un pour les variables booléennes de la P.C.

. Un pour les variables d'interface.
Pour les variables d'interface l'adressage est indirect .
A l'adresse calculée par "Base + Déplacement", on trouve un pointeur vers l'adresse réelle de la variable. Nous utilisons l'adressage indirect pour que 2 variables d'interfaces liées aient le même emplacement dans la mémoire.

L'interpréteur réserve cet espace mémoire dans trois vecteurs (TVALEUR, TBIN, TR) dont la taille augmente pour chaque nouveau module créé.

a) Représentation interne d'un module-type

Toutes les informations nécessaires pour représenter un module sont données par une entrée dans la table TABUN (Fig.8). Celle-ci contient des indications sur la taille mémoire nécessaire au module et des pointeurs, vers :

- les variables du module (TIDENT, TABVAR)
- les procédures de la P.O (TPROC)
- les instructions de contrôle (TCTRL)
- les chaînages des variables PC avec les instructions où elles apparaissent (TDPI).

Les variables

TIDENT, TABVAR sont les deux tables où sont codés les identificateurs du système :

- nom
- code (variables PO ou PC, procédure etc)
- déplacement
- dimension
- n° procédure associée (pour les variables PC)

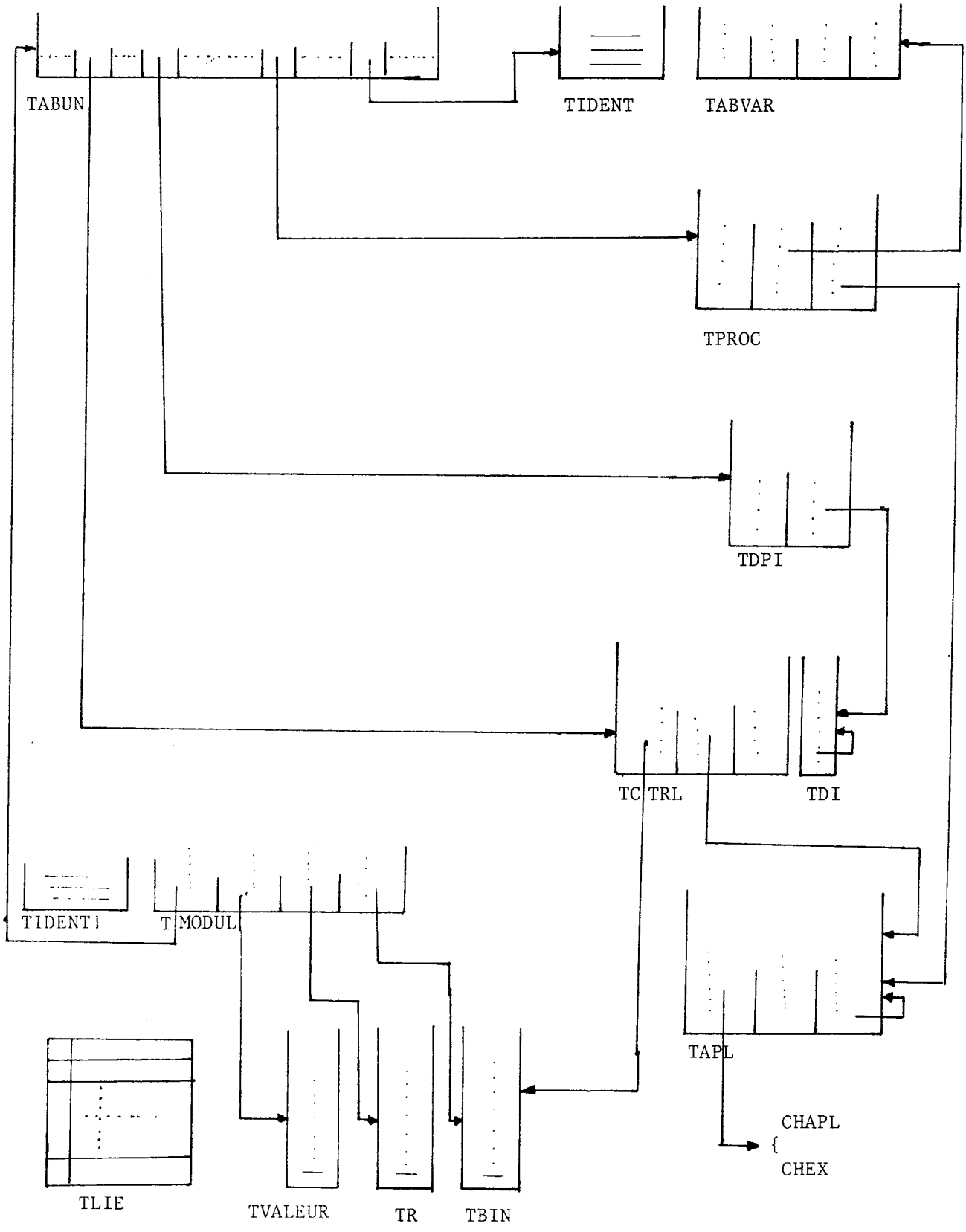

TABUN
TIDENT
TABVAR
TPROC
TDPI
TC TRL
TDI
TIDENT
T MODUL
TAPL
CHAPL
CHEX
TLIE
TVALEUR
TR
TBIN

Figure 8.

Les procédures

TPROC contient des informations concernant les procédures des P.O
- temps d'exécution
- paramètres (pointeur vers TIDENT, TABVAR)
- séquence d'instructions (pointeur vers TAPL)

Instruction de contrôle

Les instructions de la P.C sont codées dans TCTRL. Chaque instruction est représentée par un triplet : phase d'entrée, condition, phase de sortie. Une entrée dans TCTRL contient :
- les déplacements des variables booléennes associées aux phases pi, pj
- un pointeur vers la condition eij (TAPL).

Pour chaque phase la table TDPI contient un pointeur vers les instructions, où elle apparaît comme phase d'entrée, chaînées entre elles (TDI).

b) *Représentation d'une configuration*

Une entrée dans la table TMODUL est réservée pour chaque nouveau module créé. Elle y contient :
- le numéro du module-type associé (pointeur vers TABUN),
- les registres de bases de la mémoire allouée au module (indices de TBIN, TVALEUR, TR).

c) *Codage des expressions et instructions APL*

Pour les parties APL l'analyse et l'interprétation se font par l'interpréteur même d'APL. Ceci est effectué en utilisant l'opérateur d'APL "exécute".

L'opérateur d'exécution, appliqué sur un vecteur litteral X (une chaîne de caractères), a pour effet, dans la machine, de faire exécuter l'expression dont X est le texte et de substituer le "résultat final" de cette exécution à cette expression [18]. Selon les règles habituelles d'APL, le résultat final est la dernière "valeur" calculée en interprétant X de droite à gauche; mais comme X peut contenir des affectations intermédiaires, on voit que cette exécution peut avoir des effets très complexes et qu'elle ne peut désigner, à proprement parler, aucun résultat particulier.

Ainsi les instructions et expressions APL sont empilées dans un vecteur littéral (CHAPL, CHEX). La table TAPL (Fig.8) contient un pointeur vers le début de chaque instruction ou expression dans le vecteur littéral ainsi que sa longueur.

Néanmoins, les variables du langage MAS ne sont pas des variables APL. L'interpréteur de MAS remplace les identificateurs de ses variables par leurs adresses dans les vecteurs TBIN, TVALEUR et TR, connus par APL.

d) *Résultats de la simulation*

Pour suivre le déroulement du programme de simulation, l'utilisateur peut appeler la fonction RESULTAT qui est chargée d'imprimer les valeurs au terminal, soit de toutes les variables d'un module, soit de ses variables d'interface.

IV - 3. L'algorithme de la simulation

Simulation du temps

La simulation du temps est dirigée par les évènements. Le temps simulé est avancé par des intervalles variables chaque fois qu'un évènement arrive.

Deux types d'évènements sont possibles :

- la fin d'une procédure de la P.O,
- l'arrivée d'une entrée externe.

L'état du système ne change pas entre deux évènements.

L'échéancier

Ainsi l'échéancier dans lequel sont rangés les évènements futurs, est formé par deux listes, ℓext, $\ell proc$, une pour chaque type d'évènement.

L'utilisateur fournit la liste des entrées externes ℓext dans l'ordre de leur arrivée, et dans cet ordre elles sont prises en compte par le simulateur (FIFO).

L'autre liste $\ell proc$ est rangée dans une table (VTEMP). Cette table a autant d'entrées que de phases dans les réseaux. Chaque entrée contient (Fig.9) :

- Le temps ti où la procédure associée à cette phase aura terminé. ti prend la valeur zéro quand la procédure est inactive, ou quand la phase n'est pas associée à une procédure.
- Un pointeur, pour chaîner les éléments de la table, avec ti $\neq$ 0, dans l'ordre croissant de ti.

La variable TETE pointe le 1er élément dans la liste.

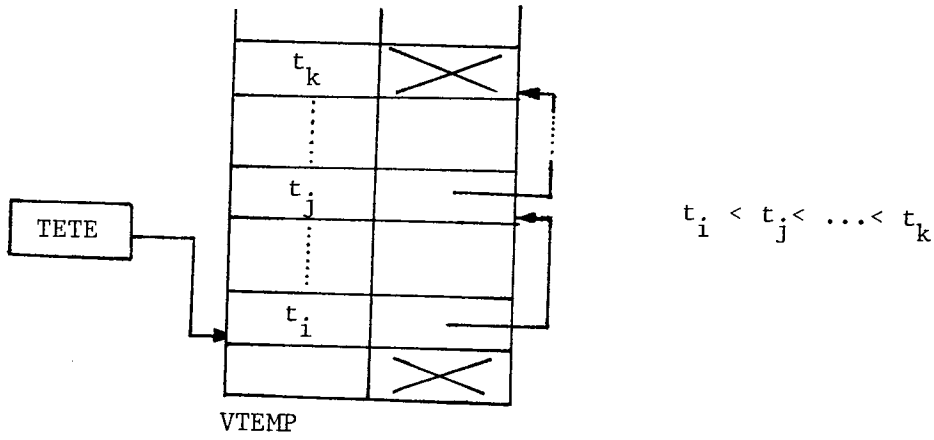


Figure 9.

Pour insérer un évènement, avec un temps d'apparition ti, on parcourt la liste, en commençant par TETE, jusqu'à l'élément qui aura tj > ti. On extrait toujours le premier élément pointé par TETE.

L'algorithme

L'organigramme présenté sur les Fig. 10 et 11 illustre l'algorithme de la simulation du fonctionnement d'un système décrit par MAS.

A l'arrivée d'un évènement, le temps de la simulation est avancé à l'heure de cet évènement. A ce moment, on exécute les opérations liées à l'évènement (exécution d'une procédure ou affectation des entrées externes) et on effectue les changements possibles à l'état du système, causés par l'évènement.

Les changements dans une P.O peuvent provoquer des évolutions à la P.C associée, mais aussi aux P.C d'autres modules, au moyen de variables d'interfaces.

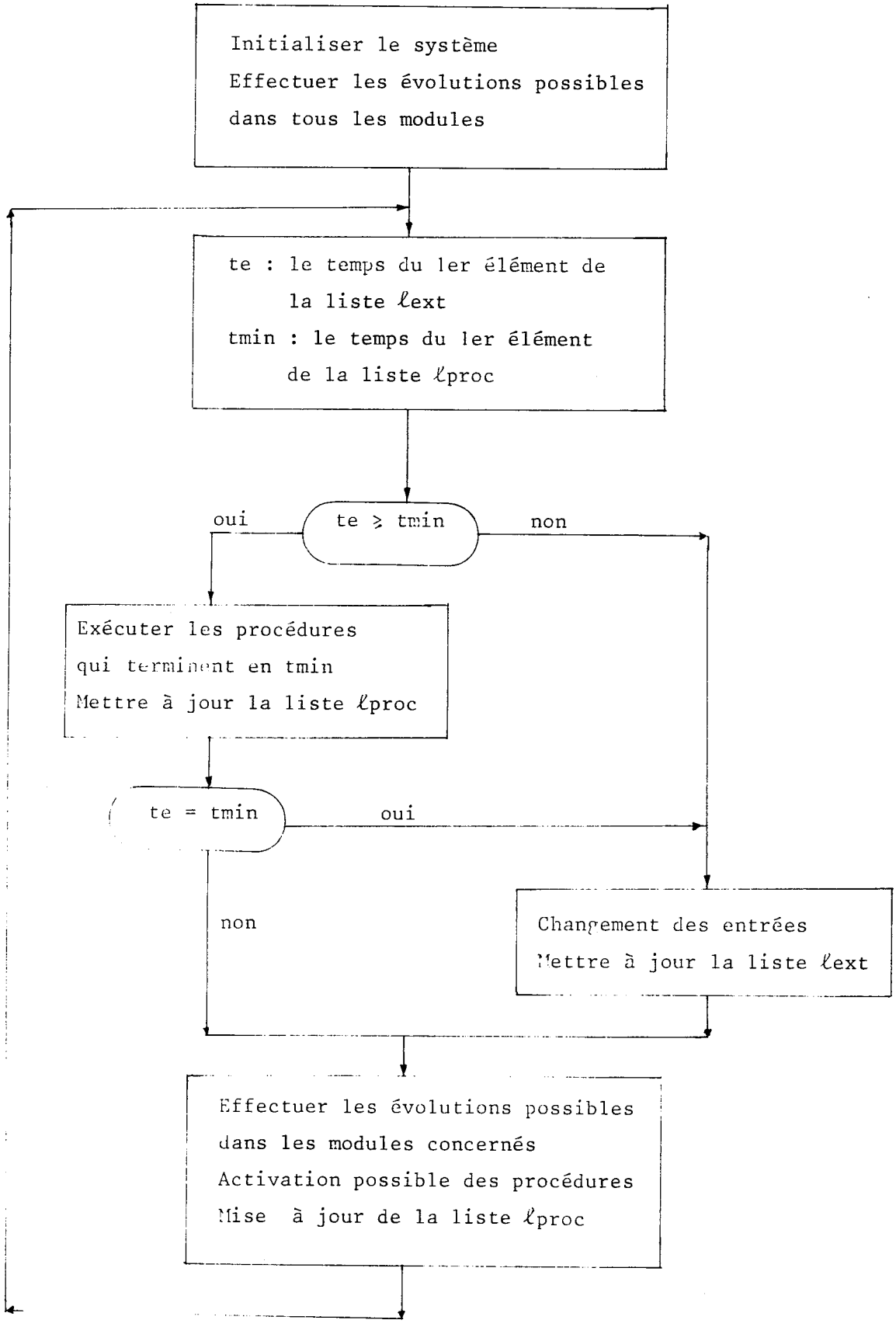

Initialiser le système
Effectuer les évolutions possibles
dans tous les modules
te : le temps du 1er élément de
la liste ℓext
tmin : le temps du 1er élément
de la liste ℓproc
oui
te ≥ tmin
non
Exécuter les procédures
qui terminent en tmin
Mettre à jour la liste ℓproc
te = tmin
oui
non
Changement des entrées
Mettre à jour la liste ℓext
Effectuer les évolutions possibles
dans les modules concernés
Activation possible des procédures
Mise à jour de la liste ℓproc

Figure 10.

