Un langage de description et de programmation de systèmes de conduite de procédés industriels

Joël Pleyber

THESE

présentée à

Université Scientifique et Médicale de Grenoble

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE

Informatique

par

Joël PLEYBER

UN LANGAGE DE DESCRIPTION ET DE PROGRAMMATION DE SYSTEMES DE CONDUITE DE PROCEDES INDUSTRIELS.

Thèse soutenue le 10 mars 1978 devant la Commission d'Examen :

Président : L. BOLLIET

Examinateurs : P. DESCHIZEAUX
Ph. JORRAND
R. PERRET

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

Monsieur Gabriel CAU : Président
Monsieur Pierre JULLIEN : Vice Président

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	AMBLARD Pierre	Clinique de dermatologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	I.S.N.
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme.	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOU Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de pédiatrie et puériculture
	BELORIZKY Elie	Physique
	BERNARD Alain	Mathématiques pures
Mme.	BERTRANDIAS Françoise	Mathématiques pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques pures
	BEZEZ Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Clinique gastro-entérologique
Mme.	BONNIER Marie-Jeanne	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BOUTET DE MONVEL Louis	Mathématiques pures
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologique
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmocodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques pures
	CHARACHON Robert	Clinique oto-rhino-laryngologique
	CHATEAU Robert	Clinique de neurologie
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique

Mme.	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumophtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée (IUT I)
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	GAGNAIRE Didier	Chimie physique
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique générale
	KOSZUL Jean-Louis	Mathématiques pures
	KLEIN Joseph	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
Mme.	LAJZEROWICZ Janine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LE ROY Philippe	Mécanique (IUT I)
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Pierre	Sciences nucléaires
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Melle	LUTZ Elisabeth	Mathématiques pures
MM.	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Clinique cardiologique
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NOZIERES Philippe	Spectrométrie physique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET Jean	Semeiologie médicale (Neurologie)
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REVOL Michel	Urologie
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique (IUT I)

MM.	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme.	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS ASSOCIES

MM.	CRABBE Pierre	CERMO
	DEMBICKI Eugéniuz	Mécanique
	JOHNSON Thomas	Mathématiques appliquées
	PENNEY Thomas	Physique

PROFESSEURS SANS CHAIRE

Melle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Gilbert	Géographie
	BENZAKEN Claude	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique (IUT I)
	BUISSON René	Physique (IUT I)
	BUTEL Jean	Orthopédie
	COHEN ADDAD Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie
	CONTE René	Physique (IUT I)
	DELOBEL Claude	M.I.A.G.
	DEPASSEL Roger	Mécanique des fluides
	FONTAINE Jean-Marc	Mathématiques pures
	GAUTRON René	Chimie
	GIDON Paul	Géologie et minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biologie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et médecine préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme.	KAHANE Josette	Physique
MM.	KRAKOWIACK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique (IUT I)
	LUU DUC Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
Mme.	MINIER Colette	Physique (IUT I)
MM.	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Melle	PIERY Yvette	Physiologie animale

MM.	RAYNAUD Hervé	M.I.A.G.
	REBECQ Jacques	Biologie (CUS)
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme.	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme.	SOUTIF Jeanne	Physique générale
MM.	STIEGLITZ Paul	Anesthésiologie
	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	ARMAND Yves	Chimie (IUT I)
	BACHELOT Yvan	Endocrinologie
	BARGE Michel	Neuro-chirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamie
MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme.	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B) (Personne étrangère habilitée à être directeur de thèse)
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme.	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FAURE Gilbert	Urologie
	GAUTIER Robert	Chirurgie générale
	GIDON Maurice	Géologie
	GROS Yves	Physique (IUT I)
	GUIGNIER Michel	Thérapeutique
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	JALBERT Pierre	Histologie
	JULIEN-LAVILLAVROY Claude	O.R.L.
	KOLODIE Lucien	Hématologie
	LE NOC Pierre	Bactériologie-virologie
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MALLION Jean-Michel	Médecine du travail
	MARECHAL Jean	Mécanique (IUT I)
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT I)

MM.	NEGRE Robert	Mécanique (IUT I)
	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique (IUT I)
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B) (Personnalité étrangère habilité à être directeur de thèse)
	PEFFEN René	Métallurgie (IUT I)
	PERRIER Guy	Géophysique-Glaciologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD André	Hygiène et hydrologie (Pharmacie)
	RAMBAUD Pierre	Pédiatrie
	RAPHAEL Bernard	Stomatologie
Mme.	RENAUDET Jacqueline	Bactériologie (Pharmacie)
MM.	ROBERT Jean-Bernard	Chimie physique
	Romier Guy	Mathématiques (IUT B) (Personnalité étrangè habilité à être directeur de thèse)
	SCHAERER René	Cancérologie
	SHOM Jean-Claude	Chimie générale
	STOEBNER Pierre	Anatomie pathologie
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	DEVINE Roderick	Spectro physique
	HODGES Christopher	Transition de phases

Fait à SAINT MARTIN D'HERES, NOVEMBRE 1976.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Monsieur Philippe TRAYNARD : Président

Monsieur Pierre-Jean LAURENT : Vice Président

PROFESSEURS TITULAIRES

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

PROFESSEUR ASSOCIE

M.	ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

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

MAITRES DE CONFERENCES

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

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan	Automatique

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

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

Je remercie Monsieur BOLLIET, Professeur à l'I.U.T. de Grenoble, qui m'a fait l'honneur de présider le jury de cette thèse.

Je remercie Monsieur PERRET, Professeur à l'Institut National Polytechnique et Directeur du Laboratoire d'Automatique de Grenoble, qui a bien voulu s'intéresser à mon travail et a accepté de le juger.

Philippe JORRAND, Maître de Recherches au laboratoire Informatique et Mathématiques Appliquées de Grenoble, a été mon Directeur de thèse. Je tiens à lui exprimer ma gratitude pour ses encouragements, ses critiques, ainsi que pour la confiance qu'il m'a témoignée.

C'est grâce à Pierre DESCHIZEAUX, Chargé de Recherches au Laboratoire d'Automatique de Grenoble, que j'ai pu aborder et étudier les problèmes posés par l'utilisation de l'Informatique en Automatique. Je le remercie vivement pour les nombreuses discussions que nous avons eues sur ce sujet.

Una parte importante de este trabajo fué realizada en colaboración con Manuel SILVA, investigador en el Laboratorio de Automática de Grenoble. Yo no olvidaré la amistad que me brindó durante los dos años de trabajo en comun.

Madame DIAZ et Elisabeth DUBOIS ont fait en sorte que je puisse disposer le plus rapidement possible du texte dactylographié de cette thèse. Je les en remercie bien sincèrement.

Je remercie les membres du service de reproduction du C.I.C.G. , qui ont assuré la réalisation de ce document.

TABLE DES MATIERES

Chapitre I

INTRODUCTION

Le développement de l'informatique vers la réalisation de systèmes plus sûrs, moins coûteux, et décentralisés, permet celui de l'automatique industrielle vers la mise en place d'applications à fort degré d'intégration : il est constaté à l'heure actuelle que de nombreux problèmes méthodologiques se posent pour maîtriser la technologie désormais disponible. Ce travail est une contribution dans ce sens en ce qui concerne la description et la programmation sur ordinateur de systèmes de conduite de procédés industriels.

Les aspects méthodologiques de la conduite sont abordés dans le chapitre II. Un langage est proposé dans le chapitre IV, basé sur un modèle décrit dans le chapitre III. Enfin, une implémentation d'une version simplifiée du langage, adaptée au contexte des automatismes logiques, est décrite en V.

Les théories et les méthodes de l'automatique industrielle ont été développées dans deux domaines d'application distincts : les automatismes logiques et les automatismes de régulation continue. La distinction provient de la différence fondamentale en ce qui concerne les modélisations et les technologies de réalisation utilisées.

Les automatismes logiques regroupent des applications où le système de commande, à partir de signaux à niveau, de comptages et de temporisations, agit sur le procédé à l'aide de signaux à niveau ou impulsionnels. Les automatismes de régulation utilisent essentiellement des mesures et des actions variant de façon continue; le rappel historique fait au paragraphe suivant reflète cette distinction. Cependant, les progrès de l'électronique permettent maintenant la réalisation de systèmes de conduite programmés où les deux aspects sont intégrés, et il importe donc de rechercher des méthodes de description globale des algorithmes.

I.1. - RAPPEL HISTORIQUE

I.1.1. - Les automatismes logiques [Daclin 77]

En 1950, les constructeurs d'automatismes logiques ignoraient encore les termes "séquentiel" et "booléen" et aucune théorie ni langage de description fonctionnelle n'étaient utilisés dans la pratique pour la synthèse : le cahier des charges était rédigé dans le langage naturel puis traduit directement en schémas éclatés de réseaux de relais dynamiques électromécaniques. Cependant, le degré important de sécurité et de disponibilité de réalisations complexes comme les réseaux ferroviaires et les centrales thermoélectriques ont montré la grande efficacité du savoir-faire de cette époque.

A partir de 1960, l'essai a été fait de remplacer les relais dynamiques par des relais statiques à l'aide des semi-conducteurs : les progrès attendus n'eurent pas lieu pour des raisons techniques d'utilisation en milieu industriel.

Durant toute cette période, de nombreuses recherches furent effectuées en ce qui concerne la description et la synthèse. La méthode la plus connue est celle d'Huffman, brièvement décrite ci-après car elle reflète l'esprit de méthodes utilisées actuellement pour certaines réalisations en logique programmée. Elle est basée sur la spécification de l'automate de commande à partir d'équations de la forme :

$$Q(t+1) = f(X(t), Q(t))$$
$$Z(t) = g(X(t), Q(t))$$

où X(t) est le vecteur des entrées, Z(t) le vecteur des sorties, Q(t) l'état interne de l'automate, à l'instant t. Le processus consiste donc à ramener sous forme combinatoire l'évolution de l'automate en prenant en compte à chaque instant l'ensemble complet des entrées et des sorties. Le concepteur décrit l'automate à l'aide d'une table où les lignes représentent les états internes et les colonnes les différentes valeurs (2^n pour n entrées) du vecteur des entrées. La méthode s'est avérée impraticable

pour la synthèse à cause des dimensions trop importantes des tables, de l'impossibilité de décomposer de façon non globale l'automate en sous-automates, et de la perte de la signification physique intervenant quand des techniques de minimisation sont utilisées.

Vers 1970, les circuits intégrés de semi-conducteurs commencent à être introduits dans les automatismes logiques pour mettre en oeuvre des fonctions complémentaires de celles réalisées à l'aide de relais électromécaniques. Les progrès réalisés depuis en ce qui concerne le degré d'intégration des circuits, la sûreté et le coût ont conduit au développement des automates programmables, évoqués en I.2.1.

I.1.2. - Les automatismes de régulation [Foulard 77]

Les automatismes de régulation sont ceux pour lesquels l'activité essentielle est le contrôle en continu de flux d'énergie et de matières traversant le procédé. Ils concernent en particulier de grands systèmes industriels, principalement dans la chimie, la sidérurgie, le ciment, la production d'électricité et l'avionique. Ils sont souvent associés à des automatismes logiques, en général assez simples à décrire, tels que des procédures occasionnelles comme le démarrage ou l'arrêt d'une installation, ou exceptionnelles (détection et traitement des alarmes). Les systèmes de régulation sont réalisés à l'aide de chaînes analogiques, qui tendent à être remplacées par des machines informatiques.

Le rôle de l'automatique est d'élaborer des méthodes pour déterminer quelles actions doivent être effectuées sur les entrées du procédé afin de satisfaire des objectifs donnés sur les entrées et les sorties. Le problème est donc la détermination d'une représentation du fonctionnement du procédé à partir duquel puisse être défini le système de conduite. Il est généralement possible d'utiliser, autour d'un point de fonctionnement, des modèles linéaires à base d'équations différentielles (discrétisées quand un ordinateur est utilisé). Une tendance apparaît vers les années

55 vers l'adoption d'équations de la forme :

$$x(t) = Ax(t)+Bu(t)$$
$$y(t) = Cx(t)+Du(t) \qquad (1)$$

où x représente l'état du procédé, u ses entrées, y ses sorties, les matrices A, B, C et D ayant comme coefficients des fonctions de ceux des équations différentielles initiales. La motivation de l'adoption d'une représentation dans le domaine temporel (auparavant la représentation était transposée dans le domaine fréquentiel à l'aide des transformations de Laplace ou en Z) a été la recherche de méthodes pour prédire l'évolution des sorties dans un but d'optimisation. Les objectifs étant en général contradictoires (par exemple une consommation minimale d'énergie et une production maximale de produits finis), des recherches ont alors été effectuées pour définir et minimiser un critère sur ces objectifs. Le critère de type quadratique, étudié à partir de 1960, mais pour lequel des méthodes pratiques d'utilisation n'ont été mises au point qu'en 1971, a apporté les meilleurs résultats car il permet une synthèse plus aisée de l'algorithme de commande. La synthèse consiste à déterminer les coefficients des matrices de l'équation (1). Quand les lois de la physique qui régissent le procédé ne sont pas connues, les matrices sont calculées en faisant fonctionner le procédé et en effectuant une série de mesures sur les entrées et les sorties : c'est ce qu'on appelle l'identification qui fait appel à des techniques de recherche d'un écart minimum entre le fonctionnement réel et celui d'un modèle de référence idéalisé.

C'est l'utilisation possible de l'ordinateur qui a été la motivation de ces études. En effet, si la régulation monovariable (1 entrée et une sortie) peut encore tout aussi bien être réalisée à l'aide de chaînes analogiques, la complexité des calculs nécessite l'utilisation de l'ordinateur pour le cas multivariable (plusieurs entrées et plusieurs sorties).

L'ordinateur permet aussi de prendre en compte des facteurs d'ordre économique ou organisationnel : des études méthodologiques, évoquées dans

le chapitre II, sont menées depuis les années 70 dans ce sens pour définir des structures générales de systèmes de conduite : l'orientation a d'abord été prise vers des structures hiérarchisées mais diverses raisons énoncées au II.4.2. font qu'elle l'est davantage maintenant vers des structures décentralisées.

L'ordinateur a été utilisé d'abord vers les années 55 pour la surveillance et la détection des alarmes et c'est en 1959 qu'apparaît le premier système de conduite programmé : la raison de cette orientation est la possibilité d'avoir de nombreuses boucles de régulation de façon économique et une amplitude de contrôle autour d'un point de consigne plusieurs centaines de fois plus importante que celle obtenue avec des chaînes analogiques. Des raisons de sûreté (centralisation de la conduite, disponibilité insuffisante du matériel) ont limité, jusqu'à l'apparition des circuits intégrés l'emploi de l'ordinateur. La partie logique du système de conduite est en général soi câblée, soit programmée et alors liée au système d'interruptions. Il est maintenant admis que l'automatisation complète n'est pas souhaitable et que le fonctionnement doit avoir lieu sous la conduite d'un opérateur, d'où l'utilisation de ce terme de conduite pour qualifier le système aussi appelé de commande : le dialogue de celui-ci avec l'opérateur prend donc une importance particulière et la question de modification en ligne de l'application doit être examinée.

Enfin, les réalisations industrielles de système de conduite multivariab sont encore rares et l'industrie chimique en constitue à l'heure actuelle le domaine d'application le plus important.

I.2. - OBJECTIFS POURSUIVIS

Les objectifs seront décrits à partir de la démarche suivie pour arriver à la définition du langage proposé.

I.2.1. - L'élaboration de Systèmes de Conduite pour automatismes logiques

Le modèle d'automate défini dans le chapitre III a été étudié initialement pour servir de support à la programmation, basée sur une description fonctionnelle, de la partie logique (combinatoire et séquentielle) de systèmes complexes de production. Son étude se situe dans la suite des travaux effectués dans ce domaine au Laboratoire d'Automatique de Grenoble autour de procédés pilotes de distillation, décrits par exemple dans [Ramos 76] . Dans ce cadre, Lebourgeois [Lebourgeois 67] a étudié la programmation d'automates séquentiels à l'aide de tables de transition et la décomposition d'automates en sous-automates.Ramos-Niembro [Ramos 75] a utilisé des graphes d'états pou décrire la partie logique de la conduite du procédé pilote, dans le but de la réaliser à l'aide de C.U.S.A. (Cellules Universelles pour séquences asynchrones [David 69]); les automates décrits ont été câblés, leur coordination étant assurée par un programme. Enfin, Silva a étudié les méthodes [Silva 76 et les structures matérielles [Silva 77-2] liées au domaine voisin des automates programmables. C'est plus particulièrement dans ce dernier contexte qu nous avons proposé un langage de description et de programmation des système de conduite des automatismes logiques [Silva 77-1] , le but recherché étant d'avoir un formalisme le plus simple possible qui puisse servir à la fois au concepteur, au réalisateur, et à l'opérateur de conduite, ainsi que de permettre une vérification statique des programmes. Ce langage contient les primitives relatives aux comptages et aux temporisations. Il est caractérisé par :

1 - La définition symbolique de l'interface du système de conduite avec le procédé et l'opérateur, c'est-à-dire des voies d'entrées et de sorties logiques.

2 - L'utilisation de graphes d'états interconnectés à évolutions synchrones pour décrire plusieurs automates d'enchaînement d'actions; les événements (conditions d'évolution) sont des fonctions combinatoires d'entrées logiques, d'états des autres automates, et de prédicats calculés.

Le choix d'un outil graphique répond à la nécessité de pouvoir visualiser le déroulement des opérations sur le site et oblige le concepteur à prendre directement en compte cet aspect; d'autre part, il substitue la notion de réceptivité (un automate n'est réceptif à un instant donné qu'à l'ensemble des événements qui conditionne son évolution) à celle d'état total utilisée par exemple dans la méthode d'Huffman.

Le choix des graphes d'états a été influencé par le faible parallélisme rencontré généralement dans le domaine d'application envisagé. Les Réseaux de Petri (conformes et non purs avec inhibiteurs) auraient pu être retenus mais la remarque précédente nous a conduits cependant retenir les graphes d'états pour les raisons suivantes :

- Décomposition algorithmique directement incluse dans le modèle
- Temps maximum pour un cycle de traitement (entre deux scrutations des entrées) toujours inférieur, d'où la possibilité d'avoir un plus grand nombre d'automatismes.

3 - La définition d'un mécanisme d'appels de calculs d'actions ou de conditions d'évolution complexes, en particulier d'actions dites de fond dont l'exécution peut être différée. Cependant, le langage de programmation de ces calculs n'est pas défini, ce qui limite les possibilités de vérification statique des programmes.

Le système se distingue des réalisations existantes essentiellement par le langage de spécification directe des Graphes d'Etats.

Les automates programmables existants [Silva 77-2] peuvent être répartis en trois classes principales :

1) Les automates programmables actuellement commercialisés, dont la structure est conçue pour permettre la programmation à partir d'équations booléennes ou de schémas à relais : la raison de ce choix est le problème de transposition technologique pour les utilisateurs. La nécessité de calculer à chaque cycle de traitement toutes les équations conduit à des temps d'exécution élevés et au risque d'aléas de fonctionnement. Comme le remarquent divers auteurs [Tourres 76, Daclin 77], il semble peu naturel d'utiliser une machine programmable, par nature séquentielle, et de construi des programmes à l'aide de modèles combinatoires, définis justement pour minimiser le caractère séquentiel du fonctionnement.

2) Des systèmes qualifiés d'informatiques car programmés à l'aide d'organigrammes [Anceau 75, Peirano 75].

3) Des systèmes basés sur une programmation à partir d'un Graphe d'Etat [Clark 75, Daclin 77], de plusieurs Graphes d'Etats [Girshig 76, Sabathé 77] ou de Réseaux de Petri [Tourres 76, Daclin 77].

Le système le plus proche du nôtre est celui décrit dans [Girshig 76], mais la programmation est faite en assembleur; il est utilisé pour la conduite de fours sidérurgiques. Enfin, on peut citer des modèles de description de systèmes logiques récemment développés et apparentés aux Réseaux de Petri les schémas de Réseaux de Petri [Valette 76], les Réseaux de Contrôle de Processus Parallèles [Moalla 76] et les organiphases [Brard 76], les deux derniers étant identiques; corrélativement à ces travaux, une standardisatio a été proposée en France [AFCET 77] d'un formalisme graphique de description des automatismes logiques, appelé GRAFCET.

I.2.2. - L'élaboration de systèmes de conduite pour automatismes généraux

Le modèle utilisé pour les automatismes logiques contient des mécanismes de synchronisation et d'appels de sous-programmes qui ont été développés afin de parvenir à une description de système de conduite plus généraux (comportant par exemple des algorithmes de régulation), en utilisant la notion de hiérarchie d'automates [Pleyber 77-2].

Il en résulte que les algorithmes de régulation continue sont alors dirigés par les algorithmes logiques (les automates). Dans d'autres systèmes, comme celui décrit ci-après, l'utilisation des notions de tâche et d'interruption peut faire apparaître une hiérarchie inverse, puisque les modules fonctionnels sont banalisés à l'aide du même concept algorithmique de tâche. Par exemple, une tâche de régulation peut activer une tâche séquentielle de détection de seuil sur une mesure. Cependant, on peut remarquer que l'algorithme de régulation est en fait modifié par l'occurrence du dépassement de seuil : il est donc tout aussi naturel de considérer la prise en compte de l'événement comme hiérarchiquement supérieure à l'activation de la régulation.

La structure des systèmes informatiques de conduite d'automatismes de régulation complexes est étudiée au Laboratoire d'Automatique de Grenoble en particulier par P. Deschizeaux et P. Ladet, à partir d'expériences sur le langage Temps Réel PROCOL [Deschizeaux 74]. Les résultats qu'ils ont acquis seront d'abord décrits, puis les propositions présentées dans cette thèse situées par rapport à l'ensemble des langages existants.

A) Les outils de structuration développés au Laboratoire d'Automatique de Grenoble [Ladet 77, Deschizeaux 77].

La modifiabilité en ligne des automatismes complexes, la recherche d'une structure décentralisée et celle d'une description fonctionnelle de l'application amènent à définir les trois intervenants suivants :

- les outils de base, qui regroupent la définition des entrées-sorties industrielles, les programmes de calcul numérique ou de gestion ne faisant pas intervenir le temps réel, et les événements;

- les processus, construits à partir des outils de base, et correspondant à des tâches fonctionnelles;
- la structure de l'application, c'est-à-dire l'ensemble des liens unissant les processus et les événements.

L'option fondamentale est la possibilité de modifier en ligne chacun de ces intervenants, indépendamment des autres. c'est le seul système, du moins où est défini un formalisme de haut niveau correspondant, qui a cette caractéristique.

1°) Les entrées-sorties

Elles sont définies extérieurement au processus afin de pouvoir modifier leurs caractéristiques indépendamment de ceux-ci. Les caractéristiques d'une entrée-sortie sont son numéro de voie physique et un ou plusieurs traitements associés (sous-programmes paramétrables de conversion, filtrage) : différents traitements peuvent donc être associés à une même mesure ou action de commande.

2°) Les événements

Ils sont liés aux interruptions matérielles en provenance du procédé ou des horloges internes, ou bien déclenchés par des processus ou l'opérateur. Une priorité est attachée à l'événement.

3°) Les processus

Ce sont des programmes séquentiels qui peuvent activer des événements mais ne peuvent se mettre en attente (plusieurs processus doivent alors être définis).

Le mécanisme de communication entre processus a été choisi pour faciliter la décentralisation de la conduite, il est réalisé :

- soit par émissions de messages entre processus, les processus récepteurs ne prenant généralement en compte que le dernier message envoyé, et ignorant les précédents.

- soit par interface sur fichier externe partagé par les processus (cette deuxième solution est plus rapide quand un processus émetteur doit envoyer le même train de messages vers plusieurs processus récepteurs).

4°) Les liens événements-processus

Un processus peut être activé sur occurrence d'un événement dans le passé, dans le futur, ou dans le passé et le futur. L'activation peut avoir lieu une fois ou autant de fois que d'occurrences. Tous les événements définis pour l'application sont mémorisés et comptabilisés. Un lien est défini dans un processus ou par l'opérateur.

La modification en ligne est possible sur ces quatre types d'objets. La réalisation a été obtenue par adaptation du moniteur Temps Réel supervisant le langage PROCOL.

B) Situation du langage proposé par rapport aux Logiciels existants

La recherche d'une structure décentralisée et d'une description fonctionnelle a également guidé la définition du langage.
Cependant, les remarques suivantes peuvent être faites :

1 - La modifiabilité en ligne du système de conduite n'a pas été considérée. Son utilité n'est pas mise en cause pour des systèmes complexes, mais elle pose des problèmes de fond qui seront discutés dans la conclusion.

2 - L'hypothèse d'une scrutation périodique des entrées a été largement prise en compte car elle est le principe des algorithmes de régulation et l'interprétation synchrone des Graphes d'Etats a conduit à considérer de la même façon l'acquisition des mesures logiques. Ceci a pour conséquence de rendre implicite la lecture des entrées et donc plus concise la description. Un critère de décomposition algorithmique, celui de rythme de scrutation des entrées, est introduit dans le modèle de description (chapitre III). les primitives de

temporisation, et les automates, peuvent être utilisés pour les traitements à caractère Temps Réel et non périodiques.

3 - La notion d'événement lié aux interruptions matérielles n'est pas considérée, car cette relation est à notre avis un problème d'implémentation et non pas de description. La notion de niveau de priorité d'événement est rendue implicite par l'utilisation de systèmes d'automates décrits par Graphes d'Etats, et en particulier de hiérarchie d'automates.

4 - La synchronisation des tâches est complètement séparée de leurs descriptions alors que les processus contiennent les instructions de synchronisation.

Les deux langages qui viennent d'être comparés, appartiennent à la classe des langages d'application, l'autre classe étant celle des langages généraux, c'est-à-dire comprenant des instructions permettant la programmation en Temps Réel, mais où ne sont pas, ou peu, pris en compte les aspects spécifiques de la conduite de procédés industriels.

Une étude des logiciels d'application peut être trouvée dans [Gertler 75], et dans [Ladet 77] en ce qui concerne plus particulièrement les langages de programmation, analysés des points de vue entrées-sorties industrielles, événements, et synchronisation.

1°) Les langages d'application [Pike 70]

Ils présentent un certain nombre de caractéristiques intéressantes relatives à la séparation à des degrés divers de différents composants des systèmes : entrées-sorties, boucles de régulation, événements, liens entre les événements et les actions. Certains, comme BICEPS et PROSPRO nécessitent en partie une programmation sur formulaires. Le plus connu est AUTRAN, qui a une syntaxe proche de Fortran : des instructions y permettent de lier des événements et des calculs ou des actions. Cependant l'enchaînement global des tâches fonctionnelles n'est pas clairement défini; le parallélisme n'est

n'est pas traité et les événements sont seulement liés aux interruptions matérielles en provenance de l'extérieur du calculateur.

2°) Les langages généraux

Ce sont les plus répandus et leur développement a été principalemen conditionné par les applications militaires. Les possibilités de leurs stru tures de contrôle et de données s'étendent depuis celles de Fortran, étendu pour pouvoir synchroniser des tâches, jusqu'à celles de PL/1, comme c'est l cas pour HAL/S [Fylstra75] défini par la NASA pour la conduite des vols spatiaux ou PROGRESS [Mor 72] défini en RFA pour de grosses applications in dustrielles; ces deux derniers langages ont des structures de données inspi rées de celles d'ALGOL 68 et nécessitent des moyens importants : le compila teur de PROGRESS, par exemple, occupe 300K octets de mémoire.

Une première catégorie est issue d'extensions apportées à des langa existants, en particulier Fortran, et caractérisée par l'utilisation de la notion de tâche : une tâche est un programme qui peut être activé, arrêté, suspendu, ou repris; des instructions correspondantes sont incluses dans le langage. Une remarque importante est que les instructions de synchronisatio les traitements des entrées-sorties et les programmes de calcul sont décrit au même niveau.

Fortran Temps Réel fait l'objet d'une standardisation [Pike 70] aux Etats-Unis et comprend, pour la gestion des tâches, seulement les trois primitives suivantes : activation après un délai ou à un instant donné, suspe sion après un délai : CORAL 66 [Woodward 70] et RTL/2 [Barnes 76], largemen utilisés en Grande Bretagne, ne contiennent pas d'instructions de synchroni- sation de tâches proprement dit mais offrent des facilités pour définir et utiliser des macros-instructions dans le langage de la machine support. PROCOL [Ritout 72] et PEARL [Elzer 75] ont été développés depuis 1970 res- pectivement en France et en RFA; ils contiennent un jeu d'instructions plus complet en ce qui concerne la structure de contrôle des programmes (inter- ruptions logicielles, sémaphores) ainsi que des entrées-sorties industrielle

pour lesquelles les traitements associés sont définis dans les déclarations correspondantes.

Une deuxième catégorie de langages, récente, est caractérisée par la recherche d'une description plus synthétique des synchronisations. La notion de tâche est abandonnée pour celle de module, mais celle-ci peut être comprise dans des sens assez différents. Trois exemples, correspondant à des approches différentes, sont donnés dans les paragraphes suivants.

a) MODULA [Wirth 77-1] est un langage dérivé de PASCAL, orienté vers la programmation modulaire des systèmes sur mini-calculateurs. Chaque module comprend un en-tête où sont spécifiées les variables qui peuvent être observées ou modifiées de l'extérieur (ce qui élimine la nécessité d'avoir des procédures d'accès, comme dans d'autres modules comme ceux définis dans [Cheval 77]). Les périphériques sont gérés par des modules particuliers ("Device Modules") utilisant une instruction spéciale nommée "doio" pour spécifier l'attente d'une interruption en provenance du périphérique; pour assurer la communication entre modules, des modules d'interface sont distingués, où une seule procédure peut être activée à la fois. Des objets de type chaîne de bits peuvent être déclarés et la programmation peut se faire en prenant en compte la structure de la machine par ce moyen. Dans [Holden 77], une comparaison est faite, sur le plan de l'utilisation, avec CONCURRENT PASCAL [Brinch Hansen 75], conçu également pour l'écriture des systèmes opératoires : il est remarqué que ce dernier nécessite des moyens plus importants et qu'il est impossible d'y programmer des périphériques dans le langage lui-même (il ne peut être considéré comme un véritable langage Temps Réel).

Mentionnons également dans le même ordre d'idées le langage MORAL [Jackson 76] qui permet la programmation à partir de diagrammes décrivant les tâches et leurs interconnexions. Un programme MORAL est exécuté sous le contrôle du système MASCOT dont l'originalité est la possibilité de permettre la définition de modules langage (MORAL, CORAL, ALGOL 68), la communication étant faite par zones communes ou par échanges de messages.

b) GAELIC [Le Calvez 77-2] est un langage de description globale des synchronisations et des relations entre événements (au sens interruptions) et actions. Il est issu de travaux sur la description de l'ordonnancement de tâches à partir de représentations graphiques et de leur mise en oeuvre à l'aide d'automates d'états finis [Mendelbaum 77].

