N° d'ordre : 310

50376
1989
11

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE

en

PRODUCTIQUE, AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

par

Etienne CRAYE

DE LA MODELISATION A L'IMPLANTATION AUTOMATISEE DE LA COMMANDE HIERARCHISEE DE CELLULES DE PRODUCTION FLEXIBLES DANS L'INDUSTRIE MANUFACTURIERE

soutenue le 13 Janvier 1989 devant la Commission d'Examen

Membres du Jury :	M. VIDAL	Examinateur, Président
	M. POSTAIRE	Rapporteur
	M. SILVA	Rapporteur
	M. VERON	Rapporteur
	M. CORBEEL	Examinateur
	M. GENTINA	Examinateur, Directeur de Thèse
	M. GRAVE	Invité
	M. THUREL	Invité

AVANT - PROPOS

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Informatique Industrielle de l'Institut Industriel du Nord sous la direction scientifique de Monsieur GENTINA, Professeur à l'I.D.N. et en collaboration avec Monsieur le Professeur VIDAL du Laboratoire d'Automatique de LILLE I qui me fait l'honneur de présider le Jury. Qu'ils trouvent ici le témoignage de ma profonde gratitude.

Je tiens également à remercier très sincèrement :

Monsieur POSTAIRE, Professeur à l'U.S.T.L.F.A.,

Monsieur SILVA, Professeur à l'Ecole d'Ingénieurs de SARAGOSSE,

Monsieur VERON, Professeur à l'Université de Nancy I,

pour avoir accepté d'être les rapporteurs de mon travail.

Je suis très reconnaissant à Monsieur CORBEEL, Maître de Conférence à l'I.D.N., de faire partie de mon Jury et d'accepter de juger ce travail.

Je suis également flatté de la présence à ce Jury de Messieurs GRAVE et THUREL de la direction Recherche et Développement de la société TELEMECANIQUE, qui ont bien voulu consacrer du temps pour assister à la présentation de mon mémoire.

Qu'il me soit permis de rendre hommage à Monsieur GENTINA, ainsi qu'à Tous les membres du Laboratoire d'Informatique Industrielle de l'I.D.N., pour l'aide précieuse qu'ils m'ont apportée tant sur le plan scientifique que sur le plan humain. Leur sympathie, leur disponibilité et leur spontanéité ont été d'une grande aide durant ces trois années.

Enfin, je remercie très sincèrement les personnes qui ont assuré la réalisation matérielle de ce mémoire : Mesdames TRICOT et VERIN pour la dactylographie et Monsieur VANGREVENINGE pour la reprographie.

SOMMAIRE

INTRODUCTION GENERALE p. 17

CHAPITRE I p. 25

CHAPITRE II p. 77

CHAPITRE III p. 179

CONCLUSION GENERALE p. 241

BIBLIOGRAPHIE p. 247

ANNEXES p. 265

CHAPITRE I

Introduction p. 29

I Description du modèle p. 31
- I1 Introduction p. 31
- I2 Les réseaux de Petri Structurés p. 31
  - I21 Introduction p. 31
  - I22 Graphe de processus p. 32
  - I23 Les liaisons inter-processus p. 33
  - I24 Conclusion p. 34
- I3 Les réseaux de Petri Structurés, Adaptatifs et Colorés p. 35
  - I31 Introduction p. 35
  - I32 Les réseaux de Petri Adaptatifs p. 35
  - I33 Les réseaux de Petri Colorés p. 36
  - I34 Conclusion p. 37
- I4 Transposition des RdPSAC en Grafcet p. 38
  - I41 Introduction p. 38
  - I42 Transposition du modèle de base p. 38
  - I43 Transposition des réseaux Adaptatifs p. 44
  - I44 Transposition de la coloration p. 47
  - I45 Conclusion p. 50
- I5 Conclusion p. 51

II Présentation du projet CASPAIM p. 55
- II1 Introduction p. 55
- II2 La démarche p. 55
- II3 Limites et extensions p. 64
  - II31 Exemple 1 p. 67
  - II32 Exemple 2 p. 69
- II4 Situation de nos travaux p. 72
- II5 Conclusion p. 73

Conclusion p. 75

CHAPITRE II

Introduction p. 81

I Le fonctionnement en mode de marche normale p. 83

I1 Introduction p. 83

I2 La ressource p. 84

I21 Définition du problème p. 84

I22 Solution de Dekker p. 86

I23 Solution répartie exprimée en Grafcet p. 88

I24 Solution utilisant un superviseur p. 92

I3 Le producteur/consommateur p. 96

I31 Définition p. 96

I32 Solution répartie exprimée en Grafcet p. 97

I33 Solution utilisant un superviseur p. 100

I34 Solution incluant la coloration p. 101

I4 La synchronisation avec accusé de réception p. 104

I41 Définition p. 104

I42 Solution répartie exprimée en Grafcet p. 104

I43 Solution utilisant un superviseur p. 107

I5 Les conflits et indéterminismes p. 108

I51 Introduction p. 108

I52 Accès à des ressources exclusives p. 109

a) Interfaçage avec la partie commande p. 111

b) Structuration du niveau hiérarchique p. 113

I53 Les indéterminismes directionnels p. 115

I6 Les blocages p. 120

I7 Conclusion p. 123

II Sureté de fonctionnement p. 125

II1 Introduction p. 125

II2 Définitions p. 126

II3 La surveillance interne p. 129

II4 La surveillance séparée p. 133

II41 Interfaçage partie commande <-> niveau hiérarchique p. 133

II42 Mécanismes d'interaction sur la commande p. 135

a) Le gel p. 135

b) La reprise p. 137

c) La configuration dynamique p. 139

II43 Le niveau hiérarchique p. 143
II5 Conclusion p. 145

III Réalisation du niveau hiérarchique p. 147
III1 Introduction p. 147
III2 Les bases de faits p. 148
III21 Définitions p. 148
III22 Représentation interne des faits p. 148
III3 La base de connaissance p. 150
III31 Définition syntaxique d'une règle p. 150
III32 Base de règles et base de méta-règles p. 152
III4 Le moteur d'inférence p. 153
III41 Introduction p. 153
III42 Justification du fonctionnement p. 154
a) Chaînage avant / Chaînage arrière p. 154
b) Comparatif entre chaînage avant et chaînage arrière p. 155
c) Intérêt du chaînage avant pour le niveau hiérarchique p. 159
d) Fonctionnement du moteur <<en largeur d'abord>> p. 162
e) Fonctionnement irrévocable / Fonctionnement par tentatives p. 163
f) Système dynamique p. 165
III43 Conclusion p. 169
III5 Cycle complet de résolution p. 170
III6 Conclusion p. 176

Conclusion p. 177

CHAPITRE III

Introduction p. 183

I Réalisation effective du logiciel p. 185
- I1 Introduction p. 185
- I2 Organisation générale du logiciel p. 185
  - I2a Les modules d'édition et de modification p. 186
  - I2b Le module d'évaluation p. 187
- I3 Implantation informatique p. 191
  - I31 Introduction p. 191
  - I32 Le logiciel LINK7PC p. 193
  - I33 Interfaçage Le_LISP <-> Langage C p. 195
  - I34 Conclusion p. 198
- I4 Conclusion p. 199

II Exemple de réalisation p. 205
- II1 Introduction p. 205
- II2 Présentation de la cellule p. 206
- II3 Application de la méthodologie p. 210
  - II31 Elaboration du prégraphe p. 210
  - II32 Structuration p. 211
  - II33 Simulation p. 217
- II4 Implantation p. 218
  - II41 Introduction p. 218
  - II42 Répartition des graphes de commande p. 219
  - II43 Programmation des Grafcets et du niveau hiérarchique p. 222
  - II44 Intérêt des métarègles p. 233
  - II45 Bilan de mise en œuvre de l'exemple d'implantation p. 235
- II5 Conclusion p. 236

Conclusion p. 239

INTRODUCTION GENERALE

Réduire les encours, raccourcir les délais, augmenter le taux d'engagement des machines-outils, plus généralement accroître la productivité, ..., autant d'arguments qui plaident pour la mise en œuvre d'unités de production flexibles. Potentiellement, un grand nombre d'entreprises et de secteurs d'activité est concerné par ce changement d'autant que le contexte économique actuel tend vers un développement de la sous-traitance pour des petites séries de pièces de petits volumes impliquant un changement rapide des produits et une adaptation quasi-permanente de la production.

Les moyens techniques de mise en œuvre sont, dans ce domaine, en avance sur les outils et modèles de conception associés. Tout concepteur dispose de (micro-)ordinateurs industriels, d'automates ou commandes numériques perfectionnées, de réseaux locaux de communication, de robots, de machines flexibles, ... La difficulté de réalisation ne provient ainsi pas en premier lieu d'un problème matériel. Elle se situe ailleurs et plus particulièrement dans l'absence de formalisation des concepts de base pour une automatisation intégrée [MOR 88], et dans l'absence de méthodologie de conception efficace visant à définir une démarche **progressive** et **modulaire** concernant la conception et la mise en œuvre d'un atelier flexible. Il en résulte des coûts d'études allant jusqu'à 40 % de l'investissement global (Citroën / Meudon a demandé 75.000 heures d'étude dont 25.000 pour le logiciel [DEF 85]) pour des réalisations qui, une fois terminées, ne répondent pas forcément à l'attente initiale et dont la rentabilité n'est pas toujours significative.

La définition d'une démarche méthodologique est devenue indispensable à tous les niveaux du cycle de vie d'un projet d'automatisation poussée des moyens de production. Une analyse descendante et structurée doit être menée, des spécifications informelles du cahier des charges jusqu'à la réalisation effective de l'installation. Des recherches allant dans ce sens sont menées depuis quelques années dans plusieurs laboratoires. Des logiciels d'aide à la spécification et à la réalisation de commande et simulation ont déjà été proposés. Ces travaux sont l'objet de nombreuses communications et sont recensés lors des réunions de la Commission Système Logique de l'A. F. C. E. T. qui sert en partie de lien relationnel entre les laboratoires et les industriels confrontés aux problèmes de conception d'automatismes programmés. Il ressort de l'ensemble des expériences acquises que la conception d'une commande flexible doit procéder de façon cohérente et

selon quatre étapes essentielles :

(i) En premier lieu, l'analyse des besoins exprimés dans le cahier des charges doit être abordée, suivie d'une traduction formelle à partir de laquelle une pré-étude peut être menée. La formalisation permet en particulier de mettre en avant une analyse de complétude de la description et à contrario les incohérences de la spécification.

(ii) La modélisation structurée, assistée par logiciel et progressive, fait suite à la première étape. La structuration doit permettre de dissocier la description par fonctionnalité et séparer par exemple la définition des algorithmes généraux de stratégie de production par rapport à la mise en œuvre du séquentiel des automatismes. Elle doit également participer à la définition des outils de modélisation de façon analogue aux concepts retenus en informatique classique.

(iii) Le troisième point consiste en une simulation de la commande afin d'obtenir une évaluation qualitative et quantitative de la description. Elle est fondamentale et ne doit pas se contenter de valider un résultat que l'on estime acquis par ailleurs. Elle participe activement à la chaîne de conception et reste le dernier rempart "off-line" de l'étude.

(iv) Enfin, l'implantation effective des commandes sur les supports informatiques disponibles achève la démarche. Cette étape reprend les modèles pour une traduction en code implantable et exécutable. Elle est confrontée à des difficultés, classiques mais problématiques, de l'informatique : communication, parallélisme, asynchronisme, ... Elle peut être également révélatrice d'une mauvaise adéquation entre les modèles théoriques utilisés en phase de conception et la transposition effective de ces mêmes modèles pour la réalisation concrète.

C'est sur cette ossature minimale que s'est développé le projet C. A. S. P. A. I. M. (Conception Assistée de Systèmes de Production Automatisés pour l'Industrie Manufactu-

rière). Trois modèles correspondant chacun à un niveau fonctionnel différent ont été retenus pour parvenir à la description complète d'un système de production flexible. La partie opérative est représentée à partir d'un langage orienté objet. Les techniques associées à l'héritage et l'instanciation facilitent l'intégration et la description des constituants du procédé. Cette modélisation a pour objectif de donner une image réaliste du système de production en phase de simulation. Elle sera ainsi à même de refléter le comportement du procédé et permettra d'évaluer la commande en cas de pannes ou de changements de mode de marche. La partie commande, située au second niveau et concernant la modélisation du séquentiel des automatismes, utilise comme modèle de représentation, les réseaux de Petri. De par leur pouvoir de modélisation (d'autant plus grand que des extensions au modèle de base ont été définies), ils s'adaptent parfaitement à la description d'une large classe de systèmes automatisés à évènements discrets. Les aspects graphiques et procéduraux de ce modèle facilitent de plus leur utilisation. Enfin, le troisième modèle utilise un langage déclaratif ; chaque unité de connaissance est exprimée indépendamment des autres, sous forme d'une règle de production. Il constitue le niveau hiérarchique : sa fonction est de superviser la commande, de régler les indéterminismes émanant du modèle réseaux de Petri non autonome en intégrant des stratégies de fonctionnement complexes. Le formalisme des règles de production autorise une grande souplesse dans la définition des heuristiques de contrôle. La dissociation de ce dernier niveau vis à vis de la commande des automatismes en facilite les modifications a posteriori rendues nécessaires de par la difficulté à cerner la stratégie optimale de fonctionnement et le caractère éminemment flexible de l'unité de production.

Nos travaux se sont situés plus particulièrement en aval du projet, et sont axés sur l'implantation de la commande (niveau II) et du niveau hiérarchique (niveau III). A ce titre, nous avons donc orienté nos recherches afin de satisfaire les contraintes essentielles suivantes :

- Nous avons tout d'abord cherché à systématiser la traduction réseaux de Petri en Grafcet. L'objectif est ici de minimiser les erreurs de transcription pour obtenir un code dont le fonctionnement corresponde rigoureusement à la modélisation et à la simulation qui en a été faite.

- En second lieu, nous proposons de modifier la modélisation initiale en cas de répartition inter-automates. Cette étape induit la définition d'un protocole de communication nécessaire aussi bien pour une gestion tout en Grafcet que pour une solution utilisant le niveau hiérarchique.

- Enfin, nous nous sommes intéressé à la définition et mise en œuvre effective du niveau hiérarchique tant pour le mode de marche normal que pour les modes dégradés. Une technique de surveillance de type "espion" a été retenue pour sa réalisation.

Nous avons ainsi tenté d'apporter une assistance informatique à la phase d'implantation pour une transposition rigoureuse et automatisée du modèle théorique en un modèle d'implantation. Il en résulte à terme une fiabilité accrue par une standardisation du code généré qui en augmente la maintenabilité.

Ce mémoire comprend trois chapitres.

Le premier chapitre reprend le modèle de la commande exprimé en réseau de Petri et en présente une première traduction en Grafcet, dans une approche intra-automate. Cette traduction a été rendue obligatoire par l'offre disponible sur le marché qui ne propose pas encore d'automates acceptant une programmation en code source réseau de Petri. La seconde partie de ce chapite concerne également la présentation du projet C. A. S. P. A. I. M.. Après en avoir dégagé l'esprit et en insistant sur son intérêt, nous montrerons certaines de ces limites. Nous nous attacherons en particulier à démontrer comment, par une approche ensembliste de la conception et malgré une décomposition en trois niveaux fonctionnels différents, la modélisation finale peut se révéler inadaptée en phase d'implantation. Pour y remédier, nous apporterons des solutions qui consistent essentiellement à prendre en compte dès la modélisation, la configuration informatique réelle disponible.

Le deuxième chapitre aborde l'implantation dans une optique de décentralisation avec intégration du niveau hiérarchique. Dans un premier temps, il étend la traduction en Grafcet afin d'intégrer un contexte inter-automate. Il propose en premier lieu, des solutions

tout en Grafcet puis d'autres, mettant en œuvre le niveau hiérarchique. Il en résulte alors une simplification dans la réalisation par une diminution sensible des modifications à apporter au modèle initial. Dans un deuxième temps, ce chapitre présente l'intérêt du niveau hiérarchique pour une implantation orientée sûreté de fonctionnement. Il montre comment, par un interfaçage simple entre les modèles de niveau II et III, la commande est relevée des algorithmes complexes de gestion de production et de changement de modes de marche. L'intégration de ces concepts au niveau hiérarchique en augmente la puissance tout en facilitant leurs définitions, éventuellement a posteriori, sur un organe externe mais centralisateur de l'information. La dernière partie de ce chapitre développe les solutions retenues pour la réalisation du niveau hiérarchique. Elle y justifie l'utilisation d'un moteur d'inférence et y détaille son cycle de fonctionnement complet, asynchrone vis à vis de la commande.

Enfin, le dernier chapitre présente les problèmes informatiques rencontrés. Il montre comment la coopération entre des langages différents comme le LISP, le langage C et l'assembleur 8086 a été réalisée sur IBM PC/AT connecté par réseau aux automates programmables chargés de la commande d'une installation flexible. Il se termine par la présentation d'un exemple complet de réalisation d'une cellule.

CHAPITRE I

METHODOLOGIE GENERALE

INTRODUCTION

L'existence du Grafcet en tant que modèle de spécification d'un automatisme est une contribution essentielle à la conception, à la fois en raison de la puissance de description du modèle et de son caractère normatif. Cet outil est particulièrement adapté à la description et l'élaboration de la commande de procédés de production discontinus. Autour de cette méthodologie, se sont développés des systèmes de conception assistée s'appuyant sur une démarche comportant quatre phases principales :

- spécification de la commande,
- modélisation,
- validation à plusieurs niveaux,
- implémentation.

Ces systèmes s'inscrivent dans le cadre des travaux sur un poste de travail pour automaticien (projet PTA puis projet «base PTA» [RHE 88]) tels PIASTRE [PRU 87], les logiciels DEFI et ADELAIDE développés au C. R. A. N. de Nancy [TIX 88], le progiciel OMEGA de la Société 3IP ou la nouvelle gamme d'automates programmables TSX série 7 de Télémécanique. Les méthodologies mises en œuvre dans de telles stations ont pour objectifs principaux :

- l'apprentissage rapide par un non-informaticien des méthodes de conception,
- l'intégration importante de l'outil de représentation graphique à tous les niveaux : de l'édition à la mise au point,
- l'analyse descendante par un découpage en sous-systèmes de plus en plus élémentaires et détaillés (notion de "Macro-Etape").

Cependant, la prise en compte des fondements de l'analyse et de la programmation structurée constitue l'approche la plus sûre permettant d'aborder la complexité de grands systèmes industriels. Ainsi, de par la rigueur induite dans sa mise en œuvre, la structuration permet, grâce à l'apport d'outils de conception littéraux et surtout graphiques hautement directifs, l'introduction d'éléments de validation formelle de même qu'une première documentation systématique. En outre, la structuration est un point de passage obligatoire si l'on a pour objectif à terme l'intégration des modèles de représentation, de la spécifica-

tion à l'implantation, dans un système de conception automatisée de systèmes de commande.

Pour répondre à ces besoins, notre travail constitue une contribution aux recherches de méthodologies de conception développées au Laboratoire d'Automatique et d'Informatique Industrielle (L. A. I. I.) de l'I. D. N.. Ce projet intitulé C. A. S. P. A. I. M. (Conception Assistée de Systèmes de Production Automatisés pour l'Industrie Manufacturière) est plus particulièrement basée sur l'utilisation des Réseaux de Petri Structurés, Adaptatifs et Colorés (RdPSAC).

Ce chapitre se décompose en deux parties :

(i) La première partie a pour objet de rappeler la description du modèle de base (RdPSAC) et de quelques primitives littérales et graphiques associées. La transposition de ce modèle en Grafcet y est présentée et constitue l'une de nos contributions en vue de l'élaboration automatique du modèle d'implantation.

(ii) La présentation globale du projet C. A. S. P. A. I. M. constitue la seconde partie. Elle y détaille ses objectifs et limites et permet plus précisément de situer nos travaux dans le cadre de ce projet.

I - DESCRIPTION DU MODELE

I.1 - Introduction

Tant par l'étendue de leurs résultats théoriques que par la diversité et le nombre de leurs applications (Informatique, Automatique, Logiciel ou Matériel, ...), les réseaux de Petri constituent aujourd'hui le modèle formel le plus avancé et le plus complet pour la description logique des structures de contrôle du parallélisme. De nombreux travaux de recherche ont défini différentes classes d'extension par rapport au modèle de base : réseaux à capacité, à prédicats, à arcs inhibiteurs, ...

Afin d'obtenir une modélisation à la fois "simple" et puissante des process discontinus dans l'industrie manufacturière, nous avons restreint le modèle réseau de Petri à l'utilisation de réseaux structurés, adaptatifs et colorés. L'apport de la structuration analogue à celui qui découle de l'utilisation de langages informatiques tels PASCAL ou ADA, permet de concevoir et de modéliser de façon rigoureuse et sûre, la commande de grandes installations. Les arcs adaptatifs traduisent la flexibilité du système ; leur emploi permet de rendre déterministe le modèle et sert d'interface avec d'autres représentations (ex : modélisation d'un niveau hiérarchique, supervisant la commande et conditionnant son fonctionnement par l'intermédiaire de ces arcs adaptatifs). La coloration, qui est une abréviation des réseaux de Petri à prédicats où une marque peut recevoir un n-uple de valeurs, permet de différencier les différentes classes d'objets circulant dans le système. Elle modélise également par interprétation sur la couleur, l'état d'avancement de l'objet dans sa gamme opératoire.

I.2 - Les réseaux de Petri structurés [COR 79] [COR 80]

I.2.1 - Introduction

Un système de n processus est défini par la donnée :

- de chacun des n processus,
- des liaisons représentant les interactions entre ceux-ci.

L'analyse qui permet la décomposition d'un système industriel en n processus tient compte d'un certain nombre de critères :

- regroupement fonctionnel de tâches élémentaires et séquentielles formant un processus,
- implantation répartie sur différents organes de commande nécessitant de fait la création de processus séparés aux fonctions distinctes,
- respect du parallélisme imposé par la nature du procédé à piloter,
- impératif de temps de réponse découlant du temps de cycle de scrutation de chaque unité fonctionnelle.

Une fois la décomposition en n processus terminée, chacun de ceux-ci est alors décrit indépendamment des autres, à l'aide d'un graphe de processus.

I.2.2 - Graphe de processus

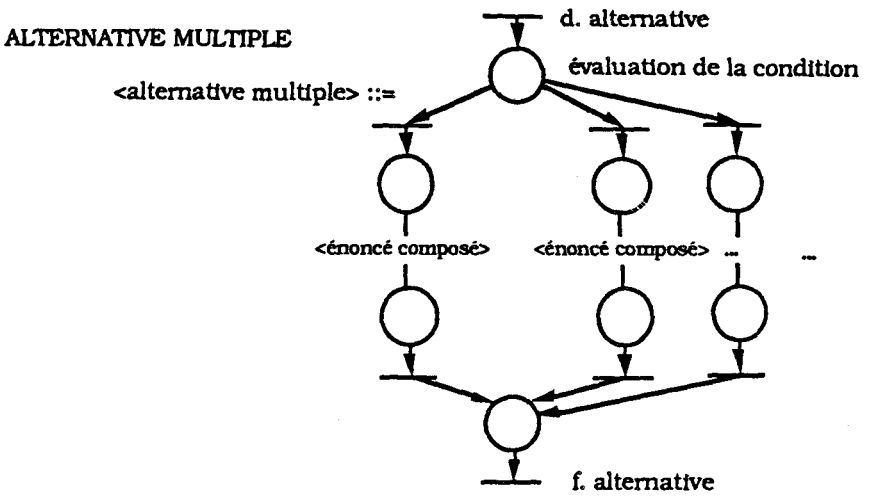
Un processus est décrit comme un enchaînement séquentiel de tâches élémentaires modélisé en réseau de Petri et qui utilise, pour sa définition, trois structures de base : l'action, l'alternative et la répétitive. La syntaxe associée est reprise en annexe.

Deux extensions complémentaires ont été adoptées pour étendre la syntaxe de base. Elles permettent d'obtenir une écriture plus concise pour la définition d'un graphe de processus sans pour autant nuire à sa lisibilité globale.

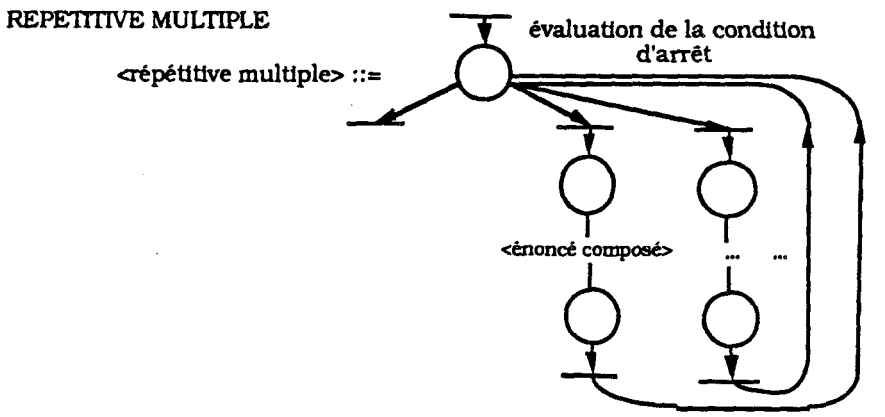
La première extension autorise l'écriture d'une alternative multiple sans utiliser la structure imbriquée de la forme :

si (condition) alors ... sinon si (condition) alors ... fin-de-si

prêtant souvent à confusion. Sa structure est calquée sur celle de la primitive "case" en Pascal et donne la syntaxe suivante :



La deuxième extension permet de modéliser une répétitive comportant plusieurs tests exclusifs dans son corps principal. Elle correspond en fait à l'intégration de l'équivalent d'un "case" dans sa boucle "while" et se représente ainsi :

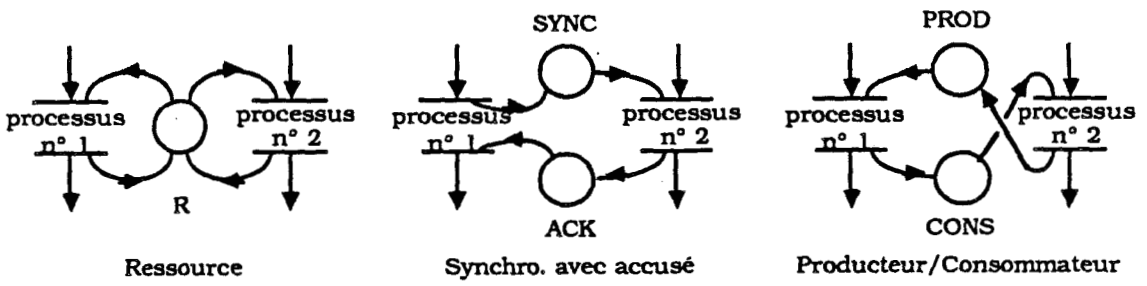


Enfin, le concept de bloc a été défini pour permettre primitivement le regroupement d'un ensemble de tâches élémentaires et séquentielles sous un même nom fonctionnel. Il a ensuite été étendu comme point de départ et de terminaison pour les primitives de liaison inter-processus.

I.2.3 - Les liaisons inter-processus

Trois primitives de liaison ont été retenues pour permettre l'interconnection des différents graphes de processus. Elles sont suffisantes pour modéliser les problèmes de communication et prennent en compte l'asynchronisme des process (Fig. 1).

La première primitive de liaison implémente l'exclusion mutuelle. Elle assure un accès exclusif à la ressource qu'elle protège. Son existence est rendue obligatoire aussi bien par le fonctionnement asynchrone des process modélisés que par le résultat d'une implantation répartie ultérieure sur différents organes de commande. La second primitive de liaison est la synchronisation avec accusé de réception. Elle permet, en des points particuliers de fonctionnement du système, de coordonner en les synchronisant des tâches élémentaires situées sur un ou plusieurs process. Enfin, la dernière primitive de liaison est le producteur / consommateur. C'est la seule primitive qui autorise un échange réel d'information entre deux graphes de processus.



Primitives de liaison

FIGURE 1

I.2.4 - Conclusion

L'utilisation de cette syntaxe, légèrement restrictive par rapport au modèle de base que sont les réseaux de Petri, permet néanmoins d'affirmer que la description globale sera sauve. Moyennant quelques précautions supplémentaires concernant la localisation des liaisons inter-processus, elle sera également vivante et réinitialisable. Cette modélisation présente l'intérêt d'être aisément implémentable et minimise les erreurs de conception par sa structuration durant l'analyse des problèmes.

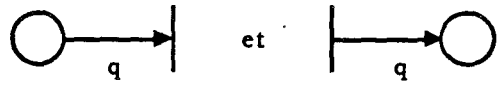
I.3 - Les Réseaux de Petri Structurés Adaptatifs et Colorés

I.3.1 - Introduction

Face à la complexité toujours croissante des systèmes de production, les réseaux de Petri structurés n'ont pas un pouvoir de description suffisamment puissant pour intégrer simplement tous les concepts imposés à la commande. La flexibilité notamment, critère obligatoire pour répondre aux impératifs de productivité et aux changements de mode de marche nécessités par la production ou par le procédé, ne peut être prise en compte par le modèle de base. De même, le nombre important des différentes gammes opératoires traitées aujourd'hui sur une même cellule de production, rend la description de la commande lourde et complexe sans autre outil que les réseaux de Petri structurés. Deux extensions ont donc été apportées au modèle de base pour répondre à ces contraintes.

I.3.2 - Les réseaux de Petri adaptatifs

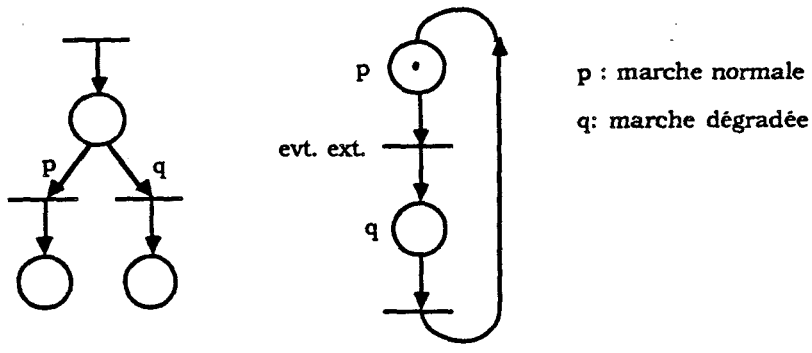
Rendre flexible la description de la commande par réseau de Petri a été l'objet de nombreux travaux visant à étendre la modélisation. Nous pouvons ainsi citer les réseaux à arcs inhibiteurs [HAC 75a], les réseaux à priorités [HAC 75b], les réseaux automodifiants [VAL 78], ... Dans le cadre des travaux du laboratoire, ont été retenus les réseaux adaptatifs dont la définition inclut les précédents. Un réseau de Petri adaptatif est défini comme un graphe ayant des arcs de la forme :



Si q = 1, le tir de la transition est identique au fonctionnement du modèle de base. Mais q peut également être le nom d'une autre place du graphe. Dans ce cas, le marquage de la place en amont de l'arc pondéré par q doit être supérieur ou égal au marquage courant de la place q pour que la transition soit déclenchée, à condition que la place q ne soit pas vide.

Les réseaux adaptatifs sont ainsi capables de modifier dynamiquement leur propre condition de fonctionnement et intègrent donc un comportement flexible. Nous pouvons

simplement par ce moyen, modéliser par exemple un changement de mode de marche nécessité par l'état du procédé (Fig. 2).

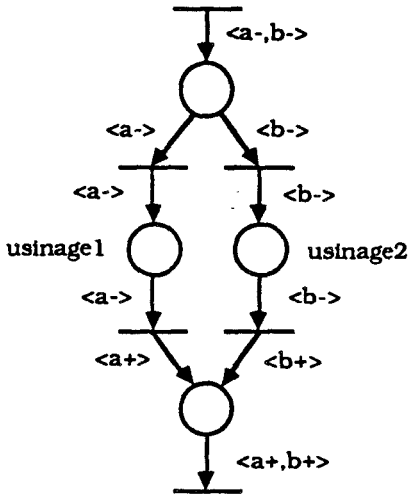


Changement de mode de marche
FIGURE 2

Si la place p est marquée l'arc adaptatif pondéré par la place q empêche l'évolution de la partie droite du graphe. Réciproquement, si la place q est marquée, c'est l'évolution vers la partie gauche qui est inhibée.

I.3.3 - Les réseaux de Petri colorés [COR 85]

La coloration a pour objectif de simplifier la structure d'un réseau en reportant des informations sur le marquage des places et en associant aux arcs les transformations effectuées sur ce marquage. Son utilisation permet de modéliser, au sein d'une même place, différentes classes d'entités sans avoir recours à plusieurs graphes. Elles autorisent également la représentation de l'état d'avancement d'un objet dans sa gamme opératoire par simple interprétation de la couleur de cet objet. La modélisation de deux usinages différents, fonctions de la classe de l'objet présenté, mais réalisés sur la même entité physique peut ainsi être traduite avec un nombre minimal de places (Fig. 3).



Modélisation de deux process d'usinage

FIGURE 3

Au sein de la première place, sont regroupées les différentes classes d'entités circulant dans le système (couleur a^- et b^-). Le choix de l'usinage est réalisé en identifiant la classe de l'objet et en étiquetant l'arc en fonction de la couleur admissible. A ce stade, la coloration permet de distinguer les classes ; par contre, à la fin de l'usinage, la couleur est modifiée (remplacement du suffixe "-" par le suffixe "+") pour traduire cette fois l'état d'avancement d'un objet dans sa gamme opératoire. Les différents objets sont enfin regroupés dans une même place de sortie car il n'y a plus lieu alors de leur associer des opérations spécifiques. La coloration utilisée ici a été simplifiée dans l'esprit de la structuration ; elle ne concerne que les objets et en aucun cas les ressources physiques (machines, systèmes de transport). L'abstraction réalisée reste ainsi facilement maîtrisable et compréhensible.

I.3.4 - Conclusion

La puissance d'abréviation des arcs adaptatifs ainsi que de la coloration favorisent la modélisation de systèmes en vraie grandeur. De plus, leur utilisation limite au minimum l'interprétation du modèle de la commande. Les transformations élémentaires (usinage, assemblage, ...) sont représentées de façon explicite ; la flexibilité de comportement est également intégrée au modèle. Enfin, les arcs adaptatifs permettent d'interfacer ce modèle avec une description située à un niveau différent (modélisation du procédé, du niveau

hiérarchique) par l'intermédiaire des places pondérant ces arcs.

I.4 - Transposition des Réseaux de Petri Structurés Adaptatifs et Colorés en Grafcet

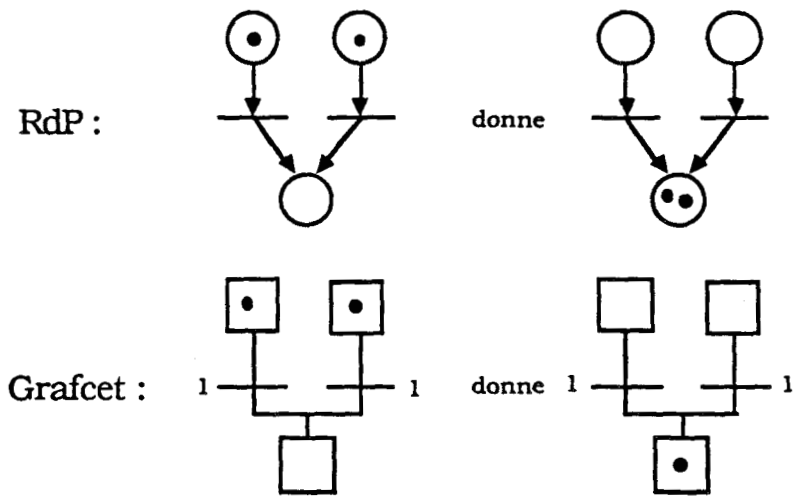
I.4.1 - Introduction

La définition des Réseaux de Petri Structurés Adaptatifs et Colorés (RdPSAC) est une contribution importante dans la mise en œuvre d'une méthodologie générale visant à concevoir des systèmes de production à haut degré de parallélisme. Toutefois, le modèle réseau de Petri, et plus encore ses extensions, ne sont pas à l'heure actuelle supportés par des organes de commande industriels. Les constructeurs d'automates programmables préférent de loin utiliser le modèle Grafcet comme outil de modélisation et de programmation. La Télémécanique, par exemple, bien que conservant pour des raisons historiques un langage à contacts proche des réalisations électriques cablées, propose le logiciel de programmation PL7-3 sur ses automates. Ce logiciel permet, suivant la configuration retenue, d'utiliser le Grafcet en mode graphique pour décrire le fonctionnement d'une application. Un langage littéral lui est associé pour la définition des actions couplées aux étapes et des réceptivités conditionnant le franchissement des transitions. Ainsi, la modélisation établie en RdPSAC doit être transposée en Grafcet afin de pouvoir être implantée sur les organes réels de commande de l'installation. Cette transposition, point de passage obligatoire dans une méthodologie de conception utilisant les réseaux de Petri à la base, doit être assistée, sinon automatisée, le plus possible afin d'en minimiser les erreurs. C'est dans ce sens que nous proposons, dans une première démarche, d'étudier la faisabilité de l'automatisation de la transposition. Cette démarche constituera ultérieurement l'une des étapes du projet C. A. S. P. A. I. M. relevant de la phase d'implantation automatique.

I.4.2 - Transposition du modèle de base

Les graphes de processus, tels qu'ils ont été présentés au § I.2.2, définissent la structure de base du modèle retenu. Ils sont par construction saufs, c'est-à-dire mono-marqués. Au sein d'un processus, ne circule qu'une unique marque, chaque place étant ainsi 1-bornée. Cette propriété fondamentale est nécessaire pour établir de façon simple la

correspondance avec le modèle Grafcet. En effet, une étape Grafcet ne possède au plus, par définition, qu'un seul jeton. L'exemple suivant montre la différence de comportement entre les deux modèles dans le cas de non respect de la propriété sauf pour les réseaux de Petri.



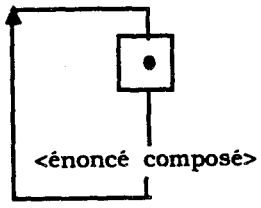
Différence de comportement entre RdP et Grafcet

FIGURE 4

Cette situation ne peut se produire avec les réseaux de Petri retenus et présentés sous forme de graphe de processus ; la transposition d'un modèle à l'autre peut donc à ce niveau se limiter à un simple changement des normes graphiques utilisées. La syntaxe dans le cas du modèle Grafcet devient donc la suivante :

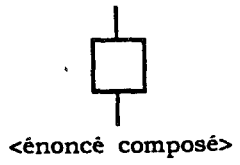
PROCESSUS

<processus> ::=



ENONCE COMPOSE

<énoncé composé> ::= <énoncé simple> / <énoncé simple>

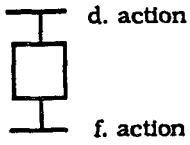


ENONCE SIMPLE

<énoncé simple> ::= <action> / <alternative> / <répétitive>

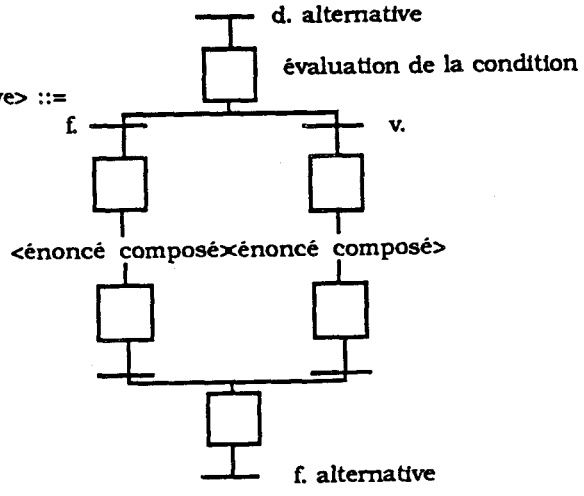
ACTION

<action> ::=



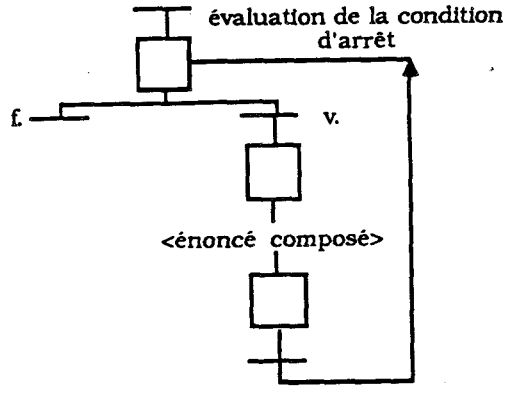
ALTERNATIVE

<alternative> ::=



REPETITIVE

<répétitive> ::=

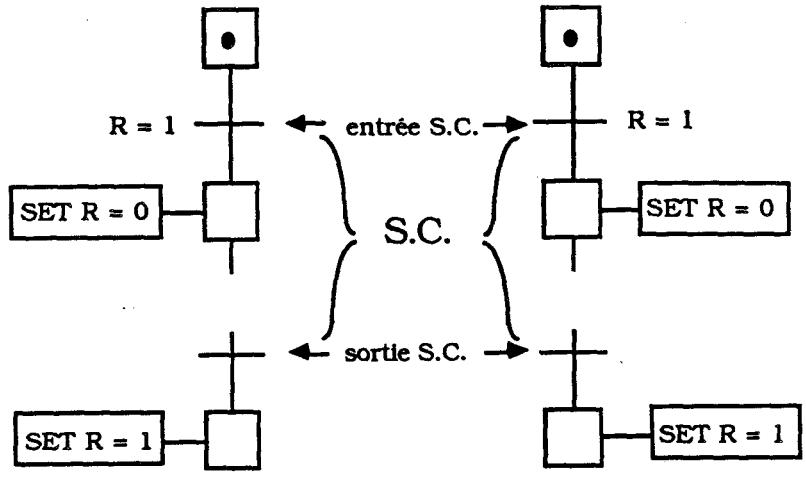


De façon identique, les extensions aux modèles que sont l'alternative et la répétitive multiples sont transposées en Grafcet en conservant rigoureusement la même ossature graphique.

Nous présenterons en détail au Chapitre II, la réalisation effective des primitives de liaison inter-processus. Le symbolisme utilisé en RdP les représentant sous la forme de places connectées aux différents graphes de processus ne peut être retenu avec le modèle Grafcet. Deux raisons essentielles nous amène à ce constat. Les places de liaison ne sont pas obligatoirement mono-marquées dans le cas d'une ressource ou d'un producteur/consommateur. La modélisation fait abstraction des problèmes liés à une implantation répartie des graphes de commande sur des organes physiques distincts.

Néanmoins, dans l'hypothèse d'une implantation centralisée sur un unique organe de commande, la transposition du Grafcet utilisera des variables internes (bits ou mots) à la place d'étapes pour la définition des primitives de liaison. La gestion de ces dernières va dépendre étroitement du degré de parallélisme vrai des automates programmables. Le cycle de fonctionnement couramment employé amène le scheduler d'un automate à évaluer et à déclencher en parallèle les transitions tandis que les actions associées aux étapes actives sont évaluées en séquence et de façon indivisible. C'est cette indivisibilité qui va garantir l'accès exclusif à la primitive de liaison.

L'exemple suivant correspond à une mise en œuvre erronée d'une ressource exclusive en prenant en compte l'hypothèse d'une évaluation en parallèle des réceptivités des transitions.

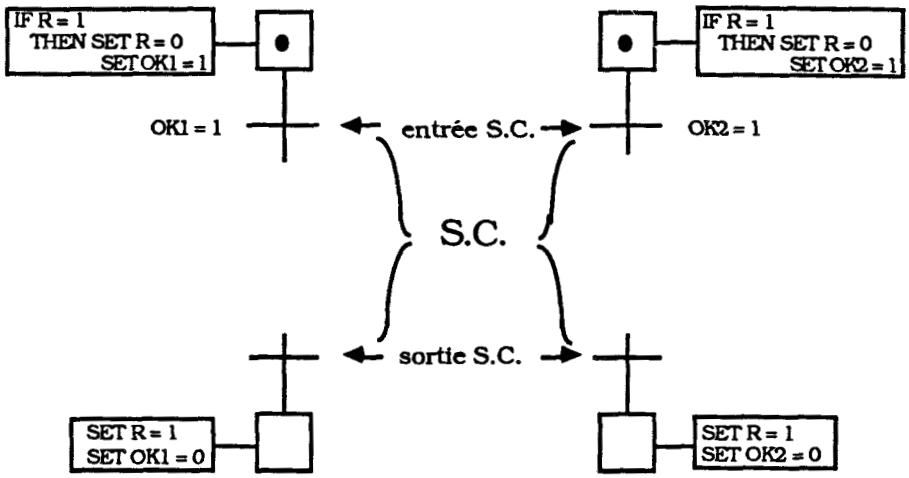


Gestion erronée d'une ressource

FIGURE 5

Le cycle du scheduler est tel que les deux réceptivités (R = 1) vont être évaluées en parallèle. Comme la ressource est libre, les deux transitions sont déclenchables et seront effectivement déclenchées, ce qui aura pour effet d'amener simultanément les deux process dans leur section critique respective. Seules les actions associées aux étapes actives sont évaluées en séquence et de façon indivisible. L'automate effectuera donc, après le tir des transitions, une première fois l'action SET R = Ø puis après une seconde fois cette même action pour le second processus concerné. Cette réalisation ne peut donc servir, en cas de conflit, à protéger une ressource critique. La protection d'une ressource critique ne peut être envisagée au niveau des transitions, comme cela est généralement le cas en réseaux de Petri, mais doit être abordée au niveau des étapes pour une réalisation en Grafcet sur des automates industriels.