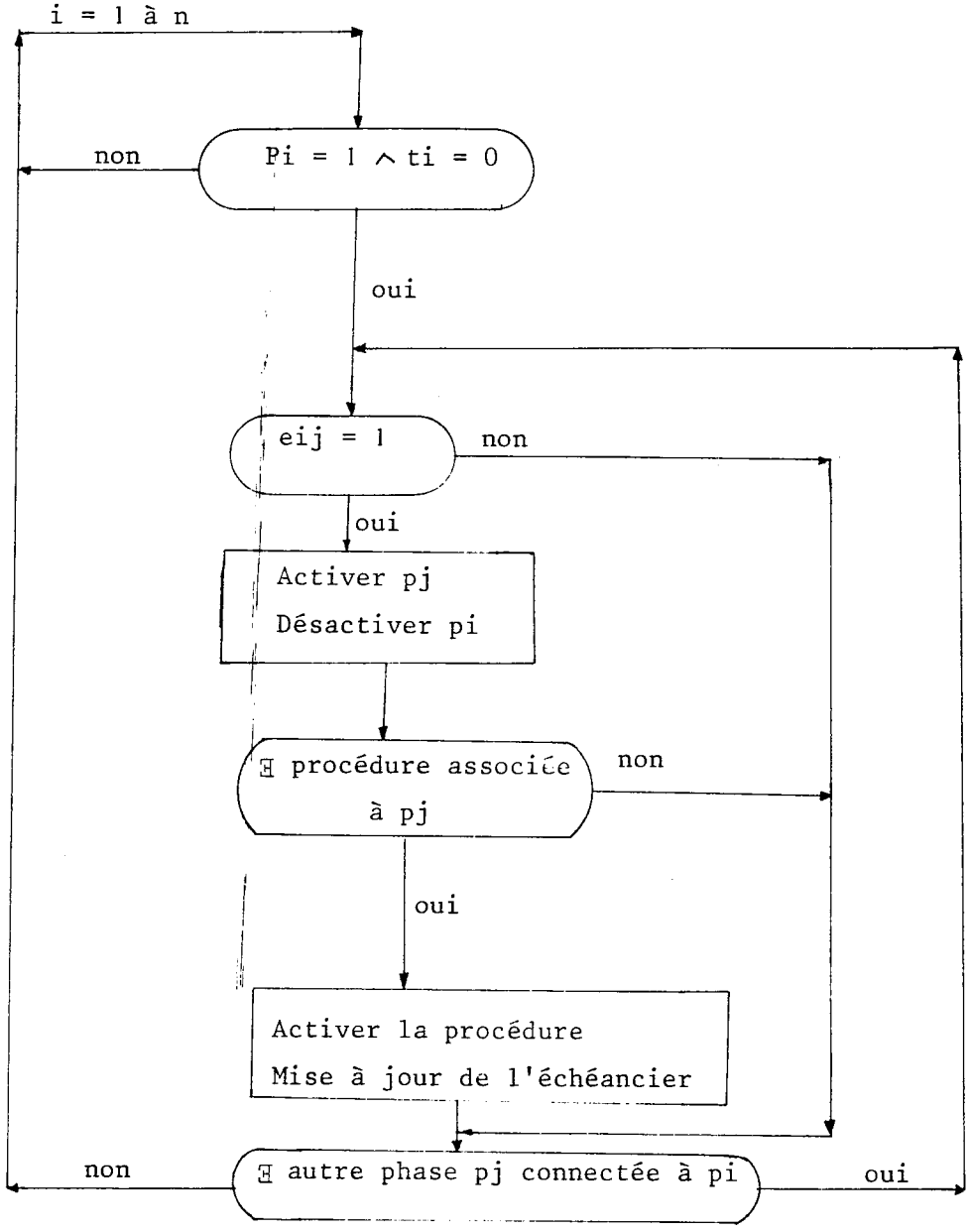

i = 1 à n
non
Pi = 1 ∧ ti = 0
oui
eij = 1
non
oui
Activer pj
Désactiver pi
∃ procédure associée à pj
non
oui
Activer la procédure
Mise à jour de l'échéancier
non
∃ autre phase pj connectée à pi
oui

Figure 11.

Une matrice binaire de lien (TLIE) (Fig.8) désigne les modules qui ont des variables d'interfaces communes. Quand l'exécution d'une procédure se termine dans un module m, on examine les P.C du module m et des modules liés avec m par un interface.

Quand il arrive une entrée externe on examine les P.C des modules affectés.

L'état des phases des réseaux de P.C, évolue conformément aux règles d'évolution des RCPP interprétés qui sont données au Chap.I, § III-4 (Fig.11). Selon ces règles, toutes les évolutions possibles, à un instant t dans un RCPP, se font en parallèle (simultanément). En réalité, les évolutions sont simulées en quasi-parallélisme l'une après l'autre.

On utilise une copie temporaire du vecteur TBIN (état de phases) dans laquelle on effectue tous les changements durant l'examen de toutes les phases. Dès que cet examen sera fini, on la recopie dans TBIN. On utilise aussi une copie temporaire des temps de fin des procédures qui sont activées, pour pouvoir tester certains cas de conflit.

Pour chaque P.C toutes les phases du réseau sont examinées. Seules les phases actives et non bloquées ($pi = 1 \wedge ti = 0$) sont retenues, pour examiner les conditions eij associées aux transitions (pi, pj). Si la valeur (actuelle) de eij est égale à 1 alors la phase pi est désactivée, pj et la procédure associée sont activées, et le temps de son échéance est inséré dans l'échéancier.

Dans le cas d'une simulation asynchrone, l'algorithme de la Fig.11 est répété jusquà la stabilisation du réseau, c'est-à-dire une configuration où des évolutions ne sont plus possibles sans l'arrivée d'un nouvel évènement.

Dans le cas d'une simulation synchrone les évolutions dans les P.C sont synchronisées. L'algorithme est répété jusqu'à la stabilisation du réseau, à chaque top d'horloge en tenant compte des nouveaux évènements possibles.

On prélève l'évènement suivant dans l'échancier en comparant les temps des premiers éléments dans les deux listes *lext* et *lproc*.

Remarques

- Les phases initialisées à 1 par la primitive INIT sont supposées être actives et non bloquées. Ainsi, les procédures associées ne seront pas exécutées initialement. Pour éviter des confusions possibles, il est conseillé de ne pas associer des procédures aux phases initiales.

- Il ne faut pas, également, associer des procédures aux phases utilisées dans un JOINT.

Dès que la phase P1 (Fig.12) est activée, la variable associée prend la valeur 1. Ceci peut désactiver la phase p2 indépendamment du bloquage de p1 durant un temps t. L'effet de la synchronisation sera alors perdu si p1 et/ou p2 restent bloquées pendant un temps non nul.

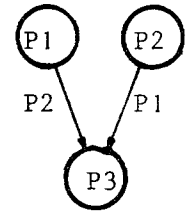


Figure 12.

- Une entrée impulsionnelle sera remise à "zéro" tout de suite après son apparition, aucune notion de durée n'étant rattachée à sa description. Par contre, une entrée à niveau reste à la valeur "1" jusqu'à la prochaine séquence de données, où lui sera affectée la valeur "0".

C H A P I T R E III

PERFORMANCES ET AMELIORATION DU LANGAGE

Dans ce chapitre nous étudierons l'efficacité et les moyens d'amélioration du langage MAS :

- Analyse de performances en fonction des paramètres de l'application à simuler.
- Problème de l'environnement temps réel.
- Investigation du domaine d'utilisation du langage.

I - EVALUATION DES PERFORMANCES DU LANGAGE

On évalue les performances du langage à l'aide des paramètres de l'application intervenant dans l'algorithme de la simulation, utilisé par l'interpréteur. On parlera ensuite de l'incidence du choix du langage d'implémentation (APL).

I - 1. Evaluation de l'algorithme de simulation

Deux parties de l'algorithme peuvent influencer considérablement ses performances :

- La simulation du temps et donc la gestion de l'échéancier.
- Le traitement des évolutions dans les P.C des modules du système à un instant donné.

a) *La gestion de l'échéancier* : L'actuelle organisation de l'échéancier permet d'extraire facilement un évènement, le premier dans la liste. Par contre, pour l'insertion d'un évènement on doit parcourir la liste jusqu'à l'évènement qui aura un temps d'apparition plus grand que celui de l'évènement à insérer.

Le coût de cette partie de l'algorithme dépend de la longueur de cette liste, c'est-à-dire du nombre d'évènements dans l'échéancier,lequel est égal au nombre de procédures activées à un instant donné, et donc du degré de parallélisme dans le système.

Ce nombre est un facteur dynamique qui varie beaucoup selon les applications. Il dépend aussi du nombre de modules du système, et de l'échelle de temps sur laquelle seront simulés ces modules.

Àinsi, nous pouvons envisager des applications où cette liste (l'échéancier) sera assez longue et par conséquent, l'insertion d'un nouvel évènement durera longtemps.

Aussi, proposons-nous d'accélérer cet algorithme en utilisant d'autres approches pour l'organisation de l'échéancier, notamment en utilisant des arbres binaires.

La recherche dans un arbre binaire repose sur le même principe que la recherche binaire dans les tables. Dans le cas idéal (arbre binaire et bien balancé), chaque test élimine la moitié des éléments qui restent et le temps de recherche, pour N éléments, est proportionnel à $\log_2 N$. Si l'arbre n'est pas balancé, on aura en moyenne 1,4 $\log_2 N$ accès dans l'arbre pour la recherche d'un élément [19].

Dans la recherche linéaire, chaque test élimine un seul élément et par conséquent le temps est proportionnel à N. Quand N augmente la recherche binaire devient plus rapide par rapport à la recherche linéaire.

L'utilisation des arbres binaires devient intéressante si le nombre N d'évènements dans la liste dépasse un certain seuil No (autour de 30 à 50), variant avec l'algorithme de gestion de l'arbre [20].

b) *Traitement des évolutions des P.C* : Cet algorithme peut être divisé en trois parties :

- La recherche des modules où des évolutions sont possibles après l'arrivée d'un évènement.
- La recherche des phases actives et non bloquées Pi dans un module particulier.
- L'évaluation des expressions eij pour chaque phase Pi active et non bloquée.

i) Recherche des modules où des évolutions sont possibles

Le nombre de modules où des évolutions sont possibles dépend du type d'évènement qui arrive (Cf. Ch.2).

- Dans le cas d'un changement d'entrées externes du système, nous examinerons les modules auxquels appartiennent ces variables d'entrée.

- Dans le cas de la fin du temps d'exécution d'une procédure, on examine d'une part le module qui contient la procédure et d'autre part les modules ayant un interface commun avec le premier.

Si n_{mod} désigne le nombre moyen de modules à examiner, chaque fois qu'un évènement arrive, le coût d'un pas dans l'algorithme de la simulation sera donné par l'expression :

$$n_{mod} \times A \qquad (1)$$

où A représente le coût de la recherche pour un module (Cf. §ii).

Le n_{mod} dépend du nombre total N de modules représentant le système et de la densité d_{mod} de liaisons entre eux. La densité de liaison est définie comme la probabilité pour que deux modules soient liés (alors $n_{mod} = N \times d_{mod}$).

ii) Recherche des phases actives et non bloquées dans un module (facteur A de (1))

Les phases actives et non bloquées sont déterminées par un test effectué sur toutes les phases du réseau de contrôle. Si n_{phases} désigne le nombre moyen de phases dans les réseaux de contrôle du système, on a :

$$A = n_{phases} \times b + B \qquad (2)$$

où b est le coût du test effectué pour chaque phase
et B est le coût de l'évaluation des expressions eij pour chaque phase active et non bloquée (Cf. iii).

iii) Evaluation des expressions eij (terme B de (2))

Pour chaque phase pi active et non bloquée on évalue K expressions eij, j = 1, 2,.. K, où K est le nombre moyen de phases directement successeur de pi dans le réseau. De plus, si n_{active} désigne le nombre moyen de phases actives et non bloquées dans un réseau à un instant donné, on a alors :

$$B = n_{active} \times K \times C \qquad (3)$$

où C est le coût moyen d'évaluation d'une expression eij.

Le facteur n_{active} dépend de la densité d'activation dans un module : probabilité pour qu'une phase soit active et non bloquée.

iiii) Coût moyen d'une évolution (un pas de l'algorithme de simulation)

En remplaçant A et B par (2) et (3) respectivement dans (1) on obtient la formule :

$$n_{mod} [n_{phases} \times b + n_{active} \times K \times C] \qquad (4)$$

Pour estimer le coût moyen qui prendra un pas de l'algorithme de simulation, on doit ajouter à (4) le temps moyen de calcul dans la P.O (exécution d'une procédure, affectation des variables externes).

$$\mathcal{C} = n_{mod} [n_{phases} \times b + n_{active} \times K \times C] + C_{po} \qquad (5)$$

avec

n_{mod} : le nombre moyen de modules liés avec un module particulier

n_{phases} : le nombre moyen de phases dans les RCPP des P.C

n_{active} : le nombre moyen des phases actives et non bloquées dans les RCPP

K : le nombre moyen d'arcs sortant d'une phase

b : le coût du test pour déterminer si une phase est active et non bloquée

C : le coût moyen d'évaluation d'une expression

C_{po} : le coût moyen des calculs dans la P.O

Remarques

- Les différents coûts, b, C, C_{po} et $\mathcal{C}$, peuvent être évalués en cycles machine.
- n_{mod}, n_{phases}, K et n_{active} dépendent de l'application; b dépend de l'algorithme de simulation et de son implémentation; C et C_{po} dépendent de l'algorithme et de l'application.
- Tous les facteurs intervenant en (5) sont statiques et peuvent être calculés dès la définition du système, sauf n_{active} et C_{po} qui eux sont des facteurs dynamiques calculés par l'évolution du système.

I - 2. Langage de l'implémentation

L'interpréteur du langage MAS est écrit en APL. APL étant un langage interprété, tourne plus lentement qu'un langage compilé. Il permet, en outre, une écriture facile et compacte, mais au prix d'un coût d'interprétation assez élevé, dû à la réinterprétation de chaque fonction du programme. Ceci influence aussi l'estimation du coût de la formule (5) (Cf. §I.1); C et C_{po} dépendent directement d'APL par l'utilisation de son opérateur "exécute" (Cf. Ch.II §IV.2).

De plus, APL ne se prête pas facilement à l'utilisation de listes et de pointeurs. Les opérations sur listes sont remplacées par les opérations sur les tableaux qui sont facilités par APL.

Ainsi, nous estimons qu'une utilisation plus rentable de MAS devrait envisager son implémentation par un autre langage, assembleur ou langage compilé.

II - AMELIORATION DES PERFORMANCES DU LANGAGE

Une telle amélioration doit, à notre avis, se faire en agissant sur les facteurs cités au paragraphe précédent.

Dans la formule (5) le facteur n_{mod} a une grande importance et doit être réduit en premier. Par le même algorithme on minimise aussi le facteur K du deuxième terme. Des améliorations sont aussi proposées pour les facteurs C et n_{phases}.

II - 1. Réduction des facteurs n_{mod} et K

Le coût d'un pas, dans l'algorithme de la simulation, évalué par la formule (5) peut être diminué par un travail préalable effectué au moment de l'analyse du programme. Pour cela nous avons besoin de lier les évènements aux évolutions du réseau.

La sensibilité des RCPP aux évènements est définie par un graphe biparti dont les arcs relient les variables du système aux expressions eij, où apparaissent ces variables.

Ce graphe permettra de déterminer les expressions eij dont la valeur peut changer après l'arrivée d'un évènement et de n'évaluer, ainsi, que celles-ci.

Le changement d'une variable peut être effectué soit par l'exécution d'une procédure, soit par l'arrivée d'une entrée externe. Nous avons alors besoin, pour chaque procédure de la liste, des variables affectées au cours de son exécution et pour chaque expression la liste de ses variables. Ces deux listes étant construites au moment de l'analyse du programme, nous pouvons définir les graphes bipartis de la sensibilité des RCPP (Fig.1).

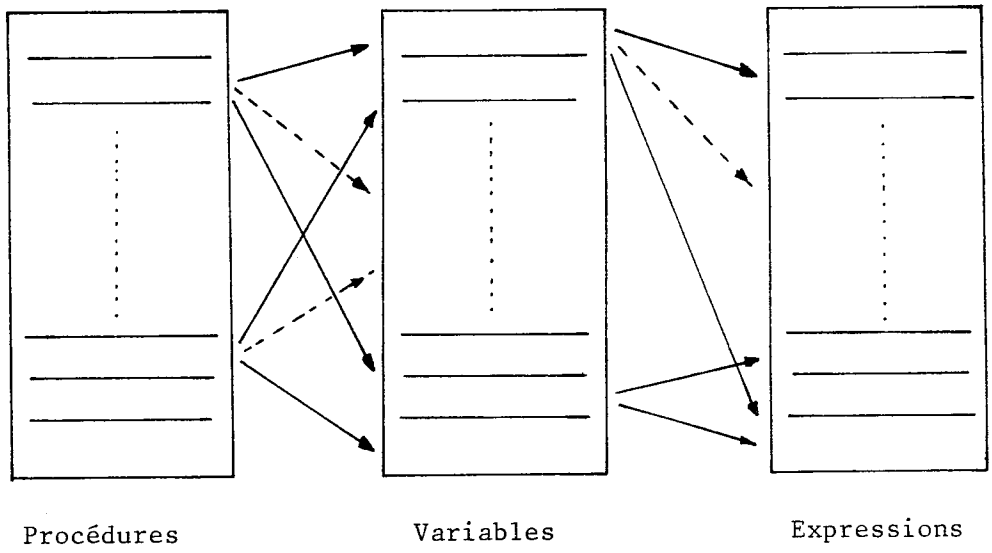


Figure 1.

En fait le schéma de la Fig.1 se réduira à celui de la Fig.2, en liant les procédures directement aux expressions; seules les variables externes resteront liées aux expressions.