La séparation de la description et de l'enchaînement des tâches est effectuée dans le but de faire une vérification statique du programme et de pouvoir changer les modules indépendants du reste. Un programme GAELIC comprend plusieurs paragraphes, qui sont :

- la description de l'environnement : événements (interruptions matérielles), noms des modules liés à ces événements, des modules de calcul, et des périphériques partagés par des modules.
- le schéma de contrôle. Les liens entre les modules, ainsi qu'entre les modules et les événements, sont décrits au moyen de déclarations :
  . de variables communes,
  . de portes de synchronisation,
  . d'ensembles de chemins de contrôle parallèles ou exclusifs, définis par leurs noms et leurs nombres,
  . de modules et de noms de sous-schémas de contrôle.
- l'enchaînement global, décrit à l'aide d'instructions Temps Réel (DESQUE...ALORS, etc.) semblables à celles définies dans [Ladet 77] et d'instructions de synchronisation, associant aux noms des chemins parallèles ou exclusifs, la description de ces chemins.

Ce langage peut être implémenté à l'aide de systèmes d'automates d'états finis [Mendelbaum 77]; une traduction des programmes vers le langage Temps Réel PROCOL est proposée dans [Le Calvez 77-1]. Le nombre important de concepts, la division et la redondance au niveau de la spécification des synchronisations en font à mon avis un langage compliqué à utiliser et se prêtant mal à une décomposition fonctionnelle de la structure de contrôle.

c) Les PARCs (Parallel Constructs) [Haase 77] sont des commandes gardées [Dijkstra 75] adaptées pour décrire des enchaînements d'actions en Temps Réel. Elles sont sélectives ou itératives et ont respectivement la forme :

```
IF   parc name
Guard 1 ——→ program 1
Guard 2 ——→ program 2
⋮
Guard n ——→ program n
FI   parc name
```

et

```
DO   parc name
Guard 1 ——→ program 1
Guard 2 ——→ program 2
⋮
Guard n ——→ program n
```

où $Guard_i$ représente une expression booléenne et $program_i$ un programme séquentiel.

L'exécution d'un PARC consiste à calculer simultanément toutes les "gardes" et à exécuter les programmes correspondant à celles qui ont la valeur VRAI. L'exécution d'un PARC itératif est abandonnée quand toutes les gardes prennent la valeur FAUX.

Afin de satisfaire à la nécessité d'une représentation graphique pour les problèmes de conduite, la sémantique des PARCs est décrite sous forme de Réseaux de Petri ayant une place initiale et une place finale. Cependant, la motivation principale conduisant à l'utilisation de commandes gardées est la facilité d'en définir la sémantique à l'aide de transformateurs de prédicats, à partir de celle des programmes séquentiels: cet aspect est également étudié dans [Haase 77].

I.3. - REMARQUES GENERALES

L'orientation générale des langages de programmation à caractère Temps Réel est la modularité et la séparation des descriptions de la structure de contrôle et des traitement strictement séquentiels.

Il est significatif de remarquer qu'une démarche semblable est suivie dans des travaux récents appartenant à des domaines voisins :

- les langages de description et d'écriture de systèmes opératoires, avec les concepts d'expressions de chemin [Lauer 77] et de compteurs de synchronisation [Robert 77];
- les langages de description et de simulation de matériel logique, dont une analyse de ce point de vue peut être trouvée dans [Zachariades 77]

Les automatismes logiques, récemment concernés par la programmation, font l'objet de modélisations nombreuses dérivées des Réseaux de Petri, mais peu de langages d'application ont encore été définis, la programmation étant faite en assembleur.

Les langages récents effectivement utilisés pour la programmation des automatismes de régulation sont des langages généraux, offrant parfois des traitements spécifiques au niveau des entrées-sorties industrielles (PROCOL et PEARL). Seuls, AUTRAN et celui défini dans [Ladet 77] prennent en compte le point de vue des opérateurs de conduite et sont structurés en conséquence

La séparation entre les langages d'application utilisés dans les deux domaines est caractéristique de la situation actuelle et reflète celle existant au niveau de la description des automatismes complexes. Si l'on considère par exemple l'organigramme de conception des automatismes d'une centrale nucléaire à l'EDF (Fig. 1), on peut constater que les automatismes de régulation ("de réglage") et les automatismes logiques sont d'abord étudiés à partir du "niveau d'automatisation souhaitée" et sont regroupés seulement au niveau de l'architecture d'ensemble du système" : la description globale des automatismes de conduite de la centrale n'apparaît pas dans cet organigramme.

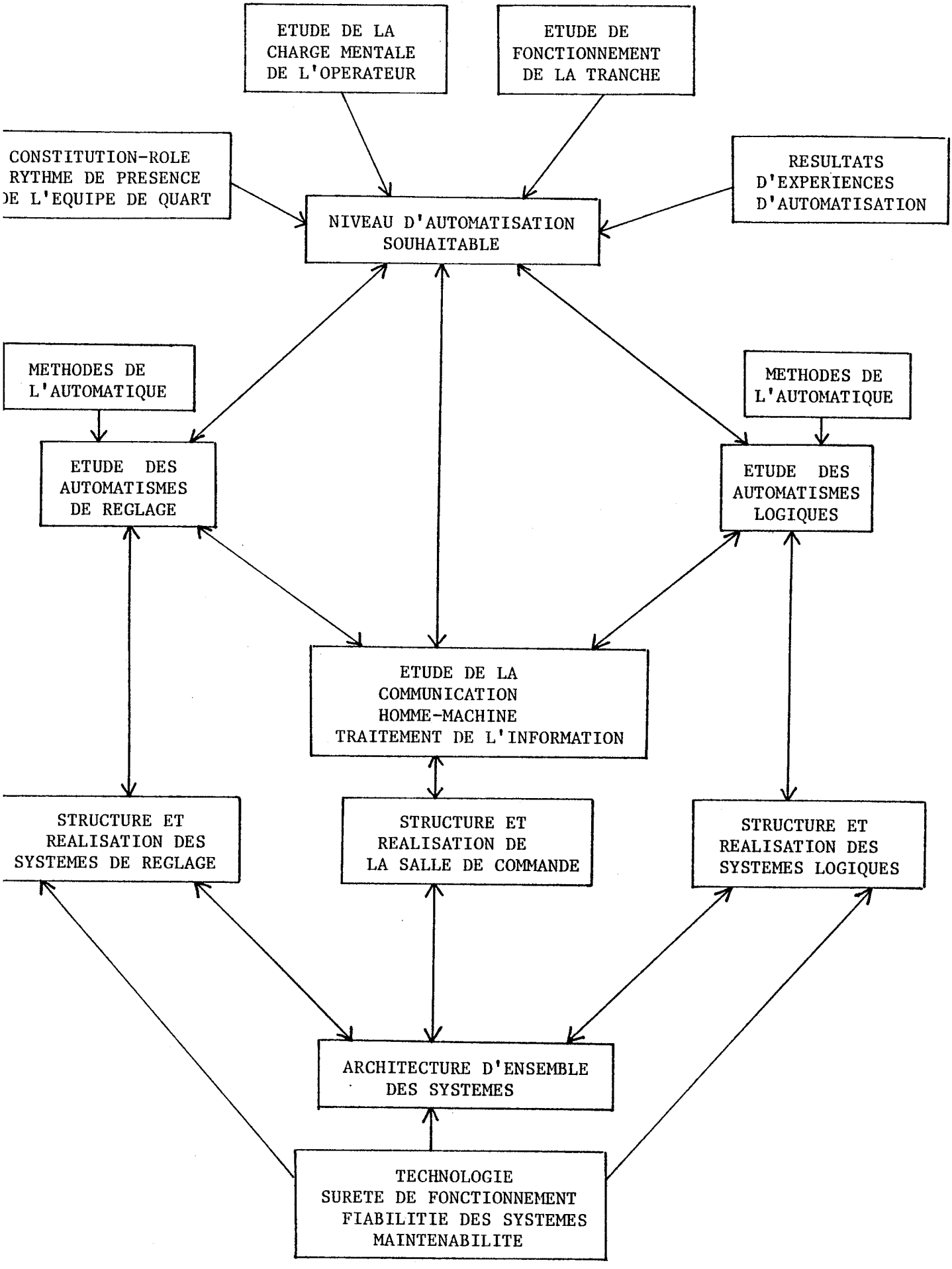


Fig. 1 : Organigramme de conception d'une centrale nucléaire.
Extrait de [Copin 77].

La signification de cette thèse est une tentative pour combler cette lacune. Le niveau choisi a été celui de la description abstraite des algorithmes, où sont utilisées les spécifications fonctionnelles de l'application mais non celles d'une implémentation particulière.

Chapitre II

ASPECTS METHODOLOGIQUES DE L'ELABORATION D'UN SYSTEME DE CONDUITE

II.1. - GENERALITES

L'élaboration d'une application informatique comprend essentiellement deux phases :

1) l'analyse;

2) la programmation.

Le but de l'analyse est l'établissement de spécifications précises (sinon formelles) du problème à résoudre, à partir desquelles le programme doit être construit. Dans un article récent consacré à ce sujet [Parnas 77], Parnas rappelle les motivations de l'obtention de telles spécifications :

- décrire le problème de façon non ambiguë;
- permettre le dialogue entre les différents spécialistes concernés par la réalisation de l'application;
- libérer chaque programmeur de la nécessité de savoir comment fonctionnent les parties du système qui ne le concernent pas;
- permettre le développement de plusieurs versions du programme;
- pouvoir reprendre des choix effectués au cours de l'analyse en fonction des contraintes dues à la programmation;
- permettre la vérification de la cohérence des choix effectués au cours de l'analyse.

Les motivations suivantes sont également à considérer :

- faciliter une mise en place progressive de l'architecture logicielle et matérielle;
- assurer une adaptabilité satisfaisante de l'application des points de vue de la portabilité et de la transformation ou l'extension des objectifs.

La rigueur des méthodes et formalismes utilisés est considérée d'autant plus nécessaire que les applications à réaliser sont complexes. Cependant, il ne faut pas sous-estimer leur importance même pour des applications simples. En effet :

- le coût du matériel devient de moins en moins considérable par rapport à celui des études et du logiciel [Rousseau 76];
- la forte possibilité d'adaptation et la puissance d'un système numérique (en particulier en Automatique par rapport aux systèmes réalisés à l'aide de technologies plus anciennes) ne peut être réellement exploitée que si l'on dispose de moyens sûrs de modifier partiellement les objectifs ou les programmes;
- l'abstraction de toute réalisation programmée pose des problèmes de sémantique qui nécessitent un "cadre syntaxique" rigoureux pour être résolus.

II.2. - METHODES ASCENDANTES ET DESCENDANTES

Certaines des motivations énoncées pour l'établissement de spécifications précises montrent que les résultats de l'analyse sont dans le cas général modifiés lors de l'écriture du programme. Aussi convient-il de considérer globalement les méthodes d'analyse et de programmation.

Il est classique d'en distinguer deux grandes catégories, qualifiées respectivement de descendantes et d'ascendantes.

Une méthode descendante consiste à partir d'un objectif global défini en faisant abstraction des moyens permettant sa réalisation, puis arriver par décompositions successives à réduire la complexité jusqu'à ce que ces moyens puissent être choisis. Une méthode ascendante consiste inversement à donner les moyens puis, par compositions successives, à grouper ces moyens pour réaliser des objectifs partiels jusqu'à l'obtention de

l'objectif global. Il est toute fois clair que :

- les objectifs sont limités par les moyens dont il est possible de disposer;
- les moyens n'ont de signification que par l'usage qui en est fait, c'est-à-dire par les objectifs qu'il est possible d'atteindre.

L'évidence de ces remarques amène à dire que le problème du choix entre méthodes descendantes et ascendantes est un faux débat qui semble parvenir d'une certaine confusion entre les notions d'analyse de problème, de description de l'application, de conception de programme, et de mise en place de l'application.

L'analyse de problème est une activité où il est nécessaire de procéder par "allers-retours" entre l'énoncé de l'objectif global et celui des spécifications précises. Il en est de même pour la conception de programmes et, ceci a déjà été souligné, pour les interactions entre les phases d'analyse et de programamtion.

La mise en place d'une application sur son site d'utilisation est en général ascendante, et ceci est particulièrement vrai en Automatique [Deschizeaux 75].

Enfin, il est clair qu'une méthode d'analyse et de programmation doit conduire à un logiciel satisfaisant correctement les objectifs mais doit permettre également d'obtenir un document décrivant :

- les spécifications obtenues par l'analyse, en faisant apparaître clairement l'historique des choix effectués au cours de cette analyse et leurs motivations;
- les programmes ainsi que la façon dont leurs structures ont été construites.

Les principales qualités souhaitables pour une telle description sont la clarté, l'analysabilité et la possibilité de l'utiliser pour maintenir

et modifier l'application. L'expérience a jusquà présent montré qu'une description descendante donnait de meilleurs résultats. La raison en est principalement qu'il est facile de mettre en évidence les implications sur l'arbre global de modifications sur un noeud de cet arbre. Pour y parvenir,il faut disposer d'un certain nombre de règles de décomposition et de composition adéquates pour définir les différents niveaux d'abstraction qui sont ceux des arbres d'analyse et de programmation. C'est pourquoi il est proposé dans les paragraphes suivants de dégager un certain nombre d'étapes dans l'analyse et la programmation permettant d'obtenir une description descendante. Ceci est fait dans le cas général (II.3) puis dans celui d'un système de conduite : le contexte d'utilisation du langage de programmation proposé dans les chapitres suivants sera ainsi précisé.

Remarque : Dans la pratique, la difficulté suivante apparaît dans l'établissement d'une description descendante : pour décrire un niveau, faire abstraction de la description des niveaux inférieurs, tout en tenant compte des contraintes et critères associés. Le problème de l'acquisition de la discipline de travail correspondante est-elle à l'origine de la lenteur avec laquelle les méthodes proposées sont adoptées par les praticiens ? On peut penser cependant qu'au fur et à mesure que les concepts seront précisés, des systèmes automatiques interactifs d'aide à la conception pourront être développés, qui produiront un guide de travail intéressant; la facilité de la conception et la validation des applications, objectifs sous-jacents, seront alors fortement améliorées, créant par là-même une motivation.

II.3. - UNE METHODE GENERALE D'ANALYSE ET DE PROGRAMMATION

Les quatre étapes suivantes (subdivision inspirée de [Finance 77]) aboutissent chacune à l'écriture de spécifications (énoncés ou programme) dont l'ensemble constitue alors la description (descendante) de l'application.

1 . Enoncé informel

Le problème est formulé par l'utilisateur final de l'application sous forme d'objectifs à atteindre et de contraintes d'utilisation et de mise en oeuvre. Le langage naturel est utilisé, introduisant en général des ambiguités.

2 . Enoncé implicite

Le but est d'obtenir une formulation abstraite du problème non nécessairement constructive, c'est-à-dire où la solution peut rester implicite.

Une première décomposition est en général possible en séparant divers objectifs liés hiérarchiquement (décomposition verticale) et pour un même niveau ceux qui sont indépendants (décomposition horizontale). L'indépendance de deux objectifs signifie que les algorithmes correspondants pourront prendre indépendamment des décisions quant à la coordination de ces objectifs. La décomposition verticale est guidée principalement par les contraintes de mise en oeuvre (on réalisera d'abord le niveau le plus bas) tandis que la décomposition horizontale l'est davantage par les contraintes d'utilisation.

Un système est associé à chaque sous-objectif. On est donc amené à le nommer et à définir son milieu extérieur et les liens qui l'unissent à celui-ci, c'est-à-dire ses entrées et ses sorties. Le milieu extérieur d'un système comprend en particulier le système du niveau supérieur, ceux de même niveau, et ceux des niveaux inférieurs.

L'abstraction porte ici sur les fonctions associées aux objectifs et résumées par des noms, mais également sur la représentation des entrées et des sorties au moyen d'ensembles numériques, alphanumériques, logiques, etc., et plus généralement d'ensembles produits (valeurs structurées). La fonction associée à chaque système peut alors être représentée au moyen de relations algébriques et de lois de composition définies sur ces ensembles. Il en résulte une décomposition fonctionnelle qui peut être poursuivie de façon à obtenir une complexité intuitivement acceptable pour l'élaboration de l'énoncé explicite.

3. Enoncé explicite

Cette étape aboutit à la spécification de schémas d'algorithme et de structure abstraite pour chacun des systèmes isolés par l'étape précédente.

Le concept de schéma d'algorithme (de programme) introduit en 1954 par IANOV puis repris à partir de 1968 consiste à définir l'algorithme sans donner le domaine des variables, ni la description des fonctions et des prédicats. Une abstraction importante est faite ici, qui consiste à reconnaître la classe d'algorithmes à laquelle appartient chaque fonction. Cette abstraction induit une nouvelle décomposition, algorithmique, sur les fonctions et prédicats du schéma initial, liée au choix de structures abstraites; ce choix est guidé par des critères algorithmiques (algorithme le plus rapide, le moins encombrant,...) et des contraintes techniques (possibilité de représentation concrète et de décomposition à l'aide des langages de programmation disponibles, segmentation des programmes).

4 . Programme

La structure abstraite est représentée dans une mémoire concrète à l'aide de structures de données d'un langage de programmation; les calculs sur la structure abstraite sont transformés en une suite d'instructions du langage de programmation. De nouvelles décompositions peuvent avoir

lieu, liées aux possibilités du langage (macro-instructions, sous-programmes de types divers,...) et aux contraintes dues au matériel (segmentation, nombre de processeurs, ...).

Le programme spécifie des calculs. On distingue :

a) Les calculs abstraits qui consistent à découvrir l'invariance de certains prédicats au travers des transformations décrites par le programme et à les utiliser pour la vérification statique.

b) Les calculs concrets, c'est-à-dire l'exécution des instructions du programme pour certaines valeurs des entrées, susceptibles, dans le cas général, de varier au cours des calculs et d'en modifier le déroulement.

III.4. - ELABORATION D'UN SYSTEME DE CONDUITE

Un automatisme peut être schématisé structurellement par la figure 1. Trois composantes principales sont en jeu : les opérateurs, le système de conduite et le procédé, liés fonctionnellement par la hiérarchie décrite par la figure 2.

Le schéma de la figure 2 est assez général pour prendre en compte une boucle de contrôle directe entre les opérateurs et le procédé (actions manuelles et comptes rendus directs). Remarquons que les actions manuelles ont une influence sur la commande par l'intermédiaire du procédé, dont elles modifient l'état. Dans la description du système de conduite, nous considérerons qu'elles font partie des perturbations, mesurables ou non. Elles représentent cependant une composante du contrôle et c'est pourquoi nous les distinguons ici, car elles ont une conséquence voulue sur l'état du procédé.

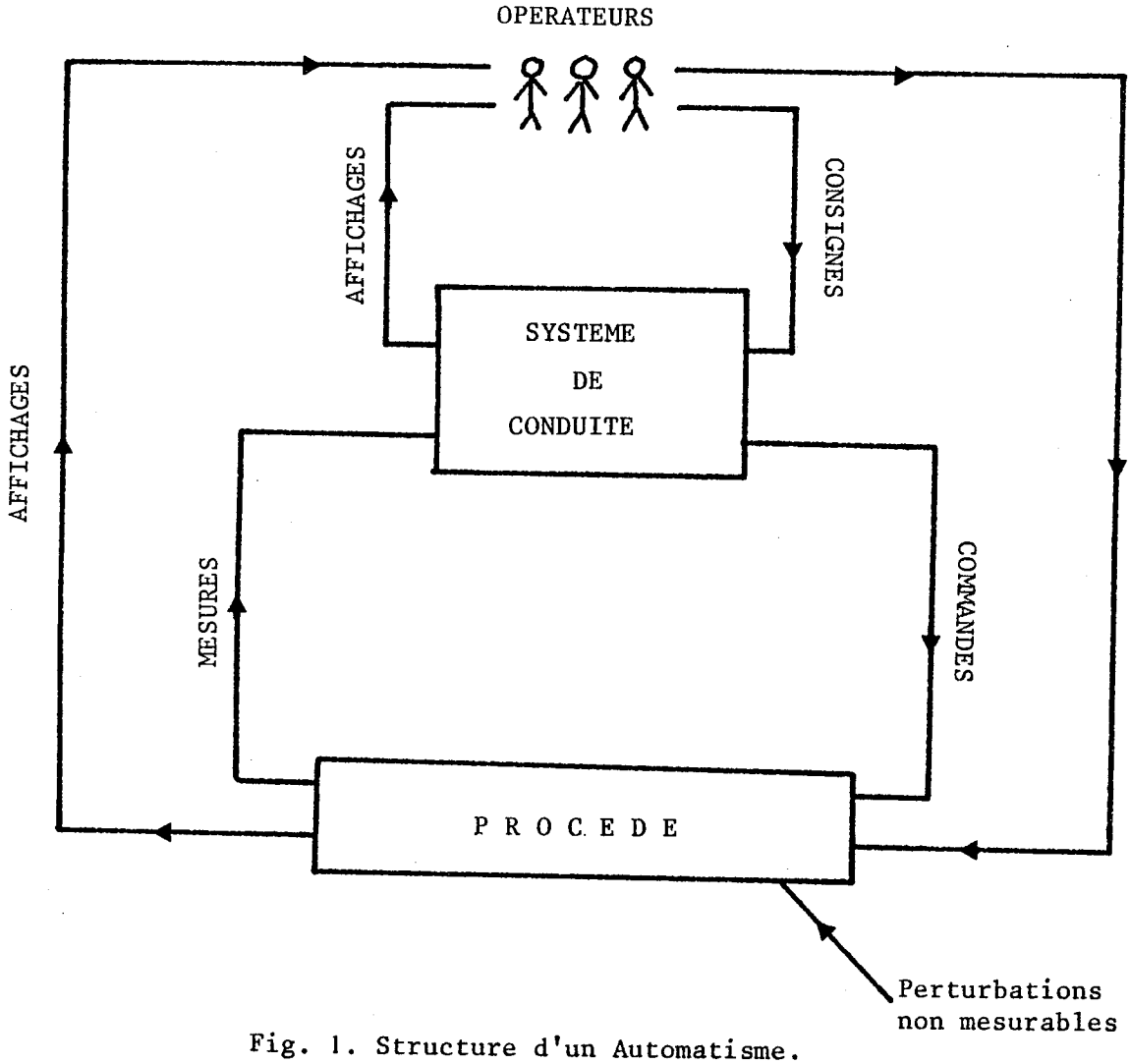


Fig. 1. Structure d'un Automatisme.

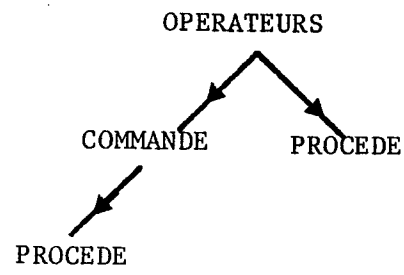


Fig. 2. Relations hiérarchiques dans un Automatisme.

Précisons les autres liens entre les opérateurs, le système de conduite, et le procédé.

A) Les entrées du système de conduite : les mesures et les consignes

Une mesure ou une consigne est prise en compte à un moment fixé par l'algorithme de conduite et peut être :

- analogique : un registre contient une représentation d'une valeur numérique pour laquelle il convient généralement d'effectuer (1) un contrôle physique destiné à vérifier le bon fonctionnement du coupleur, (2) une conversion et une linéarisation puis un filtrage afin d'éliminer les mesures aberrantes faites par le capteur, (3) un contrôle logique de détection de seuil d'application d'algorithm
- logique : elle peut prendre 2 valeurs, notées "1" et "0". Une mesur de ce type est souvent appelée alarme quand sa prise en compte peut présenter un caractère d'urgence ou événement quand l'algorith me est en attente d'un front de montée ou de descente du signal.
- alphanumérique : les entrées de ce type sont essentiellement celles qui permettent aux opérateurs d'envoyer des consignes au système de conduite (télétype); cependant, on peut considérer leur utilisation dans le cas plus général d'interconnexions entre systèmes (par exemple dans le contexte de la mise en place d'un système entr un opérateur et le système initial). Une chaîne de caractères introduite par une telle entrée doit être interprétée pour pouvoir être utilisée; elle peut être une simple valeur de consigne (par exemple un coefficient nécessaire à un calcul) ou une phrase d'un langage de commande.

B) Les sorties du système de conduite : les commandes et les affichages

Une commande ou une mesure peut être :

- analogique : une conversion doit être effectuée préalablement à l'envoi de la valeur vers le dispositif de commande ou d'affichage;
- logique (à niveau);
- impulsionnelle : le calibre de l'impulsion est déterminé par le matériel;
- alphanumérique.

Les affichages peuvent être très variés. On distingue en particulier :

- la visualisation, information synthétique permanente permettant à l'opérateur de surveiller le déroulement des opérations et de prendre éventuellement des décisions à caractère immédiat (n'ayant pas forcément un caractère d'urgence) par l'intermédiaire des consignes ou des commandes manuelles;
- l'archivage, ou établissement de bilans physiques, économiques ou comptables, et en particulier du journal de bord, conduisant à des décisions pouvant être différées.
- les alarmes, nécessitant de l'opérateur une décision ayant un caractère d'urgence.
- les affichages proprement dit, par exemple certaines informations plus synthétiques que la visualisation, ou au contraire plus détaillées, et appelant une décision de la part de l'opérateur (semi-automatisation).

Les différentes composantes entrant en jeu dans un automatisme ayant été définies, les différentes étapes rencontrées depuis l'exhibition d'un problème de conduite jusqu'à la programmation de celle-ci peuvent maintenant être examinées.

II.4.1. - L'énoncé informel

Une commande est d'abord connue à partir d'un ensemble de spécifications constituant l'avant-projet du cahier des charges de l'automatisme présenté par l'utilisateur au concepteur. Ce cahier des charges est ensuite modifié progressivement au cours de la recherche d'une solution et devient alors le document contractuel liant l'utilisateur et le concepteur

Remarquons à ce sujet qu'un langage formel - par exemple un langage de spécification ou un langage de programmation - n'est pas juridiquement reconnu pour décrire un système. Ceci constitue un des obstacles à l'utilisation d'un "langage commun" entre l'utilisateur et le concepteur.

1) Les spécifications fonctionnelles

L'automatisme est décrit du point de vue externe par les flux de matières, d'énergie et d'informations en amont et en aval du procédé. Des objectifs sont précisés pour les flux en aval et en amont en utilisant au plus haut niveau des critères définis à priori, c'est-à-dire indépendamment de l'automatisme.

2) Les spécifications opérationnelles

Les conditions d'utilisation sont relatives à :

- l'exploitation du procédé, comprenant :
  . le suivi de l'automatisme en "temps réel", c'est-à-dire d'une part quelles décisions et quelles parties du procédé doivent être sous le contrôle direct d'un opérateur humain, d'autre part ce qui doit, ou peut, être automatisé. Ceci conduit à définir un niveau d'automatisation, où entrent en jeu des contraintes techniques, ergonomiques, et de sûreté.
  . le suivi de l'automatisme en "temps différé", c'est-à-dire l'archivage d'informations devant être transmis à un niveau de décision et de contrôle supérieur, pouvant amener une modification des objectifs.

- la sûreté de fonctionnement, qui peut être décomposée par exemple en fiabilité, crédibilité, maintenabilité, sécurité et disponibilité. Ces composantes sont définies [UPS 76] en termes de probabilités et prennent une importance plus ou moins grande suivant l'application : la disponibilité est très importante pour un système téléphonique mais c'est la sécurité qui est cruciale pour un système de transport en commun.

- la modifiabilité de la structure du procédé et de celle du système de conduite.

3) Les spécifications technologiques

Elles décrivent les contraintes résultant :

- d'un environnement préexistant : interfaces nécessaires avec d'autres systèmes déjà mis en oeuvre, géographie du site, système d'alimentation en matières et en énergie, conditions atmosphériques;

- des technologies de réalisation disponibles, en particulier pour les capteurs, les actionneurs, ainsi que la commande.

4) Les spécifications administratives

Elles contiennent les aspects financiers et juridiques relatifs aux études, à la réalisation et à l'utilisation.

Ces spécifications ne sont bien entendu pas indépendantes mais participent toutes à l'élaboration des spécifications fonctionnelles à partir desquelles va pouvoir être défini le système de conduite.

II.4.2. - L'énoncé implicite

Le système de conduite peut être globalement représenté par la figure 3. Son milieu extérieur comprend le procédé et les opérateurs.

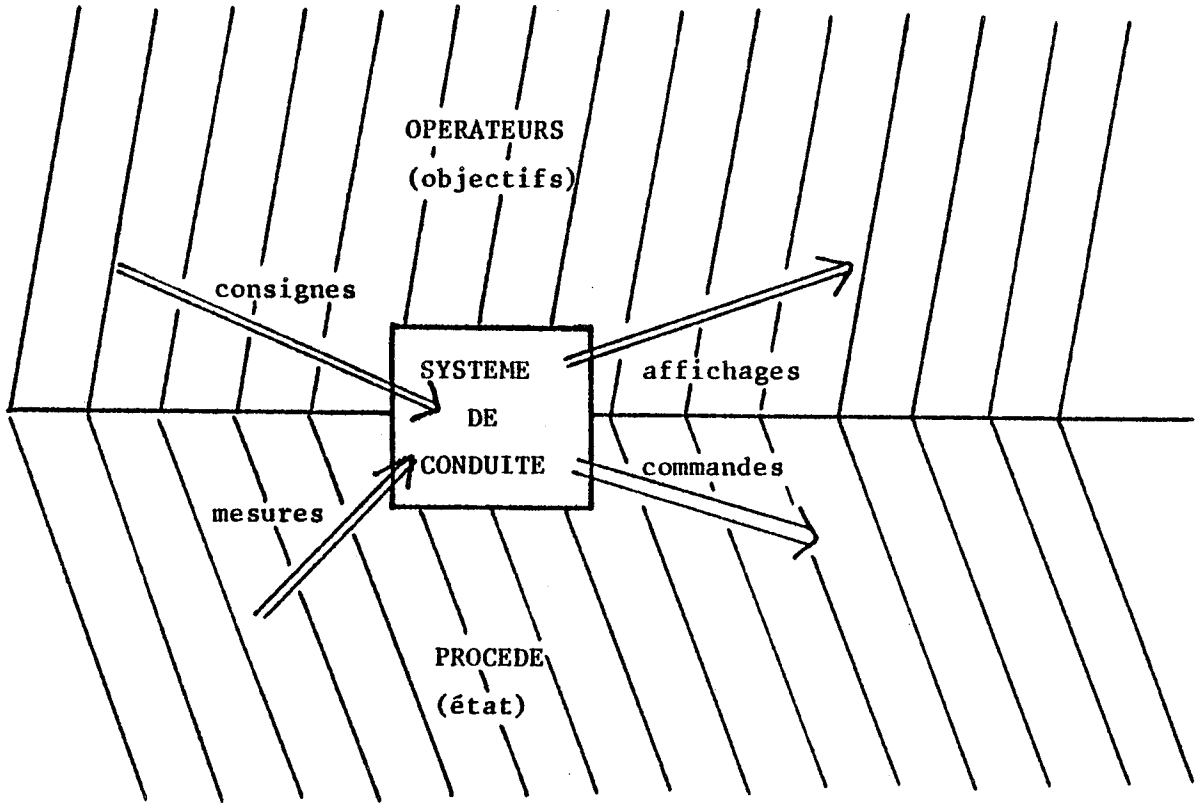
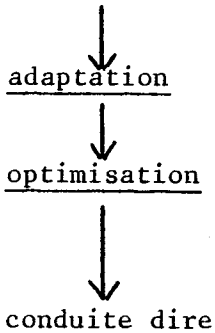


Fig. 3

Il a pour mission d'assurer la satisfaction d'objectifs sur les entrées et les sorties du procédé. Pour les automatismes de régulation, les objectifs sont dans le cas général contradictoires et la solution est alors définie en termes de distance entre les vecteurs de consignes et de mesures (des sorties et des entrées mesurables du procédé) [Fossard 72, Foulard 77, Binder 77].