Ainsi, la solution correcte pour cet exemple est la suivante :

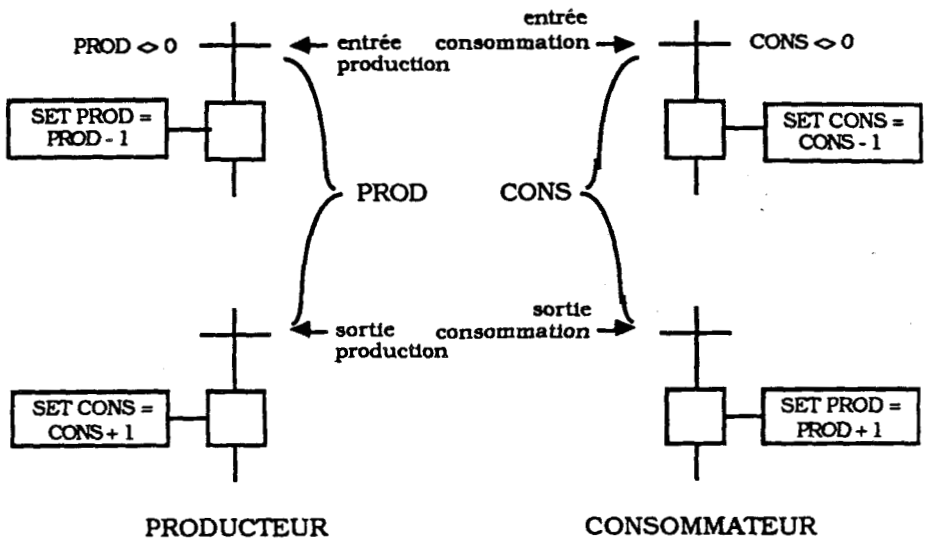


Gestion correcte d'une ressource

FIGURE 6

Les actions associées aux étapes étant évaluées en séquence, seul le processus dont l'étape est la première à être scrutée, sera à même de consulter la variable R dont la valeur traduit la disponibilité de la ressource. La mise à zéro immédiate de cette variable empêchera tout autre processus de pénétrer également dans la section critique. Sachant que les étapes sont généralement scrutées par numéro d'ordre croissant, une priorité implicite est donc donnée dans cette exemple, en cas de conflit, au processus dont le numéro d'étape est le plus petit.

De façon analogue, un producteur / consommateur sera modélisé, dans le cadre d'une implantation centralisée sur un seul automate, avec deux variables entières caractérisant le marquage des places PROD et CONS.



Réalisation d'un producteur / consommateur

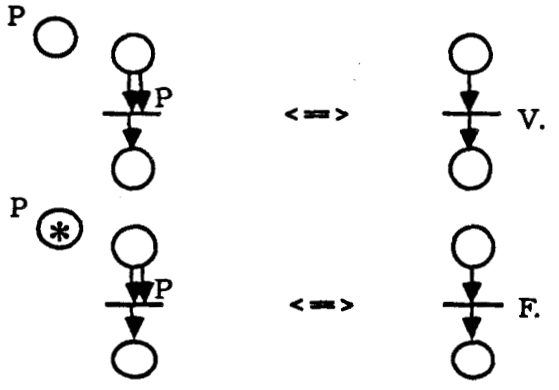
FIGURE 7

L'accès simultané aux variables PROD et CONS est interdit toujours par l'indivisibilité des actions au sein d'une même étape. Les solutions présentées ici n'ont de sens que dans le cadre d'une implantation centralisée. Leur simplicité de mise en œuvre réside dans l'utilisation de variables accessibles en lecture / écriture par les différents process et dans l'existence d'un sémaphore exclusif implicite, le processeur de l'automate, pendant l'évaluation des actions d'une étape.

I.4.3 - Transposition des réseaux adaptatifs

Formellement, les réseaux de Petri adaptatifs imposent que le marquage courant de la place dont l'arc est étiqueté soit supérieur ou égal au marquage de la place étiquetant cet arc. Pratiquement, ils ne sont utilisés que pour empêcher l'évolution d'un graphe dont les réceptivités seraient validées. En effet, comme les graphes de processus sont par construction saufs, le marquage courant d'une place est en tout état de cause au plus égal à 1. L'ajout d'un arc adaptatif est donc équivalent à la définition d'une variable booléenne paramétrant, par un "et" logique avec les autres conditions, la réceptivité d'une transition. Si la place pondérant l'arc n'est pas marquée, la variable booléenne est alors dans l'état vrai et la transition est franchissable ; si la place est marquée, la variable est dans l'état faux et la

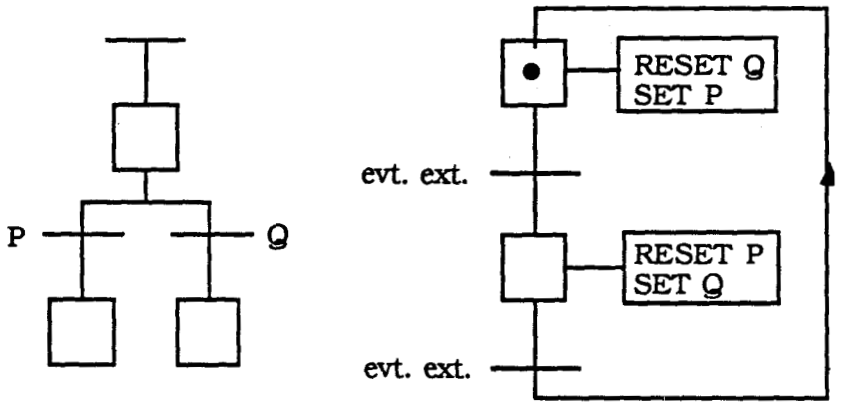
transition est bloquée (Fig. 8).



Equivalence pour des process saufs

FIGURE 8

La transposition du modèle en Grafcet implantable sur des automates reprend cette constatation. Deux solutions sont offertes à l'utilisateur dépendant des fonctionnalités du langage de programmation. La première solution, plus restrictive, suppose que l'utilisateur ne peut accéder au marquage de son Grafcet et qu'il ne dispose donc pas de variable booléenne traduisant l'activité de ses étapes. Face à ce constat, il devra gérer explicitement la valeur des booléens conditionnant les transitions. Ainsi, l'exemple représenté Figure 2 sera traduit de la façon suivante :



Gestion explicite des réceptivités

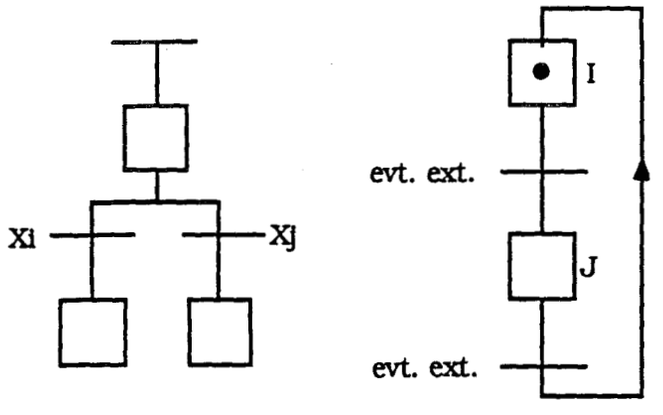
FIGURE 9

En fonction du marquage du graphe traduisant le mode de marche retenu, l'une des

deux transitions peut être franchie alors que l'autre est gelée. Deux variables p et q ont été utilisées pour correspondre étroitement à la description en RdPSA ; une seule est bien sûre suffisante dans le cas présent, leur état respectif étant complémentaire.

La seconde solution suppose que l'utilisateur ait à sa disposition une table de données caractérisant le marquage courant de son Grafcet. Cette opportunité est par exemple disponible avec le langage de programmation PL7-3 des automates de la série 7 de la Télémécanique. A chaque étape du Grafcet est associé un identificateur unique, un entier dans le cas présent. Le système d'exploitation de l'automate maintient en permanence une table traduisant l'activité de ses étapes ; cette table est accessible par l'intermédiaire de variables booléennes référençant de façon rigoureuse les étapes. Ainsi avec le langage PL7-3, les variables réservées Xi , où i désigne le numéro d'une étape, caractérisent l'activité du Grafcet. L'utilisateur n'a plus alors à gérer explicitement l'évolution du marquage pour traduire de façon cohérente le concept d'arcs adaptatifs.

L'exemple précédent est cette fois implanté dans l'esprit du modèle adaptatif en reportant au niveau des transitions sous forme d'une condition associée au prédicat de marquage d'une étape, le paramétrage du poids d'un arc (0,1).



Transposition rigoureuse du modèle RdPA

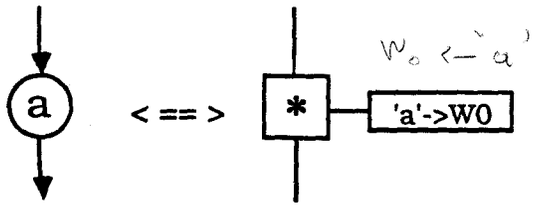
FIGURE 10

Les indicateurs i et j sont les numéros des étapes symbolisant le mode de marche retenu. La variable Xi est automatiquement à l'état vrai car l'étape i est marquée, tandis que la variable Xj est fausse. La seule différence entre la modélisation et l'implantation

réside dans la localisation des conditions régissant l'évolution des modèles. Là où les RdPA utilisent des arcs pondérés pour modifier dynamiquement leur propre condition de franchissement, l'implantation en Grafcet reporte cette modélisation au niveau de ses transitions. Les deux modélisations sont donc équivalentes dans le cas d'une utilisation des RdPA saufs.

I.4.4 - Transposition de la coloration

Les réseaux de Petri colorés augmentent le pouvoir de modélisation de la commande en reportant au niveau du jeton circulant au sein des places une information supplémentaire traduite par la couleur. Le nombre d'informations véhiculées par le marquage et symbolisées par la couleur est, par définition, fini [BOU 88]. Le modèle Grafcet, plus restrictif, ne permet d'associer à la marque aucune information supplémentaire. Sa présence traduit simplement l'état d'avancement du processus dans son cycle et active les actions associées à l'étape couramment marquée. La couleur ne pouvant être affectée directement à la marque, l'implantation devra utiliser des variables internes aux automates où seront stockées les informations initialement modélisées au niveau des jetons. Les graphes de processus étant saufs par construction, une seule variable par processus est suffisante pour réaliser le codage de la coloration en Grafcet.



Coloration en Grafcet

FIGURE 11

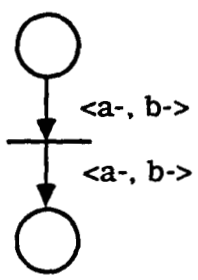
La couleur a n'est plus directement associée au jeton mais est stockée dans la variable WØ interne à l'automate dès que l'étape correspondant à cette action est activée.

Les règles d'évolution des RdPC sont déterminées par la présence d'étiquettes au niveau des arcs indiquant le chemin admissible en fonction de la couleur du jeton. Comme pour les arcs adaptatifs, l'implantation réalisera les tests de couleur au niveau des transitions alors que la modélisation initiale les intègre au niveau des arcs.

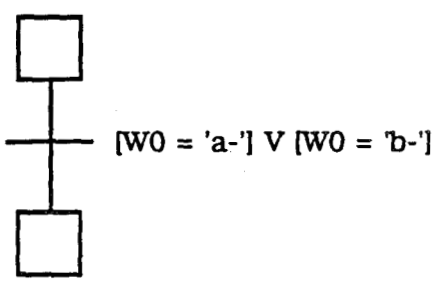
Définissons par exemple un processus autorisant l'usinage de deux pièces de nature différente :

- une pièce de couleur a^- devra être tournée,
- une pièce de couleur b^- dont les caractèristiques géométriques sont différentes de la précédente nécessitera un programme d'usinage spécifique sur le même tour.

Schématiquement, la modélisation en RdPC intégrant la coloration afin de valider ou non l'évolution du graphe est la suivante :

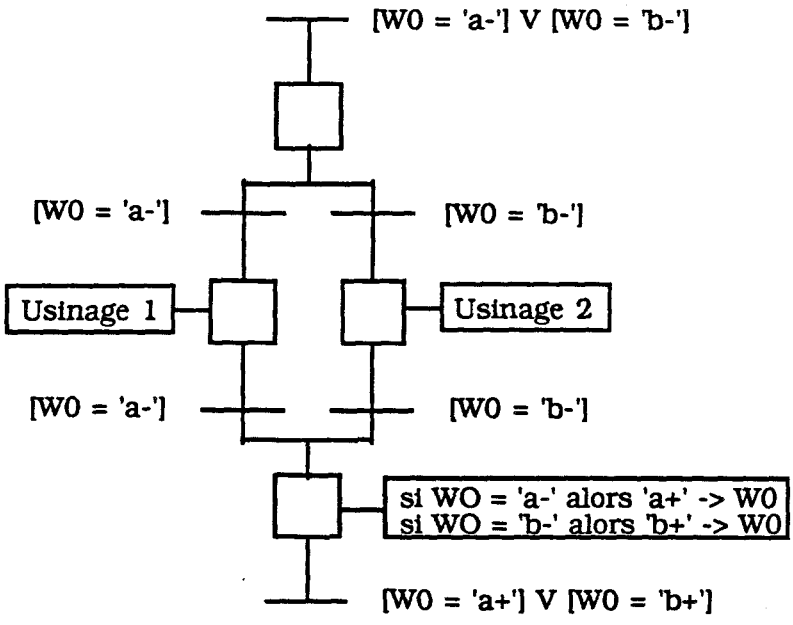


Les ensembles de couleurs admissibles $< a^-, b^- >$ étiquettent les arcs PRE et POST du modèle afin d'empêcher toute évolution d'un jeton dont la couleur n'appartiendrait pas au domaine $< a^-, b^- >$. L'implantation en Grafcet reprend ces tests au niveau des conditions paramétrant la réceptivité de la transition.



La couleur de la marque véhiculée dans ce processus est contenue dans la variable WØ. De façon analogue à la modélisation en RdPC, sa non-appartenance au domaine

$< a^-, b^- >$ aura pour effet de geler l'évolution du Grafcet. L'exemple présenté Figure 3 peut ainsi être transposé **systématiquement** en évitant l'introduction de nouvelles erreurs entre la phase de modélisation et la phase d'implantation.



Exemple de coloration en Grafcet

FIGURE 12

Les tests déterminant les domaines de couleurs admissibles pour un marquage et initialement modélisés au niveau des arcs PRE sont reportés au niveau des transitions. Par contre, les étiquettes sur les arcs POST indiquant la nouvelle couleur de la marque, sont traduites par des actions au niveau de la place en amont de ces arcs POST. En effet, elles ne modélisent pas des tests mais des règles de ré-écriture symbolisant une transformation de l'entité véhiculée. Ces règles de ré-écriture sont des actions élémentaires logiquement implémentées au niveau des étapes dans une spécification en Grafcet. Dans l'exemple présent, les étiquettes des arcs POST n'ont pas toutes été traduites en action dans les places amonts. La plupart d'entre elles reprenaient simplement la couleur spécifiée au niveau de l'arc PRE et ne modélisaient donc pas une transformation de la couleur. Comme cette dernière est contenue dans une variable du processus, l'information codée dans le mot WØ reste dans ce cas valide et ne nécessite pas de mise à jour. Seules les étiquettes traduisant effectivement une transformation de la couleur doivent être explicitement spécifiées par programmation au niveau des actions à la place amont de l'arc. C'est ce qui a été

réalisé dans la dernière place de la Figure 12.

I.4.5 - Conclusion

De par leurs propriétés, les RdPSAC se prêtent aisément à une traduction en Grafcet. Le fait que les graphes de processus soient saufs par construction autorise une transcription systématique du modèle de base. Nous avons montré également comment, par l'utilisation de variables internes, les primitives de liaison standardisées étaient réalisées sous l'hypothèse d'une implantation centralisée sur un unique organe de commande.

Le Chapitre II présentera différentes solutions quant à la réalisation de ces primitives de liaison distribuées sur plusieurs organes de commande. Ces solutions seront complètement spécifiées en Grafcet, ou mettront en œuvre un niveau hiérarchique, organe centralisateur et coordinateur de la commande.

Les extensions apportées au modèle de base, les arcs adaptatifs et la coloration, sont également implémentables en Grafcet par simple transposition. Néanmoins, l'implantation ne parvient pas à conserver la concision du modèle RdPSAC. En effet, ces extensions ne faisant pas partie intégrante du modèle graphique Grafcet, elles doivent être gérées exclusivement par programmation. Les tests portant sur les arcs adaptatifs et sur les domaines de validité de la couleur doivent être intégrés au niveau des transitions, tandis que les actions modifiant la couleur d'une marque sont réalisées par logiciel au niveau des étapes.

Hormis cette remarque, il s'avère que le modèle RdPSAC peut être transcrit de façon rigoureuse et systématique en Grafcet. Une telle constatation permet d'envisager la définition et la réalisation d'un logiciel qui, à partir des spécifications du modèle initial RdPSAC et en fonction de la norme Grafcet NFC03190 enregistrée à l'AFNOR en novembre 82, effectuera automatiquement la transposition d'un modèle à l'autre. Il reste, à ce stade, à définir un post-processeur adaptant le résultat normalisé aux différents organes de commande disponibles sur le marché afin d'aboutir à un code directement implantable (problème analogue à celui de la C. F. A. O. qui délivre un programme de commande numérique

normalisé de type PROMO ou encore APT qu'il faut transférer sur un contrôleur spécifique comme une NUM 760). Cette transcription, rendue obligatoire par l'offre du marché qui ne propose pas d'automates acceptant une programmation par RdP, présente l'avantage d'être automatique et ne constitue donc pas une source d'erreurs supplémentaires dans une méthodologie de conception qui, à partir du cahier des charges, cherche à aboutir à la commande finale des organes de production.

I.5 - Conclusion

Le modèle RdPSAC présente un pouvoir de modélisation adaptable à la complexité du système de production étudié. La structuration banalise le parallélisme et les communications ; les arcs adaptatifs symbolisent les interférences de comportement entre sous-réseaux, autorisent le règlement des indéterminismes et permettent d'interfacer la description de la commande avec des modèles d'un autre niveau ; enfin, la coloration facilite l'agrégation du modèle en intégrant la représentation des classes d'objets circulant au sein de la production et en schématisent les transformations effectuées sur ces objets. Le modèle RdPSAC semble donc suffisant pour représenter correctement la partie commande d'un système flexible de production.

La seule lacune inhérente au modèle provient de l'impossibilité d'assurer le suivi unitaire des entités circulant dans le système de production. La coloration permet d'identifier un objet par rapport à sa classe (cette entité est de couleur <x> et appartient ainsi à la gamme qui doit subir les opérations r, s, t sur les machines m_i et m_j) ou traduit également l'état d'avancement d'un objet dans sa gamme opératoire. Cette information unique ne peut être utilisée pour identifier sélectivement des objets appartenant à la même classe, ni même pour associer à ces objets des attributs supplémentaires. Il peut cependant s'avèrer nécessaire lors de la définition de la commande d'un système de production de gérer un plus grand nombre d'informations ou même de pouvoir individualiser les objets du modèle.

Donnons un exemple d'illustration : soit un système au sein duquel circulent deux classes différentes d'objets se partageant les mêmes unités de fabrication. La coloration est utilisée pour distinguer les deux classes entre elles, ce qui permet d'adapter les

programmes d'usinage en fonction de l'entité présentée et également pour traduire le nombre d'opérations déjà subies par l'objet. Supposons que les spécifications du cahier des charges précisent que les objets arrivent par lots et que ces lots sont contraints de respecter une date limite au-delà de laquelle la fabrication du lot doit obligatoirement être achevée. La coloration, par interprétation, ne peut qu'imparfaitement intégrer cette information supplémentaire. On peut très bien associer à chaque classe d'objets une "date due" spécifique, ce qui va permettre d'agencer dynamiquement la production en privilégiant une classe par rapport à l'autre en fonction de la date. Mais que faire si plusieurs lots appartenant à la même classe et possédant des dates différentes doivent circuler dans le système ? La couleur, information unitaire, ne pourra dans ce cas autoriser la codification des dates.

L'exemple présent nous montre, fonction de la complexité du système étudié, que les RdPSAC peuvent se révéler insuffisamment puissants pour modéliser tous les problèmes rencontrés dans la commande d'une production flexible. Les réseaux de Petri à prédicats / transitions [BRA 83] qui associent au jeton, non plus un identificateur unique comme la coloration, mais un n-uples de valeurs et dont la réceptivité des transitions est calculée par une fonction de ce n-uples de valeurs, apportent une solution satisfaisante à ce problème. Le premier paramètre du n-uples peut très bien être la couleur au sens du modèle RdPSAC afin d'en conserver la philosophie, tandis que les paramètres supplémentaires seraient déterminés suivant la nécessité en fonction des spécifications du cahier des charges. Par contre, il ne semble pas souhaitable d'intégrer toute cette modélisation exclusivement au niveau du modèle de la commande. La définition des prédicats et surtout la description des fonctions paramétrant les transitions alourdiraient considérablement le modèle de base dont la fonction principale est exclusivement la détermination des séquences d'opérations mettant en œuvre les différentes gammes opératoires. Le modèle de base a en effet pour objectif de mettre en avant le parallélisme et la flexibilité du procédé sans pour autant qu'il soit de son ressort d'intégrer toutes les stratégies de fonctionnement. L'utilisation d'un niveau hiérarchique distinct de la partie commande [VAL 88] [MAR 88] [BAR 88] est donc essentielle pour la définition des mécanismes de prises de décision. Grâce à ce niveau hiérarchique, la commande est dispensée de la modélisation du n-uples de valeurs. A un jeton est associé un identificateur unique (concept différent de celui de la coloration car l'identificateur peut cette fois être pris dans un ensemble infini de valeurs) alors que les

réceptivités des transitions ne sont plus seulement validées par l'état de la commande mais également par décision émanant du niveau hiérarchique.

La transposition des RdP à prédicats / transitions en Grafcet reprend largement les méthodes présentées pour intégrer la coloration et les arcs adaptatifs. L'identificateur unique associé au jeton est reporté sur une variable interne qui sera étroitement liée à l'évolution du Grafcet. Il n'est pas directement exploitable mais sert d'index pour accéder à une table de données où sont effectivement rangés les n-uples de valeurs qui caractérisent complètement le status du jeton.

Les réceptivités des transitions utilisant les n-uples de valeurs pour calculer leur état sont quant à elles pondérées par de simples booléens. Les valeurs de ces derniers seront déterminés par le niveau hiérarchique qui possède la description complète des fonctions conditionnant les transitions et dont les paramètres sont les n-uplets associés au marquage. Ce niveau hiérarchique qui intègre une base de données et un ensemble de fonctions complexes peut être directement implanté sur l'automate où réside le Grafcet. Il faut pour cela que l'automate soit suffisamment puissant pour accepter la description des données ainsi que la programmation des fonctions régissant la réceptivité des transitions. Si les performances de l'organe de commande sont trop limitées ou si la mise en œuvre d'une stratégie de fonctionnement nécessite une connaissance globale qui ne soit pas limitée au seul automate et à ses propres Grafcets, le niveau hiérarchique sera alors installé sur un calculateur dissocié de la commande directe mais ayant une visibilité effective de l'ensemble de l'installation. Ce niveau hiérarchique régulera le fonctionnement des Grafcets comme précédemment par l'intermédiaire des variables booléennes liées aux transitions des graphes.

L'utilisation des RdPSAC et éventuellement l'emploi des réseaux à prédicats / transitions conviennent parfaitement à la modélisation de la commande de processus discontinus dans l'industrie manufacturière. Leur transposition, imposée par l'offre du marché, en Grafcet est pour une grande part automatique et ne devra donc pas être une source d'erreurs supplémentaires dans une démarche générale de conception. L'apport d'un niveau hiérarchique complémentaire de la commande permet avantageusement d'accroître la puissance de modélisation sans complexifier inutilement la description de

base de la commande. Nous montrerons par la suite, principalement au Chapitre II, l'importance de ce niveau hiérarchique tant pour la définition d'un mode de fonctionnement normal et flexible que pour la gestion des différents modes de marche.

II - PRESENTATION DU PROJET C. A. S. P. A. I. M.

II.1 - Introduction

Le projet C. A. S. P. A. I. M. dans lequel s'intègre notre travail tente de définir une méthodologie générale de conception et de réalisation de la commande au sens large (commande directe, supervision, ...) de cellules flexibles de production. Cette démarche partant du cahier des charges comme base de travail va formaliser tout au long de son déroulement les données initiales pour aboutir à un code implantable sur les différents organes de commande en prenant en compte la nature réelle des procédés pilotés ainsi que le concept de distribution de la commande sur différents organes. Une telle méthodologie présente de nombreux avantages. Elle permet en premier lieu de gagner un temps important dans la phase de conception et de mise au point du système de commande. Elle diminue le nombre d'erreurs tant dans la phase de spécification que dans les étapes de synthèse du système de commande. En effet, l'automatisation d'une démarche par un ensemble de programmes informatiques exempts d'erreurs (si cela est possible !) assure naturellement un résultat final cohérent et fiable. Cette constatation renforce d'autant l'économie de temps consacré au projet. Enfin, elle aura pour résultat de standardiser la réalisation ce qui facilite la maintenance ultérieure et a pour conséquence d'améliorer la documentation.

Dans cette partie, nous présenterons les grandes lignes du projet C. A. S. P. A. I. M. pour en dégager les points principaux. Cette méthodologie de conception permet en particulier d'aboutir à un modèle RdPSAC décrivant un modèle de la commande à partir duquel nous proposons dans ce mémoire d'étudier la transposition en un modèle d'implantation. C'est à partir des résultats de cette analyse que nous nous sommes attachés à définir et à automatiser la transposition réelle en Grafcet et que nous avons réalisé un niveau hiérarchique d'implantation interférant directement sur les graphes des automates.

II.2 - La démarche

Le schéma directeur du projet C. A. S. P. A. I. M. se décompose en quatre phases fondamentales (Fig. 13) :

- une phase d'analyse et de spécification du cahier des charges (Phase I),
- une phase de modélisation (Phase II),
- une phase de simulation (Phase III),
- une phase d'implantation visant à mettre en œuvre le résultat de la modélisation après validation par le simulateur (Phase IV).

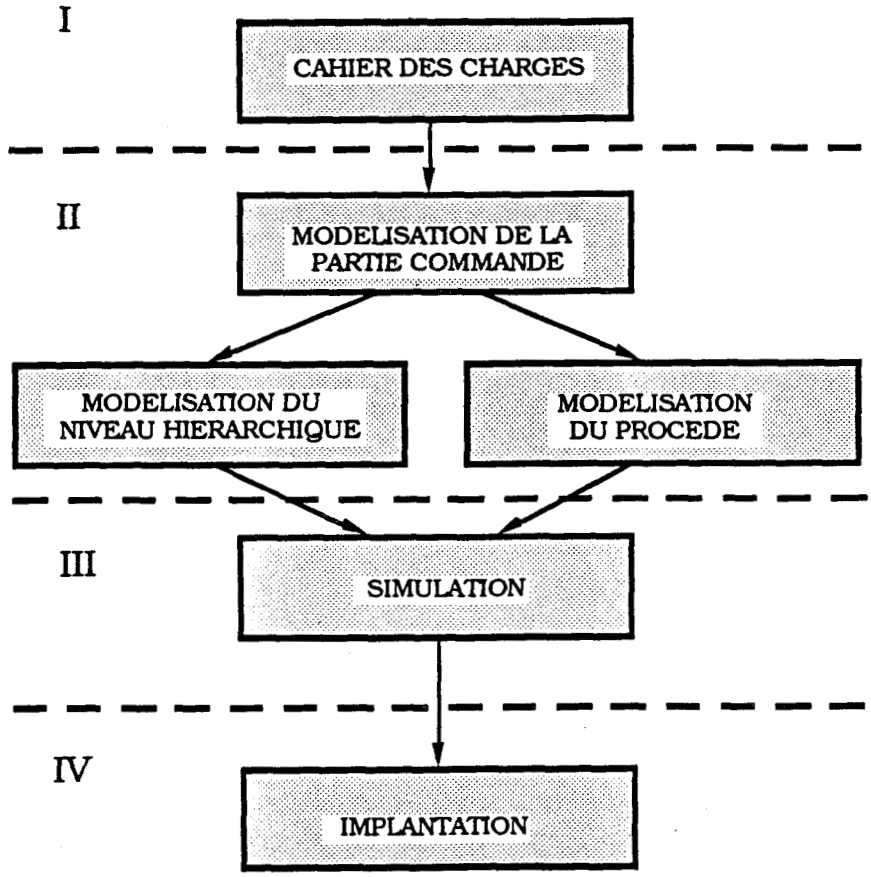


Schéma directeur

FIGURE 13

L'ordre induit dans cette présentation ne reflète pas exactement la réalité. La simulation complète du système est bien sûr menée après l'obtention des différents modèles. Elle permet de valider, qualitativement et quantitativement, le fonctionnement global de l'installation, de déterminer d'éventuelles erreurs de conception et, par temporisation, d'avoir des statistiques relativement fines concernant les performances de production. Néanmoins, une première simulation peut être effectuée sur un modèle préliminaire, juste après la

phase de spécification. Elle aura pour effet de donner de premiers résultats rapides concernant certaines particularités de fonctionnement du système. Notamment, les blocages caractérisant un engorgement du procédé en ses points critiques (zones intermédiaires de stockage de capacité finie, tampons d'entrées / sorties des machines, ...) seront détectés beaucoup plus facilement que par une simulation portant sur le modèle développé de la commande.

Le schéma directeur détaillé du projet C. A. S. P. A. I. M. reprend ces quatre phases en y développant les principales étapes (Fig. 14). Cette formalisation n'est ni limitative, ni exhaustive. Certains aspects de la méthodologie peuvent être éclatés de façon plus fine encore pour insister sur l'aspect directif et rigoureux de la démarche. D'un autre côté, fonction de l'avancement des travaux du laboratoire, d'autres étapes restent sujettes à caution concernant leur définition et pourront sensiblement évoluer après une formalisation stricte des solutions retenues.

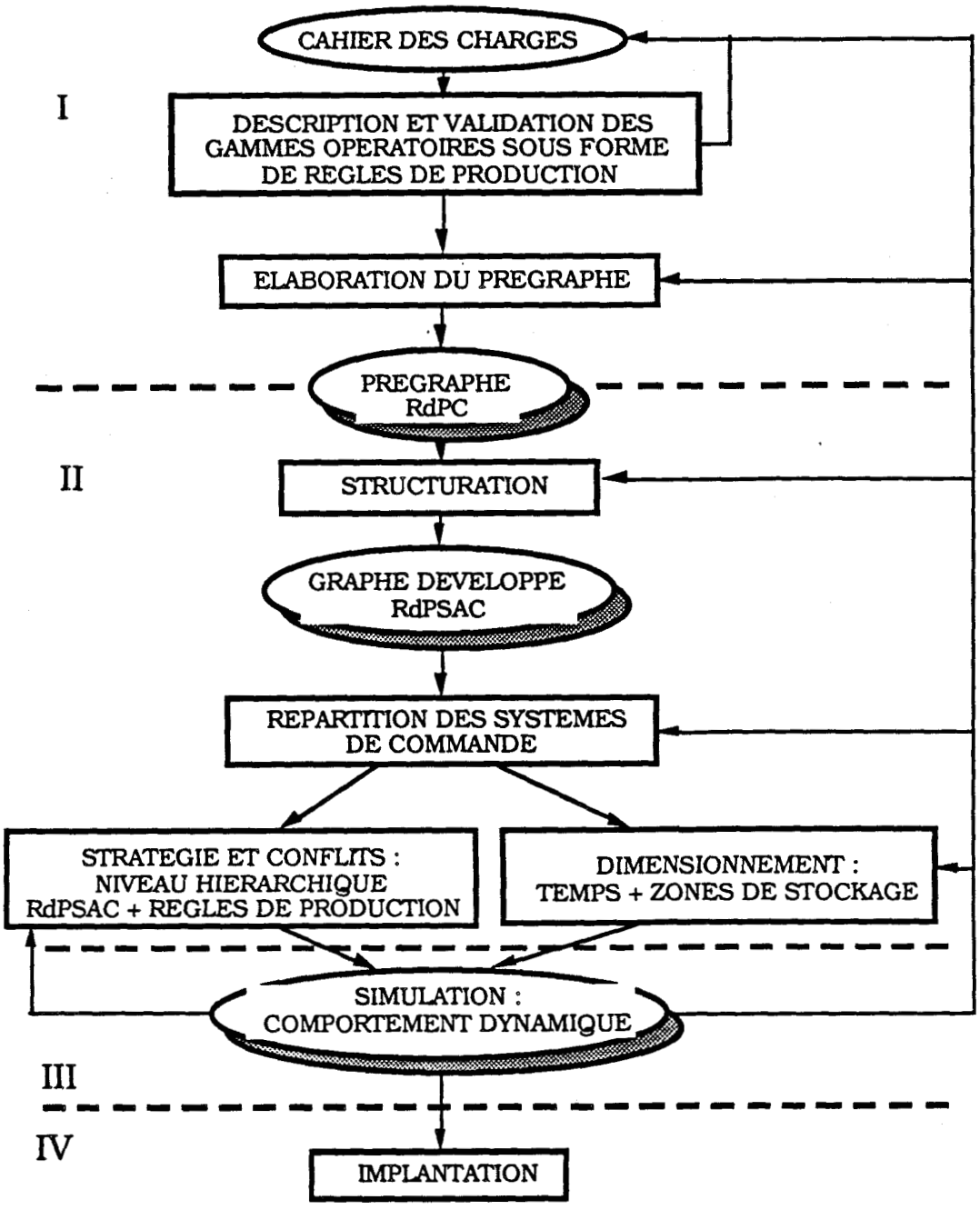


Schéma directeur détaillé

FIGURE 14

La première phase vise à obtenir à partir du cahier des charges, une description formelle des gammes opératoires [KAP 87] [KAP 88]. Cette description, par une analyse de cohérence et de complétude, va permettre de s'assurer que la formalisation respecte bien les impératifs spécifiés au niveau du cahier des charges et concernant les gammes opératoires à mettre en œuvre. Concrètement, l'analyseur délivre une description des gammes

sous la forme banalisée et simple de règles de production de type prémisses → conséquents ; chaque prémisse ou conséquent étant constitués d'un triplet (présence pièce lieu). Son utilisation s'effectue suivant deux modes principaux : la mise au point, la récapitulation.

La mise au point assiste le raisonnement de l'opérateur dans la constitution étape après étape des règles de production décrivant les gammes opératoires d'une unité de production. Elle permet d'éviter les oublis, les lacunes de description et assure l'enchaînement correct des règles de production ainsi que le lien entre un ensemble de faits initiaux et de faits terminaux eux-mêmes éventuellement déterminés progressivement.

La récapitulation permet à l'utilisateur de revoir l'enchaînement des règles et d'apporter si nécessaire certaines modifications. Après l'analyse, une étape de modulation vise à enrichir la description de places média intermédiaires (lieux de stockage, d'assemblage, ...). Elle permet également la duplication, le retrait de places (machines, sites) pour aboutir, par traduction, à un prégraphe en RdP coloré. Le prégraphe a pour fonction de modéliser la structure de base agrégée du graphe de commande et de transfert de l'unité de production. C'est le point d'entrée de la seconde phase.

Un développement structuré [BOU 87] (au sens du modèle RdPSAC présenté dans la première partie de ce chapitre) est mené à partir du prégraphe.

L'originalité de la démarche réside dans le fait que, contrairement aux méthodes de modélisation qui consistent à agréger un graphe par l'utilisation d'un modèle de haut niveau (réseaux de Petri stochastiques [BAL 87] ou à prédicats [DES 85], l'objectif est d'aboutir à partir du modèle initial prégraphe, à un modèle développé et structuré. Ceci permet de mettre en évidence le parallélisme du système ainsi que la concurrence entre les tâches élémentaires. La structuration du modèle facilite le suivi du cheminement des entités au sein du système, elle minimise le nombre de primitives employées dans la description du fonctionnement et autorise de la sorte une traduction aisée en code implantable (cf. première partie de chapitre). La standardisation qui en résulte facilite, par la banalisation des structures employées, les modifications a posteriori (ajout des modes dégradés, par exemple) tout autant que la maintenance logicielle. Cette opportunité

contribue largement à la validité de la méthodologie car un système de production n'est jamais figé mais sujet à des modifications en cours d'exploitation.

Le développement structuré consiste en une construction modulaire du modèle RdPSAC de la partie commande par processus liés entre eux par des graphes de liaison. Cette étape se double d'un enrichissement informationnel interactif concernant la nature du procédé : type de machines (usinage, assemblage, ...), nombre et structure des tampons d'E/S, ... Elle aboutit dans un premier temps à un graphe développé exprimé sous forme de RdPSAC et à parallélisme maximal. La prise en compte des contraintes fonctionnelles telles les exclusions mutuelles, les synchronisations, limite les parallélismes et autorise une détection systématique de tous les conflits potentiels. Faisant suite au développement, la répartition des systèmes de commande, en fonction de leurs spécificités propres éclate encore la description tout en la rendant hétérogène. Le modèle RdPSAC qui intègre le parallèlisme, décrit le séquencement naturel des actions élémentaires. La flexibilité, apportée par les arcs adaptatifs, le rend non-autonome. A ce stade sont donc définies les heuristiques qui paramètrent dynamiquement le modèle de la commande. Ces règles de décision, exprimées sous la forme de règles de production, constituent le modèle du niveau hiérarchique, seul à même d'intégrer les différentes stratégies de fonctionnement ainsi que la résolution des indéterminismes ou conflits. Parallèlement, un modèle spécifique du procédé est créé utilisant pour cela un langage objet. A ce niveau, les actions de la partie commande sont prises en compte par temporisation ; l'interprétation des transitions est également possible.

La troisième phase récupère ces différents modèles :

- partie commande,
- niveau hiérarchique,
- procédé,

pour en effectuer plusieurs simulations [CAS 87] [CAS 88]. Elles ont pour objectif de valider les solutions retenues dans les précédentes étapes de la démarche. A ce stade, deux études sont nécessaires : la validation qualitative et la validation quantitative. L'aspect qualitatif vérifie que le fonctionnement dynamique de l'installation ne comporte aucune aberration dans son évolution. Il consiste ainsi à pouvoir détecter, analyser et résoudre en particulier les blocages et indéterminismes induits par une mauvaise conception ou une

définition insuffisante de la commande. La validation quantitative relève du dimensionnement du procédé (tampons d'entrées / sorties, nombre de machines, taux d'engagement de ses machines). Elle procède par temporisation et vérifie que les performances globales de la production rentrent dans les normes établies au niveau du cahier des charges.

La configuration modulaire du simulateur permet en effet de ne prendre en compte que la partie commande, la commande et le niveau hiérarchique ou le procédé ou bien les trois entités simultanément, pour une validation globale. La simulation peut être temporisée ; le choix des séquences d'entrées (aléatoires, fixes, au plus tôt, ...) est laissé au libre arbitre de l'utilisateur. Les données statistiques recueillies permettent après interprétation, un retour arrière dans les deux précédentes phases de modélisation. Le niveau hiérarchique est remis en cause si les stratégies précédemment définies se révèlent peu performantes tandis que la partie opérative et en particulier le dimensionnement des zones tampons sont revus. La répartition des systèmes de commande (ajouts ou retraits de ressources, synchronisations inadéquates entre deux processus) ou même la structuration par modification du type de machines utilisées sont également conditionnées par les résultats de la simulation. Induisant un coût nettement supérieur dans la démarche, des modifications au niveau de la Phase I pourront être apportées. L'élaboration du prégraphe pourra nécessiter la duplication de certains sites (étape de modulation) alors que la définition même des gammes opératoires sera remise en cause au niveau du cahier des charges si nécessaire. Lorsque les résultats de la simulation répondent à l'ensemble des critères imposés, les modèles obtenus sont retenus pour la dernière phase.

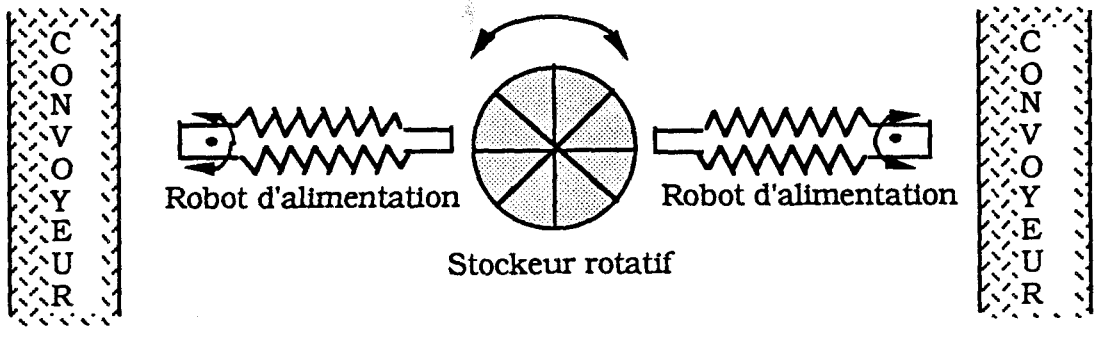
L'implantation, directement concernée par l'architecture et la nature des systèmes à piloter, reprend essentiellement les deux modèles de description (partie commande, niveau hiérarchique) afin de les rendre opérationnels sur site. La partie commande, dont le parallélisme fonctionnel maximal avait pour l'instant toujours été recherché, va être maintenant regroupée sur différents contrôleurs :

- automates programmables industriels (A. P. I.),
- armoires de commande numérique (C. N. C.).

Ces regroupements seront de deux natures : fonctionnels et économiques. Dans la mesure du possible, tous les processus participant à la gestion d'une même entité

physique, seront associés au sein d'une unique commande : c'est le regroupement fonctionnel.

Prenons l'exemple d'un stockeur rotatif en anneau dont le chargement et le déchargement sont réalisés par deux robots différents (Fig. 15). Les graphes de processus assurant la commande des deux robots ainsi que celui chargé du stockeur rotatif, sont décrits indépendamment les uns des autres mais synchronisés avec un très fort degré de connexité. Afin de simplifier la communication entre ces processus et d'améliorer les performances globales du système, ces graphes seront regroupés dans un unique automate qui assurera une communication interne avec des variables communes accessibles aux différents processus.



Structure du stockeur rotatif

FIGURE 15

L'aspect économique relève de deux concepts. Il est imposé par le nombre fini des commandes chargées du fonctionnement de l'installation. Des processus complètement indépendants se verront ainsi implantés sur un même automate pour la simple raison qu'il est le seul disponible. La localisation géographique de l'organe de commande vis à vis du procédé à piloter est le second concept rentrant en ligne de compte. La répartition des processus sera fonction de la proximité de la commande par rapport aux actionneurs. Il en résulte une économie significative dans le cablage à réaliser. Cette dernière contrainte sera atténuée quand les réseaux locaux de niveau 1 de type F. I. P. [THO 85] seront plus largement utilisés dans l'entreprise. Avec cette hypothèse, les actionneurs seront directement connectés sur le réseau et non plus par cablage avec l'automate chargé de les

commander. L'intelligence embarquée dans l'actionneur lui permettra de reconnaître les trames d'information qui lui sont adressées et d'agir en conséquence. Les automates communiqueront ainsi par l'intermédiaire du réseau avec les capteurs / actionneurs et ne nécessiteront plus d'être placés à proximité du procédé.

Une fois les organes cibles déterminés, le modèle RdPSAC doit être transposé en Grafcet (cf. § I.4). Nous avons vu précédemment dans les grandes lignes, comment la traduction est systématique et facilement automatisable. L'idéal serait bien sûr de pouvoir conserver dans son intégralité la description de base en RdPSAC. A défaut, il serait souhaitable que les automates soient plus ouverts et acceptent une programmation en code source Grafcet. Chaque station disposerait d'un compilateur propre pour générer son exécutable. Au niveau du projet C. A. S. P. A. I. M., la transposition se limiterait alors à l'écriture d'un post-processeur qui, ayant pour entrée le modèle RdPSAC, générerait le code source Grafcet qu'il suffirait ensuite de télécharger dans les automates. Ces fonctionnalités n'étant pas opérationnelles à l'heure actuelle, la programmation reste pour le moment exclusivement manuelle.