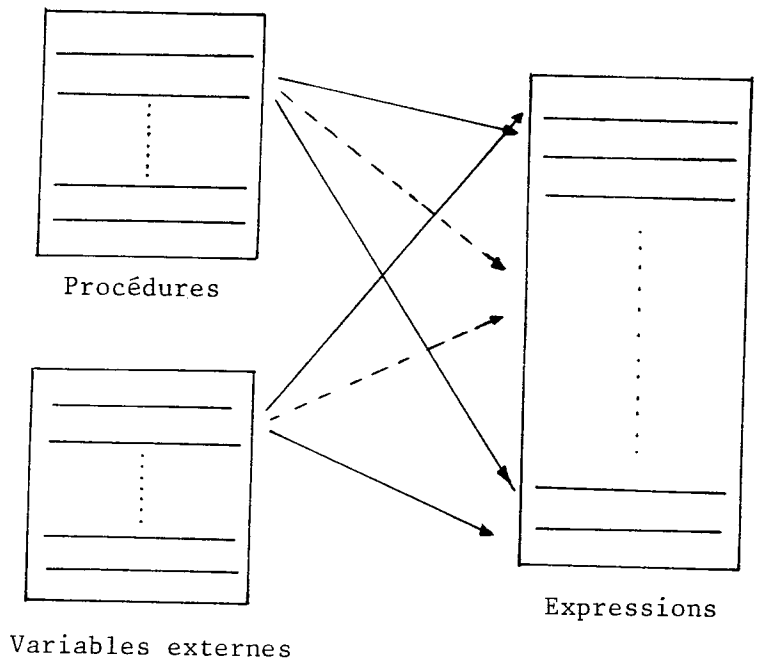


Figure 2.

Les grandes lignes du nouvel algorithme proposé seront les suivantes :

- Après l'arrivée de chaque évènement, on cherchera par le graphe biparti, les expressions sensibles à cet évènement. Un signal spécial : "valeur indéterminée" remplacera la valeur de ces expressions, les autres étant inchangées.

- Pour chaque phase pi active et non bloquée, on cherchera la valeur des expressions eij. eij sera évaluée si elle est marquée par le signe "valeur indéterminée", sinon sa valeur sera donnée directement par les résultats des calculs antérieurs.

Cet algorithme apportera une amélioration par diminution de deux facteurs : K et n_{mod} de la formule (5) :

. K par réduction du nombre d'expressions eij qui seront évaluées pour chaque phase active et non bloquée.

. n_{mod} par réduction du nombre de modulesoù l'algorithme sera appliqué : seuls les modules dont une variable d'interface a changé seront examinés.

Mais pour estimer l'efficacité de l'amélioration qu' apportera un tel algorithme il faut considérer aussi le type des applications.

- Le nombre de modules et la densité de leur liaison peuvent influencer sur la diminution du facteur n_{mod}; cette diminution sera d'autant plus considérable que le nombre de modules sera élevé et que leurs interfaces seront complexes.

- La complexité des procédures et des expressions eij peut diminuer le gain théorique sur le facteur K. Un nombre élevé d'affectations dans une procédure et un nombre élevé de variables dont dépend chaque expression conduiront au fait que peu d'expressions ne seront pas marquées du signe "valeur indéterminée".

II - 2. Réduction des facteurs n_{phase} et C

Le terme $n_{phase} \times b$ de (5) peut être supprimé en procédant au chaînage des phases actives et non bloquées.

Une autre possibilité d'amélioration des performances du langage est la diminution du coût d'évaluation d'une expression eij, c'est-à-dire le facteur C de la formule (5).

A la place de l'expression eij nous pouvons calculer sa dérivée booléenne par rapport à la variable qui vient de changer. Cette règle ne peut pas être appliquée dans tous les cas. Un calcul formel, automatique ou manuel, effectué au début du programme, peut nous renseigner sur la complexité des dérivées partielles d'une expression [21]. Si on trouve des dérivées beaucoup plus simples à calculer que l'expression elle-même, alors nous pouvons envisager l'application de cette règle pour quelques expressions et pour certaines variables.

Enfin, une analyse plus poussée peut conduire au remplacement du réseau par des formules mathématiques, exprimant les relations entre les phases et les expressions. Ces formules auront la forme des équations décrivant les RCPP (Cf. Ch. I, § III.4.1).

Toutes ces modifications et améliorations sont en cours d'étude et d'application sur la version actuelle.

Il est clair que, dans l'estimation du coût, il faut ajouter :

- Le coût des opérations sur les listes : temps de recherche et de mise à jour,
- Le coût de la construction de ces listes au moment de l'analyse du programme : temps de calcul de la sensibilité et de dérivées booléennes.

III - UTILISATION DU LANGAGE EN TEMPS REEL

Nous étudions ici l'utilisation du langage MAS en temps réel pour le contrôle des processus.

Nous constatons qu'une partie de l'interpréteur actuel ne servira plus et par contre que l'autre partie utilisée en temps réel doit être très performante. Nous pourrons représenter par un schéma les fonctions contenues dans la version actuelle de l'interpréteur (Fig.3).

La partie I représente tout ce qui est conservé intégralement en temps réel. Par contre, la partie II représente tout ce qui est simulé et elle sera donc remplacée par un système physique.

La simulation des P.C se situe entièrement dans la partie I de l'interpréteur. Les P.O peuvent contenir des procédures qui simulent un processus physique mais aussi des procédures qui effectuent des calculs nécessaires même en temps réel. Ainsi une partie de la P.O sera gardée en temps réel, l'autre étant remplacée par le processus physique à contrôler.

Nous avons donc intérêt à améliorer les performances de la partie I, critique pour l'utilisation de MAS en temps réel, et pour cela les propositions du paragraphe précédent seront utiles.

Mais d'autres dispositifs logiciels seront mis en fonction :

- Traitement des interruptions : l'activation d'une procédure correspond, en temps réel, à l'envoi d'un signal vers le processus physique à commander. La fin d'une procédure sera signalée par une interruption. Le système doit être capable de prendre en compte les interruptions, de les classer par ordre prioritaire et de les gérer.

- Passage en manuel ou arrêt : dans le cas d'une défaillance, le système doit pouvoir assurer un fonctionnement non dangereux, soit par utilisation en manuel, soit par l'arrêt complet.

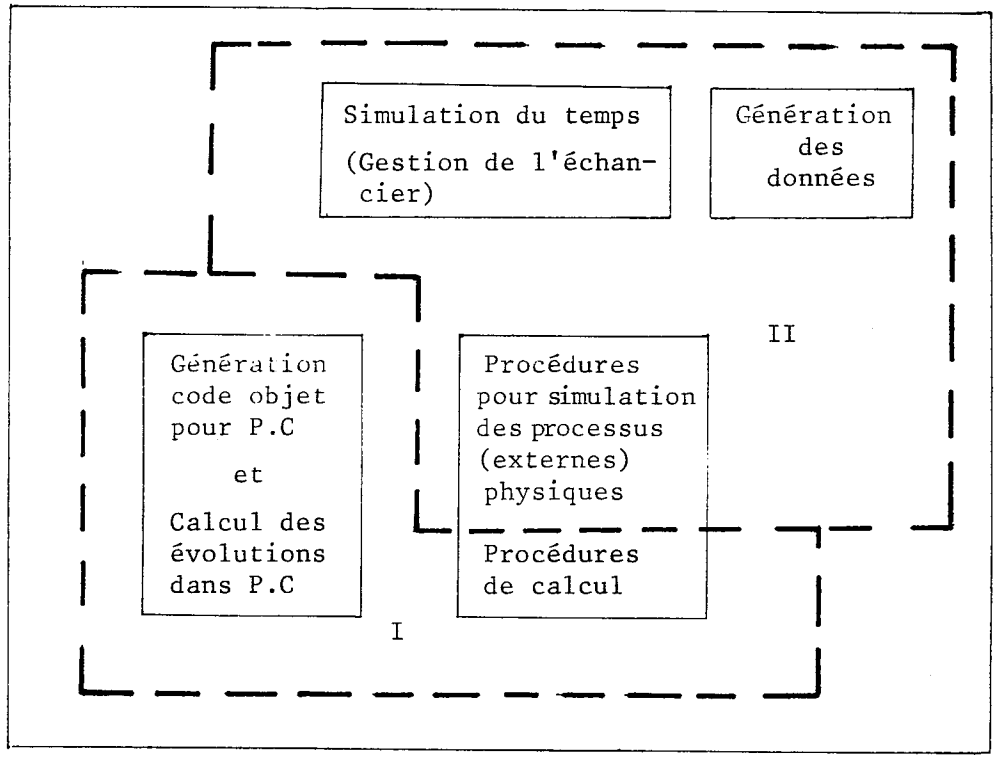


Figure 3.

Contraintes temps réel

Avant l'implémentation de MAS en temps réel on doit, pour une application donnée, évaluer les contraintes temporelles et estimer la "puissance installée nécessaire".

Par le processus physique à contrôler, on mesure la fréquence d'arrivée des signaux envoyés par lui : δ_{ext}.

L'analyse du réseau d'application détermine une borne supérieure du nombre d'évolutions, N_{max}, à exécuter entre 2 arrivées successives de ces signaux. On définit, ainsi la fréquence d'évolution δ_{max} demandée au système :

$$\delta_{max} = \delta_{ext} \times N_{max}$$

La puissance installée nécessaire est définie comme le produit

$$P = C^{*}_{x}\delta_{max}$$

où C^{*} est le coût moyen (en cycle machine) d'une évolution dans les P.C.

Exemple :

Si δ_{ext} = 3 signaux/sec et N_{max} = 20 évolutionspour traiter un tel signal on en déduit que : δ_{max} = 60 évolution/sec.

Si C^{*} = 10.000 cycles machine
il nous faut une puissance de

P = 10.000 x 60 = 600.000 cycles/sec.

Remarques

1. C^{*} est donné par la formule (5) du §I.1 réduite par le coût de l'exécution des procédures se situant dans la partie II de l'interpréteur. C^{*} n'étant qu'une moyenne, un coefficient de sécurité est nécessaire, à moins qu'on sache calculer C_{max} (remplacer les n de la formule (5) par des n_{max}).

Si par l'analyse il est difficile de mesurer ces paramètres avec précision, une évaluation plus fine de ces durées peut être faite sous simulation par l'utilisation d'une primitive TIME. En effet, dans certains systèmes et certains langages il est donné la possibilité de lire l'heure de l'horloge machine dans une variable. En utilisant une telle primitive (TIME(X)), au début et en fin d'opération (TIME(Y)), on peut par une simple soustraction (Y-X) connaître la durée de cette opération.

Tout ceci permet de s'assurer que la puissance de calcul est suffisante pour l'application donnée.

2. Si la puissance d'un processeur Po est inférieure à la puissance de l'application P, on pourra implémenter le système en multiprocesseur.

On sait qu'il faudra certainement, au moins, P/Po processeurs et probablement plus.

Une répartition judicieuse de l'application sur plusieurs processeurs reste cependant un problème délicat.

IV - EXTENSION DU LANGAGE

Dans son état actuel le langage est suffisant pour des applications assez simples (Cf. Ch. IV). Néanmoins, des applications plus complexes auront besoin de facilités et de possibilités plus grandes.

Nous pouvons en citer quelques unes qui sont déjà en cours d'étude :

- Ajouter des primitives de sorties qui permettront à l'utilisateur de préciser la trace qu'il veut avoir pour suivre le déroulement de la simulation du fonctionnement de son système.
- Améliorer la façon d'interconnecter les modules entre eux, en définissant plusieurs types d'interfaces selon la direction dans laquelle circule l'information (bidirectionnelle ou unidirectionnelle) [2].
- Ajouter des primitives qui permettront de modifier dynamiquement la configuration d'un système. Pouvoir par exemple définir de nouveaux modules au cours de la simulation, ou supprimer des modules qui ne seront plus utilisés.
- Dans le système actuel tous les temps sont considérés comme fixés (temps de réponse d'une procédure, temps d'arrivée des évènements). On peut ajouter des primitives d'entrée permettant l'introduction de temps aléatoire (utilisation de générateur de nombres, au hasard, pour simuler des distributions données, par exemple : loi de Poisson).
- On peut aussi donner la possibilité d'évaluer dynamiquement le temps de l'exécution d'une procédure de calcul, en fonction du temps effectif de sa simulation. Ceci apportera un changement à l'algorithme de simulation, la procédure étant simulée au moment de son activation et non plus à la fin. On peut obtenir ainsi une simulation plus fidèle à la réalité.
- Ajouter une primitive pour accéder à la variable qui représente l'horloge du système simulé, en principe, pour lire son contenu dans une variable du programme (Cf. Ch. IV, Ex 1).

V - AIDE A LA CONCEPTION

Pour compléter l'aide à la conception il faudra donner au concepteur le moyen d'effectuer un certain nombre de mesures :

- Calculer la fréquence relative d'appel des différentes procédures. Cette mesure montre le degré d'utilisation des différentes parties du système et peut être utile pour déterminer la fiabilité nécessaire aux modules matériels qui exécutent ces parties.

- Estimer le temps écoulé entre 2 appels d'une même procédure. Cette mesure peut être utile pour le test en cours de fonctionnement pour savoir le temps qui s'écoule entre deux passages par une phase obligatoire [22].

Ainsi nous devons ajouter des compteurs "espions" dans les procédures, dont le contenu pourrait être consulté à tout instant par l'utilisateur.

Le concepteur doit aussi détecter les conflits logiques dans son modèle ainsi que les blocages :

- nouvelle activation d'une procédure, avant que la précédente activation ne soit terminée,

- affectation d'une même variable par des procédures différentes qui sont actives en parallèle (détection des sections critiques).

Pour cela, la liste des variables de chaque procédure doit être classée en deux catégories : variablesen lecture et variables en écriture.

Au moment de l'appel d'une procédure on détecte les conflits, c.a.d une variable en lecture affectée par une autre procédure active, ou une variable en écriture lue ou affectée par ailleurs (en parallèle).

Le simulateur doit sortir des messages explicites précisant la nature et le lien du conflit.

Pour améliorer la sécurité du système l'utilisateur pourraît vérifier certaines conditions :

- vérifier des conditions invariantes, soit à chaque évolution (ce qui est très cher), soit en des points déterminés; l'utilisateur peut insérer des invariants qui assurent une protection minimale contre des incidents (Cf. assertions exécutables);

- déterminer la contamination d'erreurs; une erreur est symbolisée par une variable fixée à une valeur indéterminée W; le système sera simulé avec cette valeur et en définissant des règles de propagation de l'erreur on pourra déterminer les parties du système contaminées et choisir une procédure de reprise optimale [22].

C H A P I T R E IV

EXEMPLES D'APPLICATION

L'utilisation du langage MAS est illustrée par l'application à deux exemples :

- La modélisation de systèmes de télécommunication (en collaboration avec le CNET).
- La modélisation d'une structure multiprocesseur pipe-line réalisant une transformée de Fourier rapide (en collaboration avec la SINTRA).

D'autres domaines d'application sont présentés dans [11] et [13].

I - LES LIAISONS LU DU SYSTEME E10/128

Cet exemple porte sur la modélisation d'une partie du système de télécommunication E10/128.

Il s'agit de la description des modules d'interface des liaisons LU, reliant les Unités de Raccordement (UR) aux Marqueurs (MQ) [23]. Le marqueur est l'organe chargé d'assurer la liaison entre les Unités de raccordement, les multienregistreurs, le réseau de connexion et l'organe de contrôle. Les Unités de Raccordement sont reliées avec les lignes d'abonnés.

Chaque liaison LU est multipliée sur les 2 marqueurs MQ1 et MQ2 et dessert un groupe de 16 unités de raccordement (Fig.1). Pour raisons de sécurité, les liaisons LU de chaque groupe d'UR sont doublées (LU_{ia}, LU_{ib}) et l'émission se fait en parallèle sur les 2 liaisons a et b. Les échanges des marqueurs vers les UR sont réalisés par les liaisons impaires LU1 à LU15. Les échanges des UR vers les marqueurs sont réalisés par les liaisons paires LU2 à LU16.

L'exemple traité a été limité à l'étude des liaisons d'un groupe seulement de UR avec les 2 marqueurs. Deux fonctions de base sont examinées :

- Traitement des échanges à l'initiative d'un MQ.
- Traitement des échanges à l'initiative d'une UR.

La base de temps

Les intervalles de temps sont organisés en trames de 125 µs, chacune étant divisée en 32 voies temporelles t_0 à t_{31}. La première moitié de la trame (t_0 à t_{15}) est utilisée pour la signalisation entre marqueurs et UR (essai rapide de la liaison dans les 2 sens et test de disponibilité de l'organe récepteur). La seconde partie est utilisée pour la transmission d'informations; le créneau t_{16}-t_{23} est réservé à l'échange d'information avec le marqueur 1, le créneau t_{24}-t_{31} à l'échange avec le marqueur 2.

Chaque UR est repérée par la voie temporelle de même rang : t_i affectée à UR_i (Fig.2).

