Un formalisme pour la spécification des contrôleurs temps-réel de procédés discrets

Claudio Walter

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
Génie Informatique

par

Cláudio WALTER

UN FORMALISME POUR LA SPECIFICATION DES CONTROLEURS TEMPS-REEL DE PROCEDES DISCRETS

Thèse soutenue le 24 juin 1981 devant la Commission d'Examen :

Monsieur	L. BOLLIET	Président
Messieurs	F. ANCEAU	Examinateurs
	M. JOUMARD	
	Ph. JORRAND	
	R. PERRET	
	G. VERROUST	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

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

PROFESSEURS DES UNIVERSITES

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

.../...

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

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

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

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

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

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

.../...

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

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

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

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

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

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

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

E.N.S.H.G.

M. DELHAYE Jean-Marc

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

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

A mes parents,

à Lia et à Daniel.

Je tiens à remercier

Monsieur L. BOLLIET, Professeur à l'Université de Grenoble II, qui a bien voulu me faire l'honneur de présider le jury de cette thèse,

Monsieur F. ANCEAU, Professeur à l'Institut National Polytechnique de Grenoble, qui a bien voulu m'accepter au sein de l'Equipe de Recherche en Architecture des Ordinateurs, et pour m'avoir prodigué ses conseils, ses critiques et ses encouragements tout au long de cette étude,

Monsieur G. VERROUST, Professeur Associé à l'Institut de Physique Nucléaire d'Orsay, qui a bien voulu juger mon travail et faire partie du jury,

Monsieur M. JOUMARD, Directeur Technique à la Societé MERLIN-GERIN,
Monsieur Ph. JORRAND, Maître de Recherches au CNRS, et
Monsieur R. PERRET, Directeur du Laboratoire d'Automatique de Grenoble,
pour avoir accepté de participer du jury,

Monsieur H.G. MENDELBAUM, Professeur à l'Université Paris V et Monsieur J. SIFAKIS, Chargé de Recherches au CNRS pour leurs conseils précieux,

tous mes collègues, membres de l'équipe de Recherche en Architecture des Ordinateurs, avec qui j'ai eu des échanges fructueux,

Madame H. DIAZ et Madame C. CHALAND, pour leur dévouement et leur rapidité à dactilographier ce texte,

le Service de Réprographie de l'ENSIMAG, qui a assuré le tirage de ce document,

l'Universidade Federal do Rio Grande do Sul, Porto Alegre (BRASIL),
la CAPES (Coordenação de Aperfeiçoamento do Pessoal de Ensino Superior),
la FAPERGS (Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul)
et le Ministère Français des Affaires Etrangères,
pour leur soutien financier.

S O M M A I R E

CHAPITRE 1

INTRODUCTION

1. LES LANGAGES DE SPECIFICATION
2. PLAN DU TRAVAIL

INTRODUCTION

Parallèlement au développement de l'informatique, on assiste à une prolifération, en quantité aussi bien qu'en complexité, des applications de celle-ci au contrôle de procédés en temps réel.

Ainsi qu'il arrive dans plusieurs activités techniques, il n'existe pas de panacée généralement acceptée, mais plutôt un spectre assez large de solutions adaptées aux particularités de certains problèmes. Une excellente analyse de l'état de l'art à travers la classification des problèmes et des solutions, a été développée par [GERT 75].

Un des domaines qui a reçu beaucoup d'attention dans les dernières années est constitué par les procédés discrets. Un procédé est dit discret, ou discrétisable, si ses variables apparentes admettent un domaine discret de valeurs. Ces procédés sont souvent associés aux procédés continus dans les applications réelles.

1. LES LANGAGES DE SPECIFICATION

Si la pénétration de l'informatique est due à un constant perfectionnement technique lié à une diminution des coûts, elle devient "vendable" au moment où les éventuels utilisateurs disposent de moyens de communication avec les ressources informatiques, c'est-à-dire quand ils disposent de moyens plus ou moins faciles d'exprimer leurs désirs de façon non ambigüe. L'informatique a des moyens techniques et économiques pour résoudre un grand ensemble de problèmes ; la question est maintenant de savoir les poser.

On peut mettre en évidence quelques classes d'applications où la question de la spécification a été plus ou moins résolue :

a) l'algorithmique scientifique : le langage FORTRAN a été spécialement orienté vers les utilisations scientifiques. La nature séquentielle, ou facilement séquentialisable, des problèmes auxquels ce langage est très bien adapté, permet aux utilisateurs d'écrire normalement leurs propres programmes.

b) La gestion : le langage COBOL a été développé pour satisfaire les besoins de la gestion comptable, du personnel, de la production, ... Pourtant, et probablement due à la complexité et au volume des fichiers, une bonne connaissance du système d'exploitation est souvent nécessaire, ce qui a amené la création d'une couche intermédiaire d'analystes et de programmeurs. Les problèmes de spécification sont là déjà plus importants.

c) Le contrôle de processus : cette dénomination très générale comprend les applications qui imposent les contraintes suivantes :
   i) temps réel : le système de contrôle évolue en parallèle avec son environnement (le procédé contrôlé), ce qui introduit des contraintes de temps à respecter ;
   ii) parallélisme du procédé : le procédé contrôlé est composé de plusieurs dispositifs évoluant en parallèle et interconnectés, dont la synchronisation doit être assurée.

Ces caractéristiques se retrouvent dans la conception des systèmes d'exploitation des ordinateurs d'usage général. Or, vue leur complexité, la spécification des systèmes d'exploitation est réservée à des informaticiens spécialistes dans ce domaine.

La spécification des systèmes de contrôle de processus doit être réalisée essentiellement par des spécialistes des domaines d'application, que ce soit des ingénieurs ou des techniciens, physiciens, chimistes, éventuellement avec la collaboration d'automaticiens. Bien que les connaissances en informatique soient de plus en plus répandues, leur niveau n'est et ne sera pas pendant longtemps, suffisant pour permettre l'expression des problèmes complexes de parallélisme, par exemple.

De plus, les contraintes de sécurité d'un système de contrôle de processus sont, de par la nature même des dispositifs concernés, plus sévères que celles d'un système d'exploitation d'usage général. Une défaillance matérielle ou logicielle dans une aciérie peut avoir des conséquences bien plus graves que l'arrêt de l'ordinateur dans un centre de calcul.

La spécification d'un système de contrôle de processus est un document qui matérialise un accord entre les responsables du procédé contrôlé et les responsables de la réalisation de son contrôleur. Elle constitue donc un instrument de communication et un instrument juridique.

En tant qu'instrument de communication, la spécification doit être, dans la mesure du possible, facile à écrire, à lire et à modifier.

En tant qu'instrument juridique, la spécification doit permettre l'établissement des responsabilités. En cas de mauvais fonctionnement de l'installation, il est important de pouvoir établir si la panne est due à une mauvaise spécification (faute de l'utilisateur), ou à une implémentation incorrecte (faute informatique). Ce découpage net est assez idéaliste ; il peut cependant être considéré comme un but à atteindre.

Le programme de spécification, écrit dans un langage déterminé, n'est pas nécessairement utilisé comme moyen de programmation directe du contrôleur. Il doit servir comme point de départ pour la réalisation du programme de contrôle, en passant éventuellement par des étapes intermédiaires qui peuvent être franchies manuellement, ou de préférence automatiquement (figure 1.1.).

Ce travail est principalement consacré à la spécification des contrôleurs de systèmes discrets ou discrétisés, c'est-à-dire des systèmes dont chaque variable apparente admet un ensemble fini d'états. Les systèmes discrets paraissent souvent couplés à des systèmes continus dans les applications. Par ailleurs, ils constituent une classe particulière de systèmes de contrôle, avec en général des caractéristiques et des problèmes qui leur sont propres.

Dans une forte mesure, les progrès dans la spécification des systèmes ont consisté jusqu'à présent en une élévation du niveau des primitives des *langages de programmation*. Cette approche est réaliste (la réalisation de chaque niveau étant assurée par l'existence a priori du niveau immédiatement inférieur). Mais le chemin à parcourir entre les désirs de l'utilisateur et le programme qui les réalise est encore long.

Le but est d'avoir un langage aussi proche que possible de l'utilisateur, en laissant le maximum de travail de transformation à l'équipe chargée de la réalisation du contrôleur. Ce but a été atteint pour quelques classes de processus relativement simples, avec des langages orientés vers les applications, comme par exemple pour la spécification de l'usinage de pièces mécaniques [HATV 73].

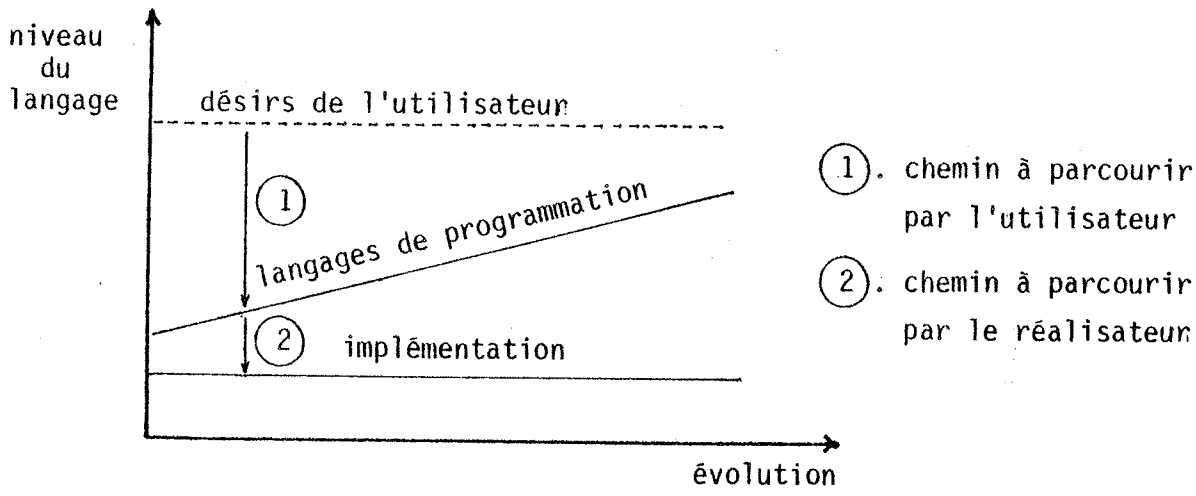


Figure 1.1. Spécification et programmation

Pour les problèmes plus complexes et généraux, les efforts de spécification sont assez récents et peu testés dans l'environnement industriel. La plupart des applications sont actuellement spécifiées à l'aide d'un cahier des charges plus ou moins informel. Ensuite, une équipe de programmeurs traduit ce cahier des charges en un programme, en général à l'aide de langages d'assemblage ou de langages de programmation classiques.

La distance entre l'utilisateur et les langages disponibles est telle que, très fréquemment, l'utilisateur n'a qu'une vague idée des possibilités offertes par les ordinateurs et de ce qu'il pourrait en tirer. Par conséquent, le cahier des charges est en grande partie écrit par le personnel informatique lui-même.

L'adoption de langages de spécification plus accessibles permettrait une plus grande acceptation des outils informatiques, avec un bénéfice certain, par les utilisateurs aussi bien que par les informaticiens et les fournisseurs de matériel informatique.

Spécification, programmation, vérification :

La spécification d'un contrôleur industriel correspond à une phase précise dans la procédure de conception d'un système de contrôle (figure 1.2.).

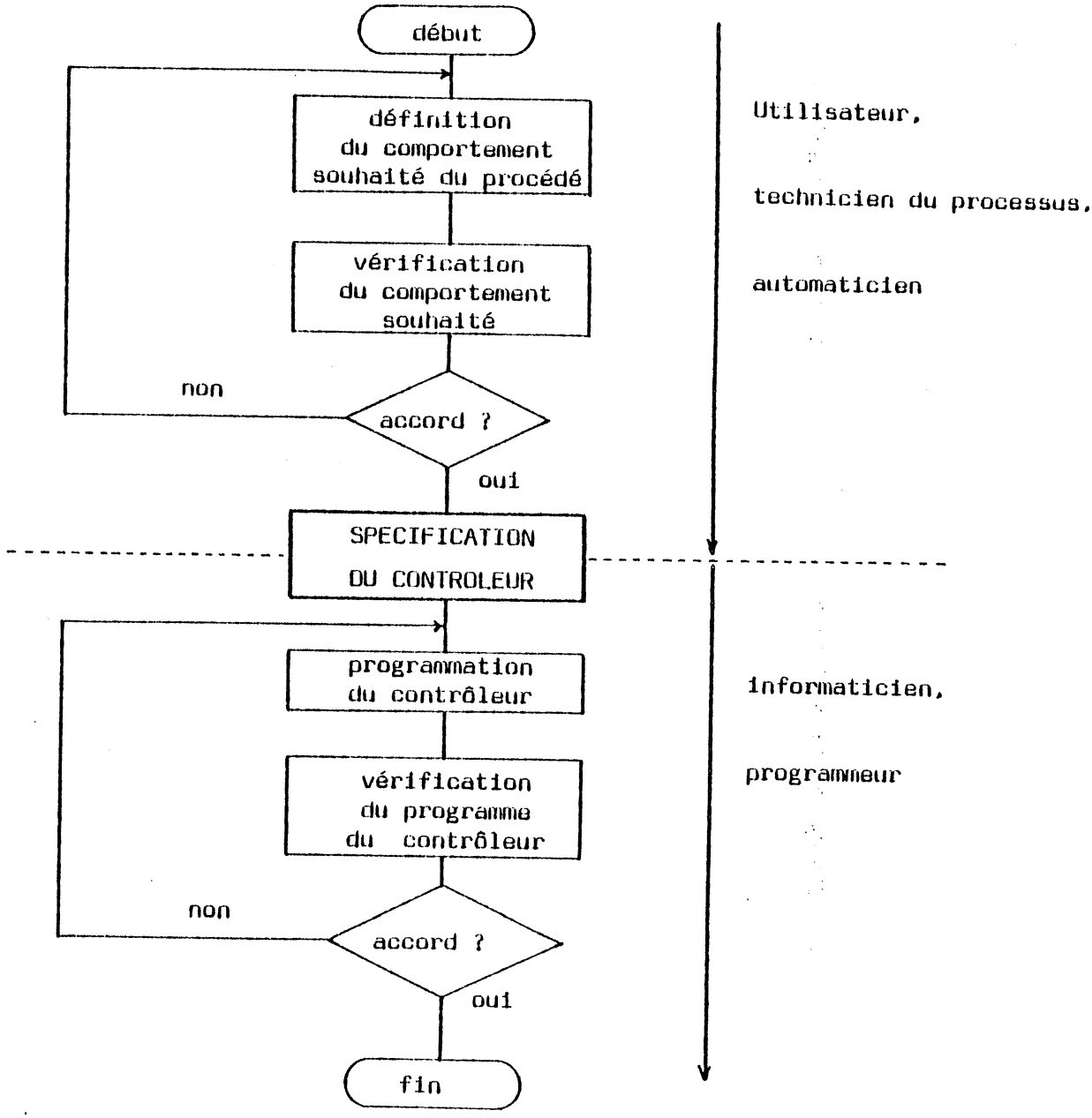


Figure 1.2. Conception d'un programme de contrôle

La spécification du contrôleur est le document d'interface entre l'équipe qui définit le contrôleur et l'équipe qui le réalise.

Dans le diagramme de la figure 1.2. on remarque deux phases de vérification :
a) la vérification du comportement souhaité du procédé,
b) la vérification du programme écrit, par rapport à la spécification du contrôleur.

La vérification du comportement souhaité s'effectue avant la spécification du contrôleur. Ce que l'on vérifie est la cohérence du processus. En fait, les systèmes de contrôle de procédés peuvent être caractérisés par deux aspects :
. le *volume* d'informations concernant le procédé,
. la *complexité* des interactions entre les composants du procédé.

Les problèmes de vérification découlent de ce dernier aspect. Les interactions à vérifier sont plutôt rares et normalement isolables dans les systèmes industriels. Pourtant, ils peuvent être de solution difficile, et les utilisateurs disposent d'outils spécifiques pour chaque type de problème. Pour les systèmes continus, la vérification concerne principalement la stabilité ; pour les systèmes discrets, la vérification concerne la disponibilité des ressources nécessaires pour réaliser chaque étape de la séquence des événements prévus. Pour ces systèmes, un outil de vérification très répandu est le réseau de Pétri, avec ses extensions [PETE 77].

Une fois vérifiée la cohérence du comportement souhaité, l'utilisateur spécifie la fonction du contrôleur qui satisfait ce comportement.

A partir de ce document, l'équipe informatique conçoit le système de contrôle (architecture matérielle, programmation). Il y a ensuite un rebouclage programmation - vérification, jusqu'à ce que le programme satisfasse les spécifications du contrôleur

2. PLAN DU TRAVAIL

Le point de départ de ce travail est la présentation d'un formalisme général de description (chapitre 2) qui induit une méthode d'analyse et de structuration du contrôleur. L'objectif est de proposer une méthodologie (chapitre 3) pour la spécification,et éventuellement la programmation, des systèmes de contrôle de processus en temps réel.

Les concepts de base de cette méthodologie sont en réalité utilisés intuitivement dans tous les projets de systèmes de contrôle, mais le fait de les rendre explicites fournit une méthode unifiée qui nous semble particulièrement appropriée à la description de contrôleurs de systèmes ayant un degré élevé de parallélisme.

2.1. Formalisme et méthodologie de description

Le modèle sur lequel est basée la méthodologie par des hypothèses suivantes (machine de Mealy) [MEAL 55] :

- pour commander un procédé, il faut disposer ou reproduire toutes les composantes de son vecteur d'état nécessaires pour le calcul de la commande ;
- si l'on connaît l'état complet du procédé, la fonction de commande devient purement combinatoire (fig. 1.3).

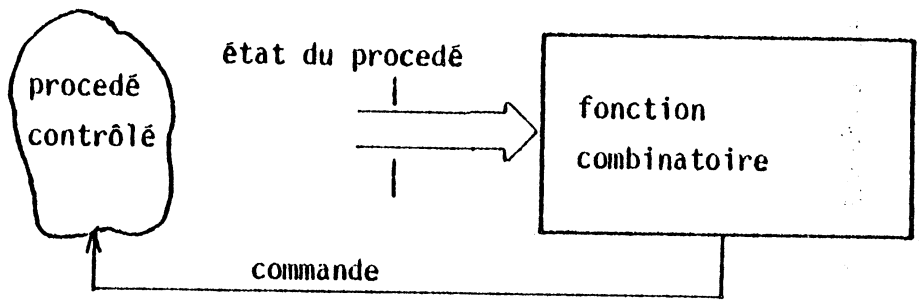


Figure 1.3. Fonction de commande

La fonction de commande combinatoire correspond aux fonctions proposées par J. Backus dans la Programmation Fonctionnelle [BACK 78], comme une alternative à la programmation séquentielle (langages de type Von Neumann).Elle n'est pas sensible à "l'histoire du procédé. La seule information importante, à chaque instant, est l'état du procédé

(l'entrée de la fonction de commande). La séquence des événements qui ont mené à cet état n'est pas considérée. La fonction de commande n'a donc pas de mémoire (nous ne considérons pas les éléments de mémoire utilisés pour l'interprétation de la fonction de commande).

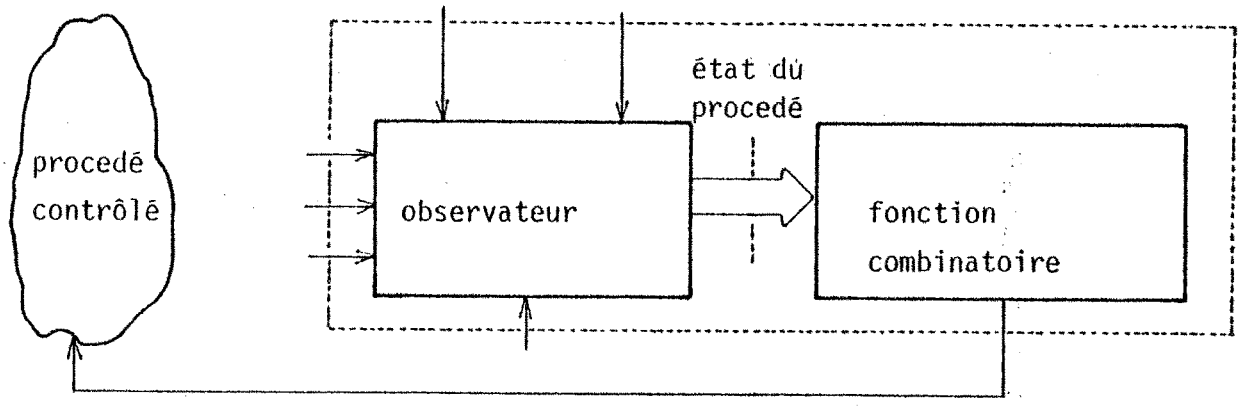


Figure 1.4. Reconstitution de l'état du procédé

Normalement, une ou plusieurs composantes du vecteur d'état du procédé contrôlé ne sont pas directement accessibles à la fonction de commande, à cause de l'ensemble des capteurs disponibles. Ces composantes doivent alors être reconstituées à l'aide d'automates séquentiels. L'ensemble de ces automates constitue ce qu'on appèlera l'observateur. Par extension, nous définirons comme observateur tout dispositif qui restitue à la fonction de commande le vecteur d'état "complet" du procédé (fig.1.4).

Quelques types de variables physiques, comme par exemple la température et la tension électrique, peuvent facilement être mesurées par des capteurs physiques, mais d'autres comme la position, ne sont en général connues qu'à un intervalle de discrétisation près. D'autres, comme par exemple l'épaisseur d'une couche de métal électro déposée (galvanoplastie), peuvent être trop complexes ou trop chères pour être directement mesurées dans certaines applications ; dans ce cas précis, l'épaisseur de la couche peut être calculée à partir de l'indication d'un compteur de charges électriques (épaisseur proportionnelle aux charges déposées par la surface de déposition).

Inversement, on peut dire que tous les automates séquentiels d'un contrôleur sont associés aux composantes du vecteur d'état du procédé contrôlé.

Les raisons de l'existence de ces automates sont de deux ordres :

- économique : l'existence d'automates permet normalement une simplification et/ou une diminution du nombre des capteurs et des lignes de communication procédé - contrôleur, qui deviennent de plus en plus chers par rapport au matériel de traitement informatique proprement dit ;
- technique : l'accès à des automates représentant l'état du procédé est plus facile à réaliser que l'accès au procédé proprement dit, notamment lors de la gestion du partage dynamique de ressources.

Ces considérations nous ont conduits à développer un formalisme et à poursuivre la recherche dans deux directions :

a) l'analyse des outils de description et des systèmes existants [WALT 81],

b) la proposition d'un langage de description [WALT 80].

2.2. Analyse des outils de description

L'analyse des outils de description a été faite dans le but de retrouver au travers de ceux-ci et dans les systèmes existants, les primitives du formalisme proposé et les raisons de leur utilisation. Cette analyse ne se prétend pas exhaustive, elle se concentre plutôt sur les aspects représentatifs des différentes classes de langages et systèmes, comme par exemple :

- le partage dynamique de ressources (chapitre 4) où nous analysons le rôle des sémaphores et des moniteurs dans l'esprit de la notion d'observateur,
- la microprogrammation (chapitre 5) (interprétation d'une instruction par des algorithmes d'instructions de niveau inférieur), applicable aux processeurs informatiques aussi bien qu'au contrôleur de procédés industriels.

2.3. Le langage CSL

Nous proposons ensuite un langage (chapitre 6) dans lequel les éléments du formalisme sont introduits de manière aussi explicite que possible. Les primitives de ce langage, appelé CSL (Control Specification Language), sont adaptées à la description des contrôleurs de systèmes discrets, bien que l'inclusion de fonctions analogiques soit également possible. CSL est un langage de haut niveau, qui encourage la structuration du contrôleur et qui met en évidence les possibilités du parallélisme.

Ce langage se rapproche conceptuellement du langage utilisé dans la programmation des automates programmables par les diagrammes d'interconnexion de relais ("relay ladder diagrams") [SILV 78], qui concernent en fait une importante tranche du marché des contrôleurs. Ce fait peut être expliqué par la simplicité des concepts utilisés ("la commande est une fonction de l'état") qui facilite sa programmation par des techniciens.

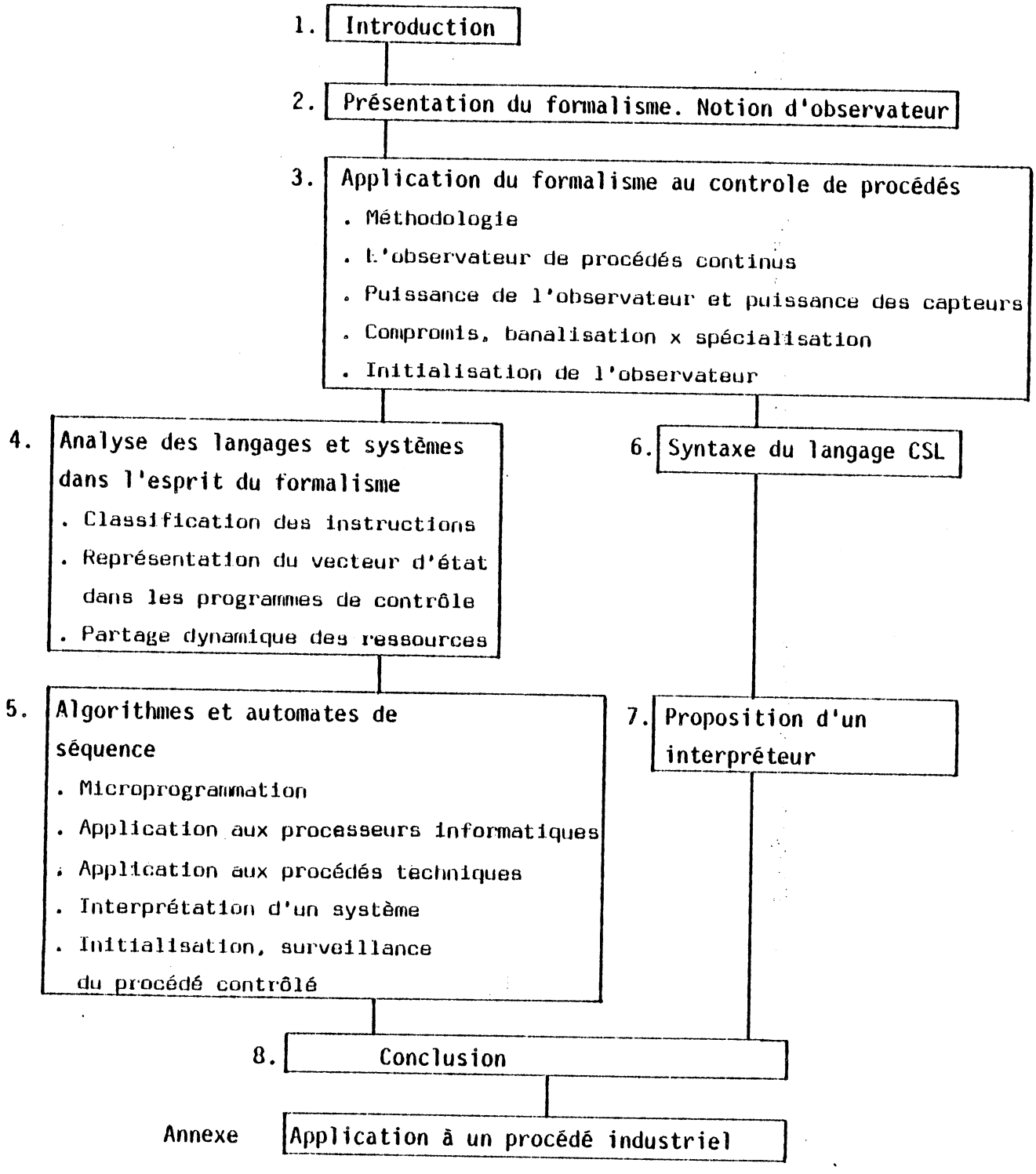
L'objectif principal du langage CSL a été de fournir un outil qui permette l'application de la méthodologie proposée à des exemples concrets de dimensions réalistes, de manière à tester en même temps la validité du formalisme et sa facilité d'emploi.

Cependant, on peut aussi regarder CSL comme un langage de programmation potentiel. Dans ce but, nous présentons au chapitre 7 les caractéristiques principales de deux méthodes d'interprétation de CSL :
(1) par un mécanisme de type "automate programmable",
(2) par un mécanisme dirigé par les données.

2.4. Exemple d'application du langage CSL

La spécification du contrôleur d'une application industrielle est développée en annexe ; ceci a d'ailleurs permis d'améliorer la compréhension du formalisme et la syntaxe du langage.

Plan du travail:



CHAPITRE 2

PRESENTATION DU FORMALISME. NOTION D'OBSERVATEUR

1. PROCEDE CONTROLE
2. CONTROLEUR
   - CONTROLEUR COMBINATOIRE
   - CONTROLEUR SEQUENTIEL: PARTIE COMBINATOIRE ET OBSERVATEUR
   - LE CONTROLEUR SEQUENTIEL ET LE MODELE DE MEALY
3. VARIABLES ET MECANISMES D'OBSERVATION

CHAPITRE 2 - PRESENTATION DU FORMALISME. NOTION D'OBSERVATEUR

1. PROCEDE CONTROLE

Nous admettons que le procédé à commander, ou environnement, peut se représenter par une machine séquentielle de Moore :

PROC : $\langle Q, q_0, E, S, \delta, \gamma \rangle$

où :

$Q = \{q_0, q_1, \ldots\}$ est l'ensemble fini des états internes du procédé,

$q_0 \in Q$ est son état initial,

$E = \{e_0, e_1, \ldots\}$ est l'ensemble fini des vecteurs d'entrée qu'il peut recevoir

$S = \{s_0, s_1, \ldots\}$ est l'ensemble fini des vecteurs de sortie qu'il peut émettre, tel que card $S \leq$ card Q

δ est la fonction qui définit l'évolution de ses états,

$\delta : E \times Q \rightarrow Q$

telle que $q^{t+1} <= \delta(q^t, e^t)$

γ est la fonction qui définit les sorties

$\gamma : Q \rightarrow S$

telle que $s = \gamma(q)$

En général, la fonction γ n'est pas bijective et définit une partition π sur l'ensemble d'états du procédé

$q_i = q_j(\pi) <=> \gamma(q_i) = \gamma(q_j)$

Remarque :

Nous appelons "procédé observable" un procédé contrôlé dans lequel les sorties sont en bijection avec l'état interne Q.

$PROC^{observable}$: $\langle Q, q_0, E, I, \delta, \gamma \text{ bijectif} \rangle$

2. CONTROLEUR

Le contrôleur a pour effet d'amener l'état du procédé contrôlé à une valeur déterminée. Cette action peut se représenter par une fonction :

$$X : I \rightarrow E$$

où I est isomorphe à l'ensemble Q d'états du procédé,

E est l'ensemble d'entrées de commande du procédé.

2.1. Contrôleur combinatoire

Il découle de cette définition, que si l'état du procédé est accessible (procédé observable), le contrôleur peut être réalisé de manière purement combinatoire (fig. 2.1) (contrôleur du procédé observable), puisqu'il consiste en la simple réalisation de X appliqué à l'ensemble I des sorties du procédé (isomorphe à Q).

Inversement, nous admettons que la réalisation combinatoire du contrôleur implique l'isomorphie des sorties du procédé avec ses états internes. Cela signifie que nous avons éliminé de la définition du procédé vis à vis du contrôleur les composants de son vecteur d'état pour lesquelles le contrôleur est indifférent.

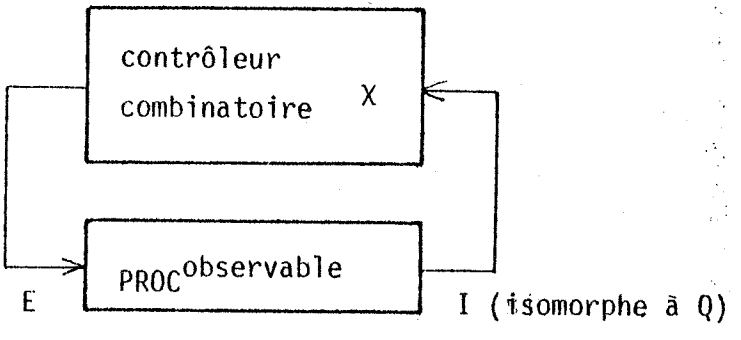


Figure 2.1. Le contrôleur combinatoire

2.2. Contrôleur séquentiel

Lorsque la fonction γ n'est pas bijective, le contrôleur ne peut plus être combinatoire. Le calcul de E en fonction de S doit être fait par une machine séquentielle (figure 2.2.) :

CONTR : $\langle X, x_0, S, E, \omega, \chi \rangle$

où

$X = \{x_0, x_1, \ldots\}$ est l'ensemble fini des états du contrôleur,

$x_0 \in X$ est son état initial,

$S = \{s_0, s_1, \ldots\}$ est l'ensemble de ses entrées (sorties du procédé)

$E = \{e_0, e_1, \ldots\}$ est l'ensemble de ses sorties (entrées du procédé)

ω est la fonction qui définit l'évolution de ses états

$\omega : S \times X \rightarrow X$

telle que $x^{t+1} = \omega(x^t, s^t)$

χ est une fonction qui définit la sortie

$\chi : S \times X \rightarrow E$

telle que $e^t = \chi(x^t, s^t)$

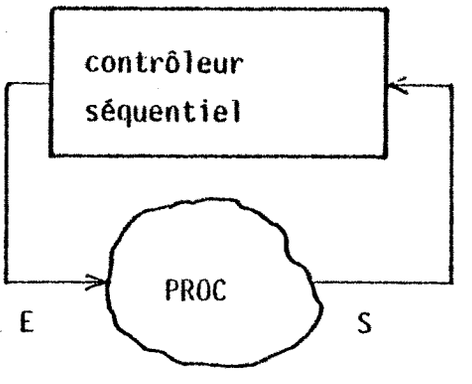


Figure 2.2. Le contrôleur séquentiel

Le contrôleur séquentiel peut être réalisé sous forme de la mise en série de deux machines : la partie combinatoire et l'observateur.

2.2.1. Partie combinatoire

La partie combinatoire (COMBIN) correspond au contrôleur du procédé observable $PROC^{observable}$:

$$COMBIN : \langle I, E, X \rangle$$

où

$I = \{i_0, i_1, \ldots\}$ est la sortie de $PROC^{observable}$ (isomorphe à Q)

$E = \{e_0, e_1, \ldots\}$ sont les commandes du $PROC^{observable}$

$X : I \rightarrow E$ est la fonction de commande

2.2.2. Observateur

Nous appelons "observateur" du procédé PROC une machine séquentielle OBS qui reconstitue un ensemble I, isomorphe à l'état Q du procédé, en fonction de ses sorties S.

$$OBS : \langle X, x_0, S, I, \omega \rangle$$

où

X est l'ensemble fini de ses états internes

$x_0 \in X$ est son état initial

S est l'ensemble de ses vecteurs d'entrée

I est l'ensemble de ses sorties, isomorphe à Q, tel que $I = X \times S$ (fig. 2.3.a

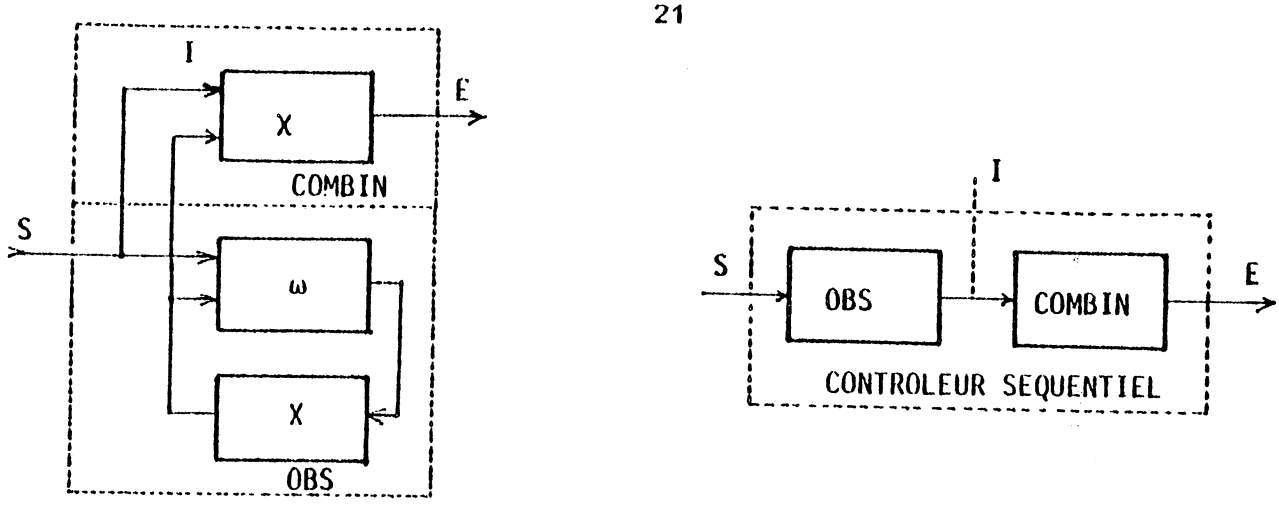
ω est la fonction qui définit l'évolution de ses états

$$\omega : X \times S \rightarrow X$$

L'état interne de l'observateur représente les composantes de Q qui ne sont pas observables(distinguables) par l'ensemble des sorties S du procédé contrôlé.

2.3. Le contrôleur séquentiel et le modèle de Mealy

Le contrôleur séquentiel CONTR = (OBS, COMBIN), peut être vu comme une réalisation du modèle de Mealy (fig. 2.3.a). Une transformation graphique génère le modèle proposé du contrôleur (fig. 2.3.b).



a) Le modèle de Mealy

b) Le modèle proposé

Figure 2.3. Le contrôleur séquentiel et le modèle de Mealy

Remarque :

Les machines PROC et OBS peuvent être vues comme une décomposition série (notation : ⊕) [HART 66] de la machine $PROC^{observable}$ (fig. 2.4).

$PROC \oplus OBS = PROC^{observable}$

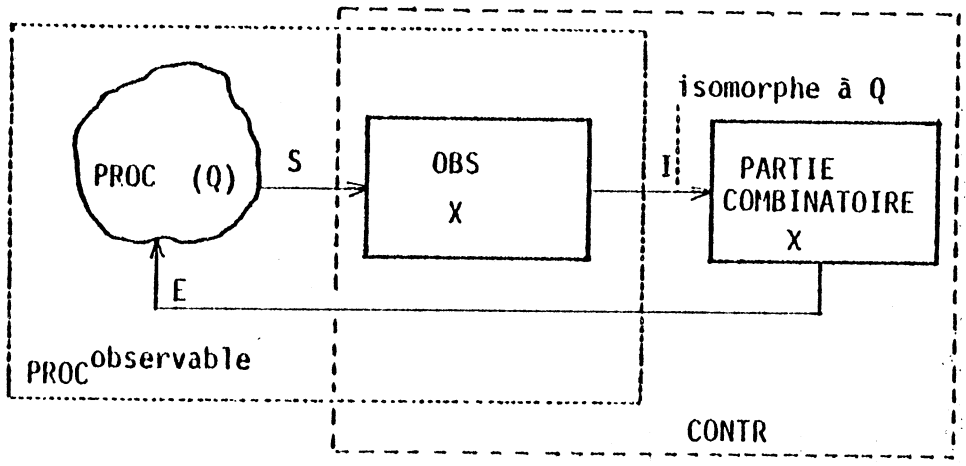


Figure 2.4. Décomposition du procédé observable

3. VARIABLES ET MECANISMES D'OBSERVATION

Les composantes des vecteurs qui constituent l'ensemble de sortie I de l'observateur sont appelées variables visibles I_i. Chaque vecteur de I sera appelé vecteur d'état visible :

$$\text{vecteur d'état visible} = (I_1, I_2, \ldots I_i, \ldots)$$

Chacune des variables I_i a un domaine de définition :

$$\mathfrak{D}(I_i) = \{i_{i1}, i_{i2}, \ldots\}$$

tel que

$$Q \subseteq \mathfrak{D}(I_1) \times \mathfrak{D}(I_2) \times \ldots$$

S'il existe un sous-ensemble de l'ensemble d'entrées de l'observateur, isomorphe à $\mathfrak{D}(I_i)$, alors on dira que la variable I_i est observée par "TRANSMISSION".

Dans le cas contraire, I_i est reconstitué à l'aide d'un automate, et on dira que I_i est observé par "INTEGRATION".

CHAPITRE 3

APPLICATION DU FORMALISME AU CONTROLE DE PROCEDES

1. METHODOLOGIE DE SPECIFICATION
2. L'OBSERVATEUR DE PROCEDES CONTINUS
3. DISCRETISATION DE VARIABLES CONTINUES
4. LA NOTION DE FIDELITE
5. PUPITRES DE COMMANDE
6. PUISSANCE DE L'OBSERVATEUR ET PUISSANCE DES CAPTEURS
7. FILES, PILES ET AUTRES STRUCTURES DE DONNEES
8. COMPROMIS BANALISATION X SPECIALISATION DU CONTROLEUR
9. INITIALISATION DE L'OBSERVATEUR

CHAPITRE 3 - APPLICATION DU FORMALISME AU CONTROLE DE PROCEDES

Ce chapitre propose une méthodologie pour la spécification du contrôleur de procédés temps réel, à partir du formalisme présenté au chapitre précédent.

La méthodologie est appliquée à des exemples "didactiques" (dans la suite de ce travail, on présentera la syntaxe d'un langage basé sur la méthodologie et un exemple "réel").

Le contrôleur séquentiel, caractérisé par un ensemble fini d'entrées, d'états et de sorties, peut être utilisé pour commander des procédés continus dont l'ensemble infini des états est "quantifié" (discrétisé). La réponse de l'état discrétisé du procédé continu aux commandes du contrôleur n'est alors pas nécessairement immédiate et doit être signalée explicitement par des capteurs placés sur le procédé, ou par des "simulateurs" (définis dans le texte).

Le formalisme et la méthodologie qui en découle permettent de mettre en évidence une gamme de choix dans la conception du contrôleur d'un procédé. Ce choix porte sur la puissance des capteurs (définie dans le texte) et sur la connaissance que possède le concepteur du comportement du procédé, incorporé à l'observateur. Ce choix met aussi en jeu l'adaptabilité du contrôleur, c'est-à-dire sa faculté d'adaptation à des évolutions des spécifications. Nous verrons l'influence de cette connaissance sur l'adaptabilité dans les différents niveaux d'une hiérarchie de commande.

Enfin, on analysera l'initilisation de l'observateur, c'est-à-dire la définition de son état lorsque, pour une raison quelconque, il a cessé de représenter l'état du procédé contrôlé.

1. METHODOLOGIE DE SPECIFICATION

Le formalisme présenté dans le chapitre précédent conduit à une méthodologie pour la spécification des contrôleurs de procédés temps réel discrets. Cette méthodologie se décompose en deux phases :

a) Spécification de la partie combinatoire:

spécification de la fonction de commande sous la forme d'un ensemble de fonctions combinatoires dont,

(a) les opérandes sont les composantes du vecteur d'état de l'environnement (procédé contrôlé et consignes du niveau de commande supérieur), et

(b) les résultats sont les commandes vers l'environnement

$$\boxed{\text{commandes} = \chi\,(\text{état de l'environnement})}$$

Le choix et la décomposition du vecteur d'état du procédé constituent une attribution du concepteur. Ce choix peut affecter fortement la complexité et le coût du contrôleur.

Exemples: dans le contrôle d'un réseau ferroviaire, le vecteur d'état peut être composé de deux manières :

(a) par les variables (position, vitesse) concernant chaque train,

(b) par des variables concernant chaque segment de rails (libre, occupé).

Dans le premier cas, le volume d'informations nécessaire sera proportionnel au nombre de trains ; dans le deuxième cas, il sera proportionnel au nombre de segments du réseau. On remarque donc que le choix sera une fonction du rapport trains/segments et du coût des capteurs nécessaires pour la détection de chaque variable.

Dans un procédé de fabrication, le vecteur d'état peut être composé en fonction du produit fabriqué ou en fonction de la machine. Le choix sera une fonction de la quantité et du coût des capteurs ainsi que de l'observateur nécessaire dans chaque cas.

L'état de l'environnement est fourni par l'observateur dont la spécification constitue la deuxième phase. En fait, pour la spécification de l'observateur, il est important d'avoir défini la composition du vecteur d'état. La fonction de commande proprement dite peut être spécifiée ou modifiée a posteriori.

b) Spécification de l'observateur

L'observateur est une machine séquentielle dont la fonction est de générer un vecteur I isomorphe au vecteur d'état Q de l'environnement, à partir d'informations qui lui sont fournies par des capteurs et des commandes générées par la partie combinatoire.

Lorsque la fonction de sortie γ du procédé contrôlé est bijective, c'est-à-dire lorsque l'ensemble des sorties est isomorphe à l'ensemble de ses états, l'observateur ne contient pas d'automates séquentiels.

On remarque donc que la spécification de l'observateur est fortement dépendante de l'ensemble de capteurs disponibles, et vice-versa.

1.1. Exemple d'application

Un chariot (figure 3.1) doit réaliser le cycle X Y Z X. Le procédé fournit 5 sorties : P = mise en route du cycle ; H, D, B et G = contacts de fin de cours en haut, à droite, en bas et à gauche respectivement, et il reçoit deux entrées correspondant aux deux moteurs (actionneurs) MOTVERT et MOTHORIZ dont les valeurs peuvent être (MONTER, DESCENDRE, ARRET) et (ADROITE, AGAUCHE, ARRET) respectivement. La commande P est prise en compte (acceptée) uniquement lorsque le chariot se trouve au point X.

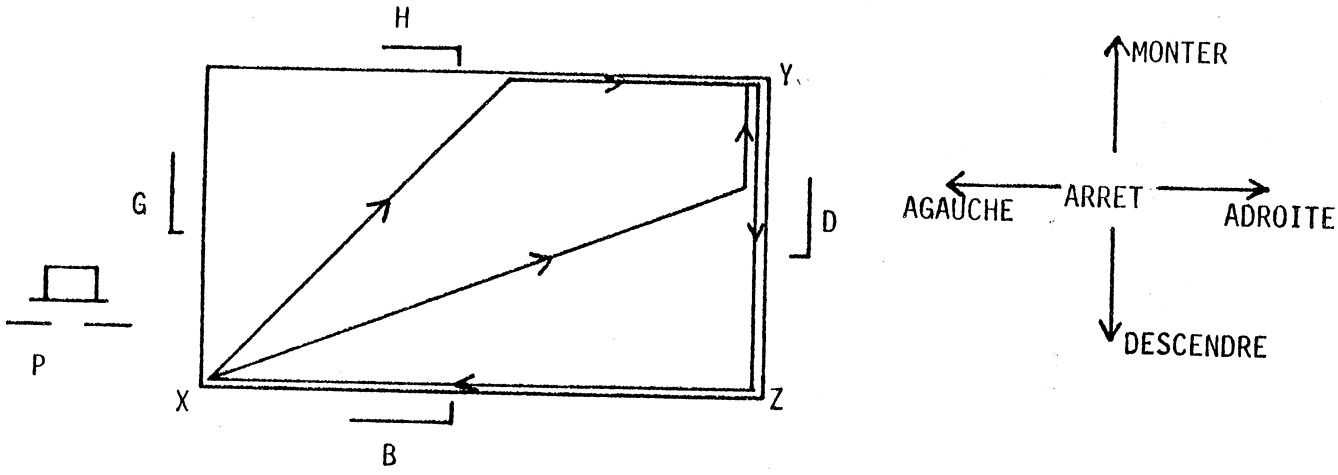


Figure 3.1. Cycle du chariot

Spécification du contrôleur :

comm observateur :

BIT GM (set : G ; reset : D) ;

comm "BIT" correspond à une bascule bistable ; le front montant des signaux connecté à SET et à RESET, met la bascule dans les états "vrai" et "faux" respectivement.

comm si GM vrai, le dernier "mur" heurté est le mur de gauche, sinon c'est le mur de droite.