Enfin, parallèlement à l'implantation de la commande, la mise en œuvre du niveau hiérarchique comporte plusieurs possibilités. En fait, deux cas limites sont envisageables. Le niveau hiérarchique est, dans la première hypothèse, directement implanté sur les différents automates. Hormis la simplicité qui en résulte, cette solution présente de nombreux inconvénients. Les performances du niveau décisionnel sont nécessairement bridées par le calculateur hôte (en l'occurence, un A. P. I.). Sa programmation ne pourra se faire que dans un langage de bas niveau, au mieux un Basic dérivé. Mais surtout, son implantation, dans une configuration multi-automates, ne lui autorise que des prises de décisions locales limitées à la vision qu'il a de la commande de l'automate sur lequel il est implanté. La seconde hypothèse, revient à installer le niveau hiérarchique sur un calculateur externe connecté aux différents automates, par un réseau local. Sa délocalisation, par une vision globale de la commande, lui permet désormais des prises de décisions générales mettant en œuvre l'ensemble de la commande. Sa puissance de résolution n'est en rien limitée par les performances du calculateur qui dispose de langages de programmation de haut niveau. La principale difficulté réside dans ce cas, dans l'établissement d'un protocole de communication performant autorisant le niveau hiérarchique à accèder aux états internes

des différents automates (marquage courant des Grafcets, valeur des variables, ...). Ces problèmes seront largement abordés au Chapitre II ; nous y présenterons en particulier la solution que nous avons retenue, consistant à implanter le niveau hiérarchique sur un calculateur IBM et communiquant par un réseau local de niveau II avec les automates Télémécanique chargés de la commande.

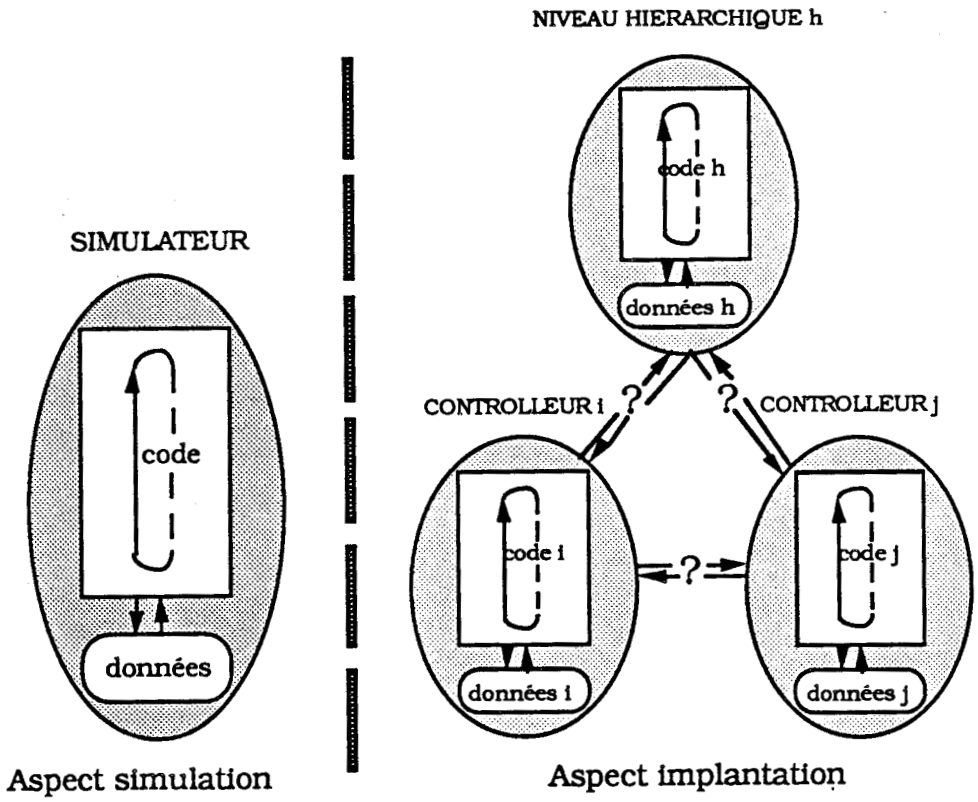
Le projet C. A. S. P. A. I. M. relève d'une démarche de conception générale mettant en œuvre des travaux complémentaires qui, rassemblés, participent à l'élaboration complète de la méthodologie. Face à la complexité croissante des systèmes de commande de production flexible, il est devenu nécessaire de réaliser, de façon assistée et automatique, la construction des modèles de la commande. A ce titre, le simulateur est un des maillons fondamentaux de la chaîne de conception assistée. Il est le seul à même de pouvoir mettre en évidence la validité de la démarche. Enfin, c'est la dernière phase logicielle off-line. Après validation de ses résultats, débute la phase d'implantation où la moindre modification a cette fois des répercussions temporelles et économiques largement préjudiciables à la viabilité du projet.

II.3 - Limites et extensions

Comme pour tout projet de longue haleine, innovant dans des secteurs de pointe, certaines lacunes demeurent ou même certains choix volontaires sont susceptibles d'être réenvisagés mettant en cause la philosophie du projet. Nous avons, dans la première partie de ce chapitre, soulevé les lacunes inhérentes au modèle RdPSAC. Retenir la solution des réseaux à prédicats / transitions augmenterait la puissance de modélisation et permettrait ainsi d'intégrer plus aisément des stratégies de production performantes. Nous allons maintenant détailler une limite de la simulation et montrer comment y remédier théoriquement et pratiquement.

Jusqu'à maintenant, la simulation a été abordée de façon globale et centralisée. Le simulateur gère une structure de données générale dans laquelle sont modélisées la description statique de la commande et du niveau hiérarchique et les données dynamiques caractérisant l'évolution du marquage des graphes. Ces informations sont manipulées par un programme unique ; en effet, le simulateur en assure l'évolution dans le temps. Cette

approche diffère de la réalité imposée par l'implantation sur deux points essentiels. Le code de la commande et du niveau hiérarchique est réparti sur différents contrôleurs ; son séquencement dans le temps est fondamentalement asynchrone et ne peut être reflété correctement par un programme unique centralisé. De façon analogue, la structure de donnée est éclatée. Aucun des organes de commande n'a de vision ensembliste générale de l'état du système à un instant donné t. Ces différences sont schématisées par la Figure 16. L'ensemble des codes i, j et h correspond globalement au code du simulateur. Par contre, leurs évolutions respectives asynchrones ne sont que très imparfaitement approchées par le cycle de fonctionnement du simulateur qui impose un "scheduling" rigoureux : évaluation du niveau hiérarchique puis, après seulement, évaluation de la commande. Les données soulèvent un autre problème. L'ensemble des informations réparties correspond bien à la structure centralisée au sens où toutes les informations sont effectivement représentées. Mais la centralisation masque complètement le problème de communication entre les contrôleurs afin qu'ils puissent échanger des données qui ne leur sont pas directement accessibles, sur leurs états respectifs. La cohérence de l'information entre la valeur d'une donnée émise vers un organe extérieur alors qu'elle est susceptible de continuer son évolution pendant la transmission du fait de l'asynchronisme des commandes, est également occultée par la simulation.



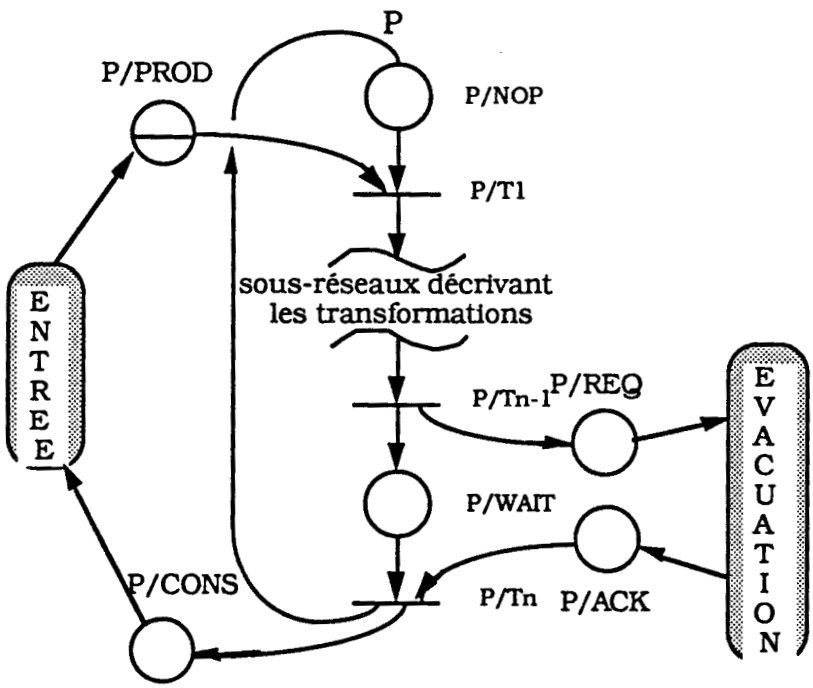
Différence simulation / implantation

FIGURE 16

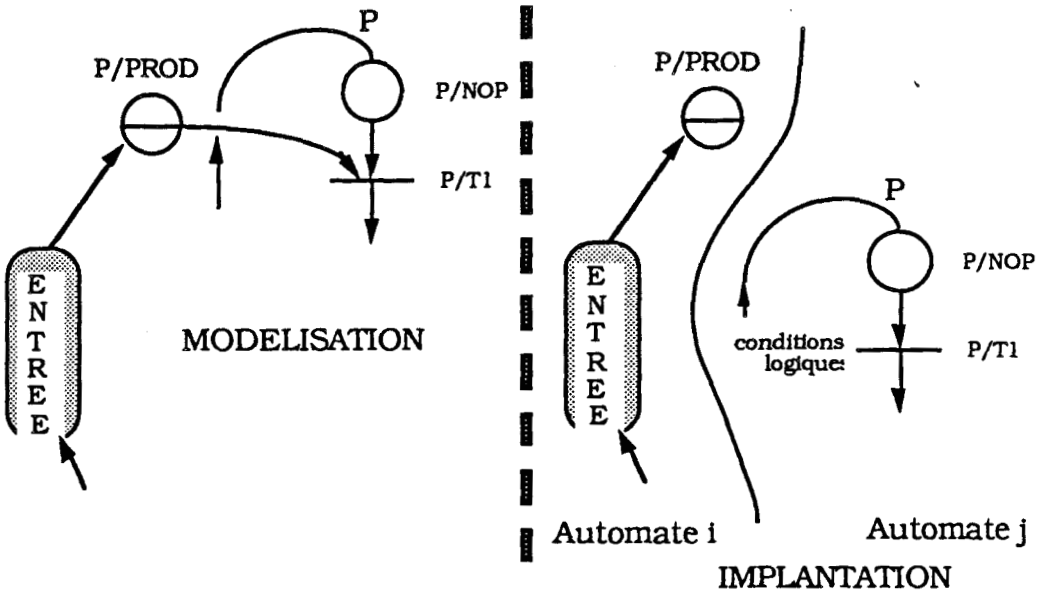
Pour bien insister sur les différences de comportement entre la simulation et l'implantation, nous allons présenter deux exemples mettant en avant pour le premier, les difficultés liées à l'asynchronisme, et pour le second, la répartition des données.

II.3.1 - Exemple 1

Un processus de transformation, dont l'ossature générale est la suivante :



est implanté sur un automate donné j. Il communique en entrée par l'intermédiaire d'un producteur / consommateur que l'on suppose modélisé sur un organe i, différent du précédent. L'arc reliant la place P/PROD à la transition P/T1 n'existe donc pas en réalité (cf. Fig. 17).



Asynchronisme

FIGURE 17

Lors de l'implantation, la programmation de la transition P/T1 doit explicitement prendre en compte le fait qu'elle n'est pas seulement conditionnée par le marquage de la place amont P/NOP. Les conditions logiques qui lui servent d'attribut sont validées par l'automate i en fonction de la présence d'une marque dans la place P/PROD. L'algorithme de décision permettant au processus P d'évoluer peut donc être le suivant :

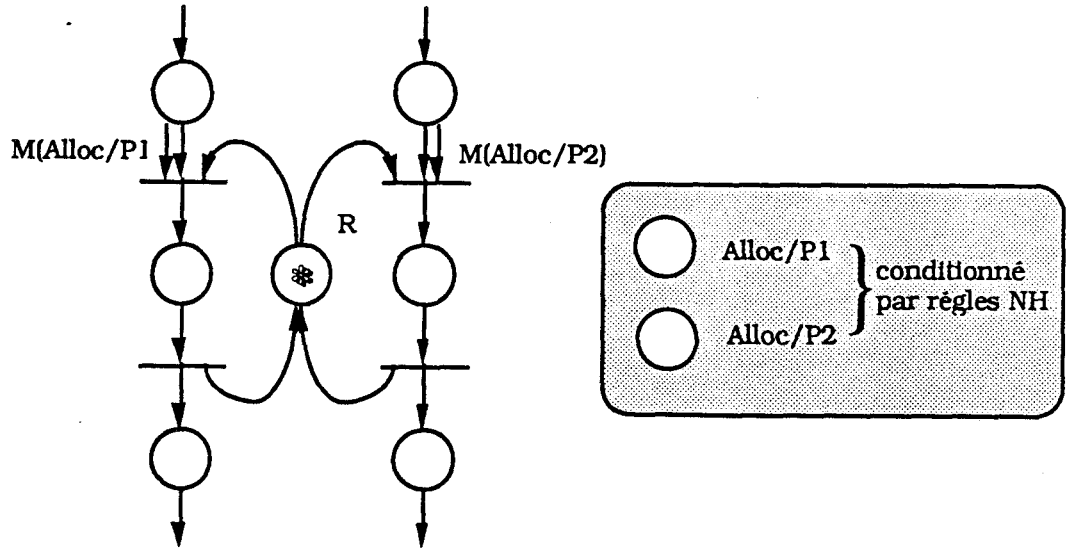
"si la place P/NOP est marquée" et "si la place P/PROD est marquée"
alors "décrémenter la place P/PROD d'une entité"
et "affecter à vrai les conditions logiques de la transition P/T1"

Cet algorithme est par hypothèse, implanté au niveau hiérarchique. En phase de simulation, le cycle est le suivant : on évalue le niveau hiérarchique, la règle concernée est donc déclenchée, puis on évalue les graphes de commande, la transition est franchissable et est ainsi effectivement tirée. Par contre, en implantation réelle, l'asynchronisme peut être tel que le niveau hiérarchique, si son cycle est nettement plus rapide que celui de l'automate, est susceptible de déclencher plusieurs fois la même règle avant que le marquage du processus P n'évolue. Cela aura pour effet erroné de décrémenter plusieurs fois la place P/PROD, alors que le processus P est resté figé. Les conditions logiques ne

permettent donc que de synchroniser qu'imparfaitement les processus. Pour la simulation, elles sont simplement inutiles. La boucle de simulation est telle que le niveau hiérarchique doit obligatoirement consulter l'état du marquage de la place P/NOP. Si l'on supprime ces attributs pour l'implantation et que l'on suppose cette fois que la commande évolue plus rapidement que le niveau hiérarchique, la transition P/T1 sera immanquablement franchie sans que le niveau hiérarchique ne s'en rende compte. L'effet contraire du précédent est ainsi obtenu : la place P/PROD n'est jamais décrémentée.

II.3.2 - Exemple 2

Deux processus se disputent l'accès exclusif à une ressource. La modélisation retenue dans la démarche de conception décide d'intégrer l'état de la ressource (libre ou occupé) au niveau de la commande, tandis que les règles d'attribution sont du ressort du niveau hiérarchique. Cela est traduit par le graphe suivant :



Partage de données

FIGURE 18

Le marquage des places Alloc/P1 et Alloc/P2 est conditionné par les règles décisionnelles. Cette modélisation se prête sans aucun problème à une simulation. La description de la commande réside dans une structure de données globale, unique et accessible dans

sa totalité par le simulateur. L'attribution de la ressource au processus de gauche, par exemple, sera concrétisée par la présence d'un jeton dans la place Alloc/P2. L'indéterminisme est ainsi levé et les graphes évoluent naturellement selon les règles de fonctionnement des réseaux de Petri. Par contre, qu'advient-il de cette modélisation si les deux processus sont implantés sur deux automates différentes ? En effet, les données ne sont alors plus globales, ni indivisibles. En particulier, la ressource symbolisée par la place R, ne peut être programmée telle qu'elle a été modélisée. Elle se trouve désormais "à cheval" entre deux automates qui ont chacun besoin de sa description pour compléter leur commande. Il s'ensuit que la réalisation effective sur des organes de commande industriels s'éloigne de la modélisation initiale. Les résultats qualitatifs de la simulation validant le bon comportement général du système sont caduques pour la bonne raison que ce qui est modélisé ne correspond pas à ce qui est implanté.

Ces deux exemples montrent bien les différences fondamentales qui existent entre la modélisation simulée et le résultat réel de l'implantation. Pour y remédier, il est nécessaire que, durant la Phase II où est abordée la répartition des systèmes de commande (cf. Fig. 14), ce qui relève plus particulièrement du niveau hiérarchique ou de la commande effective ou encore du procédé ne soit pas seulement étudié et dissocié. Il est obligatoire que soit également prise en compte à ce niveau la répartition des process caractérisant la commande sur les différents organes informatiques utilisés. Elle doit intégrer deux concepts révélés par l'implantation :

(i) *Le découpage de la description en base de données.* Ce point doit mettre en avant les problèmes de communication entre les processus. Il est en effet erroné de considérer que toutes les données caractérisant le fonctionnement sont simultanément accessibles par tous les processus. La gestion d'un protocole de communication devra ainsi être explicitement mise en œuvre (requêtes avec accusé de réception, par exemple) pour échanger des données entre deux organes distincts.

(ii) *L'asynchronisme réel entre les programmes implantés sur des supports informatiques différents.* Ce deuxième aspect, engendré par le découpage du point précédent, soulève les problèmes de cohérence liée à l'évolution

dans le temps des process. L'asynchronisme conditionne le fonctionnement du niveau hiérarchique vis à vis de la commande. En effet, si un mode de supervision "espion" est choisi (surveillance à l'initiative exclusive du niveau hiérarchique avec accès aux données complètement transparent pour la commande), il faut nécessairement que la commande intègre des points d'arrêts aux endroits critiques pour s'assurer que l'organe de supervision prenne en compte son état courant à ces endroits précis. Sinon, rien ne prouve que ces informations aient été acquises par le niveau hiérarchique.

La segmentation des données peut être efficacement réalisée sous Le - LISP par l'utilisation du concept de packages [CHA 86]. A chaque ensemble de données appartenant à un organe informatique unique est associé un package du nom de cet organe. Ainsi, les informations de deux supports informatiques seront dissociées du fait de leur appartenance à des packages de noms différents. L'accès à des données extérieures à son propre environnement devra explicitement être prévu par programmation et ne sera plus transparent. L'asynchronisme, plus complexe au premier abord, peut être efficacement réglé par la puissance du logiciel Le - LISP. En effet, la fonction (parallel $< e_1 > \ldots < e_n >$) lance en parallèle l'évaluation des différentes expressions $< e_1 > \ldots < e_n >$ [CHA 86]. Chaque expression $< e_i >$ pouvant être la simulation effective d'un package associé à un support informatique unique, nous obtenons une simulation multitâche réelle et non plus une tentative d'approche de l'asynchronisme liée au cycle du simulateur. Cette fonction tourne sur mini (VAX Digital, ...), mais également sur PC, fondamentalement monotâche a priori. Elle utilise pour ce faire, l'horloge interne et réalise un partage du temps ("scheduling") très efficace.

Fort de cette approche, segmentation et asynchronisme, la simulation sera réalisée cette fois sur un modèle plus réaliste. L'implantation peut alors être, dans ses grandes lignes, conforme au modèle simulé et seules peuvent subsister éventuellement les erreurs de transposition du modèle RdPSAC en Grafcet. Encore faut-il que le niveau hiérarchique du simulateur ait le même mode de fonctionnement que celui proposé par l'implantation. Le simulateur étant antérieur à une étude approfondie du niveau hiérarchique, la simulation qui en est faite est donc plus sommaire que le fonctionnement réel. Pour le moment, il se contente d'évaluer en séquence les fonctions LISP traduisant les règles de décision. Au

niveau de l'implantation pratique, un véritable moteur d'inférence d'ordre 0+ évalue par contre les règles (cf. Chapitre suivant). Des différences dans la mise en œuvre subsistent donc ; elles devraient disparaître lors d'une nouvelle version du logiciel de simulation intégrant complètement les remarques soulevées : segmentation, asynchronisme et moteur d'inférence.

II.4 - Situation de nos travaux

La définition des différentes phases successives visant à établir le schéma directeur de la démarche générale du projet C. A. S. P. A. I. M. est source de nombreuses discussions et séminaires. L'avancement du projet a conduit souvent à des remises en cause significatives des options choisies.

Au sein de cette équipe, nous nous sommes plus particulièrement attachés aux problèmes liés à l'implantation des modèles. Dans un premier temps, l'étude dont les grandes lignes ont été présentées en première partie de ce chapitre, s'est axée sur la transposition des RdPSAC en Grafcet, prenant en compte l'absence d'automates sur le marché acceptant les réseaux de Petri. Très vite, il est apparu que le point clef de toute implantation réside dès l'origine dans la prise en compte effective de la répartition des processus sur les organes informatiques disponibles. Le paragraphe précédent a contribué à illustrer les limites d'une description centralisée théorique n'intégrant pas la répartition des systèmes de commande. Conjointement à l'implantation, notre travail s'est porté sur la définition et la réalisation d'un niveau hiérarchique, organe centralisateur et fédérateur du fonctionnement global de l'installation. En effet, retenir un niveau décisionnel, externe et hétérogène vis à vis des automates qu'il régule, se révèle être à la fois la solution la plus performante et la plus évolutive. Pour ce faire, certaines solutions techniques, fonctions du matériel utilisé, ont été imposées et ont en partie déterminé le mode de fonctionnement de l'organe de supervision. Nous présenterons en particulier un mode de surveillance "espion" qui représente, dans notre mémoire, la solution proposée. Il est bien évidemment possible de choisir d'autres fonctionnements comme un mode de communication du type requête / accusé de réception qui peut dans certaines configurations, conduire à de meilleurs résultats.

Nos travaux se situent donc en aval de la démarche. Toutefois, certains résultats liés à l'implantation ont contribué à améliorer la démarche de conception. En particulier, les lacunes inhérentes à la façon ensembliste d'aborder les modèles ont été mises en évidence. Une modélisation intégrant le concept de répartition, non seulement fonctionnel mais également lié à l'implantation, doit être conduite. La réalisation effective ne présentera plus alors de difficulté tant les modèles "off-line" se rapprocheront de la réalité. Un simple téléchargement des différents packages suffira pour que le fonctionnement se déroule normalement, tel qu'il a été envisagé et validé. Le projet C. A. S. P. A. I. M. vise à faire disparaître les démarches empiriques et pragmatiques actuelles quant à la définition des systèmes de production flexible. Il est à rapprocher dans ce sens (toute chose égale par ailleurs) de la méthode de conception de projet informatique MERISE [COL 86]. Allant à l'encontre d'une étude parcellaire et fragmentée, il tente tout au long du cycle de conception de la cellule flexible, de définir une démarche de structuration des systèmes de commande visant à aboutir à une réalisation homogène, normalisée et facilement maintenable.

II.5 - Conclusion

Le projet C. A. S. P. A. I. M. a atteint actuellement un niveau de développement important ; il reste cependant en constante évolution. L'une de ses originalités principales, réside dans son interactivité importante avec l'utilisateur. Le dialogue est conçu de telle sorte qu'il laisse transparent les étapes techniques de modélisation et qu'il n'intervient que pour des spécifications de haut niveau. L'utilisateur n'est de plus jamais enfermé dans un cadre défini par la structure des logiciels ou par l'interface de communication. Cette souplesse provient en grande partie de l'utilisation du langage Le - LISP qui apparaît tout au long de la chaîne de conception. La modélisation est ainsi homogène et le passage d'une phase de la démarche à une autre est immédiat.

Le projet C. A. S. P. A. I. M. permet ainsi de tester et de valider différentes configurations de l'unité spécifiée par le cahier des charges. La conception est **modulaire** et **interactive** et procède selon un schéma directeur rigoureux qui apporte une assistance significative tout au long du projet.

CONCLUSION

Dans ce chapitre, nous avons en premier lieu présenté le modèle retenu pour la description de la commande de processus discontinus. Nous nous sommes ensuite intéressés à la transcription de ce modèle en Grafcet en vue d'une implantation effective. Dans ce cadre, nous nous sommes limités à la traduction d'une représentation intra-processus. Les problèmes liés à la répartition sur différents automates, bien qu'évoqués, seront largement traités au Chapitre II. De cette présentation, il ressort que la transposition est automatique et ne doit donc pas être une source d'erreurs dans la réalisation de la commande.

Nous avons en second lieu, précisé la situation de notre contribution au projet C. A. S. P. A. I. M. réalisé par l'équipe de recherche du L. A. I. I. de l'I. D. N.. Nous avons insisté sur ses principaux points tout en n'hésitant pas à relever certaines limites. Notre travail se situe en aval de la chaîne de conception et reste très tributaire des recherches amont. Il s'avère néanmoins, que ces mêmes recherches sont influencées par les résultats de nos travaux ; c'est en partie ce que nous avons voulu mettre en évidence dans le paragraphe intitulé "limites et extensions".

CHAPITRE II

IMPLANTATION ET NIVEAU HIERARCHIQUE

INTRODUCTION

Pour répondre aux impératifs de productivité (i.e. taux d'engagement des machines, diminution des stocks, réduction des délais, ...), les systèmes de production flexible ont vu leur complexité croître de façon exponentielle. Leur mise en œuvre implique la coopération étroite entre de multiples systèmes de commande informatique. Nous pouvons citer à titre d'exemple, les commandes numériques des machines outils, les armoires de pilotage des robots, les automates programmables régissant tous les automatismes, enfin le site central qui doit lancer les ordres de fabrication et gérer la production [BAR 85]. Historiquement, la coopération entre les différents modules de commande au niveau de l'atelier a pu être réalisée par l'intermédiaire d'entrées / sorties tout-ou-rien, leur multiplicité ne parvenant pas à combler la pauvreté de l'information échangée. Face à ce constat, les constructeurs de matériels proposent aujourd'hui des réseaux locaux industriels qui autorisent la connexion de matériels, parfois hétérogènes, et permettent d'échanger des trames d'informations plus riches sémantiquement [MIN 87] [CIN 87] [GAL 87].

L'évolution des matériels autorise donc désormais une vision globale des procédés ainsi que de la commande. Les systèmes de production flexible intègrent les notions de gestion de production [BON 85] :

- définition des stratégies,
- ordonnancement dynamique,
- lancement des opérations,
- analyse de la qualité [VER 86], ...

et les différents modes de fonctionnement.

- marche de production normale,
- marche dégradée,
- vérification, ...

Ces concepts, de par leur complexité, ne peuvent en aucun cas se contenter d'informations partielles, ni locales. Ils imposent un raisonnement sur une quantité d'informations importantes et réparties sur différents organes de commande afin d'aboutir généralement à une modification de la commande ou du procédé. Dans ce chapitre, nous allons définir

une classification des problèmes rencontrés lors de la supervision d'une cellule flexible et nous justifierons de cette façon, l'utilisation d'un niveau hiérarchique, organe de contrôle découplé de la commande directe des procédés.

I - LE FONCTIONNEMENT EN MODE DE MARCHE NORMALE

I.1 - Introduction

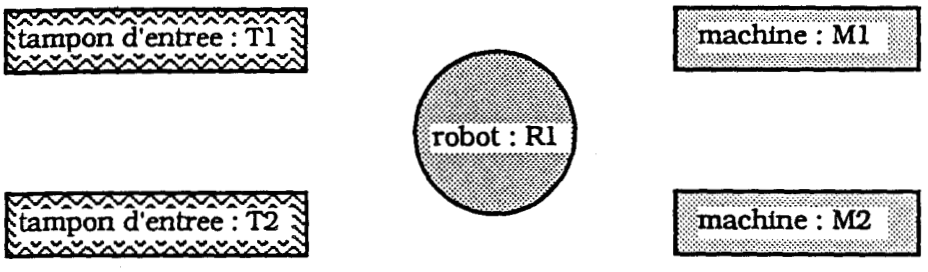
Lors de la définition et de la conception des modèles régissant la commande de son installation, le concepteur ne se préoccupe pas en premier lieu de la répartition des différents process mis en œuvre. Il utilise au mieux les outils banalisés qui sont mis à sa disposition afin d'aboutir à un graphe normalisé correspondant rigoureusement aux spécifications de son cahier des charges. Néanmoins, la modélisation obtenue laisse en suspend deux considérations à prendre en compte en phase d'implantation. Les réseaux de Petri structurés colorés et adaptatifs ne sont pas autonomes du point de vue du fonctionnement. L'utilisation de la coloration et également d'arcs adaptatifs pondérés par des places non directement connectées à des process rendent le fonctionnement du réseau non déterministe. Enfin, l'implantation effective des graphes de commande nous oblige à considérer les groupements fonctionnels de process et a contrario, l'implantation répartie de process ayant un fort degré de connexité sur des organes différents. Cette démarche a pour effet de rendre locales des variables dont l'évolution ne peut être coordonnée qu'à partir d'une vision globale des procédés. Ces constatations induisent naturellement l'utilisation d'un niveau hiérarchique qui a pour première fonction, le rôle de coordonnateur dans le fonctionnement global de l'intallation [FRO 84].

Nous allons maintenant détailler les différentes primitives de communication, montrer comment leur traduction en Grafcet peut être systématique et enfin comment l'utilisation du niveau hiérarchique simplifie la traduction en diminuant de façon notable la taille des Grafcets tout en augmentant la puissance de résolution du système.

I.2 - La ressource

I.2.1 - Définition du problème

Soit, par exemple, la configuration suivante :

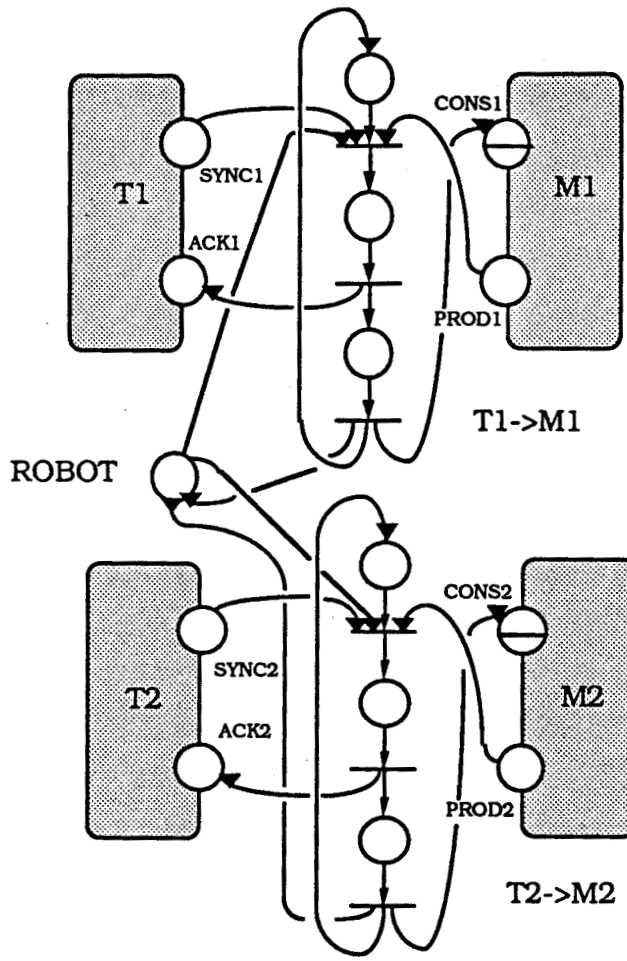


Configuration

FIGURE 1

Un tampon d'entrée T1 est dévolu à l'alimentation exclusive de la machine M1, réciproquement le tampon d'entrée T2 alimente uniquement la machine M2. Pour des raisons économiques dues à une faible fréquence dans l'alimentation des machines M1 ou M2, un seul robot est jugé amplement suffisant pour la gestion du système.

Une simulation temporisée a pu montrer que l'utilisation dans cette configuration, de deux robots n'était pas rentable. La structuration du système de commande conduit à 2 modules chargés du transfert, avec l'utilisation de primitives de communication banalisées [BOU 88] qui se partagent le robot constituant une ressource exclusive.



Modélisation de la commande

FIGURE 2

L'implantation impose pour des raisons pratiques, l'utilisation de deux automates, l'un dédié au tampon d'entrée T1 et à la machine M1, donc également au transfert modélisé par le process T1 → M1 ; le second est dédié respectivement à T2, M2 et T2 → M2. Nous proposons de régler la gestion de cette ressource commune.

Si les deux automates ne peuvent en aucune façon communiquer l'un avec l'autre, le problème d'attribution est de toute évidence indécidable.

Si nous prenons en compte l'existence d'une communication dont les spécifications sont les suivantes :

- un automate peut lire et écrire ses propres variables,
- un automate peut lire des variables d'autres automates, configuration géné-

ralement rencontrée dans l'industrie (hypothèse réalisée par exemple dans le contexte du réseau TELWAY des matériels de la série 7 de la Téléméca-nique).

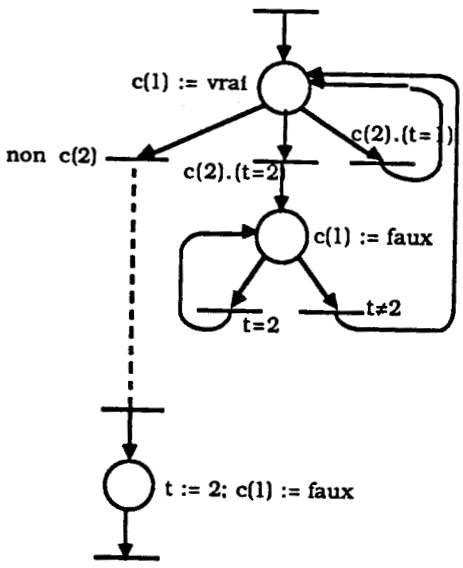
En l'absence de zone mémoire commune, la communication se fait par entrées / sorties, ce qui correspond bien aux spécifications précédemment décrites. Si une variable binaire d'un automate doit être accessible en lecture par un autre automate, il est suffisant d'associer à cette variable une sortie qui en sera l'image fidèle et qui constituera une entrée du second automate. Le second moyen de communication disponible, souvent moins rapide mais plus riche en informations transmises, consiste en l'existence d'un réseau sur lequel circule des variables globales en lecture seulement [BLA 85].

I.2.2 - Solution de Dekker

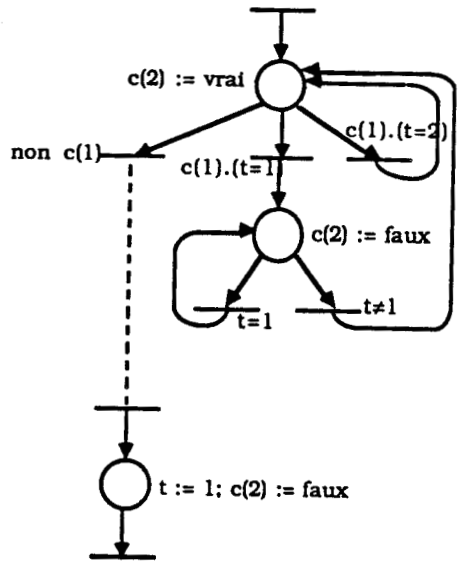
La solution à ce problème a été donnée par DEKKER [CRO 75] dans le contexte du partage de ressources informatiques (mémoire centrale, entrées / sorties, impri-mantes, ...) dans le cadre d'une utilisation d'un système multitâches. L'algorithme proposé nécessite l'emploi de deux variables binaires et d'une variable entière, toutes trois acces-sibles globalement en lecture. Il induit de plus une politique d'attribution gérée en FLIP/FLOP (en cas de demandes simultanées, la ressource est alternativement donnée à l'un ou à l'autre). L'algorithme minimal trouvé et sa traduction en réseaux de Petri équiva-lents sont les suivants :

```
entier t ;
booléen tableau c(1:2) ;
t := 1 ; c(1) := c(2) := faux ;

process Pi :
        début
        Ai :    c(i) := vrai
        Li : si c(j) alors
                début
                        si t = i alors aller à Li
                        c(i) := faux
        Bi :            si t = j alors aller à Bi
                        aller à Ai
                fin ;
                ...
          t := j ; c(i) := faux
             reste du programme
          aller à Ai
        fin ;
```

Algorithme de DEKKER



Processus P1



Processus P2

Traduction en RdP

La variable c(1) (resp. c(2)) traduit la demande d'attribution de la ressource de la part du processus P1 (resp. P2). L'entier t détermine la priorité d'attribution en cas de demandes simultanées.

Cette solution a pour effet de noyer l'algorithme de gestion de la ressource dans le cadre même du programme complet. Nous allons proposer plutôt une solution qui délocalisera l'algorithme de résolution par rapport au reste du programme. Cette scission aura l'avantage de clarifier les graphes de commande obtenus. La traduction automatique du processus de transfert exprimé en RdPSAC donnera un Grafcet qui aura bien pour fonction de se charger de la commande des automatismes du robot sans être alourdi par la gestion de la ressource.

I.2.3 - Solution répartie exprimée en Grafcet

Définitions

- Les variables REQi_PRISE_R et REQi_REST_R sont des variables accessibles en lecture / écriture pour l'automate i ($i \in \{1,2\}$) et en lecture seulement pour l'automate 3.

- Les variables ACKt_PRISE_R et ACKt_REST_R sont des variables accessibles en lecture / écriture pour l'automate 3 et en lecture seulement pour l'automate t ($t \in \{1,2\}$).

L'automate 3 est celui sur lequel est implanté l'algorithme d'attribution de la ressource (il peut éventuellement être l'automate 1 ou l'automate 2). Par souci de cohérence de l'information et afin d'éviter de dupliquer inutilement la représentation de la ressource, il est souhaitable de centraliser la gestion de cette dernière sur un seul et même automate (automate 3 en l'occurence). Pour palier l'absence de variables globales en lecture / écriture sur plusieurs automates, il est nécessaire d'éclater les transitions de prise et restitution de la ressource, cela pour chaque processus, afin de faire apparaître les communications avec l'automate assurant la gestion de cette ressource. Le problème de requêtes simultanées est résolu par l'utilisation de priorités statiques ou dynamiques

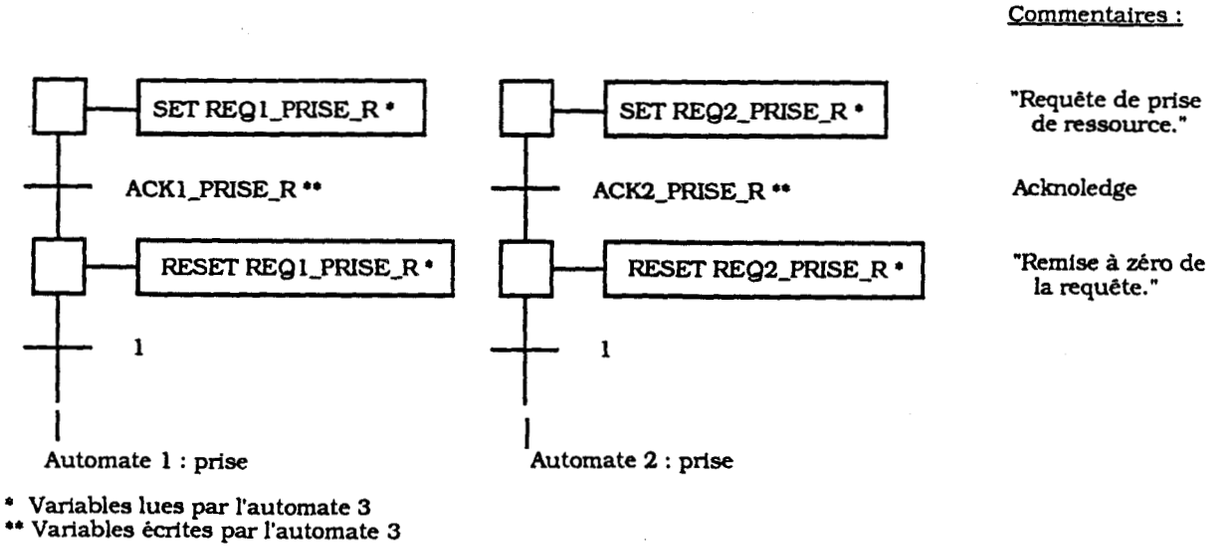
(PRIO1 et PRIO2) qui assurent un accès exclusif à la ressource. La définition de ces priorités (choix de l'affectation à l'un ou à l'autre en cas de demande simultanée) est laissée à la charge de l'utilisateur.

D'après les hypothèses et définitions précédemment énoncées, nous obtenons les graphes de commande suivants :

Figure 3 : prise de ressource

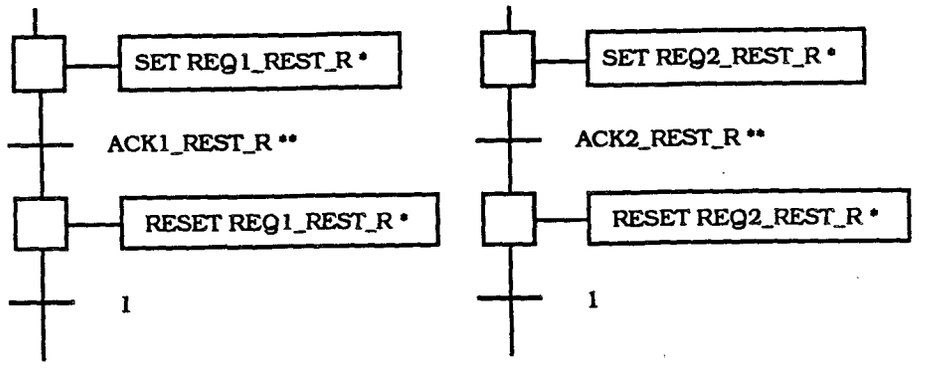
Figure 4 : restitution de ressource

Figure 5 : gestion de l'attribution



Prise de ressource

FIGURE 3



Commentaires :

"Requête de restitution de la ressource."

Acknoledge

"Remise à zéro de la requête."

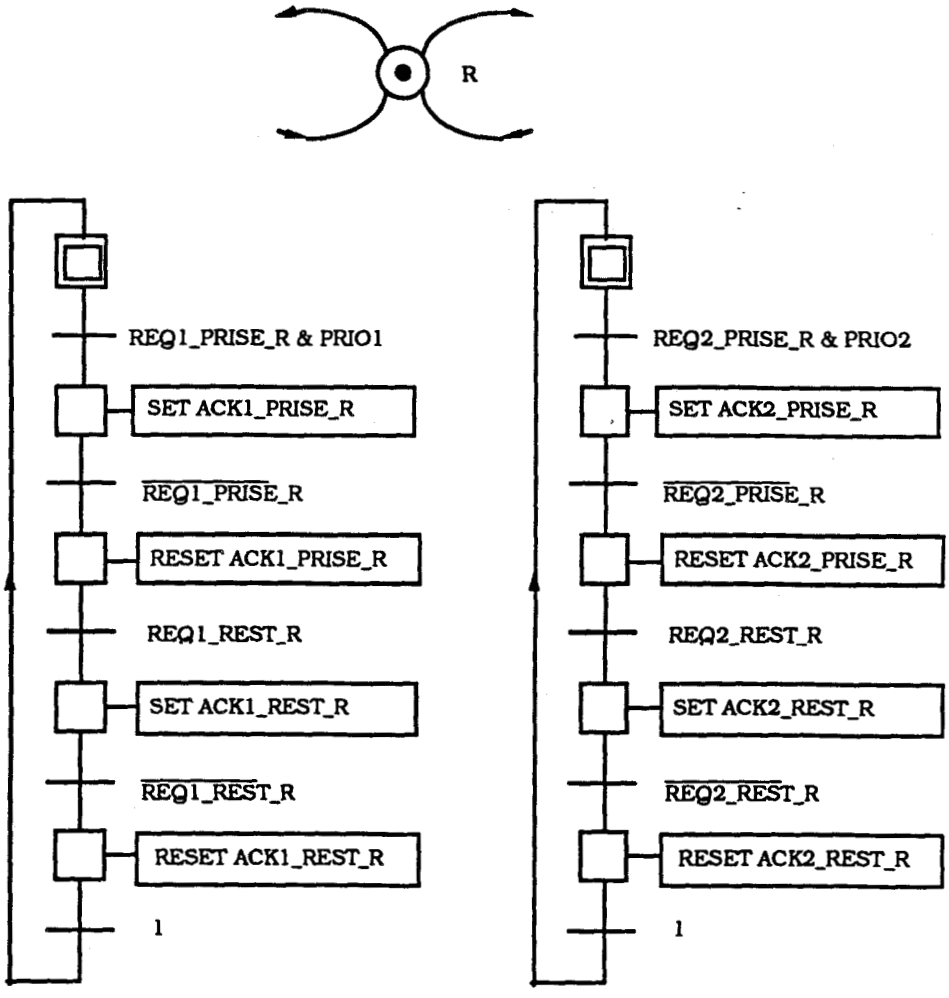
Automate 1 : restitution

Automate 2 : restitution

* Variables lues par l'automate 3
** Variables écrites par l'automate 3

Restitution de ressource

FIGURE 4



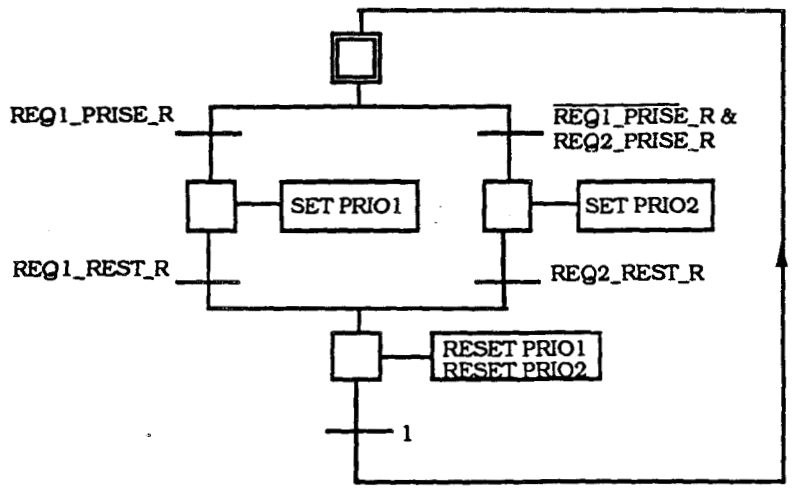
Automate 3 : gestion des variables locales de l'automate 1

Automate 3 : gestion des variables locales de l'automate 2

Gestion de l'attribution

FIGURE 5

Les variables REQ1 _ PRISE _ R et REQ2 _ PRISE _ R peuvent être utilisées au niveau global de l'automate 3 pour élaborer la priorité en cas de conflit. L'attribution de la ressource à l'automate 1 ou 2 se traduira alors par l'affectation exclusive de la variable PRIO1 ou PRIO2. Un exemple de solution pour la gestion des variables PRIO1 et PRIO2 en Grafcet local à l'automate 3 est présenté ici. En cas de demande simultanée, la solution retenue donne systématiquement la priorité à l'automate 1.