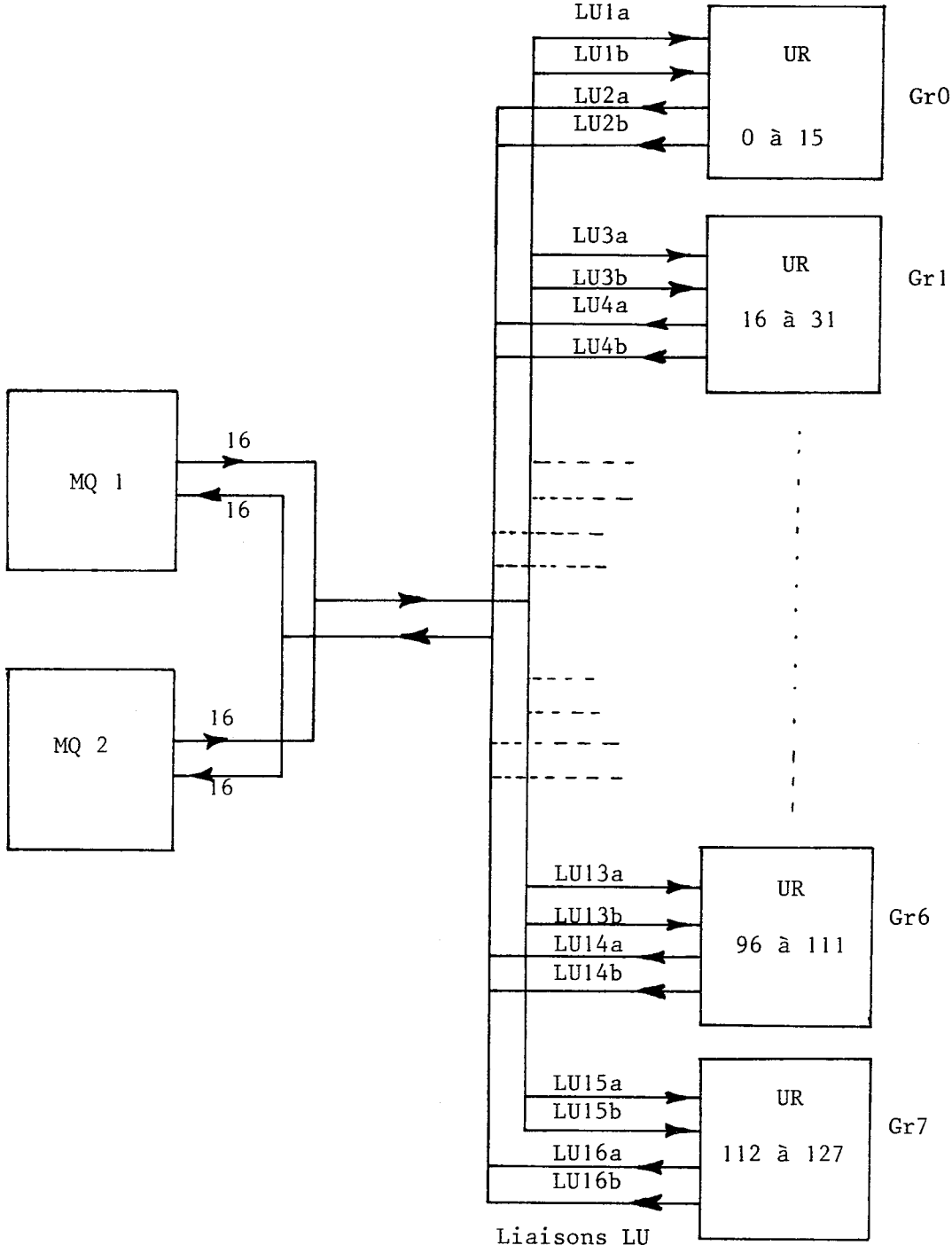
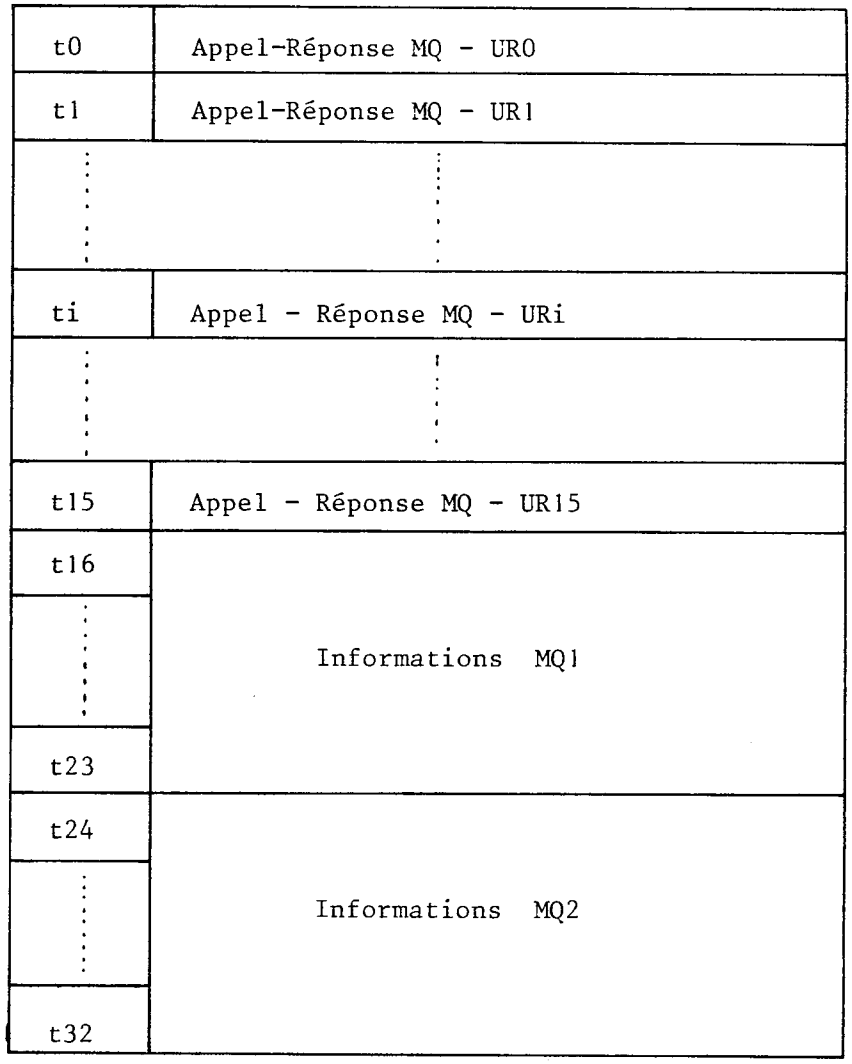

LU1a
LU1b
LU2a
LU2b
UR
0 à 15
Gr0
LU3a
LU3b
LU4a
LU4b
UR
16 à 31
Gr1
MQ 1
16
16
MQ 2
16
16
LU13a
LU13b
LU14a
LU14b
UR
96 à 111
Gr6
LU15a
LU15b
LU16a
LU16b
UR
112 à 127
Gr7
Liaisons LU

Figure 1.



Format de trames

Figure 2.

Les échanges sur les liaisons LU utilisent le rythme θ(1,28 MHZ). Ainsi chaque voie temporelle t_i comprend 5 instants caractéristiques θ_1 à θ_5.

Les liaisons LU permettent 2 échanges d'informations par trame : l'une avec le MQ1, l'autre avec le MQ2 (2 échanges dans le même sens ou 1 échange dans chaque sens). Les conflits de prise des MQ par les UR sont réglés en affectant aux UR un θ d'appel différent vers chaque MQ (Fig.3).

θ1	Essai liaison avec URn
θ2	Appel MQ1 → URn
θ3	Réponse MQ1 → URn
θ4	Appel MQ2 → URn
θ5	Réponse MQ2 → URn

LU impaire
(MQ → UR)

θ1	Réponse à essai (trame suivante)
θ2	Appel URn → MQ1
θ3	Réponse URn → MQ1
θ4	Appel URn → MQ2
θ5	Réponse URn → MQ2

LU paire
(UR → MQ)

tn : 0 à 15

Echanges signalisation MQ-UR

Figure 3.

Echanges à l'initiative d'un MQ

a) Procédure appel-réponse et essai de la liaison

Le marqueur émet un élément binaire (eb) (BAMU) d'appel dans le tn de l'UR, répété toutes les trames jusqu'à la réponse de l'UR. Le marqueur 1 appelle l'UR_n en tn θ2, le marqueur 2 appelle l'UR_n en tn θ4.

L'appel d'un marqueur est accompagné en tn θ1 d'un eb d'essai (BEMU) chargé de vérifier la continuité de la liaison dans les 2 sens et l'état en service de l'organe récepteur. L'UR doit systématiquement répondre en θ1 du même temps de la trame suivante, chaque fois que cet eb d'essai est reçu et si l'UR est "en service". Si le marqueur ne reçoit pas la réponse à l'appel (BAUM) il analyse l'eb réponse à l'essai (BEUM). En son absence le MQ ne donne pas suite à l'appel et une faute est signalée à l'organe de contrôle (OC) (FMRU).

En cas de réponse à l'essai, le marqueur maintient son appel vers l'UR qui doit répondre dès qu'elle est libre et en priorité au MQ 1 (réponse au MQ 1 en tn θ3 et au MQ 2 en tn θ5). En cas de non réponse au bout de 65 ms le MQ signale une faute à l'OC (FDEB).

b) *Transmission du message*

Le marqueur envoie les informations (MMU) accompagnées de l'eb d'imparité (BIMU) dans la trame de réponse de l'UR.

En cas de faute d'imparité détectée par l'UR dans le sens MQ → UR :

- l'UR n'émet pas l'eb "bien reçu" (BRUM)
- l'UR commute la liaison LU
- l'UR signale la faute à l'OC avec indication de la liaison mauvaise et le numéro du MQ (FIMPU)
- le MQ renouvelle une fois le message erroné, y compris la procédure appel réponse. En cas de non réponse le MQ signale une faute (FNBR).

c) *Réponse de l'UR*

En cas de bonne réception, l'UR exécute la fonction demandée par le MQ et émet le message réponse (MUM)(dans le créneau correspondant au MQ et sans procédure appel-réponse) le plus rapidement possible, compte tenu des contraintes précisées dans la spécification technique de chaque type d'organe. Ce délai peut aller de 1 trame à plusieurs dizaines de trames selon le type de l'UR (UR abonnée locale ou UR abonnée distante gérée à travers un canal sémaphore).

En cas de réception normale du message le marqueur contrôle l'imparité (BIUM) et émet le bien reçu (BRMO) dans la trame de réception du message.

En cas de faute d'imparité :

- le MQ n'émet pas l'eb "bien reçu"
- le MQ commute la liaison LU
- le MQ signale la faute à l'OC avec l'indication de la liaison mauvaise et de l'UR concernée (FIMPU)
- l'UR renouvelle 1 fois son message (toujours sans procédure appel-réponse).

Le marqueur attend ce message. En cas de non-réception, il signale une faute d'échange à l'OC et en informe le multienregistreur (MR) pour la suite de la communication.

Si le marqueur reçoit le message il effectue le contrôle d'imparité et émet le "bien reçu" dans la même trame. En cas de faute d'imparité :

- le MQ n'émet pas l'eb "bien reçu"
- le MQ signale la faute à l'OC avec l'indication de la liaison mauvaise et de l'UR concernée
- le MQ avise ensuite le MR s'il y a lieu, puis se libère.

d) Remarques

Les échanges à l'initiative du MQ se composent d'un message de commande sens MQ → UR et sont suivis d'un message de réponse dans le sens UR → MQ.

Les figures 4, 5, 6 montrent les RCPP interprétés qui décrivent les échanges à l'initiative d'un marqueur.

CAM : est un compteur qui simule le délai de 64 ms accordé à un MQ pour répéter son appel.

TEMPI : est une variable représentant le délai de réponse d'une UR après exécution de la fonction demandée par le MQ.

CROM, C, CPTTRAM : sont des variables utilisées pour contrôler les répétitions des appels et des messages.

MQD, URD : sont deux drapeaux qui sont mis à 1 quand un marqueur ou une UR sont respectivement libres et à zéro s'ils sont occupés.

AUR : variable externe qui se met à 1 pour provoquer une demande d'échange à l'initiative d'un marqueur.

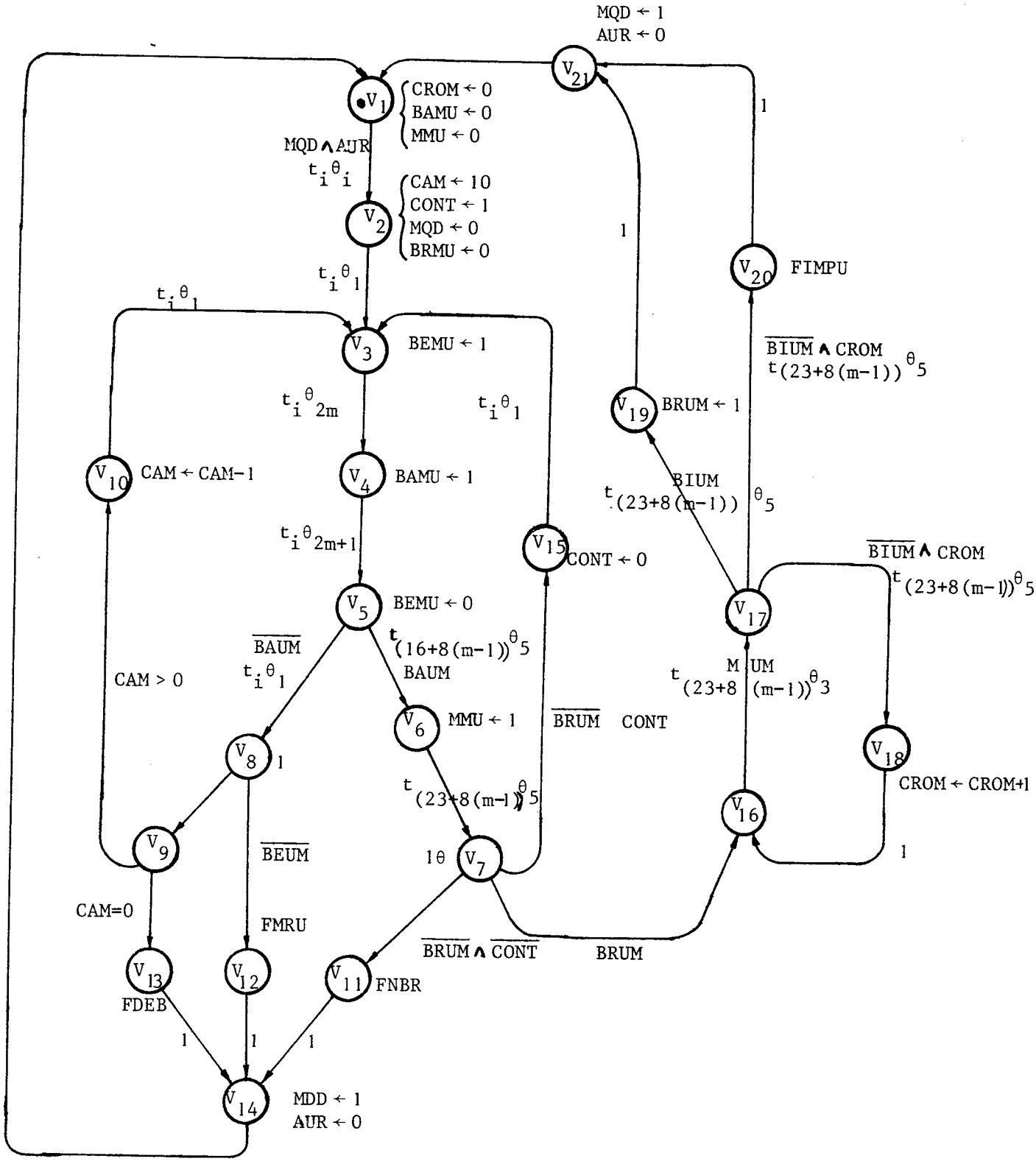


Figure 4.

RCPP interprétés représentant les échanges à l'initiative d'un marqueur MQ_m dans ce marqueur.