BIT BM (set : B ; reset : H) ;

comm si BM vrai, le dernier mur heurté est le mur d'en bas, sinon c'est le mur d'en haut.

BIT ALLER (set : P ; reset : GM'.BM') ;

comm GM'.BM' indique que le chariot est arrivé à Y ; non(ALLER) = retour

comm partie combinatoire :

MOTVERT = CAS

SI BM.ALLER ALORS MONTER ;

SI BM'.ALLER' ALORS DESCENDRE ; SINON ARRET ; FINCAS;

MOTHORIZ = CAS

SI GM.ALLER ALORS ADROITE ;

SI GM'.ALLER' ALORS AGAUCHE ; SINON ARRET ; FINCAS;

comm (1) ni l'initialisation du procédé, ni celle de l'observateur n'ont été considérées ;

(2) le contrôleur n'a pas été structuré.

2. L'OBSERVATEUR DE PROCEDES CONTINUS

Le vecteur d'état des procédés contrôlés réels est constitué de composantes continues, c'est-à-dire que les composantes de leur vecteur d'état ${}_pQ$ sont isomorphes à des segments de la droite réelle.

La discrétisation de ${}_pQ$ peut être représentée à l'aide d'une relation ${}^{discr}R$ telle que :

. l'ensemble quotient $Q = {}_pQ / {}^{discr}R$ soit fini, et que

. chaque classe de l'ensemble Q contienne un ensemble de valeurs contigües de ${}_pQ$.

La relation ${}^{discr}R$ représente l'action des capteurs et/ou des dispositifs de discrétisation prévus pour cet usage (§ 3 ci-après).

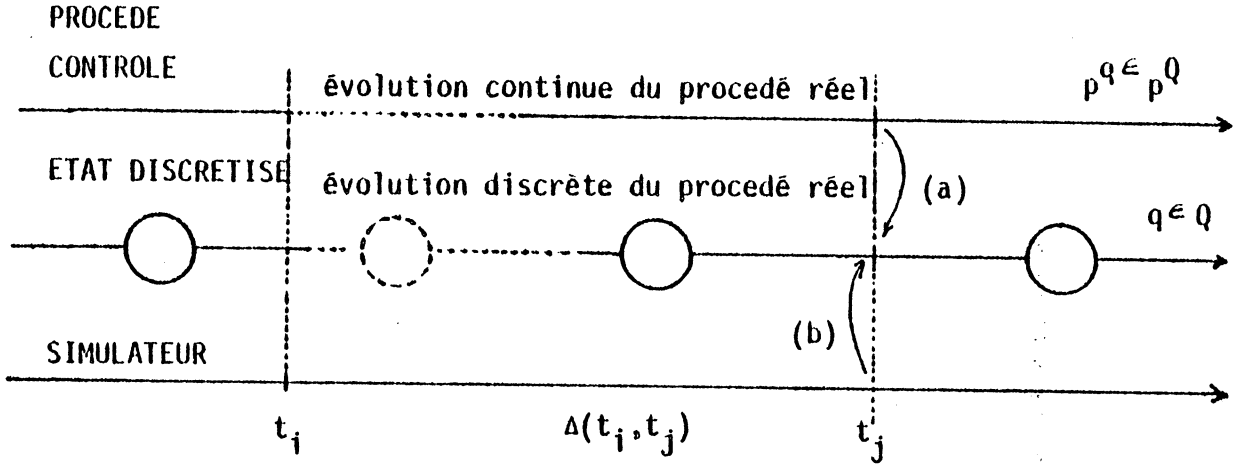


Figure 3.2. Activation de l'observateur

Comme conséquence de la discrétisation, l'état $q \in Q$ discrétisé du procédé et la sortie n'évoluent pas simultanément aux entrées $e \in E$ (les commandes). Il existe un retard, prévisible ou non, associé à l'évolution continue de l'état ${}_p^{q \in} {}_pQ$ du procédé entre des valeurs contigües appartenant à une même classe de Q.
Les instants auxquels cet état quantifié évolue constituent une suite dénombrable d'instants :

$$t_0, t_1, t_2, \ldots$$

On dira de ce fait que le temps du procédé discrétisé évolue de manière discrète.

Les instants représentatifs de l'évolution du procédé doivent se reproduire dans le contrôleur, où ils font évoluer l'état des automates de l'observateur. Les entrées de ce dernier peuvent provenir (fig. 3.2) :
a) des sorties du procédé (capteurs),
b) du simulateur.

Nous appelons simulateur un dispositif ou un ensemble de dispositifs qui simulent les états ou les instants de transition entre les états du procédé contrôlé, lesquels sont inaccessibles par des capteurs placés sur celui-ci (soit pour des raisons techniques, soit pour des raisons économiques).

La conception du simulateur est possible grâce à la connaissance a priori de la progression continue des états du procédé. Pour leur réalisation, on utilise des éléments tels que par exemple des temporiseurs.

Composition de l'observateur étendu

Nous appelons "observateur étendu" la réunion de :
. l'observateur OBS (tel que défini au § 2.2.2.2.);
. le simulateur.

Les éléments de l'observateur OBS changent d'état à l'instant d'arrivée d'un stimulus (événement). Exemple : compteur activé par un événement du procédé. Une fois que le stimulus cesse, l'état de l'observateur se stabilise.

Le simulateur a une évolution indépendante après l'arrivée d'un stimulus. Exemple : temporiseur, retard. Les éléments du simulateur jouent, du point de vue du contrôleur séquentiel (partie combinatoire et observateur proprement dit), le même rôle que le procédé contrôlé : ils sont commandés et peuvent générer des événements. Physiquement, pourtant, le simulateur appartient au contrôleur.

Dans la suite de ce travail, on désignera "l'observateur étendu" simplement comme "observateur", lorsqu'il n'y aura pas possibilité d'ambiguité.

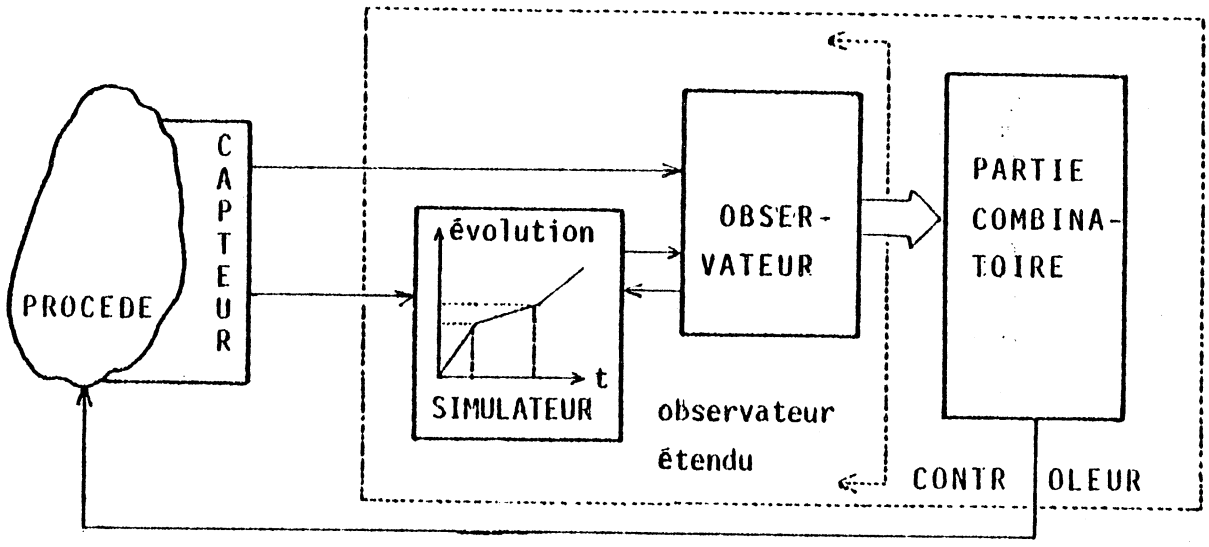


Figure 3.3. L'observateur étendu

3. DISCRETISATION DE VARIABLES CONTINUES

On peut distinguer deux mécanismes de discrétisation : la conversion et la discrétisation directe.

a) La conversion est une fonction de discrétisation réalisée par un dispositif physique (le convertisseur analogique - numérique) à partir d'une mesure continue fournie par un capteur (figure 3.4). Ce capteur n' a donc aucun rôle dans le découpage du domaine de la variable continue en valeurs discrètes.

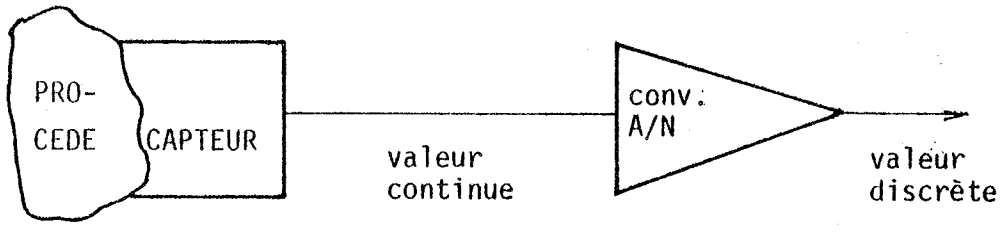


Figure 3.4. Discrétisation par convertisseur A/N

b) Discrétisation directe (figure 3.5) : le découpage du domaine est réalisé par un ensemble de capteurs qui détectent directement soit l'état discrétisé, soit l'occurence d'un état intermédiaire dit "de transition" (correspondant au franchissement d'un seuil). L'observateur interprète les sorties de ces capteurs à l'aide de fonctions combinatoires et d'automates, et reconstitue ainsi la variable d'état du procédé.

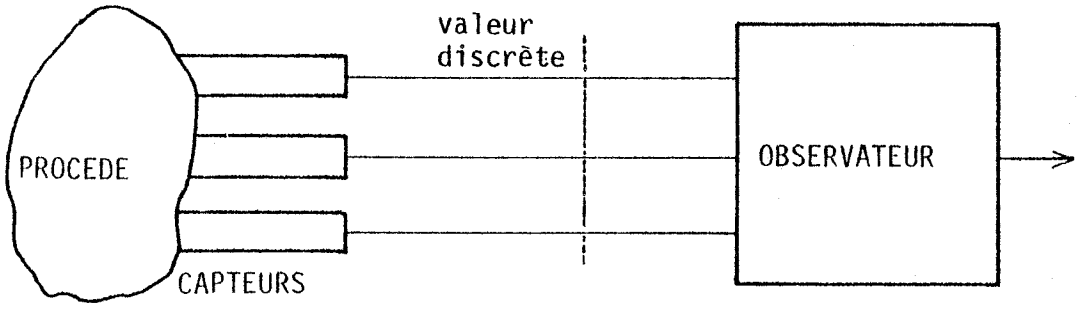


Figure 3.5. Discrétisation par des capteurs

Exemples :

. Fréquencemètre mécanique par résonnance : un ensemble de lames accordées à des intervalles discrets d'une gamme de fréquences, est utilisé pour mesurer une fréquence. La lame i dont l'oscillation a la plus forte amplitude est celle dont la fréquence de résonnance s'approche le plus de la fréquence mesurée (ou de son multiple). Chaque lame est alors un capteur "réel" et l'observateur réalise la fonction combinatoire :
fréquence = fréq résonn. (MAX (amplitude de la lame i)).

. Mesure chimique de température, pH, etc ...

4. NOTION DE FIDELITE

Soit $SUC(q_i)$ l'ensemble des états successeurs immédiats d'un état $q_i \in Q$ du procédé contrôlé. La transition de l'état q_i à l'état $q_j \in SUC(q_i)$ est signalée à un automate de l'observateur par un événement issu du procédé contrôlé (ou d'un simulateur).

Lorsque $Card(SUC(q_i)) = 1$, la réception de l'événement (et la connaissance de l'état précédent) est suffisante pour déterminer le nouvel état.
Par contre, si $Card(SUC(q_i)) > 1$, il y a deux possibilités :

. soit l'événement identifie le prochain état, c'est-à-dire que les capteurs placés sur le procédé émettent un signal différent pour chaque état successeur,

. soit l'événement est commun à plusieurs états successeurs ; il faut alors une information supplémentaire pour l'identifier. Cette information peut être fournie par les commandes (les entrées $e \in E$ du procédé, figure 3.6).

Si l'on assimile l'événement au *module* de la dérivée de l'état (il signale sa modification), alors la *direction* de la dérivée (dans le cas où il y en a plusieurs) peut être calculée à l'aide des commandes. On remarque une notion de "fidélité" implicite : pour pouvoir utiliser la commande du procédé pour l'activation de l'observateur, il faut présumer que le procédé ait effectivement "obéi", c'est-à-dire qu'aucune interférence de l'environnement n'ait affecté la direction de l'évolution imposée par le contrôleur.

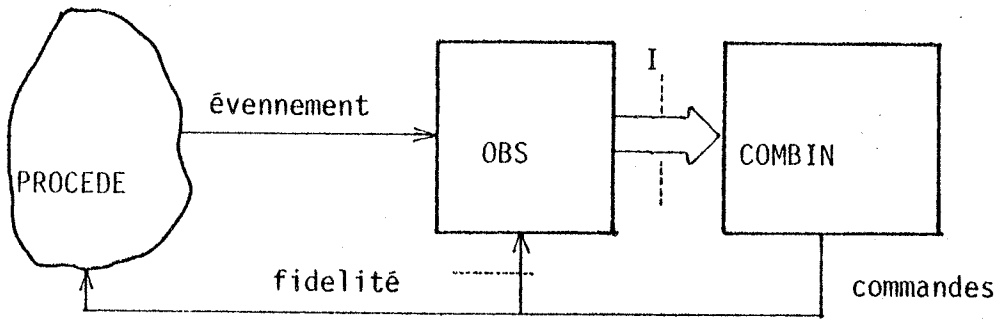


Figure 3.6. Fidélité dans l'activation de l'observateur

Exemple : détermination de la position angulaire d'une roue de bicyclette (fig. 3.7). L'observateur de la position est constitué par un compteur actionné par l'événement "interruption d'un faisceau lumineux par un rayon de la roue". La décision sur le sens de la transition (incrémenter ou décrémenter) peut alors être prise à l'aide de la commande du moteur (sens horaire ou anti-horaire respectivement). Cette information de la commande pourrait être remplacée par un capteur de direction placé sur la roue.

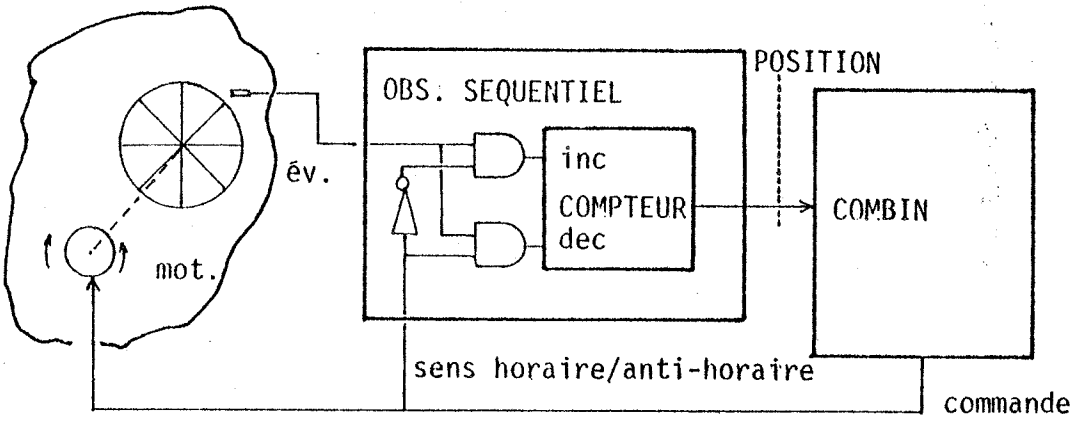


Figure 3.7. Détermination de la position angulaire d'une roue

5. PUPITRES DE COMMANDE

Du point de vue du contrôleur, les pupitres de commande se situent au même niveau que le procédé contrôlé "technique" (fig. 3.8).

En fait, les pupitres incluent deux types de commandes :

- celles qui agissent directement sur le procédé contrôlé "technique" sans interaction directe avec le contrôleur ; ces commandes ne sont pas mentionnées dans la spécification du contrôleur ;
- celles qui agissent sur le procédé par l'intermédiaire du contrôleur ; elles appartiennent à la spécification.

Les capteurs et les activateurs, c'est-à-dire l'interface entre le pupitre de commande et le contrôleur, correspondent respectivement aux dispositifs d'entrée (boutons-poussoirs, claviers, ...) et aux dispositifs de sortie (visus, indicateurs)

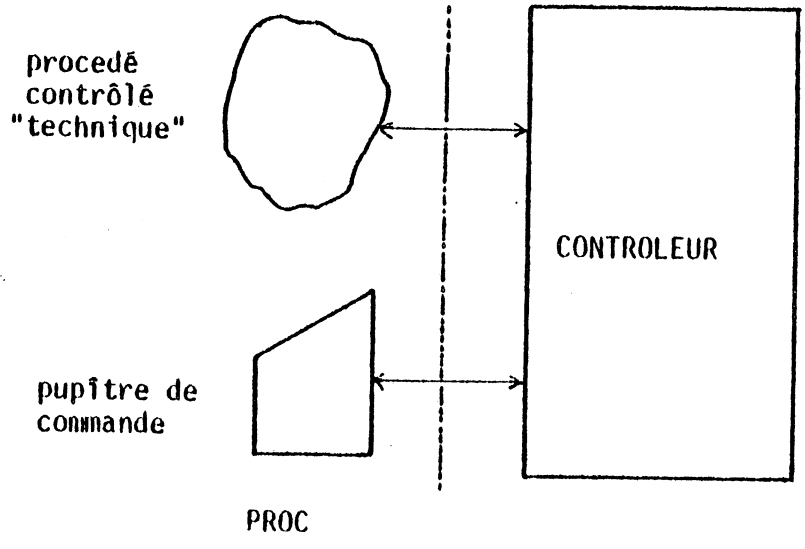


Figure 3.8. Le pupitre de commande

6. PUISSANCE DE L'OBSERVATEUR ET PUISSANCE DES CAPTEURS

La conception du contrôleur d'un procédé présente normalement une gamme de possibilités dont l'un des axes est le compromis entre la puissance des capteurs et la connaissance à propos du comportement du procédé contrôlé incorporée à l'observateur (pour simplifier, on l'appelera dorénavant "puissance de l'observateur").

La "puissance des capteurs" est le quotient entre la cardinalité de l'ensemble S des sorties et l'ensemble Q des états du procédé contrôlé :

$$\text{puissance des capteurs} = \frac{\text{Card (S)}}{\text{Card (Q)}}$$

Puisque Card(S) ≤ Card(Q), on a : 0 ≤ puissance des capteurs ≤ 1
La puissance sera nulle lorsqu'il n'y aura pas de capteur et égale à 1 lorsque la fonction de sortie γ sera bijective.

La solution choisie doit correspondre à une minimisation du coût global (le coût étant pris dans son sens le plus large) du projet, pour une performance déterminée. On essayera donc d'analyser le rôle des différents éléments du modèle proposé dans ce compromis.

D'une façon générale, on peut dire que la puissance nécessaire des capteurs diminue avec la puissance de l'observateur (figure 3.9).

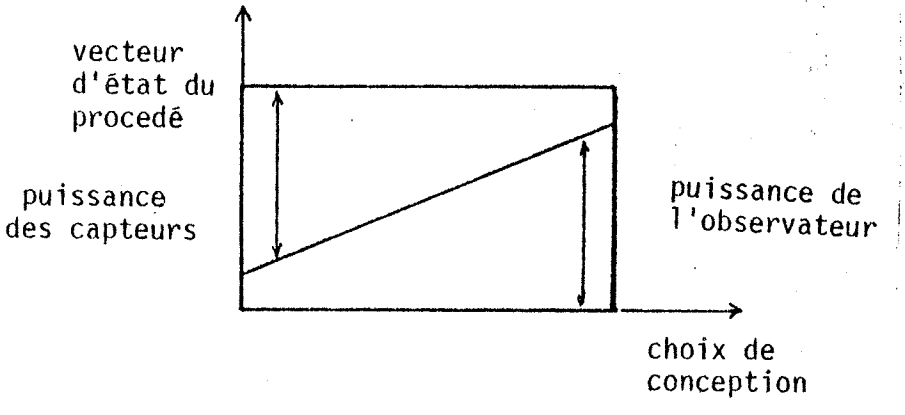


Figure 3.9. Visualisation très schématique du compromis capteurs/observateur

La puissance de l'observateur se manifeste sous deux aspects :
. prédictabilité de l'évolution des variables d'état du procédé contrôlé,
. connaissance de l'évolution temporelle du procédé.
Ces aspects se reflètent respectivement sur la conception de l'observateur OBS du contrôleur séquentiel et du simulateur.

a) Prédictabilité

C'est la connaissance préalable de l'ensemble des états successeurs possibles d'un état donné du procédé contrôlé. Cette connaissance permet une réduction de la quantité d'informations d'entrée nécessaire au contrôleur par rapport à celle qui serait nécessaire à la détection du nouvel état par transmission (cf. § 2.3). Cela se voit dans le fait que le nombre de successeurs d'un état est inférieur au nombre d'états total. Dans le cas où un état a un seul successeur, une variable booléenne suffit pour commander la progression de l'observateur.

Exemple : usinage d'une pièce mécanique (figure 3.10) : la fabrication d'une pièce mécanique comprend une séquence d'opérations de tournage, de perçage, de fraisage, etc .. réalisables automatiquement sur un poste de travail. A chaque opération, il est nécessaire de connaître le résultat de l'opération précédente, c'est-à-dire l'état de départ de la pièce vis à vis de l'opération suivante. Il y a de nouveau deux possibilités :
. soit on mesure l'état (les dimensions) de la pièce à chaque début d'opération;
. soit ces mesures sont connues a priori ; on peut alors établir un fichier dont chaque élément contient l'ensemble des dimensions de départ d'une opération ; ces éléments sont adressés par un pointeur dont le contenu constitue l'état codé de la pièce.

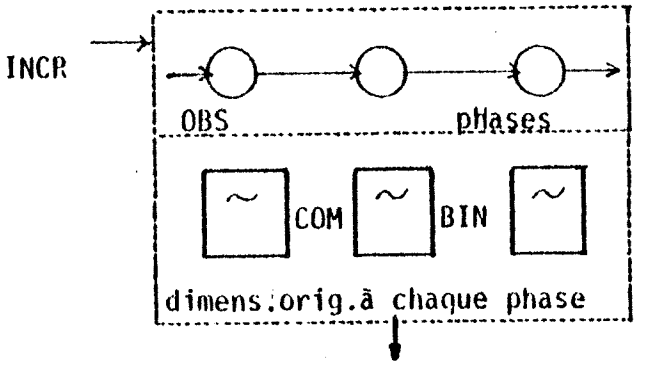


Figure 3.10. Application de la prédictabilité

b) Connaissance de l'évolution temporelle d'un procédé

permet de construire un simulateur qui évolue et qui signale le passage vers l'état successeur sans aucun besoin de capteurs. Cette possibilité est normalement utilisée dans des zones d'opération limitées du procédé. Le non-déterminisme (c'est-à-dire la non-connaissance) des événements du procédé en est le facteur limitant.

Exemple : sur une chaîne d'assemblage, on souhaite laisser un espace entre deux pièces qui se suivent, issues d'un magasin. Cet espace dépend de la nature des pièces. Une cellule photoélectrique détecte la face postérieure de la pièce. La vitesse de la chaîne est connue. On désire déposer une nouvelle pièce lorsque la précédente aura atteint une position qui correspond à l'espace convenu.
Le simulateur, un temporiseur appelé POSITIONNONATTEINTE, est déclenché par la transition obscur → clair de la cellule photoélectrique, et reste actionné pendant TEMPS secondes. En langage de spécification CSL (chapitre 6), on écrira :
TMR POSITIONNONATTEINTE (S : CELLULEPHOTO, T : TEMPS) ;

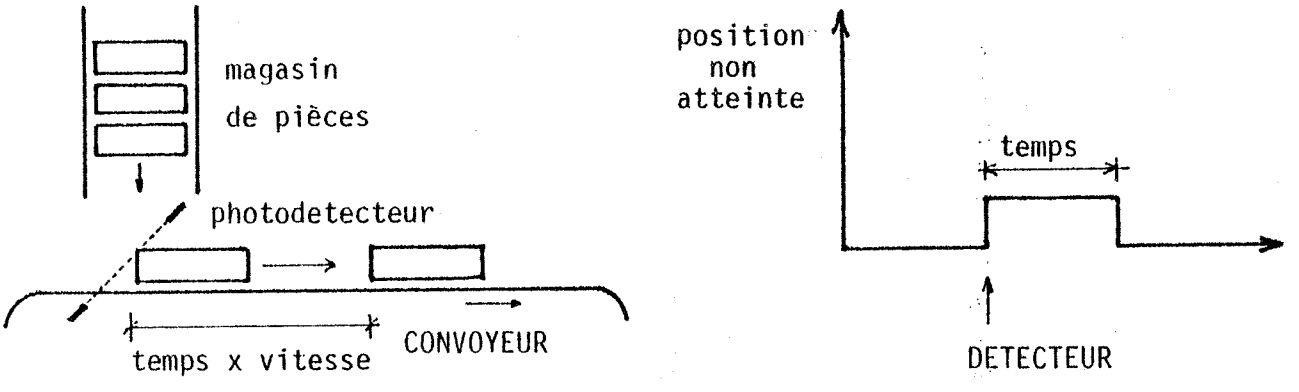


Figure 3.11. Simulation de position

On remarque qu'une variable peut, en principe, être obtenue par deux méthodes extrêmes :

a) exclusivement à partir des capteurs physiques, donc sans aucun dispositif supplémentaire,

b) en simulant la totalité de l'environnement, donc sans aucun capteur physique.

b.1. Observateur de mesure par intégration

L'observateur mesure le temps nécessaire pour qu'un phénomène (fonction connue de la variable mesurée) génère un certain événement. Cela peut se réaliser avec un compteur dont la fréquence est connue, mis à zéro au début du phénomène et arrêté par l'événement.

Exemples : 1) *mesure de distance* par temps de propagation d'une oscillation dont la vitesse de translation est connue. Ce principe est utilisé par RADAR, SONAR, mise au point en photographie, mesure de couche de métallisation. Phénomène : translation d'une onde. Evénement : arrivée de l'écho.

2) *convertisseur analogique numérique* (fig. 3.12):phénomène : charge d'un capacitéur. Evénement : sortie du comparateur de tension.

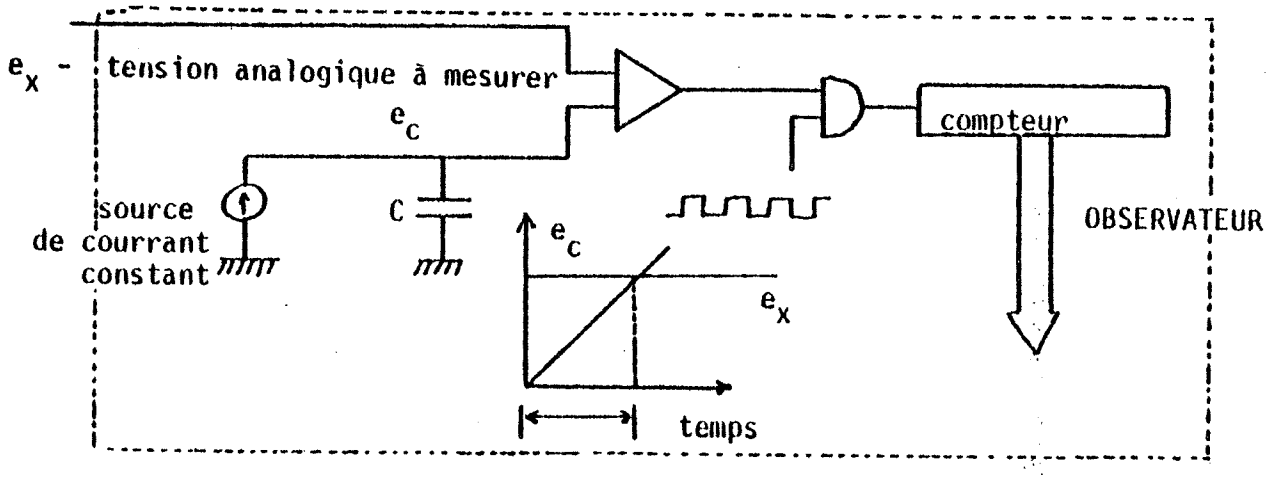


Figure 3.12. Convertisseur analogique - numérique.

7. FILES, PILES ET AUTRES STRUCTURES DE DONNEES

Les files, les piles, etc .. sont des structures de données de l'observateur qui simulent des structures correspondantes d'éléments du procédé contrôlé, selon des critères explicites ou implicites.

Exemples : 1) Les files d'attente de la ressource processeur dans un système informatique. L'ensemble des utilisateurs peut être ordonné par le temps d'attente. Puisque le temps n'est pas une variable d'état, il représente d'autres variables implicites, telles que "l'urgence de traitement", "l'impatience" de l'utilisateur du terminal, etc ..

2) Le fichier d'attente d'utilisation d'une unité de disque à bras mobile est ordonné en fonction des cylindres concernés par chaque requête, de façon à minimiser les mouvements du bras.

3) Des voitures en cours de montage sur une chaîne d'assemblage ; leur manipulation sur chaque poste de travail automatique dépend d'un ensemble de caractéristiques fournies ou mesurées à l'entrée de la chaîne. Entre le point d'entrée des voitures et le premier poste de travail, ainsi qu'entre les postes de travail consécutifs, il y a des files de voitures. Ces files sont reproduites par des structures informatiques analogues, contenant les caractéristiques des voitures concernées, dans l'observateur du contrôleur de chaque poste de travail.

8. COMPROMIS BANALISATION x SPECIALISATION DU CONTROLEUR

En principe, la connaissance a priori du procédé permet de construire un contrôleur plus économique, c'est-à-dire mieux adapté aux besoins de l'application spécifique, mais moins adaptable à des changements.

Il faut pourtant faire quelques distinctions dans les critères coût/adaptabilité pour les différentes parties d'un contrôleur. Le contrôleur peut être décomposé en deux types de composants :

. le contrôleur proprement dit (traitement de l'information),

. les capteurs et les lignes de communication procédé/contrôleur.

a) Le contrôleur proprement dit

Il peut être programmé, câblé, ou hybride. Ces notions ne sont pas très précises, face aux multiples options existantes ; néanmoins, prenons-les selon leur sens usuel. Un contrôleur "programmé" utilise un "processeur". Un des aspects les plus importants est le fait de pouvoir réaliser des types différents d'opérations à l'aide du même matériel (processeur), mais une seule opération à la fois.
Un contrôleur "câblé" est un dispositif dont le support matériel est une image plus ou moins directe de la fonction spécifiée. Il est donc plus rapide et moins flexible que le contrôleur programmé.

A priori le coût du contrôleur câblé devrait être plus ou moins équivalent à celui du contrôleur programmé, pour une puissance de calcul équivalente. Cela se vérifierait si pour ces deux contrôleurs la conception s'effectuait à l'unité et si les composants de base étaient les mêmes. Or, pour être adaptables, les contrôleurs programmables (latu sensu) sont fabriqués en grandes quantités, ce qui diminue leur prix en raison de l'amortissement des coûts de conception du matériel. De plus, ces grandes quantités rentabilisent l'intégration (LSI), ce qui en diminue encore le prix grâce à l'utilisation de composants de base plus économiques.

Il faut encore remarquer que la mise au point d'un programme est, sauf pour les applications très simples, moins coûteuse que la modification d'un circuit câblé de complexité équivalente.

En somme, les ordinateurs sont plus avantageux que les circuits câblés, sauf si :
. l'application exige une grande vitesse de traitement, et/ou si
. l'application est très simple (équivalente à quelques centaines de portes).

Or, si le contrôleur est programmé, le coût d'un programme "adaptable" n'est pas beaucoup plus élevé que celui d'un programme "particulier" (peut-être un peu plus de place mémoire), ce qui est probablement compensé par le fait qu'un programme adaptable permet une modularité plus poussée, donc facilité, fiabilité et économie de programmation.

b) Les capteurs et les lignes de communication procédé - contrôleur

Des contraintes physiques n'ont pas permis, du moins jusqu'à présent, une banalisation des capteurs comparable à celle des dispositifs de traitement. Les plus simples photodétecteurs ou détecteurs de fin de course sont aujourd'hui aussi chers qu'un microprocesseur. Un câble coaxial de quelques mètres de longueur, avec des connecteurs et le travail de pose, ont également un coût de cet ordre. Or, dans une installation industrielle, on peut compter des centaines, voire des milliers, de capteurs. Il est vrai que, dans un cas pareil, le contrôleur n'est généralement pas un petit microprocesseur, mais de toute façon le rapport des coûts (capteurs, lignes) / unité de traitement a augmenté de manière significative dans les dernières années.

L'économie des capteurs et des lignes est donc encore et de plus en plus souhaitable. C'est là qu'intervient plus fortement l'intérêt de "particulariser" le projet, ce qui demande les connaissances les plus complètes possibles ("connaissance" du procédé contrôlé, connaissance du domaine d'évolution des variables, prédictabilité de la séquence, connaissance de l'évolution temporelle). Cette connaissance peut être définie par niveaux de la hiérarchie de commande. Si la connaissance est forte par rapport aux niveaux les plus proches du procédé contrôlé, on peut optimiser le projet localement (avec économie locale de matériel et perte d'adaptabilité), sans affecter les niveaux supérieurs. Si la connaissance est plus globale, il est alors possible de pousser plus loin l'optimisation, avec plus d'économie de matériel et de plus grandes pertes d'adaptabilité. Il s'agit donc de choisir le niveau auquel on connaît le comportement du procédé, ce qui permet l'optimisation jusqu'à ce niveau, sans limiter l'adaptabilité des niveaux supérieurs.

9. INITIALISATION DE L'OBSERVATEUR

La réinitialisation de l'observateur est nécessaire lorsque pour une raison quelconque son état a cessé de représenter l'état du procédé contrôlé. On examine ici les modes d'initialisation en fonction des mécanismes d'observation cités dans le § 2.3.

Pour le mécanisme de transmission, le problème ne se pose pas, puisqu'il n'y a rien à effacer et donc rien à initialiser. Le problème se pose lorsque l'observateur est constitué par des éléments de mémoire volatiles dont l'état peut s'effacer pendant par exemple une panne d'alimentation ou avant le démarrage du procédé. Dans quelques modèles de contrôleurs, l'état des bascules qui constituent l'observateur proprement dit, peut être rapidement sauvegardé en cas de coupure d'alimentation. L'état de l'observateur n'est donc à vrai dire pas effacé ; il change simplement de support physique. Si l'état n'est pas sauvegardé, il faut le reconstituer. Il y a alors deux possibilités :

a) ou bien le procédé se met dans un état connu à chaque initialisation : l'observateur peut alors être mis dans ce même état par un signal de remise à zéro ("power on preset"). On peut citer comme exemple tous les cas où l'état d'une variable du procédé est une fonction de l'alimentation électrique, tel que les électrovannes (lorsque le courant est coupé, elles se mettent automatiquement dans une position de repos) ;

b) ou bien l'état du procédé est inconnu. Le mécanisme d'intégration est alors inutile, puisque le nouvel état est une fonction du précédent et celui-ci est par hypothèse inconnu. L'état de l'observateur est alors reconstitué grâce à un capteur qui le positionne avec une référence absolue. Cela peut impliquer une séquence d'initialisation du procédé (§ 5.5), principalement si un seul état du procédé a un tel capteur "absolu" et les suivantes sont mises à jour par "intégration".

Exemples : 1) Détermination de la position du bras d'une unité de disque. Lors d'une coupure d'alimentation, il faut prévoir le retour du bras au cylindre zéro pour réinitialiser le compteur.

2) Comptage des caractères d'un bloc dans une transmission de données. La mise à zéro s'effecute à l'entête du bloc et l'incrémentation à chaque caractère. En cas de coupure d'alimentation, il faut relire le bloc, car l'information de comptage de caractères est perdue (en plus du checksum et de quelques caractères).

3) Machines outil automatiques. En cas de coupure d'alimentation, les outils doivent revenir à une position d'origine pour réinitialiser l'observateur de coordonnées avant la suite de l'usinage.

CHAPITRE 4

ANALYSE DES LANGAGES ET SYSTEMES DANS L'ESPRIT DU FORMALISME

1. CLASSIFICATION DES INSTRUCTIONS ET DES MODULES
2. REPRESENTATION DE L'OBSERVATEUR DANS LES PROGRAMMES DE CONTROLE PROCEDURAUX
3. LE PARTAGE DYNAMIQUE DES RESSOURCES

CHAPITRE 4 - ANALYSE DES LANGAGES ET SYSTEMES DANS L'ESPRIT DU FORMALISME

On examinera dans ce chapitre la manière dont s'expriment les éléments du modèle, proposé dans les chapitres précédents, dans les systèmes et langages de programmation temps réel. En fait, les éléments du modèle sont déjà présents dans toutes les formes de description, de façon plus ou moins implicite. On essayera ici de les rendre apparents.

Dans un premier temps, on analysera les types d'instructions - continues et transitoires - et l'interface de communication entre les modules formés par ces types d'instructions. La représentation du vecteur d'état du procédé contrôlé dans les langages est étudiée.

On regarde ensuite le mécanisme de partage dynamique des ressources, plus particulièrement l'interprétation des sémaphores, du point de vue du modèle et des différents types de langages. On remarque ainsi les limitations des langages purement combinatoires (par exemple les langages booléens des automates programmables) et la façon dont ces limitations peuvent être couvertes.

1. CLASSIFICATION DES INSTRUCTIONS ET DES MODULES

1.1. Instructions transitoires et continues

Les instructions d'un langage de programmation peuvent être de deux types, selon leur mode d'activation : transitoires ou continues.

a) Les instructions transitoires sont activées par un événement externe ; elles ont un effet instantané et affectent l'état d'un automate d'états finis. L'affectation y est représentée par le symbole ":=".
Exemples :

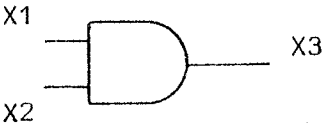
- soit l'instruction X3 := X1 et X2. La suite des actions est exécutée lors de son activation :
  "réaliser l'opération X1 et X2, puis ranger le résultat dans l'emplacement réservé à la variable X3" ;
- "ALLERA étiquette" signifie qu'il faut modifier l'état de l'automate "compteur ordinal".

b) les instructions continues représentent des définitions (notées par le symbole "=") qui se vérifient toujours, sauf pendant les transitions (lesquelles sont "presque" instantanées).

Exemple :

. Soit l'instruction X3 = X1 et X2

Interprétation :



1.2. Modules procéduraux et non-procéduraux

Un module écrit dans un langage déterminé peut être procédural ou non-procédural [BARB 75]. Nous utilisons ici le mot "module" latu sensu, pour désigner un ensemble d'instructions sémantiquement et lexicalement adjacentes.

Un module est dit procédural si l'ordre lexicographique des instructions correspond à leur ordre d'exécution, à moins qu'une instruction de contrôle de flot explicite (ALLERA, APPEL procédure, RETOUR procédure) ne provoque un saut. Ces instructions, tout comme la fin de l'exécution des autres, modifient l'état d'un automate "compteur ordinal" qui ordonne l'exécution séquentielle des instructions du module.

Un module est dit non-procédural si l'ordre lexicographique ne joue aucun rôle dans l'exécution des instructions.

Les modules constitués par des instructions transitoires peuvent être (1) procéduraux ou (2) non procéduraux, tandis qu'un module constitué par des instructions continues est toujours (3) non procédural (figure 4.1.).

L'exécution d'instructions transitoires dans des modules non procéduraux (de type 2) est déclenchée par la vérification d'une condition booléenne, telle qu'une étiquette, ou une condition (SI ... ALORS ....).

Exemples de langages qui utilisent des modules de type 2 :

- langages de transfert de registres [BARB 75], utilisés pour la description et la simulation de systèmes digitaux basés sur les registres,
- le langage CONDOR [TAKA 80], proposé pour la description et la réalisation de contrôleurs de systèmes discrétisés,
- langages dérivés des "commandes gardées" [DIJK 75] : l'activation de chaque "commande gardée" est non procédurale, bien que les instructions internes à la commande puissent constituer un module procédural. Ces langages, tels que CSP [HOAR 78], Distributed Processes DP [BRHA 78] et PARCS [GRIE 76], utilisent la non procéduralité comme un moyen de faciliter la description du non déterminisme des relations entre processus communicants,
- l'activation des automates dans CSL.

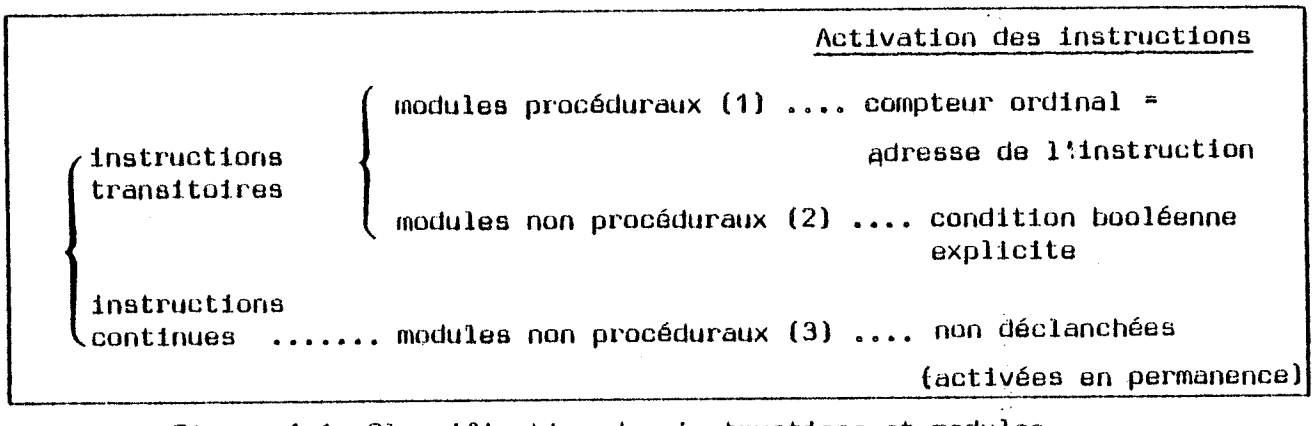


Figure 4.1. Classification des instructions et modules

Si les instructions sont continues, alors l'interprétation n'est pas déclenchée (les ressources utilisées pour l'interprétation de chaque instruction sont virtuellement disponibles en permanence).

Exemples :

- les langages de programmation des automates industriels basés sur les "relay ladder diagrams" (diagrammes de relais). Ces langages sont inspirés des schémas utilisés par des techniciens pour représenter les circuits "câblés" d'interconnexion de relais et sont encore aujourd'hui très répandus dans l'industrie pour des applications de contrôle relativement peu complexes [SILV 78] ;
- les expressions de contrôle continu (analogique) ;
- les expressions combinatoires du langage CSL, proposé dans ce travail.

Un module continu peut être transformé en module transitoire si l'on associe un événement au calcul de chaque variable et si l'on conditionne l'exécution de chaque instruction à la présence de ses données.

La classification ci-dessus concerne des modules plutôt que des langages. En fait, un langage est un ensemble compatible de règles de syntaxe qui peut admettre l'inclusion de plusieurs mécanismes décrits. Pourtant dans la littérature, on parle souvent de types de "langages" par référence à leurs aspects les plus caractéristiques.

1.3. Communication entre modules

Les instructions transitoires sont activées (interprétées, exécutées) par l'arrivée de "messages" qui sont des événements qui peuvent être accompagnés de paramètres et qui sont pris en compte discrètement dans le temps. L'événement correspond à la vérification d'une condition booléenne (par exemple, l'adresse de l'instruction, dans un module procédural, devient égale au compteur ordinal). Les paramètres sont des variables quelconques. L'activation d'une instruction transitoire a comme résultats :

- le chargement ou la modification de la valeur des variables mémorisées,
- la modification de ces variables peut provoquer l'émission de nouveaux messages qui à leur tour activent d'autres instructions transitoires.

La communication entre modules composés par des instructions transitoires ne pose pas de problèmes, puisque des deux côtés les instructions sont activées par des messages.

La communication entre modules composés d'instructions continues (de type 3) ne pose pas de problèmes non plus, puisque dans chacun de ces modules les instructions (définitions) ont des "états" comme opérandes et comme résultats.

L'interface entre les modules continus (3) et les autres nécessite des précautions supplémentaires, puisque du côté continu l'activation est permanente et du côté transitoire, elle s'effectue par des messages. Les éléments qui constituent cet interface font normalement partie du matériel d'interprétation des modules transitoires (figure 4.2.).

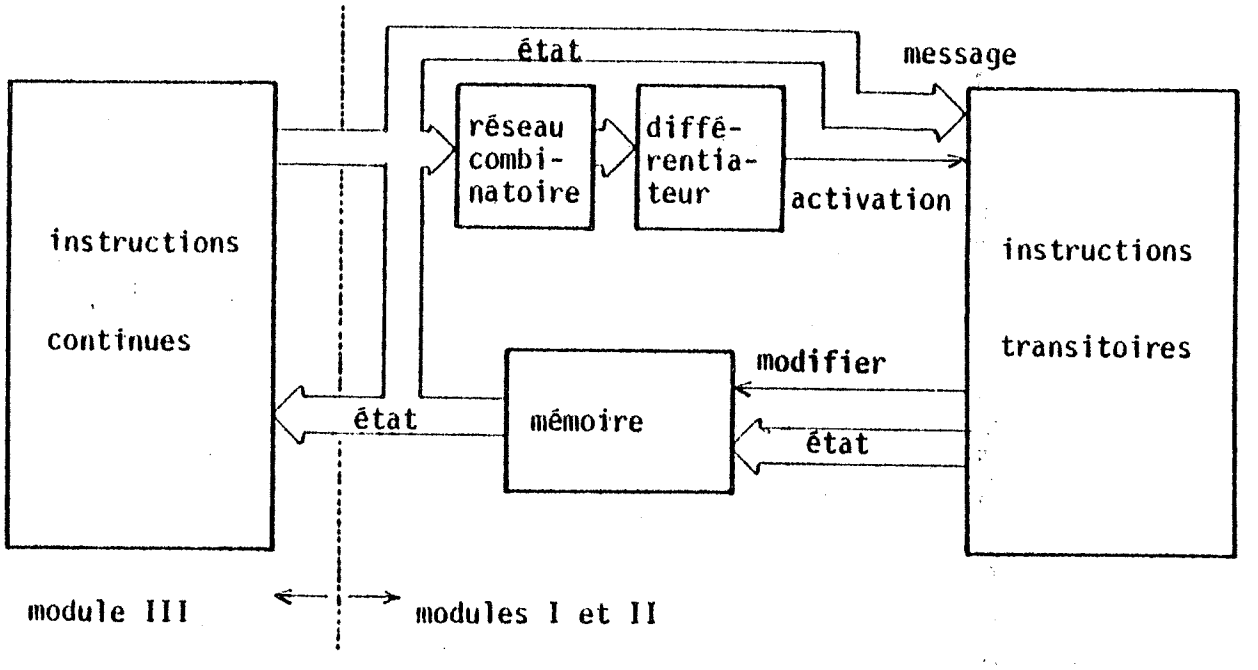


Figure 4.2. Communication entre modules

Pour réaliser la conversion message → état, on utilise un automate séquentiel. Pour réaliser la conversion en sens inverse, on peut utiliser des circuits sensibles à une transition d'une variable d'état (des "différentiateurs").