Cette méthode de transcription, présentée pour deux processus, n'est en aucun cas limitative. L'ajout d'un processus supplémentaire utilisant la ressource n'entraîne que la création d'un processus de gestion des requêtes ainsi que la mise à jour des priorités. De même, le concept d'exclusion mutuelle peut facilement être étendu à la gestion de m ressources pour n process ; il suffit alors au niveau des graphes de commande gérant l'attribution, d'intégrer non plus un accès exclusif (assuré par PRIOi), mais d'assurer une politique d'affectation tenant compte également de la multiplicité des ressources.

I.2.4 - Solution utilisant un superviseur

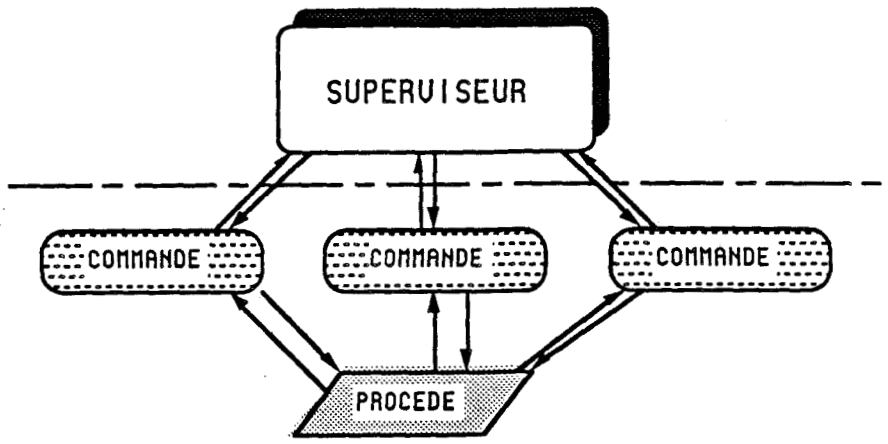
La modélisation obtenue précédemment induit implicitement une mise à plat des différentes fonctionnalités de la commande. Un seul modèle de représentation est utilisé (le Grafcet) pour décrire aussi bien la commande des automatismes purement séquentiels que les algorithmes permettant de faire coopérer ces automatismes. Autant le Grafcet est un modèle de représentation graphique du comportement dynamique de la partie commande d'un système automatisé, autant il se prête difficilement à la définition et à la modélisation de stratégies visant à faire coopérer les différents systèmes de commande dans le cadre d'une implantation répartie. De plus, le langage utilisé est, par essence même, séquentiel ou encore procédural par intégration du concept de sous-programme (extension de la norme Grafcet NFCØ3-190 [GUI 85]) alors que la résolution d'indéterminisme peut très bien nécessiter un formalisme plus puissant exprimé de façon déclarative. Enfin, la définition d'un organe de supervision différent des organes de commande permet au concepteur de classer les différentes natures de problème dont il a la charge et lui

permet ainsi de focaliser son attention sur une seule difficulté à la fois :

- premier temps : définition de la commande des automatismes séquentiels,

- deuxième temps : résolution des indéterminismes (ressource), ou mise en œuvre des primitives de communication (producteur / consommateur, synchronisation avec accusé de réception) sans modifier le graphe précédemment établi.

Dissocier la description de la commande des automatismes séquentiels, de la politique générale d'ordonnancement de ces automatismes permet de structurer la démarche de modélisation en demandant au concepteur de ne se préoccuper que d'une seule nature de problème à la fois. L'utilisation d'un langage séquentiel (le Grafcet) pour la représentation dynamique des automatismes et l'emploi d'une démarche déclarative traduisant les stratégies d'ordonnancement de l'installation permet d'utiliser au mieux les spécificités de chacun de ces deux formalismes. Une approche déclarative facilite particulièrement la mise au point des différentes stratégies. Elle autorise par simple modification, ajout ou retrait de règles, le test de nombreux cas de figure de fonctionnement sans pour cela remettre en cause la description préalable de la commande, et favorise ainsi de façon simple la recherche du fonctionnement optimal. En outre, les modifications rendues nécessaires pendant la durée de vie du système (ex : changement de priorité pour accélérer la production d'une gamme opératoire par rapport à l'ensemble des gammes circulant dans le système) sont facilitées de façon analogue à la mise au point, par l'approche déclarative. Cette classification des problèmes relevés par la structuration nous amène à introduire un niveau hiérarchique (Fig. 6) dissocié des organes de commande et permettant dans un premier temps, le fonctionnement global de l'installation dans le cadre d'une implantation répartie.



Structure du système de surveillance

FIGURE 6

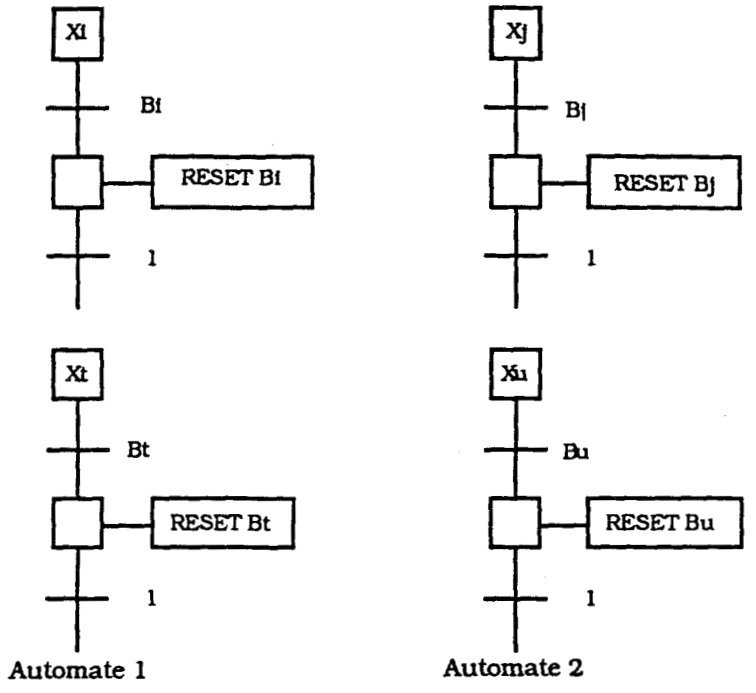
Définitions et limitations

Nous supposerons par la suite, que l'organe de supervision implanté au niveau hiérarchique est capable de rapatrier automatiquement et de manière transparente pour l'utilisateur, les informations suivantes :

- état du marquage des Grafcets des différents automates,
- valeur des variables internes des automates : booléens, entiers, ...

Il peut notamment modifier automatiquement l'état des graphes de commande par écriture dans les variables booléennes et entières des automates. Nous n'avons pas retenu l'idée d'autoriser la modification dynamique du marquage des Grafcets pour des raisons évidentes de sécurité. Le superviseur ne doit pas se substituer au scheduler des automates pour permettre l'évolution des graphes. Ainsi, dans l'exemple d'attribution de la ressource, la mise en œuvre se fera de la façon suivante (Fig. 7) :

- les réceptivités en amont de la section critique (bits B_i et B_j) sont initialement et a priori dans l'état faux,
- les réceptivités en aval de la section critique (bits B_t et B_u) sont initialement et a priori dans l'état faux.



Modélisation d'une ressource au niveau commande

FIGURE 7

L'organe de supervision, en fonction des informations dont il dispose (marquage des places en amont de l'attribution et de la restitution) et de la stratégie définie (ici exprimée sous forme de règles de production), décidera de la politique à suivre et modifiera indirectement le marquage des graphes de commande en jouant sur la valeur des réceptivités des transitions critiques (bits B_i, B_j, B_t, B_u). La transcription de la phase de modélisation à la phase d'implantation devient alors automatique. La modélisation de la ressource : une place en réseau de Petri, est reportée par une interprétation des transitions concernées. Sa gestion est du ressort de l'organe de supervision (Fig. 8), utilisant un formalisme déclaratif et susceptible d'être modifié a posteriori sans pour autant remettre en cause la commande. Cette modification peut aussi bien provenir de résultats de simulation, prouvant que telle politique d'attribution est meilleure que celle utilisée, ou de l'expérience de l'utilisateur confronté à des essais sur le site même.

ATTRIBUTION :

si (egal ressource libre) et (egal Xi marquee)
alors (affecte ressource occupee) et (affecte Bi 1)

si (egal ressource libre) et (egal Xj marquee) et (egal Xi non_marquee)
alors (affecte ressource occupee) et (affecte Bj 1)

RESTITUTION :

si (egal ressource occupee) et (egal Xt marquee)
alors (affecte ressource libre) et (affecte Bt 1)

si (egal ressource occupee) et (egal Xu marquee)
alors (affecte ressource libre) et (affecte Bu 1)

Règles de gestion de la ressource

(priorité fixe à l'automate 1)

FIGURE 8

I.3 - Le producteur / consommateur

I.3.1 - Définition

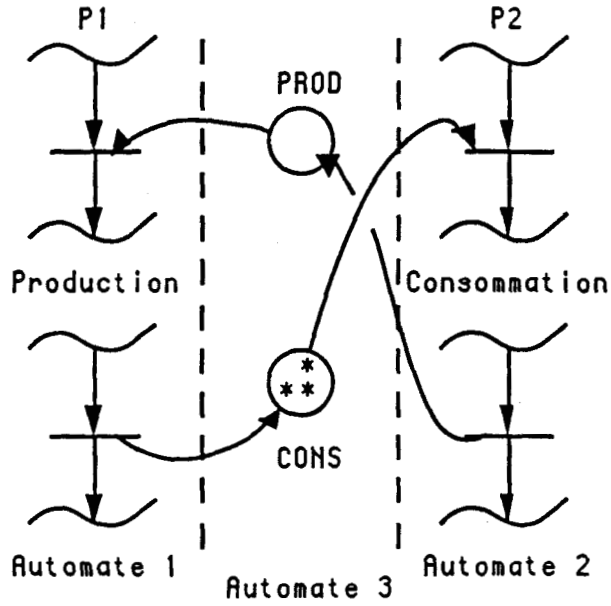
Le problème des producteurs / consommateurs est le type même de problème rencontré lors de la conception et de la mise en œuvre des systèmes distribués. C'est un exemple "canonique" que l'on retrouve dans toutes les présentations de langage incluant le concept de processus parallèle (ADA en est le schéma type). Un processus producteur produit des messages que le consommateur doit absorber. Le système producteur / consommateur doit assurer l'invariant suivant : il existe au plus n messages produits et non consommés. La structure du producteur / consommateur exprimé en réseau de Petri a été donnée au Chapitre I. Dans l'exemple présenté au § I.2.1, nous obtenions automatiquement la création de deux producteurs / consommateurs comme primitives de communication banalisées entre le processus de transfert T1 → M1 (respectivement T2 → M2) et la machine M1 (resp. M2). Nous nous attachons maintenant à définir leur mise en œuvre dans le cadre exclusif d'un système distribué (en effet, l'implantation d'un producteur / consommateur sur un site centralisé ne pose aucun problème particulier).

I.3.2 - Solution répartie exprimée en Grafcet

Différents travaux visant à définir les algorithmes de fonctionnement et prenant en compte les limitations inhérentes à la communication ont amené plusieurs solutions [PLO 87] :

- les variables de contrôles (les compteurs) modifiées par le producteur sont placées sur le site de production et les variables modifiées par le consommateur sur le site de consommation,
- principe du jeton transportant les variables à distribuer en assurant l'exclusion mutuelle,
- distribution par duplication, ...

Dans le cadre de nos préoccupations, nous avons retenu une solution qui symétrise parfaitement le problème. Par souci de simplicité, d'homogénéïté et de sûreté, la transcription en Grafcet conserve la même philosophie que celle utilisée pour la transcription de la liaison d'exclusion mutuelle. Ainsi, un processus P1 implanté sur l'automate 1 produit des entités pour un processus P2 implanté sur l'automate 2. La gestion des places PROD et CONS sera à la charge d'un automate 3 différent a priori des deux précédents (Fig. 9).



Répartition du producteur / consommateur

FIGURE 9

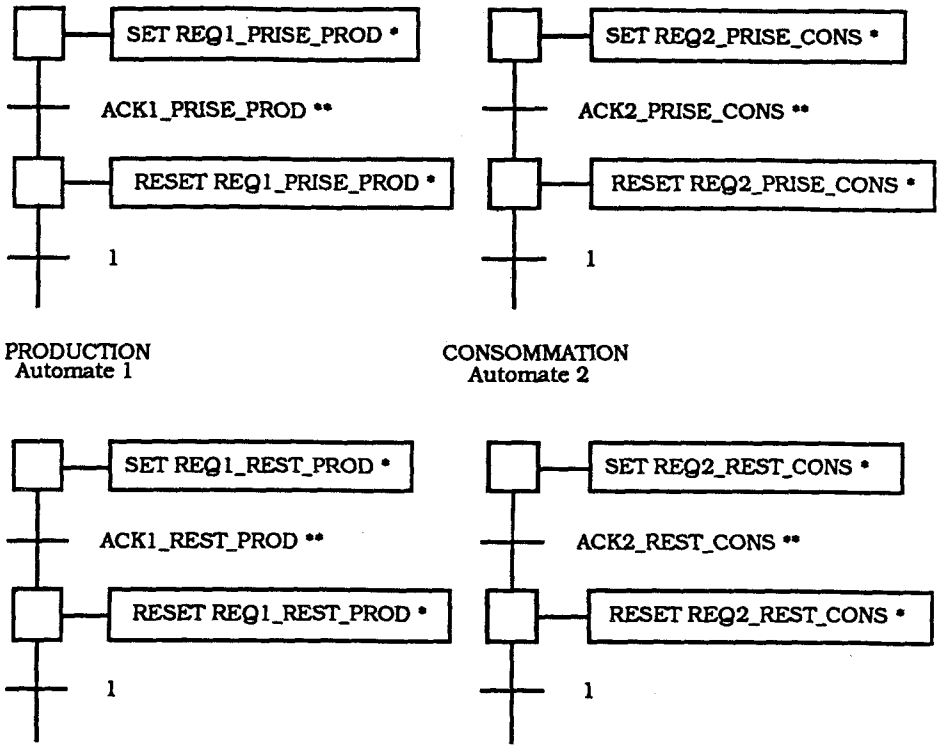
Définitions

- Les variables REQ1 - PRISE - PROD et REQ1 - REST - PROD sont des variables accessibles en lecture / écriture pour l'automate 1 et en lecture seulement pour l'automate 3.

- Les variables REQ2 - PRISE - CONS et REQ2 - REST - CONS sont des variables accessibles en lecture / écriture pour l'automate 2 et en lecture seulement pour l'automate 3.

- Les variables ACK1 - PRISE - PROD et ACK1 - REST - PROD (respectivement ACK2 - PRISE - CONS et ACK2 - REST - CONS) sont des variables accessibles en lecture / écriture pour l'automate 3 et en lecture seulement pour l'automate 1 (resp. l'automate 2).

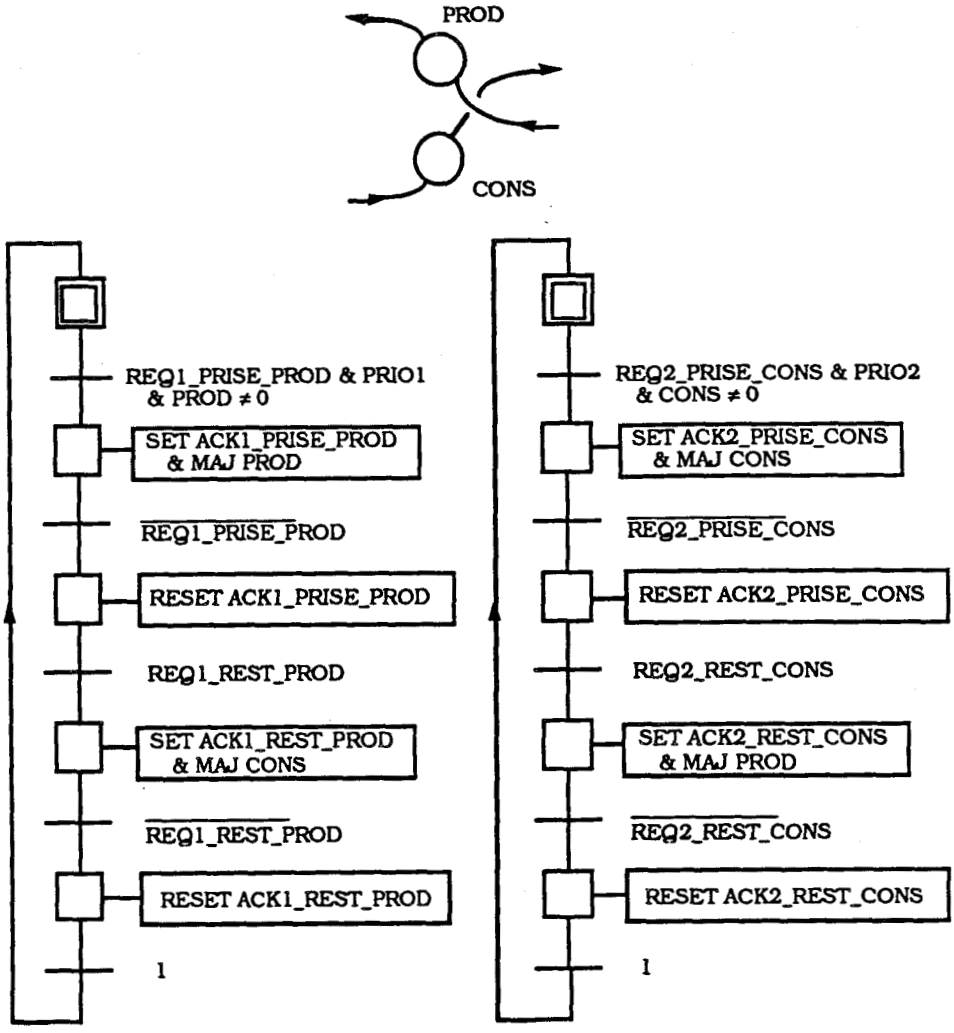
On remarque de cette façon, que les protocoles de communication entre les automates utilisant le producteur / consommateur et l'automate dédié à la gestion de ce dernier diffèrent très peu de ceux présentés pour l'exclusion mutuelle (Fig. 10 et 11). La modélisation des places PROD et CONS est assurée par deux mots internes à l'automate 3. En outre, la définition des priorités PRIO1 et PRIO2 ne s'avère utile que dans le cas de plusieurs producteurs ou consommateurs. Il convient ensuite de prendre en charge la mise à jour des places PROD et CONS : solution classique dans le contexte d'un traitement centralisé pour lequel les questions de sécurité dues à l'emploi d'un traitement réparti n'apparaissent plus.



* Variables lues par automate 3
** Variables écrites par l'automate 3

Producteur / consommateur sur automate 1 et 2

FIGURE 10



Gestion sur l'automate 3

FIGURE 11

I.3.3 - Solution utilisant un superviseur

La modélisation du producteur / consommateur exprimée en Grafcet, analogue en grande partie à celle de la ressource, en conserve donc les mêmes lourdeurs inhérentes aux primitives de communication utilisées ainsi qu'à la formalisation employée. De nouveau, nous proposons une solution utilisant un organe de supervision découplé de la commande. L'interfaçage avec la commande est non seulement automatique, mais systématique. La modélisation retenue est textuellement la même que celle employée pour la ressource critique. Les places sensibles en amont des points de production ou de consom-

mation ont leur réceptivité respective initialement fausse. Nous obtenons ainsi strictement les mêmes graphes de commande que ceux présentés à la Figure 7. Seule diffère l'interprétation exprimée au niveau hiérarchique.

La gestion du producteur sur l'automate 1 se fera à l'aide des deux règles suivantes :

- **Règle 1** : Début de production
  Si (différent PROD Ø) et (égal X_i marquée)
  alors (ajoute PROD-1) et (affecte B_i 1)

- **Règle 2** : Fin de production
  Si (égal X_t marquée)
  alors (ajoute CONS-1) et (affecte B_t 1)

La gestion du consommateur est totalement symétrique à celle du producteur. Cela est normal car le problème du producteur / consommateur est fondamentalement symétrique ; seul le marquage initial des places PROD et CONS permet de les particulariser.

- **Règle 3** : Début de consommation
  Si (différent CONS Ø) et (égal X_j marquée)
  alors (ajoute CONS-1) et (affecte B_j 1)

- **Règle 4** : Fin de consommation
  Si (égal X_u marquée)
  alors (ajoute PROD-1) et (affecte B_u 1)

I.3.4 - Solution incluant la coloration

L'exemple précédent a présenté l'utilisation d'un producteur / consommateur où deux hypothèses implicites avaient été posées :

- les marques circulant au sein des graphes de commande

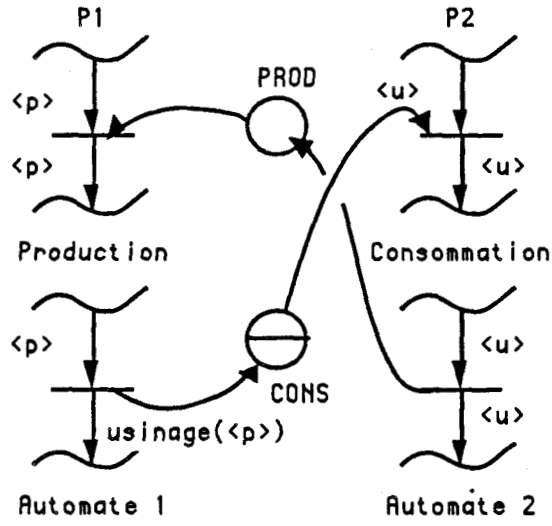
n'étaient pas colorées,

- la structure du tampon de communication (place CONS) n'était pas spécifiée.

Le concept de coloration des marques [JEN 81] donne la possibilité d'effectuer des regroupements fonctionnels ou structurels afin de diminuer la taille des réseaux. Dans le cadre de la productique, nous réalisons des regroupements fonctionnels d'opérations de même nature sur des pièces différentes mais s'effectuant sur une même entité physique.

Exemple

Etendons la définition du processus P1. Ce dernier modélise une séquence d'usinage affectant deux types de pièce brute p1,p2 qui donnent respectivement u1,u2 après leur usinage spécifique. La modélisation présentée Figure 9 devient la suivante :



avec
dom(<p>)={p1,p2}
dom(<u>)={u1,u2}
usinage(p1)=u1
usinage(p2)=u2

PROD / CONS coloré

FIGURE 12

Cette modélisation utilisant des marques colorées nécessite une transposition en Grafcet utilisant une variable supplémentaire. En effet, la marque utilisée en Grafcet sert

exclusivement à signifier l'activité d'une étape ; elle ne peut donc en aucun cas véhiculer d'informations complémentaires. La couleur sera ainsi traduite sous la forme d'une variable numérique au sein d'un mot de l'automate. Le superviseur ne se contente plus dans ce cas de gérer la place CONS comme un simple compteur. Il lui associe en fait une véritable structure de données dans laquelle il mémorise les informations relative à la coloration des différentes entités. Cette constatation remet en cause la seconde hypothèse implicite. Le tampon (place CONS) ne peut plus être modélisé sous la forme retenue précédemment.

Ceci nous amène à étendre sa structure en introduisant le concept de file d'attente. Ces files d'attente seront gérées, suivant la politique retenue par l'utilisateur, en FIFO, LIFO, anneau, ...

La définition des règles utilisées au niveau hiérarchique diffère sensiblement de celle proposée pour l'exemple précédent sans couleur. Nous posons par hypothèse, que la couleur de la marque du processus P1 (respectivement P2) est véhiculée au sein du mot WØ-1 de l'automate 1 (resp. WØ-2 de l'automate 2). La première règle caractérisant le début de production est identique car la variable PROD est un simple compteur non coloré autorisant la production.

La règle 2 devient la suivante :

- **Règle 2** : Fin de production
  Si (égal X_t marquée)
  alors (put-fifo CONS WØ-1) et (affecte B_t 1)

La variable CONS est une structure de données gérée en FIFO ("put-fifo") dans laquelle on range la nature de la pièce produite dont la couleur caractéristique est rangée dans le mot WØ-1.

- **Règle 3** : Début de consommation
  Si (différent CONS Ø) et (égal X_j marquée)
  alors (get-fifo CONS WØ-2) et (affecte B_j 1)

Dans ce cas, si la fifo est non-vide, on prend la valeur en tête de fifo (cette valeur caractérise la nature ou couleur de la pièce qui va être consommée) pour la ranger dans le mot WØ-2.

Enfin, la règle 4, de façon analogue à la règle 1, n'est pas modifiée (il n'existe pas de couleur à ce niveau).

Cette dissociation systématique des graphes de commande par rapport aux stratégies utilisées pour leur permettre de communiquer entre eux vise à rendre automatique le passage de la modélisation à l'implantation répartie. Le concepteur conserve tels quels les process gérant le séquencement des automatismes de production défini dans son modèle, supprime les primitives de communication inter-processus pour les reporter au niveau hiérarchique. Faisant cela, il ne préjuge pas de la politique d'ordonnancement global de son installation (définition de la stratégie d'attribution d'une ressource a posteriori, modélisation d'un producteur / consommateur en FIFO plutôt qu'en LIFO, ...) et conserve donc une flexibilité maximale quant au fonctionnement des unités de production.

I.4 - La synchronisation avec accusé de réception

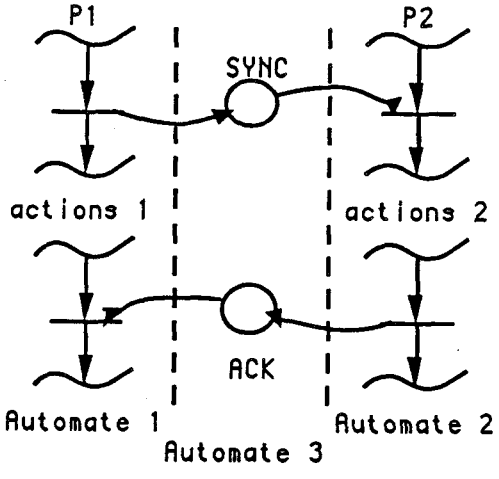
I.4.1 - Définition

Nous introduisons maintenant le troisième et dernier type de liaison banalisée utilisée dans la modélisation : la synchronisation avec accusé de réception. Cette primitive apparaît deux fois dans l'exemple introduit lors de la définition du problème concernant la ressource (Fig. 2). Elle autorise la synchronisation entre les tampons d'entrées T1 (respectivement T2) et les process de transfert T1 → M1 (resp. T2 → M2).

I.4.2 - Solution répartie exprimée en Grafcet

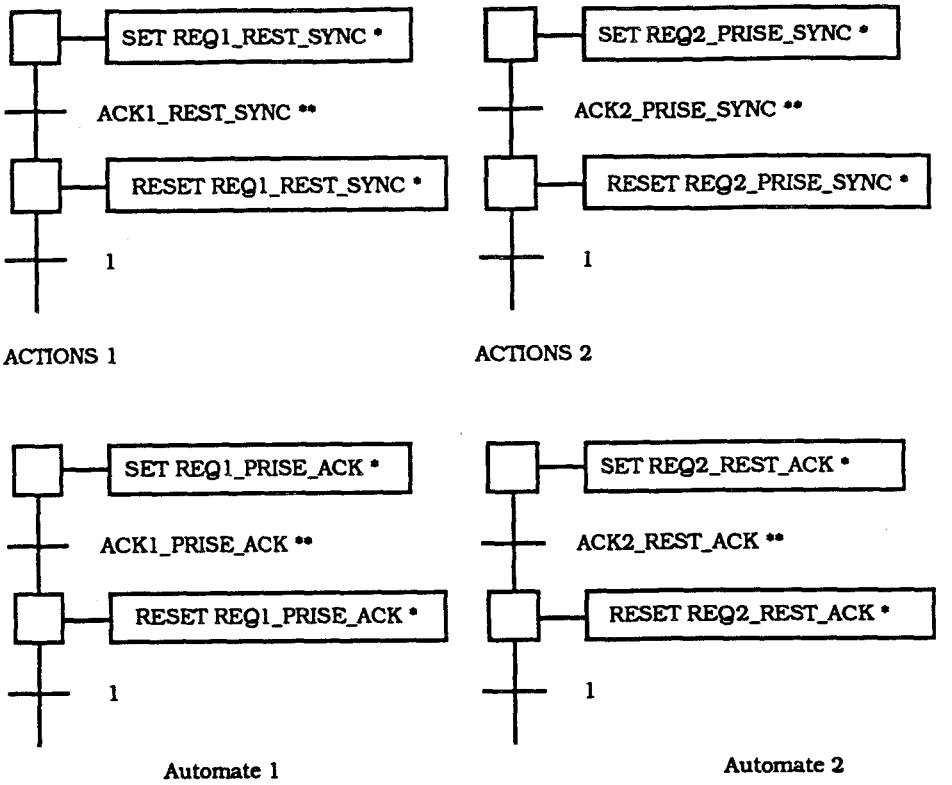
La transcription de la synchronisation avec accusé de réception dans le cadre de process répartis sur différents organes de commande est d'autant plus simple qu'elle ne diffère que sensiblement de la transcription du producteur / consommateur. En effet, cette

primitive de communication ne se particularise de la précédente que par son marquage initial. Le passage de la formalisation en réseau de Petri (Fig. 13) à une implantation en Grafcet répartie (Fig. 14 et 15) est automatique et ne pose aucun problème particulier. Les variables PRIO1 et PRIO2 se justifient dans l'hypothèse où l'on étend le nombre de process utilisant la synchro.



Formalisation en réseau de Petri

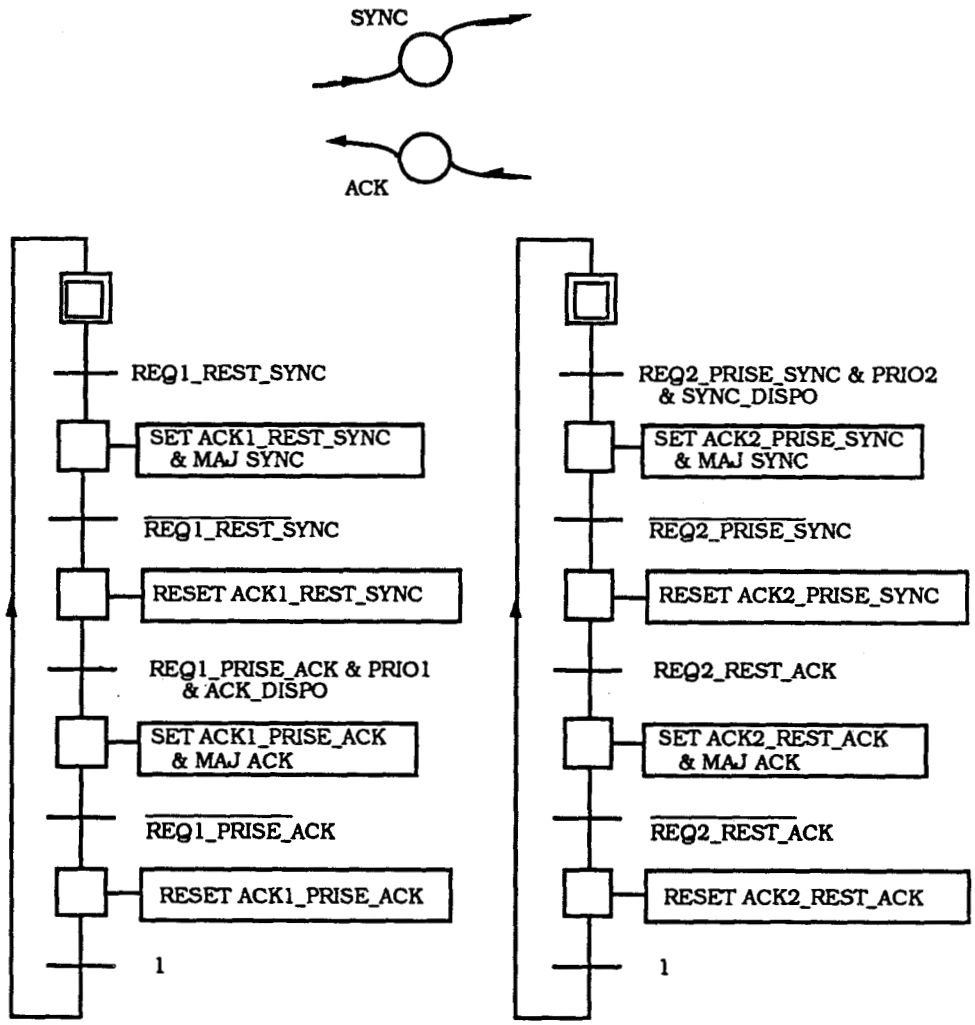
FIGURE 13



* Variables lues par l'automate 3
** Variables écrites par l'automate 3

Synchro avec accusé de réception sur automates 1 et 2

FIGURE 14



Automate 3 : gestion des variables locales de l'automate 1

Automate 3 : gestion des variables locales de l'automate 2

Gestion de l'attribution

FIGURE 15

I.4.3 - Solution utilisant un superviseur

Plus simplement encore, la solution utilisant un superviseur reprend exactement la même structure de commande utilisée aussi bien pour la ressource que pour le producteur / consommateur (Fig. 7). L'interfaçage entre les Grafcets de commande et le superviseur est ainsi totalement banalisé ; l'interprétation des règles exprimées au niveau hiérarchique permet alors de particulariser les différentes fonctionnalités des liaisons utilisées. Dans le cadre de la synchronisation avec accusé, nous obtenons simplement les 4 règles

suivantes :

- **Règle 1** : Début de synchronisation
  Si (égal X_i marquée)
  alors (affecte SYNC marquée) et (affecte B_i 1)

- **Règle 2** : Prise de synchronisation
  Si (égal SYNC marquée) et (égal X_j marquée)
  alors (affecte SYNC non-marquée) et (affecte B_j 1)

De façon symétrique,

- **Règle 3** : Accusé de réception
  Si (égal X_u marquée)
  alors (affecte ACK marquée) et (affecte B_u 1)

- **Règle 4** : Consommation de l'accusé de réception
  Si (égal ACK marquée) et (égal X_t marquée)
  alors (affecte ACK non-marquée) et (affecte B_t 1)

Les variables binaires SYNC et ACK sont purement locales au niveau hiérarchique et ne sont donc pas modélisées au niveau de la commande.

I.5 - Les conflits et indéterminismes

I.5.1 - Introduction

Il s'agit ici de régler non seulement les conflits inhérents à la structure du graphe (ce sont les conflits structurels [BRA 83]), mais également les "conflits dynamiques" n'incombant pas à la structure du graphe mais détectés par l'interprétation du fonctionnement du système.

La définition des conflits structurels inclue les conflits effectifs qui font appel à la notion de marquage et correspondent à un choix exclusif entre deux franchissements. Leur

détection s'effectue automatiquement par scrutation systématique du graphe de commande [BOU 88]. Cependant, il est possible que certains conflits structurels détectés ne deviennent jamais effectifs lors de l'évolution du graphe. En effet, le marquage donnant lieu à ces conflits effectifs peut correspondre à un état que le système n'atteindra jamais au cours de son évolution, compte tenu par exemple de l'ordonnancement des pièces, des invariants du système, ... La simulation, par contre, donnera comme résultats les seuls conflits effectifs se produisant lors de l'évolution du graphe de commande. Le simulateur [CAS 87] s'arrêtera et affichera alors la liste des transitions en conflit et le marquage mis en cause. Malheureusement, cette seconde méthode plus restrictive, ne peut jamais prouver la mise en évidence d'une liste exhaustive de tous les conflits du système.

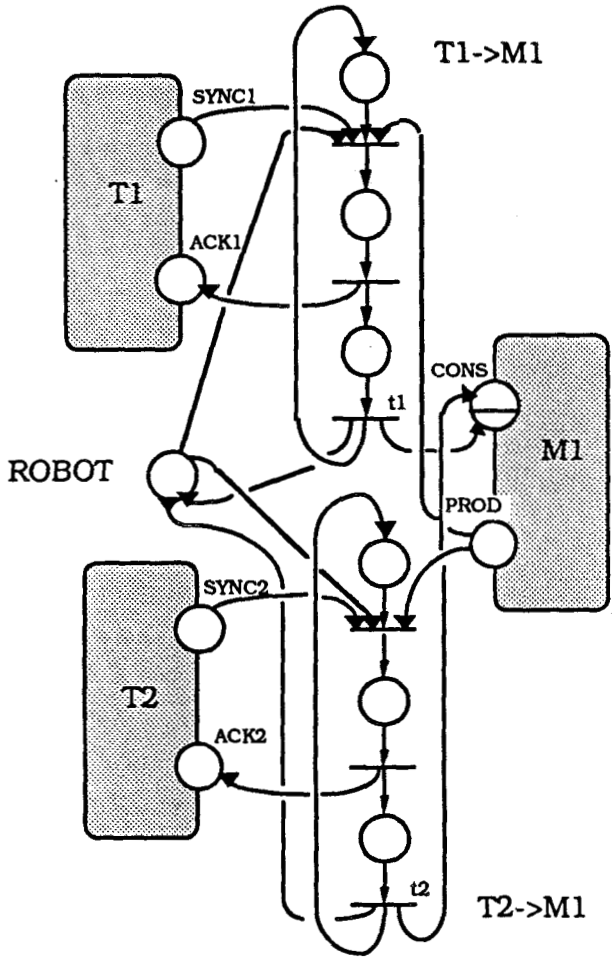
L'étude des conflits et surtout leur interprétation permet de les regrouper fonctionnellement en deux classes distinctes qui se différencient par la méthode utilisée pour les résoudre :

- accès à des ressources exclusives (robot, tampon, ...)
- indéterminismes directionnels.

Nous allons détailler la résolution de ces deux classes de conflits.

I.5.2 - Accès à des ressources exclusives

L'exemple présenté au § I.2, concernant un robot assurant deux transferts de manière exclusive, est l'exemple type du conflit effectif impliquant une ressource exclusive (le robot). Si nous modifions maintenant cet exemple en supprimant la machine M2, nous obtenons le graphe de commande suivant :



Configuration avec une machine

FIGURE 16

Faisant cela, nous introduisons un nouveau conflit qui n'est pas un conflit structurel mais un conflit dynamique. En effet, la définition du conflit structurel est la suivante :

deux transitions t1 et t2 sont en ***conflit structurel*** *si et seulement si elles ont au moins une place commune en entrée*

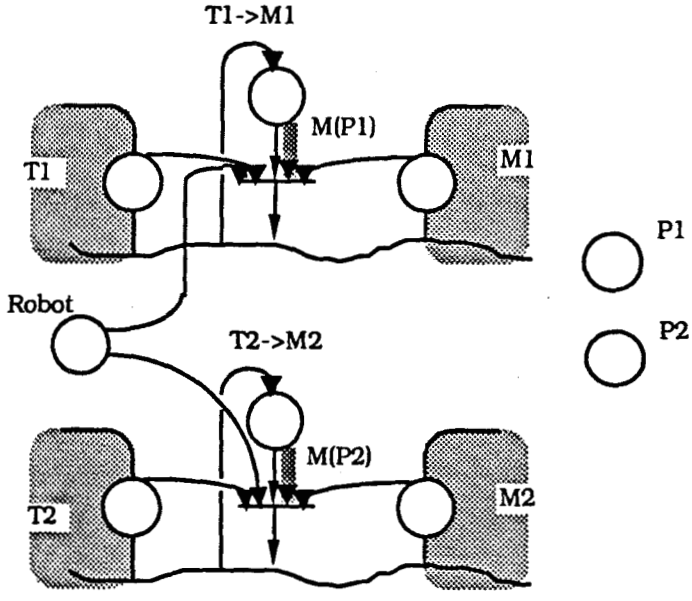
ce n'est pas le cas ici. Par contre, nous obtenons la structure : deux transitions en amont d'une place unique qui n'est conflictuelle que par l'interprétation qu'on lui donne. Le tampon d'entrée de la machine M1 n'est pas accessible simultanément par les deux process de transfert. En fait, si l'on analyse plus finement le problème, on remarque que le robot utilisé comme ressource exclusive par l'un ou l'autre des deux process de transfert protège implicitement, par l'invariant de marquage qui en découle, le tampon d'entrée de la machine M1.

Un seul transfert à la fois peut être effectué ; le conflit dynamique détecté ne deviendra jamais effectif. Dans l'hypothèse où deux robots seraient disponibles, nous obtenons par contre un conflit effectif qu'il faudra régler !

a) Interfaçage avec la partie commande

L'interfaçage entre le niveau hiérarchique et la partie commande modélisée en réseau de Petri structuré adaptatif et coloré (donc hors problème d'implantation), est actuellement réalisé par un ensemble de couples (arcs adaptatifs, place d'interface) permettant de bloquer ou non l'évolution de certains sous-ensembles du graphe développé en fonction du déclenchement des règles du niveau hiérarchique. Cette modélisation a été rendue nécessaire afin de rendre décidable l'évolution du graphe de commande et de permettre ainsi sa simulation.

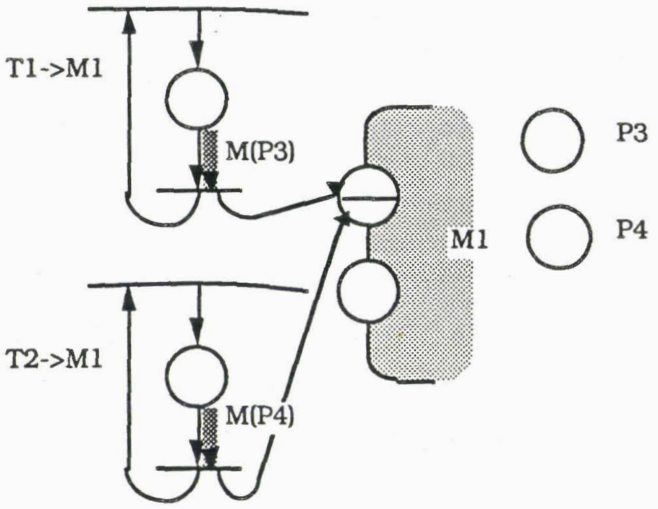
La modélisation de l'attribution de la ressource dans l'exemple du § I.2 devient la suivante :



Modélisation d'une ressource

FIGURE 17

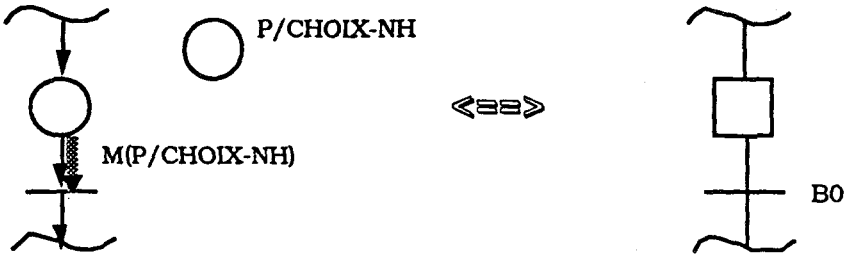
De même, la résolution du conflit d'accès exclusif au tampon d'entrée de la machine M1 présentée Figure 16, donnera de façon analogue le graphe suivant :



Accès au tampon d'entrée

FIGURE 18

Le marquage des places d'interface est conditionné par l'observation de l'état du graphe de commande et le résultat dépend des stratégies modélisées au niveau hiérarchique. Cette standardisation dans l'interfaçage nous permet d'automatiser la transcription dans le cadre de l'implantation répartie des graphes de commande. A chaque arc adaptatif conditionnant l'évolution d'un graphe, nous associons en fait une variable binaire paramétrant la réceptivité de la transition juste en aval de cet arc (Fig. 19). C'est cette technique qui a été utilisée pour gérer les différentes primitives de communication présentées précédemment. A chaque fois que le niveau hiérarchique doit être sollicité pour permettre de décider de l'évolution d'un graphe, une réceptivité est paramétrée par un booléen initialement faux dont la valeur est conditionnée non seulement par l'état des graphes de commande, mais encore par les règles modélisées au niveau hiérarchique.