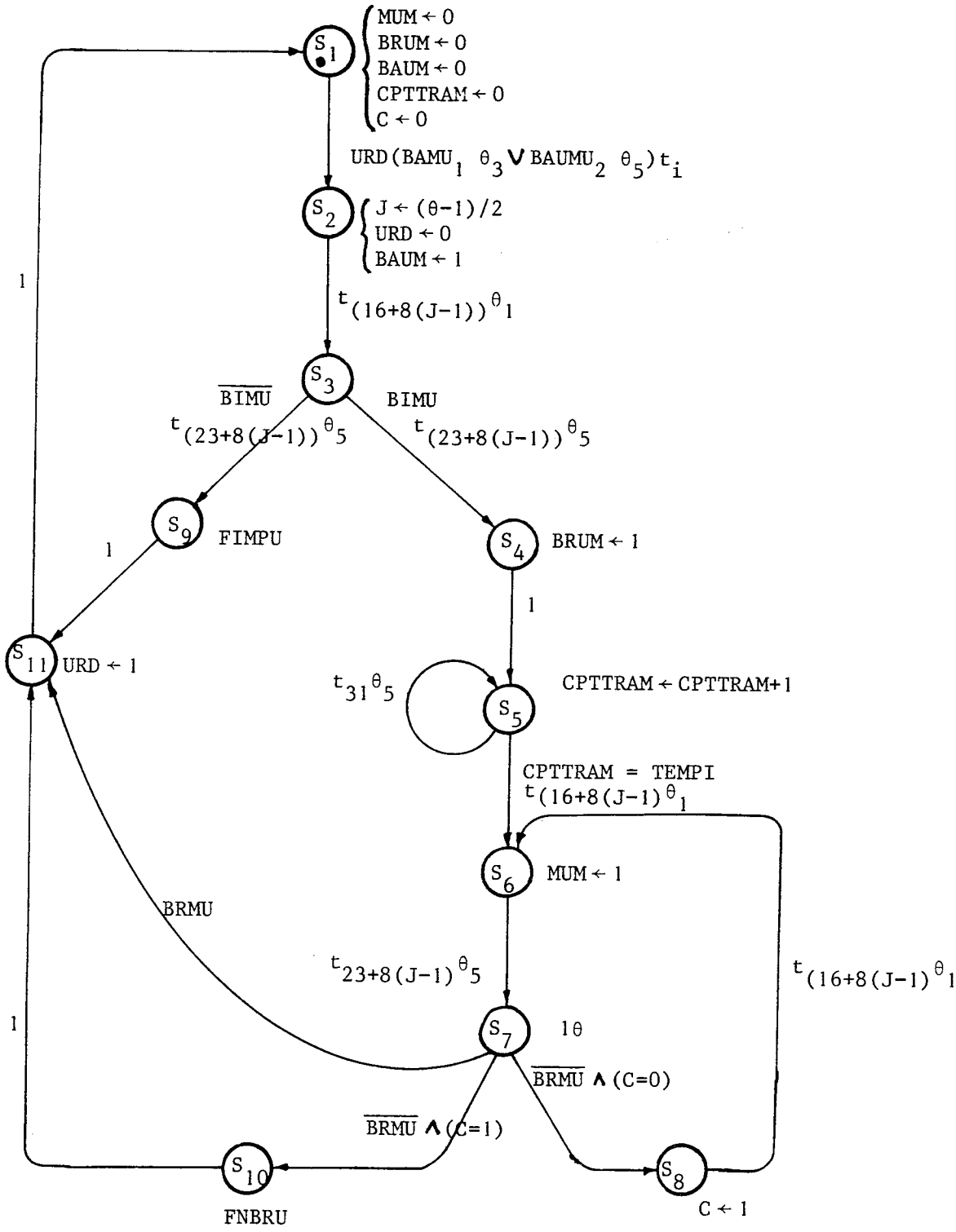


Figure 5.

RCPP interprétés représentant les échanges à l'initiative d'un marqueur dans l'UR$_i$.

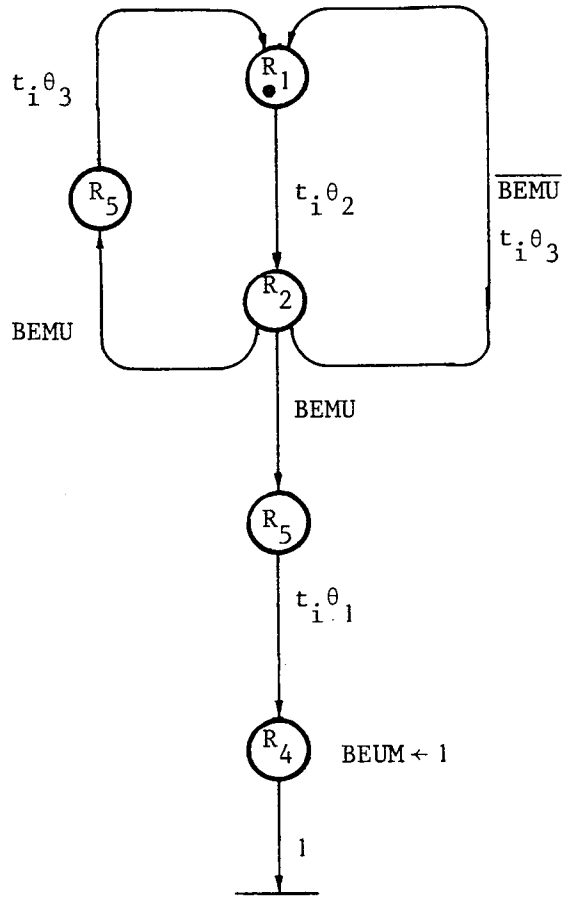


Figure 6.

RCPP représentant la réponse d'une UR au bit d'essai envoyé par un MQ.

Echanges à l'initiative de l'UR

a) Procédure d'appel-réponse

L'UR rappelle séparément chaque marqueur : appel URn → MQ1 en tn θ2 et appel URn → MQ2 en tn θ4. (BAUM)

Une UR ayant un échange à réaliser avec un MQ commence par appeler l'un quelconque des 2 MQ pendant 8 trames. En cas de non-réponse du MQ appelé, elle appelle l'autre MQ pendant 8 trames. L'alternance des appels UR vers chacun des 2 MQ se poursuivra ainsi pendant 256 ms à l'issue desquelles l'UR cesse l'émission d'appels et émet un message de faute sur l'OC (FNR). L'utilisation du tn θ4 pour l'appel vers MQ 2 suppose que l'eb de réponse à l'essai de la liaison par l'appel du marqueur soit émis en tn θ1 de la trame suivante sur les LU paires.

Les marqueurs répondent dans le tn de l'appel en θ3 pour le MQ1 et en θ5 pour le MQ2. (BAMU).

La réponse de MQ 1 dans une UR inhibe tous les échanges avec le MQ 2 dans cet UR.

b) Transmission de message

l'UR émet son message dans la trame de réponse du MQ ou dans la trame suivante, et dans le créneau correspondant à ce MQ. (MUM).

Si le contrôle d'imparité est bon,le MQ émet l'eb "bien reçu". (BRMU)
S'il est mauvais :

- le MQ n'émet pas le "bien reçu"
- le MQ commute la liaison LU
- le MQ signale une faute à l'OC précisant la liaison mauvaise et le numéro de l'UR puis se libère (FINP)
- l'UR renouvelle son message y compris la procédure appel-réponse comme s'il s'agissait d'un nouvel échange, mais sans qu'il y ait une nouvelle recherche de voie libre dans l'UR, en cas de nouvel appel. En cas de non réponse l'UR signale une faute (FNBR).

Les figures 7 et 8 montrent les RCPP interprétés décrivant les échanges à l'initiative d'une UR.

CAME : simule la trame de 256 ms accordée à une UR pour repéter les appels.

DU : est une variable externe qui se met à 1 pour provoquer un échange à l'initiative d'une UR.

L'alternance de numéro de MQ par série de 8 trames n'est pas représentée dans le réseau. Au lieu de cela l'UR choisit au hasard un MQ à chaque appel (?N : ? est l'opérateur APL qui effectue un tirage de nombres pseudo-aléatoires entre 1 et N).

Le programme MAS de description des liaisons LU est donné par la suite.

Un problème a été posé par cette description : l'accès à l'horloge du système pour tester les voies temporelles t_i et les θ_i correspondants. A cause des restrictions du langage dans ce domaine (Cf. Chap.III § IV), le temps a été simulé par un module séparé (HRLG) ayant une interface commune avec tous les autres modules. Les variables T et TH représentent les t_i et les θ_i respectivement.

Le fonctionnement du système a été simulé sur une configuration de 2 MQ et de 5 UR.

Le temps CPU consommé par cette simulation est de l'ordre de 4 minutes pour la simulation d'une trame.

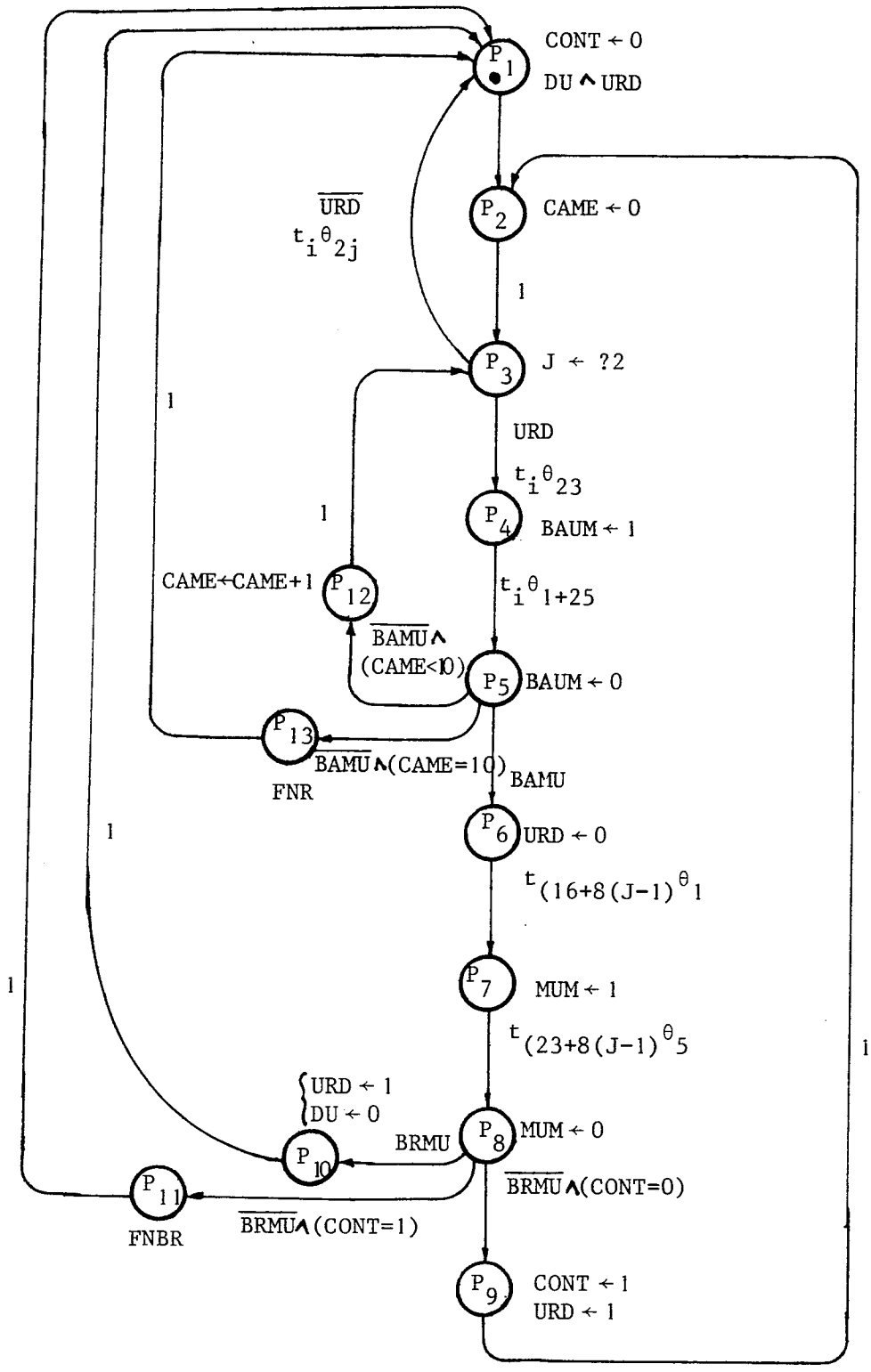


Figure 7.

RCPP interprétés représentant les échanges à l'initiative d'une UR_i dans cette UR.

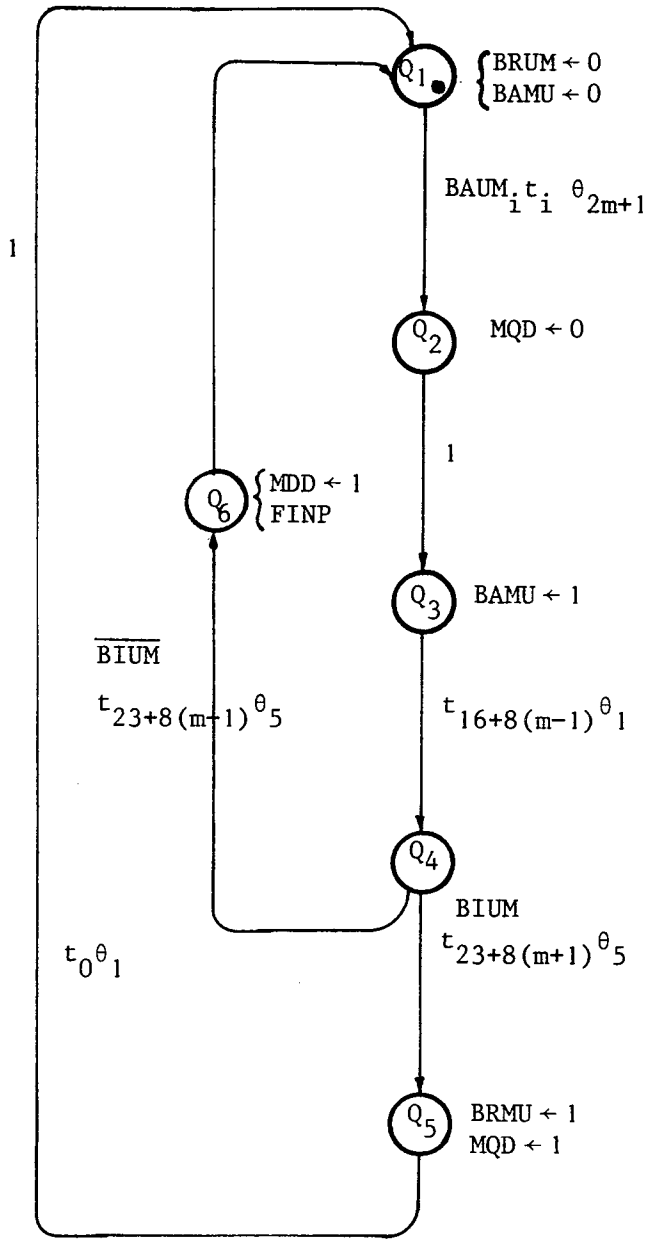


Figure 8.

RCPP interprété représentant les échanges à l'initiative d'une UR dans un marqueur