A) Décomposition verticale

Dans la théorie de la conduite hiérarchisée [Fossard 72], quatre niveaux hiérarchiques sont distingués sur les objectifs :



organisation : détermination des modes de fonctionnement.

adaptation : passage d'un mode de fonctionnement à un autre.

optimisation : détermination des points de consigne ou des trajectoires sur les sorties du procédé.

conduite directe : collecte des mesures et élaboration des commandes à une fréquence correspondant à la dynamique du procédé.

Un système est donc défini pour chaque niveau. Deux critères, outre celui de la hiérarchie des objectifs, guident cette décomposition :

- la nécessité d'utiliser des modèles différents pour spécifier les algorithmes correspondants;
- la fréquence d'intervention sur le procédé, qui croît au fur et à mesure que l'on descend dans la hiérarchie.

La mise en place ascendante est clairement définie : conduite directe, puis optimisation, etc.

Le niveau étudié dans ce mémoire étant celui de conduite directe, ce qui suit concerne uniquement le système correspondant, et éventuellement ses liens avec le niveau optimisation : le terme "système de conduite" doit donc être maintenant compris au sens "système de conduite directe". Toutefois, il apparaîtra que le modèle défini pourra faire l'objet d'une utilisation pour les niveaux optimisation et même adaptation, mais ceci n'a pas été étudié dans le cadre de ce travail.

B) Décomposition horizontale

L'analyse de la structure du procédé suggère généralement de décomposer celui-ci en unités correspondant à des ensembles physiques ayant des fonctionnements propres ou à des écoulements de matière ou d'énergie.

Cette décomposition préalable est souhaitable pour les raisons suivantes :

- la description résultante garde un sens physique, ce qui est important pour le suivi du procédé;
- une modification physique locale du procédé est possible sans remise en cause de l'ensemble du système de conduite.

Elle est en outre intéressante pour les procédés complexes à structure variable (procédé dans lesquels on peut modifier l'ensemble des unités utilisées ou les interconnexions entre celles-ci) car elle permet de simplifier la description des modifications correspondantes sur le système de conduite.

Ceci peut être illustré par les deux exemples suivants :

- le procédé de distillation du Laboratoire d'Automatique de Grenoble comprend 5 unités [Binder 77] : stockage, mélange, colonne 1, colonne 2, et remélange. Les 2 colonnes peuvent être utilisées en parallèle, en série, ou isolément (1 seule à la fois) : il y a donc 4 structures possibles.
- le mélangeur de produits chimiques décrit en V.4. comprend 3 flux principaux de matières : écoulements des produits A et B, écoulement du liquide de refroidissement. Le fonctionnement est défini à partir du Ier et du 3e, ou du 2e et du 3e : il y a donc 2 structures possibles.

Pour une application suffisamment complexe, cette décomposition affecte le niveau optimisation (c'est le cas du premier exemple ci-dessus), la hiérarchie entre les 2 niveaux pouvant alors être représentée par la figure 4 [Foulard 77].

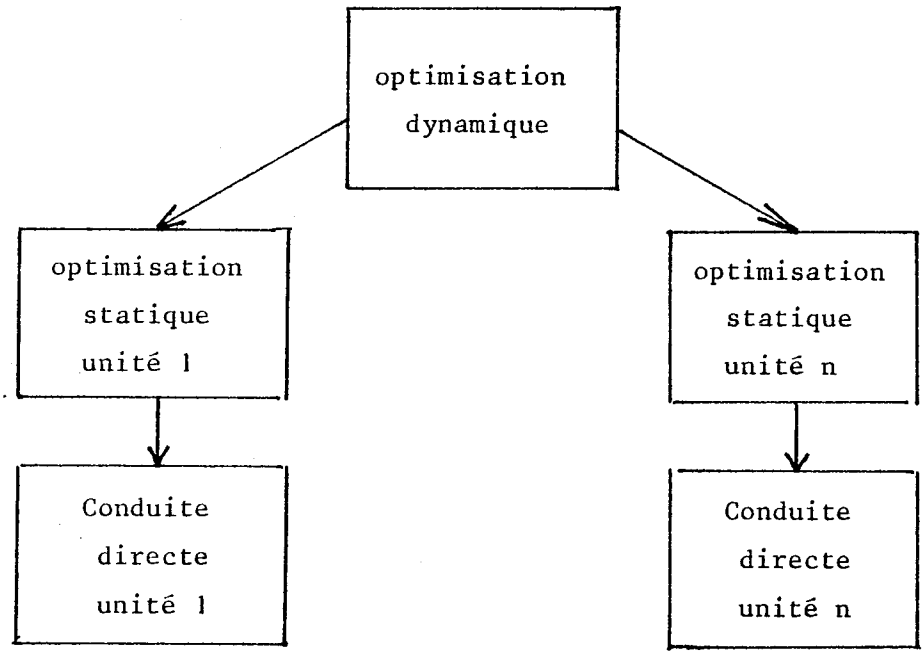


Fig. 4.

C'est donc au niveau de l'optimisation dynamique que sont définis alors les critères physiques globaux caractérisant le procédé. L'algorithme de conduite directe d'une unité dépend d'une optimisation statique, qui sera assimilée dans ce qui suit à une consigne, et envoie un compte rendu au niveau d'optimisation dynamique, assimilé à un affichage.

Une unité pouvant elle-même être décomposée en sous-unités, il en résulte que le système correspondant sera aussi décomposé en sous-systèmes

coordonnés (voir II.4.2.B) auxquels sont associés des sous-objectifs dont l'ensemble doit représenter l'objectif du système.

La coordination est décrite dans la théorie de la commande hiérarchisée de deux façons [Fossard 72] :

- par modification des sous-objectifs, effectuée par le niveau optimisation;
- par prédiction des interactions, chaque système modifiant lui-même son objectif à partir d'informations en provenance du niveau optimisation.

Dans le cas de la prédiction des interactions, il peut être suggéré que chaque sous-système acquiert ses informations directement à partir de autres systèmes. C'est le principe de la commande décentralisée, vers laquelle s'orientent les travaux les plus récents [Binder 77] . Les raiso de cette orientation sont la difficulté rencontrée dans la pratique pour définir des critères physiques globaux pour l'ensemble des unités, la moins bonne sûreté de fonctionnement des structures hiérarchisées, enfin la possibilité récente de décentralisation de la conduite à l'aide d'architectures matérielles réparties.

Il est clair que dans le cas général la solution sera mixte, un sous-système pouvant recevoir des consignes du niveau optimisation mais également prendre des décisions en recevant des informations sur l'"état" des autres sous-systèmes. Un problème actuellement posé est celui de la résolution des conflits entre les centres de décision.

En ce qui concerne le langage présenté dans ce mémoire, il permet la coordination entre sous-systèmes si celle-ci peut être résolue par l'interconnexion d'automates (voir chapitre IV); ceci n'est pas suffisant dans le cas général mais permet cependant de traiter bon nombre de problèmes. La décomposition algorithmique en automates guidera donc également celle du système.

A ce stade de la description, chaque système exhibé par la décomposition précédente peut être schématisé par la figure 5. Chacun de ses liens avec le procédé ou l'opérateur est d'un des types énoncés dans l'introduction de II.4. : analogique, logique, impulsion (en sortie seulement), alphanumérique.

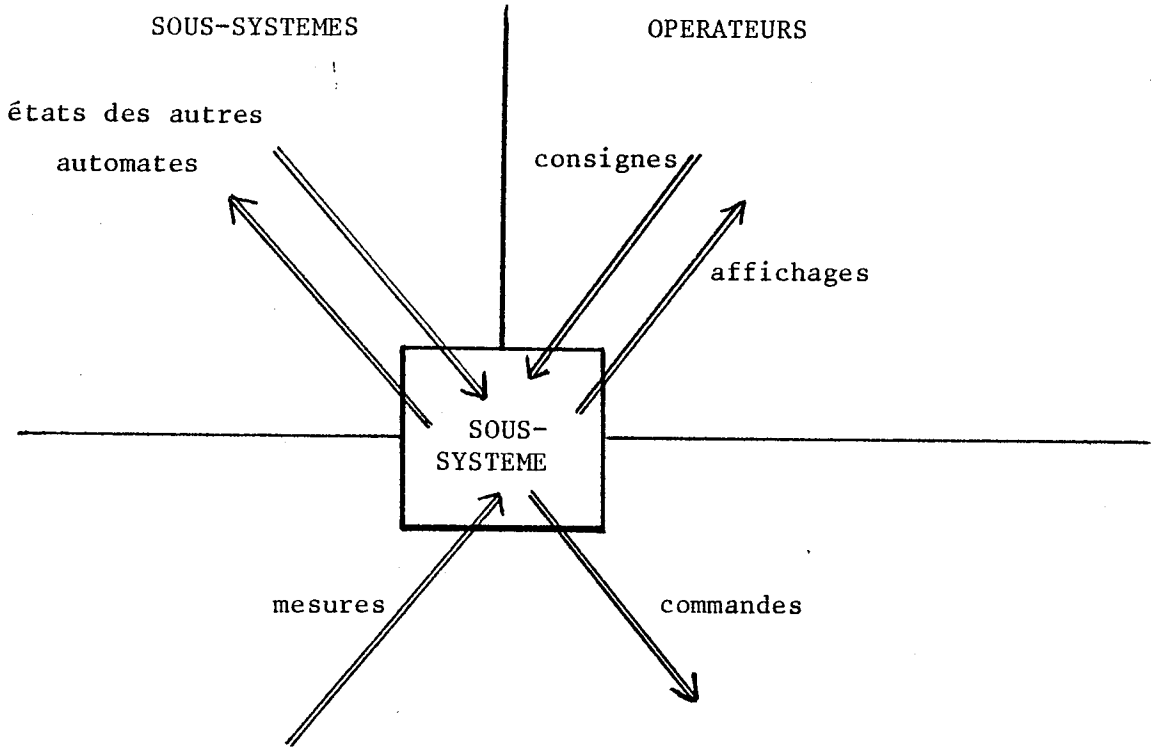


Fig. 5.

Le système lui-même fait l'objet de spécifications de relations entre ses entrées et ses sorties pour lesquelles la nature des problèmes à résoudre fait intervenir fortement le temps.

En général, la description du système fait intervenir :

- des modélisations de type "régulation". Un modèle est obtenu à partir des lois physiques caractérisant le procédé (modèle de connaissance) ou à partir de mesures des entrées et des sorties du procédé (modèle de représentation), et prend la forme d'équations récurrentes sur les instants d'échantillonnage des mesures [Foulard 77] .
- des modélisations de type logique, qui prennent en charge en particulier l'ordonnancement des actions. Elles sont essentiellement basées à l'heure actuelle sur l'utilisation de Réseaux apparentés à ceux de Petri [Valette 76, Moalla 76, Mendelbaum 77]. Il est cependant possible que des outils développés en informatique pour décrire des systèmes opératoires, tels que les expressions de chemin [Lauer 77] et les compteurs de synchronisation [Robert 77] conduisent à la définition de langages de haut niveau adaptés aux besoins de l'automatique.

Enfin, la différence fondamentale entre les 2 types de modélisation conduit à la nécessité de disposer au niveau de l'énoncé explicite d'outils de décomposition correspondants.

II.4.3. - L'énoncé explicite

C'est à ce stade qu'interviennent le langage de programmation défini dans ce mémoire (chapitre IV) et le modèle de description sur lequel il est basé (chapitre III).

La synthèse abstraite de chaque machine correspondant à chaque (sous-système) exhibé dans l'énoncé implicite est éngagée sur la base de sa décomposition en Partie Contrôle et Partie Opérative (voir par exemple [Moalla 76]) où :

- La Partie Contrôle est un automate (modèle logique) qui gère l'enchaînement des actions sur les commandes en fonction de celui d'événements, c'est-à-dire de conditions sur les mesures ou les états des autres systèmes, par rapport auxquelles la Partie Contrôle est placée en attente de leurs réalisations. Les mesures sont effectuées périodiquement. Les actions, ainsi que les fonctions et les prédicats nécessaires aux calculs des événements complexes, sont décrits dans la Partie Contrôle si les procédures associées ne dépendent pas du temps.
- La Partie Opérative contient des modules indépendants décrivant les actions et les événements nécessaires à la Partie Contrôle et non définis dans celle-ci .

Les actions sont donc définies dans l'une ou l'autre Partie. Les liens avec l'opérateur et le procédé sont répartis sur les deux Parties, et certains d'entre eux peuvent être communs.

Les liens entre les différentes machines sont réduits à la possibilité pour l'automate de chaque Partie Contrôle de connaître l'état des automates des autres Parties Contrôle.

Le critère de décomposition précisé dans le chapitre III, revient à distinguer différents rythmes de scrutation des entrées. En ce qui concerne une Partie Contrôle, l'évolution de l'automate, le calcul des événements et l'élaboration des actions sont effectués sur un rythme commun. Tous les calculs qui ne peuvent être définis sur ce rythme le sont dans les modules de la Partie Opérative.

Ce critère présente les avantages suivants :

- il contribue à rapprocher sur une base commune, celui de la scrutation périodique des entrées, les automatismes logiques et de régulation;
- l'utilisateur est contraint de raisonner en termes de Temps Réel, et peut mieux comprendre ainsi le fonctionnement de l'ensemble des algorithmes;
- l'implémentation est mieux guidée : une horloge par module de Partie Opérative (s'il y a lieu) et une pour la Partie Contrôle, regroupement dans l'architecture des éléments concourants aux calculs de même rythme.

II.4.4. - Le programme

Le langage proposé pouvant être utilisé non seulement pour la description mais également pour la programmation du système de conduite, il apparaît donc que le codage de la description dans une mémoire matérielle peut être fait automatiquement par un compilateur. Cependant, celui-ci n'a pas été écrit pour le langage complet, mais seulement pour une version restreinte (voir chapitre VI). D'autre part, les remarques suivantes peuvent être faites :

- l'orientation des réalisations vers des architectures matérielles décentralisées multi-ordinateurs implique la prise en compte de la définition de telles structures par le compilateur : les problèmes posés ne sont pas encore résolus;
- le langage de description des modules des Partie Contrôle et Opérative n'est pas défini et il est souhaitable de pouvoir le choisir en fonction de la classe d'algorithmes propre à chaque application

particulière (l'idée du langage universel de programmation semble abandonnée en informatique après la tentative d'ALGOL 68);
- l'expérience a montré qu'un langage de programmation doit être utilisé expérimentalement pendant plusieurs années avant de pouvoir l'être de façon opérationnelle : l'existence d'un compilateur limiterait une remise en cause du langage.

Par conséquent, l'écriture d'un compilateur n'est peut-être pas la meilleure solution, dans l'immédiat, à la mise en oeuvre du langage. Une autre approche peut être proposée, qui est la suivante : définir des schémas de traduction stricts du langage vers des langages existants de plus bas niveau et pour lesquels existent des compilateurs et des systèmes moniteurs. Ceci rejoint la recherche d'une programmation structurée en Procol, obtenue par adaptation du moniteur Temps Réel associé, faite au Laboratoire d'Automatique de Grenoble [Ladet 77]; une démarche semblable est proposée dans [Le Calvez 77-1]. Pour appuyer cette opinion, signalons également l'expérience satisfaisante basée sur ce principe et faite dans l'enseignement de l'informatique de gestion à l'IUT de Grenoble [Courtin 74, Chiaramella 74].

L'avantage attendu est de pouvoir proposer un langage commun aux utilisateurs tout en leur permettant d'effectuer des réalisations avec les moyens dont ils disposent. La définition de règles de traduction "à la main" est un travail moins important que celui de l'écriture d'un compilateur, et est toujours possible !

Chapitre III

LE MODELE DE DESCRIPTION

Un Système de Conduite est constitué d'un nombre fini de machines pour l'ensemble desquelles est défini un fonctionnement sur la base d'un temps discret : $\tau = \{ s_0, s_1, \ldots \}$.

Ces machines sont en relation avec l'opérateur et le procédé par l'intermédiaire de voies de communication constituant l'Interface du Système de Conduite.

III.1. - L'INTERFACE

L'interface est constituée d'entrées et de sorties.

Les entrées sont les consignes, en provenance de l'opérateur, et les mesures, en provenance du procédé.

Les sorties sont les affichages, en direction des opérateurs, et les commandes, en direction du procédé.

Une entrée est désignée par un nom (identificateur) et est caractérisée par :

- un type : logique, analogique ou alphanumérique (chaîne de caractères)
- un numéro de voie qui dépend du système d'adressage du matériel;
- un traitement pour une entrée de type analogique ou alphanumérique (voir II.4.).

Une sortie est définie de la même façon mais elle peut être également de type impulsion.

L'Interface d'une machine est l'ensemble des entrées et des sorties de l'Interface par lesquelles elle communique avec l'opérateur et le procédé. Une sortie ne peut être commune aux interfaces de plusieurs machines : dans le cas contraire, des aléas de sortie pourraient se produire car les machines ont des fonctionnements parallèles dans le temps.

Les entrées globales sont les entrées communes à au moins 2 machines

L'Interface privée d'une machine est l'Interface de cette machine sans les entrées globales.

III.2. - LA STRUCTURE D'UNE MACHINE

Une machine est constituée d'une Unité de Contrôle (Partie Contrôle) et d'une Unité de Traitement (Partie Opérative).

La décomposition est effectuée en distinguant les calculs d'après les rythmes de scrutation des entrées, qui déterminent généralement celui des évolutions de ces calculs.

L'Unité de Contrôle est une machine séquentielle dont le rythme est celui du temps discret τ du Système de Conduite.

L'Unité de Traitement regroupe les autres calculs, c'est-à-dire :

- Ceux qui ont un rythme différent de celui de τ, par exemple une régulation numérique.
- Ceux qui ont un rythme qui est celui de τ mais qui ne sont définis que pour un intervalle de τ (sous-systèmes constitués de machines interconnectées).
- Ceux qui font référence au temps absolu : il en est ainsi des Temporisations et des actions associées à des délais critiques d'initialisation et de terminaison.

Les calculs définis dans l'Unité de Traitement sont initialisés et éventuellement arrêtés par ceux de l'Unité de Contrôle, ce qui justifie le vocabulaire choisi.

Il est peut-être remarqué que la décomposition est fortement liée au caractère "Temps Réel" des applications envisagées.

III.2.1. - L'Unité de Contrôle

Elle comprend un automate d'états finis déterministe pour lequel des actions sont associées aux états et des conditions d'évolution, appelées aussi événements, aux transitions entre états.

Si l'automate est dans l'état q_1 à l'instant s_k de τ, il évolue vers un état q_2 si une condition d'évolution est définie de q_1 vers q_2 et est vérifiée à l'instant s_k; dans le cas contraire, l'automate reste dans l'état q_1 jusqu'à l'instant s_{k+1} et les conditions d'évolutions évaluées à l'instant s_k le sont alors à nouveau.

L'automate est la donnée d'un triplet (J,Q,q_0,E,T,O) où :

a) J est l'interface de l'Unité de Contrôle.

b) Q est l'ensemble des états et q_0 l'état initial de la machine.

c) E est l'ensemble des événements (conditions d'évolution) élémentaires Un élément de E peut prendre la valeur "1" ou "0" pour tout s_k de τ, mais ne change pas de valeur entre les instants s_k et s_{k+1} ; il peut être associé

- à une entrée logique L de J : il prend la valeur "1" si L est sur son "niveau haut" et "0" si elle est sur son "niveau bas";
- à un état q_i^1 d'une autre machine M_1 du Système de Conduite; il prend la valeur "1" si M_1 est dans l'état q_i^1 à l'instant s_k et "0" dans le cas contraire;
- à un prédicat défini dans l'Unité de Contrôle et calculé à l'instant s_k. Supposons par exemple que t désigne une entrée analogique : un prédicat de l'Unité de Contrôle pourra être "$t \geq c$" où c désigne une constante quelconque; si ce prédicat est requis à l'instant s_k pour évaluer une condition d'évolution, la valeur de t à l'instant s_k est alors comparée à c.
- à un prédicat défini dans l'Unité de Traitement et calculé à un instant antérieur à celui où il est requis; si son calcul n'est pas encore effectué, alors une nouvelle interrogation du prédicat est remise à l'instant s_{k+1} (si un autre événement permettant à l'automate

d'évoluer n'est pas réalisé à l'instant s_k). Par exemple, le prédicat "fin de temporisation sur l'horloge H" peut être un événement conditionnant l'évolution de l'automate; si cet événement n'est pas réalisé à l'instant s_k, la prochaine interrogation du prédicat aura lieu à l'instant s_{k+1}.

L'algèbre (binaire) de Boole construite sur E est notée Σ.

d) $TCQx\Sigma xQ$ est l'ensemble des transitions de l'automate. T doit être tel que :

- le graphe des états qu'il définit soit fortement connexe. Cette propriété apparaît souhaitable pour pouvoir utiliser la notion de "cycle de fonctionnement" d'un Système de Conduite. Elle n'introduit cependant pas de restriction puisqu'il est toujours possible d'ajouter la condition d'évolution toujours vérifiée de chaque "état terminal" vers l'état initial.
- l'automate soit déterministe, c'est-à-dire que (q_1, e_1, q_2) et (q_1, e_2, q_3) étant 2 transitions de T telles que $q_2 \neq q_3$, e_1 et e_2 doivent être exclusifs.

e) $OCQxA$ est l'ensemble des ordres de l'Unité de Contrôle, où A est un ensemble d'actions définies dans l'Unité de Contrôle ou dans l'Unité de Traitement.

Plusieurs actions spécifiées pour un même état ne doivent pas modifier les mêmes sorties.

L'Unité de Contrôle comprend également :

- Les définitions des fonctions et prédicats nécessaires à l'évaluation des conditions d'évaluation, et calculées à l'instant s_k de T où elles sont requises par le fonctionnement de l'automate. Le calcul d'une telle fonction ou prédicat effectué entre les instants s_k et s_{k+1} peut dépendre des valeurs d'entrées au même instant; il ne peut faire intervenir d'autres entrées ou sorties et il ne dépend pas explicitement du temps.

- Les définitions des actions à élaboration immédiate. Une action est à élaboration immédiate si le calcul effectuant cette élaboration doit être exécutée entre les instants s_k et s_{k+1} de τ, si s_k est l'instant où l'ordre d'élaborer l'action a été donné par l'Unité de Contrôle. Le calcul d'une telle action peut modifier les valeurs de sorties (entre s_k et s_{k+1}) et dépendre des valeurs d'entrées à l'instant s_k; il ne dépend pas explicitement du temps .

- L'interface de l'Unité de Contrôle, c'est-à-dire les définitions des entrées et des sorties (identificateurs, numéros de voie, traitements éventuels) utilisées par les fonctions, prédicats et actions définis dans l'Unité de Contrôle.

III.2.2. - L'Unité de Traitement

L'Unité de Traitement est un ensemble d'éléments (actions, prédicats et fonctions) pouvant faire intervenir le temps et en particulier définir une scrutation des entrées différente de celle de l'Unité de Contrôle.

Exemples : 1) Une action d'initialisation et un prédicat de fin de temporisation peuvent être définis pour une "horloge temps réel";

2) Une action assurant la régulation d'un vecteur Y de commandes analogiques à partir d'un vecteur X de mesures analogiques utilise, pour l'interface X, Y, un algorithme de calcul travaillant à partir d'une scrutation périodique de X.
Une telle action pourra être appelée une "machine numérique" par opposition à une machine logique (séquentielle) qu'est une Unité de Contrôle. L'algorithme travaille sur un temps discret qui est indépendant de τ .

Un élément de l'Unité de Traitement dont l'interface comprend des entrées et/ou des sorties logiques ou impulsionnelles doit être décomposé à son tour comme le Système de Conduite initial en une Unité de Contrôle et une Unité de Traitement. Il constitue alors un Sous-Système, dont les Unités de Contrôle des machines constituantes évoluent également sur la base du temps discret τ du système initial. Cette décomposition doit être renouvelée jusqu'à ce que les interfaces des éléments de l'unité de traitement ne comprennent plus d'entrée ou de sortie logique ou impulsionnelle. Cette obligation répond à un objectif de cohérence qui est celui de décrire de la même façon (par des automates) tous les traitements logiques de l'application. c'est dans un but de simplification que le temps discret est le même que toutes les unités de contrôle d'un même système, l'option contraire n'étant pas apparue nécessaire.

III.3. - FONCTIONNEMENT GLOBAL DU SYSTEME DE CONDUITE

III.3.1. - Evolution globale

Elle est définie implicitement par celle décrite pour chaque machine. A un instant s_k de τ, une évolution au plus est effectuée pour chaque automate défini dans l'application. Si le calcul d'une condition d'évolution dépend de valeurs d'entrées et d'états d'autres automates que celui contenant la condition, alors ceux-ci sont "photographiés" à l'instant s_k.

III.3.2. - Synchronisation

La synchronisation de 2 machines est effectuée en faisant dépendre des évolutions de l'automate de l'Unité de Contrôle d'une machine de l'état de celui de l'automate de l'autre machine.

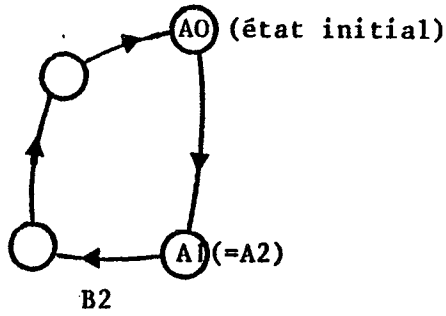
Certaines configurations peuvent conduire à des blocages structurels des automates. Ceci résulte en général d'une mauvaise formalisation d'un problème ou de la donnée d'un énoncé de problème n'ayant pas de solution.

Aussi peut-il être intéressant de dégager certaines conditions relatives à ces blocages. Ceci est fait ci-dessous pour le cas de 2 automates A et B interconnectés de la façon suivante :

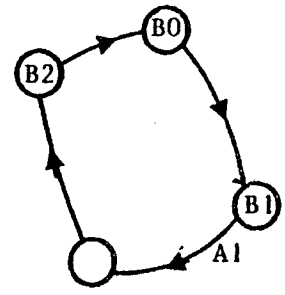
- une évolution de A à partir de A1 nécessite que B soit dans l'état B2;
- une évolution de B à partir de B1 nécessite que A soit dans l'état A2;
- aucune autre évolution de A ne dépend de l'état de B et réciproquement

Une condition suffisante de non blocage est alors : A1 = A2 ou B1 = B2.

En effet, supposons que A1 = A2; B peut évoluer jusqu'en B1 et y est bloqué tant que A n'est pas sur A1; B peut alors évoluer jusqu'en B2 et A peut évoluer à partir de A1 (figure 1).



automate A



automate B

Fig.1.

Le cas B1 = B2 est symétrique du précédent.

Si cette condition n'est pas remplie, les automates peuvent se bloquer réciproquement comme le montrent les exemples des figures 2 et 3.

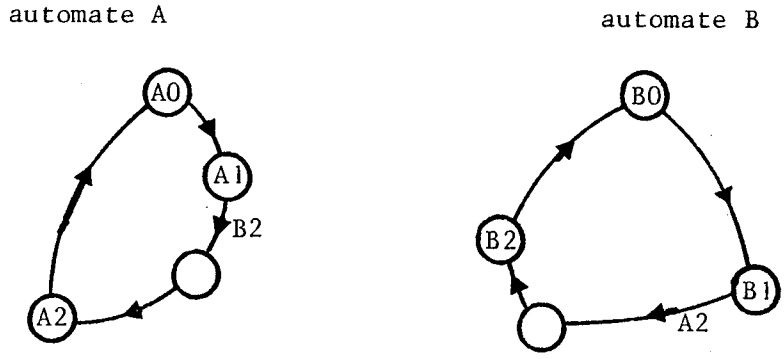


Fig. 2. Exemple de Blocage

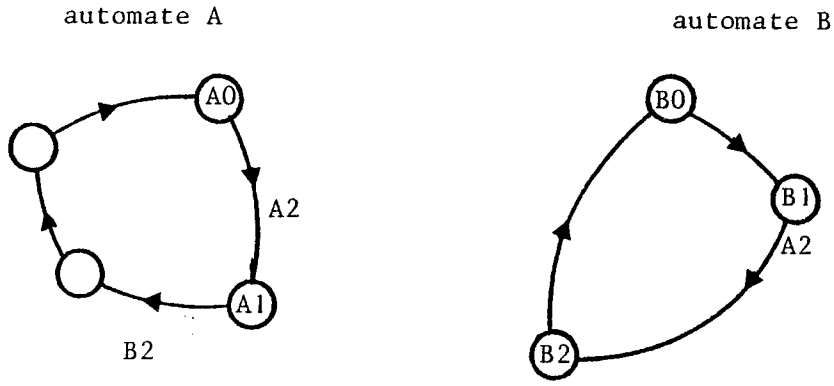


Fig. 3. Exemple de Blocage

Dans l'exemple de la figure 2, A et B évoluent respectivement de A0 et B0 vers A1 et B1 où ils restent bloqués.

Dans celui de la figure 3, si A évolue à partir de A2 avant que B n'ait atteint B1, alors A et B restent bloqués dans les états A1 et B1. Contrairement

au cas précédent, ce dernier cas peut fonctionnellement ne pas introduire de blocage.

Des schémas de synchronisation peuvent être définis par le programmeur en utilisant dans un automate des conditions définies sur les états des autres automates. Les figures 3 et 4 montrent deux exemples de tels schémas (les événements n'intervenant pas dans la synchronisation proprement dite ne sont pas spécifiés).