2. REPRESENTATION DE L'OBSERVATEUR DANS LES PROGRAMMES DE CONTROLE PROCEDURAUX

Dans un programme de contrôle procédural, l'observateur du procédé contrôlé manipule deux types de variables, représentées respectivement par deux types d'automates :

a) Automates explicites, dont la valeur est attribuée soit directement par des capteurs, soit par des expressions d'activation d'automates explicites ; exemples :

```
VOLUME := LECTUREDUTRANSDUCTEUR ;
POSITIONMOTEUR := POSITIONDUMOTEUR + 1 ;
```

b) Automates de séquence : les programmes procéduraux peuvent être considérés comme étant constitués d'une séquence d'étapes qui peuvent représenter implicitement une ou plusieurs variables d'état. Le mécanisme de contrôle de séquencement est un automate. L'étape de la séquence, ou état, est alors donnée par un "compteur d'étapes" dont la valeur est implicitement associée à la région du programme en exécution.

Exemple :

soit le segment de programme suivant :

```
        ...
        AVANCER JUSQUA FINDECOURSE ;        & état 1
ENCORE: CHAUFFER PENDANT 5 MINUTES ;        & état 2
        SI TEMPERATURE < REFERENCE
        ALORS ALLERA ENCORE ;
        OUVRIRVANNE ;                       & état 3
        ...
```

L'automate gérant la variable de séquence est explicité dans la modélisation du contrôleur suivant la méthodologie proposée (figure 4.3.).

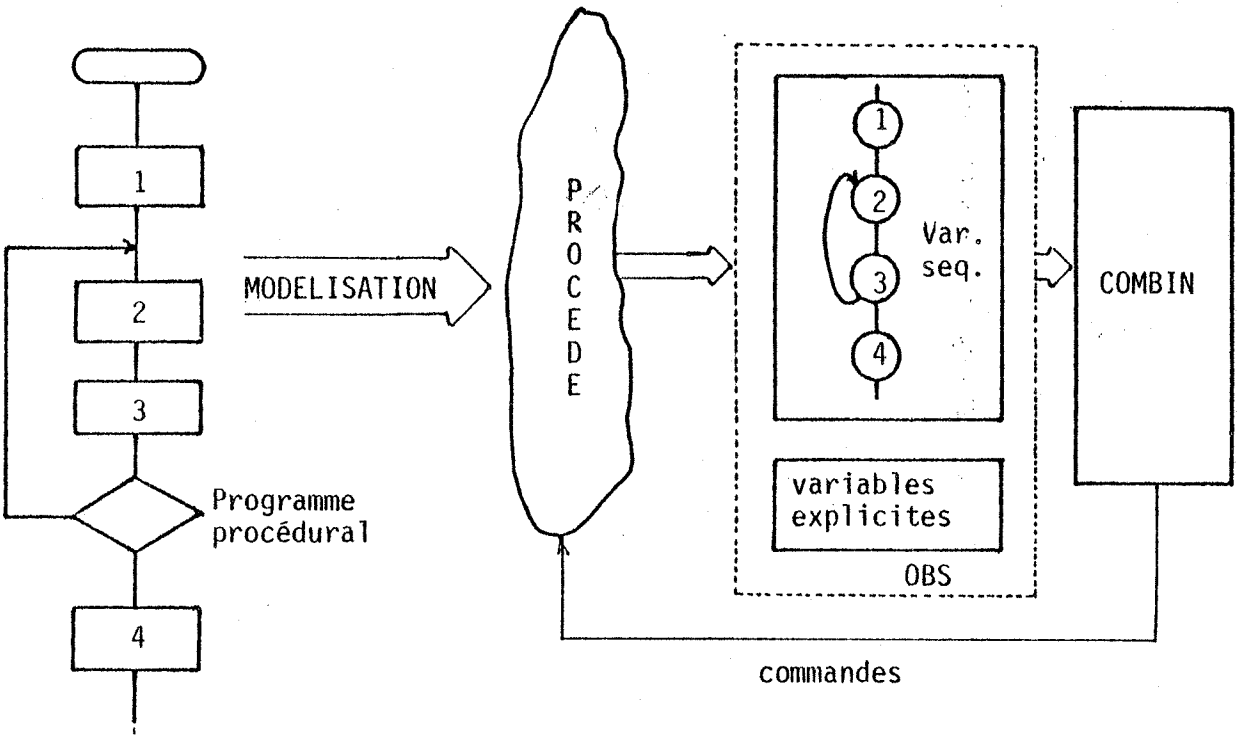


Figure 4.3. Modélisation d'un programme de contrôle procédural

Dans un programme de contrôle non procédural, l'observateur est constitué exclusivement d'automates explicites.

3. LE PARTAGE DYNAMIQUE DES RESSOURCES

Ce paragraphe a pour but de présenter une analyse des outils de partage dynamique des ressources et, par conséquent, de synchronisation de processus (par exemple les sémaphores) du point de vue du formalisme proposé. On examinera notamment la fonction des automates de l'observateur dans ces outils et la manière dont ils sont réalisés dans les langages procéduraux.

On dira que le partage est dynamique lorsque l'utilisation de la ressource se décide en temps réel (en opposition au partage "statique" où les décisions sont prises a priori (c'est-à-dire à la conception de l'algorithme de contrôle).

Les automates de l'observateur reproduisent les variables du procédé contrôlé dont l'état n'est pas visible par des capteurs. Le compromis entre l'utilisation de capteurs et l'incorporation des automates au contrôleur, est (1) économique et (2) technique. Il n'est pas toujours possible de gérer le partage dynamique si l'état de la ressource est mesuré par des capteurs. L'utilisation d'automates, c'est-à-dire de variables mémorisées, s'impose dans le contrôleur chargé de la gestion. En d'autres termes, le contrôleur ne peut pas être toujours purement combinatoire.

Les utilisateurs et les ressources partagées sont des composantes du procédé contrôlé (chacune de ces composantes constitue un "processus" dans la terminologie des systèmes d'exploitation des ordinateurs).

Le concept d'observateur intervient de deux manières :

(1) l'observateur de l'état de disponibilité de la ressource, composé de variables mémorisées ;

(2) l'observateur de l'état de chaque utilisateur par rapport à la ressource.

3.1. L'observateur de la disponibilité d'une ressource partagée

Nous définissons la disponibilité d'une ressource partagée comme la configuration du sous-ensemble des composantes du vecteur d'état d'une ressource qui sont significatives dans la décision de l'attribution de cette ressource à ses utilisateurs. La disponibilité peut être composée de variables réelles, entières ou booléennes (en général ces deux derniers). Ces composantes peuvent être consultées et éventuellement modifiées.

Pour le moment, le fait que la disponibilité soit fournie par des capteurs ou reconstituée par un automate interne au contrôleur, n'est pas significatif.

Chaque utilisateur évolue indépendemment. Pour pouvoir utiliser la ressource, il doit d'abord savoir si elle est disponible (c'est-à-dire si l'état de sa disponibilité permet son utilisation). Le cas échéant, il la prend. La ressource doit alors devenir non-disponible, ou moins disponible, de façon à ce qu'un deuxième utilisateur ne risque pas de la trouver "également" disponible.

Le danger d'incohérence est dû au temps entre la demande et la réponse. A partir de l'instant où la ressource est attribuée à l'utilisateur, il se passe un certain intervalle de temps, jusqu'à ce que :

- l'utilisateur décide de prendre la ressource effectivement,
- l'état de disponibilité de la ressource soit modifié par l'action de l'utilisateur.

Pour éviter cette incohérence, l'accès à la disponibilité de la ressource doit donc être réalisé en mutuelle exclusion entre les différents utilisateurs, c'est-à-dire que l'information de disponibilité doit rester inaccessible entre une lecture et son éventuelle modification.

Pour justifier l'existence de variables de disponibilité internes au contrôleur, essayons de concevoir la gestion de deux manières (figure 4.4.) :

a) avec des capteurs de disponibilité placés sur la ressource ;

b) avec des variables de disponibilité internes au contrôleur.

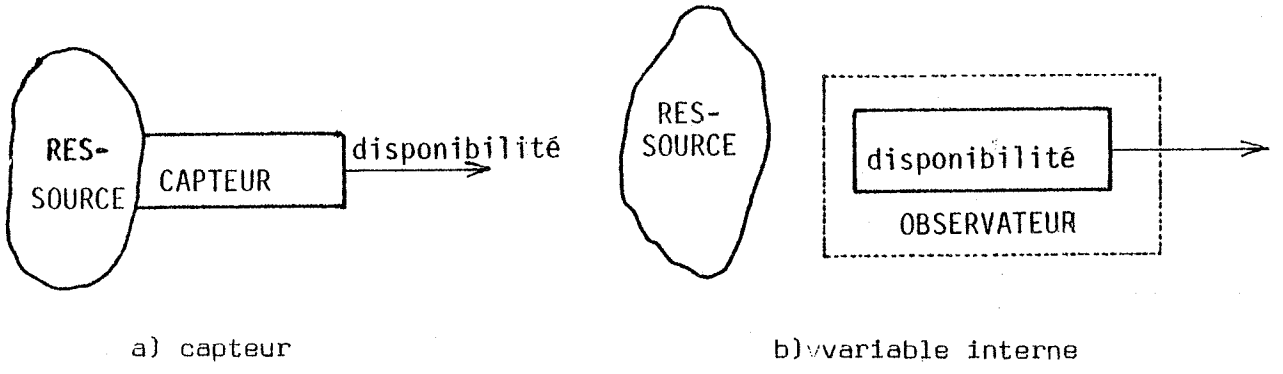
Supposons que la ressource soit un tampon dans lequel les utilisateurs producteurs déposent et les consommateurs retirent un nombre d'éléments spécifié lors de la demande. Chaque action (déposer ou retirer) prend un temps fini non nul après la permission. Plusieurs actions doivent pouvoir se réaliser en parallèle. Il faut donc faire une distinction entre (a) l'accès à la disponibilité (nécessairement en exclusion mutuelle) et (b) l'utilisation de la ressource proprement dite (éventuellement en parallèle). La permission de retirer est donnée s'il y a un nombre suffisant d'éléments dans le tampon ; la permission de déposer est donnée lorsqu'il y a un nombre suffisant de places vides.

Solution avec un capteur de nombre d'éléments (figure 4.4.a.) :
un seul utilisateur peut utiliser la ressource à la fois, puisque le capteur doit rester invisible aux autres utilisateurs pendant que sa valeur peut être modifiée par l'utilisateur présent. Le parallélisme des actions n'est donc pas possible dans ce cas.

Solution avec des variables internes (figure 4.4.b.) :
par contre, si la disponibilité est représentée par des variables mémorisées internes au contrôleur, l'accès en exclusion mutuelle à ces variables n'implique pas l'exclusion mutuelle pour l'utilisation de la ressource. L'accès aux variables de disponibilité prend alors un caractère de "réservation" d'une certaine quantité "d'unités d'utilisation" et l'utilisation proprement dite se fait postérieurement. Les variables de disponibilité peuvent alors être "en avance" par rapport à l'état de la ressource. Dans les systèmes d'exploitation, ces variables correspondent :

- aux sémaphores de Dijkstra [DIJK 68] ;
- aux variables mémorisées déclarées dans les moniteurs [HOAR 74] et dans Concurrent PASCAL [BRHA 75] ;
- aux compteurs de synchronisation [ROBE 77] et d'événements [REED 79].

Dans les contrôleurs industriels de bas de gamme, programmables avec des fonctions booléennes, le support matériel n'intègre pas en général de variables mémorisées (observateur) de disponibilité protégées par exclusion mutuelle, ce qui empêche le partage dynamique des ressources, sauf recours à des moyens supplémentaires.



a) capteur b) variable interne

Figure 4.4. Obtention de la disponibilité

3.1.1. Application de la variable de disponibilité

Pour mieux illustrer le comportement des "variables de disponibilité", nous examinons leur manifestation comme sémaphores et dans les moniteurs :

a) Sémaphores :

les sémaphores sont des variables entières, auxquelles un "nombre initial d'unités d'utilisation disponibles" est attribué à la mise en marche du contrôleur. A chaque fois qu'un utilisateur demande la ressource, la valeur du sémaphore est décrémentée et consultée. Selon le résultat:

. ou bien l'utilisateur prend la ressource immédiatement,

. ou bien il attend dans une file que la valeur du sémaphore augmente à la suite de la restitution d'une "unité d'utilisation" par un utilisateur qui a terminé de s'en servir (augmentation de la disponibilité).

La valeur du sémaphore, à chaque instant, est donc égale :

. au nombre d'unités d'utilisation initialement disponibles,

. moins le nombre d'unités déjà prises (en utilisation),

. moins le nombre d'unités réservées par les utilisateurs en file, } opérations P

. plus le nombre d'unités rendues — opération V

En fait, les sémaphores peuvent être utilisés pour l'observation d'une ressource dont la disponibilité peut être représentée par un entier. L'observateur est alors un compteur qui n'est accédé que par des opérations protégées. On peut imaginer des conditions sémantiquement plus complexes qui exigeraient des observateurs plus élaborés. L'accès à l'ensemble des variables et la prise de la ressource doivent alors être rendus indivisibles par des sémaphores de mutuelle exclusion à la section critique d'accès. Les sémaphores de mutuelle exclusion sont également des observateurs de disponibilité de la section critique, qui doivent aussi être protégés par l'exclusion mutuelle propre au support matériel

[ANCE 78]

b) Moniteurs :

dans les moniteurs et les constructions qui en découlent, les variables mémorisées qui représentent la disponibilité de la ressource sont déclarées dans un module protégé, le moniteur, chargé de sa gestion. Ces variables sont consultées à l'entrée de chaque procédure du moniteur (le moniteur est constitué d'un ensemble de procédures qui correspondent aux actions demandées à la ressource). Selon le résultat de cette consultation aux variables internes (l'observateur) :

. ou bien l'utilisateur prend la ressource, c'est-à-dire réalise l'action demandée et l'état de l'observateur est aussitôt modifié,

. ou bien, si la ressource n'est pas disponible, un indice (l'identificateur) de l'utilisateur est mis dans une file d'attente, d'où il sera retiré lorsque une autre action rend la ressource disponible.

La valeur des variables représentant la disponibilité est donc à chaque instant égale à la "disponibilité initiale", moins le nombre d'unités d'utilisation prises par les utilisateurs présents (figure 4.5.).

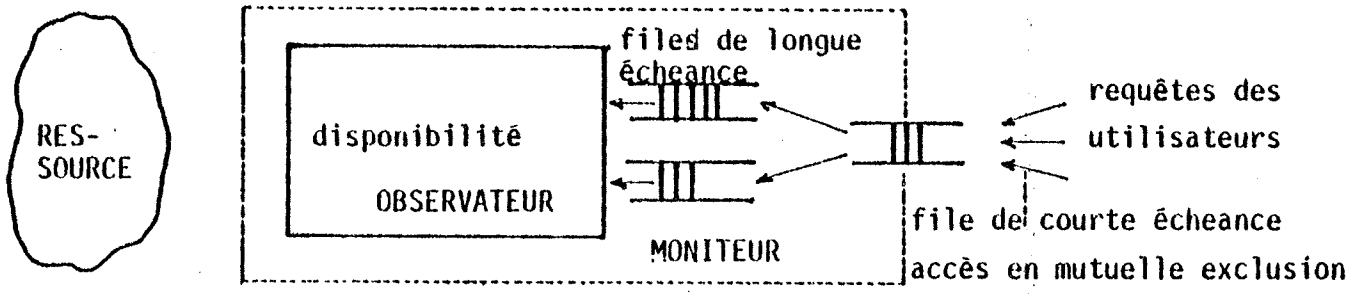


Figure 4.5. Représentation du moniteur selon le modèle proposé

Par contre, la déclaration des "classes" (dans Concurrent Pascal) qui contrôlent des ressources non partagées (un seul utilisateur) n'incluent pas de variables mémorisées qui joueraient le rôle d'observateur.

3.2. L'observateur de l'état de l'utilisateur

Vis à vis de la ressource, chaque utilisateur réalise un processus de trois états :

. repos : l'utilisateur n'utilise pas la ressource et ne l'attend pas,

. attente : l'utilisateur n'utilise pas la ressource mais l'attend,

. utilisation : l'utilisateur utilise la ressource.

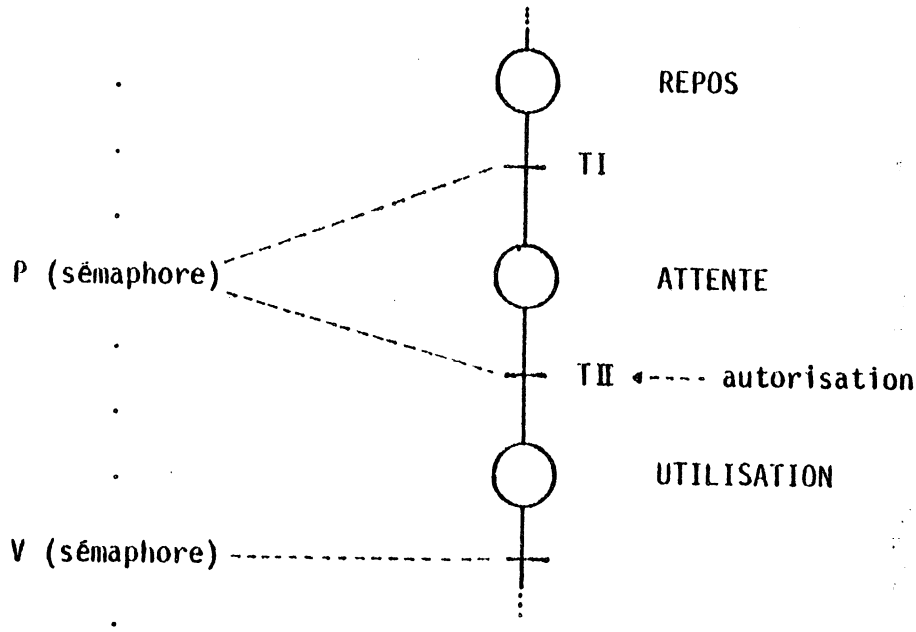
L'accessibilité limitée aux automates de disponibilité (sous la forme de sémaphores, par exemple) impose la mémorisation de la réponse du contrôleur de la ressource. Cette mémorisation peut se réaliser par l'affectation de l'état du procédé utilisateur proprement dit, ou alors par un "automate d'utilisation" implicite ou explicite (observateur) dans les programmes des utilisateurs.

Les automates observateurs de l'état de l'utilisateur sont activés par des instructions transitoires exécutées par l'utilisateur (requêtes) et par le contrôleur de la ressource (réponses).

Selon la figure 4.1., les instructions transitoires constituent des modules procéduraux et non procéduraux de type 2.

Dans les utilisateurs procéduraux, la réponse "en utilisation" est donnée par le passage du compteur ordinal à la barrière constituée par la primitive d'attente (figure 4.6.a.).

Pour un utilisateur non procédural, l'automate de la figure 4.6.b. est un automate explicite et il faut s'assurer que les transitions T1 et T2 constituent une opération indivisible.



a) module procédural b) module non-procédural

Figure 4.6. Etat d'un utilisateur vis à vis de la ressource

4. CONCLUSION

Le formalisme proposé permet de montrer assez nettement les limitations des modules continus dans la spécification du contrôleur de procédés temps réel, notamment lorsqu'elle concerne le partage dynamique des ressources.

Les langages purement combinatoires des automates programmables sont donc un moyen convenable pour la programmation de plusieurs types d'applications, mais il faut les compléter dans certains cas par des modules séquentiels.

CHAPITRE 5

ALGORITHMES ET AUTOMATES DE SEQUENCE

1. MICROPROGRAMMATION
2. APPLICATION A LA CONCEPTION DE PROCESSEURS INFORMATIQUES
3. APPLICATION AU CONTROLE DE PROCEDES TECHNIQUES
4. APPLICATION A L'INTERPRETATION D'UN SYSTEME PROCEDE/CONTROLEUR
5. INITIALISATION ET REDEMARRAGE DU PROCEDE CONTROLE
6. DETECTION DES EXCEPTIONS - OBSERVATEUR DE SURVEILLANCE

CHAPITRE 5 - ALGORITHMES ET AUTOMATES DE SEQUENCE

Ce chapitre a pour but d'examiner la manière dont s'expriment les éléments du modèle proposé, dans le contrôle des procédés dont le comportement est décrit par un algorithme, c'est-à-dire par une séquence d'états et d'actions qui réalisent une action de niveau supérieur.

Un programme de contrôle procédural est exécuté à l'aide d'un automate dont l'état indique le segment ou l'instruction en cours d'interprétation. Cet automate fait partie de l'observateur du procédé contrôlé en reproduisant quelques unes des composantes de son vecteur d'état.

Dans un premier temps, on rappelle la notion de microprogrammation. On remarque que la réalisation interprétative (microprogrammation) d'une action ou instruction n'est possible que grâce à la prédictabilité (§ 3.6.) du comportement du procédé contrôlé. En d'autres termes, on peut concevoir a priori la séquence des commandes du procédé parce qu'on connaît l'ensemble des successeurs possibles d'un état du procédé.

On regarde ensuite l'application de la notion de microprogrammation, c'est-à-dire la fonction, du point de vue de la notion d'observateur, des automates de cycle, de phase, etc .., des :

. dispositifs de traitement de données (informatiques),
. contrôleurs de procédés techniques, c'est-à-dire ceux qui impliquent des transformations mécaniques, physiques, chimiques, etc ..

Finalement on verra quelques aspects particuliers, tels que l'initialisation et la détection des exceptions.

1. MICROPROGRAMMATION

Nous définissons une "action" comme une transformation d'un élément OPERANDE en un RESULTAT :

ACTION (OPERANDE) → RESULTAT

Dans le contexte d'un langage L, les plus petites actions distinguables sont les instructions I(L). Chaque instruction peut être interprétée :

a) directement, c'est-à-dire par des actions "primitives" du matériel d'interprétation,

b) ou par le déroulement de plusieurs étapes mettant en oeuvre des actions plus élémentaires, assemblées en un algorithme d'interprétation A(I(L)) appelé microprogramme.

Le choix entre ces deux options se fait en fonction des contraintes économiques et techniques : lorsque plusieurs instructions différentes peuvent profiter des mêmes ressources d'interprétation, l'approche (b) devient intéressante, bien qu'il existe d'autres facteurs à considérer, tels que les retards et le parallélisme.

Ces contraintes découlent de la puissance limitée des organes de traitement de données et des procédés techniques. La limitation de puissance a comme conséquence que l'accomplissement d'une action (représentant une quantité de "travail" se réalise sur un certain intervalle de temps, représenté par une suite d'actions de niveau plus élémentaire.

Exemple :

un système de communication à un seul canal utilisé pour transmettre des caractères ASCII (figure 5.1.)

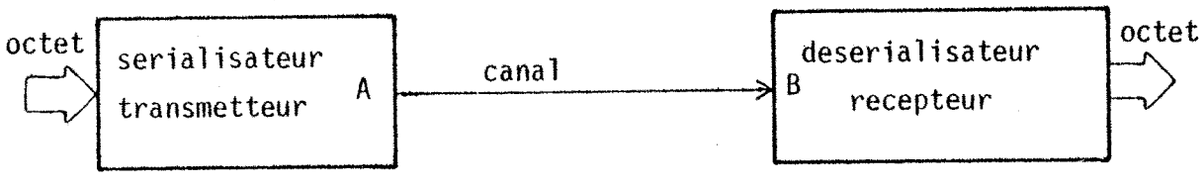


Figure 5.1. Système de communication

L'instruction à interpréter par A est "transmettre un octet". La ressource limitée est la ligne de communication qui traite un seul bit à la fois. Le transmetteur A contrôle un procédé d'allocation de la ressource ligne ; l'observateur de ce procédé est constitué d'un automate dont l'état représente le bit utilisateur (de Ø à 7) auquel la ressource ligne est attribuée à chaque phase. La structure de cet observateur est déterminée à la conception de celui-ci, en fonction d'un ordre prédéterminé des bits utilisateurs (par exemple de poids faible ou fort). Comme la séquence n'a pas de branchements, la progression de l'observateur peut être commandée par le seul signal de fin d'émission du bit courant.

L'automate est activé par simulation : l'horloge simule le temps nécessaire à l'émission du bit sur la ligne. La connaissance a priori du protocole (prédictabilité du phénomène), permet la conception d'un observateur assez simple.

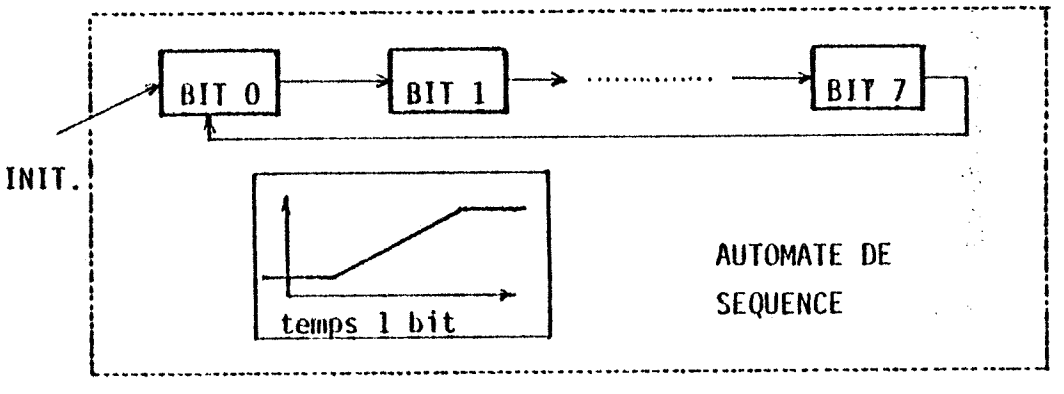


Figure 5.2. Observateur de l'état d'utilisation de la ressource "système de communication"

1.1. L'observateur de l'état de la ressource de traitement d'un microprogramme

Chaque instruction I(L) est interprétée par un algorithme A(I(L)), appelé microprogramme. Les algorithmes sont écrits dans un langage L1 dont les plus petites actions sont I1(L1). Par rapport au langage d'instructions L, L1 est appelé langage de microprogrammation. Les actions décrites en langage L1 sont les microinstructions.

Chaque "ressource d'interprétation" est capable d'interpréter un sous-ensemble des instructions I1 d'un langage L1, éventuellement toutes. Le langage L admet un ensemble d'instructions I :

$$\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$$

chacune étant interprétée par un algorithme A(I(L)) écrit avec les instructions I1(L1), lesquelles constituent à leur tour un ensemble

$$\mathcal{I}1 = \{I1_1, I1_2, \ldots, I1_n\}$$

La ressource d'interprétation du langage L aura donc Card($\mathcal{I}1$) états.

L'ensemble des microprogrammes, ou algorithmes des instructions du langage L, configure l'observateur de sa ressource d'interprétation, avec card($\mathcal{I}1$) états et les transitions données par la suite des microinstructions de chaque microprogramme.

Chaque microinstruction peut comporter les champs suivants :

- microopérations qui, soit commandent directement les portes des organes physiques d'exécution, soit constituent les microinstructions du niveau inférieur,
- adresse suivante : prochaine microinstruction à interpréter,
- paramètres divers,
- retard.

Le champ "retard" indique la durée de l'exécution d'une microinstruction, en termes de multiples de phases d'horloge. Il est normalement implicitement lié au type de microinstruction. La fin d'une phase d'horloge signale que les ressources nécessaires pour l'exécution de l'opération associée ont fini d'être utilisées ; elle est générée par "simulation".

L'horloge a donc deux fonctions :

. représenter le temps d'utilisation de l'ensemble des ressources attribuées à la phase active,

. activer le flot des données en commandant les portes.

Pour simplifier, le temps d'utilisation des ressources de toutes les phases est considéré égal au temps nécessaire à la phase la plus lente, ce qui a comme conséquence de permettre l'utilisation d'une horloge de fréquence constante.

2. APPLICATION A LA CONCEPTION DE PROCESSEURS INFORMATIQUES

Dans un processeur l'ordre des événements découlant de chaque instruction est établi au moment de la conception de la machine. On peut y distinguer deux parties :

. la partie opérative qui correspond au procédé contrôlé : elle est composée de registres, d'opérateurs et de lignes de communication ;

. la partie contrôle correspondant au contrôleur du procédé.

L'instruction à interpréter est reçue par la partie contrôle. En toute rigueur il serait possible (et éventuellement monstrueux) d'exécuter l'instruction en une seule étape à l'aide d'une partie opérative adéquate et d'une partie contrôle combinatoire, commandant directement les organes de la partie opérative en fonction de l'état de celle-ci et de l'instruction. On aurait alors besoin d'une unité de traitement adaptée en largeur de mots et en nature d'opération, ayant la possibilité d'accéder simultanément à tous les opérandes et à l'instruction.

La limitation des ressources de traitement et de communication amène à réaliser séquentiellement l'interprétation d'une instruction. La séquence est commandée par un automate qui constitue l'observateur de l'état d'utilisation de chaque ressource, selon le modèle proposé. La commande du procédé contrôlé (partie opérative) est calculée par un circuit combinatoire (normalement un PLA) en fonction de l'état de l'automate (cycle, phase), de la nature de l'instruction et éventuellement du contenu de la partie opérative (les instructions conditionnelles). Ce PLA constitue la partie combinatoire du contrôleur.

Nous avons donc :
procédé contrôlé = (registres, opérateurs, lignes de communication) ;
contrôleur = (observateur, partie combinatoire) ;
observateur = (automate de cycles, automate de phases) ;
partie combinatoire = (PLA, mémoire de microprogrammes).

2.1. Rémanence des objets dans les processeurs

Du fait que les instructions exécutées par un processeur sont de type "transitoire (§ 4.1.), il est nécessaire de mémoriser les valeurs intermédiaires dans des registres, mémoires externes, etc .., que nous appelons "objets".

Nous considérons deux cas d'enchaînement de deux instructions successives :
. ou bien ces instructions font partie d'un même algorithme du niveau d'interprétation supérieur,
. ou bien elles appartiennent à des algorithmes différents.
Dans le premier cas, l'exécution de la première instruction génère des données de sortie qui sont utilisées comme données d'entrée par la suivante. Les emplacements qui mémorisent ces informations peuvent être considérés comme la partie "objet" du procédé d'interprétation de l'algorithme.

Dans le cas des microprocesseurs et au niveau du programme exécuté, l'objet est partiellement interne - registres visibles - et partiellement placé en mémoire externe. Des objets peuvent être reconnus à chaque niveau d'interprétation.

La durée de vie, ou rémanence d'un objet, est le temps pendant lequel son état reste inchangé. Pour un même procédé, la durée de vie croît avec le niveau d'interprétation : la durée de validité d'un contexte est la durée de vie d'une tâche du niveau considéré.

Exemples :

. au niveau de l'interprétation du langage de cycles du microprocesseur 8008 on trouve, par exemple, les registres "a" et "b" (ne pas confondre avec A et B, figure 5.3.), invisibles au niveau de la programmation ;

- au niveau de l'interprétation du langage de phases d'horloge du microprocesseur M6800, les bus se comportent comme des éléments de mémorisation temporaire (quelques microsecondes).

Entre l'exécution de deux instructions du niveau immédiatement supérieur, les objets d'un niveau donné perdent leur validité.

Exemples :

- entre deux instructions, les registres "a" et "b" du 8008 peuvent être effacés,
- la capacité des bus du M6800 se décharge en quelques cycles.

2.2. Application du modèle au microprocesseur I8008

Le microprocesseur I8008 [INTE 73] fut le premier microprocesseur de 8 bits d'utilisation générale. Nous faisons ici une analyse de ses éléments internes dans l'esprit du modèle proposé. On remarque l'existence d'une machine d'états finis dont la fonction est de gérer un ensemble de ressources limitées. Cette machine d'états finis (figure 5.5.) peut être considérée comme l'automate de gestion de la ressource partie opérative. Elle est conçue avec la connaissance préalable des fonctions qui lui ont été statiquement attribuées. La figure 5.3. représente la partie combinatoire, représentée plus schématiquement par la figure 5.6.
Il y a une seule porte de 8 bits pour la communication des données avec l'extérieur. Les adresses (14 bits) et les données proprement dites (8 bits) doivent donc passer par cette porte. L'instruction JUMP, par exemple, exige trois accès à la mémoire : une pour chercher l'instruction et deux pour chercher l'opérande (de 14 bits, puisqu'il s'agit de l'adresse de destination du saut). Chaque accès à la mémoire exige trois utilisations de la porte : deux sorties pour indiquer l'adresse où se trouve l'information recherchée et une entrée pour la réception de cette information, soit au total 3 x 3 = 9 utilisations de la porte.

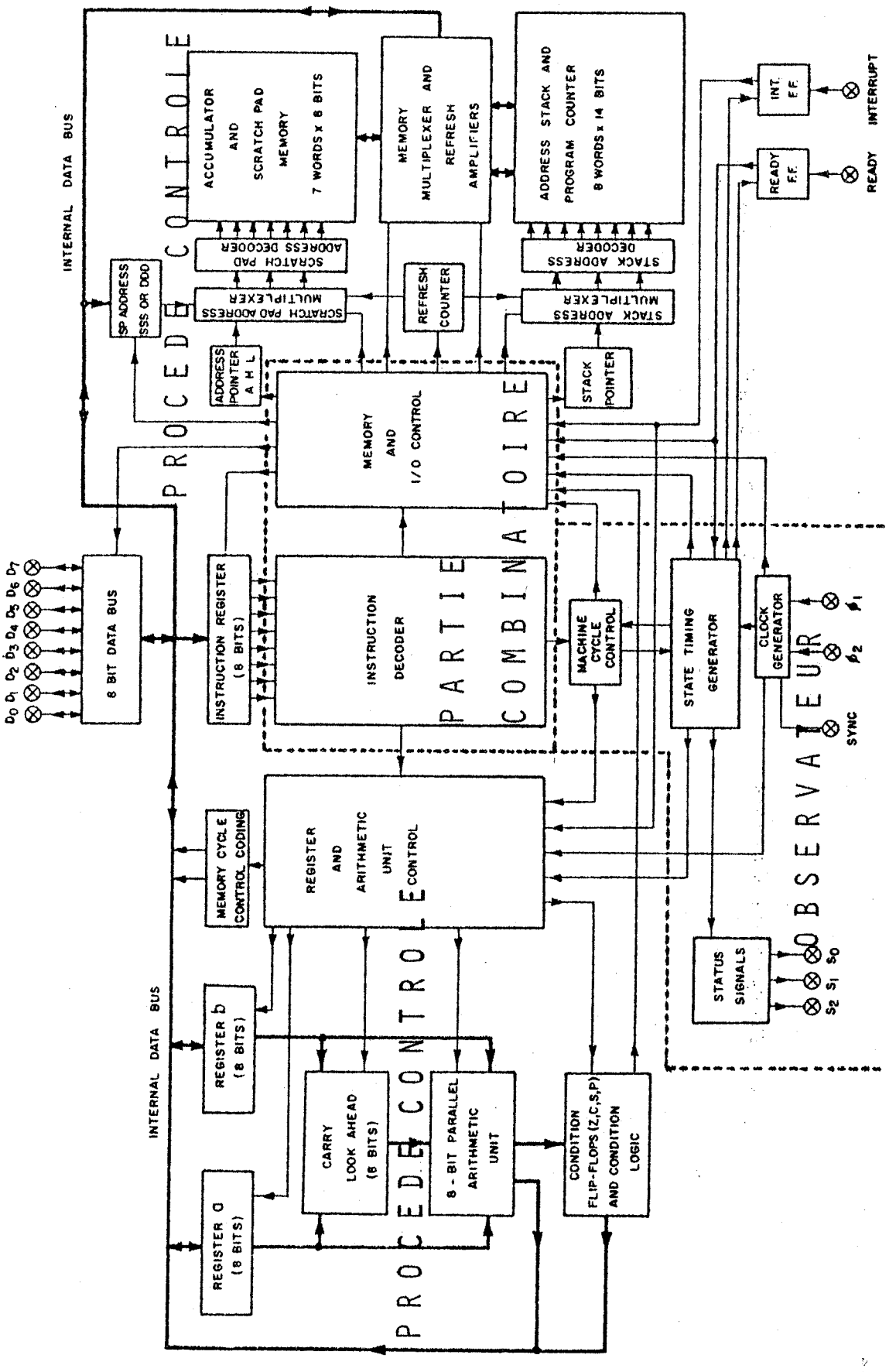


Figure 5.3. Blocs du microprocesseur I 8008

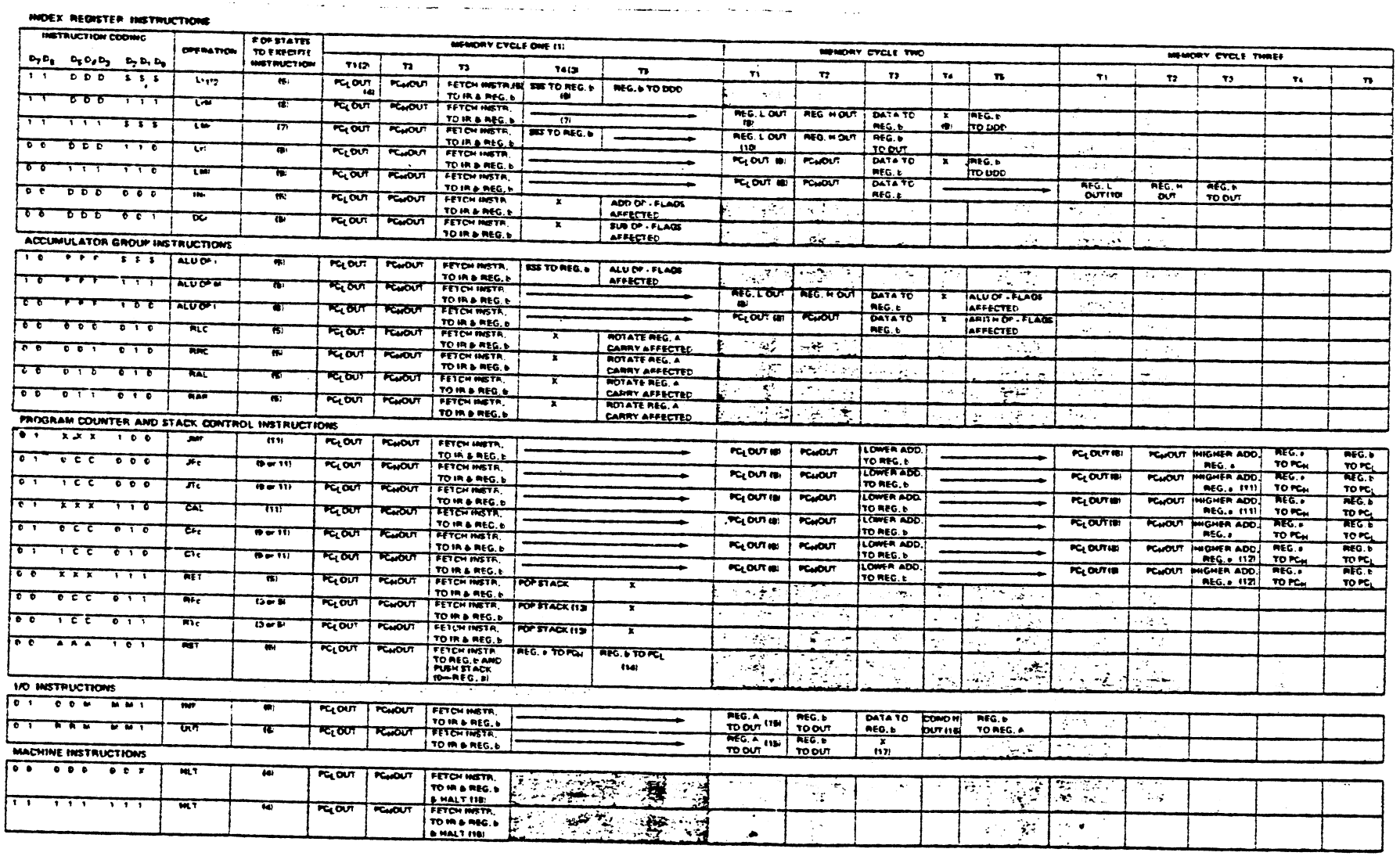


Figure 5.4. Partie combinatoire du microprocesseur I 8008

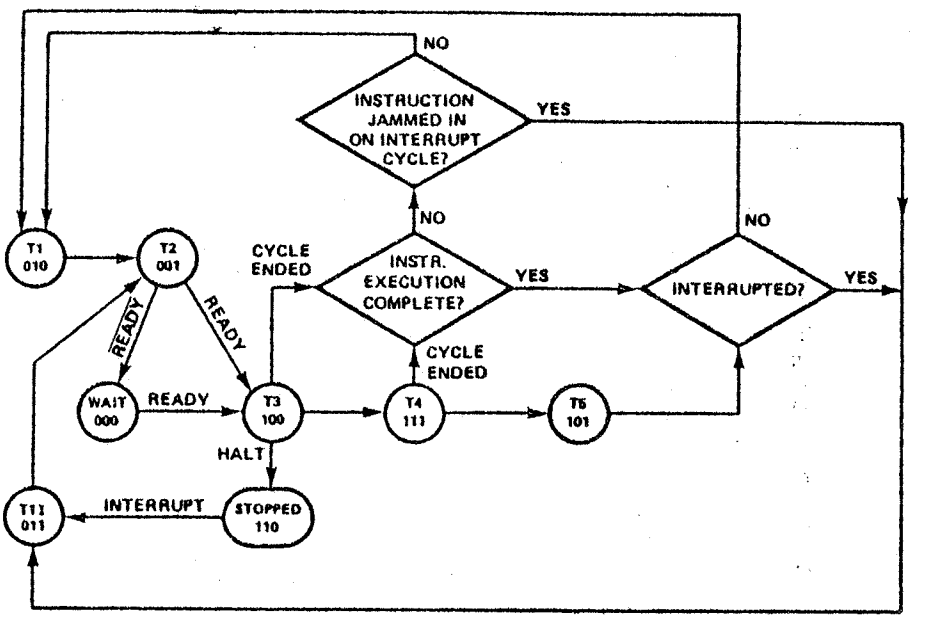
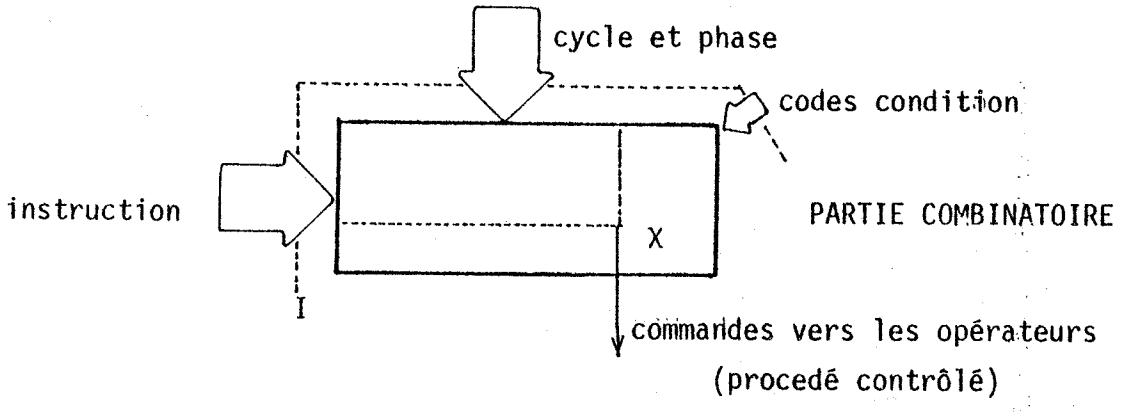


Figure 5.5. L'automate de phases du I8008



I = {instruction, cycle et phase, codes condition}

Figure 5.6. Partie combinatoire (schéma simplifié)

Pour cette instruction, l'automate de la figure 5.5. indique que la porte est dédiée à :
cycle 1, état 1 - adresse, poids faible, du 1er octet de l'instruction,
cycle 1, état 2 - adresse, poids fort, du 1er octet de l'instruction,
cycle 1, état 3 - réception de l'instruction,
etc ...

Les commandes générées par la partie combinatoire (figure 5.6.) indiquent l'action à réaliser sur ces ressources. Le calcul de l'attribution des ressources est fait à la conception des algorithmes d'interprétation de chaque instruction.

3. APPLICATION AU CONTROLE DE PROCEDES TECHNIQUES

Le concept de la microprogrammation s'applique, avec quelques adaptations, au contrôle des procédés techniques. Si dans la microprogrammation "de traitement", les organes d'opération sont l'UAL, la mémoire et les interfaces, dans la microprogrammation "de contrôle", l'organe d'exécution est un procédé technique contrôlé. Il y a cependant des différences à prendre en compte :

(a) microprogrammation "de traitement" : la fin de l'exécution d'une micro-instruction est signalée par l'horloge au niveau de l'interprétation des "phases" (très proche du matériel), ou par l'exécution d'une instruction du type "FIN" dans les niveaux supérieurs ;

(b) microprogrammation de contrôle de procédés "techniques" : la fin de l'exécution d'un microprogramme peut être signalée par un état attendu a priori et indiqué explicitement.

En fait, une commande vers le niveau inférieur peut être interprétée de deux manières (ainsi qu'une instruction interprétée par un processeur informatique) :

(a) directement au niveau du matériel physique :
exemple : "chauffer tant que $t < t_{ref}$" (figure 5.7.)