PROGRAMME MAS

```
DCLMOD UR(BAUM(2),BAMU(2),MUM(2),BRMU(2),BEMU(2),BRUM(2),BEUM(2),BIMU,TEMPI,
          DU,URD,T,TH)
   DCLPROC P1
      TEMP 0
      CONT←0
   FIN
   DCLPROC P2
      TEMP 0
      CAME←0
   FIN
   DCLPROC P3
      TEMP 0
      J←?2
   FIN
   DCLPROC P4
      TEMP 0
      BAUM[J]←1
   FIN
   DCLPROC P5
      TEMP 1
      BAUM[J]←0
   FIN
   DCLPROC P6
      TEMP 0
      URD←0
   FIN
   DCLPROC P7
      TEMP 0
      MUM[J]←1
   FIN
   DCLPROC P8
      TEMP 1
      MUM[J]←0
   FIN
   DCLPROC  P9
      TEMP 0
      CONT←URD←1
   FIN
   DCLPROC P10
      TEMP 0
      URD←1
      DU←0
   FIN
   DCLPROC P11
      TEMP 0
      URD←1
      DU←0
      'FNBR'
   FIN
   DCLPROC P12
      TEMP 0
      CAME←CAME+1
   FIN
   DCLPROC P13
      TEMP 0
      URD←1
      DU←0
      'FNR'
   FIN
```

```
DCLPROC S1
   TEMP 0
   MUM[⍳2]←BRUM[⍳2]←BAUM[⍳2]←CPTTRAME←C←0
FIN
DCLPROC S2
   TEMP 0
   J←(TH-1)÷2
   BAUM[J]←1
   URD←0
FIN
DCLPROC S4
   TEMP 0
   BRUM[J]←1
FIN
DCLPROC S5
   TEMP 0
   CPTTRAME←CPTTRAME+1
FIN
DCLPROC S6
   TEMP 0
   MUM[J]←1
FIN
DCLPROC S7
   TEMP 1
FIN
DCLPROC S8
   TEMP 0
   C←1
FIN
DCLPROC S9
   TEMP 0
   'FIMPU'
FIN
DCLPROC S10
   TEMP 0
   'FNBRU'
FIN
DCLPROC  S11
   TEMP 0
   URD←1
FIN

DCLPROC R4
   TEMP 0
   BEUM[J]←1
FIN
DCLCTRL ASYN
   DEBUT
   P1  SI       DU∧URD              ALORS P2
   P2  SI         1                 ALORS P3
   P3  SI URD∧(T=I)∧(TH=2×J)        ALORS P4
   P3  SI (~URD)∧(T=I)∧(TH=2×J)     ALORS P1
   P4  SI   (T=I)∧(TH=1+2×J)        ALORS P5
   P5  SI     BAMU[J]               ALORS P6
   P5  SI (CAME<10)∧(~BAMU[J])      ALORS P12
   P5  SI (CAME=10)∧(~BAMU[J])      ALORS P13
   P6  SI (T=16+8×(J-1))∧ TH=1      ALORS P7
```

```
P7  SI  (T=23+8×(J-1))∧TH=5          ALORS P8
P8  SI  (~BRMU[J])∧CONT=0            ALORS P9
P8  SI          BRMU[J]              ALORS P10
P8  SI  (~BRMU[J])∧CONT=1            ALORS P11
P9  SI              1                ALORS P2
P10 SI              1                ALORS P1
P11 SI              1                ALORS P1
P12 SI              1                ALORS P3
P13 SI              1                ALORS P1
S1  SI URD∧(T=I)∧((BAMU[1]∧TH=3)∨BAMU[2]∧TH=5) ALORS S2
S2  SI   (T=16+8×(J-1))∧TH=1         ALORS S3
S3  SI  BIMU∧(T=23+8×(J-1))∧TH=5     ALORS S4
S3  SI (~BIMU)∧(T=23+8×(J-1))∧TH=5   ALORS S9
S4  SI              1                ALORS S5
S5  SI (CPTTRAME=TEMPI)∧(T=16+8×(J-1))∧TH=1   ALORS S6
S5  SI     (T=31)∧TH=5               ALORS S5
S6  SI   (T=23+8×(J-1))∧TH=5         ALORS S7
S7  SI   (~BRMU[J])∧C=0              ALORS S8
S7  SI   (~BRMU[J])∧C=1              ALORS S10
S7  SI        BRMU[J]                ALORS S11
S8  SI   (T=16+8×(J-1))∧TH=1         ALORS S6
S9  SI              1                ALORS S11
S10 SI              1                ALORS S11
S11 SI              1                ALORS S1
R1  SI      (T=I)∧TH=2               ALORS R2
R2  SI        BEMU[J]                ALORS R3,R5
R2  SI    (~BEMU[J])∧TH=3            ALORS R1
R3  SI      (T=I)∧TH=1               ALORS R4
R4  SI              1                ALORS ~R4
R5  SI          TH=3                 ALORS R1
FIN
```

```
DCLMOD MARQ(BAMU(5),BAUM(5),MUM(5),BRMU(5),BEMU(5),BEUM(5),BRUM(5),MMU(5),BIUM,
          MQD,AUR,I,T,TH)
   DCLPROC Q1
      TEMP 0
      BRMU[I]←BAMU[I]←0
   FIN
   DCLPROC Q2
      TEMP 0
      I←T
      MQD←0
   FIN
   DCLPROC Q3
      TEMP 0
      BAMU[I]←1
   FIN
   DCLPROC Q5
      TEMP 0
      BRMU[I]←MQD←1
   FIN
   DCLPROC Q6
      TEMP 0
      MQD←1
      'FINP'
   FIN
```

```
DCLPROC V1
   TEMP 0
   BAMU[I]←MMU[I]←CROM←0
FIN
DCLPROC V2
   TEMP 0
   CAM←10
   CONT←1
   MQD←BRMU[⍳5]←0
FIN
DCLPROC V3
   TEMP 0
   BEMU[I]←1
FIN
DCLPROC V4
   TEMP 0
   BAMU[I]←1
FIN
DCLPROC V5
   TEMP 1
   BEMU[I]←0
FIN
DCLPROC V6
   TEMP 0
   MMU[I]←1
FIN
DCLPROC V7
   TEMP 1
FIN
DCLPROC V8
   TEMP 1
FIN
DCLPROC V10
   TEMP 0
   CAM←CAM-1
FIN
DCLPROC V11
   TEMP 0
   'FNBR'
FIN
DCLPROC V12
   TEMP 0
   'FMRU'
FIN
DCLPROC V13
   TEMP 0
   'FDEB'
FIN
DCLPROC V14
   TEMP 0
   MQD←1
   AUR←0
FIN
DCLPROC V15
   TEMP 0
   CONT←0
FIN
```

```
DCLPROC V18
   TEMP 0
   CROM←CROM+1
FIN
DCLPROC V19
   TEMP 0
   BRMU[I]←1
FIN
DCLPROC V20
   TEMP 0
   'FIMPU'
FIN
DCLPROC V21
   TEMP 0
   MQD←1
   AUR←0
FIN
DCLCTRL ASYN
   DEBUT
   Q1  SI (∨/BAUM[⍳5]∧(T=⍳5))∧(TH=1+2×M)   ALORS Q2
   Q2  SI            1                     ALORS Q3
   Q3  SI    (T=16+8×(M-1))∧TH=1           ALORS Q4
   Q4  SI  BIUM∧(T=23+8×(M-1))∧TH=5        ALORS Q5
   Q4  SI  (~BIUM)∧(T=23+8×(M-1))∧TH=5     ALORS Q6
   Q5  SI        (T=0)∧TH=1                ALORS Q1
   Q6  SI            1                     ALORS Q1
   V1  SI  AUR∧MQD∧(T≤I)∧TH=1              ALORS V2
   V2  SI        (T=I)∧TH=1                ALORS V3
   V3  SI        (T=I)∧TH=2×M              ALORS V4
   V4  SI        (T=I)∧(TH=1+2×M)          ALORS V5
   V5  SI BAUM[I]∧(T=16+8×(M-1))∧TH=1      ALORS V6
   V5  SI (~BAUM[I])∧(T=I)∧TH=1            ALORS V8
   V6  SI    (T=23+8×(M-1))∧TH=5           ALORS V7
   V7  SI          BRUM[I]                 ALORS V16
   V7  SI    (~BRUM[I])∧CONT               ALORS V15
   V7  SI    (~BRUM[I])∧~CONT              ALORS V11
   V8  SI          BEUM[I]                 ALORS V9
   V8  SI         ~BEUM[I]                 ALORS V12
   V9  SI          CAM>0                   ALORS V10
   V9  SI          CAM=0                   ALORS V13
   V10 SI        (T=I)∧TH=1                ALORS V3
   V11 SI            1                     ALORS V14
   V12 SI            1                     ALORS V14
   V13 SI            1                     ALORS V14
   V14 SI            1                     ALORS V1
   V15 SI         (T=I)∧TH=1               ALORS V3
   V16 SI MUM[I]∧(T=23+8×(M-1))∧TH=3       ALORS V17
   V17 SI BIUM∧(T=23+8×(M-1))∧TH=5         ALORS V19
   V17 SI (~BIUM)∧(~CROM)∧(T=23+8×(M-1))∧TH=5  ALORS V18
   V17 SI (~BIUM)∧CROM∧(T=23+8×(M-1))∧TH=5 ALORS V20
   V18 SI            1                     ALORS V16
   V19 SI            1                     ALORS V21
   V20 SI            1                     ALORS V21
   V21 SI            1                     ALORS V1
FINMOD
```

```
DCLMOD HRLG(T,TH)
   DCLPROC D
      TEMP 1
      TH←(TH=5)+(TH<5)×(TH+1)
      T←T+((T<31)-(T=31)×T)×(TH=1)
   FIN
   DCLCTRL ASYN
      DEBUT
      D  SI   1     ALORS D
FINMOD


DCLCONF
   CREE   MQ1,MQ2=MARQ
          UR1,UR2,UR3,UR4,UR5=UR
          HR=HRLG
   LIE    MQ1.BAMU(1)=UR1.BAMU(1)
          MQ1.BAMU(2)=UR2.BAMU(1)
               .
               .
               .
          MQ1.BAMU(5)=UR5.BAMU(1)
          MQ2.BAMU(1)=UR1.BAMU(2)
          MQ2.BAMU(2)=UR2.BAMU(2)
               .
               .
               .
          MQ1.BAUM(1)=UR1.BAUM(1)
               .
               .
               .
          MQ1.BRMU(1)=UR1.BRMU(1)
               .
               .
               .
               .
          MQ1.T=HR.T
          MQ1.TH=HR.TH
               .
               .
               .
          UR5.TH=HR.TH
FIN

INIT      MQ1.Q1,MQ2.Q1,MQ1.V1,MQ2.V1=1
          UR1.P1,UR2.P1,UR3.P1,UR4.P1,UR5.P1=1
          UR1.S1,UR2.S1,UR3.S1,UR4.S1,UR5.S1=1
          UR1.R1,UR2.R1,UR3.R1,UR4.R1,UR5.R1=1
          HR.D=1
          MQ1.M=1  MQ2.M=2  UR1.I=1  UR2.I=2  .......  UR5.I=5
          MQ1.MQD,MQ2.MQD=1  UR1.URD,UR2.URD, .........,UR5.URD=1
          HR.T=0  HR.TH=1

DONNEES
     3 MQ2.AUR=1  MQ2.I=3 ;  7 UR2.DU=1 ;  15 UR5.DU=1 ;  23 MQ1.AUR=1 MQ2.I=1 ;
       ..............
FIN
```

II - MODELISATION DE K PROCESSEURS PIPE-LINE

Cet exemple (fourni par la SINTRA) traite le cas de la coordination et du parallélisme de K processeurs fonctionnant en pipe-line, pour calculer la Transformée de Fourier Discrète (DFT*) de $N = 2^K$ échantillons par un algorithme de Transformée de Fourier Rapide (FFT**) [24].

L'algorithme de FFT est une technique efficace pour calculer le DFT d'une série temporelle. Les coefficients du DFT sont calculés itérativement, ce qui entraîne un gain de temps considérable.

Pour un nombre de $N = 2^K$ échantillons, l'algorithme a besoin de K étapes; chaque étape de FFT se réalise en N/2 cycles.

Chacun des K processeurs, p_i, effectue une étape de FFT et dès qu'il a terminé déclenche le processeur suivant p_{i+1} qui commence une autre étape. Dès que K jeux d'échantillons ont été introduits, tous les processeurs doivent fonctionner simultanément en pipe-line sans blocage et sans perte de temps.

L'algorithme FFT

RAPPEL : Pour une série $\{X_\ell\}$, $\ell=0,\ldots, N-1$, le DFT est défini par :

$$Ar = \sum_{\ell=0}^{N-1} X_\ell \exp(-2\Pi jr\ell/N),\ r = 0,..,.., N-1$$

où $j = \sqrt{-1}$ et X_ℓ peut être un nombre complexe.

On considère les deux sous-séries $\{Y_\ell\}$ et $\{Z_\ell\}$ de $\{X_\ell\}$ constituées des termes de rang pair et impair respectivement.

$$Y_\ell = X_{2\ell}$$
$$Z_\ell = X_{2\ell+1} \quad ,\ell = 0,\ 1,\ 2,\ldots,\ N/2-1 \quad (N = 2^K)$$

---

* Discret Fourier Transform

** Fast Fourier Transform

Leurs DFT sont donnés par :

$$Br = \sum_{\ell=0}^{N/2-1} Y_\ell \exp(-4\Pi jr\ell/N)$$

$$Cr = \sum_{\ell=0}^{N/2-1} Z_\ell \exp(-4\Pi jr\ell/N)$$

, r = 0, 1, 2,... N/2-1

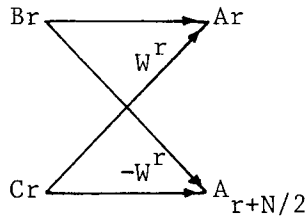
Alors le DFT de $\{X_\ell\}$ peut s'écrire :

$$Ar = Br + Cr \exp(-2\Pi jr/N)$$

$$Ar_{+N/2} = Br - Cr \exp(-2\Pi jr/N)$$

, r = 0, 1, 2,..., N/2-1

Une paire de ces équations correspond à une opération "Butterfly" (papillon) :



où $W^r = \exp(-2\Pi jr/N)$

et $0 \leq r < N/2$

La série $\{X_\ell\}$ de N échantillons ($N = 2^K$) est divisée en sous-séries successives des éléments pairs et impairs (Fig.9).

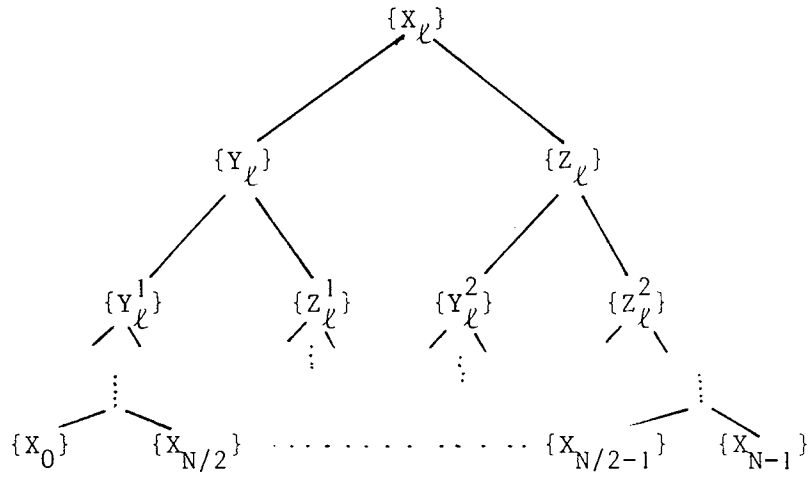


Figure 9.

L'algorithme consiste à appliquer successivement l'opération papillon comme le montre la figure 10.

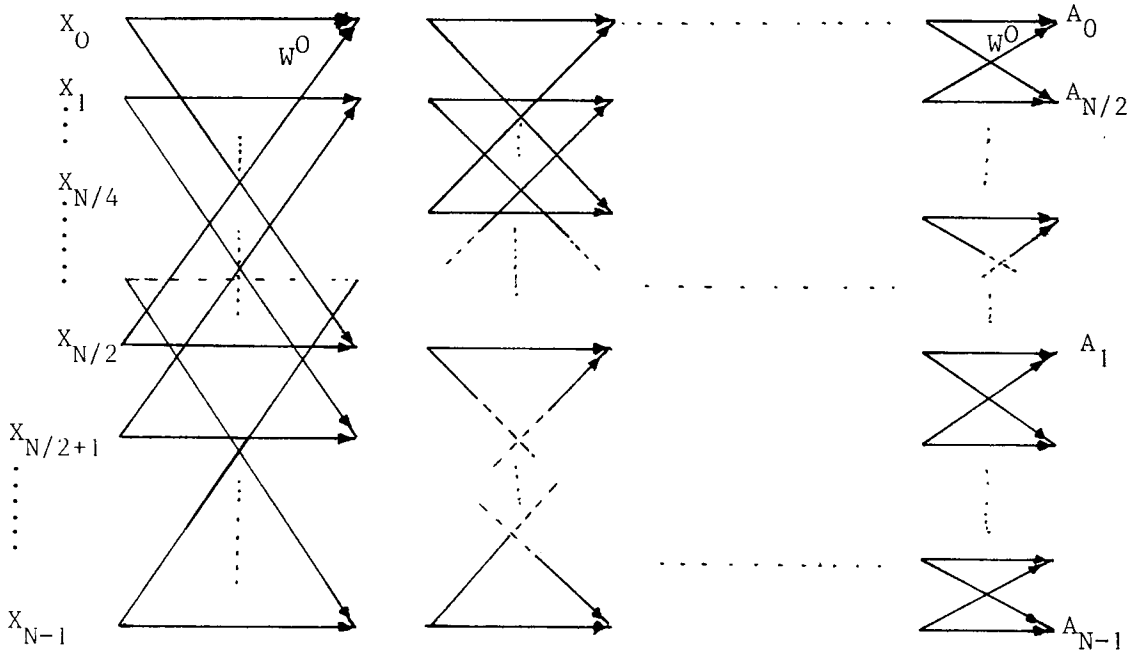


Figure 10.

On remarque que les coefficients A_i du DFT de $\{X_\ell\}$ ainsi obtenus ne sont pas rangés dans l'ordre naturel mais dans un ordre dit "décalage symétrique des bits" :

si $(b)_2$: la représentation binaire du nombre i

$(b')_2$: le nombre binaire obtenu en décalant symétriquement les bits de $(b)_2$

et i' : la représentation décimale de $(b')_2$, alors A_i se trouve à l'adresse i' dans la mémoire.

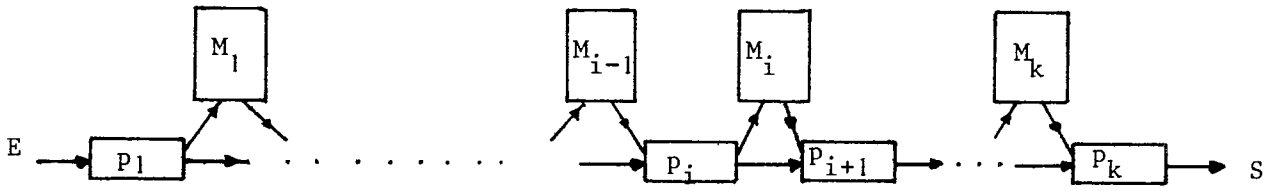
Fonctionnement du système

Un processeur typique pi qui effectue une étape de FFT se compose de deux sous-processeurs synchronisés entre eux.

- l'un est chargé du calcul des adresses pour la lecture et/ou l'écriture
- l'autre :
  - . lit les données (stockées dans la mémoire tampon M_{i-1}) aux adresses calculées par le dit sous-processeur,
  - . effectue l'opération papillon sur les données,
  - . range les données transformées en mémoire tampon M_i aux adresses calculées (Fig.11).

Ceci constitue un cycle du fonctionnement du processeur p_i.