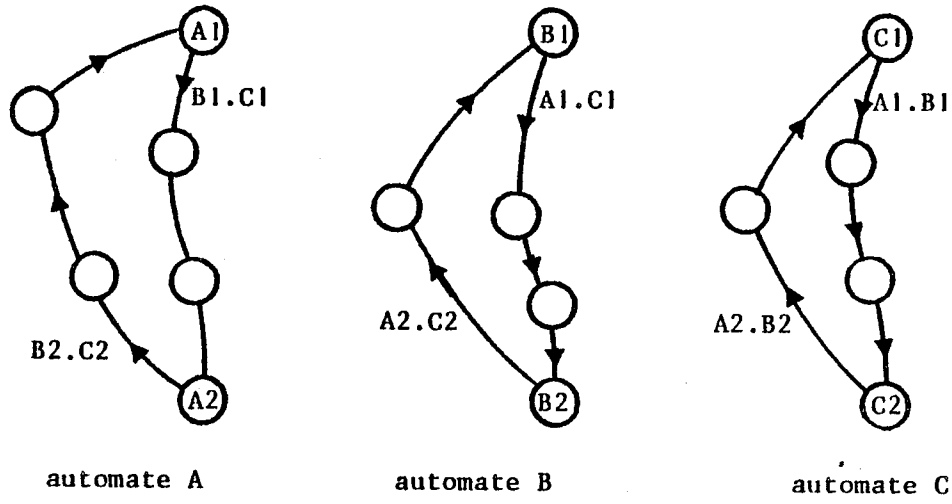


Fig. 4. Parallélisme de 3 chaînes (A1,A2),(B1,B2), (C1,C2).

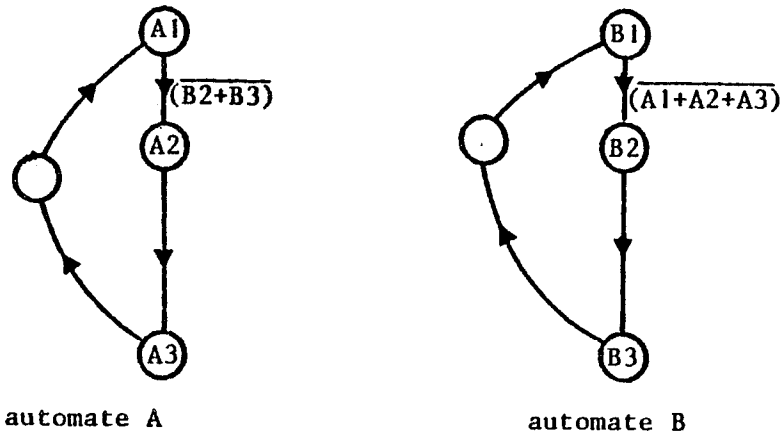


Fig. 5. Exclusion Mutuelle de 2 chaînes (A1,A3) et (B1,B3)

Chapitre IV

LE LANGAGE DE PROGRAMMATION

Le langage proposé permet de décrire et de programmer un Système de Conduite structuré d'après le modèle donné dans le chapitre III. Une syntaxe formelle en est donnée en annexe.

Le formalisme défini se rapporte essentiellement à la description des automates. Les actions, prédicats et fonctions sont décrits par des procédures, regroupées en modules lorsqu'elles partagent des variables communes.

Le langage de programmation de ces procédures n'est pas défini dans ce mémoire et un formalisme proche d'ALGOL 68 [AFCET 72] est utilisé pour les exemples. Le domaine d'application du langage conduit cependant à souhaiter les caractéristiques suivantes :

- une allocation statique de la mémoire, suffisante dans la plupart des cas, et qui permet d'avoir des objets rémanents (par exemple FORTRAN)
- La possibilité de définir, comme par exemple en ALGOL 68, des opérateurs sur des algèbres matricielles , utile en particulier pour la conduite multivariable [Foulard 77].
- La possibilité de tenir compte de la structure du matériel tout en gardant une expression de haut niveau. Ceci peut être utile pour décrire en particulier les traitements associés aux périphériques spécialisés, en particulier les capteurs et les actionneurs analogiques. Les langages MODULA [Wirth 77-1], LIS [Ichbiah 74], et LEST [Bétourné 76] sont orientés dans cette direction; cependant LIS et LEST semblent plus intéressants sur ce point particulier car y est autorisé la description structurée d'un objet à la fois aux niveaux abstrait et concret.

Le chapitre V regroupe des exemples d'applications qui servent de support aux définitions données dans celui-ci.

IV.1. - UTILISATION DE LA NOTION DE MODULE

Des fonctions, prédicats et actions utilisant un objet commun sont regroupés en modules structurés comme ceux définis dans [Cheval 77].

Un module est la mise en oeuvre d'un objet abstrait, c'est-à-dire défini par un ensemble de spécifications. Cet objet ne peut être modifié ou observé qu'au travers de procédures appartenant au module. Une procédure est associée à chaque action, prédicat ou fonction. La description par des expressions de chemins des enchaînements possibles des exécutions des procédures peut être incluse dans le module.

Un module peut être créé :

- soit directement à partir d'une déclaration de la forme :

  module <*idf*>
  .....
  fin <*idf*>

  où <*idf*> est un identificateur désignant le module;

- soit par macro-expansion à partir d'un modèle défini par une déclaration de la forme :

  modèle module <*symb*> *(&P1,...,&Pn)*
  |
  fin <*symb*>

  où <*symb*> est un symbole désignant le modèle et *(&p1,...,&Pn)* est une liste, facultative, de paramètres; un module de modèle < *symb* > peut être créé par une déclaration de la forme :
  < *symb* > < *idf* > $(q1,q2,\ldots,q_n)$ où < *idf* > est une identificateur désignant l'exemplaire créé.

L'utilisation de modules est intéressante car elle permet d'une part de valider l'unité de programmation correspondante indépendamment de son contexte et d'autre part de pouvoir modifier la définition des procédures

également indépendamment du contexte. (Un module pourra même avoir une réalisation matérielle ou logicielle dans le cas de périphériques banalisés sans que la connaissance de ceci soit nécessaire à la définition dans le langage des unités de programme utilisatrices.)

L'appel d'une procédure p incluse dans un module m est décrit sous la forme :

$m\square p(q1,q2,\ldots,q_n)$

où $q1,q2,\ldots,q_n$ représentent les paramètres de la procédure.

En Automatique, les comptages et les temporisations sont fréquemment utilisés. Des définitions possibles pour des modules correspondants sont données ci-après.

1°) Les modules de comptage

La définition d'un modèle de module de comptage peut être la suivante :

```
modèle module compteur
|  entier c;
|  action init = (entier c0):c:=c0;
|  action incr = c+:=1;
|  action decr = c-:=1
|  fonction valeur = entier:c;
|  chemin init*;(incr,decr,valeur)*finch
fin compteur
```

La valeur du compteur est rangée dans c, initialisée par "init", incrémentée (décrémentée) par "incr" ("decr"), et observable par "valeur"; l'expression de chemin impose que "init" soit exécutée avant toute exécution d'une autre procédure.

Un tel compteur est utilisé dans l'exemple donné en V.1.; d'autres modèles peuvent être définis si les primitives désirées sont différentes (par exemple un prédicat fin de comptage et par conséquent la définition dans l'initialisation d'une valeur de comptage). Il peut aussi être remplacé par un module matériel assurant incrémentations et décrémentations.

Un module de comptage peut être créé dans le programme par une déclaration de la forme *compteur* < *idf* > . Si le compteur est matériel, le modèle aura en général un paramètre qui sera le nom de l'entrée numérique correspondante.

2°) Les modules de temporisation

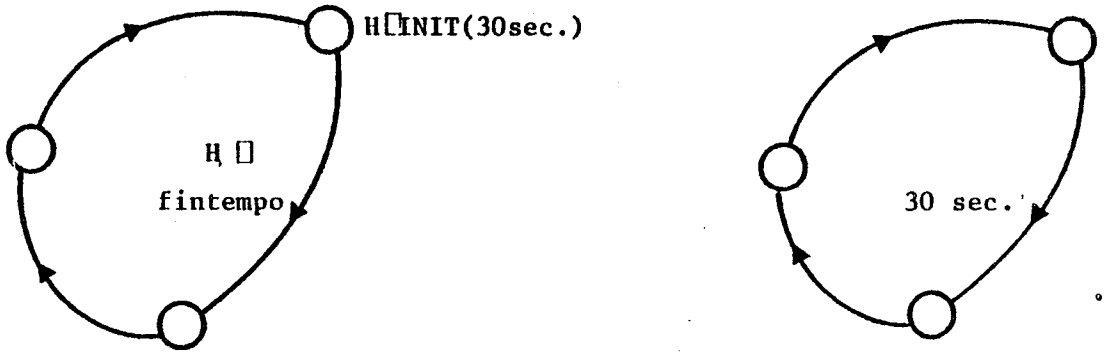
Une horloge "Temps Réel" peut être considérée comme un module matériel pour lequel 2 procédures sont définies :

- une action d'initialisation de l'horloge, dont l'appel est de la forme *init(d)* où d désigne la durée de la temporisation;
- un prédicat *fintempo* permettant de savoir si la temporisation est terminée.

L'enchaînement possible des deux procédures peut être décrit par l'expression de chemins :

chemin (init); (fintempo)*fin chemin*

L'utilisation d'un tel module peut être faite sous forme implicite dans le langage, quand une seule temporisation est utilisée à la fois par un automate et que *fintempo* est appelée seulement sur les transitions issues de l'état où est initialisée la temporisation : l'initialisation de l'horloge n'est pas exprimée et le délai de temporisation est utilisé directement comme prédicat sur les transitions. La figure 1 montre 2 automates équivalents : l'utilisation d'un module de temporisation est explicite pour - a - et implicite pour - b -.



- a - - b -

Fig. 1.

IV.2. - STRUCTURE GENERALE DES PROGRAMMES

La figure 2 montre comment les différents blocs déterminant la structure d'un Système de Conduite sont agencés et délimités à l'aide de symboles appropriés (identificateurs soulignés); les <idfi> désignent des identificateurs tous différents.

Remarques : - l'interface globale est isolée dans le paragraphage du progra
- la description des ordres associés aux états d'un automate est séparée de celle des évolutions de cet automate, afin de rendre le texte plus lisible;
- l'unité de traitement et l'unité de contrôle contiennent des procédures (actions, prédicats, fonctions) ou des modules (quand plusieurs procédures ont des variables communes).

L'unité de traitement peut contenir également des sous-systèmes, entre *système* < *idf* > et *fin* < *idf* >. Des modèles non seulement de modules, mais également d'actions, prédicats, fonctions et machines peuvent être définis;

- le paragraphe entre *automate* et *fin automate* peut être omis : ceci signifie que la machine réalise une fonction combinatoire (elle n'a qu'un seul état, implicite) et les états ne sont alors évidemment pas spécifiés dans les ordres;
- quand l'unité de traitement est vide ou ne contient que des éléments standard , la décomposition unité de contrôle-unité de traitement peut être omise dans le programme.

La description des interfaces est clairement définie dans les exemples. A chaque entrée (ou sortie) sont associés sa catégorie (consigne, mesure, commande ou affichage), son type, son numéro de voie, et, si elle est analogique ou alphanumérique, un traitement (voir II.4.) lié à l'algorithme de conduite.

La description des autres éléments constituant les unités de contrôle et les unités de traitement est donnée dans les paragraphes suivants.

```
système <idf0>
|
|  entrées globales .....................
|
|  machine <idf1>
|  |    unité de contrôle
|  |    |    interface
|  |    |    |.....................
|  |    |    fin interface
|  |    |    modèles, modules, procédures de l'Unité de Contrôle
|  |    |    automate
|  |    |    |.....................
|  |    |    fin automate
|  |    |    ordres
|  |    |    |.....................
|  |    |    fin ordres
|  |    fin contrôle
|  |    unité de traitement
|  |    |    modèles, modules, procédures, sous-systèmes de l'unité de
|  |    |    traitement.
|  |    fin de traitement
|  fin <idf1>
|
|  machine <idfn>
|  |.........
|  fin <idfn>
|
fin <idf0>
```

Fig. 1.

IV.3. - DESCRIPTION DE L'UNITE DE CONTROLE

IV.3.1. - Description des évolutions de l'automate

Une telle description constitue un paragraphe du programme délimité par automate et fin automate et commence par la spécification de l'état initial de l'automate.

Il s'agit de spécifier l'ensemble des transitions d'un automate, c'est-à-dire le graphe, fortement connexe, de ses états et de ses événements. La propriété de connexité forte étant liée à la notion de cycle, la description proposée est essentiellement celle d'un cycle.

L'objectif est de permettre au programmeur de pouvoir réaliser de façon descendante la description d'un automate. Pour cela, la possibilité est donnée de nommer des sous-graphes particuliers et de les décrire séparément.

Dans ce qui suit, un certain nombre de constructions sont définies et le formalisme de description explicité. La construction la plus complexe est appelée un cycle. Dans un programme, la description des évolutions d'un automate est celle d'un cycle.

Soit Q un ensemble d'états, E un ensemble d'événements et Σ l'algèbre de Boole construite sur E. Soit $T \subset Q \times \Sigma \times Q$ un ensemble de transitions tel que pour tout $(p,e,q) \in T$, la relation $p \neq q$ soit vérifiée:

- p est appelé le début de t et noté d(t),
- q est appelé la fin de t et noté f(t),
- e est appelé l'événement de t,
- $\{p,q\}$ est appelé l'extension de t et noté $\tilde{t}$.

Ces notations sont étendues aux constructions ci-après; d'autre part, $\forall\, p \in Q$, l'ensemble $\{p\}$ est noté $\tilde{p}$.

A) Une chaîne simple sur T est un tuple

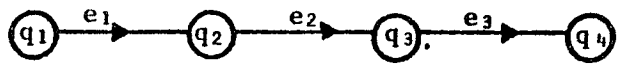
$ch = (t_1,\ldots,t_p)$, $p \geq 1$, tel que :

- $t_i \in T$, $i=1,\ldots,p$
- $f(t_i) = d(t_{i+1})$ $i=1,\ldots,p-1$
- $d(t_i) \neq f(t_j)$ $i,j=1,\ldots,p$ et $i \neq j+1$

Par définition, $d(ch) = d(t_1)$, $f(ch) = f(t_p)$ et $\overset{\sim}{ch} = \bigcup_{i=1}^{p} \tilde{t}_i$

Dans le langage, une chaîne simple (figure 3) $((q_1,e_1,q_2),(q_2,e_2,q_3),(q_p,e_p,q_{p+1})$ est notée :

q1:e1→q2:e2→q3:...→qp:ep→qp+1



notation : *q1:e1→q2:e2→q3:e3→q4*

Fig. 3. Exemple de chaîne simple

B) Une multichaîne simple sur T est un tuple

$mch = (\alpha_1,\alpha_2,\ldots,\alpha_p)$, $p > 1$, tel que :

- α_i est une chaîne simple;
- $d(\alpha_1) = \ldots = d(\alpha_p) = \delta$;
- $f(\alpha_1) = \ldots = f(\alpha_p) = \lambda$;
- $\tilde{\alpha}_1 \wedge \ldots \wedge \tilde{\alpha}_p = \{\delta,\lambda\}$.

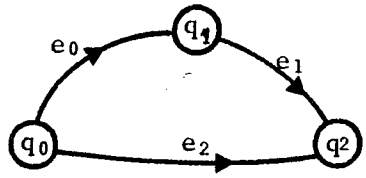
Par définition, $d(mch) = \delta$, $f(mch) = \phi$ et $\tilde{mch} = \bigcup_{i=1}^{p} \tilde{\alpha}_i$

Dans le langage, une multichaîne (figure 4)

$(((q_d, e_{1,1}, q_{1,2}).....(q_1,n_1, e_1,n_1, q_f)),$
$\vdots$
$((q_d, e_{p,1}, q_{p,1}).....(q_p,n_p, e_p,n_p, q_f)))$

est notée :

qd:(e11→q12:.....q1n1:e1n1,
⋮
ep1→qp1:..... qpnf:epnp)→qf

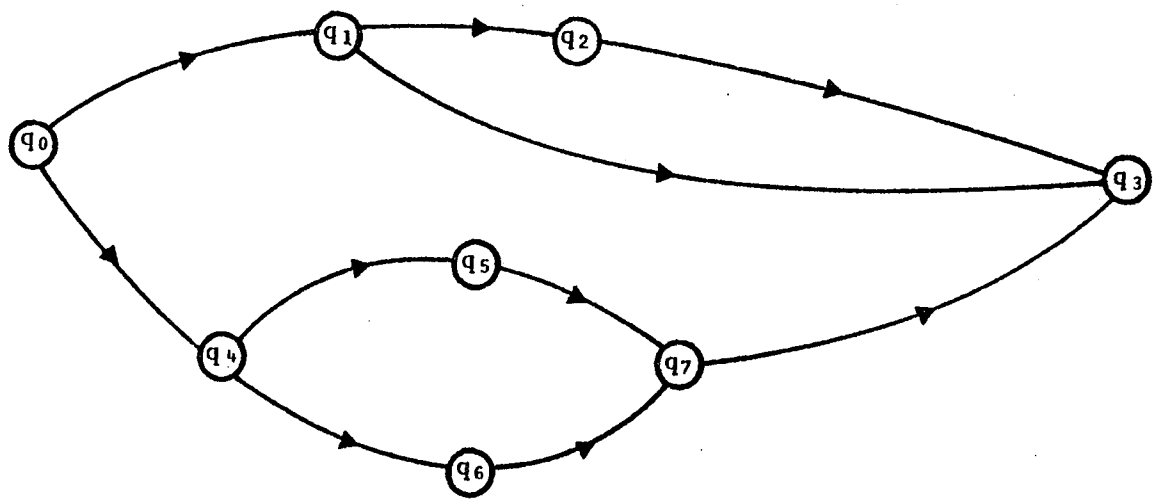


notation : *q0:(e2,e0→q1:e1)→q2*

Fig.4. Exemple de multichaîne simple

C) Une chaîne sur T est
- soit une chaîne simple,
- soit une multichaîne simple,
- soit obtenue à partir d'une chaîne ch_0 en remplaçant une de ses transitions t = (p,e,q) par une chaîne simple ou une multichaîne simple ch_1 telle que $d(ch_1) = p$, $f(ch_1) = q$, et $\widetilde{ch_0} \wedge \widetilde{ch_1} = \{p,q\}$.

La notation de la chaîne résultante est alors obtenue à partir de celle de ch_0 de la façon suivante : si $p:\varepsilon\rightarrow q$ est la notation de ch_1, et e, l'événement de t, la notation de e dans ch_0 est remplacée par ε. Un exemple en est donné sur la figure 5.



Si e_{ij} désigne l'événement entre q_i et q_j, la notation est :

q0:(e01→q1:(e12→q2:e23,e13),
e04→q4:(e45→q5:e57,e46→q6:e67)→q7:e73)→q3

Fig. 5. Exemple de chaîne

D) Un cycle sur T est un tuple

$C = (\gamma_1, \tau_{1,2}, \gamma_2, \ldots, \gamma_{p-1}, \tau_{p-1,p}, \gamma_p, \tau_{p,1})$

tel que

1°) γ_i est une chaîne ou un cycle, ou un état $q \in Q$;

2°) $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \phi$, $1 < |i-j| < p-1$;

3°) $\tau_{i,j}$ est un ensemble de transitions , appelé liaison de γ_i vers γ_j, telle que :

a - Si γ_i et γ_j sont deux chaînes :

$\tau_{i,j} = \phi$ et $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \{f(\gamma_i)\} = \{d(\gamma_j)\}$

b - Si γ_i est une chaîne et γ_j est un cycle :

$\tau_{i,j} = \phi$ et $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \{f(\gamma_i)\}$

ou

$\tau_{i,j} = \phi$, $\gamma_i \wedge \gamma_j = \phi$ et $\forall t = (p,e,q) \in \tau_{i,j}$: $p=f(\gamma_i)$ et $q \in \gamma_j$

c - Si γ_i est un cycle et γ_j est une chaîne :

$\tau_{i,j} = \phi$ et $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \{d(\gamma_j)\}$ ou

$\tau_{i,j} \neq \phi$, $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \phi$ et $\forall t = (p,e,q) \in \tau_{i,j} : p \in \tilde{\gamma}_i$, $q = d(\gamma_j)$

d - Si γ_i et γ_j sont deux cycles :

$\tau_{i,j} = \phi$ et $\exists_q$ tel que $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \{q\}$

ou

$\tau_{i,j} \neq \phi$, $\tilde{\gamma}_i \wedge \tilde{\gamma}_j = \phi$, et $\forall t = (p,e,q) \in \tau_{i,j} : p \in \tilde{\gamma}_i$ et $q \in \tilde{\gamma}_j$.

La notation d'un cycle est obtenue en juxtaposant celles des γ_i et $\tau_{k,1}$ (figure 7), et en les plaçant entre 2 crochets. Une liaison $\tau_{i,j}$ peut prendre différentes formes suivant la nature de γ_i et γ_j :

- Une liaison de chaîne à cycle est notée :

  :(e1→p1,...,ek→pk) ;

  si k=1, la notation est : *:e1→p1* ;

  si k=1 et γ_j est de la forme *[p1:.....]* , la notation est *:e1→* .

- Une liaison de cycle à chaîne est notée :

  (p1:e1,...,pk:ek)→ ;

  si k=1, la notation est : *p1:e1→* ;

  si k=1, et γ_i est de la forme *[p1:.....]* , la notation est *:e1→* .

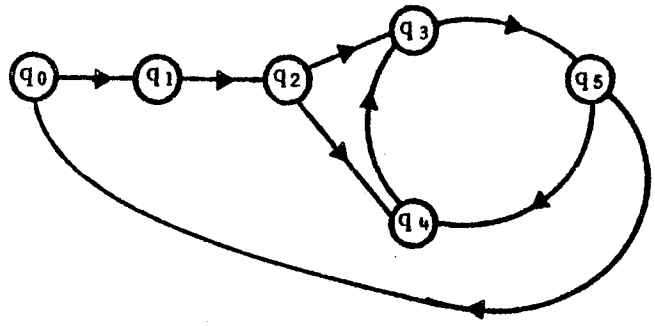
- Une liaison de cycle à cycle est notée :

  p1:e1→q1,p2:e2→q2,...,pk:ek→qk .

- Si γ_i et γ_j sont deux chaines de notations respectives *p:ω→q* et *q:ψ→r* , l'ensemble est noté *p:ω→q:ψ→r*

Enfin, si $\varepsilon\rightarrow$ est la notation de $\tau_{p,1}$, celle-ci peut être abrégée en: ε (exemple figure 7).

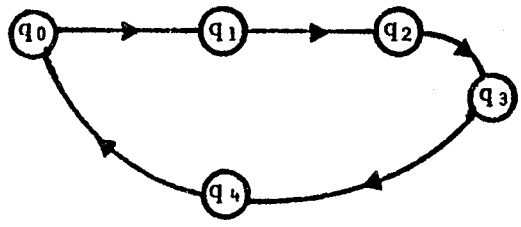
Les figures 6, 7 et 8 présentent exemples d'application du formalisme qui vient d'être défini, où e_{ij} représente l'événement de la transition ayant comme début q_i et comme fin q_j.



notation :

[(q0:e01→q1:e12→q2):(e23→q3,e24→q4)
[q3:e35→q5:e54→q4:e43]q5:e50→]

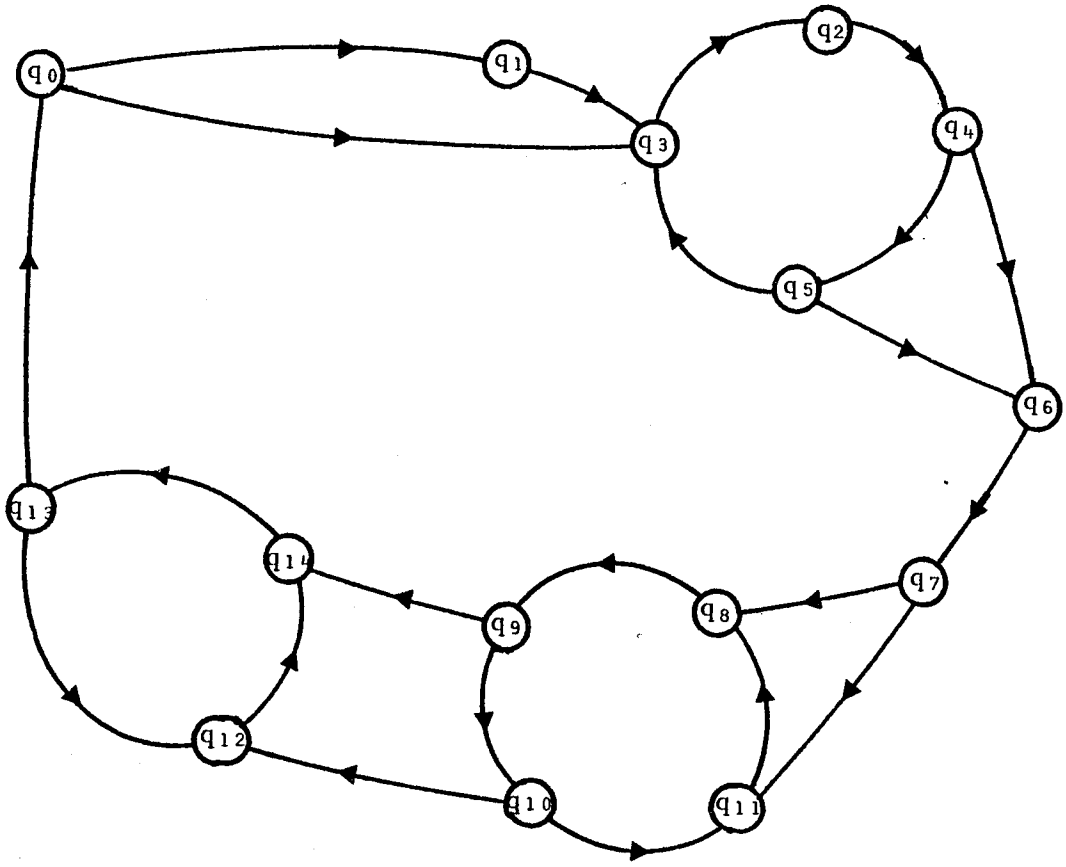
Fig. 6.



notation :

[q0:e01→q1:e12→q2:e23→q3:e34→q4:e40]

Fig. 7



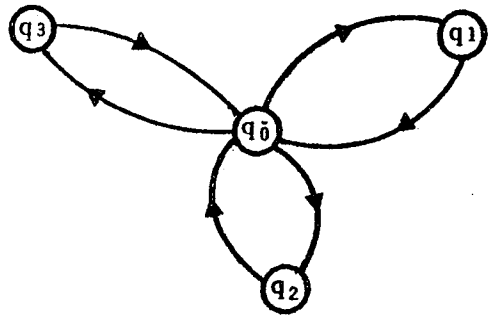
notation :

[q0:(e03,e01→q1:e13)→[q3:e32→q2:e24→q4:e45→q5:e53]
(q4:e46,q5:e56)→q6:e67→q7:(e78→q8,e7.11→q11)
[q11:e11.8→q8:e89→q9:e9.10→q10:e10.11]
(q9:e9.14→q14,q10:e10.12→q12)
[q12:e11.14→q14:e14.13→q13:e13.12] q13:e13.0]

Fig. 7.

Notation d'un cycle dans un cas particulier

Un cycle de la forme $C = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$, où $\gamma_1, \gamma_2, \ldots, \gamma_p$ sont p cycles ayant un et un seul état commun, peut être noté en faisant se suivre les notations des différents γ_i, séparées par des virgules et entourées par des crochets (voir exemple de la figure 9).



notations possibles :

α) *[q0:(e01→q1:e10,e02→q2:e20,e03→q3:e30)]*

β) *[[q0:e01→q1:e10],[q0:e02→q2:e20],[q0:e03→q3:e30]]*

Fig. 9.

Remarques sur les structures de graphes possibles

Pour être facilement utilisables, les règles de construction doivent être simples et ne pas introduire de restrictions trop importantes. Les règles proposées ici ne permettent pas de construire tout graphe fortement connexe, par exemple celui de la figure 10. Elles sont apparues cependant suffisantes pour traiter les applications étudiées, en particulier en ce qui concerne les exemples du chapitre V.

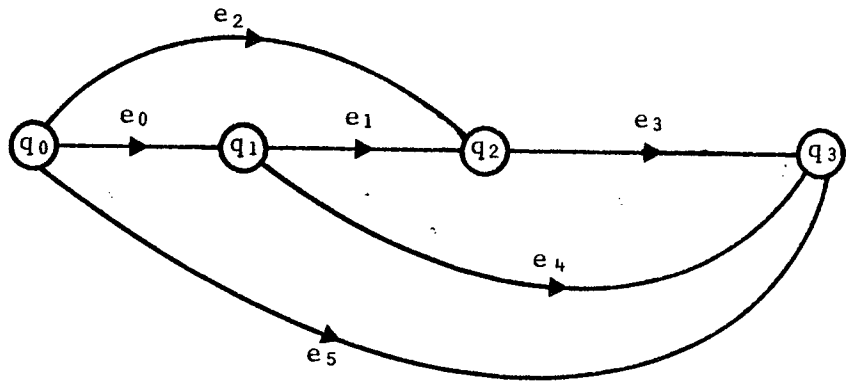


Fig. 10.

Remarquons qu'il est possible de trouver un automate déterministe équivalent à tout automate donné initial et qui respecte les règles de construction définies précédemment: il faut en général adjoindre des états supplémentaires à l'automate initial; ainsi, l'automate de la figure 11 est équivalent à celui de la figure 9, un état q'_2 étant ajouté où sont supposées spécifiées les actions associées à q_2.

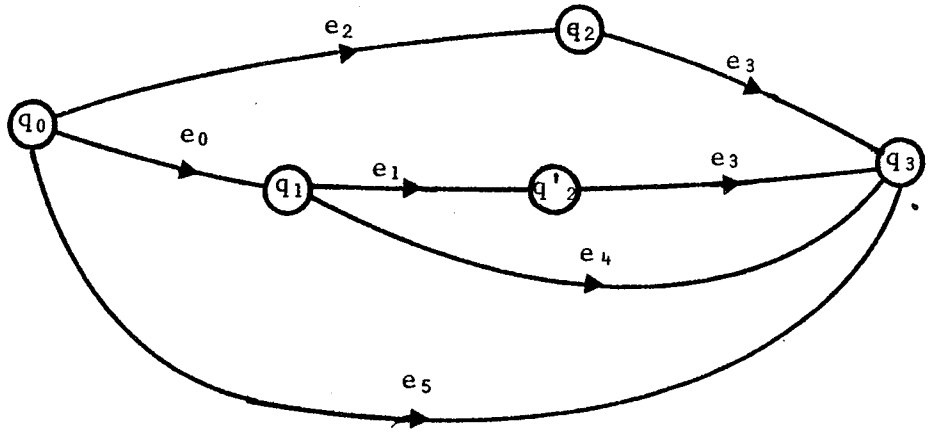


Fig. 11.

Description séparée des chaînes et des cycles

Afin d'obtenir une description plus synthétique de l'automate et de le construire de façon descendante, un cycle (resp. une chaîne) peut être désigné par un nom (identificateur) entre crochets (respec. entre parenthèses) dans un cycle ou une chaîne qui le contient; il doit alors être défini séparément par une phrase de la forme " <idf> = Φ" où <idf> est le nom et Φ la notation du cycle ou de la chaîne considéré.

IV.3.2. - Description des événements

Un événement est spécifié par une expression logique, décrite à l'aide des opérateurs "+" (union), "." (intersection) et "—" (négation), et construite à partir d'événements élémentaires; l'intersection est prioritaire par rapport à l'union.