Analogie entre arc adaptatif et bit sur réceptivité

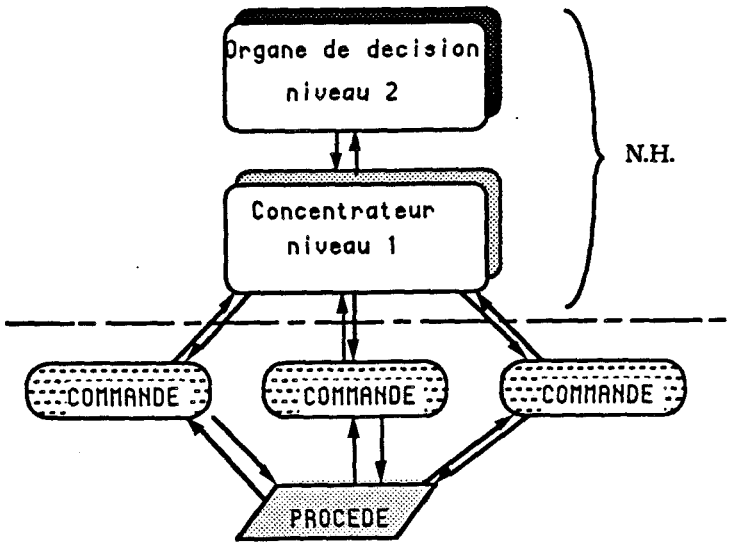
FIGURE 19

Les règles utilisées au sein du simulateur utilisent les différentes informations caractérisant l'état général du système (marquage des graphes, coloration, ...) pour permettre de décider ou non s'il faut marquer telle ou telle autre place d'interface. Le niveau hiérarchique utilisé pour l'implantation effective procède de façon strictement analogue. Le marquage des Grafcets, la couleur véhiculée par des mots servent à déterminer les actions modifiant la commande afin de lui permettre d'évoluer. Cela ne se traduit pas par le marquage de places d'interface, mais par la modification de la valeur booléenne de bits conditionnant la réceptivité des transitions "critiques". Cette analogie dans les fonctionnalités utilisées par le niveau hiérarchique mis en œuvre dans le simulateur et celui employé pour l'implantation réelle autorise une transcription systématique des règles pour passer d'un modèle à l'autre. Ce faisant, les erreurs sont minimisées et le fonctionnement validé lors de la simulation de manière off-line sera bien celui obtenu après implantation on-line. Ainsi, l'utilisation du modèle réseau de Petri structuré adaptatif et coloré traduit automatiquement en Grafcet et la transcription pratiquement textuelle des règles du niveau hiérarchique assurent une cohérence globale à toutes les phases de réalisation (de la conception à l'implantation) qui ne peut donc que minimiser les erreurs au niveau du produit final.

b) Structuration du niveau hiérarchique

L'analyse des problèmes évoqués pour la résolution des conflits nous amène à la réflexion suivante. S'il est fondamentalement nécessaire de prévoir la possibilité d'accès simultané à une ressource critique, il n'en demeure pas moins vrai que cette éventualité est la plupart du temps statistiquement fort peu probable. La limite de ce raisonnement permet d'affirmer que dans le cas où l'évolution du graphe de commande se fait en un temps quasi-

ment nul (de l'ordre de 10 ms pour les automates industriels), l'occurence que deux process asynchrones se trouvent dans un état conflictuel (demande **simultanée** d'une même ressource) est quasiment improbable. Cette constatation nous conduit à décomposer le niveau hiérarchique en deux modules distincts aux fonctionnalités séparées (Fig. 20). Le premier module est en fait un concentrateur qui gère l'attribution des ressources quand il n'y a pas de conflit ; la décision est alors immédiate. Le second module intervient de manière plus spécifique pour résoudre les **conflits effectifs**. Pour cela, il dispose d'heuristiques lui permettant de décider l'attribution en fonction des stratégies définies par l'utilisateur (priorité fixe, priorité tournante : Flip / Flop, ...).



Structuration du niveau hiérarchique

FIGURE 20

L'existence du concentrateur se justifie dans le cadre d'une implantation répartie. Dans ce cas, les deux graphes de commande se partageant la ressource exclusive n'ont aucune conscience de leur état relatif.

Une première possibilité d'implémentation du concentrateur réside dans l'utilisation d'un troisième automate "neutre" communiquant avec les autres et régissant l'attribution de la ressource en cas de non conflit. Cette technique reprend en grande partie la modélisation présentée pour implanter les primitives de communication dans le cadre d'une solu-

tion répartie exprimée en Grafcet ; elle en conserve donc toutes les lourdeurs quant à sa réalisation. La seconde reprend en fait le superviseur externe aux organes de commande en scindant sa base de règles en deux modules disjoints. Le premier module minimal (le concentrateur) aura pour fonction de décider au plus vite de l'attribution en cas de non-conflit ; le second, plus important en nombre de règles car décidant des stratégies d'attribution en cas de demande simultanée, sera utilisé sur requête du concentrateur. Cette structuration du niveau hiérarchique présente deux avantages principaux :

- classification des problèmes par niveau de complexité croissante (on va directement à l'encontre d'une mise à plat des difficultés),
- gain de performance car la majorité des problèmes est réglée au niveau du concentrateur.

I.5.3 - Les indéterminismes directionnels

Les impératifs de productivité concernant tout aussi bien :

- la réduction des stocks intermédiaires,
- la possibilité de répondre rapidement à un ordre de fabrication particulier,
- la gestion globale de l'ordonnancement suivant des techniques en temps réel [ROU 87],
- la diminution de la durée des produits et le nombre toujours plus important de leur variante,

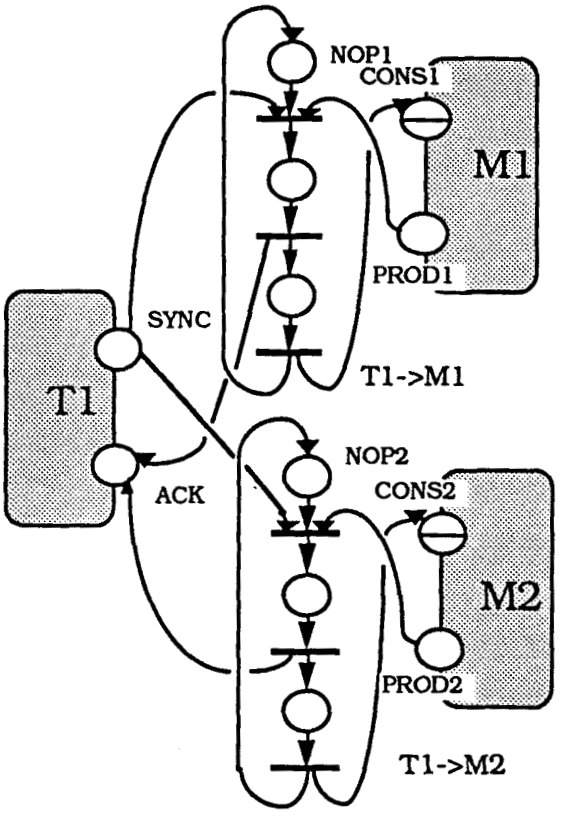
ont pour une grande partie transformé la nature des organes de production. Le développement des chaînes ou des lignes transfert automatisées, justifié pour les produits que le marché permet de fabriquer en grande série, nécessite des investissements que la petite ou moyenne série ne sauraient amortir. Cette orientation des besoins a conduit à la conception d'unités de production non spécialisées, dites "flexibles", susceptibles de mieux répondre à ces nouvelles exigences. Pour de telles unités, outre la complexité accrue des méthodes de gestion de production utilisées, la nature des problèmes d'auto-

matisation réagit fortement sur leur productivité.

La flexibilité, pour être correctement maîtrisée, doit être prise en compte dès le début de la conception de la commande et regroupe deux concepts complémentaires :

- la flexibilité de l'organe de production,
- la flexibilité dans les gammes opératoires.

La flexibilité de l'organe de production se traduit par l'utilisation de postes de fabrication (au sens large) dont les fonctionnalités multiples et paramétrables se recouvrent ou encore par l'emploi de plusieurs machines de même type. Par exemple, dans le cas de la configuration suivante : un tampon d'entrée délivre des pièces de même nature qui peuvent être usinées indifféremment sur deux machines concurrentes M1 ou M2. Le graphe de commande obtenu avec une démarche structurée est présentée Figure 21. Nous supposons dans ce cas que chaque processus de transfert dispose de son propre organe de transport.



Indéterminisme directionnel

FIGURE 21

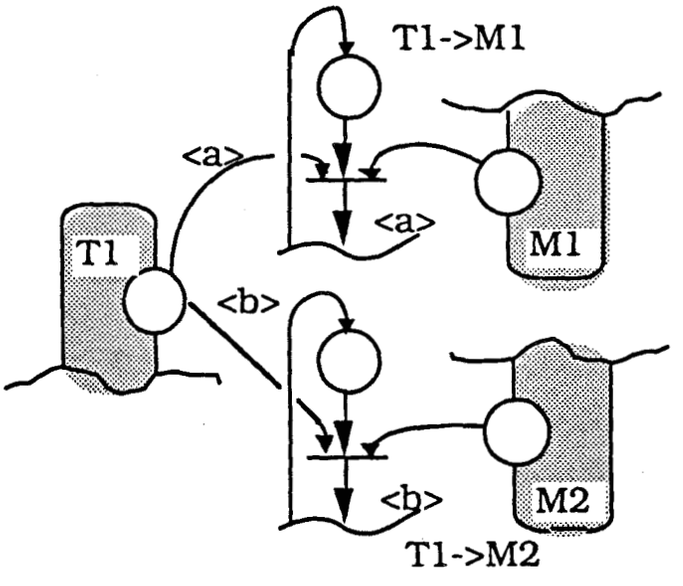
Les deux arcs partant de la place du tampon d'entrée (place SYNC) caractérisent la flexibilité du système. Cela se traduit par un indéterminisme directionnel. Ce conflit est différent, par son interprétation, de celui déjà décrit d'un accès à une ressource. L'accès simultané à une ressource met en compétition deux process qui sollicitent le même organe de transport, un tampon de stockage, ... L'attribution de cette ressource se fait forcément au détriment de l'autre. Dans le cas d'un indéterminisme directionnel, la commande a le choix entre plusieurs possibilités d'évolution. Retenir une solution plutôt que l'autre ne se fait pas au détriment de l'évolution du système, mais peut permettre par ailleurs d'optimiser les performances dans le cadre d'une stratégie globale d'ordonnancement.

Nous avons présenté un exemple de flexibilité concernant une pièce produite par le système. Nous aurions pu de façon analogue, présenter des organes du système qui sont eux-mêmes flexibles : chariots filoguidés, convoyeur avec aiguillages où plusieurs possibi-

lités de routage sont à chaque fois permises, ...

La flexibilité dans les gammes opératoires [BOU 84] [CAM 87] nous ramène exactement à la même nature de problèmes au niveau modélisation. L'indéterminisme se traduira au niveau de la coloration par des domaines de couleur non-disjoints étiquetant des arcs d'une même place. Reprenons l'exemple précédent en le modifiant sensiblement. Les machines M1 et M2 correspondent à des opérations d'usinage différentes ; à chaque opération, on associe la couleur m1 ou m2. Le suffixe "-" correspond à une opération à effectuer, le suffixe "+" à une opération déjà réalisée. La gamme à produire est la gamme m1m2 sachant que l'ordre des opérations est indifférent.

Nous obtenons alors le graphe suivant où l'indéterminisme deviendra effectif dans le cas d'une pièce de couleur $m1^{-}m2^{-}$ qui appartient aussi bien au domaine de a que celui de b.



avec
dom(<a>) = {m1-m2-,m1-m2+}
dom(<b>) = {m1-m2-,m1+m2-}

Indéterminisme au niveau des gammes

FIGURE 22

La résolution de ces indéterminismes va conditionner pour une part la productivité globale de la cellule. Le niveau hiérarchique va utiliser des algorithmes ou des heuristiques qui se différencieront par leur complexité de mise en œuvre et donc par leur coût en temps

de réponse. Ce problème peut très bien s'apparenter à celui de jeux (contexte d'opposition des joueurs) [LAU 86] où à partir de l'état courant du système, on cherche à construire une arborescence des états atteignables avec une profondeur plus ou moins limitée. Le résultat de cette recherche permet de choisir l'état "futur" le plus intéressant suivant différents critères (taux d'engagement prévisionnel des machines, nombre de pièces produites pendant la recherche, ...) et donc de lever l'indéterminisme.

Néanmoins, ces techniques sont ici très difficiles à mettre en œuvre car elles impliquent la construction dynamique de graphes d'état pour des process concurrents, asynchrones, non autonomes et fortement connectés au procédé.

Une seconde solution consiste à utiliser un "joueur" de réseau de Petri (ou de Grafcet) [SAH 87] qui, connecté à la commande, pourra dans les cas de conflit, anticiper l'évolution du système et ainsi déterminer la solution à retenir pour le faire évoluer. Une méthode analogue utilisera les résultats de simulations effectuées de manière off-line. On lance le simulateur à partir du marquage non déterministe et, en fonction des données statistiques collectées, le concepteur peut concevoir et valider la stratégie à implanter au niveau hiérarchique.

Enfin, il ne faut pas négliger non plus, le savoir-faire de l'expert qui peut très bien décider de façon pragmatique, d'utiliser des règles arbitraires dont l'emploi peut donner des résultats en terme de productivité comparables à des solutions bien plus sophistiquées, donc plus onéreuses.

Levons simplement à partir de l'exemple présenté Figure 21, l'indéterminisme directionnel.

Les performances de la machines M1 sont, par hypothèse, 30 % supérieures à celles de la machine M2. L'expert décide donc de toujours affecter en priorité l'usinage sur la machine M1. La définition des trois règles suivantes est alors suffisante pour régler le problème.

- **Règle 1** : Attribution à la machine M1

Si (égal SYNC marquée) et (différent PROD1 Ø) et (égal NOP1 marquée)
alors (affecte SYNC non-marquée) et (ajoute PROD1-1) et (affecte B11)*

- **Règle 2** : Attribution à la machine M2 si la machine M1 est déjà en train de produire

Si (égal SYNC marquée) et (différent PROD2 Ø)
et (égal NOP2 marquée) et (**égal NOP1 non-marquée**)
alors (affecte SYNC non-marquée) et (ajoute PROD2-1) et (affecte B21)*

- **Règle 3** : Attribution à la machine M2 si le tampon d'entrée de la machine M1 est plein

Si (égal SYNC marquée) et (différent PROD2 Ø)
et (égal NOP2 marquée) et (**égal PROD1 Ø**)
alors (affecte SYNC non-marquée) et (ajoute PROD2-1) et (affecte B21)*

*Remarque : Le bit B1 (respectivement B2) conditionne la réceptivité de la transition en aval de la place NOP1 (resp. NOP2)

I.6 - Les blocages

Le concept de blocage est issu de la propriété de vivacité des réseaux de Petri dont l'étude dépend non seulement de la structure du graphe mais également et essentiellement du marquage initial. L'approche structurée adoptée lors de la conception du graphe de commande facilite la validation du réseau. Les process utilisés sont en effet, saufs, vivants et réinitialisables par construction. La définition des structures de liaison permet de garantir le caractère borné du graphe et limite la recherche des invariants. Par contre, la propriété de vivacité ne peut être assurée lors de l'ajout de structures de liaison entre les différents process.

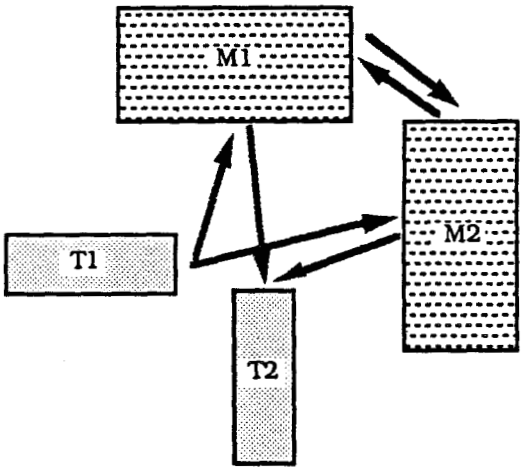
Approfondissons l'exemple présenté au sujet de l'indéterminisme directionnel (Fig. 22) et montrons comment le système peut arriver dans une situation bloquante. La définition du système complet est la suivante (Fig. 23) :

- un tampon d'entrée (T1),
- deux machines différentes sans tampon d'entrée / sortie assurant

chacune une opération spécifique notée m1 ou m2,

- un tampon de sortie (T2).

Les gammes opératoires assurées par le système sont : m1, m2, m1m2 et m2m1 ; elles ne sont pas flexibles par hypothèse.



Schématisation du système

FIGURE 23

La modélisation du système nécessite la définition de 6 process de transfert assurant la circulation générale au sein des organes de production ; chaque processus est ici représenté par une flèche orientée reliant deux zones de l'installation. L'évolution du système, tributaire de la séquence d'entrées des pièces présentées au tampon T1, amène la configuration suivante :

- la machine M1 vient d'usiner une pièce qui doit passer maintenant sur la machine M2 (marquage $m1^{+}m2^{-}$),
- la machine M2 se trouve dans une situation symétrique (marquage $m2^{+}m1^{-}$).

Cette configuration est bloquante car le transfert M1 → M2 ne peut se faire que si M2 est vide ; réciproquement, le transfert M2 → M1 nécessite la disponibilité de M1.

Les outils théoriques (étude du graphe de couverture, méthodes d'algèbre linéaire)

sont la plupart du temps inadaptés ou même insuffisants pour prouver le "bon fonctionnement" du graphe de commande. Seule la simulation [GIR 84][CAS 87] permet de mettre en évidence certains comportements bloquants. Par contre, elle ne se substitue pas à une preuve formelle et ne peut donc prétendre avoir révélé toutes les situations bloquantes.

L'emploi d'un superviseur pour résoudre les blocages de l'installation est alors obligatoire. En effet, dans le cadre d'une implantation répartie, les différents graphes de commande résident sur plusieurs automates, chacun n'ayant qu'une vision partielle du fonctionnement de l'installation. Ainsi, le processus de transfert du tampon d'entrée T1 vers la machine M2 peut très bien ne pas connaître l'état de la machine M1 ; sans concertation, ni vérification globale, le service rendu aura pour effet d'amener rapidement le système dans une situation de deadlock. Par contre, vérifier dynamiquement l'évolution du marquage en des points critiques du système (ici, l'état du tampon d'entrée et des machines M1 et M2) par l'intermédiaire du niveau hiérarchique permet d'empêcher un blocage détecté au préalable par la simulation, grâce à une perception cohérente de l'état de l'installation. De plus, le niveau hiérarchique pour peu qu'il dispose d'une entrée interactive, autorise aisément l'ajout de nouvelles règles après analyse et interprétation, si un blocage inconnu apparaît, sans pour autant remettre en cause la définition des graphes de commande, ni leur implantation. Cette démarche est logique car la commande est définie pour assurer le bon séquencement des automatismes ; elle est validée par la simulation en montrant que les gammes opératoires définies dans le cahier des charges sont effectivement réalisées. Par contre, aucun outil ne permet actuellement de prouver qu'en régime dynamique (les régimes transitoires sont encore plus difficiles à appréhender), elle ne rencontrera jamais une séquence d'entrée provoquant une situation bloquante. La conception de la commande n'est pas à remettre en cause. Il est donc naturel que la circulation dynamique des entités du système soit supervisée par le niveau hiérarchique. En cas de problème, c'est la définition de ses propres stratégies qui doit être revue et non celles de la commande. Il apparaît ici un résultat essentiel de la méthodologie proposée tant au niveau conception qu'implantation du fait du découplage des 2 niveaux : commande et hiérarchique.

Dans l'exemple présenté précédemment, la résolution des blocages est immédiate. Le niveau hiérarchique doit interdire l'entrée d'une pièce $mi^{-}mj^{-}$ tant que la machine

Mj usine une pièce du type mj mi$^-$ (pour i = 1, j = 2 ou i = 2, j = 1). Cela se traduit par deux règles régulant les entrées de pièces au niveau du tampon T1.

- **Règle 1** : Autorisation d'entrée pour la couleur $m1^-m2^-$

  Si (égal couleur-sur-T1 $m1^-m2^-$) et (différent couleur-sur-M2 $m2^-m1^-$)
  et (différent couleur-sur-M2 $m2^+m1^-$)
  alors "autoriser transfert vers M1"

- **Règle 2** : Autorisation d'entrée pour la couleur $m2^-m1^-$

  Si (égal couleur-sur-T1 $m2^-m1^-$) et (différent couleur-sur-M1 $m1^-m2^-$)
  et (différent couleur-sur-M1 $m1^+m2^-$)
  alors "autoriser transfert vers M2"

La structure locale de la commande répartie n'est en rien modifiée par ces règles.

I.7 - Conclusion

Au cours de cette partie, nous avons présenté et justifié l'utilisation du niveau hiérarchique dans le cadre d'une implantation répartie du graphe de commande d'une cellule de production flexible pour un fonctionnement en mode de marche normal. Nous avons ainsi montré comment la définition du niveau hiérarchique permet de structurer la démarche utilisée pour l'implantation :

- programmation en premier lieu des process commandant les automatismes séquentiels,
- mise en œuvre des primitives de communication banalisées.

Cette implantation structurée a été rendue efficacement possible parce que la démarche complète partant du cahier des charges jusqu'à l'obtention du modèle, est **structurée** et qu'elle conduit en fait à un réel découplage des différentes fonctionnalités du système de production.

Enfin, le niveau hiérarchique est à même de restituer le caractère de **flexibilité** inhé-

rent au modèle réseau de Petri structuré adaptatif et coloré. La transcription de ce dernier en Grafcet, en reportant l'interprétation de certaines transitions au niveau supérieur, a permis de conserver toute la flexibilité du modèle. De plus, le niveau hiérarchique, par sa délocalisation, n'est en aucun cas figé et évoluera en fonction de l'expérience acquise au cours de la production. Il pourra intégrer à terme d'éventuelles modifications compatibles avec la flexibilité potentielle de l'ensemble commande / machines de production.

II - SURETE DE FONCTIONNEMENT

II.1 - Introduction

Actuellement, tout automatisme se doit d'offrir des souplesses d'utilisation afin de minimiser les temps d'arrêt machines et de faciliter tout redémarrage et tout changement de mode de marche souvent générateur de perturbation dans le fonctionnement de l'application. La complexité des systèmes est devenu telle qu'une surveillance par un opérateur seul n'est plus suffisante. Il est donc désormais impératif d'intégrer des outils de surveillance et des mécanismes de recouvrement, dès la phase de conception, à tous les niveaux de l'installation :

- procédé,
- commande,
- niveau hiérarchique.

Le procédé est l'élément le plus sensible de la chaîne automatisée. C'est de lui que proviendront en grande partie, les anomalies dans le fonctionnement de l'installation. La détection de ces anomalies peut être envisagée de deux façons :

- détection directe par redondance de capteurs et utilisation de capteurs de sécurité,
- détection indirecte par utilisation de modèles régissant les interactions des diverses variables du procédé.

La commande n'est pas dans le cas présent, susceptible d'engendrer des anomalies de fonctionnement. Par contre, elle doit intégrer comme le procédé, et plus encore, des outils permettant la surveillance de l'installation. De plus, cette commande, établie afin de permettre le pilotage d'une production flexible, doit elle-même, être flexible pour accepter les changements de mode de marche ainsi que les modes dégradés de fonctionnement.

Le niveau hiérarchique quant à lui, en fonction du contexte restreint communiqué par le système de commande, analysera les conditions provoquant le passage d'un mode de

production à un autre. C'est lui qui va paramétrer le fonctionnement de la commande en utilisant sa flexibilité.

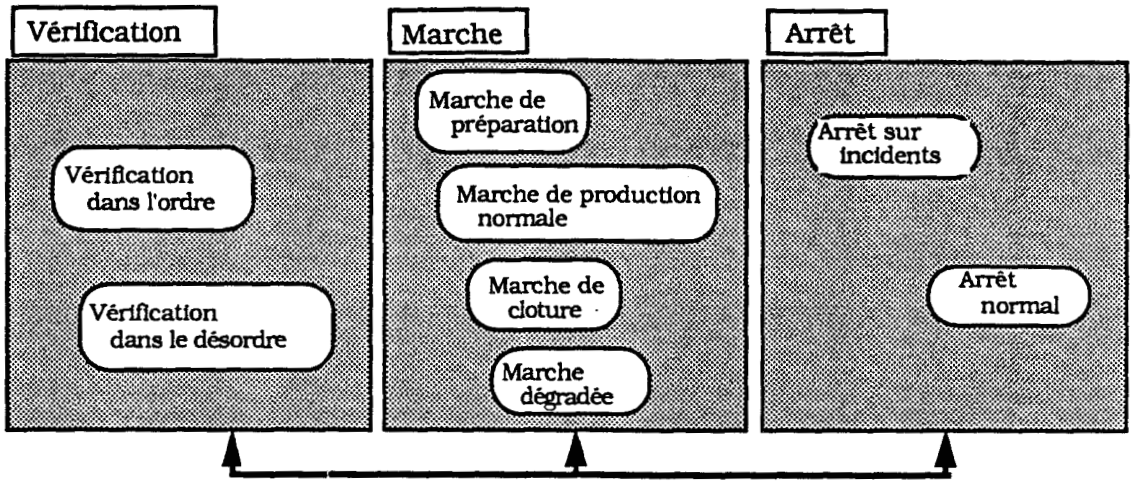
Dans un premier temps, nous allons définir le vocabulaire utilisé et présenter les différents modes de marche et d'arrêt pris en compte lors de la conception. Ensuite, nous présenterons différents outils intégrés dans la commande autorisant une surveillance directe. Enfin, nous montrerons comment l'utilisation du niveau hiérarchique et la définition d'interfaces avec la commande permettent de rendre cette dernière flexible et favorisent l'intégration des différents modes de marche.

II.2 - Définitions

Il n'existe pas, à proprement parler, de méthodologie de surveillance. L' A. D. E. P. A. propose un "outil méthode", le GEMMA : Guide pour l'Etude des Modes de Marches et d'Arrêts. Ce dernier est encore peu utilisé, mais il apporte de nouveaux concepts, définit un vocabulaire précis et propose une visualisation. Ce guide permet d'effectuer :

- le recensement des différents modes envisagés,
- la détermination des conditions provoquant le passage d'un mode à l'autre.

Il ne permet pas, par contre, de passer à la phase de réalisation ; c'est-à-dire écriture en clair sous forme d'organigrammes de l'analyse, puis transcription en langage automate. Etablie à partir du guide "GEMMA", la grille suivante retient les principaux modes de marches et d'arrêts et se veut volontairement simplifiée.



"Gemma simplifié"

FIGURE 24

Trois grandes familles de type de fonctionnement différent sont à distinguer :

- vérification,
- marche ou production,
- arrêt.

Une fois les modes retenus, il faut alors détailler l'analyse jusqu'à définir les causes voulues ou subies provoquant le passage d'un mode à un autre. En fonction des causes, le mode en cours peut être abandonné pour passer dans un autre mode ; il y a donc action sur le déroulement du traitement séquentiel. Le vocabulaire utilisé est celui qui figure dans le guide "GEMMA". La définition des principaux termes employés est rappelé ici :

- *Marche de production normale* : c'est le fonctionnement normal. C'est l'état pour lequel l'automatisme a été conçu. A ce fonctionnement est associé le Grafcet de base.

- *Marche de préparation* : c'est une marche préparatoire à la marche de production normale. C'est une marche transitoire.

- *Marche de clôture* : comme la marche de préparation, c'est une marche transitoire qui peut être nécessaire en fin de campagne, de série, de journée,

dans le but de "vider" certaines machines, ou sous-ensembles.

- *Marche dégradée* : elle consiste à arrêter partiellement le fonctionnement du procédé. La production après une défaillance de la machine, est assurée soit par le forçage de certaines informations qui influent sur le comportement de la commande, soit par l'intervention des opérateurs.

- *Arrêt normal* : de différents types, il n'est pas la conséquence d'une défaillance. L'arrêt peut s'effectuer :
  - dans l'état initial,
  - en fin de cycle,
  - dans un état déterminé.

- *Arrêt sur défaut* : c'est un arrêt subi, suite à une défaillance. L'arrêt, très souvent, est effectif après des cycles de dégagement ou après des procédures limitant les conséquences dues à la défaillance.

- *Vérification dans l'ordre* : le cycle de production est analysé par l'opérateur sous-ensemble par sous-ensemble ou séquence par séquence.

- *Vérification dans le désordre* : le contrôle de certaines fonctions ou de certains mouvements est effectué sans respecter l'ordre du cycle.

Parallèlement à la définition des modes de marches et d'arrêts, nous utiliserons la terminologie, communément admise, définie dans [LAP 85]. L'orsqu'une **faute** survient ou est commise, il y a création d'une **erreur latente** qui deviendra **effective** lorsqu'elle sera activée ; lorsque l'erreur affecte le service délivré, elle devient une **défaillance**. Autrement dit, une erreur est la manifestation d'une ou plusieurs fautes dans le système, et une défaillance la manifestation d'une ou plusieurs erreurs sur le service.

II.3 - La surveillance interne

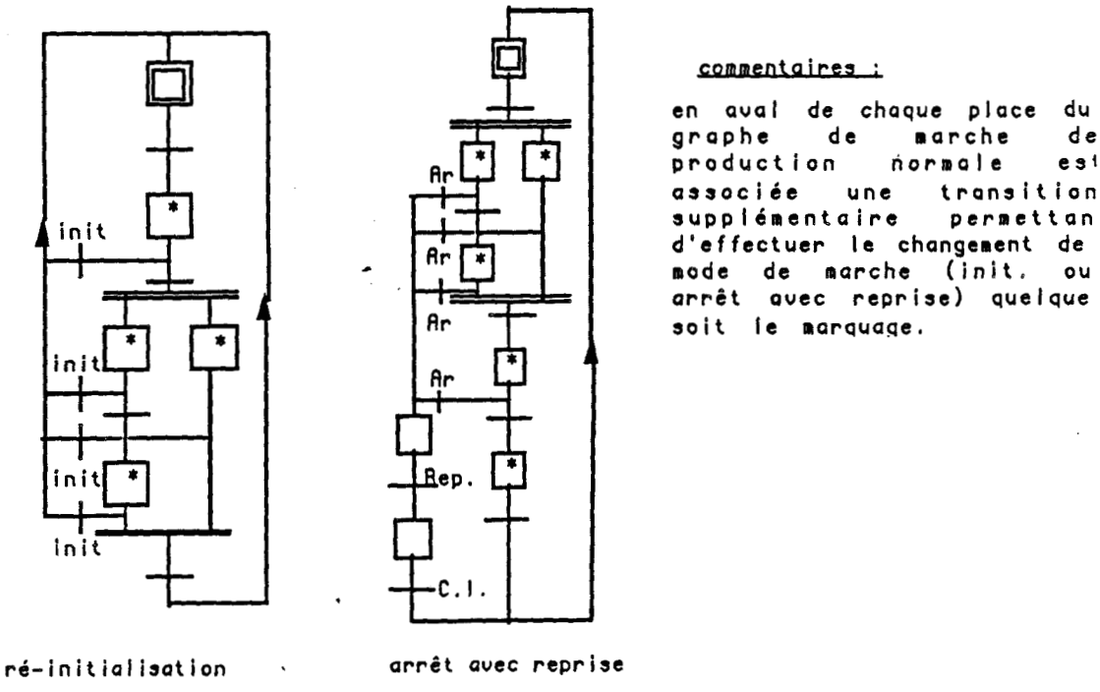
L'apparition d'une défaillance dans le fonctionnement de l'installation va nécessiter la modification dynamique du graphe de commande afin de lui permettre de passer dans un mode de marche dégradé. Cette modification dynamique ne peut consister en une reprogrammation des organes de commande car cela aurait pour effet d'arrêter obligatoirement l'installation. Il en résulterait une perte conséquente aussi bien de temps que d'argent. Il faut donc prendre en compte, si l'on désire conserver une modélisation unique, dès la phase de programmation, les différents modes de marches et surtout prévoir explicitement à l'aide du modèle implanté, l'évolution vers une situation définie quelle que soit la situation en cours.

Les deux exemples ci-dessous traduisent de tels comportements (Fig. 25). L'initialisation ou l'arrêt avec reprise sont tous deux imprévisibles. Il est donc nécessaire d'associer à chaque étape une transition supplémentaire permettant d'effectuer le changement de mode de marche quelque soit le marquage courant des graphes. De telles représentations deviennent rapidement très lourdes, voire inextricables lorsque plusieurs évolutions sont possibles par des combinaisons d'évènements. De plus, l'implantation répartie de la commande, rendue nécessaire par l'importance des installations pilotées, rend le problème encore plus complexe. En effet, la casse d'un outil sur une machine doit bien évidemment être prise en compte par la commande de cette machine. Mais la défaillance qui en résulte affecte le service délivré. Il doit donc y avoir nécessairement répercussion de cette panne sur d'autres organes de commande :

- modification dans la gestion des organes de transport pour arriver à court-circuiter la machine si cela est possible,
- modification dans la nature des pièces introduites dans la cellule pour supprimer la gamme usinée sur la machine inutilisable.

Ces modifications concernent chacun des organes de commande séparés qui n'ont qu'une vision locale de l'installation. Vouloir synchroniser l'arrêt sur défaut de la machine avec le passage en marche dégragée de l'organe de transport et de l'unité d'approvisionnement des pièces oblige très rapidement à définir des graphes de commande dont la taille peut largement dépasser celle des graphes de commande dédiés à la marche de produc-

tion normale. En procédant de cette façon, nous aboutissons en effet à une mise à plat des différents problèmes : la modélisation des différents modes de fonctionnement (marche normale et dégradée) est effectuée sur la même représentation graphique qui intègre toutes les transitions d'un mode à l'autre. Le très important degré de connexité entre les différents modes conduit à la définition d'un modèle graphique peu explicite devenant à l'occasion illisible.



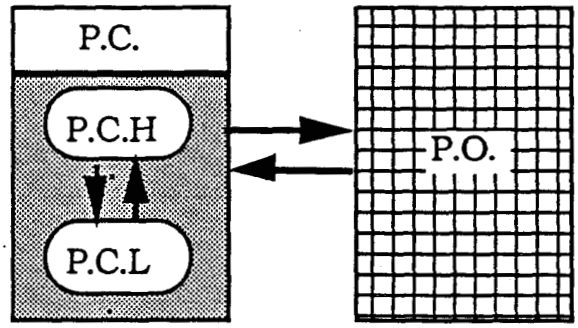
Initialisation ou arrêt

FIGURE 25

Aussi avons-nous préféré définir de tels comportements par l'intermédiaire d'une représentation hiérarchisée de la partie commande. La représentation la plus simple est la suivante [BOU 87] (Fig. 26) :

- un niveau local définissant la partie commande de process traitant les informations issues du procédé pour élaborer des ordres vers ceux-ci (Partie Commande Locale, notée P. C. L.),
- un niveau supérieur formant la Partie Commande Hiérarchisée (notée

P. C. H.) supervisant la partie commande locale et pouvant ainsi à tout moment, paramétrer son fonctionnement.

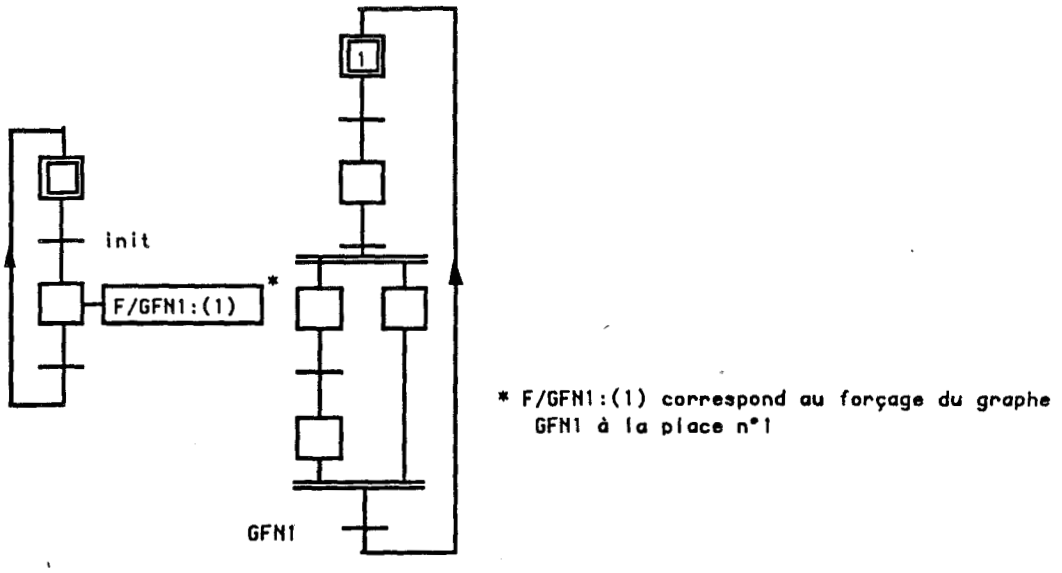


Description hiérarchisée de la commande

FIGURE 26

A la représentation hiérarchisée de la partie commande peut naturellement être associée une description par Grafcet séparé :

- un Grafcet de commande hiérarchique,
- un Grafcet de fonctionnement "normal" (Fig. 27).



Ré-initialisation avec commande hiérarchisée

FIGURE 27

Ce principe de description allège très fortement le graphisme mais ne prend une signification que si la structure de la partie commande locale (le Grafcet de fonctionnement "normal") est complètement précisée. Autrement dit, le Grafcet de commande hiérarchisée n'est utilisable que si les actions qu'il modélise sont implémentables au niveau de la commande (ordre de ré-initialisation, figeage, ...). A ce stade de la réflexion, il apparaît que l'utilisation du seul modèle Grafcet pour implanter une décomposition hiérarchique de la commande n'est pas nécessairement adaptée à tous les problèmes rencontrés dans l'industrie, notamment si l'on prend en compte les questions de taille, d'interactivité et donc de flexibilité. L'utilisation d'un modèle unique reste bien sûr un avantage indéniable pour plusieurs raisons. L'homogénéïté de la description permet d'utiliser les mêmes organes de commande (les automates) pour la mise en œuvre effective. Il en résulte que l'interfaçage entre les différents modèles de fonctionnement ne pose pas de problème particulier.

Néanmoins, deux points fondamentaux sont occultés dans cette démarche :

- Le concept de commande répartie impose à nouveau un découpage fonctionnel figé de la commande hiérarchique, ce qui restreint son pouvoir de décision aux potentialités limitées du niveau local où elle est implantée.

- La programmation des stratégies de pilotage et des changements de modes de marches est limitée par l'utilisation d'un langage impératif offrant tout au plus la notion de sous-programme. Peu d'interactivité est ici offerte à l'utilisateur, ce qui nuira à la flexibilité de sa description.

Ces limitations nous amènent à la définition d'une surveillance séparée dont nous allons maintenant justifier les principes.

II.4 - La surveillance séparée

II.4.1 - Interfaçage partie commande ↔ niveau hiérarchique

La réalisation d'une surveillance séparée (niveau hiérarchique implanté sur un calculateur différent de ceux utilisés par la commande) nécessite obligatoirement la définition d'un protocole de communication entre les différentes unités. En fait, l'observation et la modification de la commande peuvent être envisagées de trois façons [SAH 87] :

- accès direct,
- appel / réponse,
- réception continue.

Avec l'accès direct, l'observateur (au niveau hiérarchique) accède lui-même à tout ou partie des informations de la partie commande (marquage des places, valeur des informations internes : bits, mots, ...). C'est le mode "espion" qui doit présenter l'intérêt majeur de ne nécessiter aucune modification de la commande pour sa réalisation.

Le mode appel - réponse exige l'instauration d'une communication entre les deux parties. Il y a nécessairement, à un instant donné, synchronisation entre l'organe de surveillance et la commande, soit pour remonter un état à analyser, soit pour envoyer à la commande des ordres de paramétrage. Nous aboutissons à l'idée d'un mode "coopérant" où la nature des requêtes émanant du niveau hiérarchique est fixée a priori et est donc implicitement figée au niveau de la commande. Toute nouvelle requête (par exemple, modification de la valeur d'une variable d'un graphe) qui n'a pas été initialement prévue,

restera sans réponse de la part de la commande.

En dernier lieu, la réception continue est d'une certaine manière, la forme duale de l'accès direct. C'est le système observé qui envoie périodiquement les informations sur son état interne. Néanmoins, après analyse, si l'on souhaite modifier l'état de la commande, l'un des deux modes précédents devra être utilisé ; la réception continue, dans le cas présent où le "maître" est la commande, est uni-latérale et ne laisse donc aucune initiative au niveau hiérarchique.

Le mode appel - réponse est intéressant car relativement aisé à mettre en œuvre. Accéder ou modifier la commande ne pose aucun problème particulier car cela est fait par la commande elle-même après interprétation des messages issus du niveau hiérarchique. Par contre, la nature des requêtes est limitée (vu précédemment) et leur implémentation demande la réalisation d'une tâche, dédiée à la communication, mais intégrée à la commande. La réception continue, outre son aspect "sens-unique", présente l'inconvénient de noyer le niveau hiérarchique sous un flot d'informations, loin d'être toutes significatives à un instant donné. Il peut y avoir de ce fait un encombrement inutile du canal de communication qui peut éventuellement saturer les performances globales du système. Ces constatations nous amènent à retenir le mode accès direct comme moyen de communication.

Les principes essentiels de l'"accès direct" sont les suivants :

- le niveau hiérarchique doit accèder de façon complètement transparente aux informations des organes de commande,
- il doit pouvoir, de la même façon et à tout moment, les modifier.

Il en résulte que les évolutions respectives du niveau hiérarchique et des organes de commande sont complètement **asynchrones**. Certaines règles de "bon fonctionnement" doivent donc être respectées :

- Une information normalement fugitive (marquage d'une place, ...) doit impérativement être mémorisée s'il est nécessaire qu'elle soit perçue par le niveau hiérarchique. Dans le cas d'une place, par exemple, il suffit

d'imposer que la réceptivité de sa transition aval soit validée par le niveau hiérarchique pour s'assurer que son activité a bien été prise en compte (ce mécanisme a été implicitement utilisé précédemment pour la gestion des ressources, producteurs / consommateurs, ...).

- Plus généralement, l'asynchronisme total des deux niveaux ne permet pas de préjuger des temps de réaction de l'un par rapport à l'autre. Il convient alors d'envisager l'éventualité selon laquelle le niveau hiérarchique ralentirait l'évolution de la commande. Dans ce sens, nous devons limiter ces temps d'attente induits, à des délais de l'ordre de plusieurs secondes en considérant que dans une large mesure, les procédés de production en industrie manufacturière ont des constantes de temps compatibles avec ces délais (les opérations d'usinage ou d'assemblage ont généralement des temps de traitement de l'ordre de une à plusieurs minutes).

II.4.2 - Mécanismes d'interaction sur la commande

Trois mécanismes fondamentaux sont ici retenus, permettant d'influer sur le cycle d'évolution normal de la commande :

- le gel,
- la reprise,
- la configuration dynamique.

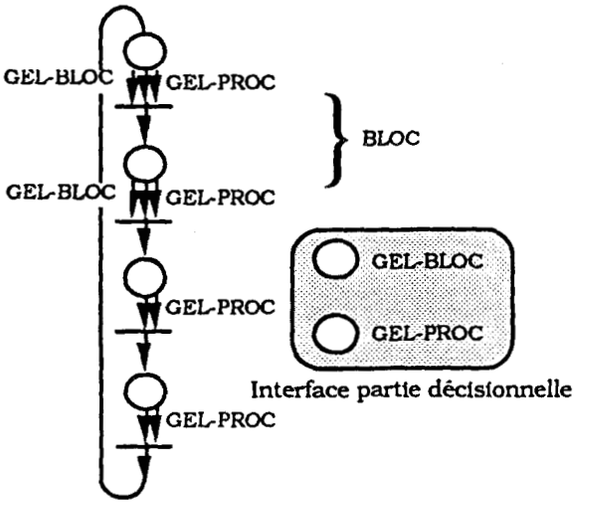
a) Le gel

Le gel permet le contrôle sélectif de l'évolution du graphe. Son objectif est d'empêcher qu'une erreur latente ne devienne effective ou, s'il est déjà trop tard, qu'elle ne se propage en engendrant une défaillance. Prenons l'exemple d'une commande défectueuse d'aiguillage affecté à l'introduction de pièces brutes dans le système de production. Une faute est survenue et a engendré une erreur latente. Le gel du processus régissant l'introduction des pièces va permettre de réparer l'aiguillage en question en bloquant momentanément l'amenée des pièces et empêchant ainsi que l'erreur latente ne devienne effective

ou ne déclenche une défaillance par propagation du blocage à d'autres tâches par communication ; mais là encore, l'erreur reste réversible.

L'utilisation d'arcs adaptatifs permet d'intégrer la notion de gel de bloc ou de processus entier dès la phase de modélisation initiale du système en réseau de Petri structuré adaptatif et coloré en évitant toute interprétation sémantique.