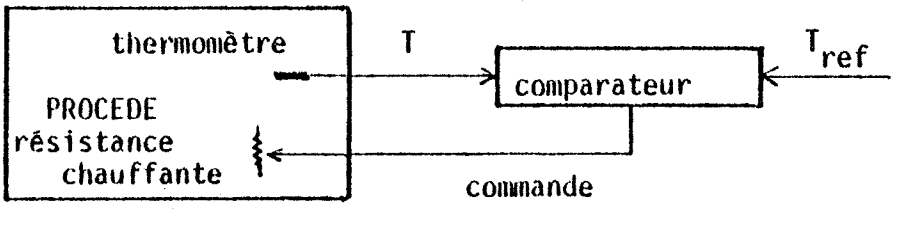
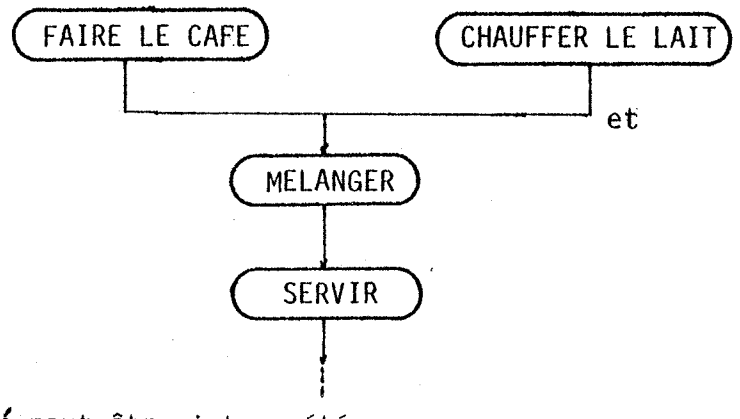


Figure 5.7. Interprétation "directe"

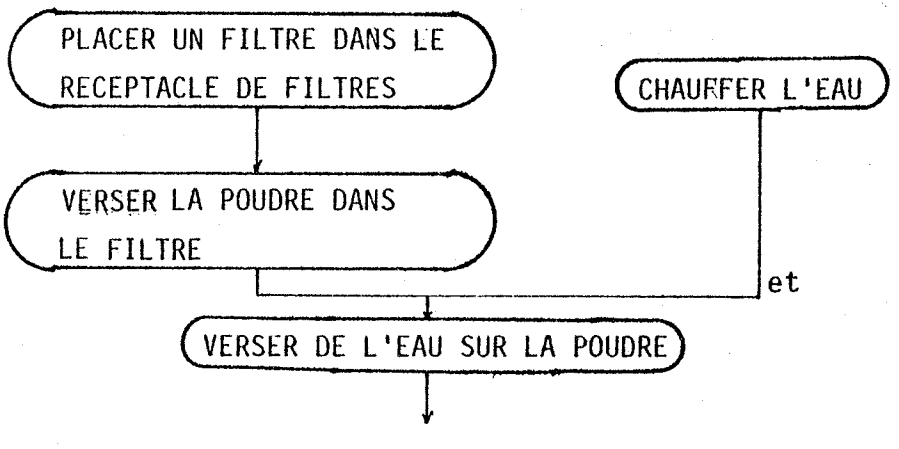
(b) comme une instruction qui déclenche une séquence de phases. Ces phases correspondent aux états du procédé contrôlé au niveau de l'interpréteur de l'instruction. La fin de l'exécution de la séquence est indiquée par l'arrivée à un état défini a priori. Cette notion a été plus ou moins explicitement utilisée dans une méthode de structuration de systèmes temps réel proposée par [MEND 75], dans laquelle le comportement de chaque composante du procédé est considéré comme un "module" dont les interconnexions sont définies par le niveau supérieur.

Un procédé technique peut donc être vu comme un interpréteur de microprogrammes. Il reçoit une commande I1 et évolue en exécutant l'algorithme A1 associé, jusqu'à ce que son état justifie une nouvelle commande. Ce mécanisme se reproduit sur plusieurs niveaux d'interprétation depuis le plus bas, qui correspond aux opérateurs câblés combinatoires des systèmes informatiques.

Exemple : la préparation d'une tasse de café au lait peut être interprétée par :



Faire le café peut être interprété par:



"Placer un filtre dans le récepteur de filtres" correspond à la fois à un ensemble assez complexe de mouvements qui peuvent être décomposés en actions plus élémentaires. L'état du procédé peut être représenté sous la forme de différents niveaux d'abstraction, correspondant aux niveaux d'interprétation. Si l'état n'est pas explicitement fourni par le procédé, il doit être reproduit par l'observateur. Il est possible de décrire celui-ci sous la forme d'une hiérarchie d'interpréteurs. Cette description de l'automate fait partie de la programmation du contrôleur et peut être interprétée, en informatique, par les mêmes ressources de traitement que pour la partie combinatoire.

Syntaxiquement, les différents niveaux d'interprétation, et par conséquent les différents niveaux d'observateurs de séquencement, peuvent être placés dans un même contrôleur (comme dans [ANCE 78]), ou bien sous la forme de procédés imbriqués, chaque procédé associé à sa propre commande constituant le procédé contrôlé de niveau supérieur (figure 5.8.).

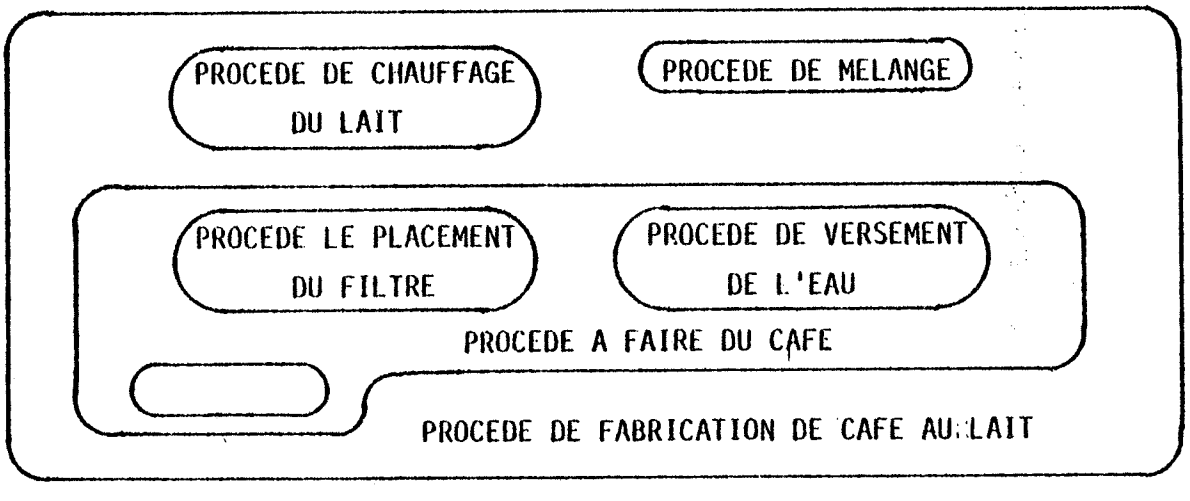


Figure 5.8. Hiérarchie des procédés techniques

4. APPLICATION A L'INTERPRETATION D'UN SYSTEME PROCEDE / CONTROLEUR

De ce que nous venons de voir dans les paragraphes précédents, il découle qu'un interpréteur existe de chaque côté de tout découpage procédé/contrôleur. Le procédé interprète les commandes du contrôleur tandis que le contrôleur interprète les événements issus du procédé. Ce découpage est arbitraire ; on a vu qu'il peut exister plusieurs couches d'interprétation, aussi bien du côté informatique que du côté technologique (application).

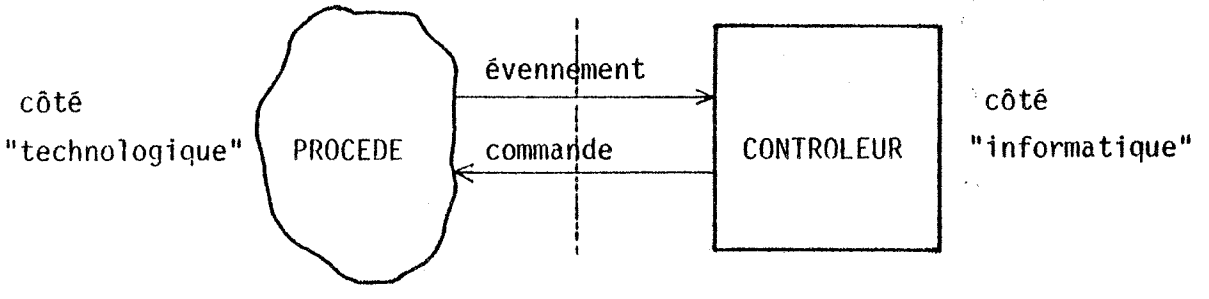


Figure 5.9. Interprétation des commandes du contrôleur et des événements du procédé.

Du côté du contrôleur, un événement doit être considéré comme une instruction du niveau le plus élevé dans la hiérarchie d'interprétation informatique. Cette instruction est interprétée par l'algorithme de calcul écrit par le programmeur du système, lequel est lui-même interprété par le langage de cycles, de phases, etc .. jusqu'aux organes câblés. La vitesse d'interprétation est une fonction de la puissance du contrôleur.

Du côté du procédé, une commande est considérée comme une instruction du niveau le plus élevé dans la hiérarchie d'interprétation technologique. Cette instruction est interprétée par l'algorithme imposé par la structure du procédé. La vitesse d'interprétation est une fonction de la nature du procédé, par exemple la vitesse d'une réaction physique ou chimique, la vitesse d'un moteur.

Les processus qui se déroulent de chaque côté du découpage ont donc leur état représenté par un observateur :

. l'observateur de l'état d'évolution de l'interpréteur de l'algorithme de calcul des commandes (partie combinatoire) ;

. l'observateur de l'état du procédé contrôlé

Les automates qui constituent l'observateur de l'état du procédé contrôlé peuvent résider sur l'ordinateur de contrôle, ou bien sur du matériel externe (par exemple, des compteurs ou des convertisseurs).

5. INITIALISATION ET REDEMARRAGE DU PROCEDE CONTROLE

Il faut tenir compte de deux aspects qui distinguent les algorithmes techniques des algorithmes informatiques, en ce qui concerne le démarrage et la reprise à la suite d'une panne matérielle :

a) La fiabilité :

dans les processeurs informatiques, la fiabilité est, du moins actuellement, beaucoup plus élevée que dans les procédés industriels, à cause du grand ensemble de possibilités d'interférence de l'environnement industriel. Ces interférences sont généralement de nature électrique ou mécanique (bourrages, fuites, etc ..). Elles doivent être prévues à la conception en tant que "conditions d'exception" qui déclenchent l'exécution de procédures spéciales pour le contrôleur. Dans les processeurs interprétant des algorithmes informatiques, les pannes matérielles sont moins fréquentes et en général il n'y a pas de procédures particulières prévues, ne fût-ce que à cause de l'inexistence d'un contrôleur supplémentaire au processeur.

b) La nature des variables d'état :

d'une façon assez intuitive, on peut dire que les objets sur lesquels agit un algorithme informatique ont moins "d'inertie" que les objets d'un procédé technique, c'est-à-dire que l'énergie, et par conséquent le temps nécessaire pour modifier leur état, est inférieur. Par ailleurs, les objets d'un procédé technique subissent des transformations essentiellement irréversibles, tandis que les données d'un algorithme informatique peuvent être facilement sauvegardées.

Par conséquent, l'initialisation d'un algorithme informatique est considéré comme à peu près évidente, c'est-à-dire qu'à chaque fois que le processeur démarre l'interprétation d'un algorithme, il est implicite que toutes les variables d'interprétation (automates de cycles, phases, etc ..) se trouvent dans l'état "de repos". Par contre, l'exécution d'un algorithme technique peut, comme conséquence d'une défaillance quelconque, laisser quelques unes de ses variables dans un état différent de l'état "de repos" ; il faut donc prévoir explicitement des procédures de démarrage et/ou de réinitialisation de ces procédés.

6. DETECTION DES EXCEPTIONS. OBSERVATEUR DE SURVEILLANCE

Une exception correspond à une configuration non autorisée du vecteur d'état du procédé contrôlé. Les critères pour déterminer qu'une configuration correspond à une condition d'exception sont assez subjectifs [GREE 79] et leur étude dépasse la portée de l'analyse développée dans ce chapitre ; elle se borne à l'interprétation des mécanismes de détection des exceptions, selon la notion d'observateur.

Réalisation de l'observateur de surveillance

a) Par une fonction combinatoire (figure 5.10) : l'observateur est alors "transparent", il ne contient par d'éléments de mémorisation.

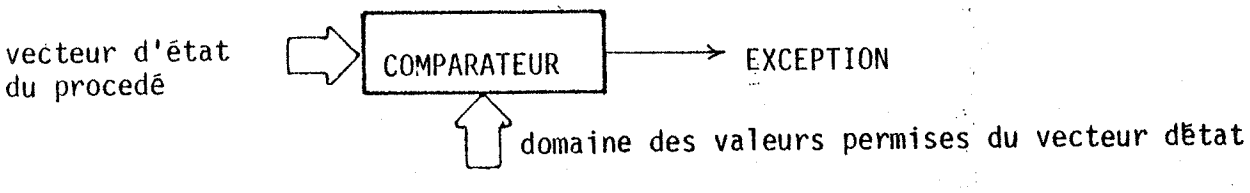


Figure 5.10. Détection combinatoire d'une exception

b) A l'aide d'un automate : le procédé arrive à une configuration d'exception lorsqu'une des composantes de son vecteur d'état - la dernière dans le temps - change de valeur vers une configuration défendue. L'exception peut être détectée en mémorisant l'état précédent du procédé et en faisant une comparaison uniquement sur une composante, au lieu de la faire sur le vecteur, ce qui revient à une simplification. L'exception pourra être signalée par la "présence" ou par "l'absence" d'un événement.

b1) par l'arrivée d'un événement indésirable (détection par présence) : le dernier événement (message, transition d'état) du procédé est comparé avec l'ensemble des événements prévus par l'état précédent (figure 5.11).

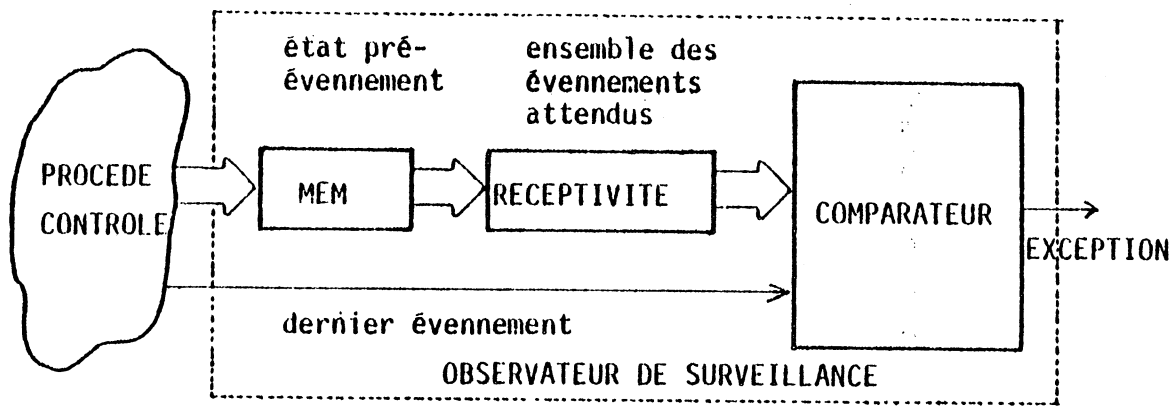


Figure 5.11. Détection d'une exception "par présence" d'un événement

b2) détection de la condition d'exception par l'absence d'un événement (figure 5.12) : l'état précédent de l'événement est mémorisé pendant l'intervalle de temps maximum présumé de cet état pour un fonctionnement correct du procédé. L'absence d'un certain événement jusqu'à la fin de cet intervalle signifie que l'état du procédé a changé en une configuration d'exception. Ce dispositif est le chien de garde (watch-dog).

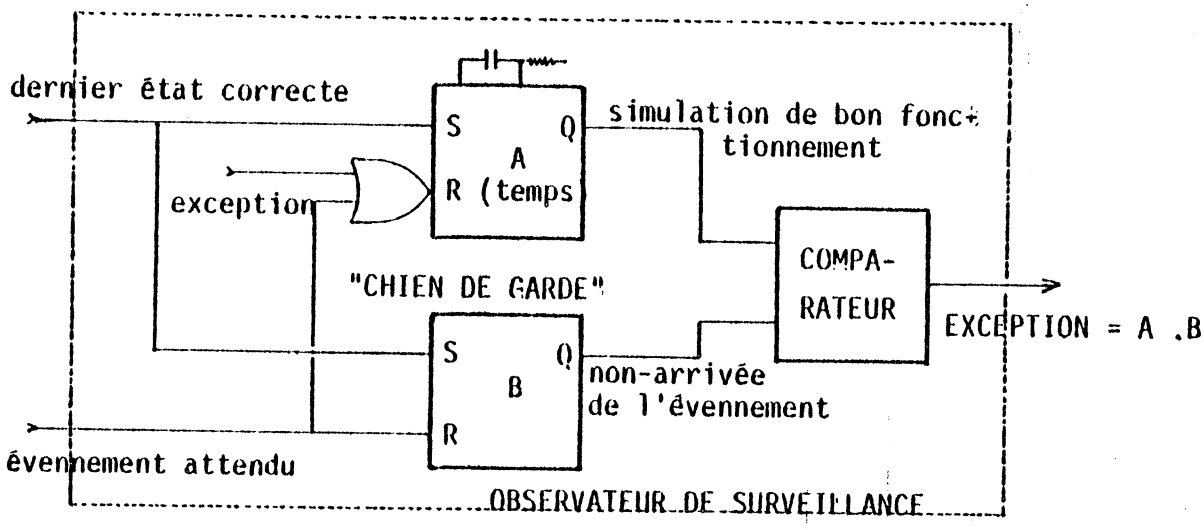


Figure 5.12. Détection d'une exception par l'absence d'un événement

CHAPITRE 6

SYNTAXE DU LANGAGE CSL

1. INTRODUCTION
2. STRUCTURATION DU CONTROLEUR
3. DECLARATION D'UN SERVICE
   - . SERVICES GENERAUX
     - . SECTION CS: CONNEXION DES SERVICES CONTROLES
     - . SECTION CF: SPECIFICATION DES FONCTIONS DU SERVICE
       - . OBSERVATEUR
       - . FONCTIONS COMBINATOIRES
   - . SERVICES INFIMAUX
     - . SPECIFICATION DES TRANSDUCTEURS
     - . LISTES DE LIAISON
   - . SERVICES MONITEURS
4. STRUCTURES DE DONNEES

ANNEXE: NOTATIONS UTILISEES DANS LA SYNTAXE

CHAPITRE 6 - SYNTAXE DU LANGAGE CSL

1. INTRODUCTION

Le langage CSL de spécification des contrôleurs de procédés discrets a été développé avec deux objectifs principaux (par ordre décroissant d'importance) :

- comme instrument d'application du formalisme proposé dans le chapitre 2 ; pour tester un formalisme sur un exemple réel, il faut en effet avoir un moyen d'expression bien défini ;
- comme proposition pour un outil de programmation des contrôleurs.

L'objet spécifié est le contrôleur ; la description du procédé contrôlé n'est pas considérée, bien qu'elle puisse être ajoutée par la suite, de façon à constituer l'objet d'un outil de vérification du système global.

Les effets d'un modèle de représentation sur un langage se manifestent plutôt dans les niveaux les plus élevés de sa structure. Par conséquent, la définition exhaustive des instructions permises ne présente pas beaucoup d'intérêt. Nous nous concentrerons donc plus particulièrement sur les aspects qui concernent plus directement le modèle proposé, en laissant ouvertes des options moins significatives jusqu'à une éventuelle implémentation du langage.

Par ailleurs, le niveau assez élevé du formalisme proposé impose des contraintes sur la forme de description utilisée. Ces contraintes sont positives dans la mesure où elles imposent une méthodologie de conception, mais négatives dans des situations où ces contraintes sont peu adaptées à la nature du système que l'on veut décrire. Cela n'est pas dû à un manque de généralité du modèle, mais à une description du système qui peut être éloignée des caractéristiques que nous voulons mettre en évidence par le formalisme. Tout en respectant le modèle, on doit donc faire quelques concessions à la facilité d'utilisation du langage. Une de ces facilités concerne la possibilité de profiter de l'énorme richesse de logiciel mathématique et de traitement de données écrit avec les langages procéduraux classiques.

2. STRUCTURE DU CONTROLEUR

La spécification du contrôleur d'un système complexe est rendue plus facile et plus souple si l'on décompose le problème en lui imposant une structure. Il existe deux critères fondamentaux de structuration :

. la structuration du comportement,

. la structuration spatiale.

La structuration du comportement définit l'organisation de l'évolution d'un système dans le temps, en identifiant les phases de ses opérations caractéristiques.

La structuration spatiale partitionne le système selon ses composants physiques. Bien que l'aspect comportement ne soit pas exclu (chaque composant physique réalise en effet un nombre réduit de tâches différentes), le modèle proposé du contrôleur se prête plutôt à une structuration spatiale.

Il y a trois manières principales, de plus en plus restrictives, de structurer topologiquement un système :

. de manière égalitaire (en un seul niveau),

. hiérarchiquement,

. de manière arborescente.

Cette structuration se manifeste sous la forme de relations entre les segments de la description d'une application.

Pour la structuration de la description CSL du contrôleur, nous avons retenu l'organisation arborescente, principalement en raison de la simplification qu'elle apporte.

Le contrôleur arborescent, structuré selon un critère spatial, peut être représenté comme un arbre inversé, dont les feuilles correspondent aux composantes du procédé contrôlé. Ces feuilles sont associées en sous-arbres de niveaux progressivement plus élevés, selon l'organisation des composantes du procédé physique. Au fur et à mesure que l'on monte dans la hiérarchie du contrôleur, une portion de plus en plus étendue du procédé est incluse, jusqu'à la racine de l'arbre, qui commande le système en entier.

2.1. Structure arborescente du contrôleur

Formellement, la structure arborescente d'un système de contrôle est obtenue par l'application de deux mécanismes :

a) partition horizontale :

en partant du modèle original, on peut décomposer le procédé contrôlé en un ensemble de sous-procédés (figure 6.1.), selon des critères géographiques, de modularité et de complexité (le contrôleur restant le même).

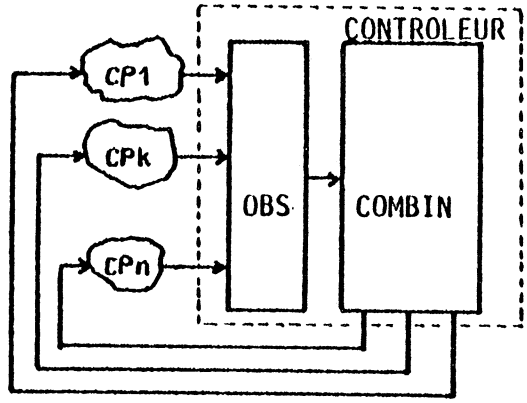


Figure 6.1. Partition horizontale (du procédé)

b) Hiérarchisation :

par une opération graphique, extrayons un des sous-procédés contrôlés (CPk, par exemple) de la surface de la page et plaçons-le par dessus le contrôleur (figure 6.2.).

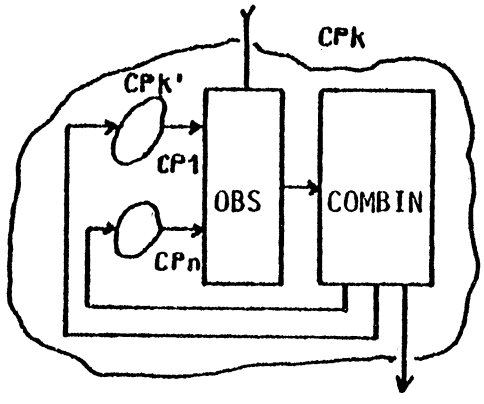


Figure 6.2. Hiérarchisation

PCk constitue maintenant l'environnement externe, ce qui revient à inclure explicitement l'environnement externe dans le procédé contrôlé. Le complément de CPK (CPk') peut maintenant être considéré comme un super-procédé, contrôlé par un contrôleur d'un niveau plus élevé, appartenant à un univers de contrôle plus étendu que l'original.

Par des applications successives de ces deux mécanismes, une hiérarchie arborescente peut être générée (fig. 6.3.).

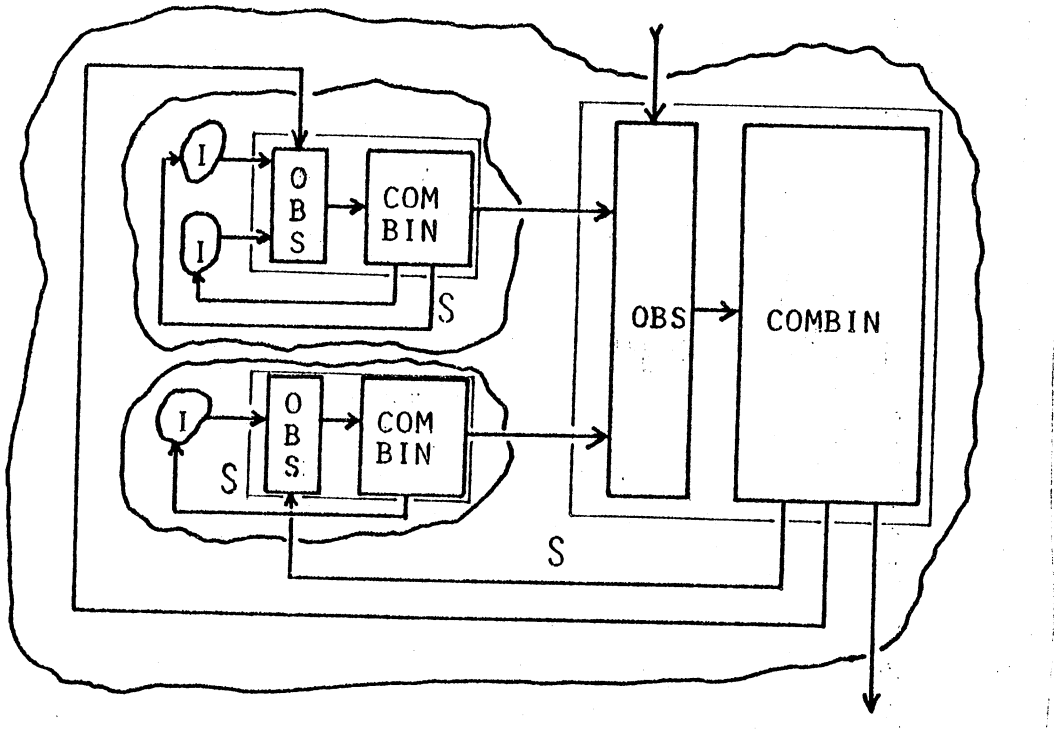


Figure 6.3. Structure du contrôleur

Nous introduisons ici une nouvelle notion, le "service", noté S, I, ou M, pour désigner un "procédé contrôlé de n'importe quel niveau". En particulier, un service de niveau zéro, ou infimal, noté I, représente le procédé contrôlé physique, ou un module matériel/logiciel indépendant. Le service moniteur, noté M, représente et gère une ressource partagée par plusieurs services-pères.

Le service est le module de base du langage. Chaque service correspond à un ensemble de fonctions de commande concernant une partie du procédé contrôlé. Comme dans MODULA [WIRT 77], un service peut être considéré comme une clôture autour des éléments (automates de l' observateur, fonctions combinatoires) qu'il contient. Tous les paramètres transmis entre les services sont déclarés.

Dans CSL, le parallélisme est implicite. A moins qu'un segment de programme ne soit déclaré explicitement séquentiel, toutes les instructions sont interprétées en parallèle. De même, les services sont par définition concurrents.

Les services constitutifs d'une description sont organisés selon une structure hiérarchique ; les services d'un niveau déterminé sont contrôlés par des services de niveau supérieur. Les services de niveau zéro, ou infimaux, correspondent au procédé contrôlé. Le niveau d'un service dans la hiérarchie reflète la portée de ses fonctions, c'est-à-dire l'importance de la partie concernée du procédé contrôlé. La hiérarchie est arborescente, c'est-à-dire que chaque service a un seul service père, à l'exception des ressources partagées. Les ressources partagées sont supposées faire partie du procédé contrôlé.

Exemple de ressource partageable : machines outil utilisées par plusieurs lignes de fabrication.

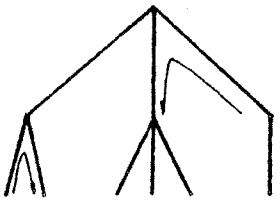
Les ressources partagées peuvent englober le matériel et le logiciel de gestion de la ressource technologique proprement dite.

La déclaration d'un service infimal comporte uniquement la mention de ses paramètres d'entrée et de sortie ; les fonctions du procédé contrôlé ne font pas partie de la déclaration.

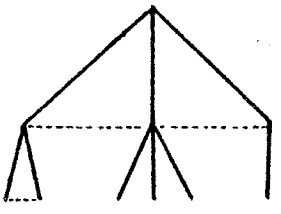
Le niveau d'un service peut être défini de façon générale comme la distance, en nombre de niveaux, qui le sépare du niveau zéro.

Dans une hiérarchie arborescente, chaque noeud/service ne communique qu'avec ses fils et avec son père. La communication directe entre frères ou cousins est interdite, elle s'effectue toujours par l'intermédiaire du père ou d'un ancêtre commun (figure 6.4.a.).

En fait, cette structure présente des avantages et des inconvénients par rapport à une hiérarchie non arborescente (figure 6.4.b.).Rappelons d'abord que nous nous trouvons au niveau de la spécification ; la différence entre les deux structures est donc de nature purement descriptive et la longueur du chemin n'a donc pas de signification physique [ROSS 77].



a) arborescente



b) non-arborescente

Figure 6.4. Types de hiérarchie

On pourrait objecter que la structure arborescente rend la description du chemin de la communication entre frères plus complexe. Cet effet existe; pourtant, la limitation simplifie beaucoup la description des autres liens. De plus, si la communication entre frères est assez fréquente, elle l'est beaucoup moins entre cousins, à cause de la structuration spaciale des contrôleurs, ce qui limite les effets négatifs de cette restriction.

Le niveau père sert donc à la commande et à la communication entre fils. Il est supposé que cela ne représente pas une limitation au niveau de la réalisation. Un système réparti peut être décrit de façon arborescente, en créant un service "chapéau" pour la communication entre composantes distribuées (figure 6.5.).

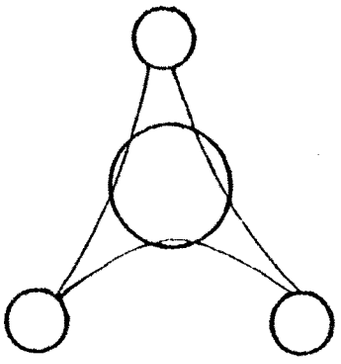


Figure 6.5. Communication par service "chapeau"

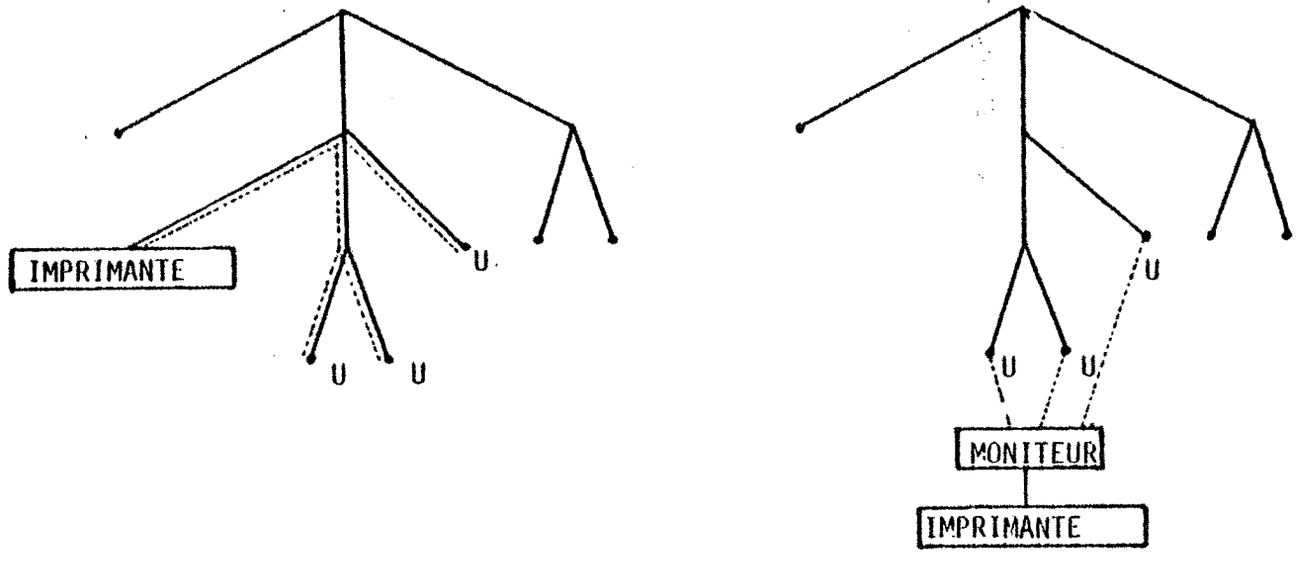
2.2. Le partage des ressources

Au niveau de la spécification, il y a un cas qui justifie une exception à l'organisation arborescente pure. Il s'agit du partage des ressources (peut-être des procédés physiques) entre différents services d'un système, ou même entre des systèmes différents.

Exemples :

- plusieurs lignes de production qui se partagent une machine outil,
- une imprimante chargée d'enregistrer les situations d'alarme dans une installation,
- un module chargé de plusieurs fonctions de supervision d'un procédé.

Il y a une solution, peu pratique d'ailleurs, qui consiste à placer la ressource comme fille du noeud hiérarchiquement commun à tous les utilisateurs, noeud qui sera chargé de la gestion de cette ressource (figure 6.6.a.).



(a) (b)

Figure 6.6. Gestion du partage d'une imprimante

Une autre solution, plus classique, consiste à utiliser un service du type "moniteur" [HOAR 74], gérant la ressource partagée. Le moniteur est fils de plusieurs pères, qui sont les utilisateurs de cette ressource (§ 4.3.1.). Cette structure correspond en fait à une hiérarchie semi-arborescente, c'est-à-dire un graphe orienté acyclique (figure 6.6.b.).

2.3. Portée des variables

La portée d'une variable, c'est-à-dire la région du programme limitée par sa validité, a des effets qui concernent principalement sa protection. Cette protection se manifeste par l'intermédiaire de la redondance introduite par la déclaration obligatoire des variables utilisées dans le contexte défini par sa portée. Ainsi, les variables d'un programme écrit en FORTRAN n'ont-elles aucune protection, puisqu'elles sont toutes globales. Les programmes écrits en ALGOL 60 sont structurés en blocs et la portée d'une variable est limitée au bloc dans lequel elle est déclarée : cette technique protège les variables d'un bloc interne contre son environnement, mais pas le contraire.

En CSL, nous avons choisi de suivre la méthode suggérée par [DIJK 76], en protégeant chaque niveau de ceux immédiatement supérieurs et inférieurs. Il n'y a donc pas de paramètres globaux. Cette protection est obtenue au prix de l'énumération explicite de tous les paramètres accessibles sur tous les interfaces entre services. Les avantages supplémentaires apportés par ce choix sont :

- une lisibilité accrue : les paramètres nécessaires à chaque service sont clairement définis et visibles,
- la possibilité de changer le nom des paramètres échangés (vis à vis de chaque service).

2.4. Déclaration et utilisation d'entités vectorielles

Il arrive souvent que des entités telles que les services, paramètres, automates observateurs, fonctions combinatoires, etc .. paraissent en exemplaires connexes et mutiples. La notation suivante peut alors être utilisée :

N <id. de l'entité> : déclare N exemplaires de l'entité qui sont identifiés par l'ordinal "i" ($1 \leq i \leq N$).

<id. de l'entité> (i) : représente l'exemplaire d'ordre i

Si i est remplacé par une astérisque * dans l'utilisation d'une entité vectorielle, ceci signifie que l'instruction est valable pour l'ensemble des entités définies.

Exemple :

6 CTR COMPTEURIMPULSIONS (INCR : IMPULSION(*), RESET : ZERO(*)) ;
signifie qu'il y a 6 compteurs, chacun étant incrémenté et mis à zéro par la source d'impulsions et le signal ZERO respectivement.
IMPULSION(1) et ZERO(1) agissent sur COMPTEURIMPULSIONS(1), etc ..

2.5. Notation des opérateurs

Le conventions suivantes sont observées :

exposant ' = inversion booléenne

+ = OU, addition

. = ET

EQ = égalité.

3. DECLARATION D'UN SERVICE

Dans ce chapitre on utilisera le mot "identificateur" pour désigner un type de service ou variable, et "nom" pour désigner une instance précise. Le "nom" est formé d'un identificateur suivi de clauses de particularisation (DE...) pour lever les ambiguités quand il y a plus d'une instance du même type dans un contexte.

```
<déclaration d'un service> ::= S <déclaration de service "général">|
                               I <déclaration de service infimal>|
                               M <déclaration de service moniteur>
```

3.1. Services généraux

Format de la déclaration :

```
S <identificateur du service> (<liste de paramètres formels>) ;
CS : <entier> <nom du service contrôlé>(<liste de paramètres effecitfs>)
     <entier> <nom du service contrôlé> (<liste de paramètres effectifs>)
     ...
CF :
     <fonction du contrôleur>
DIR : <liste des paramètres directement transmis>
FINS ;
```

où :

<liste de paramètres formels> = <liste de paramètres effecitfs> ::=

(D : <paramètres descendants> ; M : <paramètres montants>)

3.1.1. Section CS : connexion des services contrôlés

Les services contrôlés sont les fils (déclarés antérieurement) du service. La liaison avec le père se fait à travers la liste des services contrôlés de celui-ci.

La communication entre les services se fait par l'intermédiaire de paramètres. Les paramètres qui relient un service à son père sont mentionnés dans la liste de paramètres formels ; les paramètres qui relient un service à ses fils sont mentionnés dans les listes de paramètres effectifs (section CS). Pour établir la correspondance entre ces deux listes, on utilise la convention classique de position "formel/effectif".

Les listes de paramètres sont composées de paramètres "montants" qui sont transmis d'un service vers son père, et "descendants" qui descendent du père vers le fils.

Le passage des paramètres se fait "par nom", c'est-à-dire que la position du paramètre dans la liste indique une adresse commune aux services communicants.

Une déclaration de service définit un type de service. Une instance effective du service est créée à chaque fois qu'il est mentionné dans la section CS de la déclaration de son père. Si un identificateur de service est mentionné dans la section CS de plusieurs déclarations, alors autant d'instances sont créées.

L'ancêtre commun, c'est-à-dire la racine de l'arbre, définit non seulement un type, mais également une instance.

Il ne peut exister qu'un seul service-racine par description. Si la spécification intègre plusieurs sous-systèmes dont les seules relations sont, par exemple, le partage de ressources, alors un service père "chapeau" doit être déclaré dont les sous-systèmes sont les fils.

Exemple :

soit le système illustré par la figure 6.7. :

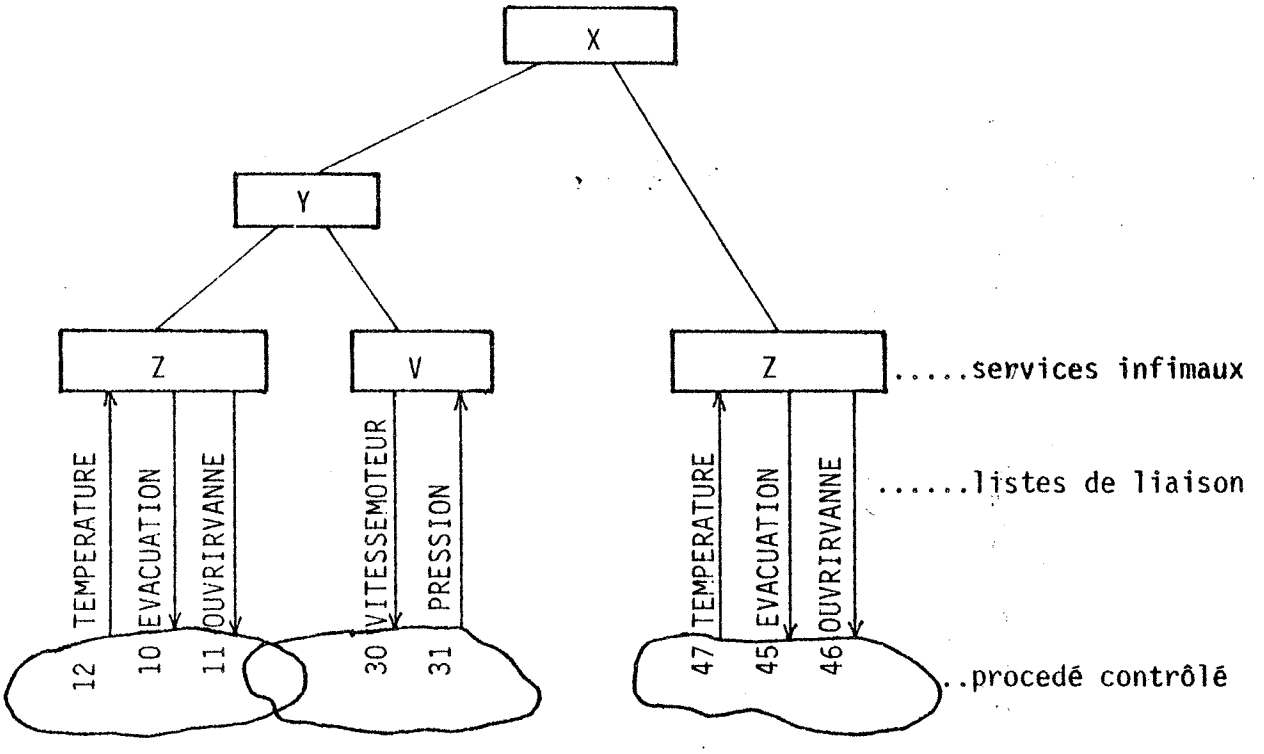


Figure 6.7. Structure du système

Extrait du programme :

```
I Z(D : OUVRIRVANNE, EVACUATION ; M : TEMPERATURE) ;
...
FINS ;
```

```
I V(D : VITESSEMOTEUR, M : PRESSION) ;
...
FINS ;
```

```
S Y( ...) ;
CS : Z(D :OUVRIR, EVAC  ; M : TEMP), V(D : VITESSE, M.: PRESSION) ;
...
FINS ;
```

```
S X
CS : Y(...), Z(...) ;
...
FINS ;
```

Par exemple, le service Z est déclaré une fois et existe en deux instances.

3.1.2. Section CF: spécification des fonctions du service

Cette section décrit l'observateur et la partie combinatoire du contrôleur. Puisque le langage est non procédural, l'ordre d'écriture de ces éléments est indifférent.

3.1.2.1. L'observateur

L'observateur est constitué essentiellement d'un ensemble d'automates d'états finis, qui représentent des composantes du vecteur d'état du procédé contrôlé ; il peut donc être spécifié sous la forme de graphes d'état, de réseaux de Petri, etc ..

En analysant des cas réels, on constate que très souvent on peut remplacer ces outils généraux par un ensemble réduit de fonctions très définies, qui peuvent être considérées comme des primitives :

- . BIT (bistable)
- . CTR (compteur)
- . RET (retard)
- . TMR (temporisateur)
- . GDE (chien de garde)
- . QUE (file)
- . automates synchrones (registres)
- . SEQ (graphe d'états général).

Activation de l'observateur

Les transmissions d'état de l'observateur sont activées par des messages constitués par le front montant de variables ou d'expressions booléennes.

INIT (initialiser) est un mot réservé qui indique une impulsion au démarrage du système.

Sous certaines conditions il peut arriver qu'un automate reçoive plusieurs messages (événements) en parallèle. Puisque par définition l'automate ne peut traiter qu'une transition d'état à la fois, un mécanisme de gestion des requêtes à l'automate devient nécessaire. On peut penser qu'un tel mécanisme contenu dans l'interpréteur, est chargé de gérer l'exclusion mutuelle et l'attente des événements dans un tampon dit "de courte échéance". Le temps d'attente des événements dans ce tampon est essentiellement fonction de la vitesse de l'interpréteur du programme de contrôle.

Par ailleurs, lorsque l'automate représente l'état de disponibilité d'une ressource partagée, une demande d'action sur la ressource correspond à une demande d'action sur l'automate, dont l'acceptation est éventuellement dépendante de son état. La demande (l'événement) doit alors attendre dans un tampon (une file) dit "de longue échéance".

Dans une spécification en CSL, les files de longue échéance doivent être programmées explicitement (dans le cas des moniteurs, une file de longue échéance est associée à chaque "condition"). Le temps d'attente dans une file de longue échéance est essentiellement fonction de la vitesse d'évolution du procédé contrôlé.

Lorsqu'il n'existe pas de file de longue échéance à la réception des événements activant un automate, ceux-ci ne sont pas mémorisés (sauf pendant un intervalle de temps "court" d'attente de l'interpréteur). Cela signifie que si l'automate est dans un état tel que l'événement ne provoque aucune transition, l'événement est "oublié". Le résultat d'une spécification manipulant des événements non mémorisés, est sa dépendance à la vitesse d'évolution du système. L'intérêt des événements non mémorisés est analysé dans [KEED 78].

Exemples :

- file de longue échéance : exemple d'utilisation du moniteur (à la fin de ce chapitre) ;
- événements non mémorisés : exemple du chariot ; la demande de cycle n'est pas mémorisée (§ 3.11).

Remarque : les différentes conditions de transition d'un automate ne sont pas nécessairement utilisées dans chaque application.

Les automates préprogrammés

a) Automate booléen (BIT)

Syntaxe :

BIT identificateur (S(et) : <cb> ; R(eset) : <cb> ; INV(erser) : <cb>) ;

où cb = condition booléenne

INV(erser): ... provoque une inversion de l'état de l'automate.

Si INIT n'impose pas l'état "vrai", alors l'automate est, par défaut, "faux" au démarrage.

Exemple :

BIT MACHINETOURNE (S : ON ; R : OFF) ;
BIT LAMPEALLUMEE (INV : INTERRUPTEUR) ;

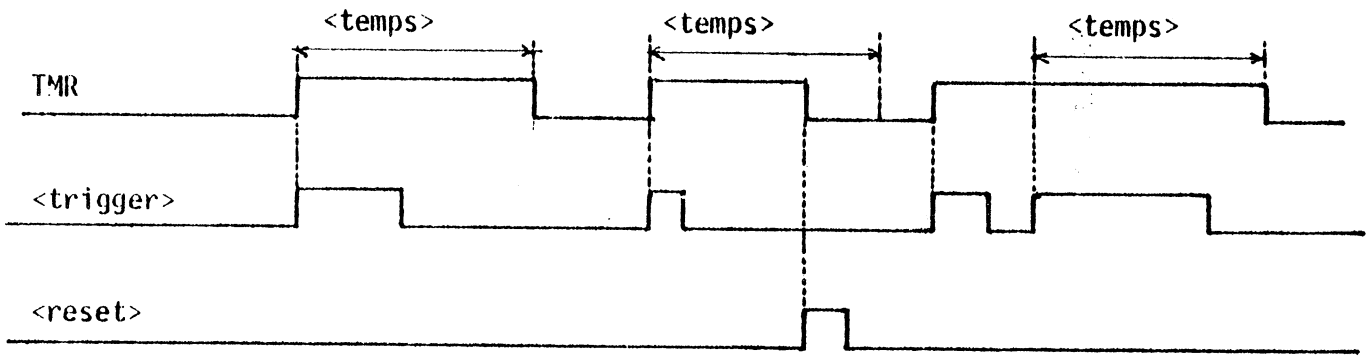
b) Temporisateur (TMR)

C'est un monostable réenclenchable, automatiquement mis à zéro au démarrage.

Syntaxe :

TMR identificateur (S : <trigger> ; T : <temps> ; R : <reset>) ;

Chronogramme :



Exemple :

L'état d'attention de l'auditeur d'une conférence peut être représenté par :

TMR ATTENTION (S : DEBUTCONFERENCE ; T : 10 minutes) ;

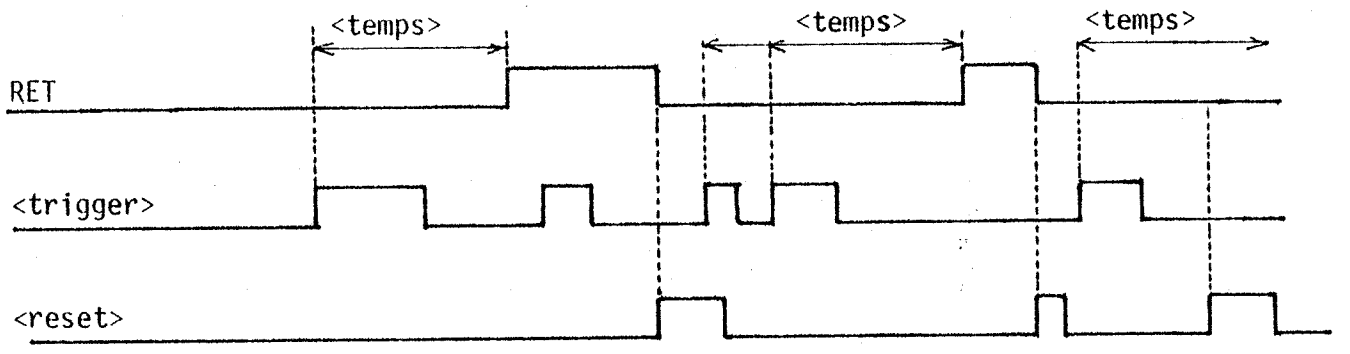
c) Retard (RET)

C'est également un monostable réenclenchable, mis à zéro au démarrage. Le retard devient "vrai" <temps> après <trigger> ; il redevient "faux" au <reset>.