Un événement élémentaire est une entrée logique, une référence à l'état d'une autre machine, ou un appel de prédicat; il est respectivement spécifié par l'identificateur défini pour l'entrée, le nom de la machine suivi du nom de l'état entre parenthèses, l'appel de la procédure correspondante.

Un prédicat ne peut modifier d'objet externe à lui-même; cette condition permet de terminer le calcul d'une expression logique sans nécessairement calculer tous les opérandes.

Enfin, pour comparer les valeurs d'entrées analogiques avec d'autres entrées de même type ou avec des objets internes de type réel, des prédicats de comparaison sont utilisés sous la forme d'opérateurs $\leq$, $\geq$, $=$, etc. (voir exemple V.4.).

Quand plusieurs conditions d'évolution sont spécifiées pour un même état, elles sont évaluées dans l'ordre où elles se présentent dans le texte. Ceci permet d'alléger l'écriture du programme (voir en particulier

l'exemple VI.3.) et rend le programme déterministe en ce qui concerne les évolutions des automates. Par exemple, si a, b et c sont des entrées logiques, la phrase :

$$[q0:(a+b,c \rightarrow q1:d) \rightarrow q2:e]$$

est équivalente à la phrase :

$$[q0:(a+b,(\overline{a+b}).c \rightarrow q1:d) \rightarrow q2:e]$$

IV.3.3. - Description des ordres

Les ordres contenus dans une Unité de Contrôle sont décrits dans un paragraphe délimité par les symboles ordres et fin ordres, et constitué de phrases de la forme

$$q: \quad \alpha_1, \alpha_2, \ldots, \alpha_n;$$

où q désigne un état et $\alpha_1, \alpha_2, \ldots, \alpha_n$ les n actions associées à q.

α_i est soit une action élémentaire (IV.3.4.), soit un appel de procédure, soit une activation de sous-système(IV.4.1.).

Un sous-système est défini dans l'Unité de traitement.
Un appel de procédure définie dans l'Unité de Contrôle, ou une action élémentaire, peut être élaborée:

a) soit entre l'instant s_k de τ où l'automate évolue vers l'état q et l'instant s_{k+1}: l'action est de la forme $a(p1,p2,\ldots,pn)$ où $p1,p2,\ldots,pn$ représente une liste éventuelle de paramètres.

b) soit pour tout instant s_k de τ où l'automate est dans l'état q, et à chaque front de montée d'une condition logique: l'action est de la forme $a(p1,p2,\ldots,pn)/\ell$ où ℓ représente la condition logique.

exemples : - si s est une sortie logique et e,f,g des entrées logiques, $s/e+f.\bar{g}$ décrit l'asservissement de s à la fonction combinatoire $e+f.\bar{g}$;

- si *cpt* est un module de modèle *compteur* et *e* une entrée logique, l'action *cpt□incr/e* compte le nombre de fronts de montée de l'entrée à niveau *e*. Les 2 automates de la figure 1 sont équivalents. Ceci montre comment, à l'aide de ce type d'action, réduire le nombre d'états d'un automate .

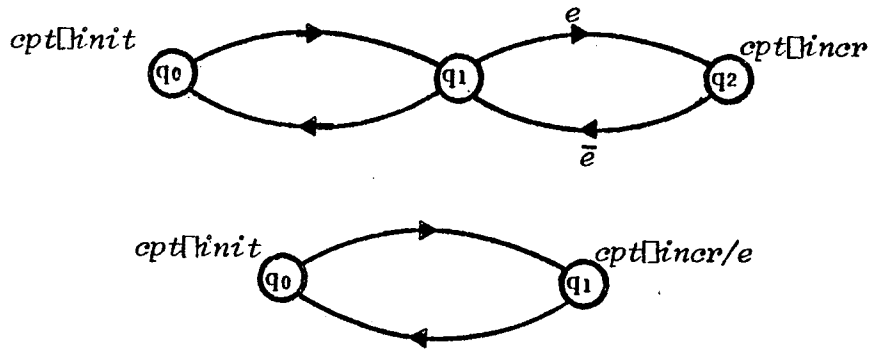


Fig. 12.

Un appel de procédure définie dans l'Unité de Traitement est de la forme :

b(p1,...,pnb) de *d1* à *d2*

où : -*b* désigne l'action et *(p1,...,pnb)* une liste facultative de paramètres.

-*d1* et *d2* sont deux intervalles de temps absolu , facultatifs, tels que *d1<d2* :

d1 est le délai critique d'initialisation spécifiant que l'action *b* doit être initialisée avant l'instant "s_k+*d1*";

d2 est le délai critique de terminaison spécifiant que l'action *b* doit être terminée avant l'instant "s_k+*d2*"; il ne doit pas être spécifié si *b* est un appel de procédure périodique (IV.4.2 ou l'activation d'un sous-système.

IV.3.4. - Les actions élémentaires

IV.3.4.1. - Lancements d'impulsions

L'action consistant à lancer une impulsion est spécifiée dans un ordre par l'identificateur associé à la sortie correspondante.

Ainsi, une commande définie dans l'interface par

"commande impulsion p=7"

peut être activée, pour un état q d'un automate, par un ordre *"q:p"*

Le calibre de l'impulsion est déterminé par le matériel. Il doit être inférieur à tout intervalle $(s_{k+1}-s_k)$ car si l'automate peut lancer une impulsion à l'instant, il peut également le faire à l'instant s_{k+1}.

IV.3.4.2. - Positionnement des sorties analogiques et alphanumériques

L'ordre correspondant est décrit par l'identificateur de la sortie, suivi de la valeur entre parenthèses; le traitement (conversions) est implicite.

IV.3.4.3. - Positionnement des sorties logiques

Il est nécessaire de spécifier pour tous les états de l'automate les valeurs de toutes les sorties logiques ("sorties à niveau"). La convention suivante permet d'alléger considérablement l'écriture (en particulier en ce qui concerne les automatismes logiques comme celui de l'exemple V.1.) : un positionnement à "1" d'une sortie logique est spécifié dans un ordre par l'identificateur associé à cette sortie, un positionnement à "0" n'est pas spécifié et est donc implicite.

En général, l'état initial d'un automate est un "état de repos" où toutes les sorties logiques sont positionnées à "0".

IV.4. - DESCRIPTION DE L'UNITE DE TRAITEMENT

L'unité de traitement peut contenir des sous-systèmes et des procédure (actions, fonctions, prédicats) éventuellement regroupées en modules.

IV.4.1. - Les sous-systèmes (Exemple V.4.)

Une machine contenue dans un sous-système a la même définition qu'une machine contenue dans le système global. Son interface est l'union de son interface privée et de l'interface de la machine spécifiant l'ordre d'activation du sous-système. Si q est l'état auquel est associé cet ordre, aucune sortie utilisée par le sous-système ne doit aussi être utilisée dan un autre ordre associé à q.

Chaque machine du sous-système est activée sur son état initial et fonctionne sur la base de temps du système global. Toutefois, les évolutions du système sont calculées avant celles du sous-système, ce qui peut permettre par exemple la mise en place d'une structure de remplacement. Il apparaît donc ainsi une hiérarchie des événements et des actions qu'ils entraînent (traitement des alarmes).

IV.4.2. - Les procédures

Une période peut être associée à chaque procédure. Dans ce cas, la procédure est exécutée au rythme de cette période, tant que l'automate de la Partie Contrôle de la machine reste dans l'état où est spécifié l'ordre d'activation. La procédure est alors décrite comme dans l'exemple ci-dessous :

action a(5")(p1,p2)=(.....);

où *5"* représente la période et *p1,p2* des paramètres de l'action. Ceci permet d'insérer de façon concise la spécification d'algorithmes de régulation continue.

Comme pour l'Unité de Contrôle, des procédures peuvent être regroupées en modules si elles partagent des objets communs. Cependant, les modules peuvent ici contenir également la définition d'un interface propre.

Chapitre V

EXEMPLES D'APPLICATION

Les quatre systèmes de conduite décrits dans ce chapitre ne comportent que le niveau de conduite directe.

Les trois premiers exemples concernent des automatismes logiques (commandes logiques seulement) pour chacun desquels le procédé a une structure unique. Les unités de traitement comprennent des modules standard de temporisation, dont les procédures sont utilisées de façon implicite : elles n'apparaissent donc pas dans les programmes.

Le dernier exemple concerne un automatisme général (commandes logiques et analogiques) où le procédé a une structure variable.

Enfin, le but étant surtout de donner des exemples de programmes, la présentation suivante est adoptée : un résumé de l'analyse, suivi de la modélisation des automates sous forme graphique, suivi du programme. Un formalisme proche de celui d'ALGOL 68 [AFCET 72] est utilisé pour décrire les procédures.

V.1. - CONTROLE D'UN CARREFOUR ROUTIER

Cette application est un automatisme strictement logique, dans le sens où toutes les mesures et les commandes sont de type logique. Le système de conduite comprend une seule machine.

Le problème et son analyse sont détaillés dans [Weiss 74]; une solutio à l'aide de compteurs de synchronisation peut être étudiée dans [Robert 77].

Résumé de l'analyse - Il s'agit d' optimiser le trafic du carrefour routier schématisé par la figure 2, où A, B, C, D, E et F représentent des couloirs de circulation munis de feux tricolores. Divers enchaînement

de configurations de feux peuvent être définis, qui doivent bien entendu servir tous les couloirs. A la fin de chaque enchaînement, le suivant est choisi de façon à ce qu'un temps de feu vert supplémentaire soit accordé à certains couloirs suivant un critère (voir module *s* du programme) qui dépend des flux calculés au cours de l'enchaînement précédent.

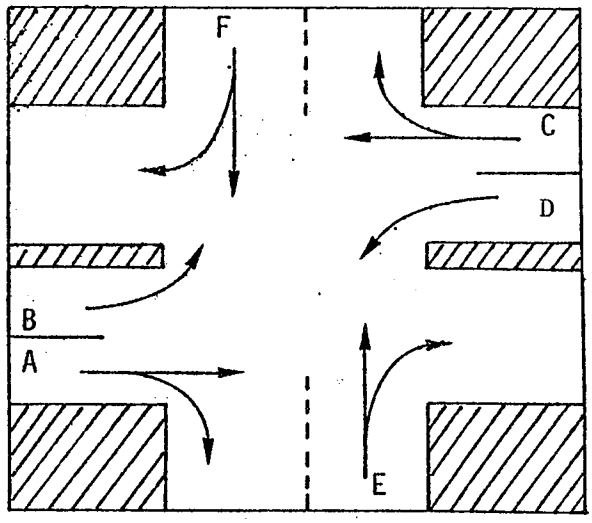


Fig. 1.

Une analyse préalable conduit à retenir cinq configurations possibles de feux (Table 1) et six enchaînements (Table 2).

	A	B	C	D	E	F
conf 1	R	V	R	V	R	R
conf 2	V	V	R	R	R	R
conf 3	V	R	V	R	R	R
conf 4	R	R	V	V	R	R
conf 5	R	R	R	R	V	V

TABLE 1

(R = Rouge, V = Vert, conf = configuration

ench 1 : conf 2, conf 4, conf 5
ench 2 : conf 2, conf 1, conf 4, conf 5 (B,D)
ench 3 : conf 2, conf 2, conf 4, conf 5 (A,B)
ench 4 : conf 2, conf 3, conf 4, conf 5 (A,C)
ench 5 : conf 2, conf 4, conf 4, conf 5 (C,D)
ench 6 : conf 2, conf 4, conf 5, conf 5 (E,F)

TABLE 2

(ench = enchaînement

Ench 1 représente un enchaînement de base où toutes les voies ont un temps égal de feu vert. les autres enchaînements se déduisent du précédent par l'insertion d'une configuration supplémentaire qui permet d'accorder un temps de feu vert supplémentaire à deux couloirs particuliers parmi les six : dans la Table 2, les noms des couloirs privilégiés par chaque enchaînement ont été mis entre parenthèses.

Le flux de chaque couloir est calculé à l'aide de six compteurs incrémentés à chaque passage d'un véhicule au feu du couloir correspondant.

Modélisation - Une première idée serait de construire un automate avec un cycle simple pour chaque enchaînement. Cependant, il a déjà été noté que les cinq derniers enchaînements sont construits à partir du premier par insertion d'une configuration particulière : ceci est utilisé pour construire un automate plus concis (figure 2). Pour ne pas alourdir le dessin, les ordres n'y ont pas été représentés. Dans un état dont le nom est de la forme CONFI ou CONFI1, la configuration numéro i est établie sur le carrefour; dans un état dont le nom est de la forme Pij ou Pijk, des feux jaunes sont positionnés pour assurer le Passage de pour assurer le Passage de la Configuration de numéro i à celle de numéro j.

Programmation

Les identificateurs suivants sont utilisés :

- *a,b,c,d,e,f* : numéros *1,2,3,4,5,6* respectivement associés aux couloirs de même nom;
- *ea,eb,ec,ed,ee,ef* : désignent chacun une entrée logique positionnée à "1" quand un véhicule est en cours de franchissement du feu du couloir correspondant;
- *ra,ja,va,rb,jb,vb,* etc. : commandes logiques actionnant les feux (*ra*= Rouge sur A, *ja* = Jaune sur A , *va* = Vert sur A , etc.);
- *cpt(1), cpt(2),..., cpt(6)* : modules standards de comptage associés à chaque couloir;
- *tb* : Temps de Base (30 secondes) affecté à chaque configuration;
- *tj* : Temps affecté aux feux Jaunes (5 secondes);
- *ts* : Temps Supplémentaire éventuellement affecté à certains couloirs au cours d'un enchaînement (20 secondes);
- *s* : désigne le module défini pour le choix de l'enchaînement suivant. Ce module comprend :

  . une procédure *choix* qui calcule le numéro de l'enchaînement suivant.

. une procédure entière *ench* qui permet d'observer la valeur de ce numéro.

Remarques : - il est suggéré une déclaration globale des entrées et des sorties logiques lorsque les numéros de voies se suivent;
- dans le même ordre d'idées, la création des six modules de modèle standard compteur est déclarée sous la forme *(1:6)compteur cpt ;*
- les intervalles de temps sont exprimés en secondes.

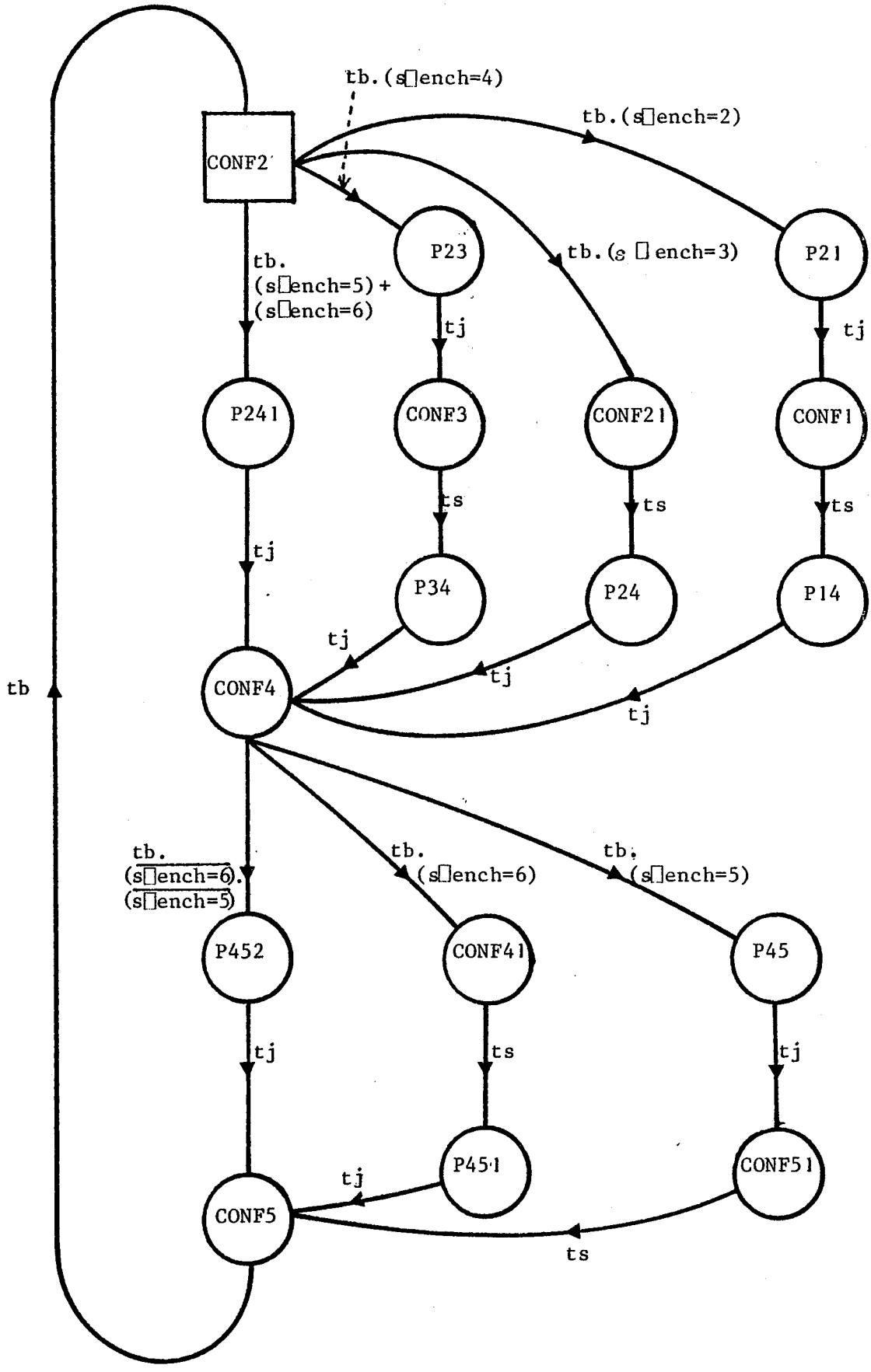

tb.(s□ench=4)
tb.(s□ench=2)
CONF2
P23
tb.(s □ench=3)
P21
tb.
(s□ench=5)+
(s□ench=6)
tj
tj
P241
CONF3
CONF21
CONF1
ts
ts
ts
tj
P34
P24
P14
tj
tj
tj
tb
CONF4
tb.
(s□ench=6).
(s□ench=5)
tb.
(s□ench=6)
tb.
(s□ench=5)
P452
CONF41
P45
tj
ts
tj
P451
CONF51
tj
CONF5
ts

Fig. 2.

```
système      C A R R E F O U R

  machine M

    interface
      mesure logique (ea,eb,ec,ed,ee,ef) = (1:6);
      commande logique (ra,ja,va,rb,jb,vb,rc,jc,vc,
                        rd,jd,vd,re,je,ve,rf,jf,vf) = (1:18);
    fin interface

    entier a=1,b=2,c=3,d=4,e=5,f=6;  délai tb=30,tj=5,ts=20;

    (1:6)compteur cpt;

    module s
      entier numéro enchaînement;
      action choix = (co les flux sont comparés à un seuil et l'algorithme
        calcule le numéro du couloir qui a le plus grand flux parmi ceux qui
        ont dépassé le seuil; un enchaînement est alors choisi co
        entier seuil=20; entier max:=1, flux max:=seuil;
        (1:6)entier num ench = (3,2,4,5,6,6);
        pour i depuis 1 jqa 6
        faire
          si cpt(i)□valeur>flux max
          alors max:=i; flux max:=cpt(i)□valeur fsi
        finfaire;
        numéro enchaînement := num ench(max);
        pour i depuis 1 jqa 6 faire cpt(i)□init(0) finfaire   );
      fonction entier ench = numéro enchaînement;
      chemin choix;(ench)* finch
    fin s;
```

```
automate   état initial:CONF2;

[(CONF2 A CONF4)(CONF4 A CONF5)(CONF5:tb→CONF2)];

CONF2 A CONF4 = CONF2:(tb.(s□ench=4)→P23:tj→CONF3:ts→P34:tj,
                       tb.(s□ench=3)→CONF21:ts→P24:tj,
                       tb.(s□ench=2)→P21:tj→CONF1:ts→P14:tj,
                       tb→P241:tj)
                                  →CONF4;

CONF4 A CONF5 = CONF4:(tb.(s□ench=6)→CONF41:ts→P451:tj,
                       tb.(s□ench=5)→P45:tj→CONF51:ts,
                       tb→P452:tj)
                                  →CONF5;
fin automate

ordres
  CONF1:ra,vb,rc,vd,re,rf,cpt(b)□incr/eb,cpt(d)□incr/ed;
  (CONF2,CONF21):va,vb,rc,rd,re,rf,cpt(a)□incr/ea,cpt(b)□incr/eb;
  CONF3:va,rb,vc,rd,re,rf,cpt(a)□incr/ea,cpt(c)□incr/ec;
  (CONF4,CONF41):ra,rb,vc,vd,re,rf,cpt(c)□incr/ec,cpt(d)□incr/ed;
  (CONF5,CONF51):ra,rb,rc,rd,ve,vf,cpt(e)□incr/ee,cpt(f)□incr/ef;
  P21:ja,vb,rc,rd,re,rf;   P23:va,jb,rc,rd,re,rf;
  (P24,P241):ja,jb,rc,rd,re,rf;
  P34:ja,rb,vc,rd,re,rf;   P14:ra,jb,rc,vd,re,rf;
  (P45,P451,P452):ra,rb,jc,jd,re,rf;
  co les passages de véhicules franchissant les feux jaunes
     sont négligés co
fin ordres

fin M

fin     C A R R E F O U R
```

V.2. - SYNCHRONISATION DE DEUX CHARIOTS

Le système de conduite comprend trois machines synchronisées de façon à réaliser une exclusion mutuelle entre des actions sur le procédé; une quatrième machine assure l'établissement d'un bilan et l'édition, à la demande de l'opérateur, d'un journal.

La modélisation de ce type de problème à l'aide de Réseaux de Petri peut être étudiée pour un cas complexe dans [Moalla 76] (circuit ferroviaire triangulaire partagé par un nombre quelconque de trains).

Résumé de l'analyse - Deux chariots A et B transportent du matériel, respectivement depuis les points de chargement CHA et CHB jusqu'au point de déchargement DEC. (Figure 3).

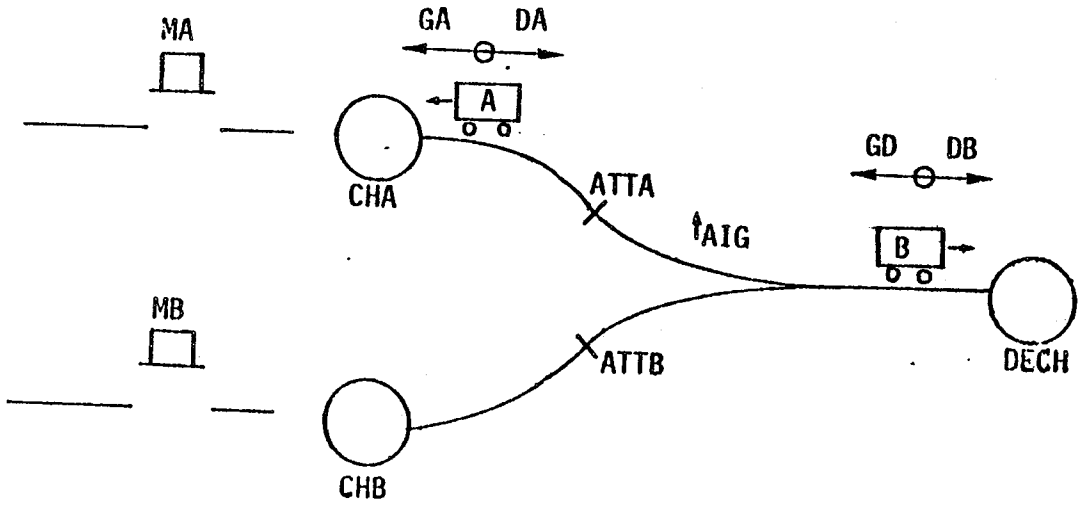


Fig. 3.

Les différents mouvements vers la gauche ou la droite sont effectués à l'aide de moteurs commandés par les sorties logiques GA, GB, DA et DB.

Si A est en CHA et si le bouton poussoir MA est appuyé, un cycle CHA-DECH-CHA est démarré avec attente éventuelle sur ATTA jusqu'à ce que la zone commune aux deux chariots soit libre. Le chariot B a un fonctionnement analogue.

Le chemin ATTA-DECH (resp. ATTB-DECH) est établi par positionnement d'un aiguillage, à l'aide de la commande logique AIG. ATTA (resp. ATTB) est une entrée logique positionnée à "1" si l'essieu avant de A (resp. B) est dans la zone ATTA-DECH (resp. ATTB-DECH). Un bouton poussoir R (consigne logique) permet de renvoyer un chariot de trouvant en DECH sur son point de chargement.

L'opérateur peut en appuyant sur le bouton poussoir APPEL (consigne logique) demander le poids total de matériel déchargé; le poids de matériel déchargé à chaque mouvement est mesuré à l'aide de l'entrée analogique POIDS.

Modélisation - Le procédé peut être décomposé en chariot A, chariot B, aiguillage, calcul du poids, auxquels sont associés les automates de la figure 4.

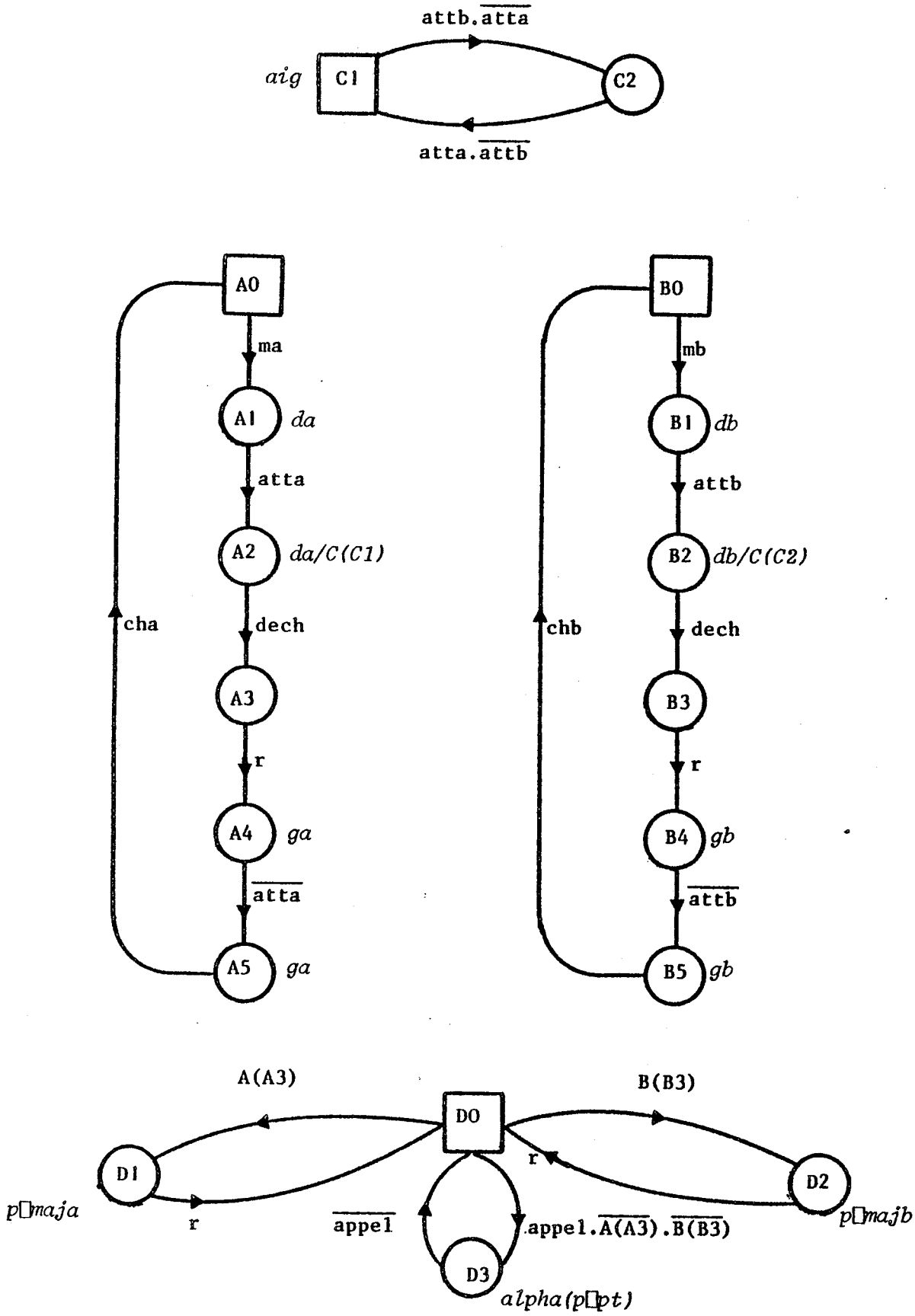

attb.atta
aig
C1
C2
atta.attb
A0
ma
A1
da
atta
A2
da/C(C1)
cha
dech
A3
r
A4
ga
atta
A5
ga
B0
mb
B1
db
attb
B2
db/C(C2)
chb
dech
B3
r
B4
gb
attb
B5
gb
A(A3)
B(B3)
D0
D1
D2
r
r
p□maja
appel
appel.A(A3).B(B3)
p□majb
D3
alpha(p□pt)

Fig. 4.

PROGRAMMATION

```
système      T R A N S P O R T

  entrées globales  mesure logique atta=1, attb=2, r=3, dech=4;

  machine A
  |
  |  interface
  |  | mesure logique cha=5; consigne logique ma=6; commande logique da=1,ga=2;
  |  fin interface
  |
  |  automate   état initial:A0;
  |  | [A0:ma→A1:atta→A2:dech→A3:r→A4:¬atta→A5:cha]
  |  fin automate
  |
  |  ordres A1:da; A2:da/C(C1); (A4,A5):ga fin ordres
  |
  fin A

  machine B
  |
  |  interface
  |  | mesure logique chb=7; consigne logique mb=8; commande logique db=3,gb=4;
  |  fin interface
  |
  |  automate   état initial:B0;
  |  | [B0:mb→B1:attb→B2:dech→B3:r→B4:¬attb→B5:chb]
  |  fin automate
  |
  |  ordres  B1:db; B2:db/C(C2); (B4,B5):gb  fin ordres
  |
  fin B
```

```
machine C
| interface commande logique aig=5 fin interface
| automate état initial:C1;
| | [C1:atta.attb→C2:atta.attb]
| fin automate
| ordres C1:aig fin ordres
fin C

machine D
|
| interface
| | mesure analogique poids = (10,(....));
| | affichage alphanumérique alpha = (20,(....));
| | consigne logique appel = 15
| fin interface
|
| module p
| | réel poids total a := 0, poids total b := 0;
| | action maja = (poids total a +:=
| | action majb = (poids total b +:= poids);
| | fonction réel pt = (réel r = poids total a + poids total b;
| |                     poids total a := poids total b := 0;  r)
| fin p;
|
| automate état initial:D0;
| | [[D0:A(A3)→D1:r],[D0:B(B3)→D2:r],[D0:appel→D3:appel]]
| fin automate
|
| ordres D1:p□maja; D2:p□majb; D3:alpha(p□pt) fin ordres
|
fin D

fin     T R A N S P O R T
```

V.3. - DEVERSEMENTS ET TIRAGES MULTIPLES D'UN LIQUIDE DANS UN RESERVOIR

Cette application concrétise un cas particulier du problème classique des "Lecteurs-Rédacteurs".