Exemple :

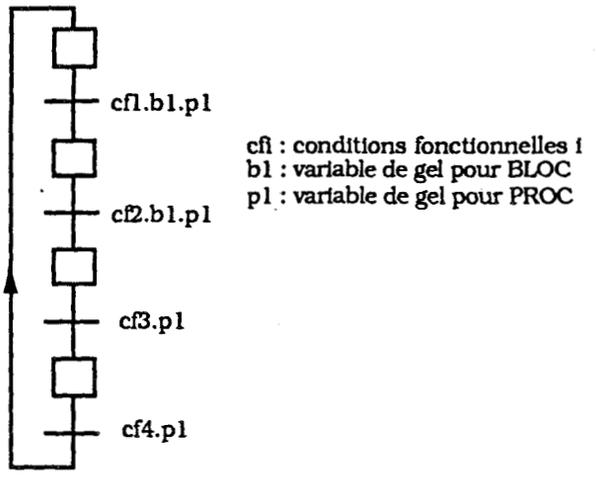


Modélisation du gel

FIGURE 28

Le gel du processus PROC est obtenu par l'introduction d'une marque dans la place GEL - PROC. Il permet d'interdire le tir de toute transition de ce processus quelqu'en soit le marquage courant. Le gel de BLOC permet d'empêcher toute évolution à l'intérieur de ce bloc alors que le fonctionnement à l'extérieur n'est pas modifié.

Lors de l'implantation en Grafcet, à chaque arc ou groupement d'arcs adaptatifs, seront associées une ou plusieurs variables booléennes conditionnant la réceptivité des transitions susceptibles d'être gelées. Un "ET" logique entre les conditions fonctionnelles normales de franchissement de la transition et la variable booléenne traduisant le concept de gel assurera la réceptivité globale de la transition (Fig. 29).



Implantation du gel en Grafcet

FIGURE 29

La valeur des variables booléennes de gel est conditionnée par le résultat de l'inférence des règles du niveau hiérarchique de façon strictement similaire à la détermination du marquage des places d'interface paramétrant le réseau de Petri adaptatif. Cette analogie entre le modèle de conception et le modèle d'implantation permet de conserver la même structure fonctionnelle (graphes et règles) qui a pu être validée préalablement par simulation. Les fautes de codage sont ainsi minimisées et la commande obtenue correspond bien au modèle conceptuel. Néanmoins, la simulation en réseau de Petri fait abstraction des actions associées aux places et dirigées vers le procédé. Le gel d'un processus réel devra par contre intégrer cette notion ; verrouiller l'évolution de la commande et figer dans l'état les actions courantes ne doivent être effectuées que si ces actions ne sont pas "critiques" vis à vis du procédé.

b) La reprise

Cette méthode a pour objectif de mettre le système dans un état antérieur à celui qui à déclenché une erreur. Elle est très importante car elle permet de spécifier et de mettre en œuvre la plus grande partie des stratégies de recouvrement d'erreurs. La difficulté vient ici de la nature des procédés pilotés. Si la commande d'un aiguillage est inopérante sur un convoyeur, l'erreur latente créée devient effective quand une palette prend la mauvaise

direction.

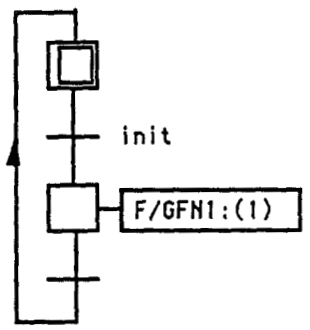
Assurer une reprise dans ce cas ne pose pas de problèmes particuliers : il suffit de repositionner la palette en amont de l'aiguillage défectueux (opération éventuellement manuelle si le convoyeur ne possède qu'un seul sens de marche) et de remettre la commande dans l'état précédent l'action sur le moteur de l'aiguillage. Par contre, la casse d'un outil lors d'un cycle d'usinage ne peut être traitée aussi simplement. Il y a alors nécessité d'interpréter l'état pour savoir si la nature des actions engagées est réversible ou non. Si la pièce est récupérable, il y a reprise juste en amont de la commande d'usinage après changement de l'outil ; sinon, la pièce est retirée de la production et la reprise consiste en fait à une réinitialisation partielle du processus d'usinage (au niveau de confinement de l'erreur).

L'implémentation de la reprise peut être envisagée de trois façons différentes :

- modélisation explicite en Grafcet de tous les points de reprise,
- utilisation d'ordres spécifiques correspondant à une extension du Grafcet,
- forçages du marquage à partir du niveau hiérarchique.

La modification dynamique et directe du marquage par le niveau hiérarchique n'est pas, pour le moment, opérationnelle dans notre système. Outre les problèmes techniques que cela soulevait, il n'a pas été jugé utile d'inter-agir avec le fonctionnement propre du scheduler des automates (cf. § I.2.4) pour des raisons de sécurité. La modélisation explicite des points de reprise a été présentée et analysée précédemment dans le cadre d'une surveillance interne (Fig. 25). Nous avons ainsi préféré retenir la solution intermédiaire qui consiste à utiliser au sein de la commande des ordres spécifiques de forçage du Grafcet. Ces ordres sont réalisés par la commande ; par contre, la décision de les déclencher incombe exclusivement à l'organe de supervision qui seul dispose d'une vision globale de l'état du système de production ainsi que d'une base de connaissances suffisante (règles de décision) pour déterminer la nature de la reprise à effectuer. La notion de point de vue est ici essentielle et rejoint en ce sens la nécessité d'une hiérarchie.

La modélisation de ces ordres au niveau de la commande peut très bien être réalisée par un Grafcet de commande hiérarchisée (cf. Fig. 26 et 27). La réceptivité des transitions est alors conditionnée par le niveau hiérarchique ou par l'utilisation d'un langage autre que le Grafcet (si l'automate le supporte) dont la validité des tests est également conditionnée par le niveau hiérarchique (Fig. 30).




```
10 ...
20 if (init =1) then F/GFN1:(1);
30 ...
```

Modélisation d'une reprise

FIGURE 30

Dans les deux cas, le niveau hiérarchique a par hypothèse accès au booléen "init" et décide de la valeur à lui attribuer pour déclencher ou non une reprise.

c) La configuration dynamique

Elle consiste en une réorganisation du séquencement de la production en fonction de l'apparition d'évènements dans le système. Son objectif est de permettre le passage d'un mode de marche à un autre. A ce niveau, deux hypothèses peuvent être envisagées :

- reprogrammation de la commande,
- intégration initiale des différents modes de marche.

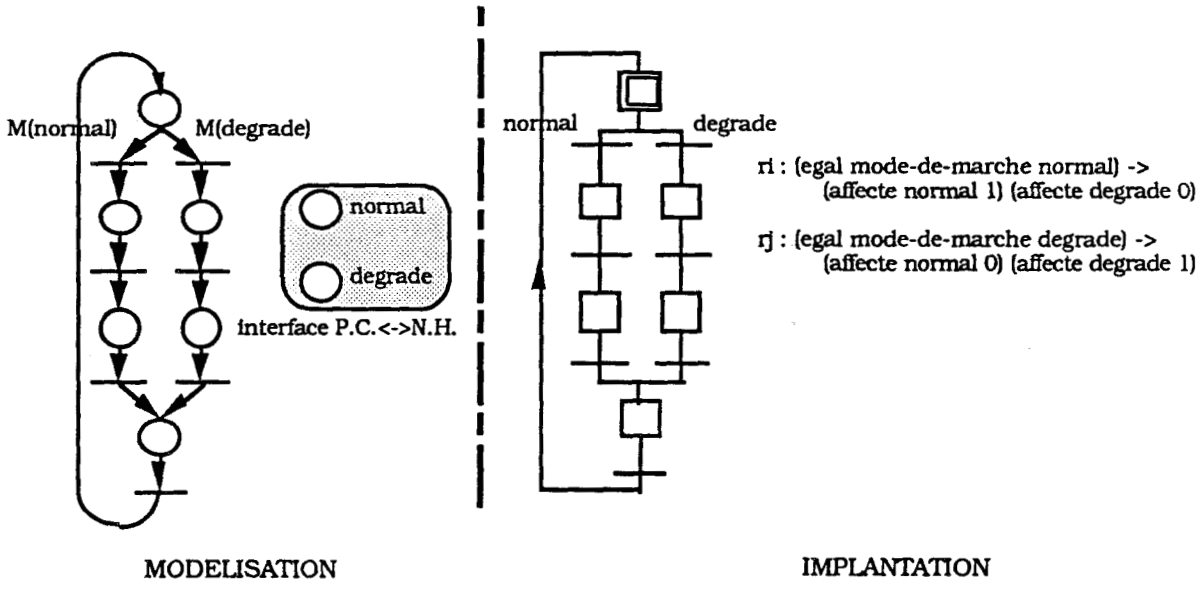
La reprogrammation de la commande pour intégrer un nouveau mode de fonctionnement implique nécessairement l'arrêt de la production pendant le chargement des programmes. Cette solution pénalise donc obligatoirement le rendement global et présente l'inconvénient majeur de laisser, pour un moment, le procédé sans contrôle. Une

telle situation peut très bien être considérée comme inacceptable pour la conduite de certains procédés (exemple : régulation continue d'un four thermique). Par contre, la reprogrammation reste envisageable lors de la phase de simulation. Les temps de réponse occasionnés sont négligeables ; la simulation est réalisée de manière off-line vis à vis de l'application ; enfin les langages couramment utilisés (PROLOG, LISP, ...) ne font pas de distinction réelle entre le code et les données, ce qui simplifie la modification des "programmes" (au sens large).

La seconde solution nécessite l'intégration initiale des différents modes de marche au sein de la commande. L'utilisateur, tout au long de la phase de conception, doit définir parallèlement au mode de marche normale, les différents modes dégradés susceptibles d'intervenir. L'utilisation d'arcs adaptatifs permet de rendre flexible le modèle de la commande et donc d'activer sélectivement des graphes ou parties de graphes suivant le mode retenu. La phase d'implantation se limitera ainsi à traduire les arcs adaptatifs en variables booléennes conditionnant les réceptivités des transitions concernées. La modélisation des différents modes de marche potentiels est donc dans ce sens, partie intégrale de la commande.

La décision de passer d'un mode à l'autre incombe cependant au niveau hiérarchique en fonction des stratégies à définir dans sa base de connaissances.

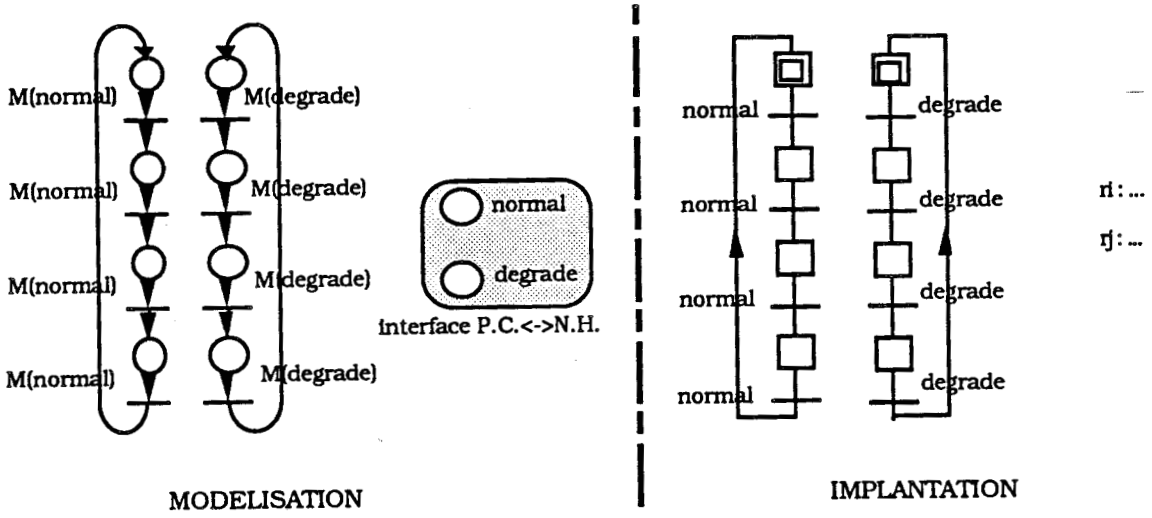
Exemple :



Marche dégradée 1

FIGURE 31

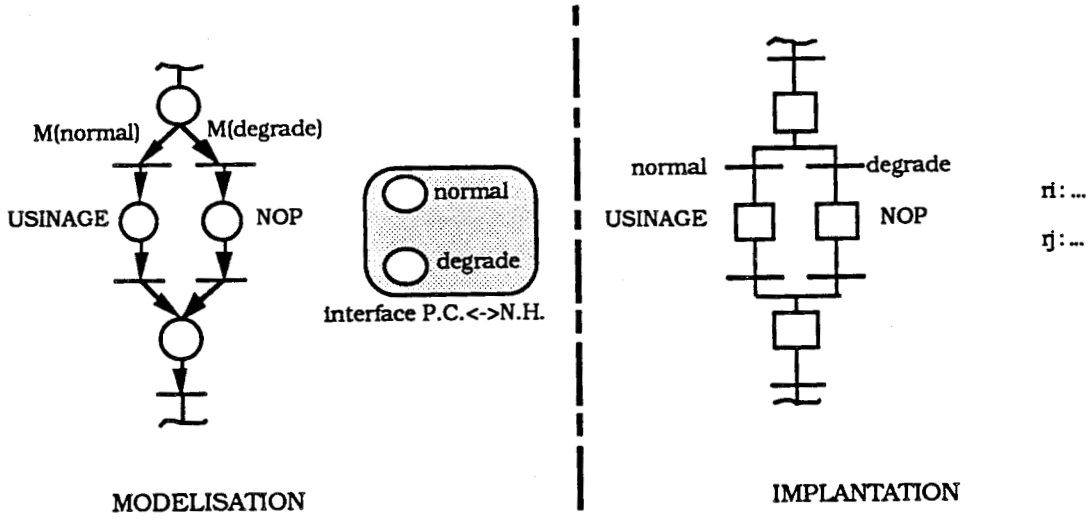
Dans cet exemple, le changement de mode de marche n'est autorisé que lorsque le processus est dans son état initial. Sa mise en œuvre est assurée par le déclenchement de la règle ri ou rj qui affecte la valeur adéquate aux booléens conditionnant les réceptivités. La détermination du mode de marche (normal ou dégradé) est elle-même assurée par le résultat de l'inférence d'autres règles introduites dans la base de connaissances et dépend aussi bien de l'état du procédé que de la configuration instantanée de la commande.



Marche dégradée 2

FIGURE 32

La modélisation, dans ce cas, permet de passer à tout moment d'un mode de fonctionnement à l'autre. La technique consiste à geler alternativement l'un ou l'autre des process en fonction de la stratégie retenue. Les règles ri et rj sont identiques à celles de la Figure 31.



Marche dégradée 3

FIGURE 33

La marche dégradée permet ici d'éviter de passer dans le module usinage. La machine étant considérée comme hors service, la circulation des pièces peut tout de même continuer au sein de l'installation en évitant de bloquer le système de transport.

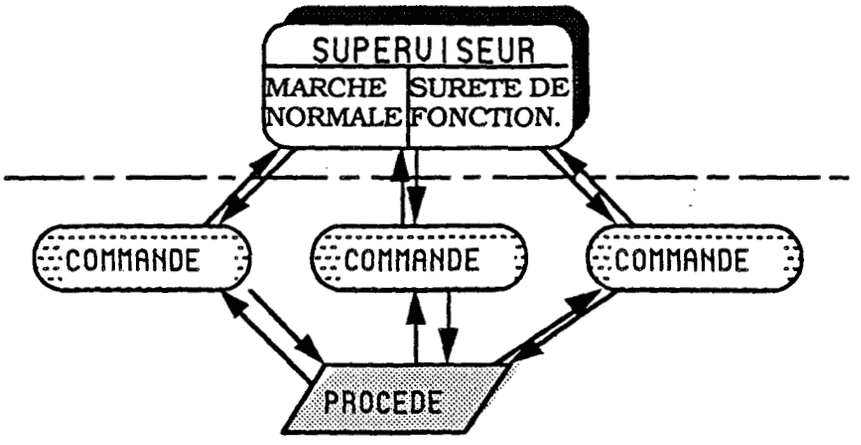
II.4.3 - Le niveau hiérarchique

La définition d'une surveillance séparée nécessite l'existence d'un outil externe aux organes de commande mais ayant la possibilité d'acquérir et de modifier l'état de ces derniers. Cet outil, le superviseur de commande, a déjà largement été utilisé dans la première partie de ce chapitre pour assurer le fonctionnement normal de l'installation dans le cadre d'une implantation répartie.

La nécessité d'intégrer, dans la mise en œuvre d'un système de production, un haut degré de sûreté [SOU 85] (i.e. : aptitude d'un système à fonctionner tel qu'il a été voulu et à assurer une continuïté de service en toutes circonstances), nous amène de nouveau à utiliser le superviseur, cette fois pour un objectif de sûreté de fonctionnement (Fig. 34).

Ses objectifs, pour pallier aux insuffisances des systèmes de commande, doivent permettre d'éviter les pièges des fonctionnements transitoires occasionnés par les changements de mode de marche ; ils doivent également permettre de diagnostiquer puis proposer des actions et régler automatiquement les incidents.

En fait, sa finalité ultime réside dans la conduite de l'installation comme le ferait le meilleur "expert" qui cumulerait la connaissance technique globale du procédé et l'expérience pratique accumulée par les opérateurs.



Supervision étendue

FIGURE 34

Un certain nombre de maquettes existent à ce jour. Les plus simples proposent des systèmes non connectés dans lesquels l'opérateur doit indiquer l'état des voyants, la phase du cycle, ... Le fait qu'elles soient déconnectées du procédé et de la commande les limite à une utilisation, a posteriori, de diagnostic de pannes. L'analyse de la panne est d'autant plus complexe que bien souvent l'état de l'installation est révélé après propagation d'erreurs.

D'autres maquettes sont connectées aux organes de commande. Elles rapatrient et filtrent les informations utiles afin de proposer des actions à réaliser, sans toutefois les effectuer par elles-mêmes [MAR 88]. La prise en compte immédiate d'un évènement nécessite ainsi la présence constante d'un opérateur qui décidera en dernier lieu, en fonction des informations à sa disposition, la conduite à tenir et les actions modifiant le procédé.

Le niveau hiérarchique que nous allons présenter dans la troisième partie de cet exposé, est connecté aux organes de commande. Il rapatrie les informations nécessaires et déduit les actions qu'il déclenche automatiquement. Sa définition et son utilisation permet de dissocier la commande proprement dite, des algorithmes d'analyse et de décision déterminant le mode de fonctionnement adéquat. Sa délocalisation lui assure une vue globale et cohérente de l'installation lui permettant d'agir dans le cadre d'une politique générale d'ordonnancement. Son formalisme, règles de production exploitées par un moteur d'inférence, ne préjuge en rien de l'évolution du système :

- les règles ou groupements de règles sont largement indépendants,
- la distinction entre connaissance et mécanisme d'exploitation permet de modifier facilement des règles sans courir les risques de la modification de la logique d'un programme classique.

Ce dernier point facilite grandement la modification des choix de fonctionnement de l'installation, aussi bien en fonction de la connaissance des opérateurs humains, que de l'expérience acquise en cours de production [MAR 87]. Enfin, l'utilisation des règles de production permet d'exprimer les expressions logiques par des expressions symboliques plus proches du langage naturel, ce qui facilite d'autant la compréhension et la mise au point des programmes [ATA 87] par l'exploitant du système de production.

II.5 - Conclusion

Dans cette partie, nous avons présenté les raisons qui nous ont conduits à ne pas retenir l'idée habituelle visant à intégrer les concepts de surveillance au sein même de la définition de la commande. L'analyse d'une telle démarche nous en a révélé les limites. Elle conduit à une mise à plat des différents aspects de commandes et de surveillances dont les fonctionnalités sont représentées sur un modèle unique. Le découpage fonctionnel de la commande imposé par l'architecture du procédé, impose une répartition artificielle de ces mêmes fonctionnalités sur différents organes de commande. Enfin, la puissance de résolution du système est limitée par les calculateurs (API) utilisés.

Ces remarques nous ont amenés à la définition d'une surveillance séparée qui, à l'aide d'outils plus puissants, orientés intelligence artificielle, permettent la mise en œuvre efficace des stratégies de surveillance grâce à une vision globale de l'état de la commande et du procédé [BOU 86] [BAR 88]. Pour ce faire, nous avons proposé une solution d'interfaçage du niveau hiérarchique avec la commande et présenté différents mécanismes d'interactions sur la commande déclenchés par le niveau hiérarchique.

III - REALISATION DU NIVEAU HIERARCHIQUE

III.1 - Introduction

Le choix d'une modélisation déclarative du niveau hiérarchique est justifié par plusieurs raisons. En premier lieu, vu du côté utilisateur, l'utilisation d'un langage déclaratif offre une convivialité que n'égale aucun des langages impératifs. Cette opportunité a permis de disposer d'une grande souplesse quant au développement du niveau hiérarchique. Enfin, le langage retenu (LE - LISP de l'INRIA) rend la définition du niveau hiérarchique homogène par rapport aux autres travaux complémentaires développés au L. A. I. I. dans le cadre du projet C. A. S. P. A. I. M.. Cela permet notamment de récupérer sans les ré-écrire, les stratégies de fonctionnement validées par la simulation et minimise ainsi les erreurs à l'implantation.

La définition des structures de contrôle pour gérer une installation ne nécessite à aucun moment, ni compilation, ni édition de liens ; cela a été rendu possible grâce à l'interactivité du langage LE - LISP. L'utilisateur dialogue avec le système sous forme "pseudo-naturelle" ; il peut indifféremment accéder à toutes les fonctionnalités de ce dernier. Il peut globalement accéder à l'intégralité de l'univers de travail, c'est-à-dire à la fois aux données relatives au modèle et aux fonctionnalités qui permettent de les exploiter (le "programme"). Cette interactivité est conservée pendant toute la durée de vie du niveau hiérarchique : conception, mise au point, exploitation. Elle est foncièrement nécessaire à la viabilité du système. Le concepteur n'appréhende qu'au fur et à mesure et souvent a posteriori, les cas limites de fonctionnement de son installation. La convivialité du système permet de ne jamais remettre en cause l'ensemble du niveau hiérarchique. Toute modification s'effectue rapidement et à moindre coût.

Le niveau hiérarchique est représenté par un ensemble de règles de production [LAU 82] gérées par un moteur d'inférence spécifique de type chaînage avant. Nous proposons dans cette partie de détailler la structure du système :

- les bases de faits,
- la base de connaissance,
- le fonctionnement précis du moteur d'inférence.

III.2 - Les bases de faits

III.2.1 - Définitions

Nous appelons fait interne, une variable typée dont la valeur initiale est donnée à la configuration du système et qui, par la suite, peut évoluer en fonction des résultats de l'inférence ; une telle variable est essentiellement locale au niveau hiérarchique. Par opposition, nous appelons fait externe, une variable typée dont la valeur est conditionnée par l'état de la commande ou du procédé. L'acquisition ou l'affectation d'un fait externe nécessite un accès à travers le réseau permettant la communication avec les organes de commande de l'installation. Il existe ainsi au sein du niveau hiérarchique deux bases de faits distincts, l'une qualifiée interne, l'autre externe.

III.2.2 - Représentation interne des faits

Pour chaque variable ou attribut, on conserve une série de propriétés :

- *nature* : interne ou externe (cf. définition ci-dessus),
- *type* : la variable peut être d'un type simple (exemple : booléen, entier, ...) mais elle peut également représenter une structure de donnée complexe prédéfinie dans le système (exemple : fifo, ...),
- *valeur* : un atome dans le cas d'un type simple ou une liste associée à la structure de donnée complexe.

Ces propriétés sont communes aux deux natures de faits : interne ou externe. Par contre, la définition d'un attribut externe nécessite des informations complémentaires. L'utilisateur doit renseigner la base de faits sur la localisation de cet attribut dans la commande : sur quel automate il se trouve et à quelle variable de cet automate il se rapporte. Soit le fait externe "ack" de type booléen ; l'utilisateur doit indiquer au système que ce fait correspond par exemple au bit "B10" de l'automate n° 3 du réseau. Ces renseignements lient la description virtuelle de la variable externe à une image réelle dont la valeur est conditionnée par la commande.

Avec ces informations supplémentaires et tenant compte de la nature et du type de fait externe, le niveau hiérarchique attache automatiquement à l'attribut, le protocole de communication à utiliser pour acquérir sa valeur ou la forcer. De cette façon, le traitement de l'information reste complètement transparent pour l'utilisateur. Jamais, il n'aura à se préoccuper des primitives réseaux et reste ainsi confiné pour plus de facilité au niveau de la couche 7 du modèle O. S. I. (Open Systems Interconnection) de l'International Standards Organization [HOS 84].

Les variables externes ont donc, dans leur structure de donnée, les informations supplémentaires suivantes :

- *station* : numéro de la station (automate) considérée,
- *numéro* : numéro de la variable physique à laquelle on se réfère,
- *fonction-acquisition* :
- *fonction-restitution* :

} primitives de communication utilisées pour acquérir ou modifier la valeur de la variable

Exemple :

```
(plist 'pl '(nature-bis          interne
             nature              mot
             valeur              5))
(plist 'ack (nature-bis          externe
             nature              bit
             valeur              non-encore-calculée
             station             3
             numéro-bit          10
             fonction-acquisition acquisition-bit-externe
             fonction-restitution restitution-bit-externe))
```

Cette modélisation se rapproche des techniques de programmation orientée objets. Les attributs indiquant les différentes propriétés du fait (nature, valeur, ...) correspondent aux champs de l'objet ; les fonctions d'acquisition et de restitution modélisent les méthodes

qui permettent de l'utiliser [HUL 84].

III.3 - La base de connaissance

Le langage d'expression de la base de connaissance est basé sur la logique des propositions avec utilisation de variables globales (logique dite d'ordre Ø+). La structure retenue d'une règle de production est la suivante :

(i) un identificateur de règle,

(ii) un énoncé du type : [condition] → [action].

La partie [condition] est constituée d'une combinaison logique de prémisses dont la valeur est conditionnée par l'état des bases de faits tandis que la partie [action] modifie ces dernières ainsi que le procédé et la commande à l'aide d'opérateurs prédéfinis.

III.3.1 - Définition syntaxique d'une règle

Cette définition est donnée sous la forme de NAUR-BACKUS :

<règle> : : = <antécédents> → <conséquents>
<antécédents> : : = [<prémisse interne> | <prémisse externe>] [<antécédents>] *
<prémisse interne> : : = <opérateur> <argument 1> <argument 2>
<argument 1> : : = <fait interne> | <constante>
<argument 2> : : = <argument 1>
<prémisse externe> : : = <opérateur> [(<fait externe> <argument 3>) |
(<argument 3> <fait externe>)]
<argument 3> : : = <fait interne> | <fait externe> | <constante>
<conséquents> : : = [<conséquent interne> | <conséquent externe>] [<conséquents>] *
<conséquent interne> : : = <opérateur> <fait interne> <argument 3>
<conséquent externe> : : = <opérateur> <fait externe> <argument 3>

Les opérateurs dans la partie <antécédents> permettent de tester l'égalité (au sens large : égal, différent, plus grand ou égal, ...) entre deux faits, un fait et une constante ou même deux constantes si l'on désire avoir une prémisse toujours vraie ou toujours fausse !

Ils permettent également de manipuler les structures de données complexes prédéfinies dans le système. Soit par exemple, l'attribut liste qui est une fifo et l'attribut vide qui est un entier initialisé à Ø. La prémisse (différent liste vide) sera vraie si la fifo contient des valeurs et fausse dans le cas contraire.

Les opérateurs dans la partie <conséquents> permettent l'affectation d'une valeur immédiate ou de la valeur d'un attribut ainsi que l'incrémentation (signe et pas de l'incrémentation quelconques) d'un attribut. Ils manipulent également les structures de données complexes : ranger dans la fifo "liste" la nature d'une pièce produite mémorisée dans l'attribut "pièce-produite" sera traduit par (put-fifo liste pièce-produite).

La définition syntaxique particularise deux types de prémisses et deux types de conséquents. Une prémisse interne est une prémisse qui n'utilise que des **faits internes** au niveau hiérarchique ; ces faits ne dépendent donc aucunement des process extérieurs. Une prémisse externe est une prémisse qui contient obligatoirement au moins un **fait externe** nécessitant ainsi une requête à un automate par l'intermédiaire du réseau pour connaître sa valeur. Un conséquent interne est un conséquent qui modifie par affectation ou incrémentation la valeur d'un **fait interne**. Cette modification peut se faire éventuellement en utilisant un fait externe ; néanmoins, cela n'occasionne aucune répercussion sur l'état de la commande, ni du procédé. Par opposition, un conséquent externe change la valeur d'un **fait externe** ; il nécessite un accès réseau et modifie l'état de la commande et du procédé. Ces distinctions entre prémisses et conséquents internes ou externes sont justifiées par les caractéristiques de fonctionnement et d'efficacité du moteur d'inférence (cf. Chapitre II, § III.5).

Afin d'augmenter la puissance de description de la connaissance, il est permis, en plus des règles de production, de définir des fonctions écrites directement en LISP. Cette facilité, limitée au concepteur ayant un niveau de compétence suffisant, se révèle essentiellement intéressante dans deux cas :

- Absence de **"sinon"** dans la structure, si <antécédents> alors <conséquents>. Cela oblige à utiliser plusieurs règles qui reprennent éventuellement comme prémisses, la négation des

prémisses des règles précédentes.

- Traitements complexes où la définition des opérateurs prédéfinis dans le système se révèle insuffisante.

Il est souhaitable que l'utilisation de fonctions LISP se limite à ces deux cas. En effet, aucun contrôle n'est effectué par le système quant à la validité des fonctions définies. Par contre, la saisie d'une règle de production est dirigée par le logiciel. Une analyse syntaxique est automatiquement assurée lors de la définition des faits et des règles. Toute référence à des attributs obligent que ceux-ci soient au préalable définies dans les bases de faits. Ce contrôle de cohérence permet de garantir que, pendant l'exécution, tout appel à un fait sera résolu. Enfin, la définition des fonctions sous forme de règles de production permet de classer automatiquement les prémisses et les conséquents suivant qu'ils sont interne ou externe. Cette opportunité optimise le fonctionnement du moteur d'inférence comme nous le verrons ultérieurement.

III.3.2 - Base de règles et base de méta-règles

Il existe dans le niveau hiérarchique deux bases de règles distinctes. La base de règles, dont la structure a été définie précédemment, gère effectivement la commande répartie et le procédé. C'est elle qui supervise l'installation et qui, en fonction des données collectées, agit sur les process. Néanmoins, pour éviter d'aller scruter en permanence la totalité de la base de règles et gagner ainsi en temps de réponse, il s'est avéré utile de définir une base de méta-règles qui sert de premier filtre lors de l'inférence.

Les méta-règles ne peuvent référencer que la base de faits internes. Ce choix volontaire rend leur inférence indépendante de la commande et interdit ainsi tout accès au réseau, tâche la plus lente du système. L'observabilité qui en résulte est suffisante dans la plupart des cas pour déterminer les règles à enlever ou à ajouter dans la base de règles. L'expérience montre que les règles du niveau hiérarchique peuvent être regroupées par catégories, chaque catégorie ayant une fonctionnalité bien précise et n'étant utilisé que pour un état donné de la commande ou du procédé. Si la commande n'est pas sur le point d'atteindre un état réglé par un ensemble de règles, il est alors inutile de laisser ces règles

participer au cycle d'inférence.

Par exemple, supposons qu'un ensemble de règles permette de décider de l'attribution d'une ressource entre différents process. Faisons par ailleurs l'hypothèse suivante : la variable interne "ressource" caractérisant sa disponibilité indique qu'elle a déjà été attribuée. Une méta-règle décidera donc d'enlever de la base de règles cette catégorie de règles. Quand la ressource aura été rendue, une autre méta-règle aura pour objet d'adjoindre à nouveau l'ensemble de règles gérant l'attribution, à la base de règles.

Le concept de méta-règles introduit ici n'a pas, comme dans MYCIN, l'objectif d'ordonner un groupe de règles par rapport à d'autres pour diriger prioritairement et plus efficacement le raisonnement vers une piste précise [FAR 85]. Plus simplement, le système cherche à minimiser la taille de la base de règles courantes car l'évaluation des prémisses externes est l'opération la plus coûteuse en temps ; cette évaluation dépend de toute évidence du nombre de règles consultées à chaque inférence.

III.4 - Le moteur d'inférence

III.4.1 - Introduction

Le principe retenu pour le moteur d'inférence réside dans un chaînage avant, en largeur d'abord. Le régime retenu est irrévocable ; toutefois, l'évolution des connaissances est dynamique. Le fonctionnement du moteur est complètement asynchrone par rapport à l'évolution des organes de commande. Sa mise en œuvre s'apparente pour une part au problème de la coopération entre systèmes experts [CHA 87] [HAU 86]. Les problèmes à traiter sont physiquement distribués sur un ensemble d'entités coopérantes. Les données relatives à ces problèmes peuvent être modifiées aussi bien par le niveau hiérarchique que par les organes de commande et le procédé. Des hypothèses devront donc être prises concernant la durée de validité de leur valeur.

Dans un premier temps, nous allons justifier les différents choix de fonctionnement caractérisant le moteur d'inférence :

- chaînage avant,
- largeur d'abord,
- régime irrévocable,
- fonctionnement dynamique,

puis nous détaillerons son cycle complet de résolution.

III.4.2 - Justification du fonctionnement

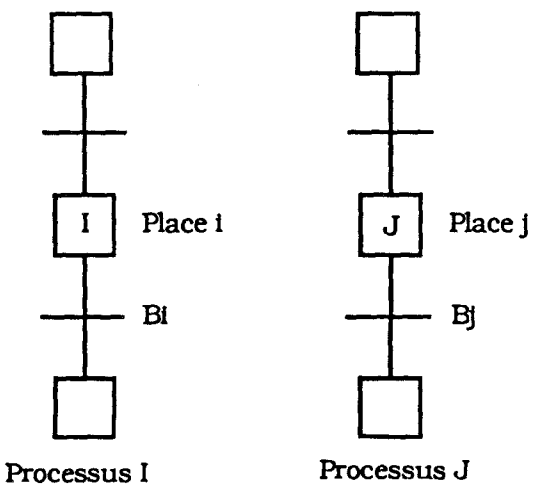
a) Chaînage avant / Chaînage arrière

Le chaînage avant consiste en un raisonnement "depuis les données vers les buts". Ainsi la règle « si A alors B » ne peut être déclenchée que si toutes les conditions exprimées par "A" sont vérifiées (connues), c'est-à-dire apparaissent dans la base de faits. Le système sélectionne pour son infférence des règles en s'appuyant sur ce qu'il connaît du cas particulier. Il les (ou la) déclenche après sélection ; ou s'il ne peut progresser par manque de renseignements, il a la possibilité de faire l'acquisition de données supplémentaires. L'objectif est principalement d'arriver à saturation de la base de faits (plus aucune règle n'est déclenchable) ou à la résolution du but que l'on s'était fixé.

Le chaînage arrière consiste en un raisonnement "depuis le but vers les données". Le système s'assigne un but à prouver et va chercher au sein de la base de règles le moyen de le résoudre. Il sélectionne une règle qui permet de conclure au but et va tenter de valider les prémisses de cette règle. Si ces dernières appartiennent à la base de faits, le problème est résolu ; sinon, les prémisses à valider deviennent autant de sous-buts à prouver. Le système réitère le processus jusqu'à validation complète des prémisses des règles sélectionnées. S'il y a échec, le système fait marche arrière, seulement s'il fonctionne en règime par tentatives et non de façon irrévocable, (back-tracking) pour sélectionner une nouvelle règle susceptible de prouver le but recherché. Il y a ainsi construction d'un graphe "ET/OU" où les nœuds "OU" représentent l'ensemble des règles pouvant déduire le même but et les nœuds "ET" l'ensemble des prémisses d'une même règle.

b) Comparatif entre chaînage avant et chaînage arrière

Soient deux processus i et j se partageant une ressource critique. L'attribution de cette ressource critique pour le processus i (respectivement j) se fera au niveau de place i (resp. place j) en validant la réceptivité de la transition bi (resp. bj) (cf. Fig. 35). En cas de conflit (demande simultanée), la ressource est alternativement octroyée soit à l'un, soit à l'autre (i.e. : mécanisme de gestion de la ressource par bascule Flip / Flop).



Gestion d'une ressource

FIGURE 35

La base de connaissances est la suivante :

R1 : si (égal ressource libre) et (égal place i marqué)
et (égal place j marqué)
alors (affecte conflit détecté)

R2 : si (égal conflit détecté) et (égal Flip / Flop au-tour-i)
alors (affecte bi 1)

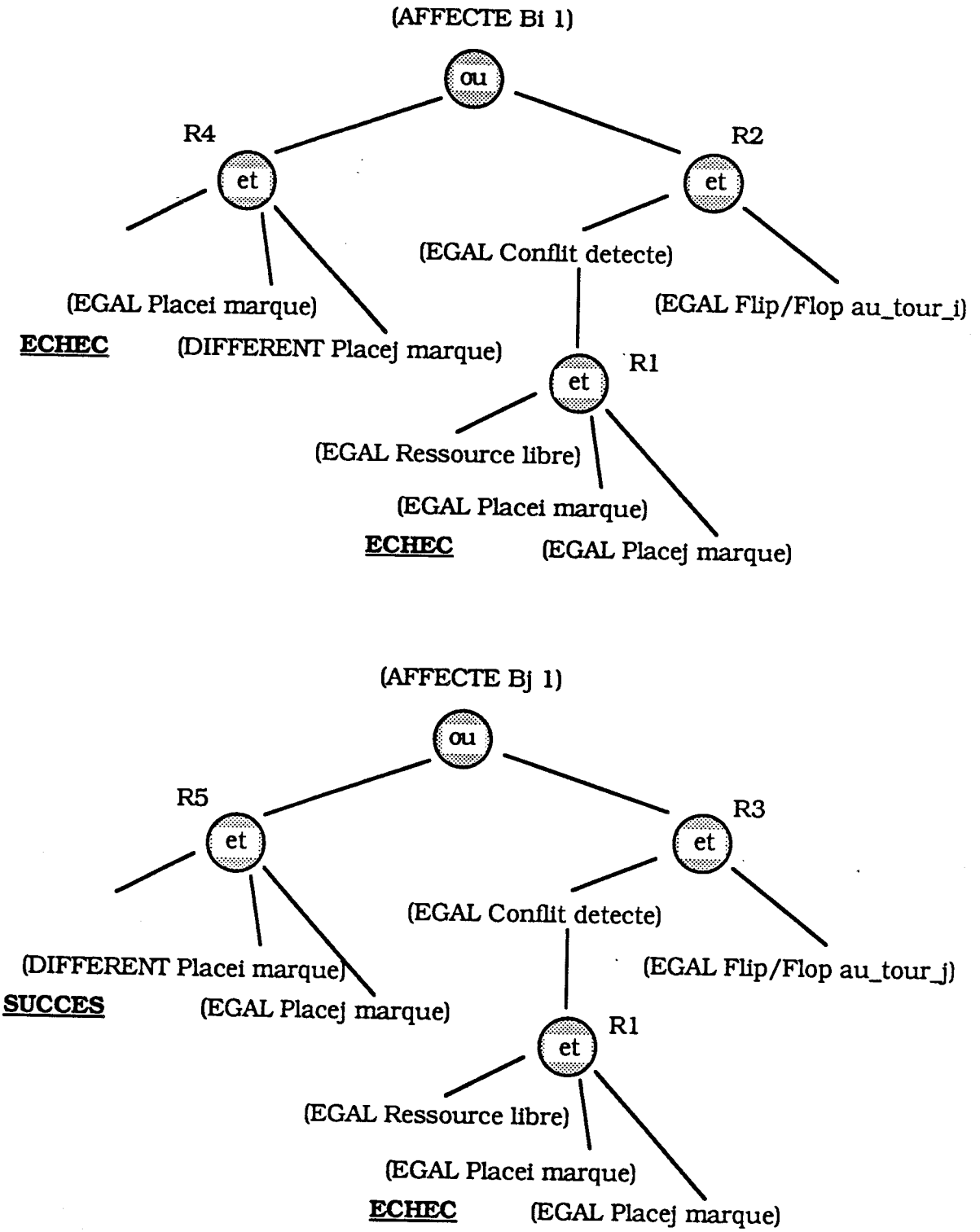
R3 : si (égal conflit détecté) et (égal Flip / Flop au-tour-j)
alors (affecte bj 1)

R4 : si (égal ressource libre) et (égal place i marqué)
et (différent place j marqué)
alors (affecte bi 1)

R5 : si (égal ressource libre) et (égal place j marqué)
et (différent place i marqué)
alors (affecte bj 1)

L'état courant de la base de faits est le suivant :

- ressource libre,
- place i libre,
- place j marqué,
- Flip / Flop au-tour-i.



Graphe "ET / OU"

FIGURE 36

En prenant pour hypothèse, un fonctionnement du moteur en chaînage arrière, le niveau hiérarchique va chercher tout d'abord à prouver le but (affecte bi 1) qui lui permet de donner la ressource au processus i (cf. Dessin graphe "ET/OU"). Le moteur sélectionne la règle R4 dont les prémisses peuvent directement être vérifiées par l'état de la base de faits ; il y a échec. Le moteur revient en arrière (back-track) pour explorer une nouvelle branche de l'arbre "ET/OU". Les prémisses de la règle R2 deviennent les nouveaux sous-buts à vérifier. La prémisse (égal conflit détecté) peut être validée par interprétation sémantique en utilisant la règle R1. Les prémisses de la règle R1, directement obtenues par l'état de la base de faits aboutissent à un échec. Il n'existe plus de branche "OU" dans notre graphe d'où échec dans la résolution du but (affecte bi 1).

Pour prouver le but (affecte bj 1), le moteur devra utiliser dans l'hypothèse la plus négative (i.e. : la résolution de conflit retient d'abord la règle R3 puis la règle R5), les règles R3 puis R1 (échec) et enfin la règle R5 pour aboutir à un succès. En résumé, l'utilisation du chaînage arrière sur cet exemple peut amener le moteur à déclencher jusqu'à 6 règles pour arriver à attribuer la ressource. Que dire si cette dernière n'était pas libre ? Dans ce cas, le problème est a priori sans solution (conclusion à laquelle aboutit le moteur d'inférence en chaînage arrière après 6 tentatives de résolution).

En gardant la même configuration pour la base de connaissances et la base de faits, l'utilisation du moteur en chaînage avant permet directement de conclure à l'attribution de la ressource au processus j. En effet, seules les prémisses de la règle R5 permettent au moteur d'évoluer et aboutissent immédiatement à la conclusion. Il apparaît ici concrètement sur cet exemple, que l'élaboration de choix de commande ou d'attribution d'une ressource relève d'une démarche de synthèse partant des faits.

c) Intérêt du chaînage avant pour le niveau hiérarchique

L'utilisation du moteur en chaînage avant a pour objectif à partir de l'état courant des bases de connaissances et de faits de prouver le plus grand nombre de buts possibles. Ces buts prouvés deviennent à nouveau des faits permettant au moteur de réitérer l'inférence. Deux critères d'arrêt sont alors envisageables.

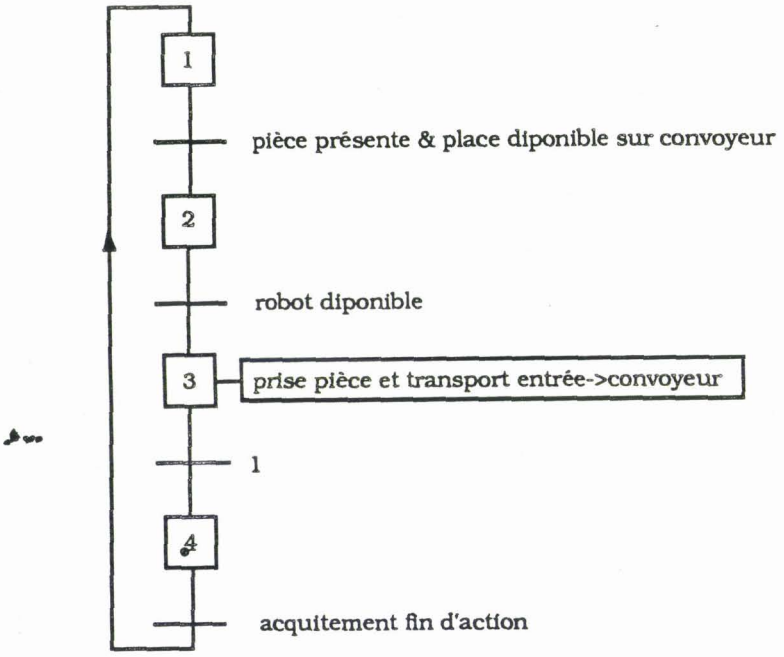
- La base de faits arrive à saturation : aucun nouveau but ne peut être déduit. Le moteur s'arrête alors car il ne sert à rien de rajouter dans la base de faits des buts qui ont déjà été démontrés au préalable.

- Il n'existe plus de règle candidate à l'inférence (i.e. : dont les prémisses ont été vérifiées). Cet état peut se produire de différentes manières suivant le fonctionnement du moteur d'inférence :
  * En logique non monotone, une règle précédemment candidate peut éventuellement ne plus l'être car le (ou les) fait(s) qui ont permis de la déclencher ont été ultérieurement réfutés par une autre règle ou par elle-même.
  * Le fonctionnement du moteur peut imposer que toute règle ne soit sélectionnée qu'au plus une fois. Ainsi chaque règle ayant servie à conclure de nouveaux buts est retirée de la base de connaissances.