Syntaxe :

```
RET <identificateur> (S : <trigger> ; T : <temps> ; R : <reset>) ;
```

Chronogramme :



Exemple :

encore l'état de l'auditeur :

RET SOMMEILPROFOND (S : DEBUTCONFERENCE ; T : 15 minutes ; R : FINCONFERE

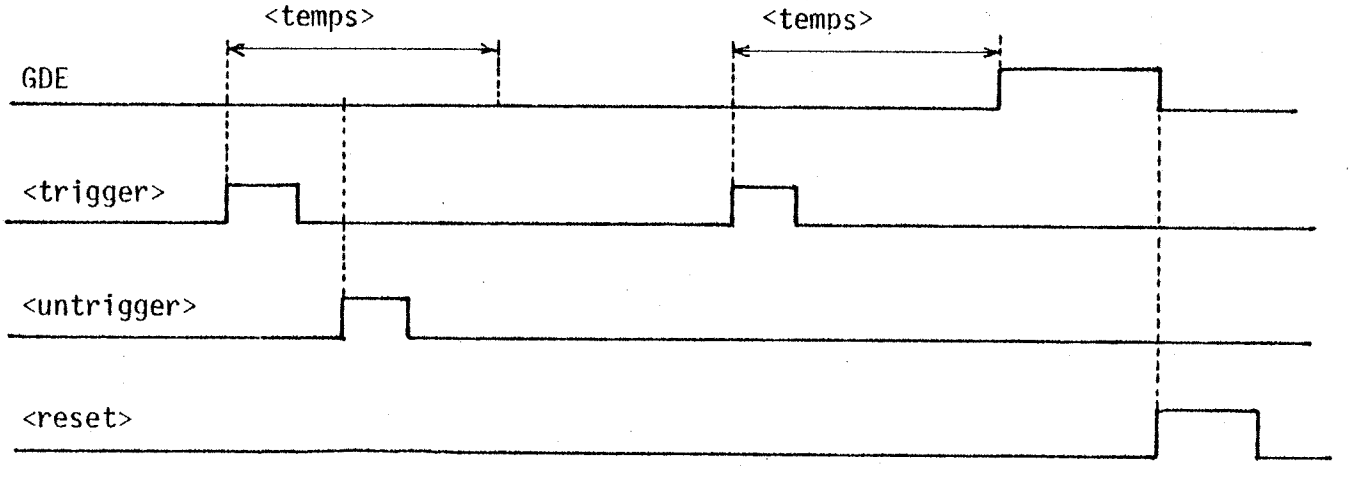
d) Chien de garde (GDE)

Syntaxe :

```
GDE identificateur (S : <trigger>, T : <temps>; A(ttendu) : <untrigger>;
                    R : <reset>) ;
```

Le chien de garde devient vrai si le front montant de untrigger n'arrive pas dans la période <temps> qui succède le front montant de trigger. Il retourne à l'état faux au front montant de <reset>. Le chien de garde est réenclanchable et automatiquement mis à zéro au démarrage.

Chronogramme :



Exemples :

1) GDE ACCIDENTAVION (S : DEPART ; T : DUREEMAXVOYAGE ; A : ARRIVEE ; R : ALARMERECUE) ;
2) Détecteur de fréquence minimale : si <trigger> = <untrigger>, alors le chien de garde signale que la fréquence de <trigger> est tombée en dessous de 1/<temps>
   GDE SYNCOPE (S : BATTEMENT ; T : 5 minutes ; A : BATTEMENT ; R : DOCTEURAPPELE) ;

e) Compteur (CTR)

Syntaxe :

CTR identificateur (I : <incrémenter>; D : <décrémenter>; R : <reset>) ;

Le compteur est incrémenté, décrémenté et mis à zéro par le front montant correspondant. Par défaut, sa valeur au démarrage est zéro. Le compteur peut aussi avoir plusieurs valeurs de "preset" qui lui sont attribuées par une instruction : voir (g) "automates synchrones".

/<condition>/ <id.compteur> :=<valeur preset> ;

Exemples :

CTR TOTALISEURTELEPHONE (I : IMPULSION ; R : REDEVANCEPAYEE) ;
CTR TAMPON (I : DEPOSER ; D : RETIRER ; R : POP) ;

f) File (QUE)

c'est-à-dire FIFO (first in first out).

Syntaxe :

QUE <idfile> (LONGUEUR : <numelements>) ;

Opérations :

. ranger un élément:réalisé par la simple affectation (:=)
  /<cond.bool.>/ <idfile> := ... ;
. lire le sommet : ... := SOMMET <idfile>:<idfile>est attribué à un registre ou comparé à une valeur.
. PRElever l'élément du sommet de la file, ce qui la fait avancer
  /<cond.bool>/ PRE(<idfile>);

g) Automates synchrones (registres)

```
/ <cond.bool> / <nom> := <contenu>;
```

Sémantique :

Le <nom> désigne un registre simple ou composé ; le <contenu> est chargé dans <nom> sur le front montant de <cond.bool>.
Certains automates peuvent subir des transitions synchrones et asynchrones. Le compteur, par exemple, est chargé en mode synchrone et incrémenté, décrémenté et mis à zéro en mode asynchrone.

Exemple :

```
/ CHARGER / TAMPON := DERNIEROCTET ;
```

L'étiquette /<cond.bool.>/ est aussi utilisée pour charger des registre externes au service. Exemple : pour imprimer un message sur un périphérique :

```
/SITUATIONALARME/ IMPRIMER ("DANGEREXPLOSION") ;
```

h) Les automates généraux (SEQ)

Dans le chapitre 5 on a analysé la notion d'observateur dans le contexte de la microprogrammation. On propose ici un outil syntaxique qui supporte une approche du type microprogrammation dans CSL. L'observateur général, qui ne correspond à aucune des formes préprogrammées, décrit un graphe d'états.

Chaque automate est constitué d'une séquence de phases, ordonnées de manière procédurale : l'ordre lexicographique correspond à l'évolution du procédé contrôlé. Chaque phase est associée à un identificateur booléen utilisé comme opérande par les fonctions combinatoires de la description.

La récursivité (utilisation d'un automate dans une séquence décrivant ce même automate) n'existe pas dans CSL. En effet, cela correspondrait à sauvegarder le contexte (c'est-à-dire l'état du procédé) dans une pile et de réutiliser le procédé, ce qui est difficile à imaginer dans la pratique.

Syntaxe :

```
SEQ <id. de séquence>
INIT : <phase>
[<phase>]+
FINSEQ ;
```

où :

<phase> ::= [<étiquette> :] <instruction ou appel de procédure> [ALLERA <prochaine phase ou étiquette>] ;|
<id.phase> <clause de transition>

<clause de transition> ::= CAS [<condition de transition> ;]$^{+}$ FINCAS;
<condition de transition>

<condition de transition> ::= JUSQUA <expr.bool.> [ALLERA <prochaine phase ou étiquette>];|
PENDANT <temps> [ALLERA <prochaine phase ou étiquette>];

La fin d'une phase est indiquée par la FIN de l'exécution d'une instruction ou procédure appelée, ou par une expression booléenne explicite (JUSQUA ...), ou encore par l'écoulement d'une temporisation (PENDANT ...).

Lors de l'initialisation ("power on") l'automate se met dans la phase indiquée par INIT. Le passage entre deux phases suit l'ordre lexicographique, sauf si un saut est expliciterment indiqué (ALLERA ...). Par ailleurs, l'automate est circulaire, c'est-à-dire qu'après l'interprétation de la phase qui précède FINSEQ, l'automate retourne à la première phase, sauf indication explicite de saut.

3.1.2.2. Les fonctions combinatoires

Les fonctions combinatoires génèrent les transitions de l'état de l'observateur, les commandes du procédé contrôlé et les compte-rendus pour le contrôleur du niveau supérieur dans la hiérarchie.

Sauf quand la séquentialité est explicitement indiquée, toutes les fonctions combinatoires sont en évaluation permanente et parallèle. Par conséquent, plusieurs formes syntaxiques habituellement utilisées, instructions séquentielles (activées puis exécutées), acquièrent une signification différente, bien que leur contenu logique soit le même.

On utilisera le symbole "=" pour indiquer l'évaluation permanente (définition) et le symbole ":=" pour indiquer l'affectation (chargement de variables mémorisées).

Principales instructions combinatoires :

a) définition inconditionnelle ;

b) définition conditionnelle simple :

<var.attribuée> = SI <condition> ALORS <valeur> SINON <valeur> ;

c) définition conditionnelle multiple :

```
var.attribuée  = CAS
                 SI <condition> ALORS <valeur> ;
                 SI <condition> ALORS <valeur> ;
                 ...
                 SINON <valeur>
                 FINCAS ;
```

Exemple :

```
ACCELERATIONMOTEUR = CAS
                     SI VITESSE < REFERENCE ALORS POSITIVE ;
                     SI VITESSE > REFERENCE ALORS NEGATIVE ;
                     SINON NULLE
                     FINCAS ;
```

d) Définition vectorielle

```
POUR <clause paramètres> FAIRE <instruction ou liste d'instructions>
 <clause paramètres> ::= <liste de valeurs>|
                         <valeur initiale> PAS <pas> JUSQUA <valeur finale>
                         <valeur initiale> JUSQUA <valeur finale>
```

La définition vectorielle dans un contexte combinatoire ne peut pas avoir un caractère itératif. On remarque que très souvent l'instruction d'itération POUR ... dans un contexte séquentiel n'a qu'une fonction de répétition vectorielle. Par ailleurs, on peut utiliser le mécanisme proposé dans le §2.4. ci-dessus en tant que cas particulier de la définition vectorielle.

```
6 CHAUFFER [*] = ETTATEL. (TEMPER [*]<VALMAX[*]) ;
```

équivaut à

```
POUR I := 1 PAS 1 JUSQUA 6
CHAUFFER [I] = ETATEL. (TEMPER[I] < VALMAX[I]) ;
```

Les fonctions combinatoires peuvent être décrites par des

- expressions algébriques, telles que celles décrites ci-dessus,
- tableaux,
- procédures/fonctions algorithmiques

Ainsi que les expressions algébriques et les tableaux, les fonctions algorithmiques sont en évaluation virtuellement permanente. Elles représentent un moyen d'exprimer un calcul complexe d'une forme souvent plus compréhensible et facile à écrire que les expressions non-procédurales. De plus, la possibilité d'exprimer des algorithmes de façon "classique" permet de profiter des algorithmes déjà étudiés.

Syntaxe :

```
ALG <nom de la procédure/fonction> ( <paramètres>)
  algorithme séquentiel
FINALG ;
```

Un exemple typique serait l'évaluation d'une racine carrée, dans un environnement où cette opération n'est pas primitive ;

```
ALG RACCARREE (D : ENTREE)  comm. méthode de Newton
  Y := 2 ;
  POUR Y > Z OU Z - Y > 1
  FAIRE DEBUT Z := ENTREE/2 ; Y := (Z + Y)/2 FIN ;
  RACCARREE := Y ;
FINALG ;
```

La même fonction pourrait être calculée sous une forme purement combinatoire, par exemple avec une série.

Les procédures algorithmiques sont connectées en permanence aux opérandes. Au niveau d'une spécification CSL, il n'y a pas de procédures partagées, il n'existe donc pas non plus de notion d'appel de procédure algorithmique.

Opérateur d'impulsion

L'opérateur d'impulsion génère une impulsion unitaire sur le front montant de l'expression booléenne opérande.

Syntaxe :

un point d'exclamation à droite de l'expression booléenne.
<expr.bool.> ! = (NON <expr.bool> (t - ε). <expr.bool> (t + ε)

Exemple :

RELAIAUTOVERROUILLE = FERMER !

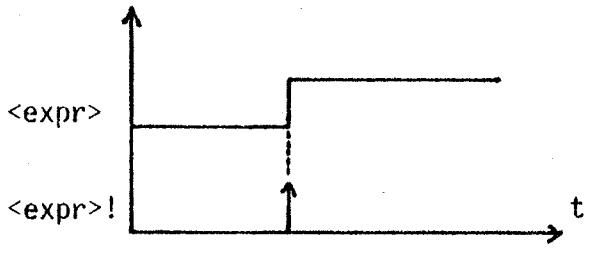


Figure 6.8. L'opérateur "impulsion"

3.1.3. Section de transmission directe (DIR)

Cette section de déclaration d'un service spécifie les paramètres transmis par ce service entre son père et un fils ou entre deux fils, sans nécessairement être mémorisé ou nécessiter un traitement mathématique quelconque.

Les paramètres DIR pourraient être reliés directement entre les services concernés (le générateur et l'utilisateur du paramètre), si l'on faisait fi de la structure arborescente imposée par le formalisme, qui empêche les connexions directes entre services non adjacents.

Cette restriction porte uniquement sur le niveau de la description et la section DIR indique en réalité les signaux qui peuvent, au niveau de la réalisation, être directement "câblés", dans un sens matériel ou logiciel, entre les services concernés, sans toucher directement le service considéré, qui se situe "sur le chemin" au sens de l'arborescence.

Exemple :

les paramètres A, B et C traversent le service Si (figure 6.9). Ils seront cités dans la section DIR de Si comme
DIR : A, B_2 = B_1, C ;

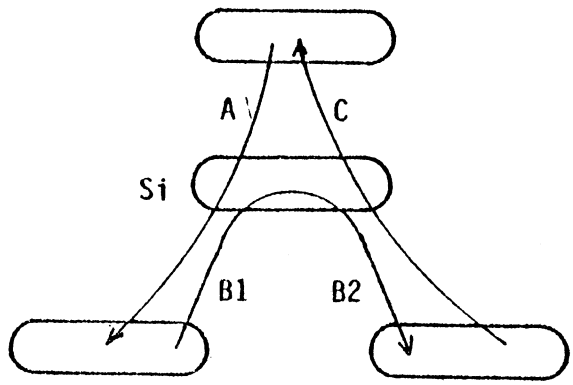


Figure 6.9. Transmission directe de paramètres

3.2. Services infimaux

Un service infimal peut représenter :

a) soit un élément du procédé contrôlé physique,

b) soit un module indépendant, c'est-à-dire un processus compilé ou considéré séparément [TURN 80].

Format de la déclaration :

```
I <identificateur du service> (<liste de paramètres formels>) ;
<spécification des transducteurs (capteurs et actionneurs)>
FINS ;
```

3.2.1. Spécification des transducteurs

Cette partie sert à spécifier la nature des variables d'entrée-sortie et transforme les variables physiques (VP) telles que "plus haut que", "bouton poussoir actionné", "température", etc .. en variables internes du contrôleur (VC).

Syntaxe de la spécification de paramètres :

```
<type de variable> <id.variable> {<liste de valeurs>} ;
<type de variable> ::= REEL | ENTIER | DISCRET | BOOLEEN | IMPULSION | CHAINE ;
```

Les VP sont transformées en VC selon les règles suivantes :

1. REEL

Les VC réels sont généralement issues de VP analogiques. Cette transformation et son inverse, se font souvent par des procédures de conversion et de conditionnement assez complexes, qui échappent au but de ce travail. Dans un contexte plus détaillé, les notions de gain, filtrage, seuil, bande passante, seraient prises en compte.
Pour le moment, les VC seront considérées simplement selon leur valeur physique nominale. Ainsi, une température de 280,5° C par exemple, sera-t-elle lue par un thermocouple, un linéariseur et un convertisseur analogique-numérique, comme un CV TEMP = 280,5, sans prendre en compte les transformations intermédiaires.

2. ENTIER

Les VP reçus sont directement transformés en leurs VC équivalents et vice versa.

3. DISCRET

Les variables discrètes admettent un nombre fini de valeurs, sans équivalence évidente avec des entiers. La liste des valeurs, dans ce cas, est constituée d'une suite de "n" valeurs.
Exemple :

> un aiguillage de chemin de fer à trois positions peut recevoir des commandes qui lui indiquent de tourner à gauche, en position médiane et à droite. La spécification sera alors :
> DISCR AIGUILLAGE(GAUCHE, MEDIANE, DROITE);

4. BOOLEEN

La liste des valeurs est constituée de un ou deux éléments. Le premier correspond à l'état physique de la VP qui rend la VC vraie et le deuxième, optionnel, à celui qui la rend fausse.

Exemple :

```
BOOLEEN MOTEUR (MARCHE, ARRET), PHOTOINTERRUPTEUR (OCCUPE) ;
```

5. IMPULSION

Il n'y a pas de liste de variables. Une impulsion VP en génère une VC.

Exemple :

```
IMPULSION PUSHBUTTONDIFFERENTIATEUR ;
```

6. CHAINE

Une adresse de ce type indique l'adresse symbolique du canal qui transmet une chaîne de caractères. Il n'y a pas de liste de variables.

Exemple :

```
CHAINE MESSAGE ;
/DEBORDEMENT/ IMPRIMER(MESSAGE) ;
```

3.2.2. Listes de liaison

Pour pouvoir utiliser CSL en tant que langage de programmation, il faut lier les identificateurs des paramètres, qui sont symboliques, aux adresses d'entrée-sortie et d'interruption effectives de l'interface procédé-contrôleur.

Dans les ordinateurs du commerce, le nombre de dispositifs périphériques est relativement réduit. De plus, l'exemplaire précis de l'imprimante sur laquelle s'effectue une sortie, par exemple, n'a généralement pas beaucoup d'importance pour l'usager. Par conséquent, les adresses effectives des périphériques sont prises en compte pendant la phase de génération du système d'exploitation et l'usager spécifie seulement le type de dispositif dont il a besoin.

Dans les systèmes de contrôle de processus, l'usager spécifie le dispositif précis auquel il désire s'adresser. De plus, les entrées-sorties peuvent exister en grand nombre, disons quelques milliers dans une grande installation. Par conséquent, il faut trouver une méthode pratique d'affectation des adresses à ces périphériques nombreux.

Une solution immédiate consiste à spécifier les adresses effectives au lieu des adresses symboliques. Cette solution a été adoptée pour les automates programmables, l'inconvénient principal étant l'incompatibilité avec le concept de déclaration de types, essentiel dans un langage de haut niveau.

Les paramètres des services sont liés aux dispositifs contrôlés par l'intermédiaire des listes de liaison. Une liste de liaison est une liste d'adresses effectives établie pour chaque service infimal.

Syntaxe :

LINK (<nom du service>) := liste des adresses des capteurs/actionneurs ;

Remarques :

- dans quelques exemplaires d'un type de service, il peut exister des capteurs/actionneur manquants, mais déclarés, qui ne sont ni utilisés ni liés à une adresse physique. Dans la liste de liaison, ces transducteurs sont représentés par un caractère "vide", c'est-à-dire deux virgules consécutives.

Exemple :

soit le système de la figure 6.7. Il y a trois services infimaux qui son liés au procédé contrôlé.

Liste de liaison :

```
LINK (Z DE Y) = 11, 10, 12 ;
LINK (V) = 30, 31 ;
LINK (Z DE X) = 46, 45, 47 ;
```

3.2.3. Pupitres de commande

Les pupitres de commande (§ 3.5) constituent un cas particulier des services infimaux.

Selon la convention adoptée, un dispositif de sortie, tel qu'un message pour une imprimante ou un indicateur lumineux, sont considérés comme des paramètres "D", puisqu'ils descendent dans l'arborescence, du contrôleur vers le pupitre de l'opérateur, considéré comme partie du procédé contrôlé, et par exemple un bouton poussoir sera considéré comme un paramètre "M".(montant).

3.3. Les services moniteurs

Les services moniteurs gèrent des ressources partagées dynamiquement par plusieurs services pères. Pour des raisons analysées au § 4.1.3, la communication avec ce type de service se fait exclusivement par des messages. Le service moniteur a la syntaxe suivante :

```
M <identificateur du moniteur> (<liste de procédures et paramètres>) ;
CS : <services contrôlés>
CF :
     déclaration et initialisation des variables locales
     déclaration des conditions
     procédures du moniteur
FINS ;
```

Les variables locales représentent l'état de disponibilité de la ressource proprement dite. On associe aux conditions des files d'attente de longue échéance qui représentent "l'état de besoin" des utilisateurs. Il y a une file par condition. Chaque condition admet deux opérations : "wait" et "signal". L'exécution de "wait.condition" met l'utilisateur qui a provoqué cette exécution dans la file.condition et le suspend. L'exécution de "signal.condition" retire un utilisateur suspendu de la file.condition et le remet en exécution. Les appels aux procédures du moniteur à partir des utilisateurs qui ont le format

<id.moniteur> . <nom procédure>

constituent la communication dans le sens utilisateur → ressource. La réponse ressource → utilisateur se fait par l'intermédiaire des opérations wait et signal.

Les files d'attente peuvent aussi être déclarées et gérées explicitement, lorsqu'on souhaite par exemple rendre le comportement du système dépendant des vitesses de ses composantes (événements non mémorisés, § 6.3.1.2.1.). Le moniteur qui gère le partage de la ressource IMPRIMANTE de l'application présentée en annexe, est conçu de façon à ce que le procédé industriel ne s'arrête pas même si le tampon file de message pour l'imprimante est plein.

Connexion des moniteurs dans la structure d'une description CSL

Un service moniteur est partagé entre plusieurs pères (les utilisateurs). Il faut donc expliciter les services qui se partagent le moniteur. La commande PART(ager) remplit cette fonction. Une déclaration de moniteur spécifie un type. Chaque exemplaire du moniteur est créé par la commande PART.

Syntaxe :

```
PART {<id.moniteur> (<liste des services utilisateurs>),}*;
```

Exemple :

Soit les quatre configurations suivantes de contrôleurs, dont on décrit la structure en CSL. Des exemplaires du moniteur H sont partagés de différentes manières :

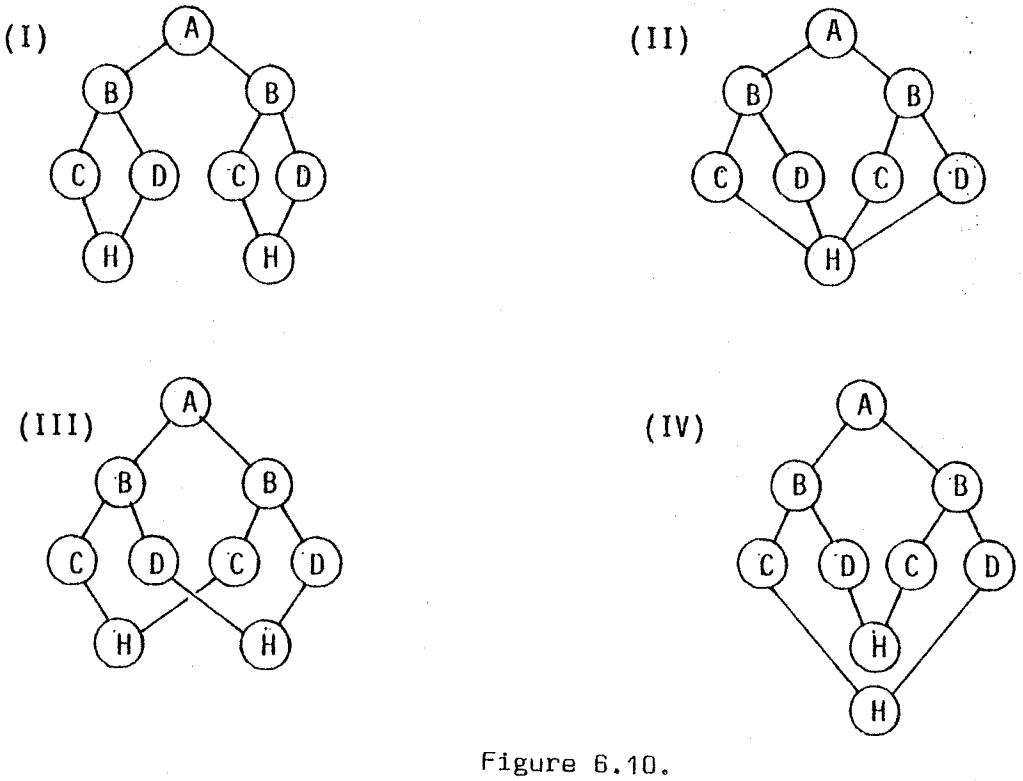


Figure 6.10.

Description commune des quatre structures :

```
S A ;
SC : 2 B ;
...
FINS ;
```

```
S B ;
SC : C, D ;
...
FINS ;
```

```
S C ;
SC : H ;
...
FINS ;
```

```
S D ;
SC : H ;
...
FINS ;
```

```
M H ;
...
FINS ;
```

Spécification du partage des moniteurs:

(I) PART 2H (C de B [*], D de B [*]) ;

(II) PART H (C de B [1], D de B [1]), C de B [2], D de B [2]) ;

(III) PART H (C de B [1], C de B [2]), H (D de B [1], D de B [2]) ;

(IV) PART H (C de B [1], D de B [2]), H (C de B [2], D de B [1]) ;

Exemple :

On veut décrire le contrôleur d'un système qui commande un trafic ferroviaire à travers un tunnel à une seule voie qui peut être empruntée par des trains arrivant dans les deux sens (figure 6.11).

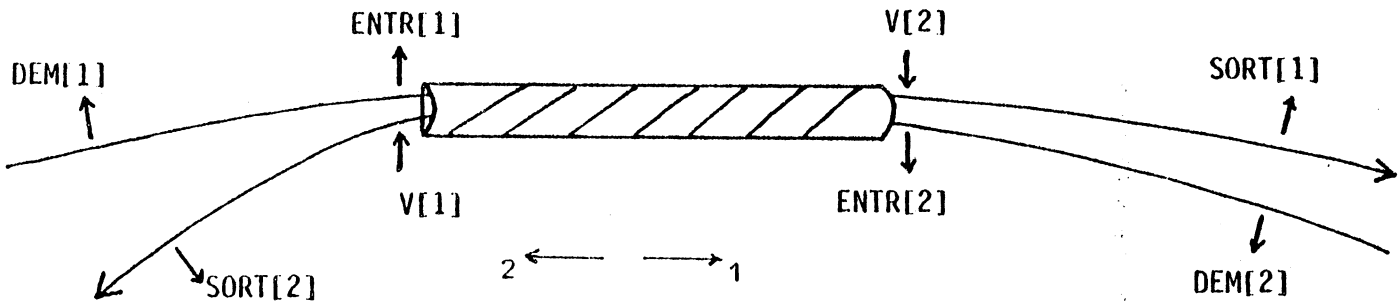


Figure 6.11. Tunnel à deux voies

Entrée du système :

DEM [1], DEM [2] : signaux indiquant qu'un train s'approche du tunnel respectivement dans le sens 1 ou 2 ;

ENTR [1], ENTR [2] : signaux indiquant que la tête du train vient d'entrer dans le tunnel respectivement dans le sens 1 ou 2 ;

SORT [1], SORT [2] : signaux indiquant qu'un train vient de traverser le tunnel respectivement dans le sens 1 ou 2.

Les sorties du contrôleur V [1], V [2], sont les feux autorisant l'accès au tunn
On veut que les sorties soient telles que V [1] = FAUX et V [2] = FAUX s'il n'y a pas de train qui traverse le tunnel ni de train en attente. La hiérarchie de contrôle de ce système peut être représentée selon la structure de la figure 6.12.

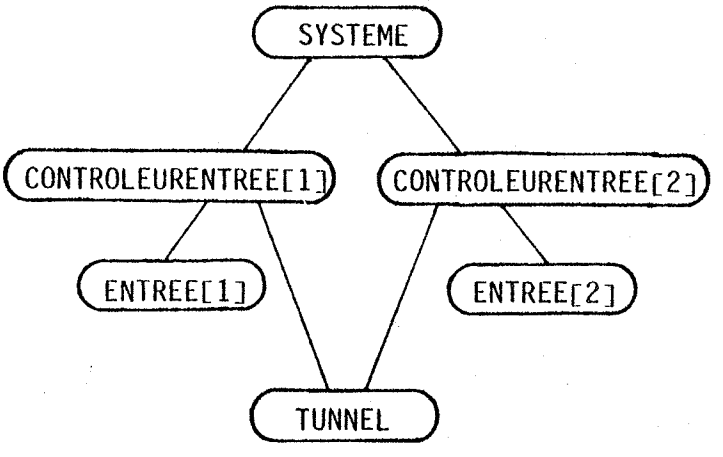


Figure 6.12. Structure du contrôleur

Programme de spécification du contrôleur :

```
I ENTREE (D : V ; M : DEM, ENTR, SORT) ;
BOOL V ; IMP DEM, ENTR, SORT ;
FINS ;
```

```
S CONTROLEURENTREE
CS : ENTREE (D : V ; M : DEM, ENTR, SORT) ;
     TUNNEL (D : ENTRER, SORTIR)
CF :
     SEQ
     INIT : ENTREELIBRE JUSQUA DEM;
            TUNNEL.ENTRER ;
            ENTREEPERMISE JUSQUA ENTR ;
     FINSEQ ;
V = ENTREEPERMISE ;
TUNNEL.SORTIR = SORT ;
FINS ;
```

```
M TUNNEL (ENTRER, SORTIR) ;
BOOL OCCUPE := FAUX ;          & valeur initiale
CONDITION : LIBRE ;
PROCEDURE ENTRER ;
   DEBUT SI OCCUPE ALORS LIBRE.WAIT ;
   OCCUPE := VRAI ;
   FIN ;
PROCEDURE SORTIR ;
   DEBUT OCCUPE := FAUX ;
   LIBRE.SIGNAL ;
   FIN ;
FINS ;
```

```
S SYSTEME ;
CS : 2 CONTROLEURENTREE ;
FINS ;
```

```
PART TUNNEL (CONTROLEURENTREE [1], CONTROLEURENTREE [2]) ;
```

Remarque :

si, par exemple et pour une raison non précisée, on avait voulu que les demandes d'ENTRER soient non mémorisées, on aurait programmé la procédure correspondante du moniteur de la façon suivante :

```
PROCEDURE ENTRER ;
DEBUT SI NON OCCUPE ALORS OCCUPE := VRAI ;
FIN ;
```

4. STRUCTURES DE DONNEES

Pour déclarer et accéder aux structures de données, on adopte une méthode basée sur le langage ALGOL 68 [ALGO 73].

a) Syntaxe de la déclaration

déclaration de structure ::= STRUCT <id.structure> (<liste de composants>) ;
<liste de composants> ::= [{<dimension>} <champ>]$^{+}$
<champ> ::= <déclaration de structure> | <id.structure> | <id.élément>

La règle de remplacement <champ> ::= <id.structure> s'applique quand <id.structu identifie une structure déclarée ailleurs. Les <id.élément> correspondent aux feuilles de l'arborescence formée par une structure.

Exemple :

description du stock d'un magasin ayant un maximum de 500 articles, existant en un maximum de 7 tailles. Les articles sont listés dans le catalogue selon un numéro d'article (de 1 à 500) et taille (de 1 à 7)

Déclaration :

STRUCT ARTICLE (NOM, 7 TAILLE) ;
STRUCT TAILLE (LONGUEUR, LARGEUR, PRIX) ;
STRUCT MAGASIN (500 ARTICLE) ;

b) Syntaxe d'accès

variable accédée ::= <id.élément> (<position dans vecteur>)
{DE <id.structure> (<position dans vecteur>)}*

Un élément de la structure est identifié en descendant l'arborescence à partir des feuillez et en spécifiant les structures intermédiaires (DE ...), uniquement quand cela est nécessaire pour éviter les ambiguïtés.

Exemple :

NOM DE ARTICLE (A) ;
LONGUEUR DE TAILLE (T) DE ARTICLE (A) ;
PRIX DE TAILLE (T) DE ARTICLE (A) ;

5. ANNEXE : NOTATIONS UTILISEES DANS LA SYNTAXE

< >	encadre chaque notion figurant dans une règle
::=	désigne un remplacement
la simple juxtaposition	représente "suivi de"
|	signifie ou bien
exposant $^{+}$	signifie répété un nombre non nul de fois
exposant *	signifie répété un nombre quelconque de fois (éventuellement nul)
[ ]	utilisé pour la mise en facteur
{ }	signifie facultatif

CHAPITRE 7

PROPOSITION D'UN INTERPRETEUR CSL

1. ASPECTS TEMPORELS
2. REALISATION: LES CALCULATEURS DIRIGES PAR LES DONNEES

CHAPITRE 7 - PROPOSITION D'UN INTERPRETEUR CSL

Ce chapitre est divisé en deux parties : dans la première, on analyse les aspects temporels liés à la réalisation de contrôleurs de procédés temps réel ; dans la deuxième, on propose une méthode de réalisation de l'interpréteur d'une description CSL.

Les caractéristiques physiques du matériel de traitement et de transmission de données imposent des retards non spécifiés lors de la description du contrôleur. Les effets de ces retards peuvent être gênants lorsqu'ils introduisent des "erreurs" dans l'information temporelle portée par un signal. On analyse les différentes techniques dont on dispose pour rendre la fonction du contrôleur aussi indépendante que possible de ces retards.

La deuxième partie de ce chapitre est consacrée à la proposition d'une méthode d'interprétation de CSL. Le parallélisme intrinsèque de CSL le rend bien adapté à être interprété par un mécanisme dirigé par les données, mais par ailleurs il est particulièrement sensible à l'occurence d'aléas dûs à la non spécification de l'ordre d'évaluation des expressions de la description (langage non procédural). Nous avons donc cherché à modifier la technique d'interprétation dirigée par les données, de façon à éviter les aléas.

1. ASPECTS TEMPORELS

D'une manière générale, lors de la définition d'un modèle, on fait implicitement quelques hypothèses simplificatrices, qui permettent de faire abstraction d'un ensemble de phénomènes de second ordre. Ces phénomènes peuvent en effet être négligés en prenant des précautions.

L'information associée à un signal est composée de deux aspects : son amplitude (information de valeur) et la période de sa validité (information temporelle [NOGU 75].

1.1. La composante temporelle de l'information

Sous certaines réserves, l'information temporelle du signal définit un instant (un événement) dont l'occurence est elle-même porteuse de sens.

Dans les systèmes discrets, l'information de valeur d'un signal ne présente pas essentiellement d'incertitude dûe au rapport signal/bruit qui peut généralement être aussi élevé que l'on veut.

Par contre, l'incertitude sur l'information temporelle de type événement, peut avoir des effets assez importants qui méritent une analyse plus détaillée. L'incertitude temporelle se manifeste par l'introduction d'une erreur dans la perception de l'intervalle de temps écoulé entre deux événements, par rapport à l'intervalle de leurs occurences effectives.

Or, lorsque le contenu du message transmis est une fonction de cet intervalle de temps, une incertitude temporelle implique une incertitude sur le contenu du mes sage, ce qui peut entraîner un comportement non-voulu du système. En praticulier, si le rapport erreur/intervalle est important, l'incertitude temporelle peut se manifester sous la forme d'une inversion de l'ordre d'arrivée des événements par rapport à leur ordre d'émission, ce qui risque d'altérer l'évolution d'une machine séquentielle.

Dans les dispositifs réels de traitement et de transmission des données, il existe toujours un retard plus ou moins important associé à la fonction de trans fert. Ce délai est dû aux phénomènes d'inertie propres à chaque technologie (mécanique, magnétique, électrique, ...) dont la nature ne concerne pas notre étude.

Il est parfois possible de calculer analytiquement ce retard, ce qui permet de reconstituer l'information temporelle. Néanmoins, cela est souvent très difficile voire même impossible (par exemple, si la ressource de traitement est partagée entre plusieurs utilisateurs). D'ailleurs, même si ce retard est calculable, cette approche présente deux conséquences néfastes :

a) le calcul du temps de retard introduit un degré de complexité supplémentaire pour la conception,
b) la définition du contrôleur devient dépendante de l'interpréteur (en fonction duquel les retards sont déterminés), ce qui en limite la portabilité : un changement de matériel implique une redéfinition de la fonction de commande.

Il est donc préférable de concevoir le contrôleur indépendamment des temps de calcul. Cela équivaut à dire que :
a) le procédé contrôlé peut attendre indéfiniment les commandes du contrôleur, ou que
b) le contrôleur n'a pas de retard.

Ces deux hypothèses sont en effet équivalentes, car elles correspondent à dire que "le temps (c'est-à-dire l'état) du procédé n'évolue pas tant que le contrôleur calcule".

Ces hypothèses correspondent à la réalité, à condition que l'état du procédé "ne se modifie pas trop" pendant le temps de calcul.

[WIRT 77] a présenté une technique pour la conception pratique du contrôleur, satisfaisant les hypothèses simplificatrices ci-dessus:

1) la spécification initiale du contrôleur ne fait aucune hypothèse sur le temps de traitement ; elle définit explicitement tous les signaux de synchronisation nécessaires ;

2) pour tout signal non fourni par le procédé contrôlé (et par conséquent fourni par le contrôleur, selon la terminologie utilisée), il faut calculer analytiquement les contraintes de temps qui permettent l'attente de ce signal ;

3) il faut vérifier si ces contraintes sont satisfaites par le contrôleur ; en d'autres termes, il faut réaliser un contrôleur qui garantisse la vitesse exigée avec une marge de sécurité raisonnable.

1.2. Les aléas des circuits combinatoires

Un des effets de l'introduction des retards dans la transmission et le traitement des informations est la production d'aléas qui sont des sorties transitoires différentes de celles définies par les entrées et les opérateurs. Les aléas apparaissent lors d'une transition des entrées.

Les aléas des fonctions combinatoires peuvent générer des séquences non désirées sur des automates séquentiels qu'ils activent. Ce problème a deux solutions :
1) éviter les aléas,
2) prendre des précautions pour qu'ils ne soient pas gênants.
Des techniques pour éviter la génération des aléas sont développées dans [McCL 65]. Elles impliquent normalement un alourdissement de la réalisation.

Au lieu d'éviter les aléas, il est parfois plus avantageux de rendre l'automate activé insensible à ceux-ci. Cela revient à mettre un filtre passe-bas à l'entrée de l'automate séquentiel (figure 7.1.). Le filtre passe-bas doit avoir une fréquence de coupure qui élimine tous les signaux ayant une fréquence plus élevée que l'inverse du retard maximal du chemin suivi par les signaux depuis l'entrée du circuit combinatoire jusqu'à sa sortie.

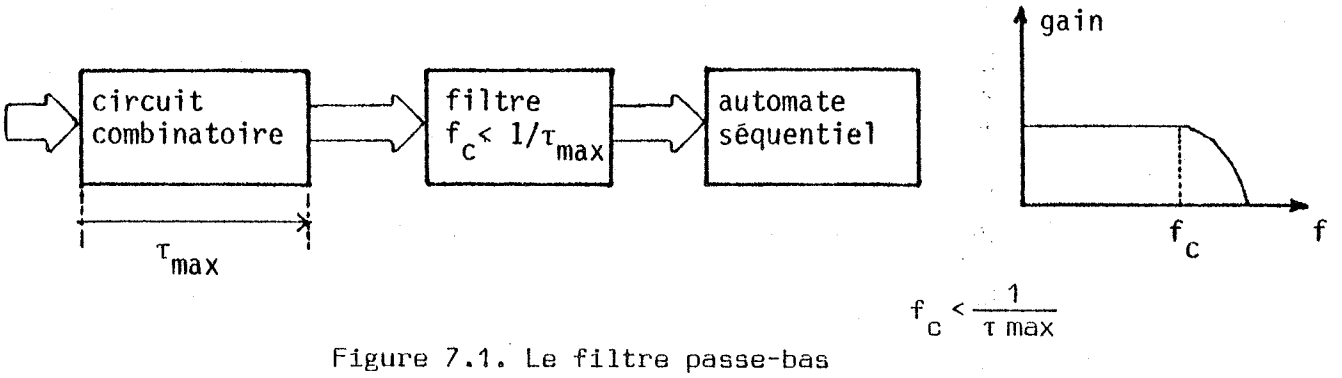


Figure 7.1. Le filtre passe-bas

Ce filtre absorbe les signaux qui ont moins qu'une énergie déterminée.

Réalisation du filtre passe-bas

Le filtre passe-bas peut être réalisé de différentes manières :

a) par un filtre analogique ;

b) le dispositif séquentiel activé par le circuit combinatoire peut incorporer implicitement un filtre, en raison de son temps de réponse ; si par exemple le circuit combinatoire commande un relais électromécanique, les aléas peuvent être absorbés par la bobine et par l'inertie mécanique ;

c) par une boucle de réalimentation négative à travers l'automate séquentiel jusqu'à l'entrée du circuit combinatoire ; les effets d'un aléa se propagent alors à l'automate, mais ils restent passagers ;

d) on peut utiliser pour le système une horloge dont la période est plus longue que le temps de stabilisation (disparition des aléas)du circuit combinatoire pour l'activation de l'automate séquentiel (figure 7.2.). D'ailleurs ceci est l'une des raisons qui ont mené à la réalisation synchrone des processeurs et de machines séquentielles.

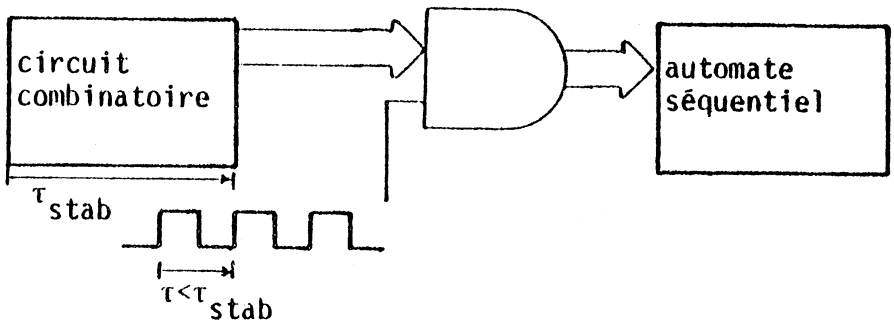


Figure 7.2. Réalisation synchrone du filtre passe-bas.

1.3. Les courses dans les automates séquentiels

Un automate séquentiel est un dispositif réalisé avec des circuits combinatoires avec des chemins de réalimentation, ce qui lui permet de mémoriser un état interne.

Soit un automate d'états finis $\mathcal{A}$ défini par le 6-uplet

$$\mathcal{A} = \langle S, s_0, I, O, f, g \rangle$$

où

$S = \{s_0, s_1, \ldots\}$ est l'ensemble fini de ses états internes

$s_0 \in S$ est l'état initial

$I = \{i_0, i_1, \ldots\}$ est l'ensemble fini de ses entrées

$O = \{o_0, o_1, \ldots\}$ est l'ensemble fini de ses sorties

f est une application $f : S \times I \rightarrow S$

g est une application $g : S \times I \rightarrow O$

Un état s est dit stable pour l'entrée i si

$$f(f(s, i), i) = f(s, i)$$

et non stable si cette égalité ne se vérifie pas.

La réalisation d'un automate admet que la transition entre deux états stables se fasse par l'intermédiaire d'une suite d'états non stables : cette évolution s'appelle "course".

La notion de stabilité peut s'étendre aux composantes du vecteur qui code les éléments de S. Une composante b_i du vecteur booléen qui code les états est dite stable pour l'entrée i si

$$f(f(b_i, i), i) = f(b_i, i)$$

S'il y a au plus une composante de l'état interne non stable à chaque instant, la course est dite non critique, et l'automate arrive à un état stable prévisible. Une course peut provoquer des aléas sur les sorties, à propos desquels les mêmes considérations faites pour les fonctions combinatoires restent valables.

Si plusieurs composantes sont simultanément non stables, alors la course est dite critique et l'état final dépend de l'ordre d'évolution de ces composantes, qui est fonction des retards non spécifiés. Les courses critiques doivent être évitées et des techniques correspondantes ont été développées [UNGE 59].

Si les événements associés à deux entrées sont trop proches dans le temps, leurs effets superposés peuvent conduire à un comportement non prévisible, équivalent à une course critique, car ils se propagent à des vitesses indéterminées. L'interdiction d'événements simultanés sur les entrées se traduit donc par l'exigence d'un intervalle de temps suffisant pour que leurs effets soient absorbés, c'est-à-dire que l'état interne et les sorties de l'automate découlant du premier événement, soient stables avant l'arrivée du second. Ceci revient à poser une condition d'exclusion mutuelle sur l'occurence des événements associés aux entrées.

1.4. Application à la méthodologie proposée

Selon la méthodologie proposée, l'observateur est constitué d'un ensemble d'automates séquentiels. Les signaux qui les activent sont issus de capteurs placés sur le procédé contrôle, de simulateurs et des commandes du contrôleur.
Il y a deux possibilités pour assurer cette mutuelle exclusion (et l'attente de la stabilisation) pour les entrées des automates:

a) ou bien les capteurs (ou autres sources) ne génèrent des événements qu'avec un intervalle suffisant, non perturbé par les moyens de communication procédé/contrôleur ;
b) ou bien il existe un tampon d'attente intermédiaire (figure 7.3) dont on extrait une transition lorsque l'automate est stable. Ce tampon agit comme un "adaptateur de fréquences" entre les sources des entrées et l'automate.

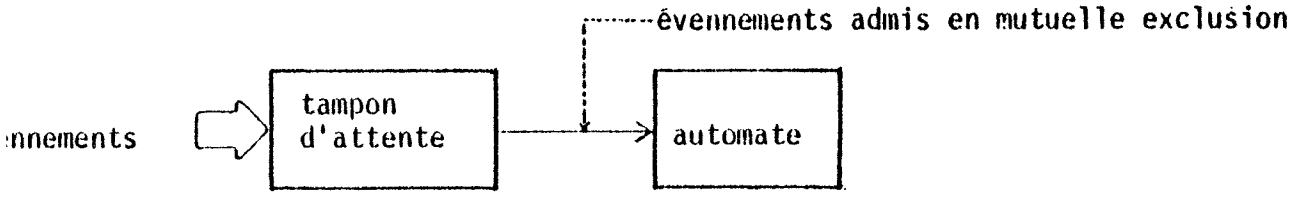


Figure 7.3. Le tampon d'attente

La fréquence moyenne des événements d'entrée ne peut pas excéder la vitesse moyenne de l'automate, qui est elle-même fonction du retard maximum de génération des commandes par le contrôleur. Le tampon doit être dimensionné en fonction de ce retard et de la fréquence maximale d'occurrence des événements.

Bien que la solution (b) soit pous générale, la première est plus économique et peut être utilisée si les événements qui activent l'automate sont générés avec un intervalle suffisant pour permettre sa stabilisation. Il faut toutefois s'assurer que la condition est respectée : le procédé contrôlé peut être vu comme un ensemble de composantes évoluant en parallèle, ce parallélisme étant limité par des contraintes imposées par la nature même du procédé et par le contrôleur (certaines transitions d'une composante sont conditionnées par l'état des autres). Par conséquent, certains événements issus de composantes différentes n'ont pas de relation de précédence par la spécification même du système ; il est donc impossible d'assurer leur non simultanéité. Il en découle que l'état du système ne peut pas être reconstitué par un seul automate sans utiliser un tampon d'attente. Il faut alors décomposer l'automate de façon à ce que chaque automate résultant de cette décomposition représente le comportement d'une composante ou d'un ensemble de composantes indépendantes.