Résumé de l'analyse : Le procédé est schématisé par la figure 5 : p conduites C1, C2,...,C_p peuvent déverser du liquide dans un réservoir R (vannes tout ou rien W1,W2,...,W_p) par l'intermédiaire d'une conduite commune; q autres conduites C'1,C'2,...,C'q peuvent tirer le liquide contenu dans R (vannes tout ou rien V1,V2,...,Vq). Chaque opération de remplissage (processus Rédacteur)doit avoir lieu séparément et dure 3 minutes; les opérations de tirage (processus Lecteurs) peuvent avoir lieu simultanément. Enfin, les remplissages et les tirages sont commandés par un opérateur au moyen des boutons poussoirs correspondants A1,A2,...,Ap, B1,B2,...,Bq. Deux niveaux analogiques NRB et NRH permettent de contrôler les niveaux respectivement bas et haut du réservoir.

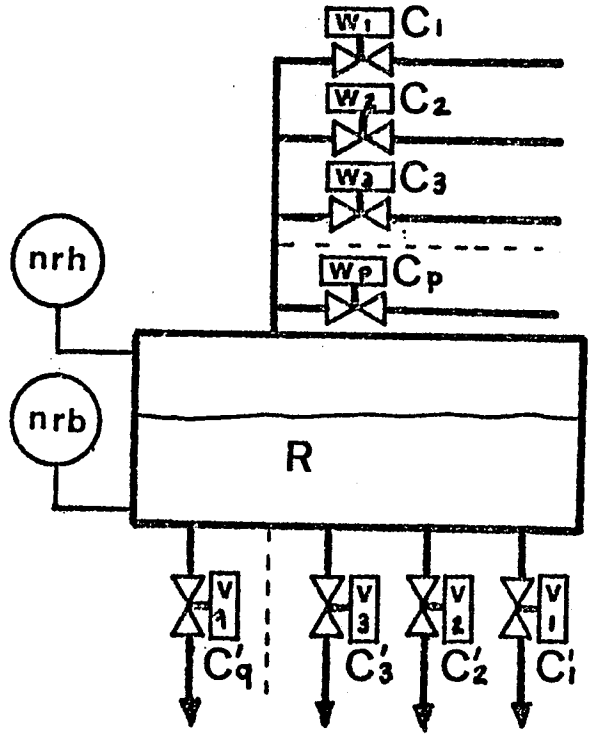
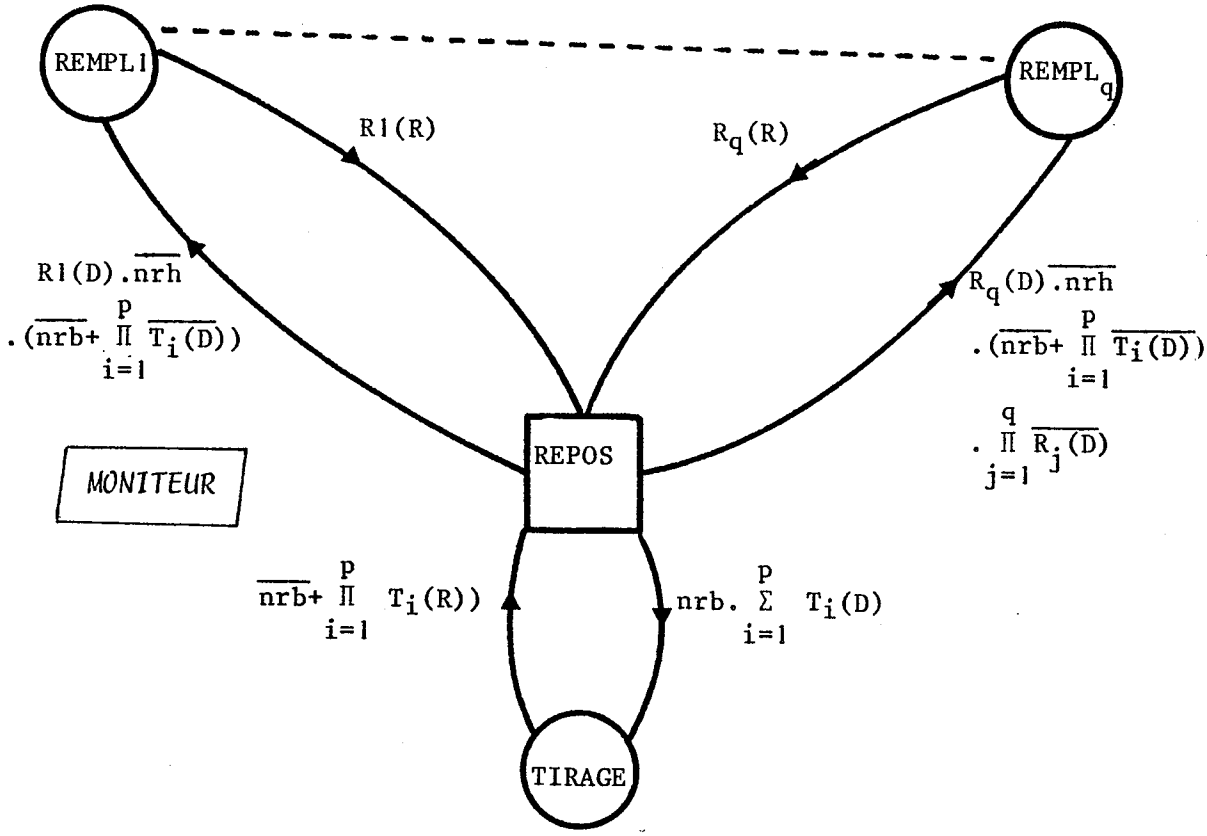
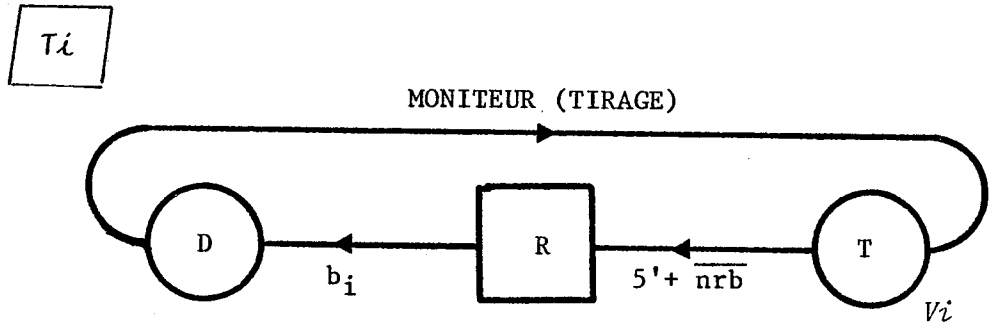


Fig. 5.

Modélisation - Le système de conduite peut être modélisé (figure 6) par exemple par une machine coordinatrice MONITEUR, p machines Lecteurs L1,L2,...,Lp, et q machines Rédacteurs R1,R2,...,Rq. La machine MONITEUR n'est pas nécessaire mais est introduite pour rendre indépendantes les machines Lecteurs et Rédacteurs.

Chaque machine Lecteur ou Rédacteur a trois états possibles : Repos, Demande d'accès et Travail. La priorité entre les Rédacteurs et celle qu correspond à leur numérotation. Le programme correspondant est donné pou p=2 et q=3. Il se compose de 6 machines interconnectées.


REMPL1
REMPLq
R1(R)
Rq(R)
R1(D).nrh
.(nrb+ Π Ti(D))
Rq(D).nrh
.(nrb+ Π Ti(D))
. Π Rj(D)
REPOS
MONITEUR
nrb+ Π Ti(R))
nrb. Σ Ti(D)
TIRAGE


Ti
MONITEUR (TIRAGE)
D
R
T
bi
5'+ nrb
Vi

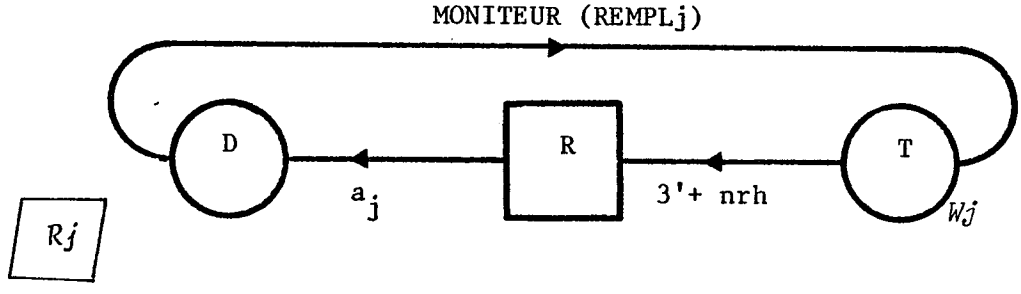

MONITEUR (REMPLj)
D
R
T
aj
3'+ nrh
Wj
Rj

Fig. 6.

PROGRAMMATION:

```
système     R E M P L I S S A G E S     T I R A G E S
|
|   entrées globales  mesure analogique nrb=(3,(....)), nrh=(4,(....));
|
|   modèle machine tireur(&b,&n,&v,&m)
|   |   interface  consigne logique &b=&n; sortie logique &v=&m  fin interface
|   |   automate   état initial:R;
|   |   |  [R:&b→D:MONITEUR(TIRAGE)→T:(5'+nrb)]
|   |   fin automate
|   |   ordres  T:&v  fin ordres
|   fin tireur;
|
|   modèle machine remplisseur(&a,&n,&w,&m,& REMPL)
|   |   ..................................  ..................................
|   fin remplisseur;
|
|   co création des machines tireur et remplisseur co
|   machine tireur T1(b1,1,v1,1),T2(b2,2,v2,2),
|           remplisseur R1(a1,5,w1,3,3,REMPL1),R2(a2,6,w2,4,REMPL2),R3(a3,7,w3,5,REMP
|
|   machine MONITEUR
|   |   automate   état initial:REPOS;
|   |   |  [[REPOS:nrb.(T1(D)+T2(D))→TIRAGE:(nrb+T1(R).T2(R))],
|   |   |   [REPOS:nrh.R1(D)→REMPL1:R1(R)],
|   |   |   [REPOS:nrh.R2(D)→REMPL2:R2(R)],
|   |   |   [REPOS:nrh.R3(D)→REMPL3:R3(R)]]
|   |   fin automate
|   fin MONITEUR
|
fin     R E M P L I S S A G E S     T I R A G E S
```

V.4. - UN MELANGEUR DE PRODUITS CHIMIQUES

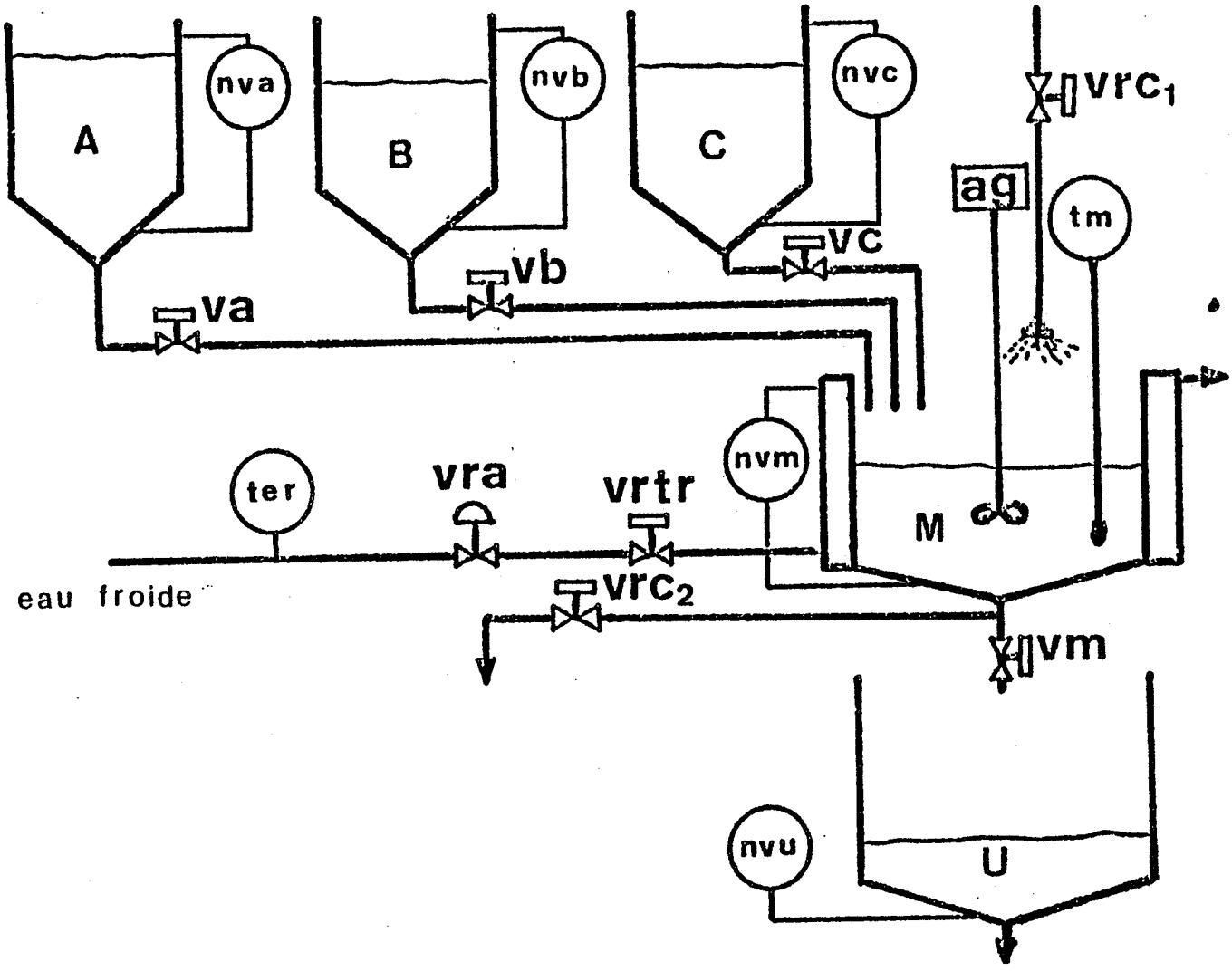
L'unité de traitement du système de conduite comprend deux sous-systèmes et un module de régulation numérique.

Le procédé schématisé par la figure 7 est destiné à fabriquer 2 produits à partir des réactifs contenus dans les bacs A et B, ou A et C. Les réactifs sélectionnés sont mélangés dans un bac M à l'aide d'un agitateur qui fonctionne seulement quand le liquide dans M recouvre les pales. Les mélanges, produisant des réactions exothermiques, sont amenés à une température de 200°F à l'aide d'un circuit de refroidissement, puis déversés dans un bac d'utilisation U. L'analyse du procédé met en évidence 3 écoulements principaux de liquide : celui du circuit de refroidissement, celui des bacs A et B, ou A et C. On peut reconnaître deux structures principales : écoulement AB et refroidissement, ou écoulement AC et refroidissement.

Modélisation - Le fonctionnement du système de conduite est décrit directement à partir des Graphes d'Etats de la figure 8.

Une machine SUPERVISEUR assure la mise en place d'une structure de commande selon le type de mélange demandé par l'opérateur (boutons poussoirs DAB et DAC). Supposons que l'opérateur appuie sur DAB : SUPERVISEUR passe alors dans l'état EAB et les machines MAB (Mélange AB) et RFAB° (Refroidissement) sont activées, tandis que l'agitateur AG est mis en position d'asservissement par rapport à la condition nvm > nvpales. La machine RFAB, positionnée sur l'état ACTIF quand MAB est sur DEVRSB, active une machine numérique PID , algorithme de régulation de la température dans le bac M (mesure TM, action sur VRA). MAB commence par vérifier que le VOLUME demandé et validé (bouton poussoir VAL) par l'opérateur est compatible avec le système physique (prédicat COMPATIBLE), puis effectue le

°*machine de modèle RF*



Interface procédé

entrées analogiques : - nva, nvb, nvc, nvm : niveaux des bacs A, B, C et M.
- tm : température M.

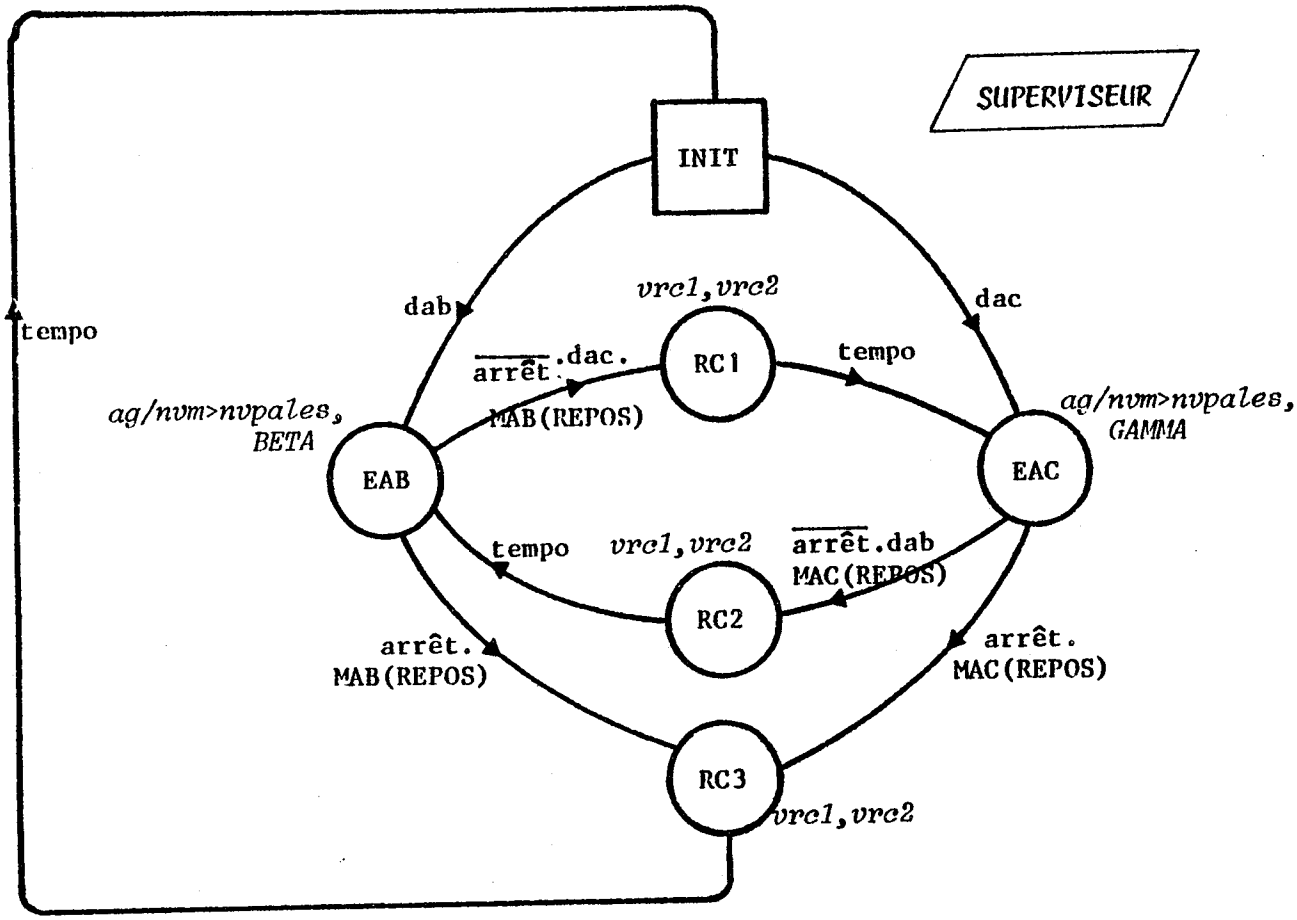
entrées logiques : - nvu : niveau bas du bac U.
- ter : témoin d'écoulement du circuit de refroidissement.

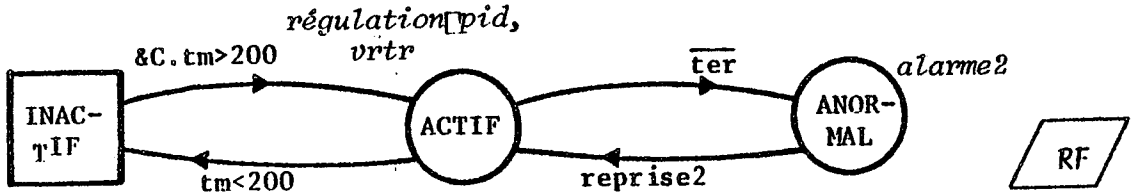
sortie analogique : - vra : vanne de régulation refroidissement.

sorties logiques : - ag : moteur de l'agitateur.
- vrc1, vrc2 : vannes rinçage.
- vrtr : vanne tout ou rien refroidissement.

Fig. 7 .

mélange. Si l'opérateur appuie sur le bouton DAC et que MAB est sur l'état ATTVOL, alors un rinçage de 3 minutes est effectué (état RC1, vannes VRC1 et VRC2) puis la commande correspondant à la structure AC est mise en place. La machine MAC n'est pas décrite car son fonctionnement est semblable à celui de MAB, bien que l'algorithme puisse être différent. Enfin, 2 reprises ont été prévues pour l'opérateur correspondant à 2 situations anormales (boutons poussoirs REPRISE 1 et REPRISE 2).


SUPERVISEUR
INIT
tempo
dab
dac
vrc1,vrc2
RC1
arrêt.dac.
MAB(REPOS)
tempo
ag/nvm>nvpales,
BETA
ag/nvm>nvpales,
GAMMA
EAB
EAC
tempo
vrc1,vrc2
arrêt.dab
MAC(REPOS)
RC2
arrêt.
MAB(REPOS)
arrêt.
MAC(REPOS)
RC3
vrc1,vrc2


régulation pid,
vrtr
&C.tm>200
ter
alarme2
INAC-
TIF
ACTIF
ANOR-
MAL
tm<200
reprise2
RF

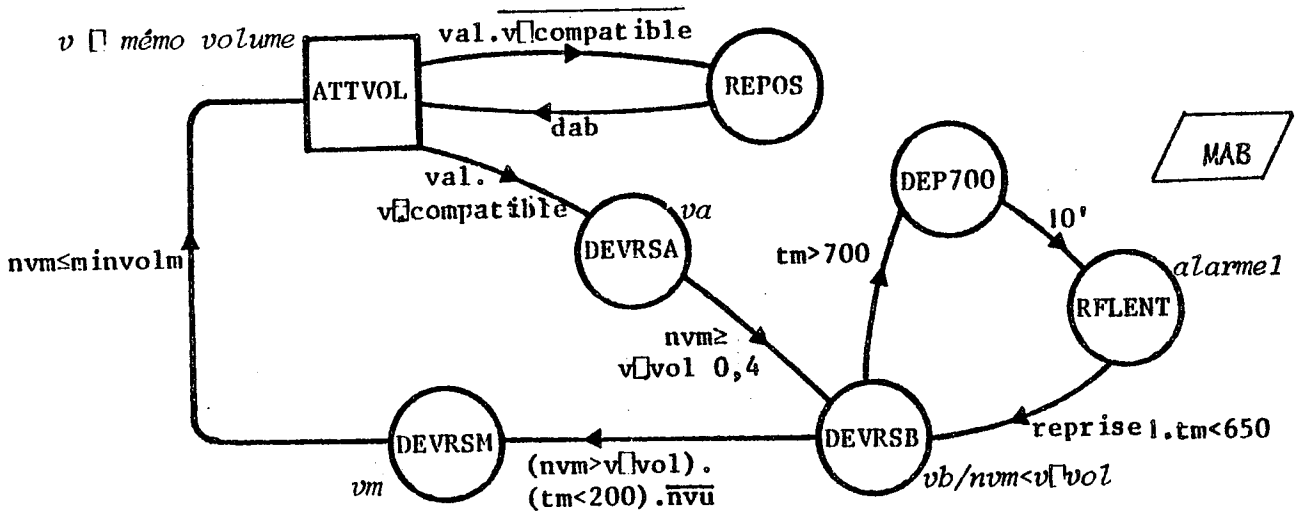

v mémo volume
val.v compatible
ATTVOL
REPOS
dab
MAB
val.
v compatible
va
DEVRSA
DEP700
10'
tm>700
alarme1
RFLENT
nvm≤minvolm
nvm≥
v vol 0,4
reprise1.tm<650
DEVRSB
DEVRSM
vm
(nvm>v vol).
(tm<200).nvu
vb/nvm<v vol

Fig. 8.

PROGRAMMATION:

```
système      M E L A N G E U R

  machine SUPERVISEUR

    unité de contrôle

      interface
        consigne logique dab=3, dac=4, arrêt=5;
        mesure analogique tm=(7,(....)), nvm=(12,(....));
        commande logique vrc1=1, vrc2=2, ag=3
      fin interface

      entier nvpales = 500;  délai tempo = 3';

      automate   état initial:INIT;
        [(RC3:tempo →INIT):(dab→EAB,dac→EAC)
                              [PRODUCTION]
                          (EAB:arrêt.MAB(REPOS),EAC:arrêt.MAC(REPOS))];
        PRODUCTION=[EAB:arrêt.dac.MAB(REPOS)→RC1:tempo→
                    EAC:arrêt.dab.MAC(REPOS)→RC2:tempo]
      fin automate

      ordres
        EAB:ag/nvm>nvpales,BETA;  EAC:ag/nvm>nvpales,GAMMA
        (RC1,RC2,RC3):vrc1,vrc2
      fin ordres

    fin contrôle
```

```
unité de traitement
    modèle machine rf(&C)
        unité de contrôle
            interface
                consigne logique reprise2=21; affichage logique alarme2=19;
                mesure logique ter=14; commande logique vrtr=5;
            fin interface
            automate   état initial:INACTIF;
                [ INACTIF:&C.tm>200→[ACTIF:ter→ANORMAL:reprise2]:tm<200]
            fin automate
            ordres  ACTIF:vrtr,régulation[]pid; ANORMAL:alarme2  fin ordres
        fin contrôle
        unité de traitement
            module régulation
                interface
                    mesure analogique tm=(7,(...)); commande analogique vra=(5,(....))
                fin interface
                action périodique(5') pid = (.........................)
            fin régulation
        fin traitement
    fin rf;

    système  BETA
        machine rf RFAB(MAB(DEVRSB));
        machine MAB
            interface
                consigne alphanumérique volume=(9,(....)),logique val=17,reprise1=20;
                affichage logique alarme1=18;
                mesure logique nvu=15, analogique nva=(10,(....)),nvb=(11,(....));
                commande logique va=2, vb=3, vm=4
            fin interface
```

```
réel minvolm=0.5;
module v co validation du volume de mélange demandé co
  réel volume demandé;
  prédicat compatible = bool:
    (co le mélange AB doit contenir 40% de A; 5000 représente, en
        litres, le volume maximum autorisé co
     volume<5000 et nva>volume×0.4 et nvb>volume×0.6);
  action mémo volume = (volume demandé:=volume);
  fonction réel = volume demandé;
  chemin mémo volume;(vol)* finch
fin v;

automate
  [[ATTVOL:val.v□compatible→REPOS:dab]:
   val.v□compatible→DEVRSA:nva>v□vol×0.4→DEVRSB[ECOULB]DEVRSB:
   nvm>v□vol.(tm<200).nvu→DEVRSM:nvm<minvolm];
  ECOULB=[[DEVRSB:tm>700→DEP700:tm<650]10'→
          RFLENT:reprise1.tm<650]
fin automate

ordres
  ATTVOL:v□mémo volume;  RFLENT:alarme1;
  DEVRSA:va; DEVRSB:vb/(nvm<v□vol); DEVRSM:vm
fin ordres

fin MAB

fin BETA;

système  GAMMA
  .....
fin  GAMMA

fin traitement

fin SUPERVISEUR

fin     M E L A N G E U R
```

L'utilisation du langage a été étudiée dans le cadre de systèmes matériels mono-processeur et multi-ordinateur pour une version simplifiée du langage décrite dans [Silva 77-1] et ayant les principales caractéristiques suivantes :

- Une décomposition en Partie Contrôle et Partie Opérative basée sur le critère suivant : la Partie Opérative contient les définitions des actions non logiques (mettant en oeuvre des sorties ou des entrée non logiques); l'interface de la Partie Contrôle ne comprend donc que des entrées et des sorties logiques. Ce critère de décomposition implique la nécessité de distinguer les actions exécutées dès que l'automate évolue vers un état (actions dites ponctuelles), les actions exécutées à chaque photographie des entrées tant que l'automate reste dans un état donné (actions dites permanentes), et les autres actions (dites de fond).
- Pas de sous-système.
- Les résultats des actions peuvent être seulement logiques et observée par l'intermédiaire de "clés", variables logiques standard modifiabl par toutes les actions définies par l'utilisateur.
- Pas de module au sens du chapitre V.
- Une description non structurée de l'automate : une phase est assurée à chaque état, décrivant les évolutions possibles (il faut donc vérifier la connexité forte du graphe résultant).

Une réalisation prototype a été effectuée sur un micro-processeur INTEL 8080, les programmes exécutés étant produits sous forme de "macro-instructions" par un traducteur automatique écrit en Algol W sur IBM 360 [Silva 77-1].

Une architecture multi-ordinateurs a été suggérée dans [Pleyber 77-1] .

VI.1. - MISE EN OEUVRE SUR MONO-PROCESSEUR

VI.1.1. - Les programmes obtenus par traduction automatique

Ils sont produits dans un langage algorithmique dont les primitives sont indépendantes d'une structure particulière de machine. Ces primitives sont cependant suffisamment de bas niveau pour que les programmes puissent être compilés sur une machine particulière en ne mettant en oeuvre que des outils du niveau macro-expansion, elles pourraient également constituer le jeu d'instructions d'une machine spécialement construite; la portabilité des programmes ainsi obtenus peut facilement être mise en oeuvre grâce à l'adaptabilité du traducteur. Les macro-expansions à envisager ne portant que sur les données, le traducteur peut en particulier facilement être transformé de façon à réaliser les macro-expansions ou même à produire des programmes binaires.

Le fonctionnement de l'Automate Programmable est d'abord décrit, puis certains des algorithmes utilisés.

VI.1.2. - Fonctionnement de l'Automate Programmable

Les programmes engendrés sont tels que le fonctionnement de l'Automate Programmable est cyclique et est celui décrit dans la figure 1, où un programme source spécifiant n automates est considéré.