L'utilisation d'un moteur en chaînage arrière a pour intérêt fondamental de focaliser la résolution d'un problème sur ce problème lui-même. Le système s'assigne ou on lui assigne un but hypothétique ; il recherche ensuite la façon de le prouver. L'avantage d'une telle démarche réside dans le fait que seules les règles participant de près ou de loin à la résolution du but fixé sont prises en compte. Encore faut-il connaître ce que l'on souhaite démontrer.

Dans le contexte qui nous préoccupe, un niveau hiérarchique implanté sur un micro ou mini-ordinateur, connecté sur un réseau d'automates qu'il supervise, il est illusoire de vouloir démontrer en chaînage arrière, l'ensemble des buts susceptibles d'être réglés par le système. En effet, le contexte extérieur (les automates programmables pilotant les process) évolue dynamiquement et peut donc prendre un nombre d'états très important. Or, pour un état donné des process extérieurs, donc de la commande, une grande partie des buts potentiellement exprimés dans la base de connaissances est sans intérêt. Considérons l'exemple d'un processus chargé d'alimenter un convoyeur à partir d'un tampon d'entrée et utilisant pour cette action un robot. Le robot et le convoyeur sont considérés

comme points critiques car ils peuvent être tous deux gérés par d'autres process pour des tâches complémentaires. Le niveau hiérarchique se charge ainsi de décider s'il attribue ou non ces deux entités au processus considéré (Fig. 37).



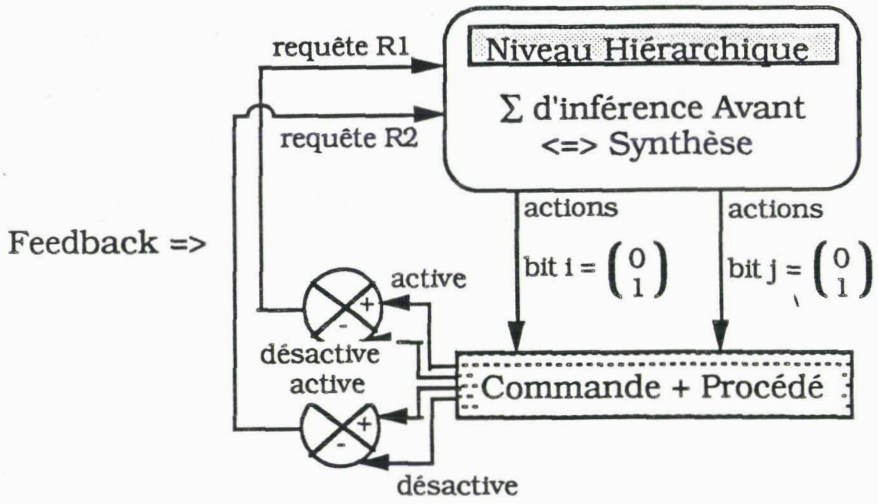
Intérêt du chaînage avant

FIGURE 37

Il dispose ainsi d'un ensemble de règles lui permettant de satisfaire le but (affecte convoyeur processus-i) et d'un autre pour autoriser l'utilisation du robot, symbolisé par le but (affecte ressource processus-i). Supposons maintenant que le processus i est marqué au niveau de sa place numéro 4 (cf. Dessin) en attente d'acquiter la fin de son action. Le fonctionnement du moteur d'inférence en chaînage arrière, lancé sur la résolution du but (affecte ressource processus-i) aurait abouti naturellement à un échec car le processus-i n'est pas demandeur de la ressource. Seulement, ce résultat aurait été obtenu après avoir tenté de résoudre un nombre important de sous-buts, dépendant de la politique d'attribution de la ressource. Le moteur aboutirait à un échec analogue en focalisant sa résolution sur le but (affecte convoyeur processus-i).

L'utilisation, dans le même contexte, d'un moteur en chaînage avant donne bien évidemment le même résultat : on ne peut ou ne doit octroyer au processus-i, ni le convoyeur, ni la ressource. Par contre, cette résolution est immédiate car le raisonnement est dirigé "depuis les données, vers les buts". La configuration de la base de faits permet de savoir immédiatement que le processus-i est dans sa phase d'évolution au niveau de la place 4 du Grafcet et qu'il est donc inutile de déclencher les règles régissant l'attribution des deux entités considérées. Cela se traduit par la présence d'une prémisse (égal marquage-processus-i place 1) ou (égal marquage-processus-i place 2) au sein des règles considérées qui empêchent de retenir ces dernières.

Le fonctionnement en chaînage avant est équivalent à une démarche de synthèse de type bottom-up, qui partant de l'état du procédé et de la commande dirige le raisonnement par les faits observés pour déterminer les actions à réaliser. Cette méthodologie est à opposer à une approche par analyse de type top-down qui, à partir d'une vision générale d'un problème et par affinements successifs des contraintes, permet de descendre au niveau de détail élémentaire. Dans le cas présent, la conception ascendante du niveau hiérarchique est rendue obligatoire de par sa délocalisation vis à vis du système de production. L'autonomie respective des différentes entités (procédé / commande et niveau hiérarchique) nécessite que les lois de fonctionnement général soient déterminées par synthèse après observation fine du comportement et des données accessibles de la commande. Le niveau hiérarchique est confronté à un système réactif dont le fonctionnement doit être régulé (Fig. 38). Le but atteint est le bon si la requête initiale de la partie commande (un ou plusieurs faits observés par le niveau hiérarchique) est, après traitement, acquitée par le couple procédé / commande. Cet acquitement introduit la notion de feedback (par analogie avec l'automatique) qui valide les actions déclenchées. Ce feedback peut être explicite par émission d'un "acknowledge" provenant de la commande après évolution en des points critiques de son fonctionnement ou implicite en constatant simplement que les faits observés qui ont déclenché un traitement ont été acquités. Le bon fonctionnement peut ainsi être validé par observation d'un "acknowledge" en un temps jugé raisonnable qui amène un arrêt de l'inférence en l'absence d'autre requête.



Analogie avec l'automatique

FIGURE 38

Enfin, l'utilisation d'un moteur en chaînage avant permet une formulation naturelle et intuitive des connaissances, aussi bien que du raisonnement. L'esprit humain est ainsi fait que le modus ponens : de "P" et "P implique Q" on déduit "Q", schématise de façon la plus simple l'heuristique utilisée pour formuler les problèmes.

Le chaînage avant permet ainsi de définir une stratégie optimale visant à saturer les bases de faits sans connaissance a priori des buts à prouver. Il s'intègre parfaitement dans le cadre d'une démarche partant des faits et exploite la notion de feedback émanant du système réactif. En dernier lieu, il facilite la définition des règles et la compréhension du fonctionnement du moteur.

d) Fonctionnement du moteur « en largeur d'abord »

Soit E, un état de la base de faits. Le moteur d'inférence va déclencher en parallèle toutes les règles qui auront satisfait les conditions de restriction et de filtrage avant d'adjoindre leurs conclusions dans E et de les utiliser pour déclencher de nouvelles règles. Appelons faits du niveau Ø, les faits initiaux, faits de niveau 1 les faits obtenus à partir du déclenchement des règles retenues d'après les faits de niveau Ø. Plus généralement, les faits de niveau n+1 sont ceux qui sont obtenus à partir des faits dont un au moins est de niveau n. Le moteur ne produira des faits de niveau n+1 seulement après avoir produit **tous**

les faits de niveau n. Concrètement, si le niveau n est l'état courant de la base, le moteur va chercher à produire le maximum de faits déductibles de cet état. Les nouveaux faits sont mémorisés temporairement dans une variable intermédiaire. Si plus aucun fait ne peut être produit à partir du niveau n, le contenu de la variable intermédiaire est vidé dans la base pour obtenir le niveau n+1 et l'inférence recommence à partir de ce dernier niveau.

Cette solution a été préférée à une stratégie avec intégration immédiate de tous les conséquents de la première règle retenue. En effet, cette dernière solution veut que l'on reconsidère toutes les règles dès qu'un fait nouveau a pu être déduit. Elle est donc généralement plus coûteuse en nombre total de cycles d'inférence par rapport à une solution « en largeur d'abord ». Une autre constatation nous permet d'affirmer que l'ensemble des règles du niveau hiérarchique peut se décomposer en différents modules disjoints ; chaque module correspondant à une fonctionnalité bien précise n'interférant que très rarement avec les fonctions des autres modules.

Soient, par exemple, les deux groupements fonctionnels suivants :

- un ensemble de règles symbolise la politique d'attribution d'un organe de transports entre différents process,
- un second ensemble modélise la gestion d'un producteur / consommateur réparti entre deux automates différents.

Le fonctionnement du moteur « en largeur d'abord » autorise ainsi une recherche **en parallèle** des différents buts à obtenir de chaque groupement fonctionnel de règles. Ce mode d'inférence est donc le plus apte à répondre efficacement à la politique générale du système visant à obtenir une saturation de la base de faits (déduction du plus grand nombre de faits) afin de pouvoir modifier de manière cohérente l'état des process extérieurs en fonction des actions déduites.

e) Fonctionnement irrévocable / Fonctionnement par tentatives

Par définition, en fonctionnement irrévocable, on considère que le déclenchement des règles retenues lors de l'inférence est irréversible, quelque soit le résultat final de cette inférence. Ce concept est ici à opposer à un fonctionnement par tentatives. En effet, ce

dernier peut au cours de la résolution d'un problème, remettre en cause les choix précédemment retenus lors de l'étape de résolution des conflits. Si pour une raison ou une autre, le moteur ne peut plus déclencher de règles tendant à solutionner un but fixé, il procède à un retour en arrière ayant pour effet de restaurer le contexte (aussi bien base de faits que de connaissances) dans un état antérieurement établi. A partir de là, l'étape de résolution des conflits lui permet d'explorer d'autres règles qui avaient été précédemment écartées.

Le choix d'une règle parmi un ensemble de règles déclenchables peut se faire d'après de nombreux critères :

- première règle trouvée,
- règle qui a le moins servi,
- règle la plus récemment utilisée,
- règle la plus informante ou la plus facile à évaluer,
- etc...

Le fonctionnement de l'interprète PROLOG correspond à l'exemple type d'un moteur par tentatives. L'exécution d'un programme est lancée par l'effacement d'une suite de buts, examinés dans l'ordre où ils apparaissent. La résolution de conflits de l'interprète consiste à prendre la première règle dont la conclusion s'unifie avec le but courant. Si la résolution aboutit à une impasse, l'interprète revient à l'étape précédente et choisit la règle suivante pouvant s'unifier avec le but à prouver. L'interprète fonctionnera de cette façon jusqu'à épuisement complet de toutes les règles unifiables avec le but : l'exécution est non déterministe au sens où l'on cherche à exploiter toutes les possibilités envisageables [GIA 85].

Dans le cadre de nos préoccupations (supervision de la commande d'une cellule flexible), nous avons retenu comme mode de fonctionnement un régime irrévocable. Une partie des justifications de ce choix a déjà été indirectement donnée au paragraphe *c)* (intérêt du chaînage avant pour le niveau hiérarchique). En effet, nous avons mis en évidence les raisons qui nous ont conduits à utiliser un moteur en chaînage avant ayant pour effet d'engendrer une stratégie visant à saturer la base de faits. Le moteur infère en chaînage avant parce qu'il ne connaît pas a priori les buts à prouver ou les actions à engendrer sur les commandes des process extérieurs. Il n'a donc pas et ne peut pas remettre en

cause les inférences précédemment réalisées, n'ayant pas conscience de la notion d'échec. De plus, l'utilisation d'une stratégie en largeur d'abord lui assure la certitude d'explorer en parallèle toutes les règles déclenchables résultant de la phase de restriction. La résolution des conflits dans notre cas, ne met donc aucune règle de côté ; le contraire aurait éventuellement pu justifier un retour arrière.

Enfin, les actions effectuées par le moteur ne sont pas irréversibles. Au cours de l'élaboration d'un plan visant à régler un indéterminisme directionnel, l'inférence peut, dans un premier temps, aboutir à la conclusion de diriger la pièce i sur la machine t (ce qui se traduit par (affecte machine-t pièce-i)). Or, le déclenchement ultérieur d'autres règles peut conclure définitivement à l'attribution de la machine u pour la pièce-i ; ce constat est réglé par une règle dont les conséquents sont les suivants : (affecte machine-u pièce-i) et (affecte machine-t libre). Ce dernier conséquent permet de rendre à nouveau disponible la machine t en évitant l'emploi d'un retour arrière pour restaurer le contexte avant l'affectation de la machine t. Chaque action élémentaire du système possède son inverse ou sa réciproque et rend ainsi inutile l'utilisation d'un fonctionnement par tentatives.

f) Système dynamique

Nous utilisons ici la définition donnée par R. Voyer [VOY 87] qui particularise les systèmes dynamiques par rapport aux systèmes non-monotones. En fonctionnement non monotone, le langage utilisé dans les règles offre des primitives qui permettent de supprimer, ou inhiber provisoirement, des connaissances (aussi bien des faits que des règles). Tel est le cas avec l'emploi des 3 primitives TUER, TUERFAITS et TUERFAIT dans le système expert SNARK [LAU 86]. Deux types d'interprétations différentes peuvent justifier pleinement la non-monotonie :

- Mise en œuvre d'un raisonnement par défauts. Une affirmation est considérée comme vraie en l'absence de toute autre information contradictoire. Soit, par exemple, la règle :

* *Si la pièce n'est pas correctement usinée alors la machine est en panne.*

En l'absence d'autre information, l'inférence continue avec pour fait "la machine est en panne". S'il s'avère par la suite que d'autres règles prouvent le contraire, le fait "la machine est en panne" ainsi que tous les conséquents qui ont pu être déduits doivent être retirés de la base.

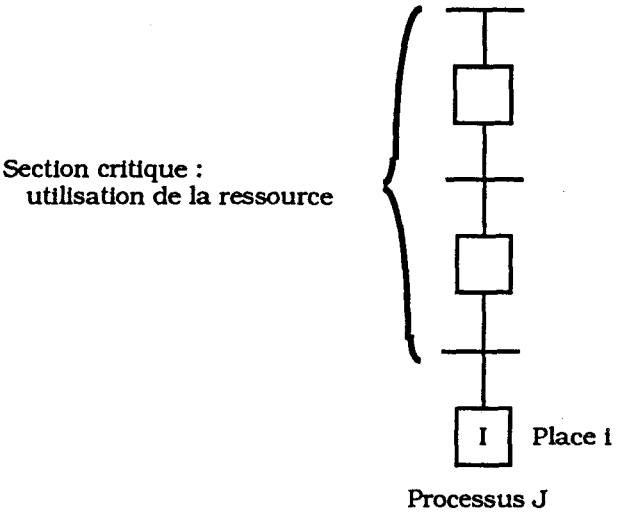
- Implémentation d'un diagnostic interactif. La non-monotonie permet de prendre en compte l'évolution de la valeur de vérité d'un fait dans le temps.

En fait, pour la modélisation et la gestion d'un univers en évolution, la non-monotonie symbolise implicitement la notion du temps qui s'écoule, induisant une évolution de l'état du procédé donc de la base de fait.

Ce dernier concept justifie l'utilisation de la non-monotonie dans notre système. Plus précisèment, le système possède un comportement dynamique intrinsèque. Il autorise ainsi l'emploi de règles dont les conséquents (qui sont des actions engendrées sur la base de faits) peuvent nier les prémisses qui ont permis de les déclencher. Prenons par exemple, la règle suivante (cf. Fig. 39) :

si (égal ressource à-proc-j) et (égal place-i marqué)
alors (affecte ressource libre)

La ressource (un organe de transport : robot, convoyeur, une machine, ...) est utilisée par le processus j mais la place i de ce processus est marquée. Cela permet d'affirmer que le processus n'a plus besoin désormais d'utiliser la ressource.



Système dynamique

FIGURE 39

En fait, l'interprétation sémantique de cette règle permet d'en déterminer le sens. A l'instant t, la ressource est effectivement attribuée au processus j ; par contre, à l'instant $t + \Delta t$, Δt correspondant à l'évolution du temps implicite pendant le déclenchement de la règle considérée, la ressource devient de nouveau disponible et pourra être prise en compte par d'autres processus en utilisant de nouvelles règles. Ce symbolisme n'est pas différent de celui de l'affectation classique utilisée en informatique : l'expression $X = X + 1$, choquante au premier abord, est pleinement justifiée une fois la sémantique de l'opérateur dévoilée.

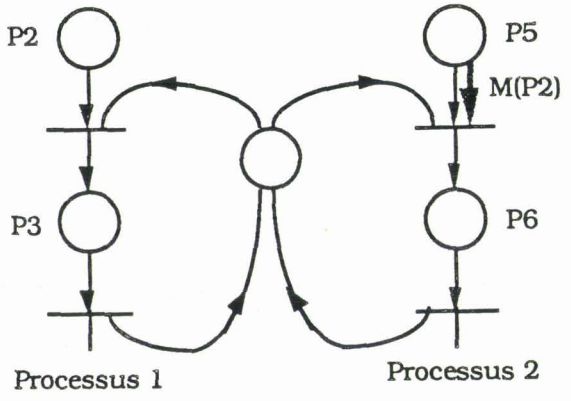
Nous aurions pu utiliser d'autres techniques pour prendre en compte la notion du temps dans notre système. Des travaux portant sur l'utilisation de la logique temporelle ont été menés au sein du laboratoire [ZHA 85]. Cette dernière met en œuvre 2 opérateurs temporels fondamentaux :

- **parfois** A : A sera vraie à un instant futur,
- **toujours** A : A sera vraie à tout instant futur.

D'autres opérateurs peuvent ensuite être déduits et sont considérés comme quantificateurs du temps. L'un des problèmes principaux, induits par l'utilisation de la logique temporelle, consiste en une explosion dans la desciption des process mis en œuvre. En effet, pour envisager et traiter toutes les séquences d'exécution d'un système, il faut cons-

truire son graphe d'états ! Enfin, le formalisme utilisé pour exprimer des formules ou règles en logique temporelle est très complexe.

Soit, par exemple, la configuration suivante exprimée en réseau de Petri adaptatif :



Logique temporelle

FIGURE 40

L'exclusion mutuelle s'exprime de la façon suivante :

toujours (non [P3 Λ P6])

ce qui signifie que l'on ne peut jamais avoir le marquage simultané de P3 et P6.

La priorité donnée au processus 1 (il existe un arc adaptatif entre P5 et P6 pondéré par le marquage de P2) consiste en :

toujours [P2 Λ P5] → non [P6] jusqu'à [P3]

en d'autres termes, on aura toujours, si P2 et P5 sont marqués simultanément, jamais P6 jusqu'à ce que l'on ait P3 !

$P_6 = P_6 \cdot [\overline{P_2} + \overline{P_5}] + P_3$

Face à ce constat, nous avons ainsi préféré retenir l'utilisation d'une logique plus classique qui, utilisée avec un fonctionnement dynamique, permet d'introduire de façon

implicite l'évolution du temps dans notre système. Néanmoins, un problème crucial, non encore résolu à l'heure actuelle reste posé : comment assurer la cohérence dans le fonctionnement des règles ? En effet, les actions engendrées par les conséquents des règles sur la base de faits peuvent être contradictoires. Définir comme règles incohérentes, deux règles ayant exactement mêmes conditions et des conclusions contraires ne suffit certainement pas à prouver que le système restera cohérent tout au long de son évolution. De nombreux travaux sont à ce jour menés [BEU 86], [AYE 86] qui, espèrons-le, apporterons des solutions satisfaisantes à ce problème dans un avenir proche.

III.4.3 - Conclusion

Le moteur d'inférence présenté ici n'a pas la prétention d'être la solution idéale pour tous les problèmes rencontrés lors de la supervision d'une cellule flexible. En fait, deux limitations peuvent lui être reprochées :

- moteur d'ordre Ø+ et non d'ordre 1,
- inférence en chaînage avant uniquement.

Un moteur d'ordre 1 introduit des variables et permet la formulation de lois générales qui sont évaluées par instanciations successives. Deux aspects nous ont amenés à retenir le moteur d'ordre Ø+ plutôt qu'un moteur d'ordre 1. En premier lieu, il semble difficile de dégager des règles générales de fonctionnement dans la supervision d'une cellule flexible. La majorité des problèmes a un caractère particulier qui se règle au cas par cas en utilisant la logique des propositions. En outre, l'utilisation de variables complique le formalisme d'expression et est donc plus difficile à appréhender. Le second point concerne les temps de réponse nettement plus important en utilisant la logique des prédicats d'ordre 1. L'instanciation des règles conduit très rapidement à une explosion combinatoire qui, pour l'éviter, nécessite soit de pouvoir limiter le pouvoir d'expression de ces mêmes règles, soit d'adopter une stratégie élaborée, complexe à mettre en œuvre [KOD 85].

L'intérêt du chaînage avant a été largement présenté pour la supervision de la commande. Néanmoins, pour des classes particulières de problèmes (ex : analyse d'une défaillance détectée et identifiée), le chaînage arrière permettrait de focaliser la résolution sur le but fixé et serait alors plus performant. Il peut cependant être mis en échec par la

présence de raisonnement circulaires qui, dans ce cas, imposent l'utilisation d'informations supplémentaires pour éviter que l'évaluation des règles ne conduise à des boucles infinies.

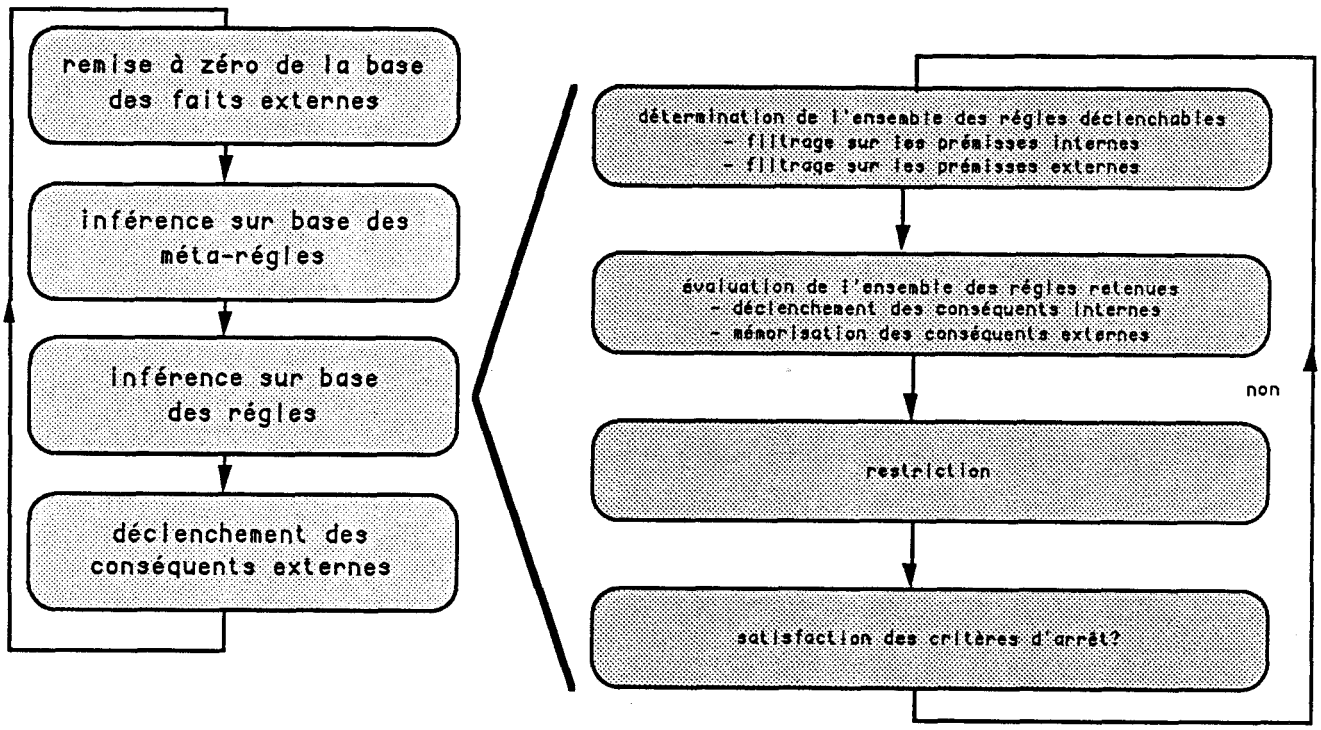
La solution retenue, par sa relative simplicité, est ainsi facile à utiliser et offre un fonctionnement robuste et relativement rapide qui, suivi en mode trace, est d'interprétation aisée.

III.5 - Cycle complet de résolution

Le cycle complet de résolution se décompose en quatre étapes principales et intègrent deux cycles d'inférence complets :

- inférence sur base des méta-règles,
- inférence sur base des règles (Fig. 41).

Nous allons maintenant préciser ces différentes étapes en détaillant particulièrement le fonctionnement de la troisième qui est la plus importante du système.



Cycle complet

FIGURE 41

ETAPE 1 : Remise à zéro de la base des faits externes

Cette étape correspond à l'initialisation du système. Le niveau hiérarchique va débuter ses inférences en ne possédant aucune information externe, ni sur la commande, ni sur le procédé. La seule modélisation à sa disposition correspond à l'état de sa base de faits internes. A ce stade, deux solutions sont envisageables :

- acquisition systématique de la valeur de tous les faits externes avant de débuter l'inférence,
- acquisition dynamique de ces faits au cours de l'inférence.

C'est la deuxième solution qui a été retenue pour le fonctionnement du système. Ainsi, ni la commande, ni le procédé, ne transmettent à un instant donné toutes les informations caractérisant leurs états. Par contre, ces données sont recherchées par le niveau hiérarchique au cours de l'inférence (Etape 3) en fonction de ses besoins. En ce sens, le niveau hiérarchique agit comme un expert qui dirige son raisonnement sur des opérations précises et ne collecte que les informations utiles à l'obtention d'une solution. Cette réalisa-

tion est analogue à celle retenue pour le système expert PICON [KNI 85] et présente l'avantage de minimiser la communication vers les organes extérieurs.

ETAPE 2 : Inférence sur base des méta-règles

L'inférence sur la base des méta-règles est dissociée de celle sur la base des règles. L'objectif est de minimiser le nombre de règles pour l'Etape 3. Le moteur utilisé est le même que celui de l'étape suivante par raison de simplicité. Néanmoins, quelques modifications sont apportées au cycle en fonction de la structure différente des méta-règles. Le filtrage sur les prémisses externes n'a pas de raison d'être, car une méta-règle n'intègre par définition que des faits internes. De même, la mémorisation des conséquents externes disparaît. Enfin, la phase de restriction se limite à la suppression des règles qui ont satisfait le filtrage et ont donc été évaluées.

ETAPE 3 : Inférence sur base des règles

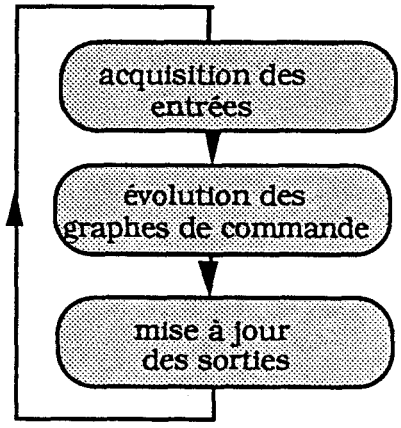
Ce cycle est le plus important du système. Il se décompose en quatre phases bouclant jusqu'à satisfaction des critères d'arrêt et concerne les règles retenues lors de l'étape précédente.

* ***Phase 1*** *: Filtrage*

Le filtrage, pour des raisons de performance, est effectué en deux temps. En premier lieu, il concerne les prémisses internes des règles. Le résultat donne un sous-ensemble de règles candidates pour le filtrage d'après leurs prémisses externes. Cette seconde opération nécessite généralement l'acquisition de la valeur d'un fait externe et conduit à un accès au réseau. Néanmoins, différentes situations permettent de s'affranchir éventuellement de la requête à travers le réseau :

- Si la valeur du fait externe a déjà été calculée lors de ce **même** cycle d'inférence, elle n'est pas remise en cause. Autrement dit, l'acquisition dynamique d'un fait externe est faite une seule fois par cycle ; la commande et le procédé sont donc artificiellement considérés comme figés pendant un cycle complet

du moteur d'inférence. Cette démarche est strictement similaire aux principes retenus dans le cadre du fonctionnement d'un automate programmable vis à vis du procédé qu'il pilote (Fig. 42). Bien que l'ensemble des entités (niveau hiérarchique, commande, procédé) soit foncièrement asynchrone, la prise en compte de leurs états respectifs s'opère à des instants privilégiés. Même si le niveau hiérarchique ne regroupe pas l'ensemble des requêtes en début de fonctionnement (cf. explications Etape 1), le comportement reste le même. Il est également identique concernant le déclenchement des conséquents externes (cf. Phase 2 et Etape 4).



commentaire :
l'évoluation des graphes de commande consiste à considérer le marquage actif et à vérifier si les réceptivités des transitions en aval de ce marquage peuvent être validées. Auquel cas, le marquage évolue d'un pas

Cycle général d'un automate

FIGURE 42

- Certaines requêtes ont été optimisées et retournent plusieurs informations simultanément. Ainsi demander l'état d'une place d'un Grafcet donne automatiquement le marquage complet de l'automate sur lequel se trouve ce Grafcet. Acquérir la valeur d'un booléen externe retourne un octet donnant l'état de huit booléens consécutifs, modulo 8 (ex : valeur du bit 12 donne la valeur des bits 8 à 15), ... Toutes ces opérations (accès au réseau, recherche des valeurs externes, mémorisation dans la base des faits externes) sont bien sûr complètement transparentes pour l'utilisateur.

* ***Phase 2*** *: Evaluation*

L'ensemble des règles retenues après l'opération de filtrage est ensuite évalué. Tous les conséquents internes sont déclenchés en "parallèle" (stratégie en largeur d'abord) et les modifications qui en résultent sont intégrées à la base des faits internes. Les conséquents externes sont mémorisés en vue d'un déclenchement ultérieur. Cette mémorisation est obligatoire car on ne remet jamais en cause la valeur d'un fait externe lors d'un même cycle d'inférence.

* ***Phase 3*** *: Restriction*

La restriction a pour objectif de supprimer certaines règles avant de continuer l'inférence. Deux critères sont retenus :

- Les règles qui ont passé avec succès la phase de filtrage et qui ont donc été évaluées sont effacées de la base de règles courantes ; ceci afin d'éviter que le système ne rentre dans une boucle infinie en sélectionnant toujours les mêmes règles et qu'il ne déclenche à chaque pas d'inférence les mêmes conséquents.

- Les règles qui ont satisfait le filtrage sur les prémisses internes mais qui ont échoué au filtrage sur les prémisses externes sont également retirées de la base courante. En effet, si une prémisse externe est fausse, elle le restera tout au long du cycle complet car, par hypothèse, un fait externe n'évolue pas pendant l'inférence. Il est donc inutile de laisser ces règles qui ralentissent inutilement le système lors de l'opération de filtrage alors qu'elles ne sont jamais retenues.

** **Phase 4** : Arrêt sur un état stable ?*

Deux critères d'arrêt terminent l'évolution estimant que les bases de faits sont arrivées à saturation :

- Toutes les règles candidates ont déjà été évaluées et ont donc été retirées de la base courante.

- La dernière opération de filtrage n'a retenu aucune règle ou les règles retenues ne possédaient que des conséquents externes et aucun conséquent interne. La base de faits internes ne subit alors plus aucune modification ; toutes les actions possibles ont donc été envisagées pour ce cyle d'inférence.

ETAPE 4 : Déclenchement des conséquents externes

De façon identique au fonctionnement d'un automate, tous les conséquents externes mémorisés lors de l'Etape 3, phase 2, sont évalués. Leur déclenchement va modifier l'état de la commande et du procédé afin qu'ils puissent évoluer en fonction des actions déduites par le niveau hiérarchique. A ce stade, le cycle complet d'évolution du système est terminé ; automatiquement, le niveau hiérarchique recommence à l'étape 1 pour intégrer le nouvel état des organes pilotés et en déduire d'autres actions.

On pourrait néanmoins, avant de relancer l'étape 1 et afin d'accroître la robustesse du système, valider à ce niveau les "acknowledge" du couple procédé / commande. Cette démarche revient à intégrer explicitement la notion de feedback dans le fonctionnement du système complet. Connaissant les faits primitifs provenant de la commande et qui, par synthèse, ont amené le niveau hiérarchique à déclencher des actions visant à les effacer, le système peut s'assurer que les requêtes extérieures ont effectivement disparu. S'il s'avèrait que certaines requêtes restaient trop longtemps affichées, cela prouverait que les actions du niveau hiérarchique sont alors ineffectives et nécessiterait dans ce cas l'activation d'un traitement d'exception.

On voit ici que le niveau hiérarchique est l'organe fédérateur qui "régule" un système réactif représenté par le couple procédé / commande.

III.6 - Conclusion

Le cycle d'inférence combine deux approches couramment employées en simulation :

- par activités,
- par évènements [BEL 85].

L'inférence sur la base de méta-règles et sur la base de règles modélise le fonctionnement par activité. L'étude de l'état des bases de faits permet de déterminer les actions à déclencher. Toute modification occasionnée à la base de faits internes traduit une évolution et nécessite de réitérer cette démarche. Ces inférences s'opèrent ainsi implicitement à temps constant vis à vis de la commande ou du procédé.

Le déclenchement des conséquents externes, puis l'effacement de leur valeur dans la base de faits externes correspond au cycle par évènement. C'est-à-dire que l'évolution du niveau hiérarchique est régie par les changements d'état des évènements extérieurs. L'évolution globale du système dans le temps dépend de cette approche par évènements.

CONCLUSION

Dans ce chapitre, nous avons mis en évidence l'intérêt de la réalisation d'un niveau hiérarchique supervisant les différents organes de commande. Nous avons par ailleurs tenté d'apporter des solutions aux problèmes soulevés par l'implantation effective, à différents niveaux. Tout d'abord, en assurant une gestion globale et cohérente des primitives banalisées de communication, nous avons pris en compte la répartition, fonctionnelle ou imposée par le procédé, de la commande. De même, les politiques de fonctionnement non explicitement précisées au sein de la commande (attribution des ressources, indéterminismes directionnels) ont été mises en œuvre lors de l'implantation, ainsi que les comportements limites (blocages) ne pouvant se contenter d'une simple observation instantanée de la commande. Enfin, en séparant les mécanismes de surveillance de la commande qu'ils contrôlent, nous avons permis d'en accroître la puissance de modélisation et l'efficacité.

La mise en œuvre d'une approche hiérarchisée dissociant le contrôle stratégique de la commande directe du procédé permet d'aboutir à une meilleure structuration de la réalisation de la commande globale. La commande des automatismes séquentiels est réservée aux automates programmables. L'ordonnancement et la supervision sont alors implantés au niveau hiérarchique. Nous proposons dans ce sens, des solutions adaptées aux problèmes complexes relevant du pilotage de cellules de production flexibles dans l'industrie manufacturière [BOU 87] sans pour autant remettre en cause les outils de commande conventionnels [BON 86].

CHAPITRE III

REALISATION ET EXEMPLE

INTRODUCTION

Ce dernier chapitre concerne la présentation d'une réalisation effective du logiciel permettant de mettre en oeuvre le niveau hiérarchique et son environnement. La première mise en oeuvre a été effectuée sur une gamme particulière de matériels et propose des solutions spécifiques qui ne pourront être systématiquement retenues pour d'autres systèmes. Après avoir décrit l'organisation générale des programmes, nous détaillerons la réalisation de la communication et l'interfaçage du niveau hiérarchique programmé en LISP avec les automates. Nous dégagerons ensuite les apports du système tout en précisant ses limites. Cette réalisation conduit à un certain nombre de conclusions préliminaires donnant les grandes orientations qui pourraient être retenues pour une amélioration sensible de l'implantation du niveau hiérarchique.

La dernière partie de ce chapitre est consacrée à la réalisation effective de la commande d'une cellule flexible. Dans ce sens, nous présenterons dans un premier temps le site initial servant de support à l'exemple. Nous décrirons ensuite rapidement, l'application des différentes phases de la méthodologie C.A.S.P.A.I.M. Enfin, nous détaillerons l'implantation réelle de la commande des automatismes séquentiels et du niveau hiérarchique.

I - REALISATION EFFECTIVE DU LOGICIEL

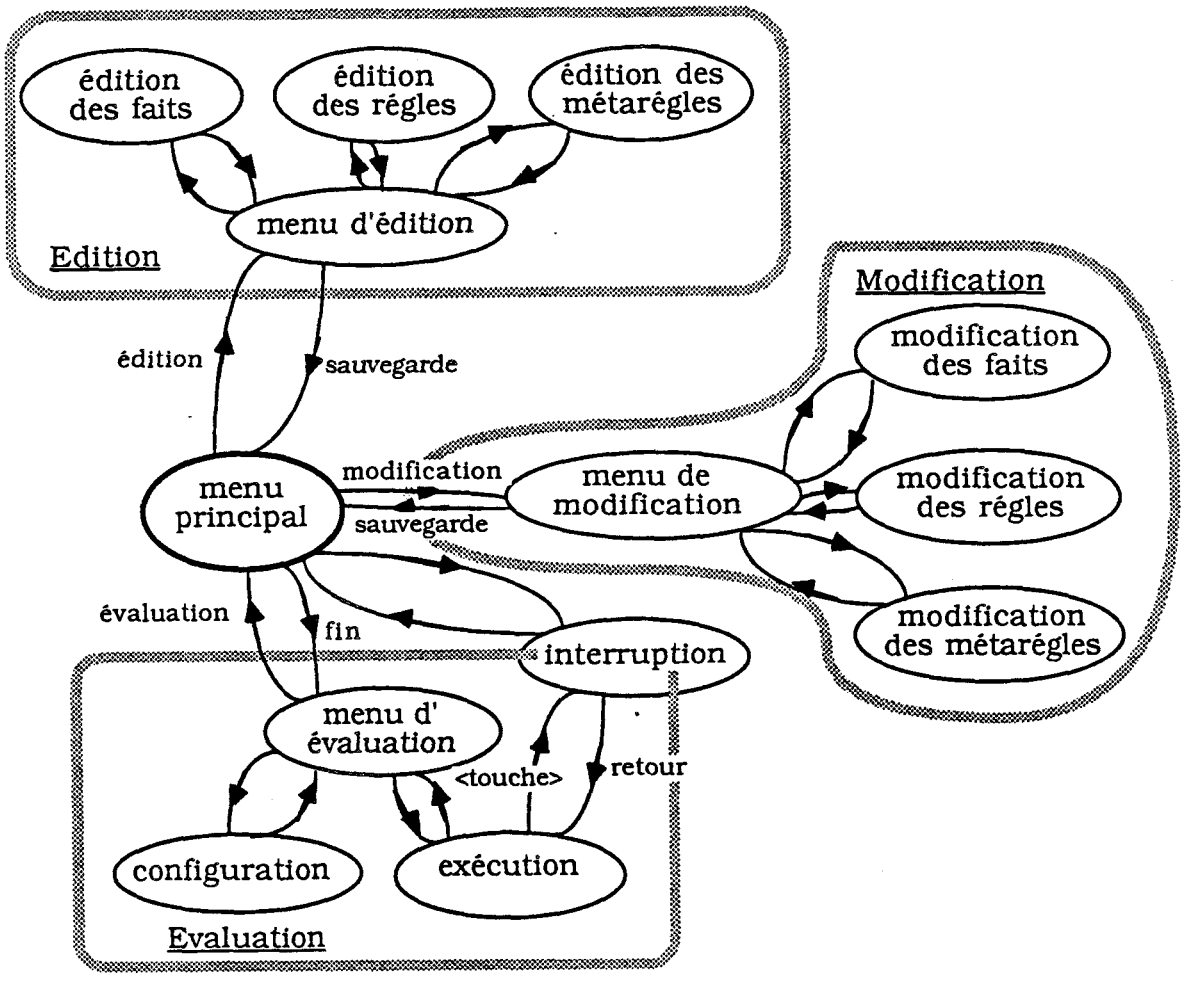
I.1 - Introduction

La structure du niveau hiérarchique a été présentée dans la troisième partie du chapitre II. Un premier prototype reprenant ces définitions a été réalisé sur VAX 750 (Digital Equipement) en utilisant le langage Le-LISP développé par l'INRIA. Ce modèle fonctionne de manière off-line (i.e. il n'est connecté à aucun processus extérieur) et a permis de mettre au point les modules de saisie d'une application et d'exécution. La maquette a validé le fonctionnement du moteur en simulant les requêtes externes sur un terminal de saisie.

La réalisation définitive a été implantée sur IBM-AT, supportant également Le-LISP, et testée on-line sur les automates programmables de la Télémécanique Série 7 via le réseau TELWAY 7. Il a fallu réaliser l'interfaçage entre le niveau hiérarchique et le réseau par l'intermédiaire de deux programmes, l'un en assembleur, l'autre en langage C. Cette démarche met en évidence l'importante indépendance du niveau hiérarchique par rapport aux réseaux utilisés. La connection à d'autres matériels utilisant des protocoles différents ne remettra a priori en cause que le programme en langage C dédié exclusivement à la gestion des requêtes et à la mise en oeuvre des protocoles de communication.

I.2 - Organisation générale de logiciel

Le logiciel est scindé en 11 modules principaux (9 en LISP, 1 en assembleur et 1 en langace C) permettant l'édition, la modification, l'archivage et l'évaluation du niveau hiérarchique. Son utilisation est régie par un ensemble de menus dont l'organisation de base est la suivante :



Organisation générale du logiciel

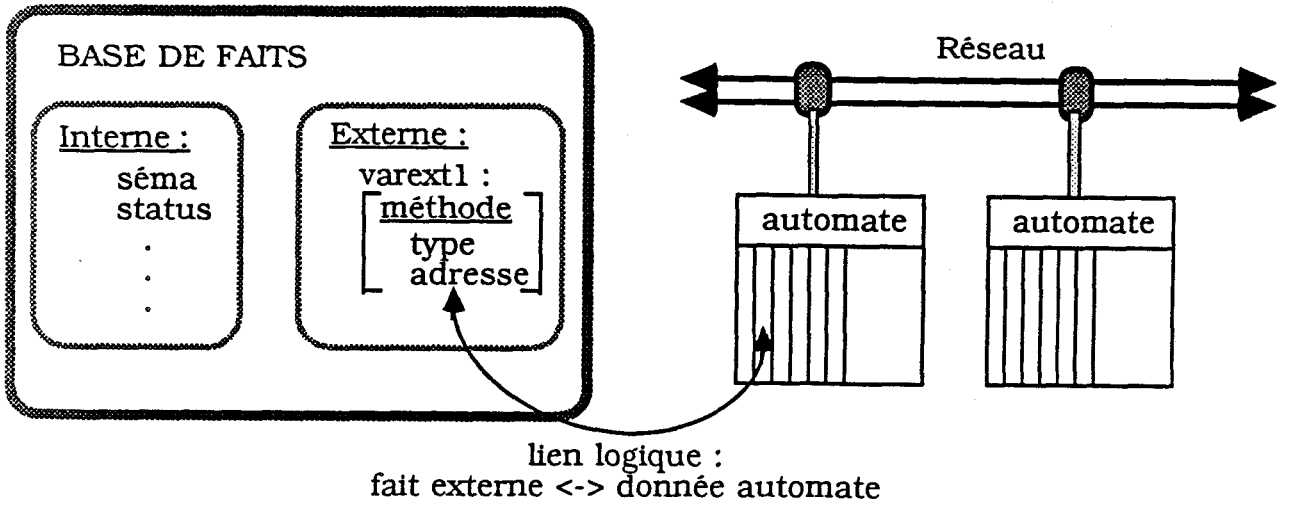
FIGURE 1

En phase d'évaluation, l'existence de l'interruption programmée permet à tout instant de suspendre l'exécution tout en sauvegardant le contexte courant, ce qui autorise la reprise "à chaud" du système en cours. De plus, l'utilisateur peut à tout moment quitter provisoirement le logiciel pour accéder à un "pseudo" toplevel LISP où il dispose cette fois de toutes les fonctionnalités de l'interpréteur LISP. Son retour au sein du logiciel restaure complètement le contexte qu'il avait quitté hormis les données qu'il aurait volontairement modifiées au niveau du pseudo-toplevel. L'interruption programmée et le pseudo-toplevel augmentent considérablement l'interactivité du système et ont été rendus possibles par la souplesse d'utilisation du langage déclaratif Le-LISP.

I.2.a - Les modules d'édition et de modification

Ces deux modules sont relativement similaires, la modification étant en fait un sur-ensemble de l'édition. La saisie procède selon un ordre logique

déterminé. L'utilisateur définit en premier lieu les variables de son système. Elles peuvent être de deux types : interne ou externe (cf chap. II.3.2) et caractérisent, soit un état interne du niveau hiérarchique (donc purement local à ce dernier), soit un état externe fonction de la commande. A une variable externe est ainsi associée la méthode permettant d'accéder à travers le réseau à sa valeur réelle stockée dans une variable d'un automate déterminé. La définition de cette méthode est automatiquement associée à la variable externe lors de sa déclaration. Elle est donc transparente pour l'utilisateur et est fonction uniquement du type et de l'adresse de la variable référencée sur le réseau.