Les variables physiques (position, température, pression, ...) sont facilement identifiables par leur nature même et permettent d'une façon sûre de construire des automates qui satisfont la condition de non parallélisme.

La méthodologie de conception du contrôleur, proposée dans le chapitre 3, amène à la construction d'automates qui regénèrent des composantes du procédé contrôlé. Lorsqu'il s'agit de variables "physiques", le parallélisme interne de chaque automate est nul et l'on peut se dispenser du tampon d'attente.

Lorsque plusieurs processus, par ailleurs parallèles, souhaitent utiliser une ressource commune (exemple : machine outil partagée par plusieurs lignes de production, ou périphérique commun à plusieurs processeurs), il faut qu'ils interrogent l'automate qui gère cette ressource en exclusion mutuelle, grâce à des primitives de synchronisation. Un tampon d'attente devient alors nécessaire pour stocker les requêtes en attente sur cette ressource.

1.5. L'incertitude temporelle

Le contrôleur (y compris les lignes de communication,etc.) introduit des retards non contrôlables, qui introduisent une incertitude, ou "bruit", dans l'information portée par l'occurrence des événements. Une technique pour remédier à cet inconvénient, est de transformer cette information temporelle en information "de valeur" le plus près possible de la source des événements, c'est-à-dire avant l'introduction des "bruits" par les moyens de communication et de calcul. Pour cela, il faut que le système dispose d'un repère temporel global.

A mesure que la distance géographique entre les composantes d'un système distribué augmente, les retards dans ses lignes de communication (physiques ou logiques [LAMP 78]) augmentent. Le synchronisme global ne pourra donc être assuré que si le retard maximum de transmission est plus petit que la période des signaux de synchronisation(période de l'horloge globale). Il est toujours possible de diminuer la fréquence de l'horloge proportionnellement à l'augmentation des distances, mais cela entraîne une diminution de la vitesse, donc de la puissance de calcul, et de la résolution temporelle proprement dite.

Une solution partielle (mais apparemment la seule possible) a été proposée par [MARQ 80] ; elle consiste à instaurer un synchronisme limité à des régions de taille limitée, appelées isochrones, de façon à capter l'information temporelle de sources relativement proches, sans pénaliser globalement le système. Cette solution correspond en effet à la proposition énoncée ci-dessus, c'est-à-dire de transformer l'information temporelle avant l'introduction de retards (non contrôlés) par les moyens de communication et de calcul.

Dans le cas de contrôleurs rapides et proches du procédé contrôlé, cette source de bruit sur l'information temporelle peut être négligée. Pour un système distribué (procédé contrôlé et/ou contrôleur distribué), on peut souvent distinguer (1) des boucles de commande rapides sur du matériel localisé, (2) des boucles plus lentes sur des fonctions de plus haut niveau dans une hiérarchie de commande. A chaque niveau et à chaque élément de la hiérarchie des contraintes temporelles sont à respecter. Dans le contrôle des systèmes physiques, il arrive souvent que (a) les constantes de temps des niveaux supérieurs soient plus importantes que celles des niveaux inférieurs, et (b) les distances physiques entre le procédé et le contrôleur augmentent avec le niveau puisqu'une partie plus étendue du procédé contrôlé est considérée. En général, ces deux effets se com-

2. REALISATION

A défaut d'une structure de contrôle de flot, telle qu'il en existe dans les langages procéduraux, il est nécessaire de savoir de quelle manière choisir les instructions qui doivent être évaluées à chaque instant par un interpréteur ayant un nombre limité de processeurs disponibles.

Nous allons étudier deux méthodes de réalisation d'un interpréteur CSL sur des processeurs séquentiels :

1) interprétation du type "automate programmable"
2) interprétation par un mécanisme dirigé par les données.

2.1. Réalisation du type automate programmable

L'interpréteur évalue en permanence toutes les étapes du programme en séquence, dans une grande boucle.

Pour éviter les aléas, il faut que :

1) l'acquisition des entrées et le calcul se réalisent dans des phases séparées (pour que le contrôleur dispose d'un vecteur d'entrée cohérent) ;

et que

2.a) ou bien le calcul de toutes les expressions s'effectue une seule fois par cycle, c'est-à-dire que les expressions sont ordonnées selon une hiérarchie telle que tous les opérandes d'une expression sont évalués avant l'évaluation de l'expression elle même.

2.b) ou bien le calcul doit s'effectuer de manière répétitive dans une boucle, jusqu'à ce que tous les signaux soient stables.

Cette solution a l'inconvénient de ne pas être très efficace, surtout la solution 2.b ; en effet, un grand nombre d'expressions sera évalué à chaque cycle, sans qu'aucun de ses opérandes n'ait subi de modification. Par contre, elle permet une réalisation assez facile et elle est très utilisée industriellement, lorsque le temps de réponse n'est pas un facteur critique. De plus, la distribution des expressions dans un système multiprocesseur doit se faire en prenant des précautions pour assurer que les conditions énoncées ci-dessus soient globalement satisfaites.

2.2. Calculateurs dirigés par les données (data driven)

2.2.1. Rappel

La conception d'ordinateurs basés sur le principe d'exécution dirigée par les données [DENN 79], est proposée comme alternative aux architectures basées sur la notion traditionnelle d'exécution séquentielle des instructions d'un programme. Ce concept est une solution possible au problème de l'exploitation efficace du parallélisme.

Fondamentalement, le concept de calcul dirigé par les données peut être vu comme une forme différente de l'exécution d'une instruction, comme une alternative au concept de von Neumann d'exécution séquentielle. Dans un calculateur dirigé par les données, une instruction est prête à être exécutée lorsque ses opérandes sont présents ; il n'existe pas de concept du "flot de contrôle". Les calculateurs dirigés par les données n'ont pas de compteur ordinal. Une des conséquences de l'exécution activée par les données est le fait que plusieurs instructions peuvent être disponibles au même moment pour l'exécution. Un calcul hautement parallèle est une conséquence naturelle.

2.2.2. Application à l'interprétation de CSL

En fait, nous reprenons les concepts développés pour l'exécution d'instructions élémentaires (un seul générateur) dans le projet LAU (Langage d'Assignation Unique) [CERT 74] et l'appliquons à un niveau plus élevé, celui de l'évaluation des instructions CSL constituées par des fonctions combinatoires et par des automates séquentiels.

La forme de l'exécution interne des instructions ne nous concerne pas ; il est cependant spécifié qu'elle prend un temps fini et que les sorties ne deviennent visibles (c'est-à-dire que leur valeur n'est altérée) qu'après la stabilisation de la fonction interne. Les aléas au niveau de chaque instruction sont ainsi éliminés. Cette spécification est facile à satisfaire dans la pratique.

Le but des paragraphes qui suivent est de proposer une solution pour éviter les aléas provoqués par les retards de propagation d'un événement dont les conséquences suivent deux ou plusieurs chemins dans l'évaluation d'un programme.

2.2.3. Le graphe de dépendance

Un programme dirigé par les données peut être représenté par un graphe orienté, appelé graphe de dépendance (GD). Les noeuds de ce graphe sont les opérateurs, ou instructions, et les arcs en sont les opérandes.

Exemple : soit le contrôleur du chariot du § 3.1.1.; son GD est représenté sur la figure 7.4.

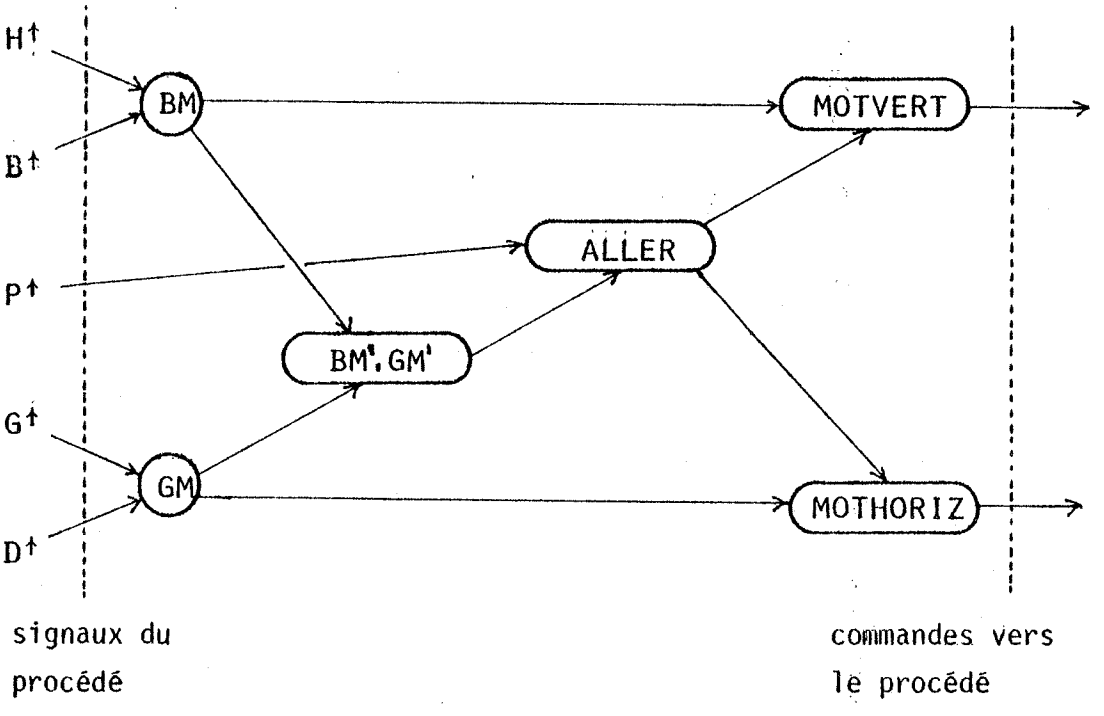


Figure 7.4. Graphe de dépendance

Exemple de l'évolution du GD à la suite de quelques événements :

P↑- ALLER↑, MOTVERT ← MONTER, MOTHORIZ ← A DROITE

H↑- BM↓, ALLER inchangé

, MOTVERT ← ARRET

etc ...

2.2.3.1. Activation d'un noeud du graphe de dépendance

Dans un mécanisme dirigé par les données, chaque noeud est activé, c'est-à-dire que les fonctions du noeud sont calculées, lorsque ses opérandes sont disponibles. Par conséquent, chaque noeud n'est activé qu'une seule fois, par configuration de son vecteur d'entrée ("encarnation" dans la terminologie de [KESS 77]). Dans un programme de contrôle de procédés, tel que CSL, le vecteur d'entrée représente en fait des variables ou des fonctions de variables du procédé contrôlé. Ces variables changent de valeur avec l'évolution du procédé. Il faut donc prévoir une réévaluation des fonctions d'un noeud à chaque fois que ses opérandes changent de valeur (si le noeud contient un automate séquentiel, un seul changement est pris en compte à la fois). Dans ce sens, on peut considérer que le noeud est activé non pas par l'arrivée des données, mais par les transitions de ces valeurs, bien que le calcul se fasse en fonction des valeurs.

2.2.3.2. Initialisation d'un noeud

La mise en marche du contrôleur est considérée comme une situation particulière. Avant qu'une valeur ne lui soit attribuée, une variable est dite "INCONNUE" et ne peut pas être utilisée comme opérande. Pour qu'un noeud soit évalué, il faut que tous ses opérandes soient connus. Au premier cycle, après le démarrage du système, un noeud est activé lorsque tous ses opérandes ont été calculés ou fournis par les capteurs du procédé. Le programme se comporte alors tout à fait comme un programme d'assignation unique [CERT 74].

2.2.4. Sous-graphe d'une variable $SG(v_i)$

Soit $V = \{v_1, v_2, v_i, \ldots\}$ l'ensemble fini des variables d'entrée du contrôleur. On appelle "noeud descendant d'une variable d'entrée v_i" tout noeud susceptible d'avoir la sortie ou l'état interne modifié par v_i ou par une fonction de v_i. Le sous-graphe d'une variable v_i est constitué par l'ensemble des noeuds descendants de cette variable et des arcs qui les relient. Le $SG(v_i)$ est un sous-graphe du graphe de dépendance GD du programme de contrôle. Le GD et les $SG(v_i)$ pourraient être construits par exemple par le compilateur à partir d'une description CSL.

Lorsqu'une transition sur une variable v_i paraît à l'entrée du GD, le $SG(v_i)$ associé est activé. Les noeuds du sous-graphe seront évalués lorsque les opérandes désignés par les arcs appartenant au sous-graphe auront fini d'être évalués.

Exemple :

Soit le graphe de la figure 7.5. :

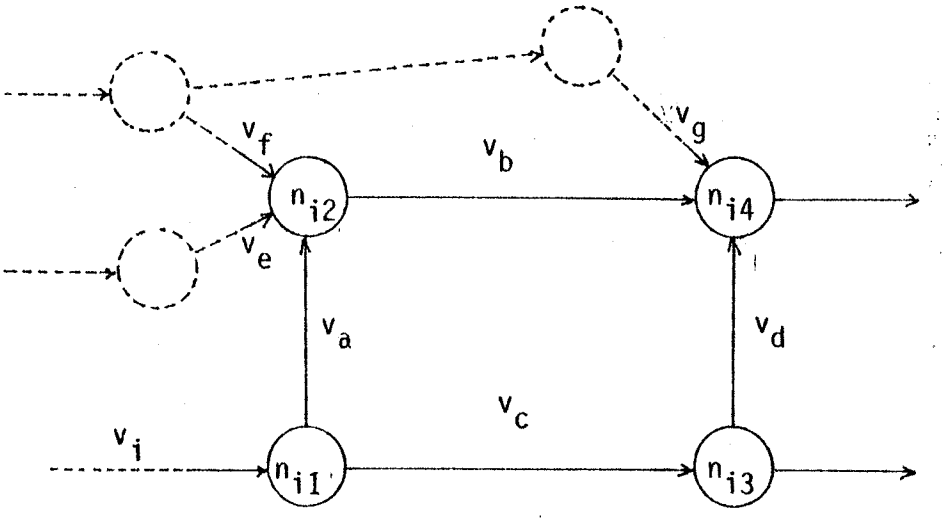


Figure 7.5. Sous-graphe de v_i

A la suite d'une modification de v_i, le graphe devient activé. Ses noeuds n_{i1}, n_{i2}, n_{i3} et n_{i4} sont évalués quand respectivement v_i, v_a, v_c et (v_b, v_d) auront été calculés.

Après son activation (§ 7.2.3.1.), le noeud est évalué.

2.2.4.1. Noeuds combinatoires

Un noeud combinatoire peut être visualisé comme étant composé de (figure 7.6) :

- un tampon d'entrée des événements,
- un tableau de dependances,
- la fonction proprement dite,
- un contrôleur.

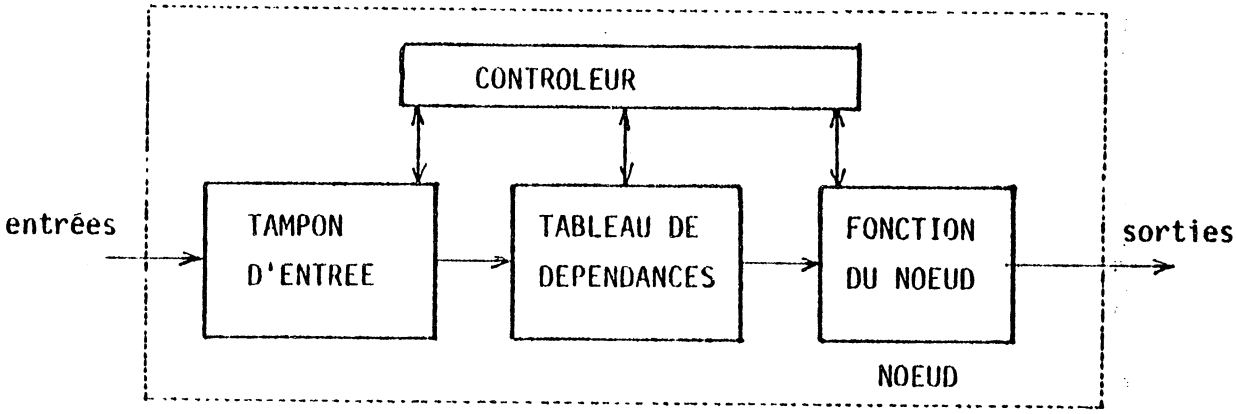


Figure 7.6. Schéma de l'architecture de l'interpréteur d'un neud combinatoire.

. Tampon d'entrée :

il contient les entrées reçues (en exclusion mutuelle), mais non encore évaluées par le noeud. Les éléments contenus dans ce tampon ont le format suivant :

variable d'entrée génératrice du SG	opérande	valeur

exemple:

P	ALLER	VRAI

. Tableau de dépendances :

il contient les informations nécessaires pour décider de l'évaluation d'un noeud. Il a autant d'entrées (éléments) que de variables d'entrée (opérandes du noeud). Chaque élément contient :

- la nouvelle valeur de l'opérande,
- un couple de cases par sous-graphe auquel appartient le noeud ; de ce couple de cases, l'une (STRU) représente la structure du SG, c'est-à-dire les arcs d'entrée, et l'autre (ACT) reçoit des "jetons" (indicateurs de "opérande disponible", symbolisés par "." dans la figure 7.7), lorsque la variable correspondante est reçue du tampon d'entrée.

Les cases STRU sont remplies par le compilateur ; les cases ACT sont remplies et effacées dynamiquement, c'est-à-dire pendant l'exécution. Chaque noeud a deux états d'exécution : LIBRE et OCCUPE. Lorsqu'un opérande arrive au noeud, il attend dans le tampon d'entrée jusqu'à ce que le noeud soit LIBRE (non en calcul). Ensuite lorsque le noeud devient LIBRE, les opérandes sont transférés un à un, dans un ordre quelconque, dans le tableau de dépendances. Le noeud devient OCCUPE lorsque tous les STRU d'un SG ont reçu des jetons dans les cases ACT correspondantes.

Le noeud évalue sa fonction. Pendant qu'il est OCCUPE, le tableau de dépendances ne reçoit pas d'entrées du tampon. Lorsque le noeud termine de calculer, tous les jetons du tableau correspondant aux sous-graphes calculés, sont effacés et le noeud redevient LIBRE.

Exemple :

La figure 7.7. présente le tableau de dépendances de la variable MOTVERT du contrôleur du chariot décrit au § 3.1.1., dont le graphe de dépendance est montré dans la figure 7.4.

opérande	valeur	SG(P)		SG(D)		SG(G)		SG(H)		SG(B)	
		STRU	ACT	STRU	ACT	STRU	ACT	STRU	ACT	STRU	ACT
BM								•		•	
ALLER		•		•		•		•		•	

Figure 7.7. Tableau de dépendances de MOTVERT

Un événement sur les entrées P, D ou G provoque la réévaluation de ALLER qui, à son tour, provoque la réévaluation de MOTVERT.
Un événement sur H ou B provoque une réévaluation de ALLER. MOTVERT ne peut pourtant être évalué avant une réévaluation de BM. Si par contre il y a une modification de BM provoquée par H ou par B, il faut attendre que ALLER ait aussi été réévalué.

2.2.4.2. Noeuds séquentiels

Les noeuds séquentiels (noeuds ayant des éléments de mémorisation explicites) peuvent être considérés de deux manières :

a) comme un ensemble de noeuds combinatoires câblés de façon à constituer des éléments de mémorisation ; on retombe alors dans le cas précédent ; l'exclusion mutuelle doit être assurée par la spécification du contrôleur ;

b) comme des entités primitives ; dans ce cas, le tableau des dépendances devient nécessaire.

La gestion des événements qu'activent les automates de l'observateur a été analysée dans le § 6.3.1.2.1. L'interpréteur dirigé par les données doit assurer l'exclusion mutuelle et l'attente des événements dans un tampon de courte échéance.

Remarque :

Cette façon d'interpréter une description CSL permet la distribution des expressions d'une forme quelconque dans un système multiprocesseur. Par contre, elle n'est pas encore très efficace ; une expression peut être évaluée inutilement, juste pour satisfaire la séquence imposée par le graphe de dépendance. Cette inefficacité est due aut fait que la connaissance de l'ordre des événements n'est pas nécessairement incorporée au contrôleur. Cette information, la prédictabilité (§ 3.6.) pourrait lui être incorporée, par exemple par l'inclusion dans l'observateur d'un graphe d'états représentatif de l'évolution du procédé contrôlé. Cette option n'est possible que dans la mesure où le procédé est déterministe.

CONCLUSIONS

CHAPITRE 8 - CONCLUSIONS

Nous venons de présenter un formalisme et une méthodologie pour la description de contrôleurs de procédés discrets en temps réel.

Ce formalisme est général et universellement applicable ; il encourage une discipline de raisonnement qui peut être utile pour la conceptualisation d'un problème, indépendemment de l'outil ou du langage disponible pour sa réalisation.

La méthodologie qui découle du formalisme permet de décomposer la conception du contrôleur en deux étapes :

- la première est la définition de la fonction de commande combinatoire ; lors de cette étape, on ne fait pas d'hypothèse sur la manière de détecter les composantes du vecteur d'état choisi (capteurs et/ou automates) ;
- la deuxième étape, qui correspond à la conception de l'observateur, est consacrée à la détection des composantes du vecteur d'état, utilisées comme opérandes de la fonction de commande. On remarque alors l'existence d'un compromis entre la puissance des capteurs et la complexité de l'observateur. Ce compromis est essentiellement économique, et le point d'opération optimal peut changer avec le coût de ces éléments.

Bien que nous ayons étudié principalement les automates séquentiels comme composantes de l'observateur, le concept peut être étendu à une gamme plus large d'éléments, tels que les convertisseurs analogiques-numériques, les fonctions statistiques utilisées pour la compression des données (le "signal averaging" utilisé dans la saisie de données astronomiques, par exemple), etc ...

Le formalisme s'est révélé très convenable dans l'analyse des aspects technologiques qui interviennent lors de la conception de l'observateur de disponibilité de ressources partagées dynamiquement. Il pourrait encore être étendu aux contrôleurs de ressources réparties, dans lesquels on utilise des observateurs locaux "approximés", dans le but de réduire le coût des transmissions entre sites [AFCE 80].

Pour tester ce formalisme sur des cas réels, nous avons proposé un langage de description du contrôleur, appelé CSL, qui a été appliqué à la description d'un grand procédé industriel de palettisation (N.B. en raison des contraintes de temps, nous nous sommes bornés à la spécification du contrôleur sans essayer son implémentation effective).

Cette étude a attiré notre attention sur les faits suivants :

- un nombre relativement réduit de types d'automates préprogrammés permet de reconstituer (observer) un sous-ensemble important de composantes du vecteur d'état d'un procédé discret ;
- la lisibilité d'une description est améliorée lorsque l'évolution des états du procédé dans le temps est représentée par la séquence lexicographique du programme. Les GRAFCET [AFCE 77], par exemple, possèdent cette qualité. Cette lisibilité est souvent obtenu au prix de la redondance de la représentation du vecteur d'état, sous la forme d'une image de l'évolution du procédé contrôlé dans le contrôleur, indépendemment des capteurs disponibles.

En CSL, le même effet pourrait être obtenu par l'inclusion systématique d'un graphe d'états, réseau de Pétri, etc .. représentant le procédé dans l'observateur. Par ailleurs, ce graphe pourrait être présenté séparément du contrôleur.

En résumé, les motivations pour l'inclusion d'un automate représentant le procédé dans l'observateur sont :
1) économiques (compromis avec le coût des capteurs),
2) techniques (facilité de l'accès à l'état),
mais aussi
3) phychologiques (visibilité de l'évolution du procédé contrôlé).

Nous avons cherché à concevoir la syntaxe de CSL de manière à refléter aussi nettement que possible les éléments du formalisme. Nous aurions pu aller encore plus loin, en séparant plus explicitement les éléments de la partie combinatoire, de l'observateur, en passant comme interface le vecteur d'état. Cela a été fait comme illustration dans un exemple (chapitre 3), mais il nous a semblé finalement aussi facile au lecteur de retrouver l'origine de chaque opérande dans la description du service. En plus, cela nous a permis de rapprocher les instructions (automates et expressions de commande combinatoires) concernant une composante ou partie spécifique du procédé contrôlé.

Le langage présenté ne se pose pas en concurrent des langages de programmation de haut niveau existants ; il a été créé plus comme l'instrument d'application d'un formalisme, que comme outil pratique. Ainsi, n'avons-nous pas attaché beaucoup d'importance à l'efficacité de son interprétation. De plus, une des propositions de réalisation du chapitre 7, est basée sur un mécanisme dirigé par les données, dont il n'existe jusqu'à présent pratiquement pas de réalisations opérationnelles.

Par ailleurs, sa simplicité et la nature des concepts utilisés dans la méthodologie de conception du contrôleur le situe comme une formalisation en même temps qu'une extension des concepts utilisés dans les automates programmables industriels. Ce fait nous semble important, dans la mesure où il permet de nous éloigner des contraintes imposées par les mécanismes d'interprétation séquentiels, pour nous ramener à des notions qui concernent essentiellement la nature du domaine d'application. Tel était notre but au début de ces études, et nous espérons l'avoir au moins partiellement atteint.

ANNEXE

APPLICATION A UN PROCEDE INDUSTRIEL

APPLICATION A UN PROCEDE INDUSTRIEL

1 - PRESENTATION DU SYSTEME

Nous décrivons ici une partie d'une installation d'emballage et de stockage de lampes électriques (fig. A1) dont le contrôleur sera spécifié selon la méthodologie et la syntaxe proposées.

Les lampes sortent de la ligne de fabrication en colis, qui sont assemblés, empilés et distribués automatiquement, selon des spécifications qui paramétrisent le contrôleur.

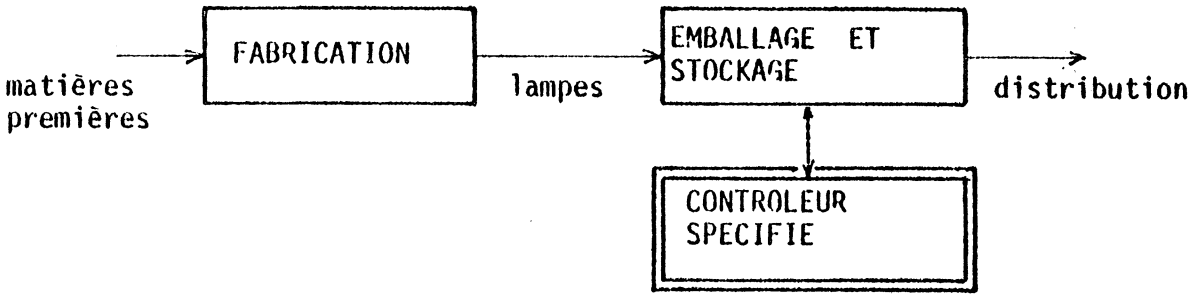


Fig. A1 - Schéma général du système

La dernière étape de la ligne de fabrication proprement dite est une station de conditionnement, d'où sortent les paquets de lampes sur une des 16 "lignes d'amenée". Ces colis sont, à la sortie de la paqueteuse, déjà sous la forme de "packs", soit de "cartons", dont les caractéristiques particulières ne sont pas significatives pour la spécification du contrôleur. Les colis sont assemblés en "couches" sur les lignes d'amenée. Les couches sont transférées sur un distributeur circulaire appelé "diplodocus". Ensuite, les couches sortent du diplodocus et sont transportées par une des trois "lignes d'alimentation palettiseur" vers respectivement un des trois palettiseurs, où elles sont conformées et empilées sur des palettes (fig. A2).

L'analyse et le programme ci-dessous concernent le système depuis les lignes d'amenée jusqu'aux palettiseurs.

Le comportement du procédé est paramétré par des "tables d'affectation" :

TALA - table d'affectation des lignes d'amenée, et

TAP - table d'affectation des palettiseurs.

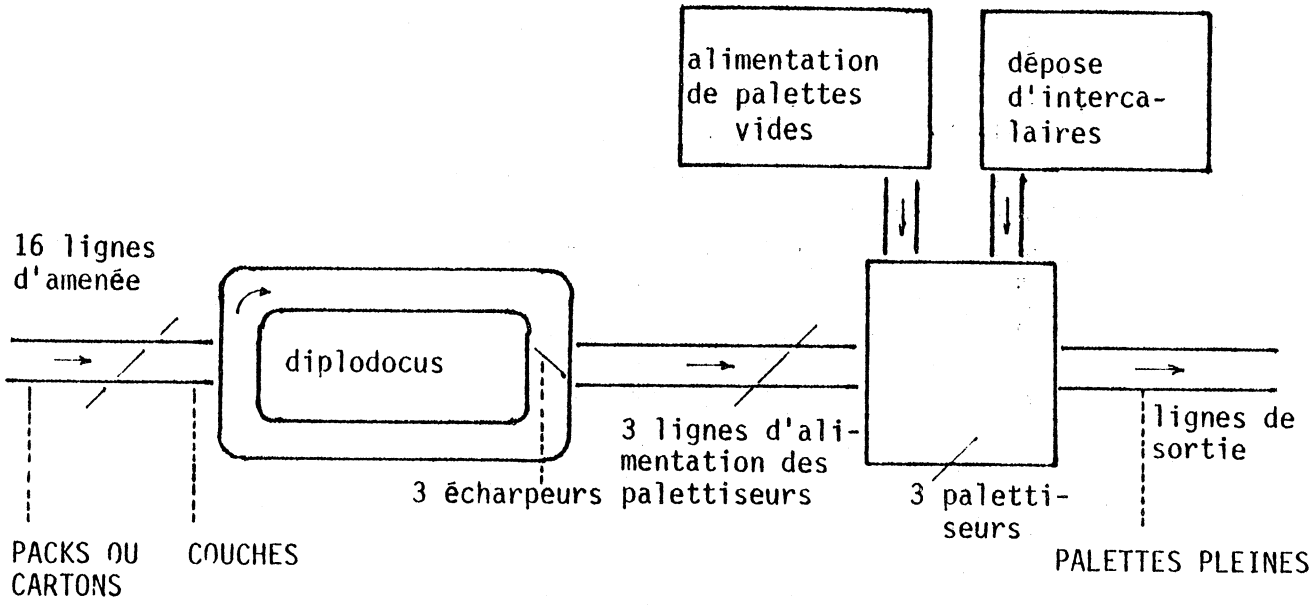
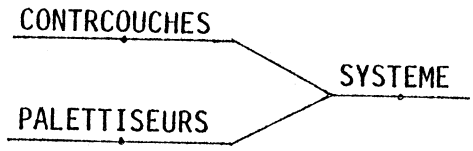


Fig. A2 - Le procédé contrôlé

Nous avons divisé le procédé, et par conséquent le contrôleur, en deux grandes parties: CONTRCOUCHES et PALETTISEURS.



La partie CONTRCOUCHES comporte les 16 lignes d'amenée, le diplodocus et les écharpeurs. La partie PALETTISEURS comporte les lignes d'alimentation palettiseur et les 3 palettiseurs proprement dits.

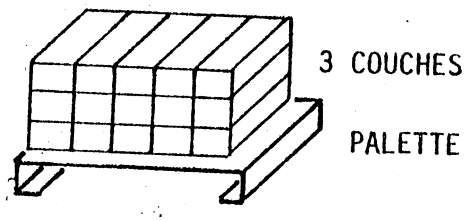


Figure A3 - Trois couches sur une palette

Remarque :

La spécification des services qui composent le contrôleur, organisés selon une arborescence, suit un ordre ascendant, c'est-à-dire qu'après avoir présenté schématiquement l'organisation de chaque service, on commence par la spécifications de ses fils, et ensuite on donne celle du père. Par conséquent, le dernier service spécifié, SYSTEME, incorpore tous les précédents.

1.1. Suivi des couches

Chaque couche est suivie à travers le système par une "fiche-couche", générée à la sortie du TRANSPORTEURB de chaque ligne d'amenée.

La fiche-couche (FC) est composée des informations suivantes :
FINLOT - booléen qui indique que la couche est la dernière de la palette,
CARTON/PACK, COLISPARCOUCHE, PALETTISEUR,
NREF - entier qui indique l'ordre de la couche sur la palette,
LSORT - entier qui indique la ligne de sortie.

Les fiches-couche sont stockées dans des tables du contrôleur, selon la localication de la couche représentée :

1 - Table Image des Lignes d'Amenée (TILA) avec 16 entrées qui représentent l'état de chargement des lignes d'amenée. Les FC de cette table sont chargées lors de leur génération et effacées lors du transfert sur le diplodocus.

2 - Table Image du Diplodocus (TID) avec 22 entrées qui représentent l'état de chargement des 22 loges. Les éléments de cette table sont chargés lors de la réception de la couche sur le diplodocus et effacés lors de leur écharpage.

3 - Table Image Queue des Lignes d'Alimentation Palettiseur (QLAP) contient les fiches-couche des lignes d'alimentation palettiseur.

4 - Table Image des Tables de Préparation du Palettiseur (TITP) avec une entrée par palettiseur.

Ces tables image constituent l'observateur des objets manipulés (les couches)

par le procédé. Une autre solution serait d'installer des capteurs qui détectent à chaque instant la position et la nature des couches, ce qui coûterait évidemment beaucoup plus cher.

FINLOT	CARTON/PACK	COLISPARCOUCHE	PALETTISEUR	NREF	LSORT

Fig. A4 - Fiche-couche (FC)

Déclarations de structure:

```
STRUCT FC(FINLOT, CARTON/PACK, COLISPARCOUCHE, PALETTISEUR, NREF, LSORT) ;
STRUCT TILA (16 FC) ;
STRUCT TID (22 FC) ;
STRUCT TITP (1 FC) ;
```

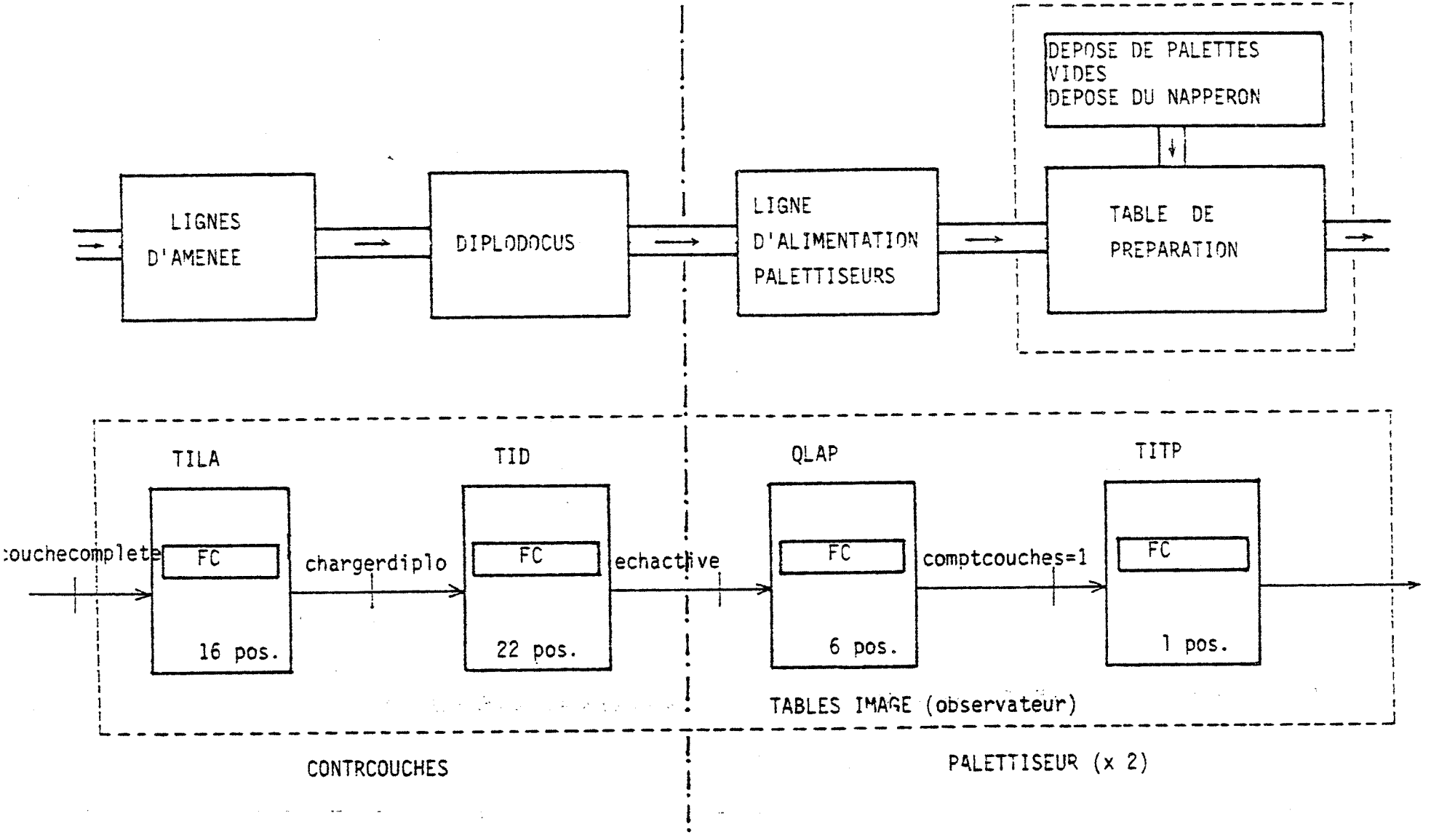


Fig. A5 - Suivi des couches

2 - Le contrôleur CONTRCOUCHES

2.1. Structure du contrôleur

Bien que la structure du contrôleur puisse être facilement déterminée d'après le texte du programme, il est confortable de la représenter graphiquement. Le numéro associé à un sous-arbre indique qu'il en existe autant d'exemplaires sortant du noeud-père adjacent. Par défaut, il y en a un seul.

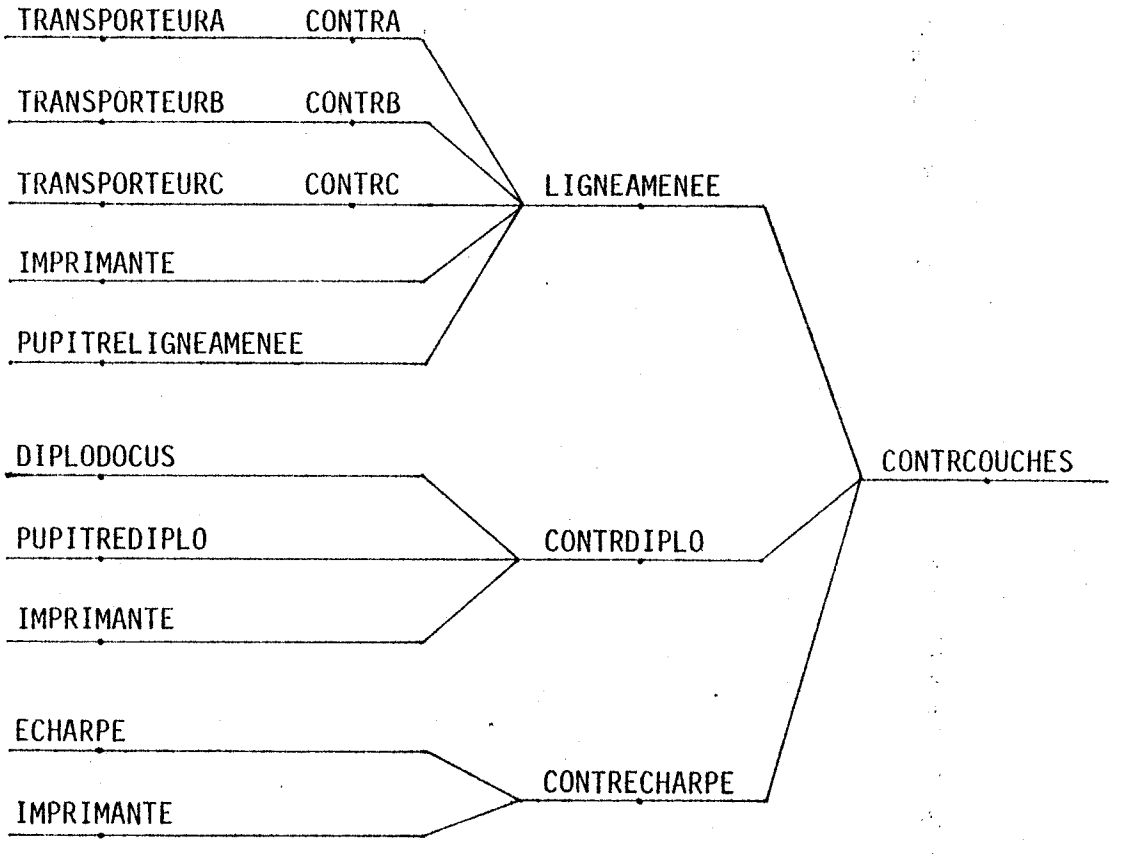


Fig. A6 - Structure du contrôleur CONTRCOUCHES

2.2. Lignes d'amenée

Il existe 16 lignes d'amenée. Chaque ligne est composée de trois transporteurs consécutifs, appelés TRANSPORTEURA, TRANSPORTEURB, TRANSPORTEURC (fig. A8).

Le TRANSPORTEURA reçoit les colis de la paqueteuse et les charge sur le TRANSPORTEURB.

Quand il y a un nombre suffisant de colis sur le TRANSPORTEURB pour former une couche, celle-ci est transférée par le TRANSPORTEURC sur le diplodocus. Chaque transporteur doit s'arrêter et attendre jusqu'à ce que le suivant soit libre: le TRANSPORTEURA s'arrête lorsqu'il y a une couche complète sur le TRANSPORTEURB, lequel attend jusqu'à ce que le TRANSPORTEURC ait transféré la couche précédente sur un emplacement convenable du diplodocus.

Chaque ligne d'amenée est paramétrée par une "Fiche d'Affectation des Lignes d'Amenée"(FALA), composée des informations suivantes (fig. A7) :

PALETTISEUR - numéro du palettiseur auquel la couche est destinée (1, 2 ou 3);
LIGNEAMENEE - numéro de la ligne (1 à 16) ;
LIGNESORTIE - numéro de la ligne de sortie (après les palettiseurs) ;
CARTON/PACK - type de colis ;
TEMPSPARCOLIS - détermine l'écartement entre colis sur le TRANSPORTEURB, c'est-à-dire le temps qu'il doit tourner sans recevoir un colis du TRANSPORTEURA ;
COLISPARCOUCHE - nombre de colis qui doivent s'accumuler sur le TRANSPORTEURB ;
COUCHESPARPALETTE - information repassée au PALETTISEUR.

L'ensemble des 16 FALA consitue la "Table d'Affectation des Lignes d'Amenée" (TALA).