Un cycle commence par une "photographie" des entrées et des états et se termine avec la photographie suivante. La durée, variable dans le cas général, doit cependant être inférieure à une valeur maximum prédéfinie.

La recherche d'une évolution possible d'un automate à partir de son état courant se fait en calculant successivement les conditions d'évolutions, dans l'ordre où elles se présentent dans le programme source, jusqu'à ce que l'une d'entre elles donne la valeur VRAI ou qu'il n'y en ait plus.

L'initialisation des actions de fond consiste en la mise sur une file d'attente des routines correspondantes avec les valeurs de paramètres.

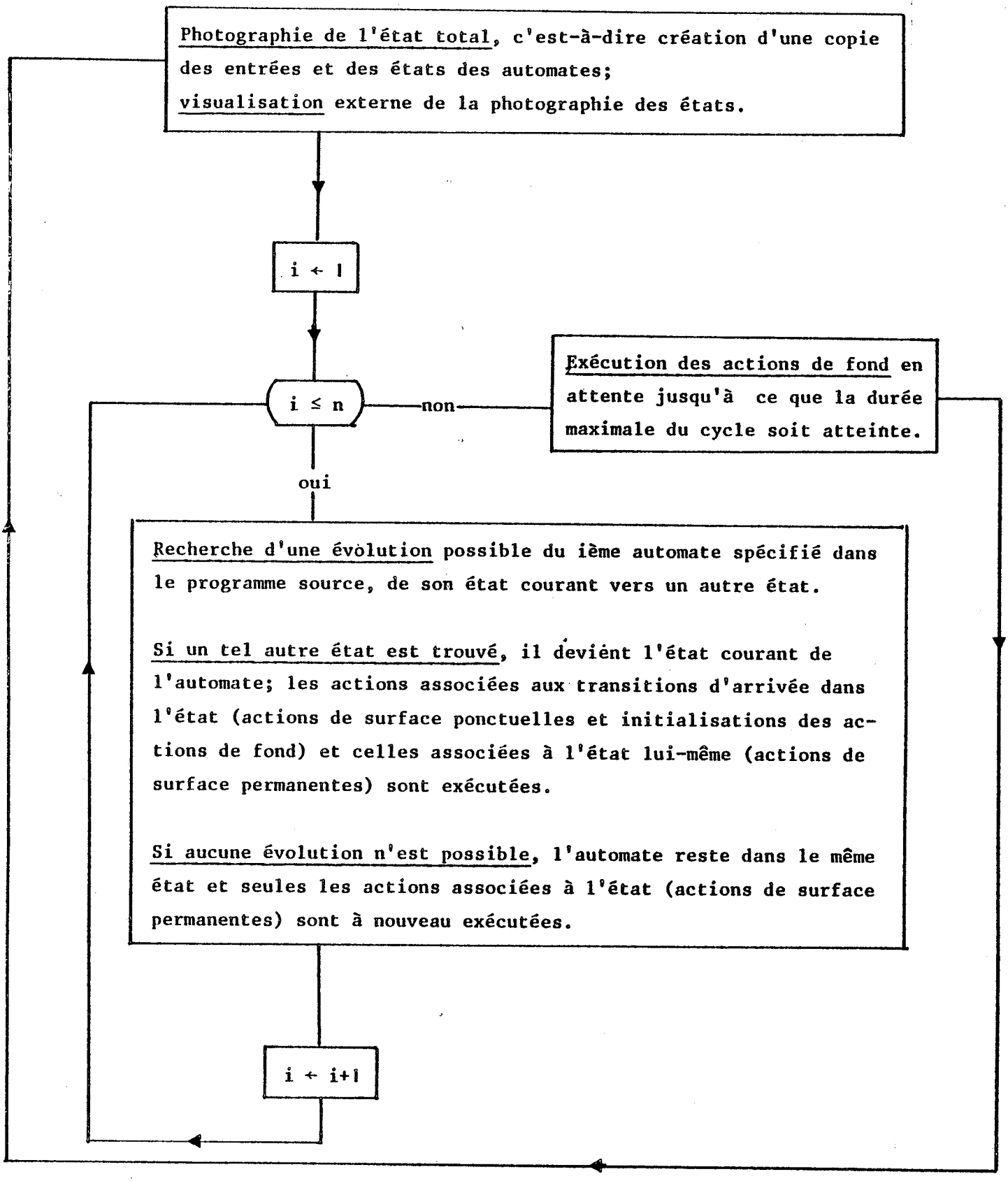

Photographie de l'état total, c'est-à-dire création d'une copie des entrées et des états des automates;
visualisation externe de la photographie des états.
i ← 1
i ≤ n
non
Exécution des actions de fond en attente jusqu'à ce que la durée maximale du cycle soit atteinte.
oui
Recherche d'une évolution possible du ième automate spécifié dans le programme source, de son état courant vers un autre état.
Si un tel autre état est trouvé, il devient l'état courant de l'automate; les actions associées aux transitions d'arrivée dans l'état (actions de surface ponctuelles et initialisations des actions de fond) et celles associées à l'état lui-même (actions de surface permanentes) sont exécutées.
Si aucune évolution n'est possible, l'automate reste dans le même état et seules les actions associées à l'état (actions de surface permanentes) sont à nouveau exécutées.
i ← i+1

Fig. 1.

La figure 2 schématise l'organisation d'un programme, l'exemple considéré étant proche de celui donné en V.2. et détaillé dans [Silva 77-2].

Un programme est la concaténation de n séquences d'instructions engendrées pour les n automates spécifiés dans le programme source. La ième séquence est elle-même la concaténation de m_i séquences engendrées pour les m_i états possibles du ième automate. L'ordre dans lequel sont concaténées les séquences n'a aucune importance.

Une séquence associée à un état possible d'un automate comprend le calcul des évolutions et des actions spécifiées pour cet état dans le programme source (figure 3); elle est structurée de façon à ce que les actions ponctuelles soient exécutées et les actions de fond initialisées seulement au cours du cycle (figure 1) où l'automate évolue vers l'état (figures 3 et 4) et que les actions permanentes soient exécutées à chaque cycle (figure 1) tant que l'automate reste dans l'état.

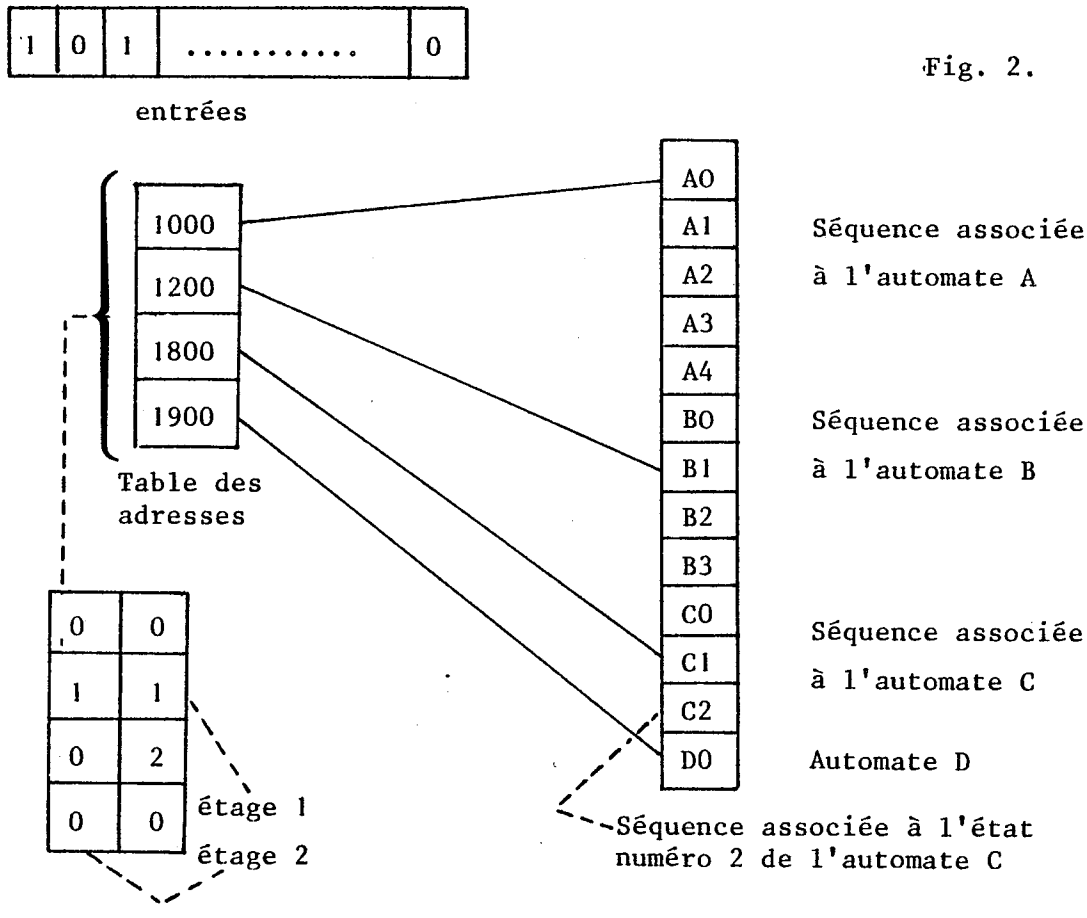
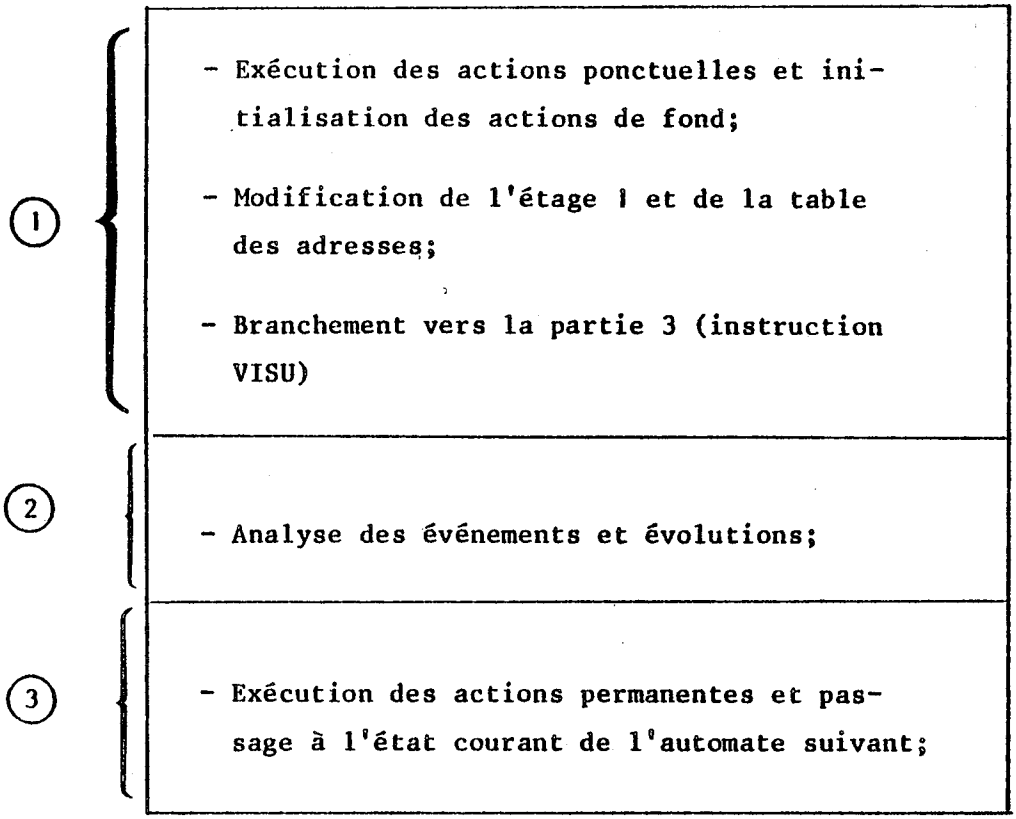


Fig. 2.

Structure générale d'un programme, ici en cours de cycle : la dernière photographie des états était A0,B1,C1,D0 (étage 2 et table des adresses) et une évolution a été effectuée pour l'automate C vers l'état numéro 2 (étage

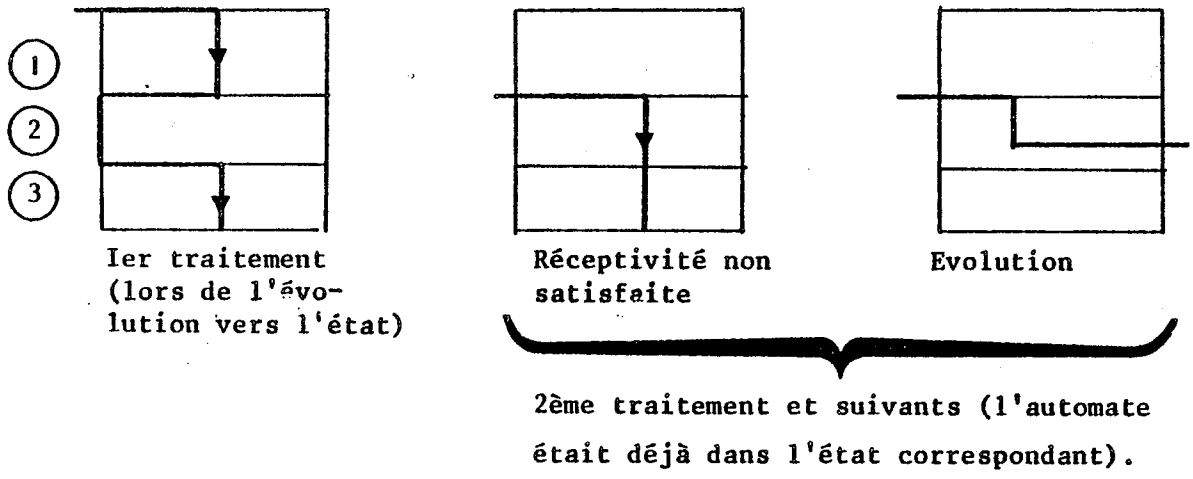
Chapitre VI

IMPLEMENTATION



Séquence associée à un état

Fig. 3.



Traitement d'un état

Fig. 4.

La table de visualisation comprend 2 étages. L'étage 1 contient à tout moment les numéros des états des automates à ce moment et est donc modifié au cours du cycle. L'étage 2 est mis à jour au début de chaque cycle par recopie de l'étage 1 (photographie des états) et est visualisé extérieurement (opérateur); il n'est pas modifié en cours de cycle.

La table des adresses des états courants contient les adresses des parties "analyse des événements et évolutions" des états courants des différents automates; elle est mise à jour lors des évolutions des automates.

Une instruction CHAU (CHangement d'AUtomate), placée à la fin de chaque séquence associée à un état effectue un branchement vers la séquence de l'état courant de l'automate suivant par indirection sur la table des adresses

Une instruction EVOL (EVOLution), exécutée lors de l'évolution d'un automate d'un état vers un autre, effectue un branchement vers la partie 1 de la séquence associée au nouvel état. Dans cette partie, est en particulier exécutée une instruction VISU qui met à jour l'étage 1 de la table de visualisation et la table des adresses et effectue un branchement vers la partie 3 de la séquence : ainsi, deux évolutions ne peuvent avoir lieu pour un automate au cours d'un même cycle.

Les photographies des entrées et des états (recopie de l'étage 1 vers l'étage 2) sont effectuées après l'instruction CHAU exécutée pour le dernier automate spécifié dans le programme source (branchement vers le microsystème).

VI.1.3. - L'implémentation sur micro-processeur 8080

Nous donnons dans ce paragraphe quelques résultats relatifs à l'encombrement mémoire et au temps d'exécution des programmes engendrés par le traducteur.

On peut dire que l'encombrement mémoire croît linéairement avec la complexité du problème, le coefficient de proportionnalité dépendant de la

machine support, du logiciel d'exécution et de la réalisation matérielle des sorties.

a) La machine support : l'INTEL 8080 ne possède pas d'adressage relati ni indirect, les indicateurs de position ne sont pas positionnés par le chargement de l'accumulateur, ce qui implique un encombrement mémoire plus grand que celui auquel on pourrait s'attendre.

b) Le logiciel d'exécution est un "micro-système" réalisant certaines fonctions telles que le dialogue opérateur et contrôlant les programmes obtenus par macro-expansion des instructions engendrées par le traducteur. Un système interprétatif pourrait être plus intéressant à partir d'un certain niveau de complexité, bien que moins rapide.

c) Les sorties peuvent être matériellement réalisées :

- soit par un double étage de mémorisation avec remise à zéro du premier lors du recopiage dans le deuxième (envoyé sur le procédé) à la fin de chaque cycle de traitement;
- soit par un seul étage avec "mise à valeur" pour les actions conditionnelles à chaque cycle de traitement. Les actions inconditionnelle: sont mises à un et à zéro respectivement à l'activation et à la désactivation de l'état.

La première solution nécessite moins de mémoire mais le temps d'exécution est plus élevé si la modélisation met en jeu de nombreuses actions exécutées seulement à l'instant d'arrivée dans l'état. Nous avons choisi la deuxième solution.

Avec un dialogue opérateur élémentaire, le micro-système occupe enviror 200 octets et le programme utilisateur un nombre d'octets approximativement proportionnel à la complexité du problème (3/4K octet en ce qui concerne le programme schématisé en VI.2., y compris le micro-système).

Pour ce programme, un cycle de traitement a une durée de l'ordre de 150 à 400 µs.

VI.1.4. - Traitement des expressions logiques

On peut envisager les quatre méthodes suivantes :

1 - utiliser des instructions logiques existant sur les calculateurs universels pour effectuer toutes les opérations;

2 - utiliser des instructions logiques du type de celles existant sur certains automates programmables commercialisés [SIL-2] : le principe est de chercher à déterminer si l'expression logique est vraie au fur et à mesure de son évaluation;

3 - utiliser des masques sur l'ensemble des variables;

4 - déterminer un graphe de décision binaire représentant la fonction pour décrire son évaluation sous forme d'une suite de branchements conditionnels.

Du point de vue temps d'exécution, la première méthode est moins bonne que les suivantes car elle nécessite l'exécution de toutes les opérations; d'autre part, la possibilité d'avoir des termes élémentaires de nature différente (variables et appels de prédicats) conduit à écarter l'utilisation de masques bien que dans certains cas (monômes, sommes de monômes monovariables) les performances soient meilleures; enfin, l'utilisation des arbres de décision binaire conduit à des programmes plus rapides que la deuxième méthode, en effet :

α) Si la fonction contient n variables, dans le premier cas il y a au plus n appels et exécutions d'instructions tandis que dans le deuxième toutes les instructions doivent être appelées parmi lesquelles un nombre variable sera exécuté.

β) Si la machine support est un calculateur universel, l'interprétation est beaucoup plus longue dans le cas de la deuxième méthode.

Du point de vue encombrement mémoire, l'utilisation des arbres de décision binaire nécessite des instructions plus longues (déplacement) que la deuxième méthode, mais le nombre en est toujours plus petit.

Le choix résultant est celui des arbres de décision binaire. Les instructions associées sont donc des branchements conditionnels.

Une expression logique est un polynôme, c'est-à-dire une somme de monômes. Chaque monôme est un produit de secondaires. Chaque secondaire est soit un polynôme, soit un atome.

Soient P un polynôme et A(P) l'adresse de la séquence de branchements conditionnels engendrés pour P.

Soient A(M) (respectivement A(S)) définie de façon analogue pour tout monôme M (respectivement secondaire S).

Un polynôme est un triplet P= {LM,C,Av,Af}

où : - LM = (M_1,M_2,...,Mp) est la suite des monômes constituant le polynome
- C est une variable logique qui prend la valeur vrai si P est sous forme complémentée, et faux s'il est sous forme directe;
- Av(P) est l'adresse de la prochaine instruction à exécuter si la valeur de P est vrai;
- Af(P) est l'adresse de la prochaine instruction à exécuter si la valeur de P est faux.

Un monôme est un triplet M= (LS,C,Av,Af)

où : - LS = (S_1,S_2,...,Sq) est la suite des secondaires constituant le monôme;
- C est une variable logique définie de façon analogue au cas du polynôme;
- Av(M) et Af(M) sont définis de façon analogue au cas du polynôme.

Les relations suivantes sont vérifiées :

1°) pour un polynôme P :

Cas où C est faux

α) Af(Mi) = A(Mi+1) i=1,2,...,p-1
Av(Mn) = Af(P)

β) Av(Mi) = Av(P) i=1,2,...,p

Cas où C est vrai

α) Af(Mi) = Af(Mi+1) i=1,2,...,p-1
Af(Mn) = Av(P)

β) Av(Mi) = Af(P)

2°) pour un monôme M :

Cas où C est faux

α) Af(Si) = Af(M) i=1,2,...,q

β) Av(Sj) = A(Sj+1) i=1,2,...,q-1
Av(Sq) = Av(M)

Cas où C est vrai

α) Af(Sj) = Av(M)

β) Av(Sj) = A(Sj+1) j=1,2,...,q-1
Av(Sq) = Af(M)

La génération des branchements conditionnels est basée sur la détermination pour chaque élément (polynôme, monôme, secondaire) des adresses de branchements dans les cas où la valeur de cet élément est respectivement vrai ou faux.

Pour chaque expression logique du programme, une structure interne est créée à partir de listes de monômes et de secondaires. Une telle structure est montrée dans la figure VI.5 pour l'expression (ab'+(c.(d+e)'.f.(g+h).k)') où a,b, etc. représentent des noms d'entrées logiques.

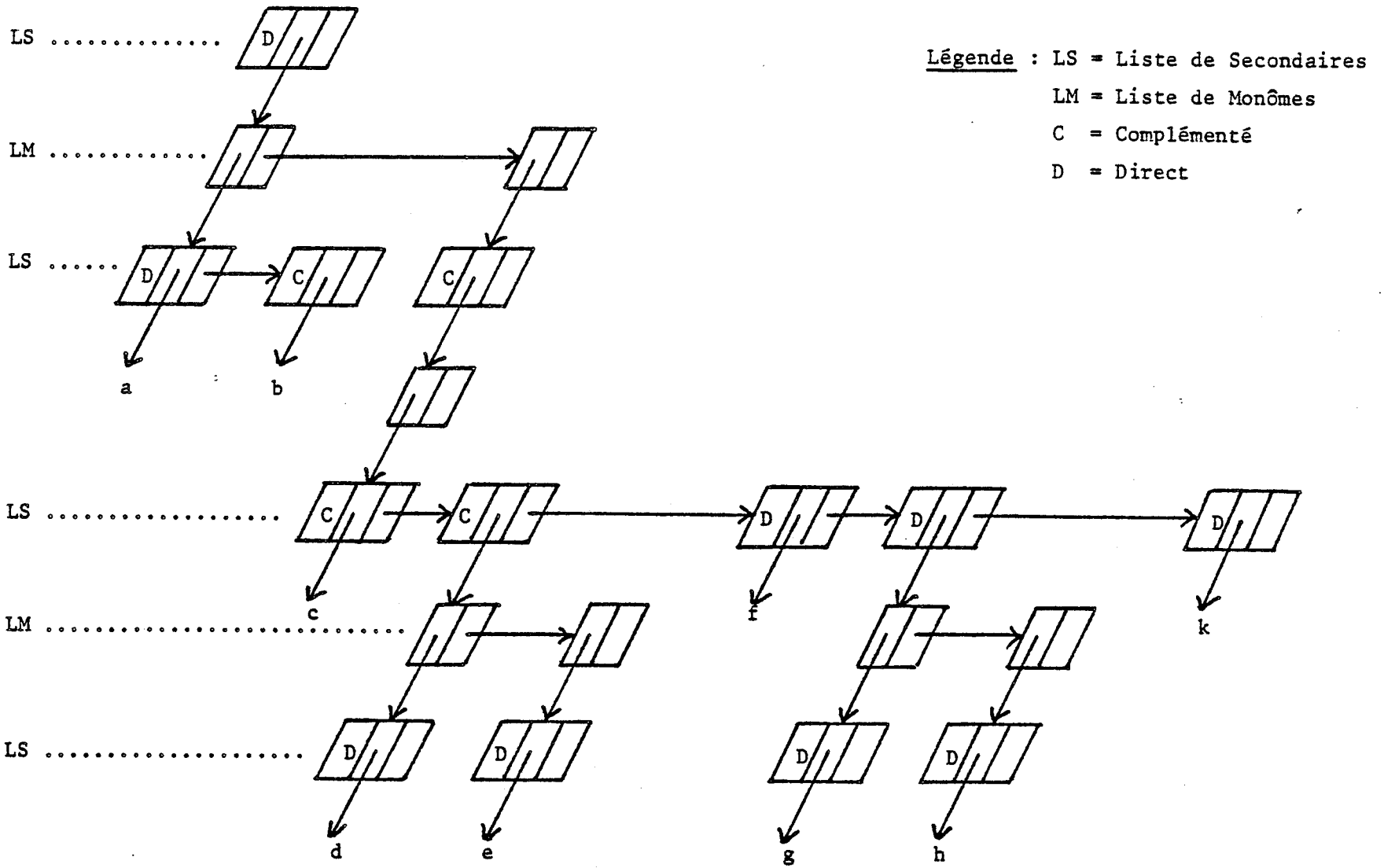


Fig. 5. Structure interne associée à l'expression a.b'+(c.(d+e)'.f.(g+h).k)'

VI.1.5. - Traitements des sorties logiques

La spécification "abrégée" des positionnements des sorties logiques dans les états des automates (IV.3.) conduit au calcul, pour chaque état, des ensembles minimum de sorties à positionner à "1" et à "0".

Appelons état précédent d'un état donné, tout état à partir duquel l'automate peut évoluer par une seule transition vers l'état donné. Les algorithmes de calcul des ensembles sont alors les suivants :

1°) Calcul de l'ensemble des sorties à positionner à "0" :
- Faire l'union des sorties (conditionnelles ou non) des états précédents.
- Supprimer dans l'ensemble résultant toutes les sorties (conditionnelles ou non) de l'état donné.

2°) Calcul de l'ensemble des sorties à positionner à "1" :
- Considérer l'ensemble des sorties inconditionnelles de l'état donné.
- Supprimer les sorties inconditionnelles communes à tous les états précédents.

VI.2. - MISE EN OEUVRE SUR MULTI-ORDINATEUR

La dynamique du procédé détermine la durée maximale d'un cycle de traitement (intervalle de temps entre 2 photographies des entrées) et le nombre d'automates pouvant fonctionner parallèlement. C'est pourquoi on peut être amené à envisager l'utilisation de plusieurs processeurs. Pour vérifier qu'une solution telle que nous la suggérons est valable, il faut évidemment tenir compte de la surcharge du système ainsi mis en place.

Deux classes de systèmes peuvent être considérées [Russo 77] :

- Les systèmes multiprocesseurs, qui opèrent sur une mémoire ou un dispositif d'entrée unique (chaque processeur traite des parties d'un même programme);

- Les systèmes multiordinateurs, qui opèrent sur plusieurs flots d'entrées et n'ont pas de système opératoire intégré (la communication entre processeurs est faite par données).

Notre préférence a été à une architecture multiordinateurs parce que les systèmes de conduite de procédé en temps réel peuvent en général être répartis efficacement en modules presque indépendants. Une décentralisation fonctionnelle est alors possible, qui accroît la sûreté (quand une panne apparaît sur un module, le reste du système garde un certain niveau d'activité).

Les autres avantages de ce choix sont :

- de meilleures performances en raison d'une surcharge réduite du système opératoire, d'un nombre plus petit d'interférences entre les modules et de la possibilité de choisir le processeur le mieux adapté pour chaque fonction;
- un coût moins élevé pouvant être obtenu par l'utilisation de processeurs spécialisés, de microsystèmes (sur une pastille) et une décentralisation géographique (ordinateur réparti).

Le principal inconvénient d'un multiordinateur est une reconfigurabilit médiocre en cas de panne sur un module. Cependant, les structures sont en général doublées dans une architecture réelle, ce qui rend cet inconvénient moins important.

La figure 6 décrit l'architecture à un niveau fonctionnel. Deux classes de modules peuvent être clairement distinguées: les modules de contrôle (automates) et les modules de traitement (ici au sens de modules qui ne sont pas des automates). Chaque module est spécialisé fonctionnellement par un programme privé.

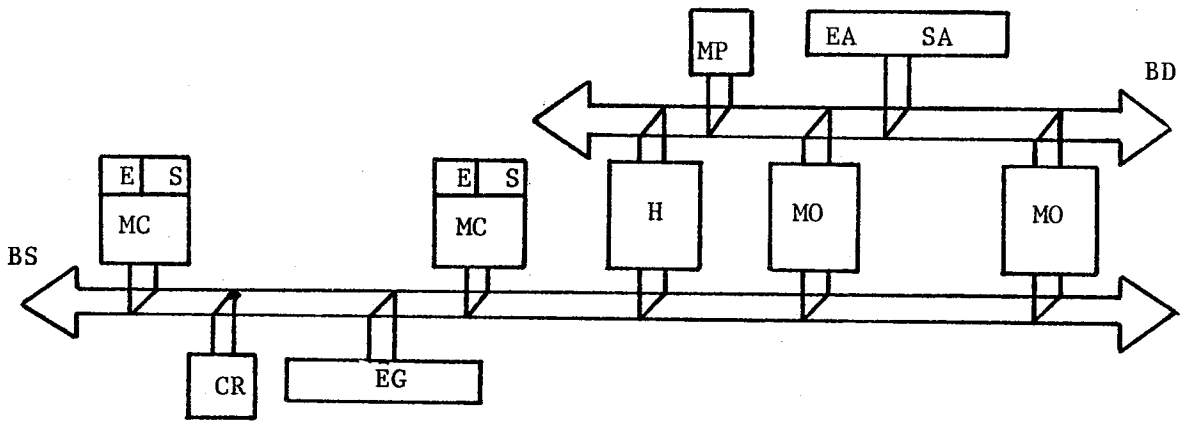
Les modules sont interconnectés par deux bus :

- BS, Bus de Synchronisation qui relie :
  - les modules de contrôle par l'intermédiaire de la table de visualisation,

. chaque module de contrôle avec les entrées logiques globales,
. les modules de contrôle avec les modules de traitement (ordres et valeurs de prédicats);

- BD, Bus de Données qui relie :
  . les modules de traitements,
  . les modules de traitement avec les entrées-sorties numériques et analogiques.

Pour permettre une certaine reconfigurabilité, ces deux bus peuvent être technologiquement confondus.



MC : Module de Contrôle; MO : Module Opératif;
H : Horloges temps réel;
E : Entrées privées; S : Sorties privées; EG : Entrées Globales;
EA : Entrées Analogiques; SA : Sorties Analogiques;
MP : Mémoire Partagée;
CR : Comptes Rendus (clés) de la Partie Opérative à la Partie Contrôle

Fig. 6.