Chaque processeur est muni d'un compteur modulo N/2 (initialisé à 1) qui compte le nombre de cycles effectués par p_i; quand p_i aura exécuté une étape de FFT (N/2 cycles), il déclenche le processeur suivant p_{i+1}.



E : entrée de N échantillons disponibles

S : sorties de coefficients du DFT

Figure 11.

Le réseau de la Figure 12 met en évidence le fonctionnement d'un processeur typique P_i pour un cycle de la FFT.

On remarque que la lecture (phase P_3) et l'écriture (P_5) ne peuvent être faites qu'après les calculs d'adresses (P_7) et (P_8) respectives.

De même, on ne recommence pas les calculs d'adresses (P_7) et (P_8) qu'après que la lecture (P_3) ou l'écriture (P_5) soient achevées.

Le processeur P_i commence l'étape i de FFT dès que sa variable d'interface INIT1 sera mise à 1 par son prédécesseur P_{i-1}. La variable d'interface INIT2 sera mise à 1 (phase S2) quand l'étape i aura fini, pour initialiser le processeur P_{i+1}.

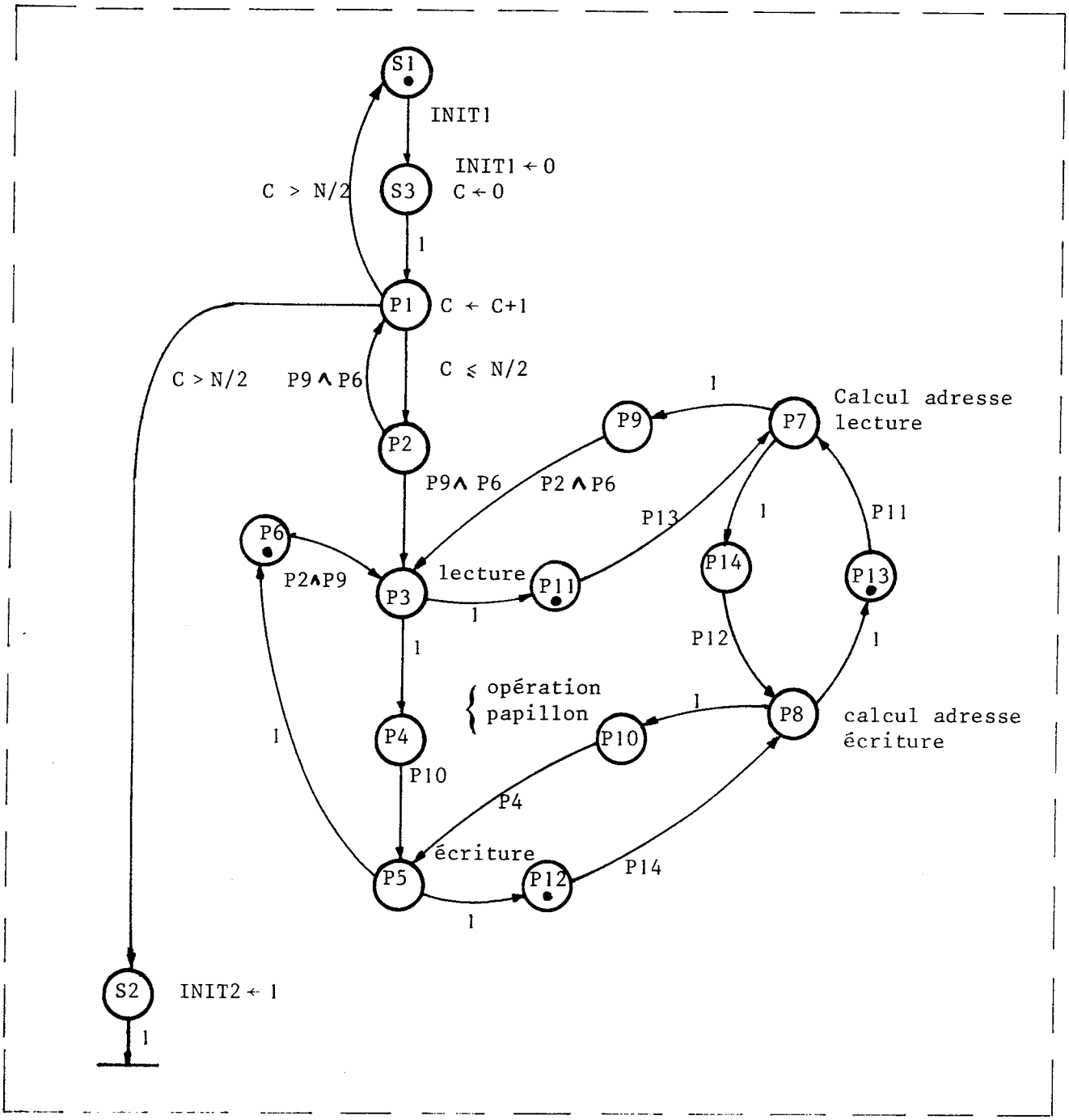


Figure 12.

Fonctionnement d'un processeur p_i

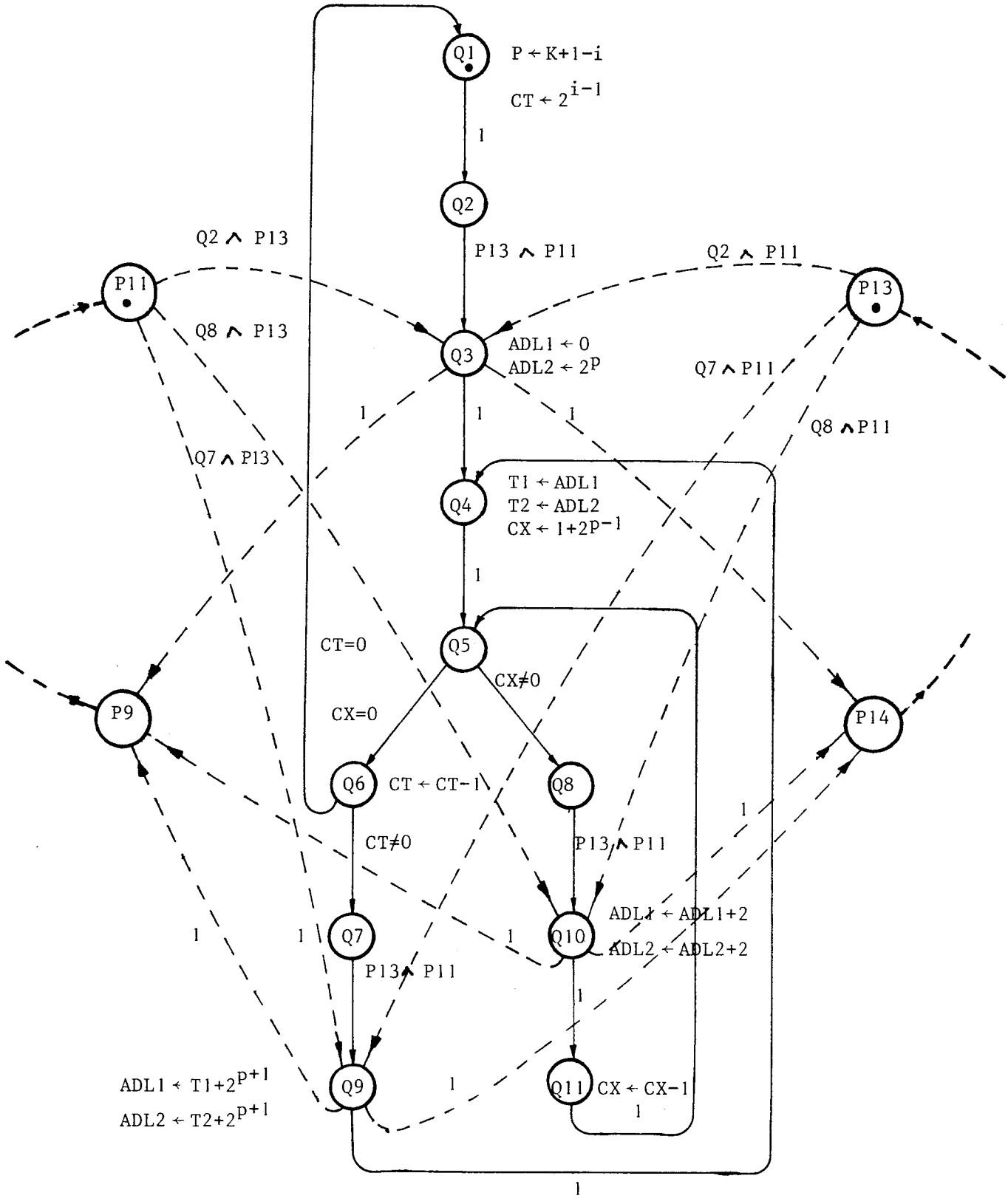


Figure 13.

Calcul des adresses

Calcul des adresses

La figure 13 montre le RCPP interprété représentant le sous-processeur, chargé du calcul des adresses. Ce réseau doit remplacer la phase P7 du réseau de la Figure 12. Nous verrons également de quelle façon le réseau est lié avec les phases P9, P11, P13 et P14.

Le calcul de l'adresse d'écriture se fera d'une façon analogue.

A l'étape i, effectuée par le processeur p_i, on applique l'opération papillon N/2 fois sur 2^{i-1} groupes des 2^{K-i} variables rangées dans des adresses successives (Fig. 10). Si AD1 est l'adresse où commence un groupe des variables les adresses du groupe suivant commencent à $AD1+2^{K-i+2}$. Chaque variable occupe 2 mots mémoire, étant donné qu'il s'agit de nombres complexes.

ADL1, ADL2 : sont des variables représentant les adresses de lecture

CX, CT : sont des compteurs pour les nombres de groupes et de variables

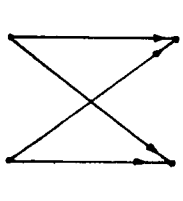
T1, T2 : sont des variables d'adresse (début de chaque groupe).

Opération papillon

Puisqu'il s'agit des nombres complexes, l'opération papillon peut-être détaillée comme suit :

Soient $Q_1 = Q_{1r} + j\,Q_{1i}$, $Q_2 = Q_{2r} + j\,Q_{2i}$ deux nombres complexes,

et $Q'_1 = Q'_{1r} + j\,Q'_{i1}$, $Q'_2 = Q'_{2r} + j\,Q'_{2i}$ deux nombres complexes obtenus de Q_1 et Q_2 par l'opération papillon :



$$Q'_1 = Q_1 + W^r Q_2 \qquad \text{où } W^r = \exp\left(-\frac{2\Pi j}{N} r\right)$$

$$Q'_2 = Q_1 - W^r Q_2 \qquad = \exp(-j\theta)$$

$$\text{avec } \theta = \frac{2\Pi r}{N}$$

Il s'ensuit que :

$$Q'_{1r} = Q_{1r} + (Q_{2r}\cos\theta + Q_{2i}\sin\theta)$$

$$Q'_{1i} = Q_{1i} + (Q_{2i}\cos\theta - Q_{2r}\sin\theta)$$

$$Q'_{2r} = Q_{1r} - (Q_{2r} \cos\theta + Q_{2i} \sin\theta)$$

$$Q'_{2i} = Q_{1i} - (Q_{2i} \cos\theta - Q_{2r} \sin\theta)$$

On peut représenter ce calcul par un programme parallèle (Fig. 14).

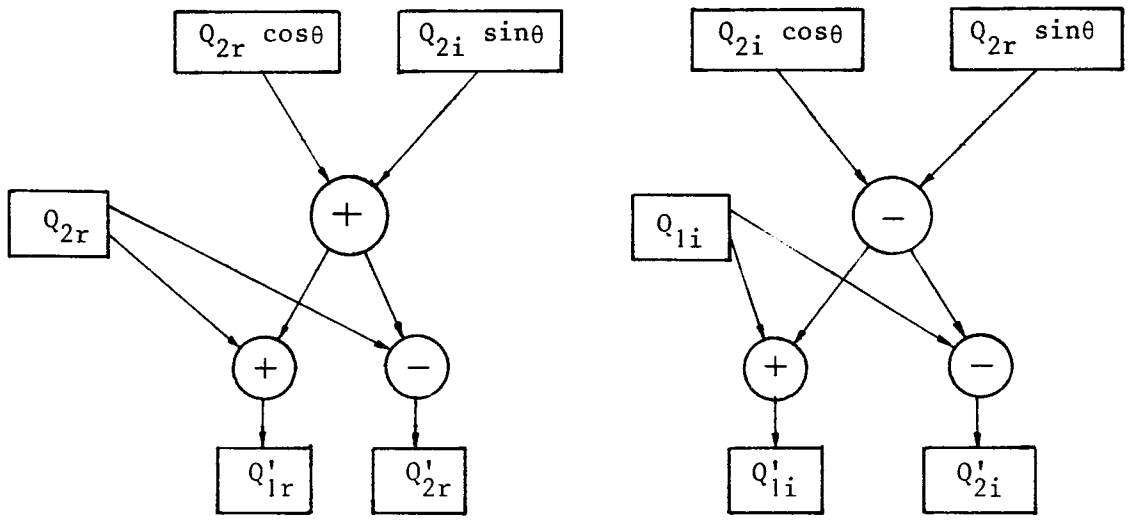


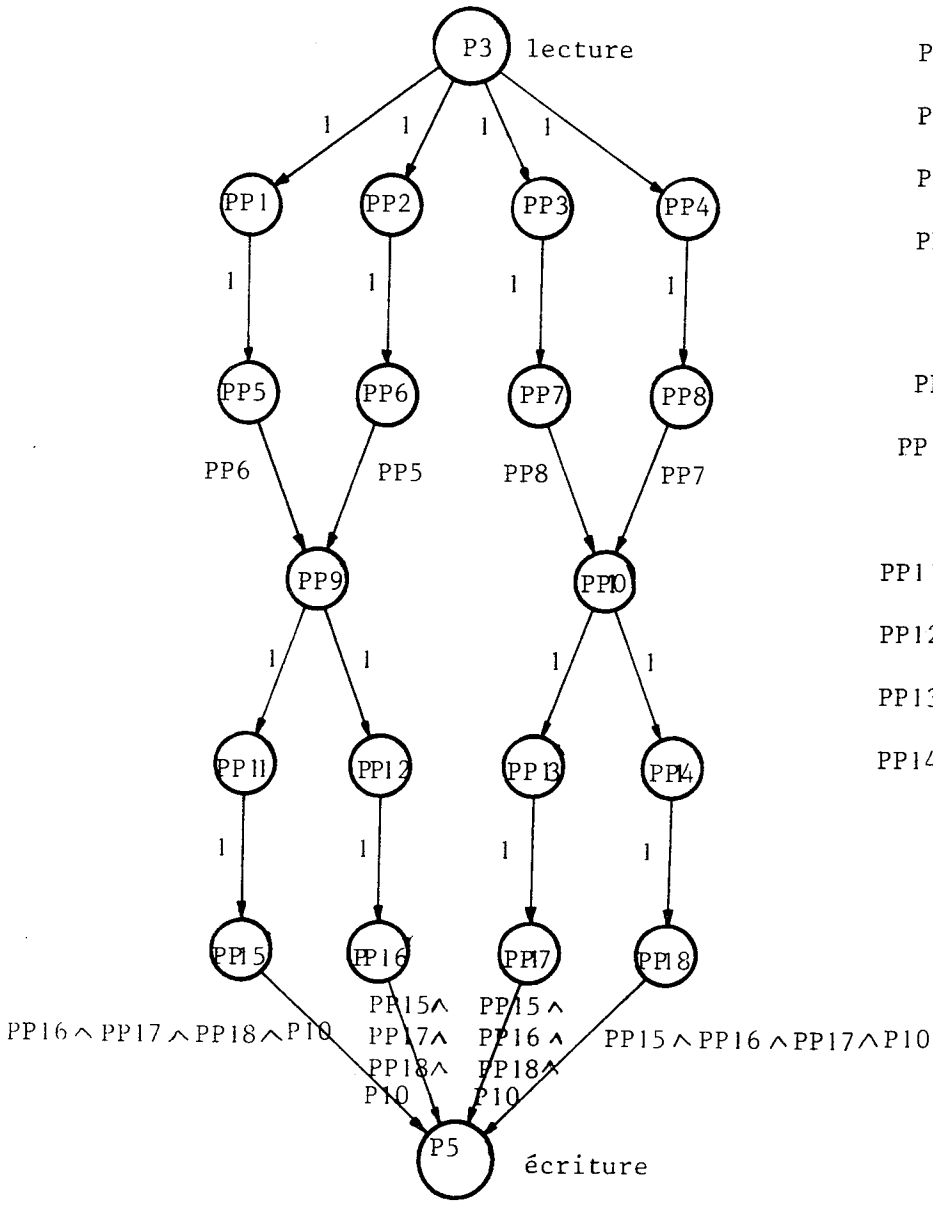
Figure 14.

La figure 15 montre le RCPP interprété décrivant l'opération papillon. Ce réseau remplacera la phase P4 de la figure 12.