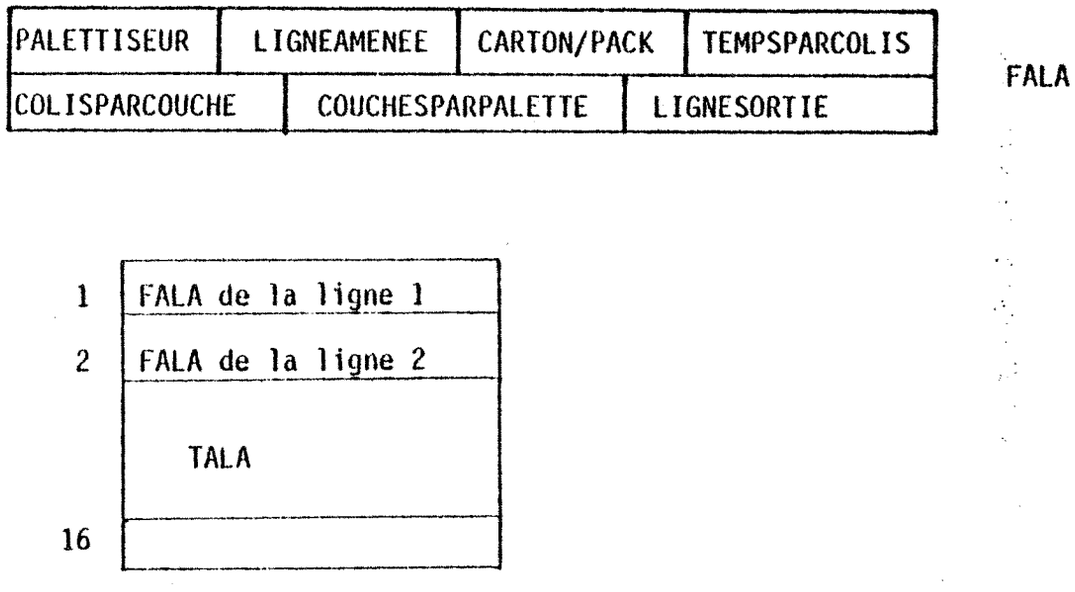


Fig. A7 - Fiche et Table d'Affectation des Lignes d'Amenée

Déclarations de structure

```
STRUCT FALA (PALETTISEUR, LIGNEAMENEE,  CARTON/PACK, TEMPSPARCOLIS,
             COLISPARCOUCHE, COUCHESPALETTE, LIGNESORTIE) ;
STRUCT TALA (16 FALA) ;
```

Le TRANSPORTEURB admet un état d'arrêt (attente), une basse vitesse, utilisée pour la réception colis par colis du TRANSPORTEURA, et une haute vitesse pour le transfert sur le TRANSPORTEURC. Il réalise donc un cycle :

colisparcouche

((arrêt ; basse vitesse) ; hautevitesse)*

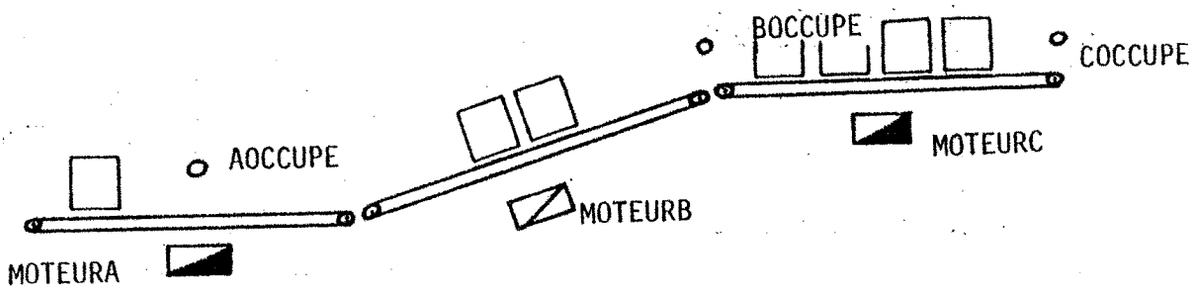


Fig. A8 - Ligne d'amenée

```
I TRANSPORTEURA (D: MOTEUR ; M: OCCUPE) ;
BOOL MOTEUR (TOURNE, ARRET), OCCUPE ;
FINS ;
```

```
S CONTRA (D: CYCLE, COUCHECOMPLETE ; M: AOCCUPE);
CS: TRANSPORTEURA (D: MOTEURA ; M: AOCCUPE) ;
CF : MOTEURA = CYCLE . (COUCHECOMPLETE . AOCCUPE)' ;
& le TRANSPORTEURA tourne en permanence, sauf s'il est occupé et il y a une
& couche complète sur le TRANSPORTEURB.
DIR: AOCCUPE ;
FINS ;
```

```
I TRANSPORTEURB (D: MOTEUR ; M:OCCUPE, HAUTEURCOLIS) ;
BOOL MOTEUR (ARRET, LENT, VITE), OCCUPE, HAUTEURCOLIS (TROPHAUT) ;
FINS ;
```

```
S CONTRB (D: AOCCUPE, COCCUPE, FALA, RESET ; M: TRANSFERTCOUCHEBC,
          COUCHECOMPLETE, COUCHETROPLONGUE, PROBLEMECOMPTAGE, PACKTROPHAUT);
CS: TRANSPORTEURB (D: MOTEURB ; M: BOCCUPE, HAUTEURCOLIS) ;
CF:
& Compteur de couches sur le TRANSPORTEURB, comparaison avec COUCHECOMPLETE
CTR COMPTCOLIS(I: AOCCUPE' ; R: TRANSFERTCOUCHEBC + RESET) ;
COUCHECOMPLETE = (COMPTCOLIS EQ COLISPARCOUCHE DE FALA) ;
& Attente des colis du type PACK ; s'il n'y a pas une couche complète sur le
& TRANSPORTEURB lors de l'arrivée d'un colis PACK sur AOCCUPE, alors il
& attend TEMPSPARCOLIS avant de tourner.
RET ATTENTEPACK (S: AOCCUPE!.COLISPACK.COUCHECOMPLETE';T:TEMPSPARCOLIS DE FALA,
                 R: AOCCUPE) ;
& Intervalle entre colis du type CARTON, si la couche n'est pas complète ;
& TRANSPORTEURB   tourne, TRANSPORTEURA est arrêté.
TMR INTERVALLECARTON (S:(AOCCUPE.COLISCARTON)!.COUCHECOMPLETE'.
    T: TEMPSPARCOLIS DE FALA, R:BOCCUPE!.(COUCHECOMPLETE.COCCUPE)) ;
& Le colis est-il du type PACK ou CARTON ?
COLISPACK = (CARTON/PACK DE FALA) EQ PACK ;
COLISCARTON = (CARTON/PACK DE FALA) EQ CARTON ;
& Transfert d'une couche du TRANSPORTEURB  sur le  TRANSPORTEURC.
TRANSFERTCOUCHEBC = COUCHECOMPLETE. COCCUPE' ;
& Commande du moteur
MOTEURB = CAS
          SI (ATTENTEPACK+COUCHECOMPLETE.COCCUPE.BOCCUPE'+INTERVALLECARTON)
             ALORS LENT ;
          SI TRANSFERTCOUCHEBC ALORS VITE ;
          SINON ARRET ;
          FINCAS  ;
& Conditions d'exception
COUCHETROPLONGUE = COUCHECOMPLETE'.BOCCUPE;
PROBLEMECOMPTAGE = COLISCARTON.COUCHECOMPLETE.(COMPTCARTON EQ COLISPARCOUCHE DE
                   FALA)' ;
CTR COMPTCARTON (I: BOCCUPE, R: TRANSFERTCOUCHEBC') ;
DIR: PACKTROPHAUT = HAUTEURCOLIS ;
FINS ;
```

```
I TRANSPORTEURC (D: MOTEUR ; M:OCCUPE) ;
BOOL MOTEUR, OCCUPE ;
FINS ;
```

```
S CONTRC (D: TRANSFERTCOUCHEBC, CHARGERDIPLO ; M: COCCUPE) ;
CS : TRANSPORTEURC (D: MOTEURC ; M: COCCUPE) ;
CF : MOTEURC = TRANSFERTCOUCHEBC + CHARGERDIPLO ;
& TRANSFERTCOUCHEBC: transfert de la couche du TRANSPORTEURB au TRANSPORTEURC ;
& CHARGERDIPLO: transfert de la couche du TRANSPORTEURC au DIPLODOCUS ;
DIR: COCCUPE ;
FINS ;
```

```
I PUPITRELIGNEAMENEE (M: DEMARRERCYCLE, ARRETCYCLE, RESET) ;
IMPULSION  DEMARRERCYCLE, ARRETCYCLE, RESET ;
FINS ;
```

```
S LIGNEAMENEE (D: CHARGERDIPLO,FALA,NUMLIGNE,ARRETDIPLOMAN ; M:COCCUPE,FC,
               VERROUILLERPAQUETEUSE) ;
CS: CONTRA (D: CYCLE, COUCHECOMPLETE ; M: AOCCUPE),
CONTRB (D: AOCCUPE, COCCUPE, FALA, RESET; M: TRANSFERTCOUCHEBC,
        COUCHECOMPLETE, COUCHETROPLONGUE, PROBLEMECOMPTAGE, PACKTROPHAUT),
CONTRC (D: TRANSFERTCOUCHEBC, CHARGERDIPLO; M:COCCUPE),
IMPRIMANTE (D: IMPRIMER (MESSAGE)),
PUPITRELIGNEAMENEE (M: DEMARRERCYCLE, ARRETCYCLE, RESET);
CF:
& Comptage des couches,destinéés à une même palette, transférées au diplodocus.
CTR COMPTCOUCHES (I: COUCHECOMPLETE ; R:(COCCUPE')! (COMPTCOUCHES EQ
                    COUCHESPARPALETTE DE FALA) ;
& Si un ordre de ARRETCYCLE arrive pendant qu'un transfert au diplodocus est
& en cours, il est mémorisé jusqu'à la fin du transfert et ensuite exécuté.
BIT ORDREARRETMEMORISEE (S: ARRETCYCLE.CHARGERDIPLO ; R: CHARGERDIPLO') ;
```

```
BIT CYCLE (S: DEMARRERCYCLE ; R: ORDREARRETMEMORISEE'+COUCHETROPLONGUE+
      PROBLEMECOMPTAGE+PACKTROPHAUT+(ARRETCYCLE!.CHARGERDIPLO')) ;
& Génération de la FICHE COUCHE (FC)
/COUCHECOMPLETE/FC := (COMPTCOUCHES = COUCHESPARPALETTE DE FALA), (CARTON/PACK,
      COLISPARCOUCHE, PALETTISEUR) DE FALA, COMPTCOUCHES ;
& Conditions d'exception
VERROUILLERPAQUETEUSE = CYCLE' ;
/CHARGEDILPO.ARRETDIPLOMAN!/IMPRIMER("URGENT DIPLO ARRET PENDANT TRANSFERT
      DE LA LIGNEAMENEE:", NUMLIGNE) ;
/COUCHETROPLONGUE/IMPRIMER("COUCHE TROP LONGUE LIGNE: ", NUMLIGNE) ;
/PROBLEMECOMPTAGE/IMPRIMER("PROBLEME DE COMPTAGE LIGNE: ", NUMLIGNE) ;
/PACKTROPHAUT/IMPRIMER("PACK NON BANDEROLLE LIGNE: ", NUMLIGNE) ;
DIR: COCCUPE, AOCCUPE, RESET, MOTEURB, FALA ;
FINS ;
```

2.3. Le diplodocus

Le diplodocus (fig. A9) est un convoyeur en mouvement circulaire permanent. Il est divisé en 22 loges dont chacune peut recevoir une couche d'une des lignes d'amenée et la tranférer sur une des trois lignes d'alimentation des palettiseurs par activation de l'écharpe respective.

L'écharpe est un éjecteur qui pousse une couche du diplodocus vers la ligne d'alimentation.

Le contrôleur détermine la position angulaire du diplodocus à l'aide d'un compteur POSITION mis à zéro par le capteur SYNC situé au début de la loge 1 et incrémenté par des impulsions issues du capteur CODE.

Chaque ligne d'amenée et d'alimentation du palettiseur est située respectivement à LM[numéro de la ligne d'amenée] et EM[numéro de la ligne d'alimentation palettiseur] impulsions du capteur SYNC. Ce sont des valeurs fixes qui permettent au contrôleur de connaître la position de chaque loge par rapport à ces lignes.

Chaque loge a une longueur d'approximativement 4m, dont 3m peuvent transporter une couche et 1m est réservé aux mouvements de l'écharpe. La détection d'un colis sur la zone d'écharpage entraîne un message d'erreur et l'arrêt du système. La décision de l'introduction d'une couche sera prise au moment où la première loge libre, et affectée au même palettiseur que la couche à évacuer, se présente devant la ligne d'amenée.

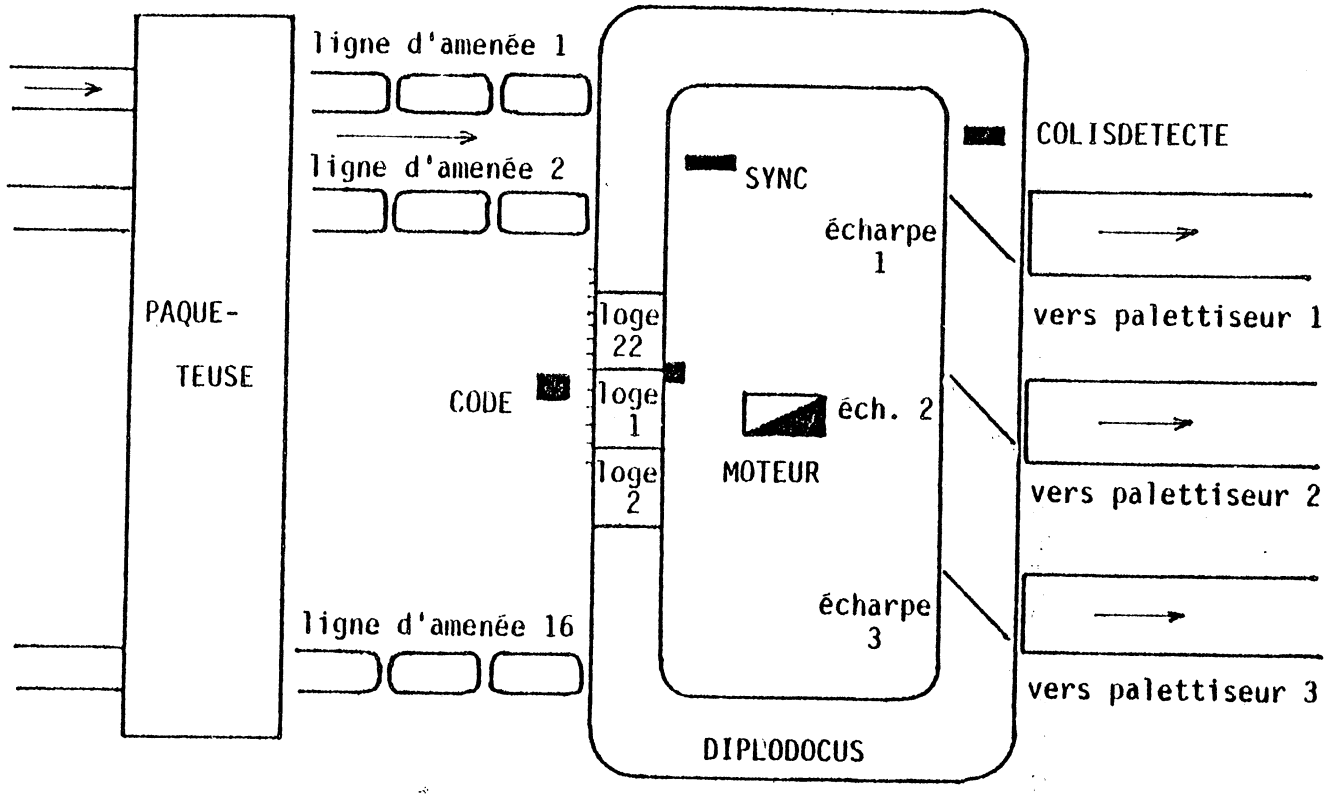


Fig. A9 - Le Diplodocus

```
I DIPLODOCUS (D: MOTEUR; M: CODE, SYNC, COLISDETECTE);
BOOL MOTEUR (MARCHE), COLISDETECTE (OBSCUR);
IMP CODE, SYNC;
FINS;
```

```
I PUPITREDIPLO (M: DEMARRERCYCLE, ARRETCYCLE, RESET; VIDANGEMAN,
                ARRETURGENCE);
IMP DEMARRERCYCLE, ARRETCYCLE, RESET, ARRETURGENCE;
& ARRETURGENCE = coupure de l'alimentation
BOOL VIDANGEMAN;
FINS;
```

```
S CONTRDIPLO ( D: PROBLEMECHARPE, DIPLOTROPCHARGE; M: DIPLOOPERATIONNEL,
               CYCLE, ARRETURGENCE, VIDANGEMAN);
CS: DIPLODOCUS (D: MOTEUR; M: CODE, SYNC, COLISDETECTE),
    PUPITREDIPLO (D: DEMARRERCYCLE, ARRETCYCLE, RESET, VIDANGEMAN,
                  ARRETURGENCE),
    IMPRIMANTE (D: IMPRIMER (MESSAGE));
CF:
& Détermination de la position angulaire
CTR POSITION (I: CODE; R: SYNC);
& Démarrage: le diplodocus devient opérationnel au premier SYNC, et
& 10 s après le démarrage.
BIT CYCLE (S: DEMARRERCYCLE; R: ARRETCYCLE + COLISSURZONEECHARPAGE + RESET +
               PROBLEMECHARPE + PROBLEMEROTATION);
MOTEUR = CYCLE;
RET TEMPSDEMARRAGE (S: DEMARRERCYCLE; T: 10 s; R: DIPLOOPERATIONNEL);
BIT DIPLOOPERATIONNEL (S: SYNC.TEMPSDEMARRAGE; R: CYCLE');
& Conditions d'exception
GDE PROBLEMEROTATION (S: CODE.DIPLOOPERATIONNEL; T: <temps> ;
                      A: CODE.DIPLOOPERATIONNEL; R: DEMARRERCYCLE);
ZONEECHARPAGE = (...< POSITION <...) + (...< POSITION <...) + ...
COLISSURZONEECHARPAGE = COLISDETECTE.ZONEECHARPAGE;
/COLISSURZONEECHARPAGE/ IMRRIMER ("ARRET DIPLO, COLIS SUR REGION D'ECHARPAGE");
/DIPLOTROPCHARGE/ IMPRIMER ("DIPLODOCUS TROP CHARGE");
DIR: ARRETURGENCE, VIDANGEMAN;
FINS;
```

```
I ECHARPE (D: ECHACTIVE; M: ARRIERE,,AVANT, SURVEILLANCE);
BOOL ECHACTIVE (AVANCE);
BOOL ARRIERE, AVANT, SURVEILLANCE ;
FINS;
```

```
S CONTRECHARPE (D: ECHACTIVE, NUMECHARPE, CYCLEDIPLO; M: PROBLEMECHARPAGE);
CS: ECHARPE (D: ECHACTIVE; M: ARRIERE, AVANT, SURVEILLANCE),
    IMPRIMANTE (D: IMPRIMER(MESSAGE));
CF:
& Fonctions de surveillance
GDE PROBLEMEAVANCE (S: ECHACTIVE; T: <tempsmax>; A: AVANT; R: CYCLEDIPLO);
/PROBLEMEAVANCE/ IMPRIMER ("ECHARPE ", NUMECHARPE, "PROBLEME D'AVANCE");
GDE PROBLEMERETOUR (S: ECHACTIVE'; T: <tempsmaxret>; A: ARRIERE;
                    R: CYCLEDIPLO);
/PROBLEMERETOUR/ IMPRIMER ("ECHARPE "; NUMECHARPE, "PROBLEME DE RETOUR");
& Un colis est considéré comme non écharpé s'il est détecté par le photo-
& détecteur SURVEILLANCE pendant le passage d'une loge qui vient d'être
& écharpée. Il y a alors un arrêt après un retard différent pour chaque
& écharpe, pour permettre au diplodocus de s'arrêter à un endroit accessible.
& Redémarrage après remise en état.
RET COLISNONECHARPE (S: SURVEILLANCE!.EHCACTIVE; T: <temps>; R:CYCLEDIPLO);
/COLISNONECHARPE/ IMPRIMER ("COLIS NON ECHARPE; ECHARPE N°", NUMECHARPE);
PROBLEMECHARPAGE = PROBLEMEAVANCE + PROBLEMERETOUR + COLISNONECHARPE;
DIR:ECHACTIVE;
FINS;
```

2.4. Contrôle du chargement et du déchargement du diplodocus

```
S CONTRCOUCHES (D: 3 DEMANDECOUCHE[*] , MAXLOGESOCC, TALA;
                M: VERROUILLERPAQUETEUSE[*] , TID, 3 ECHACTIVE[*],
                   3 NUMLOGEECHARPEE[*]  );
C8: 16 LIGNEAMENEE (D: CHARGERDIPLO[*] ; FALA[*] , NUMLIGNE[*] ,
          ARRETDIPLOMAN; M: COCCUPE[*] ; FC[*] DE TILA,
          VERROUILLERPAQUETEUSE[*] ),
    CONTRDIPLO (D: PROBLEMECHARPE, DIPLOTROPCHARGE; M: DIPLOOPERATIONNEL,
          CYCLEDIPLO, POSITIONDIPLO, ARRETURGENCE, VIDANGEMAN),
    3 CONTRECHARPE (D: ECHACTIVE[*] , NUMECHARPE[*] , CYCLEDIPLO;
          M: PROBLEMECHARPE[*]);
CF:
&   chargement d'une couche sur le diplodocus
& le TRANSPORTEURC se met en marche au moment où la tête de la loge à laquelle
& la couche est destinée se présente devant la ligne d'amenée; il s'arrête au
& moment où la fin de la zone fictive se présente devant la ligne ayant eté
& évacuée.
&
& Tableau avec 22 positions d'un bit qui indique si une loge est occupée:
22 BIT LOGEOCCUPEE[*] (S: LOGECHARGEE[*] ; R: LOGEECHARPEE[*] );
& Remplissage de la Table Image du Diplodocus (TID)
16 /CHARGERDIPLO[*] / FC[NUMLOGECHARGEE[*]]  DE TID := FC[*] DE TILA;
16 SEQ   CHARGEMENT DU DIPLODOCUS
INIT: REPOS[*]  JUSQUA COCCUPE[*] ;
      RECHERCHE[*]  JUSQUA LOGETROUVEE[*] ;
      CHARGERDIPLO[*]  JUSQUA ZONEFICTIVE[*] ';
   FINSEQ;
16 ALG LOGEACHARGER (D: RECHERCHE[*] , POSITIONDIPLO, NUMLIGNE[*],
       DIPLOOPERATIONNEL; M: LOGETROUVEE[*] , NUMLOGECHARGEE[*] ,
       ZONEFICTIVE[*] );
& Cet algorithme n'est pas détaillé; il connait la table LM, il utilise
& NUMLIGNE pour la détermination de la parité.
...
FINALG;
```

```
                        16
22 LOGECHARGEE[*] = OU (CHARGERDIPLO[i] (NUMLOGECHARGEE[i] = * ));
                       i:=1
& Surveillance
                        16
DIPLOTROPCHARGE = ( Σ   LOGEOCCUPEE[i] >MAXLOGESOCC);
                       i:=1
&
& Echarpage d'une couche du diplodocus - ligne d'alimentation palettiseur
&
& Palettiseurs 1, 2, et 3 en mode automatique
&
& Lorsqu'une    loge du diplodocus passe devant une écharpe dont la ligne d'ali-
& mentation palettiseur a demandé une couche, on examine si elle transporte une
& couche; le cas écheant on va examiner la fiche-couche, regarder sa destination,
& son repère de lot, etc... Si le déchargement est décidé, l'écharpe est comman-
& dée; sinon, on attend la prochaine loge.
& Repérage des lots: chaque fiche-couche FC possède un numéro de référence (NREF)
& qui lui est donné lors de sa constitution (établissement de la FC). Lors de
& l'examen de la FC avant la commande de l'écharpe, ce numéro de lot est examiné.
& Deux cas peuvent se produire:
& - la couche est une fin-de-lot (FINLOT EQ VRAI). On examine toutes les loges du
& diplodocus pour vérifier si une ou plusieurs couches possèdent un NREF identique.
& Si oui, on ne prend pas cette couche; si non, on la prend.
& - la couche n'est pas une fin-de-lot. On examine toutes les loges du diplodocus
& pour vérifier si une ou plusieurs couches possèdent un numéro de lot inférieur.
& Si oui, on ne prend pas cette couche. Si non, on la prend.
&
3 ALG LOGEAEVACUER  (D: TID, POSITIONDIPLO, DIPLOOPERATIONNEL, DEMANDECOUCHE[*];
                     M: ECHAUTO[*] , NUMLOGEECHARPEE[*] );
& Algorithme non détaillé
FINALG;
&
                         3
22 LOGEECHARPEE[*] = OU  (ECHACTIVE[k]. (NUMLOGEECHARPEE[j] = * ));
                        k:=1
```

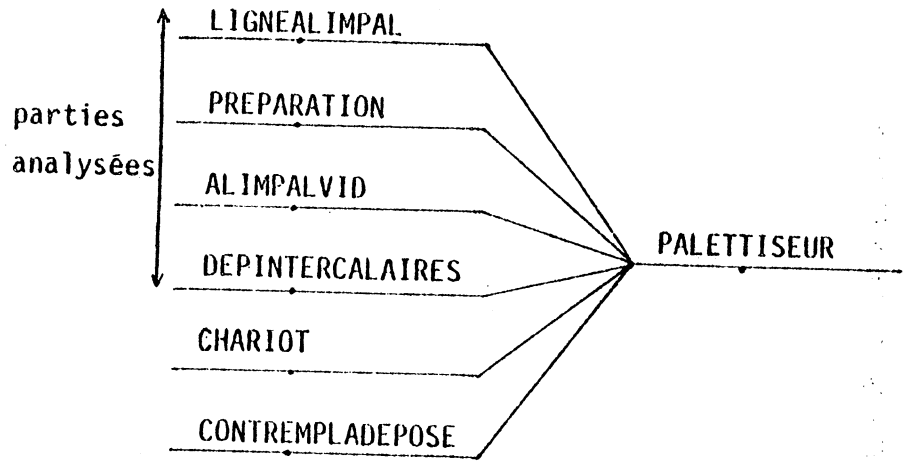
```
&
& Palettiseur 3 en vidange manuel (urgence)
&
ECHMAN[3]= VIDANGEMAN.CYCLEDIPLO';
&
2 ECHACTIVE[*] = ECHAUTO[*] ;
ECHACTIVE[3]  = ECHAUTO[3] + ECHMAN;
&
DIR: ARRETDIPLOMAN = ARRETURGENCE; 16 NUMLIGNE [*]= * ; PROBLEMECHARPE;
     3 NUMECHARPE[*] = * ; CYCLEDIPLO;
FINS;
```

3. LIGNES DE PALETTISATION

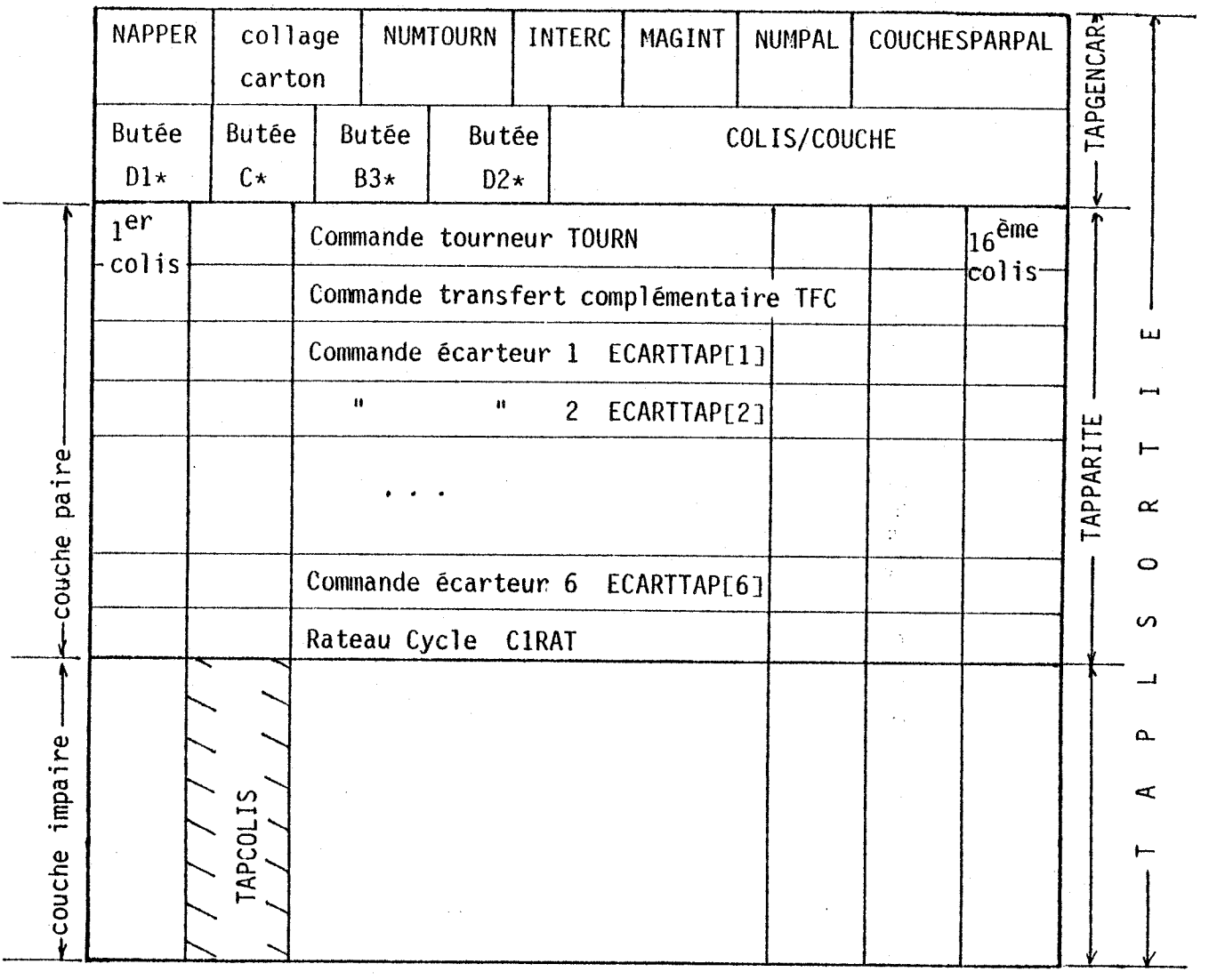
Les lignes de palettisation sont des dispositifs chargés d'empiler sur des palettes des couches de colis originaires du diplodocus. Il existe trois lignes de palettisation, dont deux intègrent des machines de palettisation automatiques et l'autre constitue une ligne de palettisation manuelle.

Les lignes de palettisation automatiques sont constituées des parties suivantes:
- ligne d'alimentation palettiseur: a la fonction de transporter les couches du palettiseur jusqu'à la table de préparation ;
- table de préparation; a la fonction de conformer les couches ;
- dépot d'intercalaires ;
- chariot mobile: dépose les couches sur la palette ;
- emplacement de dépose et accumulation des palettes pleines.

Le contrôleur des lignes de palettisation automatique a été structuré de la manière suivante :



Le fonctionnement est déterminé par des Tables d'Affectation du Palettiseur (TAP), dont il existe une par ligne de palettisation automatique (fig. A10). Cette table sert :
- à conformer la couche (elle contient le schéma de palettisation),
- à monter les palettes,
- à gérer les palettes.



* paramètres non-utilisés dans la partie du contrôleur décrite.

Fig. A10 - TAP : la Table d'Affectation du Palettiseur est constituée de 18 tables comme celle de la figure ; une par ligne de sortie LSORTIE (qui suivent le palettiseur).

Déclaration de la TAP

```
STRUCT TAP (18 TAPLSORT) ;
STRUCT TAPLSORT (TAPGENCAR, 2 TAPPARITE) ;
STRUCT  TAPGENCAR (NAPPER, collage carton, NUMTOURN, INTERC, MAGINT, NUMPAL,
        COUCHESPARPAL, butée D1, butée C, butée B3, butée D2, COLIS/COUCHE) ;
STRUCT TAPCOLIS (TOURN, TFC, 6 ECARTTAP, CIRAT) ;
STRUCT TAPPARITE (16 TAPCOLIS) ;
```

Algorithme d'accés à TAP (utilisé dans la description du service PALETTISEUR)

```
ALG
ACCES(X) = SI (X EQ TOURN) OU (X EQ TFC) OU (X EQ ECART[ ]) OU (X EQ CIRAT)
           ALORS X DE ARPCOLIS[NUMCOLIS] DE TAPPARITE [(COMPTCOUCHES MOD 2) +1]
                 DE TAPLSORT [LSORT DE TITP] DE TAP
           SINON X DE TAPGENCAR DE TAPLSORT [LSORT DE TITP] DE TAP ;
FINALG ;
&
& NUMCOLIS et COMPTCOUCHES sont des variables déterminées dans la description
& du service S PREPARATION (non décrit dans ce texte).
```

Lignes d'alimentation palettiseur

Les lignes d'alimentation du palettiseur (fig. A.11) sont constituées d'une séquence de "convoyeurs", appelés CONVALIMPAL, chacun activé par un moteur et ayant une cellule photoélectrique qui détecte la présence d'un colis.

```
I CONVALIMPAL (O: MOTEUR ; M: CELLULE) ;
BOOL MOTEUR (MARCHE), CELLULE (OBSCUR, CLAIR) ;
FINS;
```

Le transfert d'une couche d'un convoyeur sur le suivant est effectué si le suivant est libre. Un convoyeur est considéré comme libre quand le premier colis de la couche en cours de transfert a atteint la cellule photoélectrique du suivant. Deux transferts successifs se font sans arrêt intermédiaire.

Remarque :

Le convoyeur 1 est considéré comme libre après écoulement d'une temporisation déclenchée au début du transfert du convoyeur 1 sur 2 (environ 7s). Il est mis en marche lors du départ cycle (pupitre) de la ligne, et s'arrête à l'arrivée du premier colis de la couche sur sa cellule, si le convoyeur 2 est occupé.

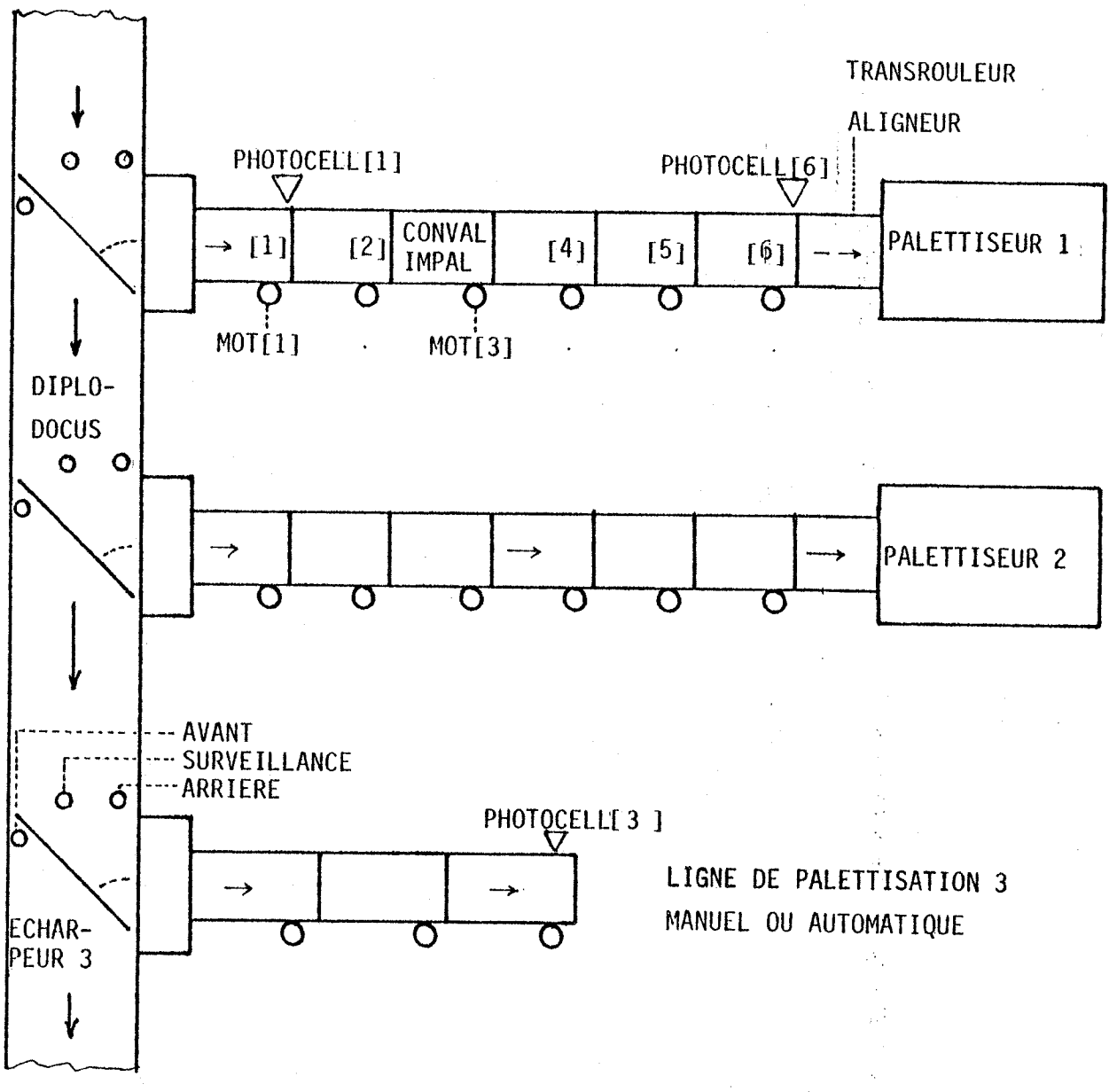
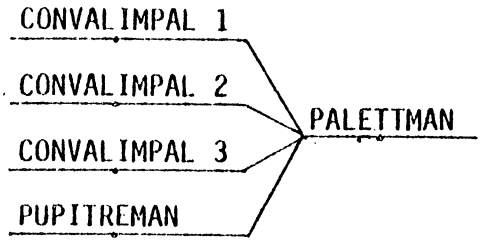


Fig. A 11 - Lignes d'alimentation palettiseurs.

Ligne de palettisation manuelle

Structure du contrôleur :

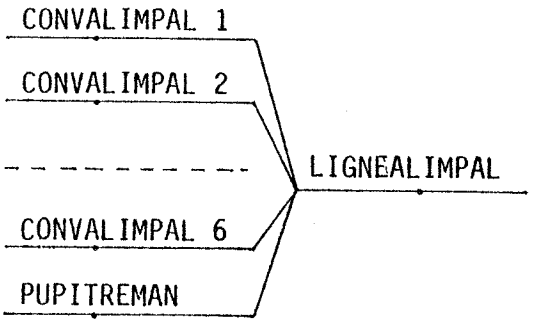



```
I PUPITREMAN (M: DEPCYCLE, ARRETCYCLE, MARCHEMANUELLE, APPELCOUCHE) ;
IMPULSION DEPCYCLE, ARRETCYCLE, APPELCOUCHE ;
BOOL MARCHEMANUELLE ;
FINS ;
```

```
S PALETTMAN (M: DEMANDECOUCHE) ;
CS: 3 CONVALIMPAL (D: MOT[*] ; M: PHOTOCELL[*]) ,
    PUPITREMAN (M: DEPCYCLE, ARRETCYCLE, MARCHEMAN, APPELCOUCHE) ;
CF:
BIT DCM (S: DEPCYCLE ; R: ARRETCYCLE) ;
& détermination de l'état libre ou occupé de chaque convoyeur
TMR TEMPSATTENTE (S: TRANSF12 ; T: 7s) ;
& le convoyeur 1 reste dans l'état OCC[1] jusqu'à 7s après le
& début du transfert sur le convoyeur 2
BIT OCC[1] (S: PHOTOCELL[1] ; R: TEMPSATTENTE') ;
BIT OCC[2] (S: PHOTOCELL[2] ; R: PHOTOCELL[3]) ;
BIT OCC[3] (S: PHOTOCELL[3] + POP ; R: APPELCOUCHE) ;
RECEPTION = DCM . OCC[1]' ;
TRANSF12 = DCM. OCC[1] . OCC[2]' ;
TRANSF23 = DCM. OCC[2] . OCC[3]' ;
MOT[1] = RECEPTION + TRANSF12 + FONCMAN;
MOT[2] = TRANSF12 + TRANSF23 + FONCMAN ;
MOT[3] = TPS + FONCMAN ;
TMR TPS (S: APPELCOUCHE ; T: 15s) ;
FONCMAN = DCM' . MARCHEMAN ;
DEMANDECOUCHE = OCC[1] ';       & demande couche au diplodocus
FINS;
```

Ligne de palettisation automatique

Structure du contrôleur :




```
I PUPITREAUT (M: DEPCYCLE, ARRETCYCLE, RAZ, MARCHEMAN, T1) ;
IMPULSION DEPCYCLE, ARRETCYCLE, RAZ ;
BOOL MARCHEMAN ;
REEL T1 ;
FINS ;
```

```
S LIGNEALIMPAL (D: BOURRAGE, PRECOMPT ; M: DEMANDECOUCHE, DCM, MARCHEMAN,TF[5]);
& BOURRAGE et PRECOMPT sont des signaux issus du contrôleur de la table de
& préparation (PREPARATION) ; T1 indique la temporisation de la marche du
& moteur du convoyeur 6.
CS : 6 CONVALIMPAL (D: MOT ; M: PHOTOCELL),
     PUPITREAUT (M: DEPCYCLE, ARRETCYCLE, RAZ, MARCHEMAN, T1) ;
CF :
BIT DCM (S: DEPCYCLE ; R: ARRETCYCLE) ;
& détermination de l'état libre ou occupé de chaque convoyeur
TMR  TEMPSATTENTE  (S: TRANSF[1] ; T: 7s) ;
& le convoyeur 1 reste dans l'état OCC[1] jusqu'à 7s après le début du
& transfert sur le convoyeur 2
BIT OCC[1] (S: PHOTOCELL[1] ; R: TEMPSATTENTE');
POUR I=2, 3, 4, 5
BIT OCC[I](S: PHOTOCELL[I] ; R: PHOTOCELL[I+1] + RAZ) ;
& le convoyeur 6 est considéré comme libre quand le premier colis a passé
& la cellule de précomptage PRECOMPT.
BIT OCC[6] (S: PHOTOCELL[6] ; R: PRECOMPT + RAZ) ;
& le transfert d'une couche sur la suivante est effectué si la suivante est
& libre.  TF[J] indique transfert du convoyeur J sur le convoyeur J+1.
```

```
        4 TF[*] = OCC[*] . OCC[*+1)' ;
& le convoyeur 5 s'arrête à l'arrivée du premier colis sur la cellule BOURRAGE.
BIT TF[5] (S: OCC[5] ; R: BOURRAGE) ;
RECEPTION = DCM. OCC[1]'              & réception par le convoyeur 1
MOT[1] = RECEPTION + TF[1] + FONCMAN ;
POUR K = 2, 3, 4, 5
MOT[K] = TF[K-1] + TF[K] + FONCMAN ; & le convoyeur roule quand il  reçoit
& une couche du précédent et quand il l'envoie sur le suivant.
& Convoyeur 6a :
TMR TEMPS (S: BOURRAGE ; T: T1)
MOT[6] = TF[5] + TEMPS + FONCMAN ;
& Fonctionnement manuel
FONCMAN = DCM' . MARCHEMAN ;
DEMANDECOUCHE = OCC[1]'  ;  & Demande une couche au diplodocus
FINS ;
& Le convoyeur 6 (avec le transrouleur aligneur - voir "Table de préparation")
& assure l'introduction de la couche sur le palettiseur.  Il permet la diminu-
& tion de vitesse nécessaire au passage du convoyeur 5 (0,6 m/s) au transrouleur
& d'orientation (0,3 m/s).
```

3.2. Table de préparation

La table de préparation (fig. A 12) sert à conformer les couches, selon la TAP.

Organisation du contrôleur :

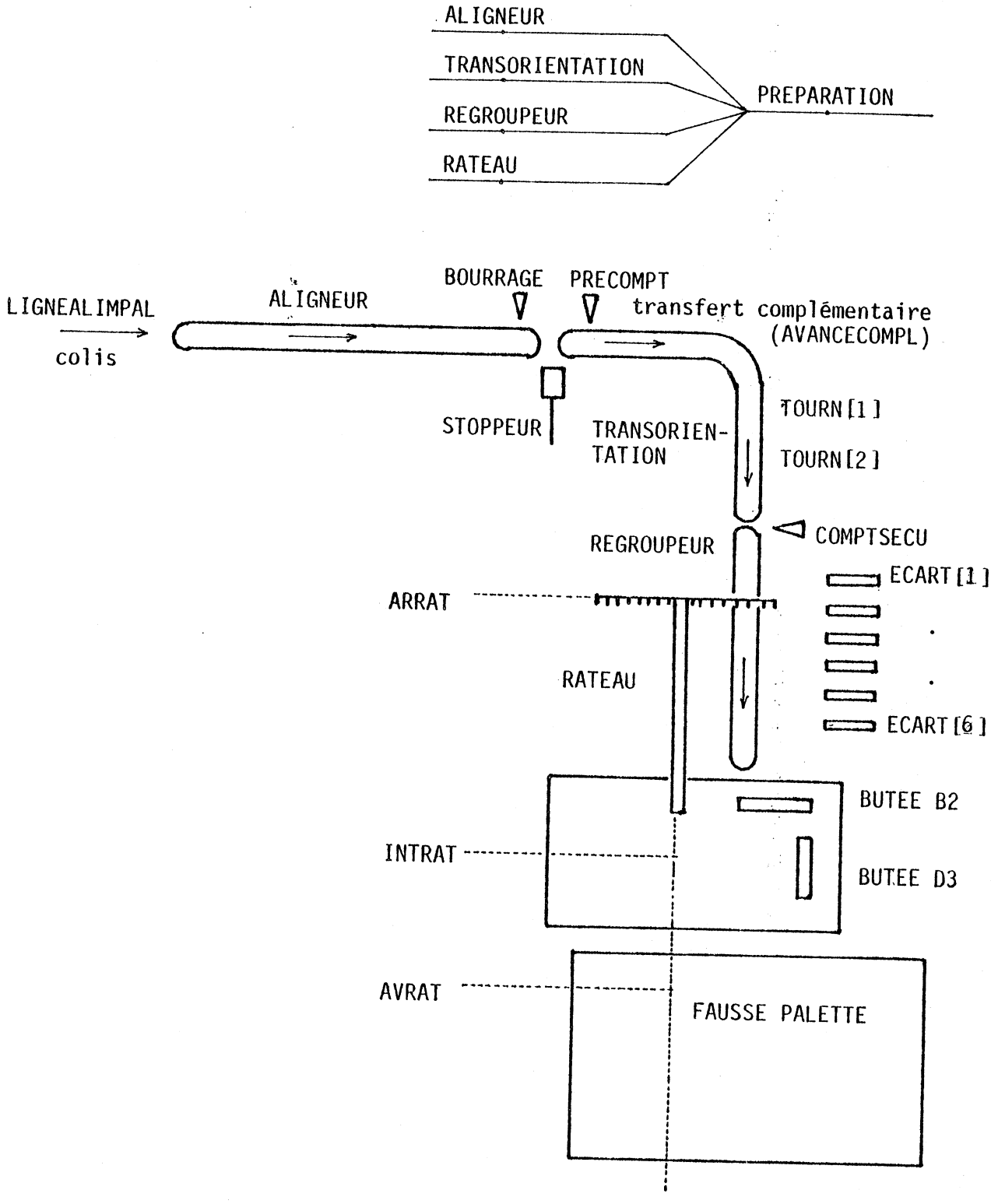


Fig. A 12 - Table de préparation du palettiseur.

Le transrouleur aligneur aligne les cartons contre une même face de référence et assure la régulation du flux de colis.

Le stoppeur assure la distribution des colis en fonction des appels machine. Il arrête les colis en position "HAUT" et autorise leur passage en "BAS".

```
I ALIGNEUR (D: MOTEUR, STOPPEUR ; M: BOURRAGE, PRECOMPT) ;
DISCR MOTEUR (ARRET, PETVIT, GDEVIT) ;
BOOL STOPPEUR (HAUT, BAS), BOURRAGE (OBSCUR, CLAIR) ;
BOOL PRECOMPT (CLAIR, OBSCUR) ;
FINS ;
```

Le transfert complémentaire (AVANCECOMPL) permet de placer deux colis côte à côte dans le sens de la longueur sur le transrouleur d'orientation. Il effectue un aller et retour pour chaque colis à transférer.

Les tourneurs 1 et 2 (TOURN[1] et TOURN[2] , resp) tournent les colis en position sortie, et laissent passer en longueur les colis en position rentrée.

Le "transrouleur d'orientation" (MOTEUR) transporte les colis dans le sens de la longueur, tournés ou transférés.

```
I TRANSORIENTATION (D: MOTEUR, 2 TOURN[*], AVANCECOMPL ; M: COMPLAVANT,
                    COMPLARRIERE) ;
BOOL MOTEUR (MARCHE), 2 TOURN[*], AVANCECOMPL, COMPLAVANT, COMPLARRIERE ;
FINS ;
```

Le"transrouleur regroupeur" (MOTREGROUP) regroupe les colis par rangées complètes devant le rateau.

Les écarteurs, au nombre de 6, permettent de créer un espace entre deux colis consécutifs. Ils sont escamotés entre les rouleaux du transrouleur regroupeur en position de repos (BAS) et émergent des rouleaux en position de travail (HAUT).

```
I REGROUPEUR (D: MOTEUR, 6 ECART[*]) ;
BOOL MOTREGROUP, 6 ECART[*](HAUT, BAS) ;
FINS ;
```

Le rateau transfère les rangées de colis du transrouleur regroupeur à la position intermédiaire. Il transfère les couches complètes de la position intermédiaire sur la fausse palette (fig. A 12 et A 13).

La butée B2 sert à faire une mise au format de la couche, au niveau de la position de stockage intermédiaire. Il effectue un aller-retour à chaque cycle.

La butée D3 a un fonctionnement identique à la précédente.