Chapitre VII

CONCLUSION

VII.1. - LA STRUCTURE DE CONTROLE DES ALGORITHMES

Le langage de programmation proposé permet la description des structures de contrôle algorithmiques d'un système de conduite de façon indépendante d'une implémentation particulière. Ceci a été possible parce que le problème n'a pas été abordé sous l'aspect général de la programmation en Temps Réel mais sur celui des caractéristiques propres au domaine d'application considéré : la conduite des procédés industriels

L'utilisation de l'ordinateur a amené à considérer largement le critère fonctionnel résultant des scrutations périodiques des entrées pour définir la structure algorithmique du système de conduite. La description dépend donc à ce niveau de la technologie utilisée, de même que celles issues des méthodes tabulaires étaient liées à la technologie des relais et celles des phénomènes continus dans le domaine fréquentiel étaient liées à l'utilisation de chaînes analogiques. Cette démarche a été préférée car elle correspond à la préoccupation de fournir un outil algorithmique permettant une description synthétique des programmes plutôt qu'une description transparente : le modèle des Graphes d'Etats aurait pu être interprété par exemple de façon asynchrone, ce qui aurait été plus proche des habitudes des automaticiens, mais les aurait éloignés d'une réelle compréhension du déroulement des programmes et compliqué inutilement la validation. D'autre part, il a été souligné que la décentralisation du système de conduite était alors davantage guidée, les différents blocs du programme pouvant être associés à des horloges internes distinctes.

VII.2. - DEVELOPPEMENTS POSSIBLES

Un certain nombre de caractéristiques du langage est repris dans les paragraphes suivants, afin de suggérer des thèmes de recherche possibles.

1°) La description des automates de conduite

Le choix des Graphes d'Etats a été justifié dans les chapitres I et II pour le domaine des applications envisagées. Cependant, celles-ci peuvent être très diverses et dans certains cas requérir des modèles mieux adaptés à choisir en fonction de critères comme l'importance du parallélisme, la nécessité d'une décomposition fonctionnelle ou géographique, ou l'exigence d'une représentation graphique. Aussi le choix qui a été fait ne doit-il pas être considéré comme la caractéristique fondamentale du langage.

Le choix d'une description non graphique peut être intéressante pour obtenir une rédaction plus synthétique et une meilleure vérification statique des programmes, mais le problème peut alors se poser de la transposition vers un modèle graphique, qui garde un sens physique; de telles transpositions ont été définies pour la description de synchronisations à partir d'expressions de chemin [Lauer 75] et de commandes gardées [Haase 77]

Une voie peu explorée est celle de l'utilisation de grammaires formelles pour décrire des enchaînements d'événements et d'actions. L'idée est de reconnaître des enchaînements d'événements qui ne peuvent l'être par des automates d'états finis, c'est-à-dire qui ne sont pas des expressions régulières sur le vocabulaire des événements (un tel vocabulaire comprend 2^n éléments s'il y a n événements élémentaires). Le mécanisme de l'enchaînement des actions peut être étudié par analogie avec la

structure d'un analyseur syntaxique dans un compilateur où des fonctions sémantiques (actions) sont éxécutées au fur et à mesure que les symboles lexicographiques (événements) sont reconnus. Pour le domaine des systèmes de conduite, il a été souligné qu'il était souhaitable que la description puisse se faire sous forme graphique. D'autre part, les contraintes temps réel nécessitent une exécution immédiate des actions sur occurrence des événements. Une solution respectant ces contraintes peut être recherchée à partir des systèmes de diagrammes de transitions définis par Conway [Conway 63] et formalisés par Lomet [Lomet 73] : la structure de contrôle se présente comme un ensemble d'automates d'états finis liés par un mécanisme d'empilement des appels. Il serait donc possible de visualiser ces automates, avec ou sans la pile des appels, suivant le niveau d'information qui doit être fournie. Lomet a montré que ces systèmes d'automates reconnaissent, sans retour arrière sur la chaîne d'entrée (événements), tous les langages déterministes à contexte libre : le modèle est donc général et permet d'exécuter de façon immédiate les actions liées aux symboles de la grammaire. L'intérêt d'un tel modèle serait une description naturelle des hiérarchies d'automates et une définition plus synthétique du contrôle en particulier en ce qui concerne les priorités relatives des événements.

2°) La "structuration" du langage de spécification des automates

L'essai d'une spécification globale des chemins de contrôle, et en particulier des cycles, est intéressant des points de vue de la vérification statique et de la lisibilité des programmes. Cependant, le but principal a été la recherche d'une prise en compte de spécifications en amont dans la conception, c'est-à-dire le guidage du concepteur dans la synthèse des automates. Il pourrait être d'envisager le même travail pour d'autres modèles graphiques, en particulier pour les Réseaux de Petri, comme cela a été suggéré dans [Valette 76].

3°) Les communications entre modules

Les machines ne peuvent communiquer entre elles que leurs états respectifs. Ceci n'est pas suffisant pour des systèmes complexes pour lesquels un mécanisme d'échanges d'informations devrait être défini. La recherche d'une structure pouvant être décentralisée amène à préférer un échange par transmissions de messages [Ladet 77] plutôt que par variables communes.

4°) La modification en ligne des systèmes de conduite

Elle n'a pas été prévue dans la description. Elle pourrait être effectuée en définissant des primitives d'inclusion, d'exclusion et de remplacement d'automates ou de modules. Elle pose des problèmes difficiles de vérification fonctionnelle statique et de risque d'erreurs de la part de l'opérateur. Aussi me semble-t-il qu'il est délicat, voire dangereux, de permettre de telles facilités, en l'absence d'une description formelle globale de haut niveau de fonctionnement du procédé. Si une telle description était possible, alors la modification en ligne pourrait consister à modifier d'abord la description du fonctionnement du procédé, ensuite à vérifier la cohérence du nouveau fonctionnement obtenu, puis à déduire la nouvelle structure du système de conduite.

VII.3. - LA VALIDATION FONCTIONNELLE D'UN AUTOMATISME

Bien que la description proposée pour un système de conduite soit indépendante de l'implémentation, elle demeure à un niveau algorithmique plutôt que fonctionnel. En effet le problème fondamental de la description du procédé, et de la vérification de la cohérence entre celle-ci et le système de conduite, n'est pas résolu. Une recherche importante serait donc la définition d'un langage de haut niveau pour décrire le

fonctionnement d'un procédé, prenant en compte globalement tous les aspects ("logique" et "régulation") et l'élaboration de méthodes pour en déduire le système de conduite; un critère à utiliser dans ces méthodes pourrait être le niveau d'automatisation souhaité : pour un même fonctionnement du procédé, comment spécifier la part des décisions qui doivent être confiées à l'ordinateur ou à l'opérateur, et comment passer d'un niveau à un autre. Il est clair qu'alors la vérification statique des programmes pourrait être plus importante et en particulier la validation structurelle des automates facilitée. Des modèles liés aux Réseaux de Petri ont été étudiés [Moalla 76] pour la description des systèmes; d'autres outils déjà cités, développés dans le domaine des systèmes opératoires, comme les expressions de chemins [Lauer 77] et les compteurs de synchronisation [Robert 77]peuvent également constituer des éléments intéressants.

A N N E X E

DEFINITION FORMELLE DU LANGAGE

La syntaxe du langage est décrite par les règles hors contexte ci-des où deux sortes d'identificateurs sont utilisées : les notions, en lettre majuscules, et les symboles terminaux, en lettres minuscules. L'axiome de la grammaire est SYSTEME.

Soient M1,M2,...,Mn des notions de la grammaire et M une notion définie par la règle :

M : M1,M2,...,Mn.

Les notations suivantes sont utilisées :

- Σ(M1,M2,...,Mn) désigne une suite non vide de productions terminales de M;
- Π(M1,M2,...,Mn) désigne une suite non vide de productions terminales de M séparées par des virgules;
- Λ(M1,M2,...,Mn) désigne une production terminale de M ou une suite non vide, entre parenthèses, de productions terminales de M.

Enfin, toute notion commençant par NOM a comme production terminale idf (identificateur).

a) *Structure générale des programmes*

a1) SYSTEME : système, NOM DE SYSTEME, ENTREES GLOBALES, MODELES DE MACHINES {g1}, Σ(MACHINE {b1}), fin, NOM DE SYSTE

a2) ENTREES GLOBALES : vide;
entrées globales Σ(DESIGNATIONS D'ENTREES {c3}).

b) *Structure d'une machine*

b1) MACHINE : MACHINE COMBINATOIRE;
MACHINE SEQUENTIELLE;
ACTUALISATION DE MACHINE.

b2) MACHINE COMBINATOIRE : machine, NOM DE MACHINE, DEFINITIONS, COMBINATOIRE {f1}, fin, NOM DE MACHINE.

b3) MACHINE SEQUENTIELLE : machine, NOM DE MACHINE, CONTROLE, fin, NOM DE MACHINE;
machine, NOM DE MACHINE, UNITE DE CONTROLE, UNITE DE TRAITEMENT, fin, NOM DE MACHINE.

b4) DEFINITIONS : INTERFACE {C1}, DECLARATIONS, MODELES DE CALCUL {g3}, CALCULS {h1}.

b5) UNITE DE CONTROLE : unité de contrôle, CONTROLE, fin contrôle.

b6) CONTROLE : DEFINITIONS, AUTOMATE {e1}, ORDRES {f2}.

b7) UNITE DE TRAITEMENT : unité de traitement, MODELES D'ELEMENTS {g5}, Σ(ELEMENT), fin traitement.

b8) ELEMENT : SYSTEME {a1};
MACHINE;
ACTUALISATION DE MACHINE;
MODULE DE TRAITEMENT {h14};
PROCEDURE PERIODIQUE {h15}.

b9) ACTUALISATION DE MACHINE : machine, NOM DE MODELE, NOM DE MACHINE, ACTUALISATION {h19}.

c) *Interface privée d'une machine*

c1) INTERFACE : vide;
interface, Σ(DESIGNATIONS), fin interface.

c2) DESIGNATIONS : DESIGNATIONS D'ENTREES;
DESIGNATIONS DE SORTIES.

c3) DESIGNATIONS D'ENTREES : PROVENANCE, Π(GROUPE), point-virgule.

c4) PROVENANCE : consigne;
mesure.

c5) GROUPE : logique, Π(IDENTITE SIMPLE);
TYPE-NON-LOGIQUE, Π(IDENTITE).

c6) TYPE-NON-LOGIQUE : analogique;
alphanumérique.

c7) IDENTITE SIMPLE : NOM DE VOIE, égal, NUMERO DE VOIE.

c8) IDENTITE : NOM DE VOIE, égal, par-gauche, NUMERO DE VOIE, virgule,
PROCEDURE DE TRAITEMENT, par-droite.

c9) NUMERO DE VOIE : nombre entier.

c10) PROCEDURE DE TRAITEMENT : "procédure décrivant l'interprétation de
l'entrée analogique ou alphanumérique".

c11) DESIGNATIONS DE SORTIES : DESTINATION, Π(GROUPE DE SORTIES), point-
virgule.

c12) DESTINATION : affichage;
commande.

c13) GROUPE DE SORTIES : GROUPE;
impulsion, Π(IDENTITE SIMPLE).

d) *Déclarations de noms de constantes*

d1) DECLARATIONS : vide;
Σ(EQUIVALENCE).

d2) EQUIVALENCE : entier, NOM DE CONSTANTE, égal, notation d'entier, point-virgule;
réel, NOM DE CONSTANTE, égal, notation de réel, point-virgule;
délai, NOM DE CONSTANTE, égal, notation de délai, point-virgule.

e) *Automate d'enchaînement*

e1) AUTOMATE : automate, ETAT INITIAL, ENCHAINEMENT, fin, automate.

e2) ETAT INITIAL : état initial, NOM D'ETAT, point-virgule.

e3) ENCHAINEMENT : CYCLE;
CYCLE, Σ(SOUS-ENCHAINEMENT).

e4) SOUS-ENCHAINEMENT : DEFINITION DE CYCLE;
DEFINITION DE CHAINE.

e5) DEFINITION DE CYCLE : NOM DE CYCLE, égal, CYCLE, point-virgule.

e6) DEFINITION DE CHAINE : NOM DE CHAINE, égal, CHAINE, point-virgule.

e7) CYCLE : crochet-gauche, Σ(CHEMIN, LIAISON) crochet-droit;
crochet-gauche, NOM DE CYCLE crochet-droit.

e8) CHEMIN : CYCLE;
CHAINE.

e9) LIAISON : CHAINE A CYCLE;
CYCLE A CHAINE;
CYCLE A CYCLE.

e10) CHAINE A CYCLE : vide;
deux-points Λ(EVENEMENT, vers, NOM D'ETAT).

e11) CYCLE A CHAINE : vide;
Λ(NOM D'ETAT, deux-points, EVENEMENT), vers.

e12) CYCLE A CYCLE : vide;
Λ(NOM D'ETAT, deux-points, EVENEMENT, vers, NOM D'ETAT).

e13) CHAINE : NOM D'ETAT, deux-points, Λ(CHAINE OUVERTE), vers, NOM D'ETAT par-gauche, NOM DE CHAINE, par-droite.

e14) CHAINE OUVERTE : EVENEMENT;
EVENEMENT, vers, NOM D'ETAT, Λ(CHAINE OUVERTE);
EVENEMENT, vers, CHAINE, deux-points, Λ(CHAINE OUVERTE).

e15) EVENEMENT : EXPRESSION LOGIQUE {il}.

f) *Ordres de machines combinatoires ou séquentielles*

f1) COMBINATOIRE : ordres, Π(SPECIFICATION D'ACTION) fin ordres.

f2) ORDRES : ordres, Σ(NOM D'ETAT, deux-points, Π(SPECIFICATION D'ACTION), point-virgule).

f3) SPECIFICATION D'ACTION : APPEL D'ACTION;
APPEL D'ACTION, sur, EXPRESSION LOGIQUE {il};
APPEL D'ACTION, de, DELAI, à,DELAI;
NOM DE SYSTEME OU DE MACHINE.

f4) DELAI : NOM DE DELAI;
notation de délai.

f5) APPEL D'ACTION : NOM DE SORTIE LOGIQUE OU IMPULSIONNELLE;
NOM DE SORTIE ANALOGIQUE OU ALPHANUMERIQUE, par-gauche, EXPRESSION, par-droite;
APPEL DE PROCEDURE.

f6) APPEL DE PROCEDURE : PREFIXE D'APPEL, NOM DE PROCEDURE, PARAMETRAGE.

f7) PREFIXE D'APPEL : vide;
NOM DE MODULE, carré.

f8) PARAMETRAGE : vide;
par-gauche Π(PARAMETRE) par-droite.

f9) PARAMETRE : notation d'entier;
notation de réel;
notation de délai;
NOM DE CONSTANTE;
NOM D'ENTREE.

f10) EXPRESSION : " expression arithmétique construite à partir de notations de constantes entières et réelles, de noms de constantes et de procédures de même type";
ou
"notation de constante alphanumérique","ou nom de constante alphanumérique".

g) *Modèles de machines, modules, et procédures*

g1) MODELES DE MACHINE : vide;
Σ(M-MACHINE).

g2) M-MACHINE : modèle, machine, NOM DE MODELE, par-gauche, Π(METAVARIABLE), par-droite, "description d'une machine {b} où des idf peuvent être remplacés par des METAVARIABLEs".

g3) MODELES DE CALCUL : vide;
Σ(M-CALCUL).

g4) M-CALCUL : modèle, module, NOM DE MODELE, par-gauche, Π(METAVARIABLE), par-droite, "CORPS DE MODULE {h4} où des idf peuvent être remplacés par des METAVARIABLEs";
modèle, CLASSE DE PROCEDURE {h10}, NOM DE MODELE, par-gau Π(METAVARIABLE), par-droite, "CORPS DE PROCEDURE {h13} où des idf peuvent être remplacés par des METAVARIABLEs".

g5) MODELES D'ELEMENTS : vide;
Σ(M-ELEMENT).

g6) M-ELEMENT : "M-CALCUL où PROCEDURE est remplacée par PROCEDURE PERIODIQUE";
M-MACHINE.

g7) METAVARIABLE : ampersand, idf.

h) *Modules et procédures*

h1) CALCULS : vide;
Σ(UNITE DE CALCUL).

h2) UNITE DE CALCUL : MODULE;
PROCEDURE;
ACTUALISATION DE MODULE;
ACTUALISATION DE PROCEDURE.

h3) MODULE : module, NOM DE MODULE, CORPS DE MODULE.

h4) CORPS DE MODULE : "déclarations des variables partagées par les procédures", Σ(PROCEDURE), CONTRAINTES, fin, NOM DE MODULE.

h5) CONTRAINTES : vide;
Σ(EXPRESSION DE CHEMIN).

h6) EXPRESSION DE CHEMIN^{o} : chemin SEQUENCE finchemin.

h7) SEQUENCE : SOUS-SEQUENCE;
SEQUENCE, point-virgule, SOUS-SEQUENCE.

h8) SOUS-SEQUENCE : Λ(NOM DE PROCEDURE).

h9) PROCEDURE : CLASSE DE PROCEDURE, NOM DE PROCEDURE, PARAMETRAGE FORMEL, égal, CORPS DE PROCEDURE.

h10) CLASSE DE PROCEDURE : action;
prédicat;
fonction TYPE.

h11) TYPE : entier;
réel.

h12) PARAMETRAGE FORMEL : vide;
par-gauche, Π(TYPE, NOM DE PARAMETRE FORMEL), par-droite.

h13) CORPS DE PROCEDURE : "programme séquentiel écrit dans un langage non défini ici".

h14) MODULE DE TRAITEMENT : "même définition qu'un MODULE mais où PROCEDURE est remplacée par PROCEDURE PERIODIQUE".

h15) PROCEDURE PERIODIQUE : CLASSE DE PROCEDURE, NOM DE PROCEDURE, PERIODICITE, PARAMETRAGE FORMEL, égal, CORPS DE PROCEDURE.

h16) PERIODICITE : vide;
par-gauche, DELAI {f4}, par-droite.

o Une SEQUENCE, une SOUS-SEQUENCE ou un NOM DE PROCEDURE, peuvent être suivis du symbole '*' pour marquer la répétition possible.

h17) ACTUALISATION DE MODULE : module, NOM DE MODELE , NOM DE MODULE, ACTUALISATION, point-virgule.

h18) ACTUALISATION DE PROCEDURE : CLASSE DE PROCEDURE, NOM DE MODELE, NOI DE MODULE, ACTUALISATION, point-virgul

h19) ACTUALISATION : par-gauche, Π(NOM DE PARAMETRE DE MODELE), par-droi

i) *Expressions logiques*

i1) EXPRESSION LOGIQUE : "expression logique construite à l'aide des opérateurs ou, et, négation et des ATOMEs".

i2) ATOME : NOM D'ENTREE;
NOM DE MACHINE, par-gauche, NOM d'ETAT, par-droite;
NOM DE PREDICAT, PARAMETRAGE {f8 }.

BIBLIOGRAPHIE

AFCET 72 — Groupe ALGOL de l'AFCET
La langage algorithmique ALGOL 68
J.ANDRE, P.BACCHUS, C.PAIR (Ed.), Hermann, 1975.

AFCET 77 — Groupe de travail Systèmes Logiques de l'AFCET.
Pour une représentation normalisée du cahier des charges d'un automatisme logique.
Automatique Informatique industrielle; n° 61, Nov. 77.

ANCEAU 75 — ANCEAU, MARINESEN, HENRY, POTET.
MSQ : a micro-segment as a monolithic component.
Euromicro Workshop, Nice, June 75.

ANDRE 72 — J. ANDRE, P. BACCHUS, C. PAIR(Ed).
Le langage algorithmique ALGOL 68.
Hermann, 1972.

AOKI 72 — M. AOKI
Some iterative schemes for decentralized dynamic control system
Congrès IFAC, Paris, 1972.

BARNES 76 — J.G.P. BARNES
RTL/2, Design and philosophy
Heyden ed, 1976.

BETOURNE 77 — C. BETOURNE & Co.
Le langage LEST.
Journées sur la production des systèmes industriels.
Groupe BIGRE de l'AFCET, Nov. 77.

BINDER 77 — Z. BINDER
Sur organisation et conduite des systèmes complexes.
Thèse d'Etat, USMG/INPG, Grenoble, Avril 77.

BRARD 76 — P. BRARD.
L'organiphase : méthode de représentation graphique descriptive d'un automatisme séquentiel.
Colloque AFCET/ADEPA sur les automatismes logiques, Paris, Dec. 76.

BRINCH HANSEN 75 — P. BRINCH HANSEN
The programming language concurrent PASCAL.
IEEE Transactions on Software Engineering 1, 199, 1977.

CHEVAL 77 — J.L. CHEVAL, F. CRISTIAN, S. KRAKOMIAK, J. MONTUELLE, I. MOSSIERE
An experiment in modular program design.
Information processing 77, IFIP, Toronto, August 77.

CHIARAMELLA 74 — Y. CHIARAMELLA
Cobol : presentation d'un sous-ensemble du langage.
Eléments de programmation structurée. Cours IUT, Grenoble, 1974-75.

CLARK 75 — CLARK, MARKLAM, BROWN
Automated design of microprocessor-based controllers.
IEEE, vol IECI-22, n° 3, August 75.

CONWAY 63 — M.E. CONWAY
Design of a separable transition-diagram compiler.
CACM vol 6, n° 7, July 1963, pp. 396-408.

COPIN 77 — B. COPIN
L'automatique dans les moyens de production d'énergie électrique.
Congrès S.E.E., Septembre 77.

COURTIN 74 — J. COURTIN, J. VOIRON
Traduction des schémas de programme en Fortran. Application aux structures de données.
Cours IUT, Grenoble, 1974-75.

DACLIN 77 — F. DACLIN, M. BLANCHARD
Synthèse des systèmes logiques.
Cepadu s éditions, 1977.

DAVID 69 — R. DAVID
Synthèse de réseaux séquentiels cellulaires.
Automatisme n° 11, tome XIV, Nov. 69, pp. 554-559.

DESCHIZEAUX 74 — P. DESCHIZEAUX, P. LADET
Test du langage et du système Procol.
Compte rendu de contrat D.G.R.S.T., n° 72 7 0544, INPG, Grenc 1974.

DESCHIZEAUX 75 — P. DESCHIZEAUX, Z. BINDER
Spécification de structures informatiques pour la conduite de procédés.
Symposium and course on mini and micro-computers, Zurich, Jun

DESCHIZEAUX 77 — P. DESCHIZEAUX, P. LADET
Programmation en temps réel des synchronisations par gestion des liens événements-Processus.
Colloque sur la Programmation globale des applications en Temps Réel, Paris, Nov. 77.

DIJKSTRA 75 — E.W. DIJKSTRA
Guarded Commands, non determinacy and Formal Derivation of Programs.
CACM, vol 18, Nb 8, August 75.

ELZER 75 — P.F.ELZER
PEARL, Journées AFCET sur la langages
Temps Réel, Paris, 1975.

FINANCE 77 — J.P. FINANCE
Formalisation de la sémantique d'un langage de type déclaratif.
Groplan n°2, AFCET, 1977.

FOSSARD 72 — A.J. FOSSARD, M. CLIQUE, N. IMBERT
Aperçus sur la commande hiérarchisée.
R.A.I.R.O. , n° août 1972, J-3, pp. 3-40.

FOULARD 77 — C. FOULARD, S. GENTIL, J.P. SANDRAZ
Commande et régulation par calculateur numérique.
Eyrolles, 1977.

FYLSTRA 75 — D.H. FYLSTRA
The Real Time Features of HAL/S
Journées Temps Réel de l'AFCET, Nov. 75.

GERTLER 75 — J. GERTLER, J. SEDLACK
Software for process control - A survey.
Automatica, vol 11, pp. 613-625, 1975.

GIRSCHIG 76 — Conduite simultanée de plusieurs processus séquentiels par un même ordinateur.
Colloque AFCET/ADEPA sur les automatismes logiques, Paris, Dec. 76.

HAASE 77 — V. HAASE, H. HALLING
Description of real-time applications using the guarded commands concept.
Colloque sur la programmation globale des Synchronisations dans les Applications en Temps Réel, Paris, Nov. 77.

HOLDEN 77 — J. HOLDEN, I.C. WAND
Experience with programming language Modula.
IFAC/IFIP Workshop on Real Time Programming, Eindhoven, June 77.

ICHBIAH 74 — J.D. ICHBIAH, J.P. RISSEN, J.C. HELIARD
The two level approach to data independant programming in the LIS system implementation language. Machine oriented higher level languages.
North Holland, 1974.

JACKSON 76 — K. JACKSON, H.F. HARTE
The achievment of well structured software in real time application.
Séminaire IFAC/IFIP sur la programmation en temps réel, Paris 1976.

LADET 77
P. LADET
Outils de structuration temps réel dans la commande des procéd industriels.
Thèse 3e cycle, INPG, Grenoble, Mars 1977.

LAUER 75
P.E. LAUER, R.H. CAMPBELL
Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes.
Acta Informatica 5, 297-332 (1975).

LAUER 77
P.E. LAUER, M.W. SHIELDS
Abstract specification of resource accessing disciplines : adequacy, starvation, priority and interrupts.
Colloque sur la Programmation globale des Synchronisations dans les applications en Temps Réel, Paris, Nov. 77.

LEBOURGEOIS 67
F. LEBOURGEOIS
Traitement de problèmes logiques et séquentiels par calculateur numérique.
Thèse Docteur-ingénieur, Grenoble, mars 67.

LE CALVEZ 77-1
F. LE CALVEZ, F. MADAULE, H.G. MENDELBAUM
Compiling Gaëlic : a global real-time language.
IFAC/IFIP workshop on real time programming, Eindhoven,June 77.

LE CALVEZ 77-2
F. LE CALVEZ
Gaëlic : un langage de description globale des synchronisations de processus.
Colloque sur la programmation globale des synchronisations dans les applications en temps réels, Paris, Nov. 77.

LOMET 73
D.B. LOMET
A formalization of Transition Diagram systems.
JACM, vol 20, april 1973, pp. 235-257.

MENDELBAUM 77
H.G. MENDELBAUM
Structuration dans la conception et la réalisation des systèmes en temps réel.
Thèse d'Etat, Paris, janv. 77.

MESAROVIC 71
M.D. MESAROVIC
Multilevel theory state of the art.
2nd IFAC Symposium on multivariable control systems, Dusseldorf, 1971.

RITOUT 72 — M.RITOUT, P.BONNARD, P.HUGOT
PROCOL: a programming system adapted for process control.
Congrès IFAC, Paris, 1972.

ROBERT 77 — P. ROBERT, J.P. VERJUS
Toward autonomous descriptions of synchronisation modules.
Information processing 77, Toronto, August 77.

ROUSSEAU 76 — R. ROUSSEAU, M. HENRY
Etat de la technologie à moyen terme.
Colloque AFRET/ADEPA sur les automatismes logiques, Paris, Dec. 76.

RUSSO 77 — M. RUSSO
Interprocessor communication for multi-computer systems,
Computer, 10-4, 67 (1977).

SABATHE 77 — P. SABATHE
Computer implementation of automata for electricity distributi control.
IFAC/IFIP workshop on Real Time Programming, Eindhoven, June 7

SILVA 76 — M. SILVA
Sur la description fonctionnelle des automatismes logiques.
N.I. LAG/INPG 77-04, Grenoble, Dec. 76.

SILVA 77-1 — M. SILVA, J. PLEYBER
Un automate programmable basé sur les Graphes d'Etats.
N.I. LAG/INPG 77-06 et R.R. IMAG/USMG 70, Grenoble, Mars 77.

SILVA 77-2 — M. SILVA
Tour d'horizon sur les automates programmables.
N.I. LAG/INPG n° 77-07, Grenoble, Mars 77.

TITLI 75 — A. TITLI
Commande hiérarchisée et optimisation des systèmes complexes.
Dunod, Paris, 1975.

TOURRES 76 — L. TOURRES
Une méthode d'étude des systèmes logiques et son application à la réalisation d'automatismes programmés.
Revue générale de l'Electricité, t. 85, n° 3, Mars 76.

MOALLA 76
M. MOALLA, J. SIFAKIS, M. ZACHARIADES
MAS : un outil d'aide à la description et à la conception des automatismes logiques.
Colloque AFCET/ADEPA sur les automatismes logiques, Paris, Dec. 76.

MOR 72
I. MOR, J. YOTT, H. ZIMA
Progress - A Programming Language for Real-Time Systems.
SIGPLAN notices, vol 7, Nb 6, 1972 June.

PARNAS 77
D.L. PARNAS
The use of precise specifications in the development of software
Information processing 77, IFIP, August 77.

PEIRANO 75
PEIRANO, VIESS
Système AUTOMAT pour la programmation d'automatismes séquentiels industriels.
Journées d'études sur les automatismes logiques, AFCET, Toulouse, 1975.

PIKE 70
H.E. PIKE
Process Control software.
IEEE Proceedings 1970, vol 58, n 1.

PLEYBER 77-1
J. PLEYBER, M. SILVA
Software specification for sequential processes.
IFAC/IFIP Real Time Programming Workshop, Eindhoven, June 77.

PLEYBER 77-2
J. PLEYBER, M. SILVA
Un langage pour la programmation de commande de procédés.
Colloque sur la programmation globale dans les applications en temps Réel, Paris, Nov. 77.

RAMOS 75
G. RAMOS-NIEMBRO
Contribution à l'étude de la commande séquentielle des procédés complexes.
Thèse 3e cycle, USMG/INPG, 1975.

RAMOS 76
G.RAMOS-NIEMBRO, R.DAVID
Une approche de commande séquentielle de procédé complexe.
Colloque AFCET/ADEPA sur les automatismes logiques, PARIS, décembre 1976.

UPS 76
UPS/UER informatique, ONERA/CERT, CNRS/LAAS
Mémoire de définition du projet pilote sûreté de fonctionnement des systèmes informatiques.
Toulouse, janvier 76.

VALETTE 76
R. VALETTE
Sur la description, l'analyse et la validation des systèmes de commande parallèles.
Thèse d'Etat, UPS, Toulouse, Nov. 76.

WEISS 74
C.D. WEISS
Basic microcomputer software : Algorithm for a traffic-light controller illustrates principles that can be used with any MOS/LSI microprocessor.
Electronic Design 9, April 76.

WIRTH 77-1
N. WIRTH
MODULA : A language for modular multiprogramming.
Software - Practise and experience 7 n° 1, 3, 1977.

WIRTH 77-2
N. WIRTH
Toward a discipline of Real Time Programming.
CACM, vol 20, Nb 8, pp. 577-583, August 1977;

WOODWARD 70
P. WOODWARD AND AL.
Official Definition of Coral 66.
HMSO, 1970.

ZACHARIADES 77
M. ZACHARIADES
MAS : réalisation d'un langage d'aide à la description et à la conception des systèmes logiques.
Thèse 3e cycle, INPG, Grenoble, Sept. 77.

Dernière page d'une thèse

VU

Grenoble, le 7 Février 1978

Le Président de la thèse L. Bollet

Vu, et permis d'imprimer,

Grenoble, le 16 février 1978

Le Président de l'Université
Scientifique et Médicale

UNIVERSITÉ SCIENTIFIQUE ET MÉDICALE DE GRENOBLE

Dr G. CALI