Par la suite, il est donné le programme MAS décrivant le système multiprocesseur. Dans ce programme on ne détaille pas les opérations de lecture et d'écriture. Les calculs s'effectuent toujours sur deux variables Q1 et Q2 dans lesquelles on devrait lire le contenu des adresses ADL1 et ADL2. A1,A2,..,A4 sont des variables de calcul.

REMARQUE

Cet exemple montre les facilités que présente le langage MAS pour une description progressive des systèmes. De la description générale donnée en figure 12 on peut aboutir à la description détaillée en remplaçant progressivement les phases par des sous-réseaux.



PP1 : $A_1 \leftarrow Q_{2r} \cos\theta$

PP2 : $A_2 \leftarrow Q_{2i} \sin\theta$

PP3 : $A_3 \leftarrow Q_{2i} \cos\theta$

PP4 : $A_4 \leftarrow Q_{2r} \sin\theta$

PP9 : $A_1 \leftarrow A_1 + A_2$

PP10 : $A_2 \leftarrow A_3 - A_4$

PP11 : $A_4 \leftarrow Q_{1r} + A_1$

PP12 : $A_3 \leftarrow Q_{1r} - A_1$

PP13 : $A_1 \leftarrow Q_{1i} + A_2$

PP14 : $A_2 \leftarrow Q_{1i} - A_2$

Figure 15

Opération papillon

PROGRAMME MAS

```
DCLMOD PROCESS(INIT1,INIT2,I,N)
   DCLPROC P1
      TEMP 1
      C←C+1
   FIN
   DCLPROC P3
      TEMP tl        } Lecture
   FIN
   DCLPROC P5
      TEMP te        } Ecriture
   FIN
   DCLPROC P8
      TEMP tca       } Calcul adresse
   FIN               } écriture
   DCLPROC S2
      TEMP 1
      INIT2←1
   FIN
   DCLPROC S3
      TEMP 1
      INIT1←C←0
   FIN
   DCLPROC Q1
      TEMP 1
      P←K+1-I
      CT←2*(I-1)
   FIN
   DCLPROC Q3
      TEMP 1
      ADL1←0
      ADL2←2*P
   FIN
   DCLPROC Q4
      TEMP 1
      T1←ADL1
      T2←ADL2
      CX←1+2*(P-1)
   FIN
   DCLPROC Q6
      TEMP 1
      CT←CT-1
   FIN
   DCLPROC Q9
      TEMP 1
      ADL1←T1+2*(P+1)
      ADL2←T2+2*(P+1)
   FIN
   DCLPROC Q10
      TEMP 1
      ADL1←ADL1+2
      ADL2←ADL2+2
   FIN
   DCLPROC Q11
      TEMP 1
      CX←CX-1
   FIN
```

```
DCLPROC PP1
   TEMP 1
   A1←QR2×2OTH
FIN
DCLPROC PP2
   TEMP 1
   A2←QI2×1OTH
FIN
DCLPROC PP3
   TEMP 1
   A3←QI2×2OTH
FIN
DCLPROC PP4
   TEMP 1
   A4←QR2×1OTH
FIN
DCLPROC PP9
   TEMP 1
   A1←A1+A2
FIN
DCLPROC PP10
   TEMP 1
   A2←A3-A4
FIN
DCLPROC PP11
   TEMP 1
   A3←QR1+A1
FIN
DCLPROC PP12
   TEMP 1
   A4←QR1-A1
FIN
DCLPROC PP13
   TEMP 1
   A1←QI1+A2
FIN
DCLPROC PP14
   TEMP 1
   A2←QI1-A2
FIN
DCLCTRL ASYN
   DEBUT
   S1   SI INIT1     ALORS S3
   S3   SI   1       ALORS P1
   P1   SI C≤N÷2     ALORS P2
   P1   SI C>N÷2     ALORS S1,S2
   P2   SI P9∧P6     ALORS P3,P1
   P3   SI   1       ALORS P11,PP1,PP2,PP3,PP4
   P5   SI   1       ALORS P6,P12
   P6   SI P2∧P9     ALORS P3
   P8   SI   1       ALORS P10,P13
   P9   SI P2∧P6     ALORS P3
   P10  SI   P4      ALORS P5
   P11  SI P13∧Q7    ALORS Q9
   P11  SI P13∧Q8    ALORS Q10
   P11  SI P13∧Q2    ALORS Q3
```

```
      P12  SI    P14      ALORS P8
      P13  SI  P11∧Q2     ALORS Q3
      P13  SI  P11∧Q7     ALORS Q9
      P13  SI  P11∧Q8     ALORS Q10
      P14  SI     1       ALORS P8
      S2   SI     1       ALORS ~S2
      Q1   SI     1       ALORS Q2
      Q2   SI  P13∧P11    ALORS Q3
      Q3   SI     1       ALORS Q4,P9,P14
      Q4   SI     1       ALORS Q5
      Q5   SI  CX=0       ALORS Q6
      Q5   SI  CX≠0       ALORS Q8
      Q6   SI  CT=0       ALORS Q1
      Q6   SI  CT≠0       ALORS Q7
      Q7   SI  P13∧P11    ALORS Q9
      Q8   SI  P13∧P11    ALORS Q10
      Q9   SI     1       ALORS Q4,P9,P14
      Q10  SI     1       ALORS Q11,P9,P14
      Q11  SI     1       ALORS Q5
      PP1  SI     1       ALORS PP5
      PP2  SI     1       ALORS PP6
      PP3  SI     1       ALORS PP7
      PP4  SI     1       ALORS PP8
      PP5  SI    PP6      ALORS PP9
      PP6  SI    PP5      ALORS PP9
      PP7  SI    PP8      ALORS PP10
      PP8  SI    PP7      ALORS PP10
      PP9  SI     1       ALORS PP11,PP12
      PP10 SI     1       ALORS PP13,PP14
      PP11 SI     1       ALORS PP15
      PP12 SI     1       ALORS PP16
      PP13 SI     1       ALORS PP17
      PP14 SI     1       ALORS PP18
      PP15 SI PP16∧PP17∧PP18∧P10 ALORS P5
      PP16 SI PP15∧PP17∧PP18∧P10 ALORS P5
      PP17 SI PP15∧PP16∧PP18∧P10 ALORS P5
      PP18 SI PP15∧PP16∧PP17∧P10 ALORS P5
FINMOD


DCLCONF
   CREE  PROC1,PROC2,.....,PROCk=PROCESS
   LIE   PROC1.INIT2=PROC2.INIT1
         PROC2.INIT2=PROC3.INIT1
                .
                .
                .
         PROC(k-1).INIT2=PROCk.INIT1
FIN

INIT     PROC1.S1,PROC2.S1,....,PROCk.S1=1
         PROC1.I=1  PROC2.I=2  ..... PROCk.I=k
DONNEES
         1  PROC1.INIT1=1    ;   ti  PROC1.INIT1=1   ;
             ......................

 FIN
```

CONCLUSION

Cette étude a montré que l'outil proposé n'était pas seulement un langage de simulation mais constituait une aide réelle à la conception d'un système complexe : description progressive, validation fonctionnelle, aide aux choix matériels (par l'étude du comportement temporel).

Les extensions de cette étude concernent :

α) L'amélioration de ce langage, en lui ajoutant certaines primitives facilitant l'étude de la validation fonctionnelle et de la protection fonctionnelle contre les pannes des systèmes.

β) L'implémentation de ce langage en temps réel;

- étude de la faisabilité d'une implémentation temps réel, sur un matériel donné, d'un système décrit et simulé par ce langage,
- étude d'un interpréteur réalisant automatiquement l'implémentation d'un système décrit par MAS sur un matériel donné (microprocesseur).

ANNEXE

ANNEXE

SITUATION DU LANGAGE MAS PAR RAPPORT AUX LANGAGES CHDL*

Nous rappelons brièvement les niveaux de description des systèmes et les 2 types de langage CHDL.

I - NIVEAUX DE DESCRIPTION [25]

- *Description algorithmique* : le comportement du système est spécifié en terme des relations entre ses variables d'entrée et de sortie.
- *Description fonctionnelle* : le système est décrit comme un algorithme, en terme de ses éléments de mémorisation. Les opérateurs peuvent ou non correspondre à son implémentation matérielle réelle. Le temps doit être pris en compte.
- *Description structurelle* : description de la réalisation matérielle; les opérateurs sont des éléments matériels, les variables des éléments de mémorisation.

II - TYPE DE LANGAGE [25]

- *Procédurel* : l'algorithme est décrit par une séquence d'instructions (étapes), avec des possibilités de contrôle du déroulement, par des instructions conditionnelles.
- *Non procédurel* : l'ordre d'écriture des instructions n'est pas significatif. Le déroulement des opérations dépend des tests "permanents" sur les variables d'état, associées aux instructions.

---

* CHDL : Computer Hardware Design Language

III - CARACTERISTIQUES DE MAS

Il a été clairement discuté dans [11],[12],[13] les options prises pour MAS et elles sont présentées dans le chapitre II de ce mémoire :

α) Langage composé de 2 sous-langages
- un de type procéduriel pour la P.O
- un de type non-procéduriel pour la P.C

β) Il est adapté à une description fonctionnelle, permettant la démarche de description progressive souhaitée.

IV - SITUATION DES AUTRES LANGAGES DE CONCEPTION

Le tableau suivant résume les caractéristiques de quelques langages CHDL. Il paraît peut être illusoire de les classer d'une façon rigide; néanmoins, le tableau ci-dessous essaye de situer les langages l'un par rapport à l'autre en fonction des critères déjà mentionnés.

Langage	niveau de description	type de langage	séparation PO-PC	Remarques
DDL [26]	Structurel	non procéduriel	non	
CDL [27]	Structurel	non procéduriel	non	
LOTIS [28]	Structurel	non procéduriel	non	une certaine séparation de la PC commence à apparaître
AHPL [29]	Structurel	procéduriel	oui	basé sur la notation d'APL
CASSANDRE [30]	Structurel	non procéduriel	non	
LASCAR [31]	Structurel	non procéduriel	non	possibilité d'une description fonctionnelle par l'utilisation des procédures FORTRAN et ASSEMBLEUR
SL - 1 [32]	Structurel	non procéduriel	oui	utilisés ensemble dans un système (System Control Processor)
FL - 1 [32]	Fonctionnel	procéduriel	oui	
DIGITEST II [33]	Multiniveau	procéduriel	oui	il utilise DIGITEST pour décrire la structure PL/1 pour le fonctionnement et le "Graphe de Contrôle" pour la P.C
DESY [34]	Fonctionnel	non procéduriel	-	langage de description de RdP

BIBLIOGRAPHIE

[1] M. MOALLA, J. SIFAKIS : "A Design Methodology for Complex Logical Systems", 2nd International Symp. on Discrete Systems, Dresden, March 1977.

[2] M. MOALLA : "L'Approche Fonctionnelle dans la Vérification des Systèmes Informatiques. Proposition d'un ensemble de Méthodologies", Thèse de Docteur Ingénieur, Université de Grenoble, Décembre 1976.

[3] R.M. KARP, R.E. MILLER : "Parallel Program Schemata : a Mathematical Model for Parallel Computation", IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory, October 1967.

[4] C.A. PETRI : "Communication with automata", Technical Rep. n° RADC-TR-65-377, vol.1, Rome Air Develop. Center, Griffis Air Force Base, New-York, Jan. 1966.

[5] C.W. ROSE : "LOGOS, and the Software Engineer", Proc. Fall Joint Computer Conf, AFIPS Press, pp. 311-323, 1972.

[6] V.G. CERF : "Multiprocessors, Semaphores and a Graph Model of Computation", Ph. D., UCLA 1972.

[7] R. VALETTE : "Sur la Description, l'Analyse et la Validation des Systèmes de Commande Parallèles", Thèse d'Etat, Univ. Paul Sabatier, Toulouse, Novembre 1976.

[8] V.M. GLUSKOV, A.A. LETICHEVSKII : "Theory of Algorithms and Discrete Processors", Advances in Information Systems Science, Vol.1, Chap I, pp.1-58, Edited by Julius T.TOU.

[9] J.M. DUMAS : "L'Organiphase : Un Modèle facilitant l'Analyse et la Synthèse des Automates Logiques, Thèse 3e cycle, Univ. de Montpellier, Avril 1974.

[10] J. SIFAKIS : "Modèles Temporels des Systèmes Logiques", Thèse Doc. Ing, Université de Grenoble, mars 1974.

[11] M. MOALLA, J. SIFAKIS, M. ZACHARIADES : "Un Langage d'Aide à la Conception et à la Simulation de Systèmes Complexes". Rapport de Recherche ENSIMAG, RR. n°16, Oct. 1975.

[12] M. MOALLA, G. SAUCIER, J. SIFAKIS, M. ZACHARIADES : "A Design Tool for the Multilevel Description and Simulation of Systems of Interconnected Modules", 3rd Annual Symp. on Comp. Architecture, Tanapa, Fla, Jan. 1976.

[13] P. LE DANOIS, M. MOALLA, G. SAUCIER, J. SIFAKIS, M. ZACHARIADES : "Multilevel Description and Simulation of Parallel Cooperating Simulation of Parallel Cooperating Processors", Kolloquium über das Parallelismus in der. Informalik, Erlangen, RFA, juin 1976.

[14] M. MOALLA, J. SIFAKIS, M. ZACHARIADES : "MAS, Un Outil d'Aide à la Description et à la Conception des Automatismes Logiques", Colloque AFCET : Automatismes Logiques, Recherches et Application Industrielles, Paris 6-8. Dec. 1976.

[15] K.E. IVERSON : "A Programming Language", New York, J. Willey, 1972.

[16] M. ZACHARIADES : "Langage d'Aide à la Conception des Systèmes Logiques", Rapport D.E.A. ENSIMAG, Juin 1974.

[17] M. GRIFFITHS, P. PELTIER : "Grammar Transformation as an Aid to Compiler Production", Centre Scientifique IBM France, 1968.

[18] G. DENARS, J.C. RAULT, G. RUGGIU : "Le Langage et les Systèmes APL", Masson et Cie, Paris 1974.

[19] D.E. KNUTH : "The Art of Computer Programming", Vol.3, Addison Wesley, Reading, Mars 1973.

[20] J.G. VAUCHER, P. DUVAL : "A Comparison of Simulation Event List Algorithms", Communications of the ACM, vol.18, n° 4, pp. 223-230, April 1975.

[21] F.F. SELERS Jr., M.Y. HSIAO, L.N. BEARNSON : "Analyzing Errors with the Boolean Difference", IEEE Trans. Comput., vol.EC-17, pp.676-683, July 1968.

[22] C. BELLON : "Degradation Progressive dans un Système Distribué", Thèse de 3e cycle, INPG, Sept. 1977.

[23] J.L. DAUPHIN : "Spécification Technique ST/RCI/6 édition 2, J.L. DAUPHIN : "Système E10/128, Liaison entre unités de raccordement et marqueurs (LU) et liaison de test (LU'), CNET, octobre 1976.

[24] Doc. 18.634 : SINTRA.

[25] M.R. BARBACCI : "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems", IEEE Trans. on Comp., vol.C-24, n°2, pp.137-150, Février 1975.

[26] D.L. DIETMEYER : "Introducing DDL", Computer, December 1974.

[27] Y. CHU : "Introducing Computer Design Language", Digest of Papers, Compcon 72, San Francisco, September 1972.

[28] H.P. SCHLAEPPI : "A Formal Language for Describing Machine Logic, Timing and Sequencing (LOTIS)". IEEE Trans. vol. EC-13, Aug. 1964.

[29] F.J. HILL, G.R. PETERSON : "Digital Systems; Hardware Organization and Design", John Willey & Sons, 1973.

[30] F. ANCEAU, P. LIDDEL, J. MERMET, C. PAYAN : "CASSANDRE : A Language to Describe Digital Systems, Applications to Logic Design". 3rd Int. Symp. Comp. and Inf. Science, Miami, F.10, Dec. 1969.

[31] D. BORRIONE : "LASCAR, un Langage pour la Simulation et l'Evaluation des Architectures d'Ordinateurs", Thèse de 3e cycle INPG, Avril 1976.

[32] F. RENCHEL : "A System Control Processor for a Microcomputer Network", Rapport UCLA-ENG-7646, June 1976.

[33] F.J. RAMMIG : "DIGITEST II : An Integrated Structural and Behavioral Language", Int. Symp. on Comp. Hardware Description Languages and Their Applications, New-York, Sept. 1975.

[34] M. COURVOISIER : "Etude des Systèmes Logiques de Commandes Asynchrones à Evolutions Simultanées", Thèse d'Etat, Univ. Paul Sabatier, Toulouse, Février 1974.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

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

VU le rapport de présentation de :

- Madame G. SAUCIER, Maître de Conférences à l'Institut National Polytechnique de Grenoble

Mademoiselle Marianthi ZACHARIADES

est autorisée à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie Informatique".

Grenoble, le 20 Juillet 1977

Ph. TRAYNARD
Président
de l'Institut National Polytechnique