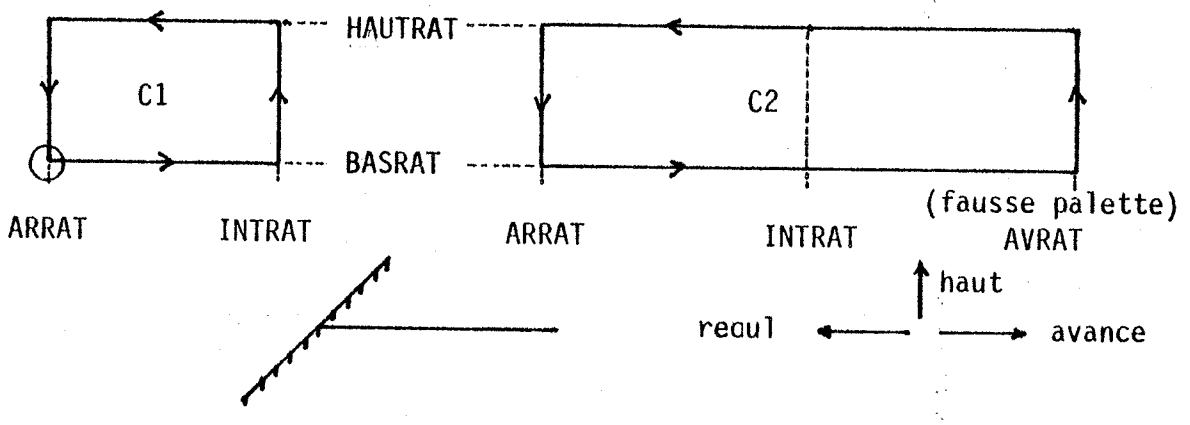


Fig. A 13 - Le rateau

```
I RATEAU (D: MVHOR, MVVERT, AVB2, AVD3 ; M: ARRAT, INTRAT, AVRAT, HAUTRAT,
          BASRAT, B2POSAV, B2POSARR, D3POSAV, D3POSARR, COMPTSECU) ;
DISCR MVHOR (AVANCE, RECUL, ARRET) ;
BOOL MVVERT (HAUT), AVB2 (HAUT), AVD3 (HAUT) ;
BOOL ARRAT, INTRAT, AV1RAT, AV2RAT, HAUTRAT, BASRAT, B2POSAV, B2POSARR ;
IMPULSION COMPTSECU ;
FINS ;
```

```
S PREPARATION (D: TF5ALIMPAL, DCM, TAP, TITP, FAUSSEPALLETTENPOS, B3POSARR,
       CPOSARR, DIPOSARR, D2POSARR ; M: ARRETCYCLE, PRECOMPT, BOURRAGE,COMPTCOUCHES);
CS: ALIGNEUR (D: MOTALIGN, STOPPEUR ; M: BOURRAGE, PRECOMPT),
TRANSORIENTATION (D: MOTORIENT, 2 TOURN[*], AVANCECOMPL ;
                  M: TFCAVANT, TFCARRIERE),
REGROUPEUR (D: MOTREGROUP, 6 ECART[*]),
RATEAU (D: MVHOR, MVVERT, AVB2, AVD3 ; M: ARRAT, INTRAT, AVRAT, HAUTRAT, BASRAT,
        B2POSAV, B2POSARR, D3POSAV, D3POSARR, COMPTSECU) ;
FC:
& Moteur de l'aligneur
& - démarrage en petite vitesse au départ cycle
& - arrêt par bourrage normal :
&     . détection par cellule BOURRAGE ; si BOURRAGE et stoppeur en position
&       haute: enclenchement d'un retard (environ 1,5s)
&     . retombée à zéro du retard si le stoppeur est baissé
&     . si écoulement du retard: arrêt du transrouleur aligneur jusqu'à la
&       descente du stoppeur
&     . redémarrage de l'aligneur après descente du stoppeur
& - conditions de marche en petite vitesse: transrouleur d'orientation marche.
& - grande vitesse: simultanément à TF[5] de la ligne d'alimentation palettiseur.
RET BOURRDEL (S: BOURRAGE! . STOPPEUR ; T: 1,5s ; R: STOPPEUR') ;
MOTALIGN = CAS
           SI (MOTORIENT . BOURRDEL') . DCM ALORS PETVIT ;
           SI TF5ALIMPAL . DCM ALORS GDEVIT;
           SINON ARRET ;
           FINCAS ;
& Stoppeur
& - en position haute au repos (hors tension)
& - descente au départ cycle de la machine (électrovanne maintenue sous tension)
& - ordre de montée: suivant matrice programme
& - ne doit jamais être monté pendant que PRECOMPT (CP2) occulté.
& Différents cas :
&                          montée                          descente
& (1) Arrière du dernier colis d'une      Dès que le rateau a quitté la position
&     rangée                              arrière
& (2) Arrière du colis à transférer       Transfert complémentaire revenu en
&                                         position arrière
```

```
& (3) Arrière du colis précédent à une      Temporisation constante
&     avance tourneur
& (4) Arrière du colis précédent un         Temporisation constante
&     recul tourneur
CTR NUMCOLIS (I: PRECOMPT', R: (NUMCOLIS EQ COLIS/COUCHE DE TITP ).AVRAT );
CTR COMPTCOUCHES(I: NUMCOLIS EQ 1 ; R: (COMPTCOUCHES EQ COUCHESPARPAL DE TAP).
      AVRAT!) ;
BIT DERNIERCOLISRANGEE (S: ACCES(CIRAT) ; R: ARRAT) ;
BIT ARRCOLISATRANSF (S: ACCES(TFC) ; R: TFCAVANT) ;
TMR ARRAVTOUR (S: PRECEDENTOURNER : T: <temps>) ;
TMR ARRECULTOUR (S: PRECEDENTOURNER' ; T: <temps>) ;
PRECEDENTOURNER = TOURN DE TAPCOLIS[NUMCOLIS+1] DE TAPPARITE[(COMPTCOUCHES
                  MOD 2)+1] DE TAPLSORT [LSORT DE TITP] DE TAP ;
STOPPEUR = DCM' + (DERNIERCOLISRANGEE +        & cas (1)
           ARRCOLISATRANSF +                   & cas (2)
           ARRAVTOUR +                         & cas (3)
           ARRECULTOUR).PRECOMPT' ;            & cas (4)
& Transfert complémentaire
& - en position arrière au repos (électrovanne hors tension)
& - avance par mise sous tension sur information de la TAP à la fin du colis
&   à transférer sur PRECOMPT ;
& - retour quand fin de course TFCAVANT atteint.
    SEQ  TRANSFERTCOMPLEMENTAIRE
INIT: $REPOS JUSQUA ARRCOLISATRANSF ;
      $AVANCETFC JUSQUA TFCAVANT ;
      $RETOURTFC JUSQUA TFCARRIERE ;
    FINSEQ ;
AVANCECOMPL = $AVANCETFC ;
& Tourneurs 1 et 2
& - rentrée en position repos (électrovanne hors tension) ;
& - sortie suivant matrice TAP, ordre de sortie donné par l'arrière du premier
&   colis à tourner ;
& - ordre de rentrée à l'arrière du premier colis à placer dans le sens de
&   la longueur.
BIT COLISATOURNER (S: PRECOMPT!.ACCES(TOURNTAP) ; R: PRECOMPT!.ACCES(TOURNTAP)');
2 TOURN[*] = (NUMTOURN DE TAP EQ * ).COLISATOURNER ;
```

```
& Transrouleur d'orientation
& - tourne en permanence en fonctionnement normal
& - arrêt temporisé lorsqu'on fait "arrêt cycle" machine ;
& - arrêt à l'information "avance transfert complémentaire ; redémarrage au
&   retour du transfert complémentaire en position arrière.
RET TAC (S: DCM' ; T: <temps> ; R: DCM) ;
MOTORIENT = (TAC + $AVANCETFC + $RETOURTFC)' ;
& Transrouleur regroupeur
& Regroupe les colis par rangée complète devant le rateau. Pendant l'avance du
& rateau le transrouleur regroupeur est arrêté. Il sera redémarré dès que le
& rateau aura atteint sa position intermédiaire. Pendant le recul du rateau,
& ainsi que pendant son avance entre la position intermédiaire et la position
& avant, une nouvelle rangée peut être constituée sur le transrouleur regroupeur.
BIT RANGEECOMPLETE (S: COMPTSECU!.DERNIERCOLISRANGEE ; R:  AVANCE1RAT') ;
MOTREGROUP = DCM.RANGEECOMPLETE' ;
& Ecarteurs
& Position basse au repos ; l'écarteur à monter reçoit l'ordre à l'arrière du
& colis précédant celui qu'il aura à retenir ; cet ordre ne sera exécuté qu'après
& un retard propre à chaque écarteur.  L'écarteur redescend au redémarrage du
& transrouleur regroupeur.
6 RET RETECARTEUR[*] (S: COMPTSECU! . ACCES(ECARTTAP[*]); T: ctemps[*] ;
      R: MOTREGROUP) ;
6 ECART[*] = RETECARTEUR[*] ;
& Rateau
& - Cycle C1: transférer les rangées de colis du transrouleur regroupeur sur
&   la position de stockage intermédiaire.
& - Cycle C2: transférer la dernière rangée d'une couche en position intermé-
&   diaire et cette couche sur la fausse palette ;
BIT DEMANDEC1 (S: RANGEECOMPLETE.(NUMCOLIS ≠ COLIS/COUCHE DE TITP); R: $MONTEE) ;
BIT DEMANDEC2 (S: RANGEECOMPLETE.(NUMCOLIS EQ COLIS/COUCHE DE TITP);R:$ARRET) ;
CYCLE = (DEMANDEC1 + DEMANDEC2). MOTREGROUP' ;
```

```
            SEQ RATEAU
INIT : $ARRIEREBASSE JUSQUA CYCLE ;
       $AVANCE1RAT JUSQUA INTRAT ; ALLERA SI DEMANDEC1 ALORS $MONTEE ; & C1,C2
       $ARRET JUSQUA FINCONFORMATION . FAUSSEPALETTENPOS ;             & C2
       $AVANCE2RAT JUSQUA AVARAT ;                                     & C2
       $MONTEE JUSQUA HAUTRAT ;                                        & C1,C2
       $RECUL JUSQUA ARRAT ;                                           & C1,C2
       $DESCENTE JUSQUA BASRAT ;                                       & C1,C2
            FINSEQ ;
MVHOR = CAS
        SI ($AVANCE1RAT + $AVANCE2RAT) ALORS AVANCE ;
        SI ($RECUL.HAUTRAT ALORS RECUL ;
        SINON ARRET ;
        FINCAS ;
MVVERT = SI $MONTEE ALORS HAUT SINON BAS ;
& Butées B2 et D3
& Arrière en position repos ; avance quand le rateau arrive en position inter-
& médiaire avec une couche complète ; recul dès position avant atteinte.
        SEQ  B2                                    SEQ  D3
INIT: REPOSB2 JUSQUA INTRAT ;          INIT: REPOSD3 JUSQUA INTRAT ;
        BUTEEB2 JUSQUA B2POSAV ;                   BUTEED3 JUSQUA D3POSAV ;
        COUCHEB2 JUSQUA COUCHECONFORMEE;           COUCHED3 JUSQUA COUCHECONFORMEE;
        FINSEQ ;                                   FINSEQ ;
COUCHECONFORMEE = COUCHEB2.COUCHED3 ;
AVB2 = BUTEEB2 ;
AVD3 = BUTEED3 ;
FINS ;
```

3.3.Alimentation de palettes vides

Architecture du procédé :

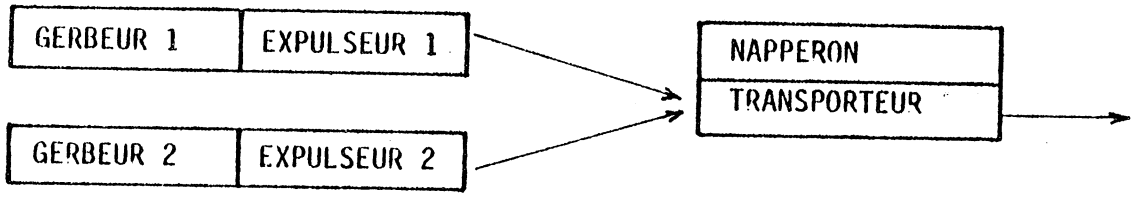


Fig. A 14 - Architecture du procédé

Organisation du contrôleur :

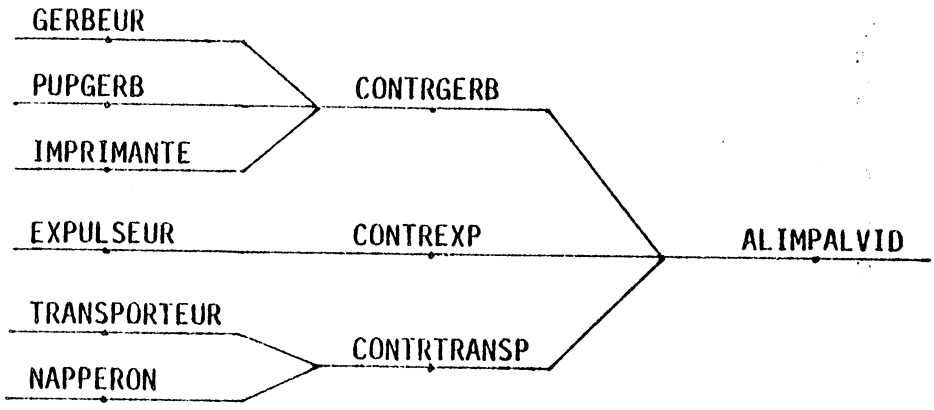


Fig. A 15 - Gerbeur de palettes

Gerbeur de palettes: au nombre de deux, ils réceptionnent les palettes vides déposées manuellement, et les déposent une à une sur l'expulseur de palettes vides.

```
I GERBEUR (D: TRANSL, MONTEE, DESCENTE ; M: HAUT, MILIEU, BAS, AVANT, ARRIERE,
           MANQUEPAL) ;
DISCR TRANSL (AVANCE, RECUL, ARRET) ;
BOOL MONTEE, DESCENTE, HAUT, MILIEU, BAS, AVANT, ARRIERE, MANQUEPAL ;
FINS ;
```

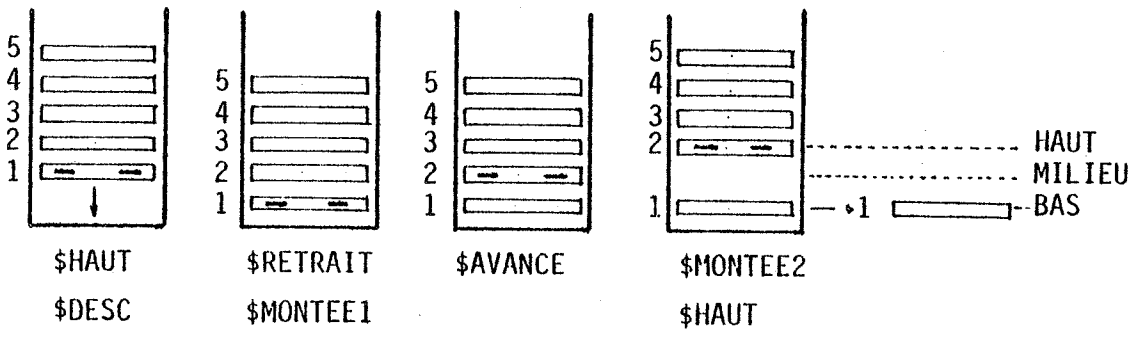


Fig. A 15 - Fonctionnement du gerbeur de palettes

```
I PUPGERB (D: AVERT ; M: BPACQ) ;
BOOL AVERT ;
IMPULSION BPACK ;       & bouton poussoir acquittement de défaut.
FINS ;
```

```
S CONTRGERB (D: GERB, PALSUREXP ; M: HAUT, AVANT) ;
CS: GERBEUR (D: TRANSL, MONTEE, DESCENTE ; M: HAUT, MILIEU, BAS, AVANT, ARRIERE,
                MANQUEPAL) ,
    PUPGERB (D: AVERT ; M: BPACQ),
    IMPRIMANTE (IMPRIMER(MESSAGE)) ;
CF:
    SEQ GERBEUR
INIT: $HAUT CAS JUSQUA GERB ; ALLERA $DESC ;
                JUSQUA GERBVIDE ; ALLERA $DEFAUTA ; FINCAS ;
      $DESC JUSQUA BAS ; ALLERA SI PALSUREXP ALORS $RETRAIT SINON $DEFAUTB ;
      $RETRAIT JUSQUA ARRIERE ;
      $MONTEE1 JUSQUA MILIEU ;
      $AVANCE JUSQUA AVANT ;
      $MONTEE2 JUSQUA HAUT ; ALLERA $HAUT ;
      $DEFAUTA JUSQUA MANQUEPAL' ; ALLERA $HAUT ;
      $DEFAUTB JUSQUA BPACQ ; ALLERA $RETRAIT ;
    FINSEQ ;
```

```
/$DEFAUTA/IMPRIMER ("IL MANQUE DES PALETTES") ;
AVERT = $DEFAUTA ;
GERBVIDE = MANQUEPAL ;    & Le magasin ne contient que 3 palettes.
TRANSL = CAS
       SI $RETRAIT ALORS RECUL ;
       SI $AVANCE ALORS AVANCE ;
       ELSE ARRET ;
       FINCAS ;
MONTEE = $MONTEE1+$MONTEE2 ;
DESCENTE = $DESC ;
DIR: AVANT, HAUT ;
FINS ;
```

Expulseur :

Au nombre de deux, ils poussent la palette déposée par le gerbeur sur le transporteur d'alimentation palettes vides, et reviennent en position arrière (fig. A 16).

```
I EXPULSEUR (D: MOTEXPULS ; M: PALSUREXP, EJAV, EJARR) ;
DISCR MOTEXPULS (AVANCE, RECUL, ARRET) ;
BOOL PALSUREXP, EJAV, EJARR ;
FINS ;
```

```
S CONTREXP (D: EXPULSION, VERROU, DEMARRAGETRANSPORTEUR ; U: PALSUREXP, EJAV,
            EJARR) ;
CS: EXPULSEUR (D: MOTEXPULS ; M: PALSUREXP, EJAV, EJARR) ;
CF: SEQ EXPULSEUR
    INIT: $ARRIERE JUSQUA EXPULSION ;
          $AVANCE JUSQUA EJAV ;
          $ATTENTEAVANT JUSQUA DEMARRAGETRANSPORTEUR ;
          $RECUL JUSQUA EJARR;
           FINSEQ ;
```

```
MOTEXPULS = CAS
            SI VERROU'. AVANCE ALORS AVANCE ;
            SI VERROU'. RECUL ALORS RECUL ;
            SINON ARRET ;
            FINCAS ;
DIR: PALSUREXP, EJAV, EJARR ;
FINS ;
```

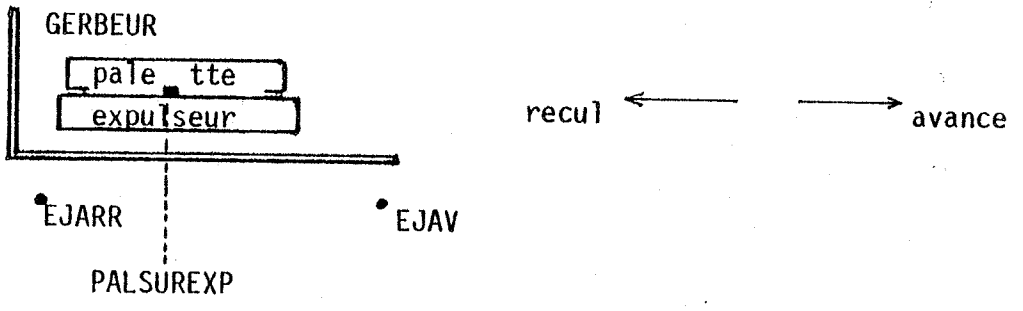


Fig. A 16 - Expulseur

Le transporteur d'alimentation de palettes vides (TRANSPORTEUR) transporte les palettes vides en provenance de l'un des gerbeurs, vers le chariot de palettisation et revient en position arrière.

```
I TRANSPORTEUR (D: MOTEUR ; M: ATTENTECHAR, POSAV, POSARR) ;
DISCR MOTEUR (AVANCE, RECUL, ARRET) ;
BOOL ATTENTECHAR, POSAV, POSARR ;
& ATTENTECHAR - palette en position d'attente devant chariot
& POSAV - transporteur en position avant
& POSARR - transporteur en position arrière
FINS ;
```

Le poste de dépose des napperons (fig. A 17) se trouve au-dessus du transporteur d'alimentation palettes vides. Il permet de déposer un napperon en film polyéthylène sur la palette vide qui passe sur le transporteur d'alimentation. Il est composé d'un dévidoir de film (MOTNAPP), d'une pince de retenue MAINTNAPP et d'une résistance chauffante (FILSEP) assurant la coupe des napperons à la longueur requise.

Dans le cas d'un besoin de napperon (NAPPER DE TAP) la palette passant devant la cellule "détection déroulement napperon" (CELLULENAPP) actionne le déroulement du napperon. Dès que la cellule n'est plus occultée, la palette s'arrête, le déroulement s'arrête, le napperon est coupé et la palette redémarre jusqu'à la position d'attente devant le chariot mobile.

```
I NAPPERON (O: MOTNAPP, FILSEP, MAINTNAPP ; M: CELLULENAPP, NAPPFERME, MANQUENAPP);
BOOL CELLULENAPP(OBSCUR), NAPPFERME, MANQUENAPP, MOTNAPP, FILSEP, MAINTNAPP ;
FINS ;
```

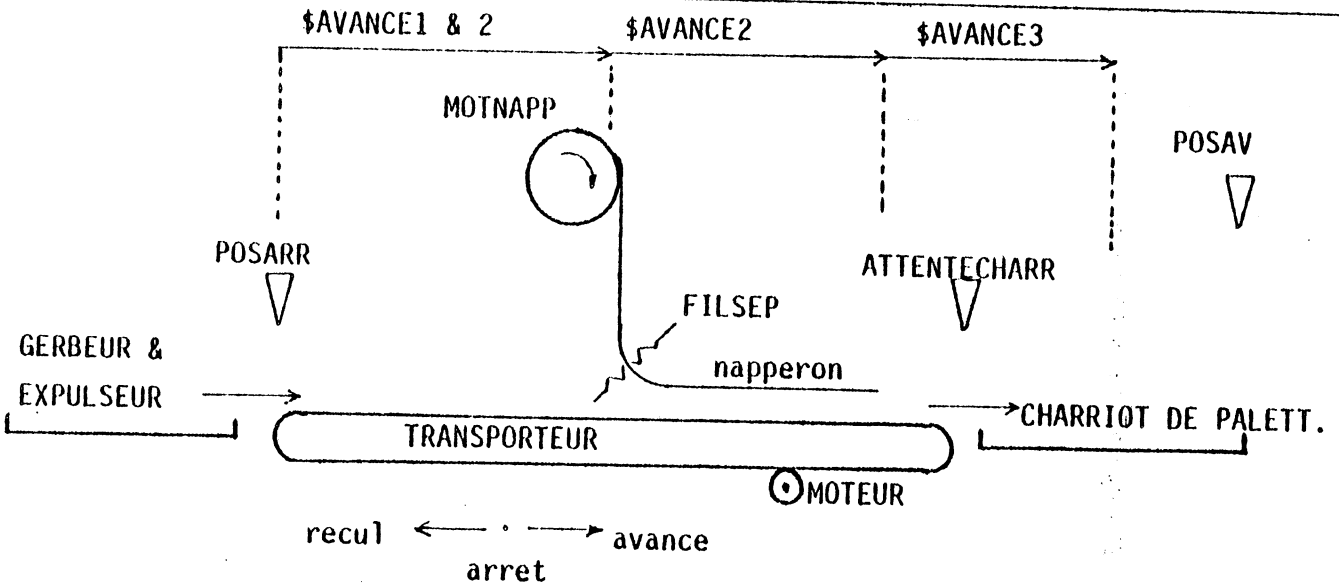


Fig. A 17 - Transporteur et napperon

```
S CONTRTRANSP (O: TRANSPORTER, TAP, CHARDEPART, TRANSPALVIDCHARMARCHE ;
               M: DEMARRTRANSP, POSARR) ;
CS: TRANSPORTEUR (O: MOTEUR ; M: ATTENTECHARR, POSAV, POSARR),
    NAPPERON (O: MOTNAPP, FILSEP, MAINTNAPP ; M:CELLULENAPP, NAPPFERME, MANQUENAPP);
```

```
CF: SEQ TRANSPORTEUR
      INIT: $ARRETE CAS JUSQUA TRANSPORTER.COUCHENAPP ; ALLERA $AVANCE1 ;
                        JUSQUA TRANSPORTER.COUCHENAPP' ; ALLERA $AVANCE2 ; FINCAS
                $AVANCE1 JUSQUA $DEROULEMENT' ;
                $ATTENTE0 JUSQUA $FERMETURE';
                $AVANCE2 JUSQUA ATTENTECHARR ;
                $ATTENTE1 JUSQUA CHARDEPART.TRANSPALVIDCHARMARCHE ;
                $AVANCE3 JUSQUA POSAV ;
                $ATTENTE2 PENDANT 0,5s ;
                $RECUL JUSQUA POSARR ;
      FINSEQ ;
MOTEUR = CAS SI $AVANCE+$AVANCE2+$AVANCE3 ALORS AVANCE ;
      SI $RECUL ALORS RECUL ;
      SINON ARRET ; FINCAS ;
   SEQ NAPPERON
   INIT: $ARRETNAPP JUSQUA NAPP ;
    $MARCHENAPP JUSQUA CELLULENAPP' ;
    $DEROULEMENT PENDANT <temps> ;
    $FERMETURE PENDANT <tempsT1> ;
   FINSEQ ;
MOTNAPP = $MARCHENAPP  + $DEROULEMENT ;
MAINTNAPP = FILSEP  *  $FERMETURE ;
NAPP = CELLULENAPP!.(NAPPER DE TAP) ;
DEMARRTRANSP = POSARR' ;
DIR: POSARR ;
FINS ;
```

```
S ALIMPALVID (D: TAP, COMPTCOUCHES, TRANSPALVIDCHARMARCHE, CHARDEPART) ;
CS: 2 CONTRGERB (D: CYCGERB[*], PALSUREXP[*]; M: GERBHAUT[*], GERBAV[*]),
    2 CONTREXP (D: CYCEXP[*], VERREXP[*], DEMARRAGETRANSPORTEUR ;
        M: PALSUREXP[*], EJAV[*], EJARR[*]),
    CONTRTRANSP (D: TRANSPORTER, TAP, CHARDEPART, TRANSPALVIDCHARMARCHE ;
                M: DEMARRAGETRANSPORTEUR, POSARRTRANSP) ;
CF: & démarrage d'un cycle gerbeur dès que l'expulseur a effectué un aller
    & et retour
```

```
2 CYCGERB[*] = EJARR[*] ;
& Demarrage d'un cycle expulseur au passage sur PRECOMPT de l'aligneur de la
& table de préparation du premier colis de la première couche destinée à la
& palette à expulser.
2 CYCEXP[*] = (COMPTCOUCHES EQ 1).(NUMPAL DE TAP EQ *) ;
2 VERREXP[*] = (GERBHAUT[*].GERBAV[*].POSARRTRANSP.PALSUREXP[*].EJARR[3-*]) ;
& Demarrage d'un cycle transporteur: à l'arrivée en position avant de l'un
& des deux expulseurs de palettes.
TRANSPORTER = EJAV[1] + EJAV[2] ;
DIR: TAP, TRANSPALVIDCHARMARCHE, CHARDEPART, PALSUREXP, DEMARRAGETRANSPORTEUR ;
FINS ;
```

3.4.Dépose d'intercalaires

Eléments constitutifs (fig. A 18) :

a/ deux magasins d'intercalaires, chargés chacun d'intercalaires en carton d'une dimension différente. L'intercalaire à déposer sera déterminé en fonction de la largeur de la palette à laquelle il est destiné.

b/ chariot de translation et de dépose intercalaire: transfère les intercalaires des magasins un à un sur les couches en attente sur la fausse palette. La translation est assurée par un chariot entraîné par moteur électrique, la montée et la descente par un vérin pneumatique, la préhension par un bras équipé de ventouses à dépression.

Fonctionnement :

Les magasins d'intercalaires sont alimentés manuellement par un préposé. A la demande d'un intercalaire, le chariot s'avance à vitesse lente jusqu'en dessus du magasin choisi et le bras descend. Pendant la descente, on injecte de l'air comprimé par les ventouses pour éviter l'aspiration des poussières. Dès que les ventouses sont en contact avec l'intercalaire, la pompe à vide se met en marche. Quand la dépression est atteinte, le bras remonte. Arrivé en position haute, le chariot avance à petite vitesse et vient se placer en-dessus de la fausse palette. Si la couche est prête, le bras descend jusqu'au contact avec la couche, la pompe à vide s'arrête et on injecte de l'air comprimé par les ventouses pour décoller l'intercalaire. Le bras remonte. Arrivé en position haute, le chariot recule à grande vitesse jusqu'à sa position arrière extrême.

Une cellule détecte le manque d'intercalaire et avertit le préposé. Dans le cas où le bras est descendu plus de deux fois pour prendre un intercalaire, et que la dépression n'est pas atteinte, ou dans le cas où l'on perd un intercalaire pendant un transfert, le cycle intercalaire est interrompu et la couche partira sans intercalaire. Le chariot de dépose intercalaire revient à sa position de repos.

Organisation du contrôleur :


```
I CHARIOT (D: TRANSL, POMPE, MONTEE, DESCENTE, SOUFLASP, KLAXON ; M: MAG1VIDE,
      MAG2VIDE, BRARR, BRMAG1, BRMAG2, BRDEP, BRHAUT, BRBAS, DEPR) ;
DISCR TRANSL (AVLENT, ARRVITE, ARRET) ;
BOOL POMPE, MONTEE, DESCENTE, MAG1VIDE, MAG2VIDE, DEPR, SOUFLASP (SOUFL, ASP) ;
& MONTEE et DESCENTE sont des électrovannes ;
IMPULSION BRARR, BRMAG1, BRMAG2, BRDEP, BRHAUT, BRBAS ;
FINS ;
```

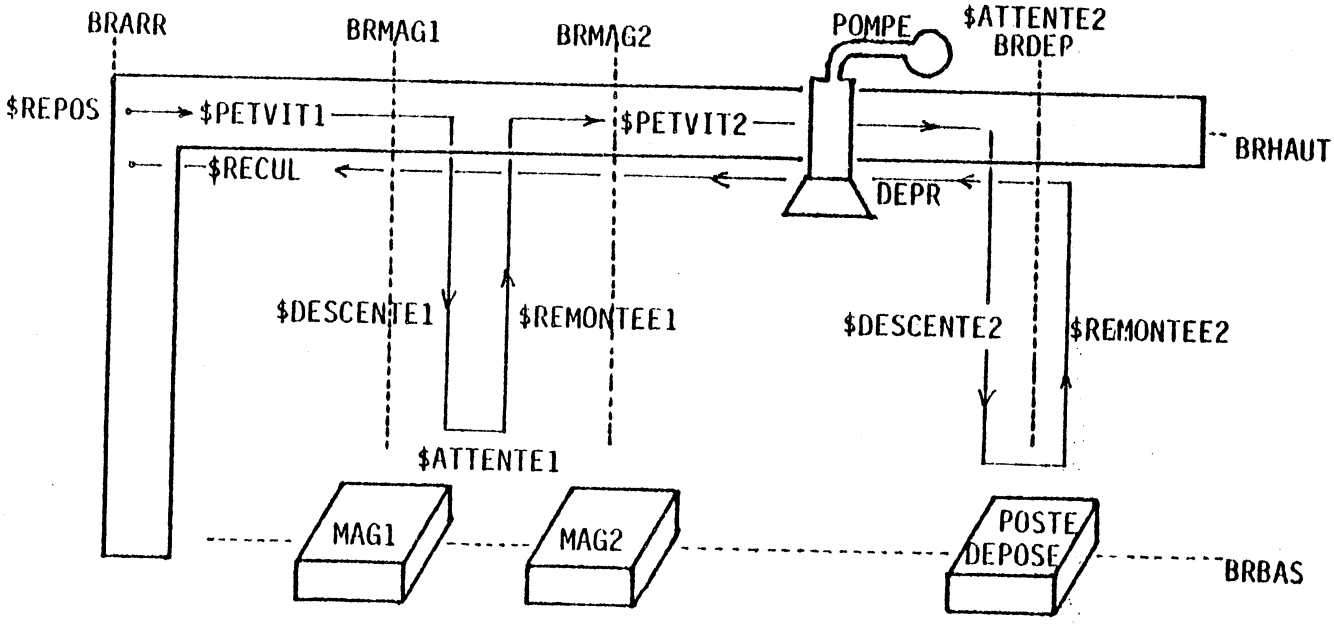


Fig. A 18 - Poste de dépose d'intercalaires

```
S INTERCAL (D: COUCHESSURCHARPAL, CYCINTERC) ;
CS: CHARIOT (D: TRANSL, POMPE, MONTEE, DESCENTE, SOUFLASP, KLAXON ; M: MAG1VIDE,
          MAG2VIDE, BRARR, BRMAG1, BRMAG2, BRDEP, BRHAUT, BRBAS, DEPR),
    IMPRIMANTE (D: IMPRIMER (MESSAGE)),
```

```
CF:
     SEQ
   INIT: $REPOS JUSQUA CYCINTERC ;  & arrière, haut
         $PETVIT1 JUSQUA CAS
                         JUSQUA (MAGINT DE TAP EQ 1) . BRMAG1! ;
                         JUSQUA (MAGINT DE TAP EQ 1)'.BRMAG2!  ;
                         FINCAS ;
         $DESCENTE1 JUSQUA BAS ;
         $ATTENTE1 JUSQUA DEPR ;
         $REMONTEE1 CAS JUSQUA DEPR' ; SI NUMPERTES<2 ALORS
                                $DESCENTE1 SINON $REMONTEE2 ;
                        JUSQUA BRHAUT ; FINCAS ;
         $PETVIT2 CAS JUSQUA DEPR' ; ALLERA $ATTENTE3 ;
                      JUSQUA BRDEP ; FINCAS ;
         $ATTENTE2 JUSQUA COUCHESURCHARPAL ;
         $DESCENTE2 JUSQUA BRBAS ;
         $REMONTEE2 JUSQUA BRHAUT ;
         $RECUL JUSQUA BRARR ; ALLERA $REPOS ;
         $ATTENTE3 PENDANT <temps> ; ALLERA $RECUL
     FINSEQ ;

TRANSL = CAS SI ($PETVIT1+$PETVIT2) ALORS AVLENT ;
             SI $RECUL ALORS ARRVITE ;
             SINON ARRET ; FINCAS ;
DESCENTE = $DESCENTE1 + $DESCENTE2 ;
SOUFLASP = POMPE' + $REMONTEE2 ;
POMPE = $ATTENTE1 + $REMONTEE1 + $PETVIT2 + $ATTENTE2 + $DESCENTE2 ;
MONTEE = $REMONTEE1 + $REMONTEE2 ;
& DEFAUTS :
& 1. Si magasin vide, klaxon et impression ;
& 2. Perte de l'intercalaire pendant le cycle, signalée par la perte de
&    la dépression.  Trois cas sont à considérer :
&    a/ la perte se produit pendant la montée ($REMONTEE1) :
```

```
&     - le bras termine son mouvement de montée,
&     - nouvelle descente, dépression et remontée avec l'intercalaire,
&     - si ce processus se renouvelle plus de deux fois consécutives, klaxon,
&       impression et retour en position arrière.
&    b/ la perte se produit pendant l'avance ($PETVIT2):
&     - arrêt de l'avance, klaxon et impression,
&     - temporisation, puis recul du bras en position arrière extrême.
&    c/ la perte se produit pendant la descente sur la couche: aucune conséquence.
& Dans tous les cas où il y a l'impression "défaut de dépose d'intercalaire",
& le palettiseur continue de fonctionner normalement, la couche étant déposée
& soit sans intercalaire, soit celui-ci est déposé par un préposé.
CTR NUMPERTES (I: DEPR!.$REMONTEE1 ; R: $PETVIT2) ;
PROBLEME = (DEPR'.NUMPERTES≥2)+MAG1VIDE+MAG2VIDE+DEPR'.$PETVIT2 ;
/PROBLEME/ IMPRIMER ("DEFAUT DE DEPOSE D'INTERCALAIRE") ;
TMR KLAXON (S: PROBLEME ; T: <temps>) ;
FINS ;
```

3.5. Palettiseur

```
S PALETTISEUR (D: TAP, TITP ; M: COMPTCOUCHES, DEMANDECOUCHE) ;
CS: LIGNEALIMPAL (D: BOURRAGE, PRECOMPT ; M: DEMANDECOUCHE, DCMALIMPAL,
                     MCAMAN, TF5ALIMPAL, DEMANDECOUCHE),
    PREPARATION (D: TF5ALIMPAL, DCMALIMPAL, TITP, FAUSSEPALETTENPOS, B3POSARR,
                    CPOSAV , CPOSARR, D1POSARR, D2POSARR ;
                 M: ARRETCYCLE, PRECOMPT, BOURRAGE, COMPTCOUCHES);
    ALIMPALVID (D: TAP, COMPTCOUCHES, TRANSPALVIDCHARMARCHE, CHARDEPART),
    INTERCAL (D: COUCHESURCHARPAL, CYCINTERC),
   *CHARIOT et CONTREMPLDEPOSE (M: FAUSSEPALETTENPOS, B3POSARR, CPOSARR,D1POSARR,
              D2POSARR, TRANSPALVIDCHARMARCHE, CHARDEPART, COUCHESURCHARPAL) ;
CF:
& Ordre de départ du cycle de dépose intercalaire donné par le passage sur
& PRECOMPT des premiers colis de la couche à laquelle l'intercalaire est destiné.
    CYCINTERC = (COMPTCOUCHES EQ 1)! . ACCESS(INTERC) ;
DIR: ...
FINS ;
```

*non décrits

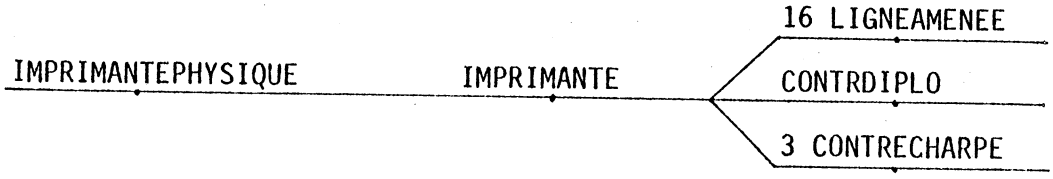
4 - GESTION DES IMPRIMANTES

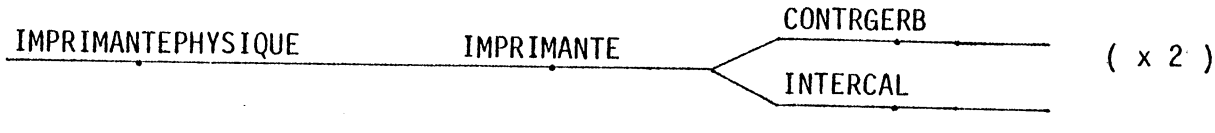
```
M IMPRIMANTE (D: IMPRIMER(MESSAGE));
CS: IMPRIMANTEPHYSIQUE (D: SOMMET(QIMPR), $IMPRESSION ; M: FINIMPR) ;
CF:
QUE QIMPR (LONGUEUR: NPOS messages de ... caractères) ;
/IMPRIMER.(QIMPR EQ PLEINE)'/QIMPR := MESSAGE ;

            SEQ
   INIT: REPOS JUSQUA (QIMPR EQ VIDE)' ;
         $IMPRESSION JUSQUA FINIMPR ;
         PRE (QIMPR) ;
            FINSEQ ;
& Il est spécifié que, si la queue QIMPR est pleine, les messages arrivant
& sont "oubliés". Une solution alternative serait de faire attendre l'utilisa-
& teur de l'imprimante jusqu'à ce qu'il y ait une place disponible dans la queue
FINS ;
```

```
I IMPRIMANTEPHYSIQUE (D: TAMPON, IMPRESSION ; M: FINIMPR) ;
CHAINE TAMPON ;
BOOL FINIMPR, IMPRESSION ;
FINS ;
```

Il existe dans le système trois imprimantes, partagées de la manière suivante:

```
PART IMPRIMANTE (16 LIGNEAMENEE, CONTRDIPLO, 3 CONTRECHARPE),
       2 IMPRIMANTE (CONTRGERB DE PALETTISEUR[*], INTERCAL DE PALETTISEUR[*]) ;
```





5 - SYSTEME (description)

```
S SYSTEME (D: MAXLOGESOCC, TALA, 2 TAP[*] ; 16 VERROUILLERPAQUETEUSE[*]) ;
CS: CONTRCOUCHES (D: 3 DEMANDECOUCHE[*], MAXLOGESOCC, TALA ;
                  M: 16 VERROUILLERPAQUETEUSE[*], TID, 3 ECHACTIVE[*],
                     3 NUMLOGEECHARPEE[*]) ,
    2 PALETTISEUR (D: TAP[*], TITP[*] ; M: COMPTCOUCHES[*] ; DEMANDECOUCHE[*]),
    PALETTMAN (D: DEMANDELIGNE3) ;
CF:
DEMANDECOUCHE[3] = DEMANDELIGNE3 ;
2 QUE QLAP[*](LONGUEUR: 6) ;
2/ECHACTIVE[*]/QLAP[*] := FC[NUMLOGECHARPEE[*]] de TID ;
2/(COMPTCOUCHES[*] = 1)/DEBUT TITP[*] := SOMMET(QLAP[*]) ; RET(QLAP) ; FIN ;

DIR: MAXLOGESOCC, TALA, 2 TAP[*], 2 DEMANDECOUCHE[*], 16 VERROUILLERPAQUETEUSE[*]
FINS ;
```

6 - CONCLUSION

L'application du langage CSL à la description d'un exemple industriel de dimension réaliste assure, dans une certaine mesure, son applicabilité pratique et une évaluation "juste" du formalisme proposé. En d'autres termes, il montre que le formalisme n'a pas été orienté pour s'adapter à une application de taille limitée ou de nature particulière, et d'autre part, l'exemple n'a pas été choisi de manière à mettre en évidence des caractéristiques "favorables" au formalisme.

REFERENCES

REFERENCES

[AFCE 77] Groupe de Travail Systèmes Logiques de l'AFCET
"Pour une représentation normalisée du cahier de charges d'un automatisme logique"
Automatique et Informatique Industrielles, n°61 et 62, novembre 1977.

[AFCE 80] F. ANDRE , D. HERMAN, J.P. VERJUS
"Contrôle du parallélisme et de la répartition"
Ecole d'Eté de l'AFCET, Aix-en-Provence, juillet 1980.

[ALGO 68] Groupe ALGOL de l'AFCET
Manuel du langage algorithmique ALGOL 68
Ed. Hermann, Paris, 1975

[ANCE 78] F. ANCEAU
"Mécanismes primitifs de synchronisation au niveau matériel"
2nd Colloque International sur les Systèmes d'exploitation
IRIA, 2/4 octobre 1978.

[ANCE 78] F. ANCEAU, J. BORDIER
"A syntatic method for programming simple industrial control applications with microprocessors"
4th EUTOMICRO Symposium on Microprocessing and Microprogramming,
Munich, October 1978.

[BACK 78] J. BACKUS
"Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs"
CACM, vol. 21, n°8, août 1978.

[BARB 75] M.R. BARBACCI
"A comparison of register transfer languages for describing computers and digital systems"
IEEE Trans. Comput., vol C-24, n°2, février 1975.

[BRHA 75] P. BRINCH HANSEN
"The programming language concurrent PASCAL"
IEEE Trans. Soft. Engin., vol. SE 1, n°2, 199/207; 1975.

[BRHA 78] P. BRINCH HANSEN
"Distributed processes: a concurrent programming concept"
CACM, vol. 21, n° 11, novembre 1978.

[CAMP 74] R.H. CAMPBELL
"The specification of process synchronization by path expressions"
Décembre 1973

[CERT 74] CENTRE D'ETUDES ET DE RECHERCHE DE TOULOUSE
"TEAU: Techniques et Exploitation de l'Assignation Unique"
Février 1974.

[DIJK 68] E.W. DIJKSTRA
"Cooperating sequential processes"
in Programming languages, F.Genuys (Ed.), Academic Press,
New-York, 1968

[DENN 79] J.B.DENNIS
"The varieties of data flow languages"
Proc. 1st International Conf. on Distributed Computing systems,
Huntsville, Alabama, octobre 1979.

[DIJK 75] E.W. DIJKSTRA
"Guarded commands, nondeterminacy, and formal derivation of programs"
Comm. ACM 18, 8 , août 1975, pp. 453/457.

[DIJK 76] E.W. DIJKSTRA
"A discipline of programming"
Prentice Hall, N.J. 1976.

[GERT 75] J. GERTLER, J. SEDLAK
"Software for process control - A survey"
Automatica, vol. 11,1975, pp. 613/625.

[GREE 79] "Reference manual for the Green programming language
rationale for the design of the green"
CII HB, Louveciennes, 1979.

[GRIE 76] P.D. GRIEM
"Approaching an easy-to-learn method of programming
real-time parallel processes"
Proceedings of the IFAC IFIP Workshop on Real time programming,
Rocquencourt, 1976.

[HART 66] J. HARTMANIS, R.E. STEAMS
"Algebraic structure theory of sequential machines"
Englewood Cliffs, NJ, Prentice Hall, 1966.

[HATV 73] J. HATVANY (Ed.)
"Computer languages for numerical control"
1971

[HOAR 74] C.A.R. HOARE
"Monitors: an operating system structuring concept"
Comm. ACM 17, 10, octobre 1974.

[HOAR 78] C.A.R. HOARE
"Communicating sequential processes"
Comm. ACM, 21, 8, août 1978.

[INTE 73] 8008 Users Manual
INTEL Corp, novembre 1973

[KEED 78] J.L. KEEDY
"On structuring operating systems with monitors"
The Australian Computer Journal, vol.10, n°1, février 1978.

[KESS 77] J.L. W. KESSELS

"A conceptual framework for a nonprocedural programming language"
CACM 20, 12, décembre 1977.

[LAMP 78] L. LAMPORT
"Time, clocks, and the ordering of events in a distributed system"
CACM, vol.12, n°7, juillet 1978.

[MARQ 80] J.M. COSTA ALVES MARQUES
"MOSAIC: une méthodologie de conception pour les circuits système VLSI"
Thèse docteur-ingénieur Informatique, INPG, Grenoble, septembre 1980.

[McCL 65] E.J. McCLUSKEY
"Introduction to the theory of switching circuits"
New-York, McGraw-Hill, 1965.

[MEAL 55] G.G. MEALY
"A method for synthetizing sequential circuits"
The Bell System technical Journal, septembre 1955.

[MEND 75] H.G. MENDELBAUM
"Automata as structured tools for easy real time programming"
1975 IFAC/IFIP Workshop on real time programming, Boston, USA.

[NOGO 75] G. NOGUEZ
"Etude d'un modèle temporel des systèmes séquentiels"
Thèse de doctorat ès-sciences, Institut de Programmation,
Paris, septembre 1975.

[PETE 77] J.L. PETERSON
"Petri nets"
Computing surveys, vol.9, n°3, septembre 1977.

[REED 79] D.P. REED
"Synchronization with eventcounts and sequencers"
Comm. ACM 22, 2, pp. 115/123, 1979.

[ROBE 77] P. ROBERT, J.P. VERJUS
"Towards autonomous descriptions of synchronization modules"
Proc. IFIP Conf. 1977.

[ROSS 77] D.T. ROSS, K.E. SHOMAN
"Structured analysis for requirements definition"
IEEE Trans. on Soft. Eng., vol. SE-3, n°1, janvier 1977.

[SILV 78] M. SILVA SUAREZ
"Contributions à la synthèse programmée des automatismes logiques"
Thèse de docteur-ingénieur, INP Grenoble, juin 1978.

[TAKA 80] H. TAKAHASHI
"An automatic controller description language"
IEEE Trans. S.E., vol. SE-6, n°1, janvier 1980.

[UNGE 59] S.H. UNGER
"Hazards and delays in asynchronous sequential switching circuits"
IRE Trans. on Circuit theory, vol. CT-6, n°1, pp. 12/25, mars 1959.

[WALT 80] C. WALTER
"A structuring language for computer controlled multilevel systems"
Proceedings of the IFAC/IFIP Workshop on real time programming,
Pergamon Press, (Ed. V.Haase), GRAZ (Austria), 1980.

[WALT 81] C. WALTER
"An analysis of discrete process control languages according to the Mealy machine"
Proc. of the VII IFAC World congress, Kyoto, août 1981.

[WIRT 77a] N. WIRTH
"MODULA: a language for modular multiprogramming"
Software - Practice and experience, vol.7, 1977, pp. 3/35.

[WIRT 77b] N. WIRTH
"Toward a discipline of real-time programming"
CACM, vol.20, n°8, août 1977.

AUTORISATION DE SOUTENANCE

Vu les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

Vu le rapport de présentation de :

- Monsieur François ANCEAU, Professeur
- Monsieur Gérard VERROUST, Ingénieur C.N.R.S.

Monsieur W'A L T E R Claudio

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

Grenoble, le 3 juin 1981

Le Président de l'I.N.P.G.