Adéquation variable externe <—> valeur réelle

FIGURE 2

Les variables ainsi définies sont rangées dans leurs bases de faits respectives sous la forme de P-listes (liste de propriétés). Elles sont modifiables a postériori et peuvent être éditées ; l'utilisateur peut également y accéder par l'intermédiaire du pseudo toplevel. A ce niveau, il est directement sous l'interpréteur LISP et dispose donc de toute la puissance du langage déclaratif.

Après avoir déclaré les faits, l'utilisateur peut introduire les règles régissant le fonctionnement de son système. Une analyse syntaxique est dynamiquement assurée lors de la définition de ces règles. Elle en détermine la validité par rapport à la grammaire du système et vérifie de plus que les variables référencées existent bien dans les bases de faits. Ce dernier point permet de garantir, lors de l'éxécution du système l'absence d'appel à des entités inexistantes. Il assure

donc un contrôle de cohérence augmentant la fiabilité du niveau hiérarchique. L'utilisateur peut également définir des fonctions LISP qui seront évaluées au même titre que les règles de production. L'emploi de ces fonctions ne se justifie que dans l'hypothèse de traitements ou calculs complexes inadaptés à une représentation sous forme de règles de production. Dans ce cas, seule une analyse syntaxique est assurée par l'interpréteur LISP. L'utilisation des fonctions doit donc être exceptionnelle d'autant qu'elle nécessite la connaissance du langage LISP. Après leur déclaration, les règles sont rangées dans leur base et ordonnées selon la nature de leurs prémisses et conséquents. Sont ainsi évaluées en priorité les prémisses internes par rapport aux prémisses externes tandis que les conséquents externes sont mémorisés pour être déclenchés seulement à la fin du cycle d'inférence conformément au fonctionnement du moteur présenté chapitre III.5.

Enfin, les métarègles peuvent être introduites dans le système. Elles ne référencent par définition que les faits internes et les règles précédemment définies. Leur objectif est de minimiser la taille de la base de règles courantes dans une optique d'optimisation ; les performances du moteur d'inférence dépendent étroitement du nombre de règles à consulter. Une analyse de cohérence est également assurée pendant la définition d'une nouvelle métarègle.

Par rapport à l'édition, le menu de modification permet de reprendre tout ou partie des définitions précédemment établies. Corriger le système se fait néanmoins au détriment de sa cohérence. S'il est impossible d'éditer une règle référençant un fait inexistant, il est par contre tout à fait possible de détruire un fait utilisé par les règles. Le menu de modification se contente simplement de mettre en garde l'utilisateur en lui signalant les règles comportant ce fait comme variable. De même la modification du type d'une variable peut amener un comportement aberrant lors de son évaluation. En effet, l'adéquation entre l'opérateur référençant une variable et cette même variable est assurée lors de la définition d'une règle. Si par la suite, le type de la variable est modifiée, cette adéquation n'est plus vérifiée. L'exemple suivant présente ce genre de modification imparfaitement réalisée :

(put-fifo liste pièce-produite) traduit l'action d'empiler dans la fifo "liste" la nature d'une pièce mémorisée dans l'attribut "pièce-produite". La cohérence de ce conséquent est assurée lors de sa définition initiale : existence des variables liste et pièce-produite ainsi que validité de l'action put-fifo vis-à-vis de la

variable liste qui doit nécessairement être une structure de données de type fifo. Si la variable liste est détruite ou plus subtilement si son type est changé, l'évaluation du conséquent déclenchera une erreur. L'opérateur put-fifo ne pourra plus référencer la variable ou il tentera d'accéder à une structure incompatible par rapport à l'action tentée.

Les modifications, bien évidemment nécessaires lors de la mise au point, doivent donc être entreprises avec prudence et cohérence pour ne pas désorganiser la logique du système complet.

I.2.b - Le module d'évaluation

Ce module comprend essentiellement le moteur d'inférence du niveau hiérarchique qui va évaluer de façon continue les règles et les faits définis pour la supervision d'une cellule flexible. Il est configurable afin de faciliter la mise au point des heuristiques de fonctionnement. Un mode d'exécution avec trace complète permet de visualiser dynamiquement les règles qui sont en cours d'évaluation. Il détaille prémisse par prémisse le résultat de l'inférence et facilite ainsi l'interprétation du comportement du système. Ce mode trace visualise également les requêtes faites à travers le réseau pour rapatrier les valeurs des variables externes dont la connaissance est nécessaire à la poursuite de l'évaluation. Les codes des requêtes affichés à l'écran en traduisent la nature : acquisition d'un bit externe, d'un mot externe, ... et la validité : erreur ouverture canal de communication, requête acquittée, ... Ils procurent donc une assistance importante en cas de disfonctionnement du système complet. Ils permettent en particulier de mettre en évidence quel automate ne répond plus sur le réseau et surtout de valider la bonne adéquation entre une variable externe définie au niveau hiérarchique, sa méthode associée lui permettant d'accéder à sa valeur, et son image réelle dans un automate précis. En plus d'une aide à la mise au point, des statistiques sur le fonctionnement général du système sont ainsi recueillies. Deux paramètres fondamentaux vont de la sorte servir à dimensionner l'application : le nombre de règles évaluées pour un cycle d'inférence complet et le nombre de requêtes émises sur le réseau. C'est en cherchant à minimiser conjointement ces deux valeurs que l'on optimisera les performances du niveau hiérarchique. Dans ce sens, les règles devront éventuellement être redéfinies pour en diminuer le nombre ou les métarègles seront réexaminées afin d'en augmenter la puissance. Parallèlement les requêtes sur le réseau pourront être améliorées par une connaissance approfondie de leur mécanisme de

fonctionnement. L'exemple de l'acquisition d'une variable bit d'un automate est à cet égard significatif. En effet, cette requête rapatrie à la fois la valeur de la variable de type bit d'adresse x mais également les valeurs des variables de même type et d'adresse comprise entre |8 x (x module 8)| et |(8 x (x module 8)) + 7| (exemple : pour la variable bit d'adresse 27, on récupère également les valeurs des bits 24 à 31). S'il est ainsi possible de regrouper judicieusement les variables significatives par groupe de huit, on diminue le nombre de requêtes par ce même facteur.

De façon similaire au mode trace, l'utilisateur peut configurer son application de telle sorte que les informations jugées essentielles soient enregistrées de façon continue dans un fichier. Ce dernier retracera donc tous les évènements importants survenus en cours de production. Il facilitera la réalisation de statistiques concernant les flux, les taux d'engagement mais répertoriera également les incidents concernant les organes de commande : de la "disparition" d'un automate sur le réseau jusqu'à l'arrêt du niveau hiérarchique.

Enfin, à tout moment pendant la production, l'utilisateur peut suspendre le fonctionnement du moteur d'inférence par l'intermédiaire d'une interruption programmée. Il a alors accès à l'intégralité du niveau hiérarchique ; toutes les fonctionnalités du système sont à sa disposition. Il peut donc envisager à l'extrême de modifier "on-line" le niveau hiérarchique lui-même : bases de faits et bases de règles. Plus raisonnablement, il a la possibilité de visualiser les valeurs courantes du système (aussi bien les valeurs des faits internes que celle des faits externes auxquels il accède de façon transparente à travers le réseau) et également de les forcer tout aussi simplement. La reprise du fonctionnement se fait en restaurant le contexte précédent l'interruption à l'exclusion des modifications volontairement réalisées. Cette opportunité ne doit bien évidemment être utilisée qu'à bon escient et n'est envisageable que si la commande des automatismes est à même de fonctionner temporairement sans l'existence du niveau hiérarchique quitte à s'arrêter provisoirement dans un mode de repli en attendant sa reprise. Une telle interactivité est une des caractéristiques du LISP et a été déterminante dans le choix de ce langage. Elle est obtenue en partie au détriment de la vitesse d'exécution quoique ce dernier point ne soit pas significatif en ce qui concerne le niveau hiérarchique. En effet, les performances générales de notre système ne sont pas bridées par l'interpréteur LISP mais essentiellement par les spécificités du réseau local utilisé et les temps de réponse induits à chaque requête. L'utilisation d'un autre langage plus rapide

en théorie n'aurait donc pas diminué de façon notable les temps de réponse du système mais aurait par contre réduit très sensiblement l'interactivité avec l'utilisateur.

I.3 - IMPLANTATION INFORMATIQUE

I.3.1 - Introduction

La diminution du coût des machines programmables a permis d'appréhender les automatismes d'une façon nouvelle. Les systèmes d'automatismes peuvent être aujourd'hui réalisés suivant le concept d'architecture distribuée. Dans ce type de structure, chaque système est composé de sous-ensembles fonctionnels et autonomes. Ces derniers ont toutefois besoin de communiquer entre eux pour coordonner leurs actions.

Deux solutions peuvent être utilisées pour résoudre ces besoins :

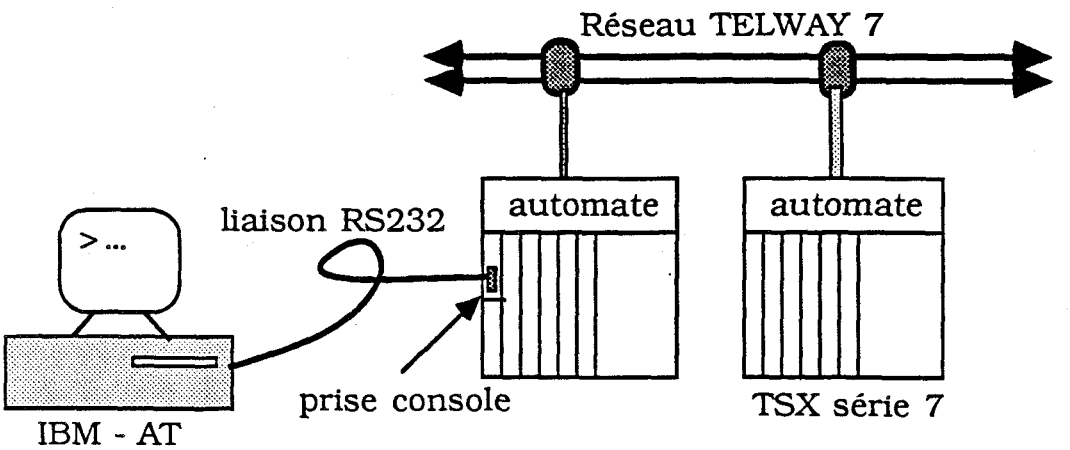
(i) Les liaisons séries asynchrones qui nécessitent d'incorporer dans le programme utilisateur un traitement spécifique.

(ii) Les réseaux locaux industriels qui, grâce à leur spécificité, conduisent à une simplification d'emploi aussi bien logicielle que matérielle.

C'est vers cette deuxième solution que se sont orientés les développements de la Télémécanique ces cinq dernières années. Parallèlement à la sortie de leur nouvelle gamme d'automates TSX série 7, a été proposé le réseau local homogène TELWAY 7 |RES 85|. Ce réseau est réalisé selon une topologie de bus ; il est géré par un maître flottant (technique qui autorise d'ajouter et d'enlever des stations sans pour autant en perturber le fonctionnement) selon la méthode du passage de jeton. Cette méthode permet de garantir un temps de réponse maximal pour la communication : elle est ainsi déterministe. Enfin la vitesse de transmission est égale à 19 200 bands. On peut remarquer que les solutions retenues sont tout à fait logiques vis à vis de l'environnement dans lequel est utilisé ce réseau. L'aspect sécurité de fonctionnement et temps de réponse garanti ont prévalu pour une utilisation dans un environnement industriel au détriment de techniques, éventuellement plus performantes, mais moins sures d'emploi.

Ce réseau permet de raccorder jusqu'à 16 automates et assure essentiellement deux services. En premier lieu, il permet d'étendre virtuellement mais de façon limitée le bus mémoire à l'ensemble des stations connectées. Chaque automate a ainsi accès à une zone mémoire commune de 64 mots de 16 bits. A chacun sont alloués 4 de ces mots auxquels il a accès en écriture. Ceux des autres stations ne lui sont accessibles qu'en lecture. Cette zone mémoire commune est mise à jour périodiquement dans toutes les stations connectées. Ce mode de communication, pour être très simple, n'en est pas moins limité. Les seules informations accessibles sur le réseau sont celles que l'utilisateur met par programme dans ses mots communs. Aucune station ne dispose de l'initiative d'aller se renseigner sur les variables d'autres automates.

Le second service permet l'utilisation à distance de toutes les fonctions du terminal de programmation. Une console Télémécanique connectée physiquement à une station quelconque du réseau peut ainsi communiquer logiquement avec toute autre station. Tous les modes d'utilisation du terminal sont accessibles et en particulier le mode "réglage" qui offre la possibilité d'accéder de façon complètement transparente vis à vis des automates, en lecture et écriture à toutes leurs données internes. C'est ce second service que nous avons utilisé pour réaliser le niveau hiérarchique. La console de programmation Télémécanique a été remplacée par un IBM-AT qui par l'intermédiaire du logiciel LINK7PC émule le fonctionnement en mode réglage. Le réseau, initialement homogène, est maintenant hétérogène et toute la puissance de l'I.B.M. est disponible pour la supervision de la commande répartie sur les différents automates.



Connection hétérogène sur le réseau.

FIGURE 3

I.3.2 - Le logiciel LINK7PC

Ce logiciel permet l'intégration de tout ordinateur IBM-PC à la structure de communication des automates TSX série 7. Fourni par la Télémécanique, il gère le protocole de communication disponible sur la prise "terminal" de tout automate. En émulant le mode de fonctionnement réglage, il permet de visualiser, de modifier, de forcer des variables telles que des bits, mots, blocs fonctions, ... jusqu'au marquage des grafcets. L'IBM est le maître logique de la liaison. L'automate sur lequel il est physiquement connecté ne fait que répondre aux requêtes élémentaires qui lui sont adressées. Si ces requêtes ne le concernent pas directement, il sert alors de passerelle pour les retransmettre sur le réseau à l'intention des autres automates. Lors d'une réponse émise sur le réseau, il retransmettra en retour les données vers l'IBM. Toute la gestion de la communication, ainsi que l'interprétation des requêtes sont complètement transparentes pour l'utilisateur. A aucun moment, il n'a à se préoccuper lors de la programmation des automatismes, des requêtes que l'automate serait susceptible de recevoir en cours de fonctionnement. Ce traitement est automatiquement assuré par le moniteur de l'automate qui est à même de dialoguer sur le réseau, d'interpréter et de répondre aux requêtes tout en assurant le "scheduling" des grafcets. Cette simplicité de mise en oeuvre dissocie ainsi complètement le niveau commande de la supervision. Elle a pour effet de bien scinder les différents niveaux de programmation en supprimant toute interférance logicielle.

Le logiciel LINK7PC consiste concrètement en un ensemble de fonctions écrites en langage C et livrées sous la forme d'une bibliothèque. Son utilisation doit se faire a priori à partir d'un programme principal en C qui appelle, selon un ordre logique fixé, les fonctions de la bibliothèque. L'établissement du dialogue entre le PC et l'automate repose initialement sur la fonction TSXOPEN. Cette primitive permet d'initialiser la ligne à partir de laquelle l'utilisateur va dialoguer avec les automates. La ligne ouverte, il peut émettre des requêtes et attendre des réponses par l'intermédiaire de la fonction TSXREP. C'est cette dernière qui est en permanence utilisée dans notre système pour renseigner la base de faits externes de la valeur des variables situées dans les automates. L'appel de cette fonction se réalise en C de la façon suivante :

iret = TSXREP (n° ligne, bufferinput, bufferoutput)

où les paramètres ont pour signification :

- **n° ligne** : correspond au numéro de la voie série sur laquelle vont s'effectuer les échanges (pratiquement : toujours à 1).

- **bufferinput** : adresse d'une structure de donnée passée comme paramètre dont les zones à renseigner sont les suivantes :

  (i) - type : type de message (FO en hexadécimal caractérise les messages en mode réglage).

  (ii) - exp : adresse de l'expéditeur (renseignée automatiquement par TSXOPEN).

  (iii) - dest : adresse du destinataire (numéro logique de l'automate avec lequel on souhaite communiquer à travers le réseau).

  (iv) - Codereq : code la requête à émettre.

  (v) - data : données à fournir selon la nature de la requête.

- **bufferoutput** : adresse d'une structure de type sortie. C'est dans cette structure que l'utilisateur récupère la réponse à sa requête ; elle est donc automatiquement renseignée à la fin de l'exéxcution de la fonctin TSXREP.

- **iret** : code indiquant le résultat de la fonction. Sa valeur permet de savoir si la communication s'est effectivement bien passée ou s'il y a lieu de réïtérer la requête.

A chaque type de requête correspond un code (codereq) spécifique caractérisant la nature du traitement que l'on souhaite réaliser (codereq = 0 pour lecture d'un bit automate, codereq = 4 pour lecture d'un mot, ...).

La bibliothèque de fonctions C, LINK7PC, constitue le noyau minimal mis à notre disposition pour dialoguer avec les automates de la Télémécanique. Leur utilisation à partir d'un programme principal en C, sans être simple, ne pose pas de problème particulier. Néanmoins, comme notre niveau hiérarchique a été développé en Le-LISP pour des raisons précédemment détaillées (homogénéïté des travaux du laboratoire, interactivité maximale, ...) il a fallu interfacer ces deux langages fondamentalement différents, interprété pour l'un et compilé pour l'autre.

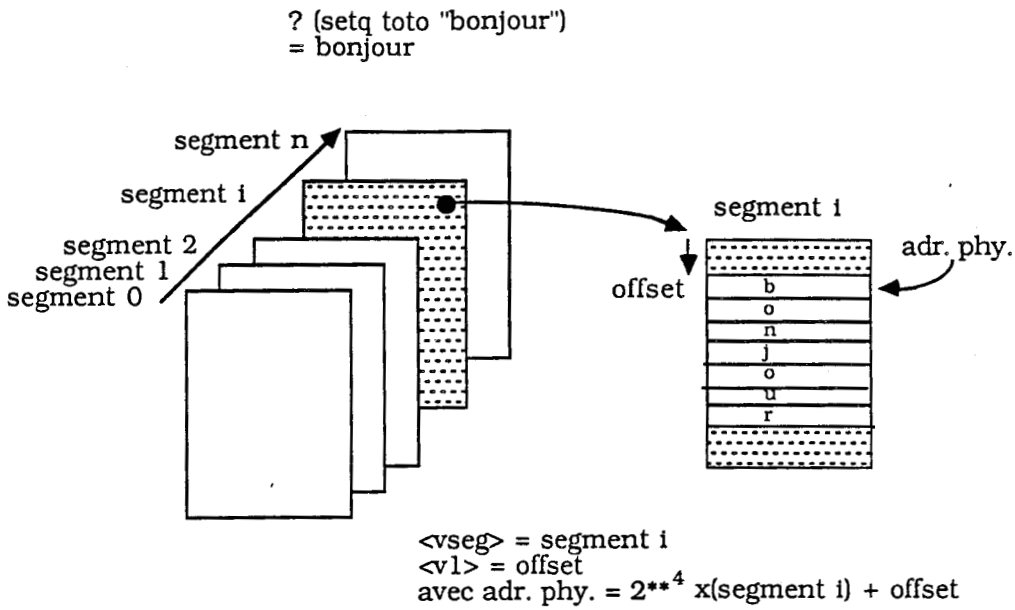
I.3.3 - Interfaçage Le-LISP <—> langage C

Paradoxalement, Le-LISP est par essence un langage essentiellement portable. Son interprète a été entièrement écrit dans le langage de la machine virtuelle LLM3. Tout programme écrit en Le-LISP tournera immédiatement sur n'importe quelle machine supportant Le-LISP. Cette portabilité est obtenue au détriment des spécificités de chaque machine ce qui fait par exemple que Le-LISP n'est pas un langage adapté à l'écriture de driver d'entrées/sorties. Néanmoins, certaines extensions sont apportées à chaque système LISP pour bénéficier des opportunités disponibles sur l'ordinateur hôte. Ces fonctions supplémentaires sont données en annexe dans la documentation technique pour bien signifier que leur utilisation se fera au détriment de la portabilité de l'application. C'est l'une d'entre-elles que nous avons utilisé pour interfacer indirectement Le-LISP avec le langage C sur l'IBM-AT.

Cette fonction |(INT86X <n> <v1> <v2> <vseg>)| est une "Subr" à 4 arguments. Elle permet uniquement d'appeler les routines internes de la machine. Elle déclenche une interruption logicielle qui autorise au programme de passer l'adresse absolue de ses paramètres. Les quatre arguments de cette fonction sont :

- <n> : n est le numéro de l'interruption DOS que l'on désire appeler.
- <v1>, <v2> : sont deux vecteurs qui contiennent respectivement la valeur des registres du processeur avant l'appel et après l'appel de l'interruption. Concrètement, ces registres servent à passer et à récupérer l'offset de l'adresse d'une chaîne de données.
- <vseg> : il contient quant à lui la valeur du segment où se trouve ces données (fig. 4).

A ce stade, il est possible de déclencher une interruption logicielle DOS et de passer les adresses des arguments en paramètre. Il convient maintenant de récupérer cette interruption pour faire en sorte qu'elle corresponde à un appel aux fonctions de la bibliothèque LINK7PC. Détaillons en premier lieu le mécanisme interne du fonctionnement d'une interruption DOS |TRI 84|. Lors du chargement du système d'exploitation au démarrage du système, une table d'interruptions est obligatoirement chargée en mémoire vive et son remplissage



Détermination de l'adresse physique d'une entité LISP.

FIGURE 4

est une des toutes premières tâches du moniteur. Cette table occupe obligatoirement les premiers 1024 octets de la mémoire centrale (de 0 à 3FFH). Elle se compose de 256 pointeurs contenant les adresses des procédures correspondant à chaque type (fig. 5). La règle pour faire correspondre un pointeur à un type est simple (le type est en fait le numéro de l'interruption <n>) : il est multiplié par quatre et donne l'adresse du pointeur. Ce pointeur comprend 4 octets (deux pour le segment et deux pour l'offset) et correspond à l'adresse réelle où se situe la procédure d'interruption à exécuter.

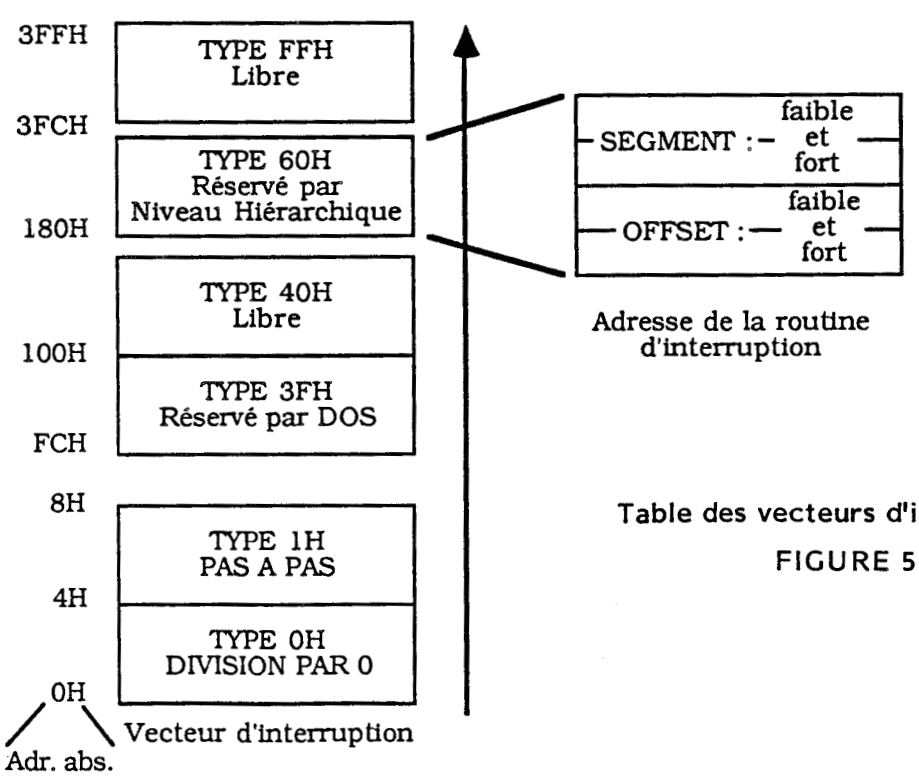


Table des vecteurs d'interruption

FIGURE 5

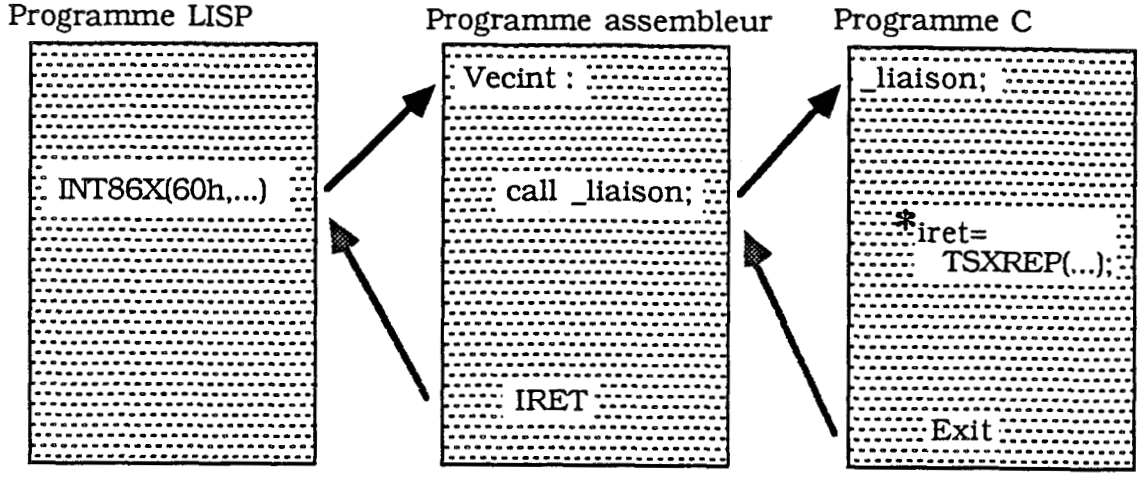
Dans notre cas particulier, nous utilisons le type 60H qui n'est pas réservé par le DOS. C'est donc cette valeur que nous donnons au paramètre <n> pour la fonction INT86X. Il faut maintenant réussir à faire correspondre les 4 octets désignés par le pointeur 60H à l'adresse d'une routine qui nous permette d'appeler les fonctions du logiciel LINK7PC. Ce dernier point est assuré par un programme en assembleur que nous avons développé et qui modifie la table des vecteurs d'interruption. Son principe en est simple : il installe de façon résidente en mémoire centrale une routine Vecint qui est l'interface entre Le-LISP et les fonctions du logiciel LINK7PC. Ce faisant, il modifie la table des vecteurs d'interruption pour que type 60H corresponde justement à l'adresse de la routine Vecint. La structure de ce programme est la suivante :

PP : programme en assembleur
call Vecini
sort en restant résidant

Vecini : subroutine en assembleur
substitue au vecteur d'interruption 60H l'adresse de la routine Vecint
RET

Vecint : subroutine en assembleur
permute les paramètres de la pile LISP vers la pile C
call-liaison ; point d'entrée du programme en C
permute les paramètres de la pile C vers la pile LISP
IRET

Une fois le programme assembleur et le C installés de façon résidente, l'instruction LISP INT86X a pour effet de passer directement la main au programme Vecint qui lui-même se charge de mettre en forme les paramètres pour le programme C. Le passage de ces paramètres est réalisé par adresse et autorise une modification directe des variables utilisées depuis Le-LISP par le langage C. Du temps est ainsi économisé et il n'existe pas de limitation de taille pour les zones mémoires passées par adresse. La figure 6 reprend les principales étapes de la communication inter-langages. Ce mécanisme, s'il peut paraître lourd et complexe, n'en est pas moins très performant. Tout le code nécessaire est résident en mémoire centrale et est référencé par une seule interruption. Le mécanisme d'appel est immédiat et n'induit aucun temps d'attente. Seule la communication avec les automates et via le réseau ralentit l'évolution de l'ensemble.



*communication via RS232 puis réseau Telway 7

Interfaçage des différents langages

FIGURE 6

Le passage des paramètres par adresse de Le-LISP vers l'assembleur a néanmoins nécessité de surveiller le fonctionnement du "Garbage-Collector". En effet, les zones mémoires qui contiennent les objets LISP sont allouées dynamiquement. Quand l'une de ces zones est saturée, un utilitaire connu sous le nom de garbage-collector est automatiquement appelé pour récupérer les objets inutilisés. Sa fonction consiste en une réorganisation générale des données LISP en mémoire. Il possède donc la faculté de déplacer les entités en changeant ainsi dynamiquement leurs adresses physiques. Déclencher un garbage-collector juste avant la fonction INT86X perturbe complètement le mécanisme du passage de paramètres par adresse. Ces dernières sont normalement calculées précédemment et leurs valeurs ne correspondent plus à leurs implantations réelles après le garbage-collector. La mise au point de la communication a ainsi nécessité la réécriture de la fonction GCALARM qui est une interruption lancée automatiquement par le système après chaque récupération de la mémoire. Son nouveau code permet de dérouter l'exécution de l'instruction INT86X pour éviter une erreur fatale et relance la détermination des nouvelles adresses des variables avant de les passer en paramètre à l'assembleur.

1.3.4 - Conclusion

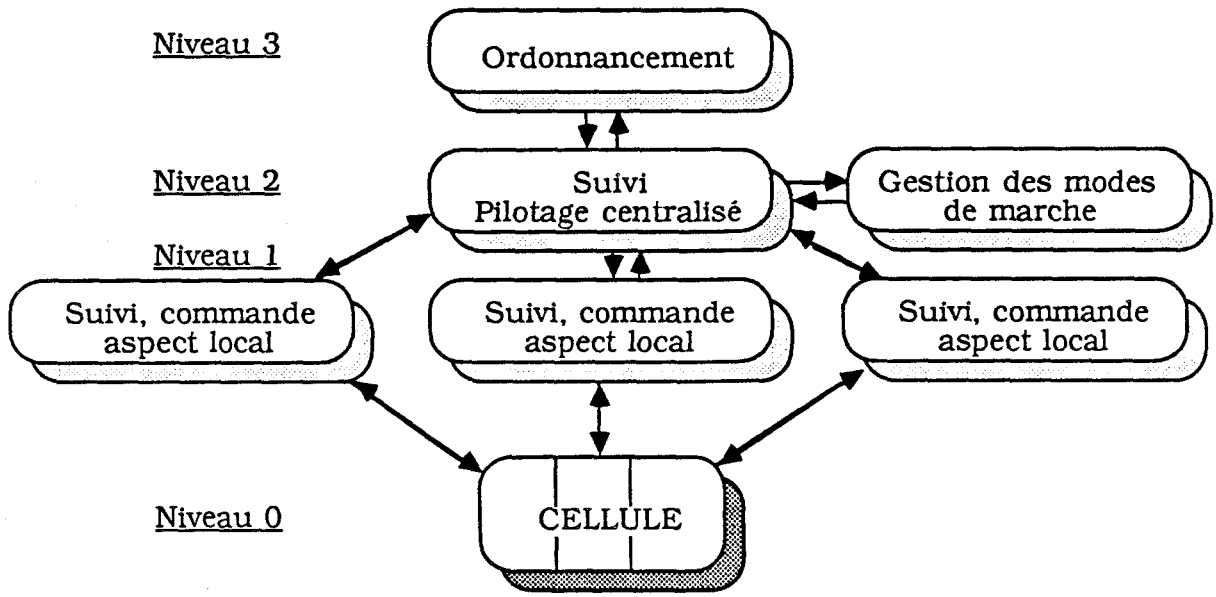
La connection du niveau hiérarchique avec les automates n'a nécessité l'utilisation que d'une seule instruction LISP spécifique à l'ordinateur utilisé. Sa portabilité sur un autre matériel est donc garantie, seule l'instruction INT86X devra être remplacée. Il faudra bien sur que le développeur réécrive le driver d'entrées/sorties dans un langage de son choix afin de pouvoir dialoguer sur le réseau. Néanmoins, ces modifications, externes au niveau hiérarchique, sont classiques à réaliser et se feront à moindre coût.

Il ressort de cette mise en oeuvre que le langage Le-LISP n'est pas seulement puissant par son symbolisme et fondamentalement interactif. Il est de plus ouvert et interfaçable avec d'autres langages. Son utilisation dans un environnement industriel est donc tout à fait envisageable. Ses deux seules lacunes principales demeurent encore sa diffusion restreinte à l'heure actuelle et sa vitesse d'exécution due à l'interprétation qui le situe sensiblement en retrait des langages compilés.

I.4 - CONCLUSION

La réalisation est aujourd'hui effective sur un IBM-AT connecté par RS232 à la prise console d'un automate Télémécanique. L'émulation du mode réglage donne accès à toutes les données internes des grafcets via, cette fois, le réseau Telway 7. Pour fiable que soit cette solution, elle n'en comporte pas moins certaines limites. En premier lieu, la connection du niveau hiérarchique est tributaire du bon fonctionnement de l'automate sur lequel il est physiquement relié. Un réseau de niveau 2 se doit d'être réellement hétérogène afin d'en augmenter la puissance ; la connection s'effectue alors directement sur le cable par l'intermédiaire d'un coupleur et n'utilise plus un automate, chargé de la commande des automatismes par ailleurs, comme passerelle. C'est une des conditions nécessaires au développement de la supervision évoluée d'automatismes industriels. Par contre, la méthode d'accès au réseau n'est pas obligatoirement fondamentale. La technique du passage de jeton, déterministe par construction, permet de garantir un temps de réponse maximal a priori indépendant de la charge. Il se peut cependant que ce temps de réponse soit incompatible avec les délais que l'on souhaite respecter. Une technique de type CSMA/CD (Carrier Sense Multiple Acces with Collision Detection), bien que non déterministe, pourra très bien convenir là où le passage du jeton est insuffisamment performant. Il suffit pour cela d'estimer l'importance de ses communications et d'en faire une simulation. Les dix Mégabauds d'un cable éthernet permettent dans la plupart des cas d'éviter une collision. Il en résulte une communication plus rapide qu'avec un jeton même si aucune preuve mathématique ne vient étayer la constatation. La Société APTOR commercialise par exemple un réseau local utilisant une technique d'accès de type CSMA/CD. Elle propose à toutes fins utiles une solution employant un double cable dont l'un est utilisé pour les messages courants tandis que l'autre est réservé aux messages urgents. Elle "garantie" de la sorte un temps maximum de communication et augmente parallèlement la fiabilité en doublant le canal du média |APT 85|.

La deuxième limite inhérente au niveau hiérarchique implantée sur IBM-AT réside dans l'aspect monotâche de la réalisation. Tributaire du système d'exploitation DOS, nous n'avons pu réaliser qu'un ensemble monotâche cyclique. L'existence de la fonction parallel sous LISP n'a pu apporter de réelle solution à ce problème. Cette émulation de multitâches bien qu'efficace, est gourmande en temps calcul. De plus, les 640 koctets de mémoire centrale gérés par le DOS sont insuffisants pour faire réellement cohabiter plusieurs applications sous LISP. Il en résulte au mieux des garbage-collectors incessants, jusqu'au déclenchement d'une erreur fatale signifiant que l'une des piles du système est pleine. Une réalisation monotâche a ainsi pour effet pernicieux de mettre à plat tous les problèmes rencontrés pour la supervision des organes de commande. Par contre un développement multitâches permet de nettement dissocier les fonctions du niveau hiérarchique par secteur d'intérêt et niveau d'abstraction (cf. fig. 7).



Décomposition en niveaux

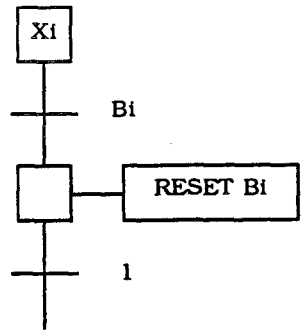
FIGURE 7

Le niveau 0 correspond à la cellule physique par elle-même. Sa commande est explicitement dissociée lors de la répartition des graphes sur les différents organes informatiques. Dans un environnement multitâches un premier processus (ou ensemble de processus) peut être défini. Il correspond au niveau 1 et est dédié à la supervision locale de la commande. Un processus peut donc contrôler le mode de marche normale d'un ou plusieurs automates. Il est à même de décider ponctuellement l'attribution d'une ressource locale ou de lever un indéterminisme directionnel simple dont il possède toutes les données. Ces processus de niveau

1, limités dans leurs traitements, sont par contre très rapides et susceptibles de réguler dynamiquement les besoins élémentaires de la commande. Au niveau 2, sont par contre répercutés tous les traitements nécessitant une vision globale de la production. Cela concerne en particulier, toutes les prises de décision impossibles à satisfaire au niveau précédent. La perception générale de l'ensemble de la production permet ici d'équilibrer la charge de la cellule en jouant sur les différents flux. Cette action se concrétise par l'émission de consignes au niveau inférieur qui vont pondérer les algorithmes ou heuristiques résolvant les indéterminismes locaux. Parallèlement au pilotage centralisé, un module indépendant est dédié à la gestion des modes de marche. L'intérêt de dissocier le pilotage centralisé de la gestion des modes de marche réside dans le fait que l'on peut utiliser deux mises en oeuvre différentes. Où le pilotage centralisé, à partir d'informations à sa disposition, cherche à déduire le maximum de faits et utilise un moteur en chainage avant pour saturer sa base, il peut être nécessaire pour la gestion des modes de marche de disposer par contre d'un moteur en chainage arrière. Ainsi dans une optique de sécurité, le moteur pourra focaliser son inférence sur un fait observé pour en déduire aussi bien les causes que les actions à réaliser pour y remédier. Les traitements à assurer au niveau 2, de natures diverses et opposées, justifient ainsi l'utilisation de deux processus indépendants pouvant néanmoins partager des informations élémentaires mais dont les algorithmes de décision sont différents. Le niveau 3 est dédié à une planification à plus long terme. Il organise un ordonnancement de la production d'après les plans de fabrication. Il peut ainsi demander au niveau 2 de favoriser la production d'une gamme donnée pour en augmenter la cadence afin de répondre aux ordres de lancement impératifs. De plus, par sa vision non limitée à la production instantanée, il peut détecter les ordres qui amènent un engorgement de la production jusqu'au blocage de la cellule.

Cette répartition en process dissociés possible dans un environnement multitâches structure les problèmes par centre d'intérêt. Elle doit faciliter la définition du niveau hiérarchique en le décomposant en modules dont la mise en oeuvre est adaptée à la nature des traitements à réaliser. La cohérence ainsi que les performances de l'ensemble y ont tout à gagner. En plus de la décomposition fonctionnelle du niveau hiérarchique, un processus détaché peut être dédié à la communication et à l'acquisition des données entre le niveau de la commande et celui de la supervision. Les problèmes typiquement informatiques et dont les solutions dépendent du matériel sont alors nettement dissociés du niveau hiérarchique. La portabilité de l'ensemble est de la sorte accrue et la maintenance facilitée.

Le troisième point important de la réalisation effective concerne le mode d'échange des informations utilisé entre le niveau hiérarchique et la commande. Le principe de la communication est ici basé sur le concept de surveillance. Toute acquisition ou modification de données est exclusivement à l'initiative du niveau hiérarchique. La réalisation, en émulant le mode réglage des automates, a permis de conduire les transferts de façon complètement transparente pour les graphes de commande. Jamais l'utilisateur ne doit se préoccuper au niveau de ses grafcets de programmer un quelconque protocole de communication. Ce mode espion permet à tout moment de rapatrier l'état de la commande quelle que soit sa configuration. Il est ainsi tout à fait adapté à la sureté de fonctionnement car il est à même d'appréhender une situation erronée et d'y remédier automatiquement. De plus toute nouvelle stratégie de fonctionnement définie a postériori est envisageable sans pour autant nécessiter une quelconque modification du séquencement des automatismes. Par contre, l'asynchronisme total de la réalisation nécessite certaines précautions. Un état dont l'information est nécessaire au niveau hiérarchique doit obligatoirement être mémorisé. L'attribution d'une ressource présentée au chapitre 2 en est l'exemple type (fig. 8). La variable Bi en amont de la ressource critique est initialement fausse, le graphe est alors figé jusqu'à ce que le niveau hiérarchique prenne conscience de ce marquage et autorise l'entrée dans la section critique en forçant la valeur de la variable.



Mécanisme d'accès

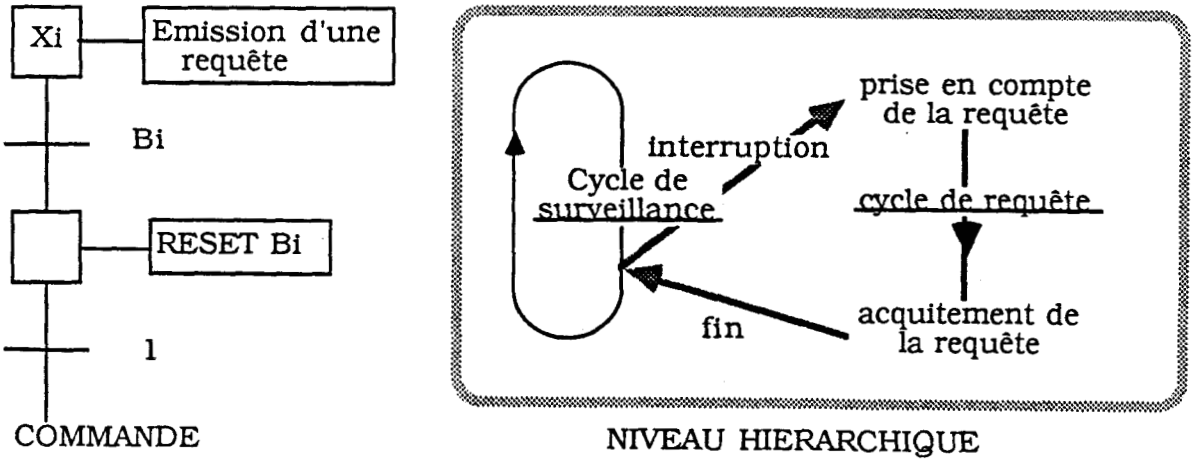
FIGURE 8

Cette mise en oeuvre comporte deux handicaps principaux :

(i) - il faut nécessairement que les règles d'attribution de la ressource soient scrutées en permanence par le moteur d'inférence. Cela diminue les performances du système inutilement d'autant que leur utilisation n'est que très ponctuelle.

(ii) - il peut se passer jusqu'à un cycle complet du moteur avant que l'on ne prenne en compte l'information. La commande est donc ralentie alors que la situation n'est même pas forcément conflictuelle.

Un mécanisme de communication par requêtes avec accusés de réception apporterait pour ce genre de problème une amélioration notable des performances de l'ensemble niveau hiérarchique-commande pour la conduite des processus en mode de marche normale. En effet, lors de la définition de la commande, toutes les situations conflictuelles (ressources, indéterminismes directionnels, ...) ont été détectées et répertoriées. Il est donc aisé de prévoir au niveau des grafcets et pour chaque situation nécessitant l'intervention du niveau hiérarchique, l'émission d'une requête. Cette dernière aurait pour effet d'interrompre l'inférence des règles de surveillance et de déclencher l'évaluation d'un cycle de règles de requête (fig. 9). Les règles qui traitent des situations conflictuelles ponctuelles ne sont donc plus évaluées qu'épisodiquement sur interruption et la prise en compte d'une requête est immédiate : l'ensemble gagne donc en performance de façon notable. Les deux modes de fonctionnement sont ainsi complémentaires : les requêtes correspondent à la gestion de situations prévues et répertoriées tandis que la surveillance par une scrutation continue détecte les défauts du système.



Coopération surveillance/requête

FIGURE 9

La réalisation informatique du niveau hiérarchique est étroitement tributaire des matériels disponibles. L'évolution incessante des micro-ordinateurs doit permettre dans un avenir proche l'intégration des points présentés

précédemment. En particulier, l'annonce d'O.S.2 comme système d'exploitation multitâches avec des fonctionnalités orientées temps réel semble correspondre parfaitement aux extensions futures du niveau hiérarchique. Parallèlement, l'ouverture hétérogène des réseaux ainsi que l'augmentation de leurs performances laissent entrevoir des communications dont les temps de réponse sont compatibles avec les performances des process pilotés. Il faut néanmoins que ces évolutions se fassent dans le sens d'une ouverture des systèmes de commande afin que le niveau hiérarchique puisse toujours accéder en lecture et écriture aux données internes des graphes conduisant les automatismes.

II - EXEMPLE DE REALISATION

II.1 - Introduction

Dans cette seconde partie, nous présentons un exemple de réalisation de la commande mettant en valeur les fonctions et l'intérêt d'une structure hiérarchique pour le pilotage d'une cellule flexible. L'objectif final est la production de différents types de pièces usinées sur deux machines outils à commande numérique organisées en cellule de production flexible.

L'implantation séparée et interactive du niveau hiérarchique se justifie par les arguments principaux suivants :

(i) - la dimension du système de commande de la cellule flexible demande plusieurs automates sur lesquels sont nécessairement implantés de façon répartie les graphes de commande respectifs. De ce fait, la coopération effective de ces différents outils nécessite une coordination externe de niveau supérieur.

(ii) - la programmation selon un langage impératif des A.P.I. du réseau n'autorise que très difficilement, en cours d'exploitation, des modifications de programme. Ces dernières relèvent pourtant du caractère flexible que doit prendre en compte la production (modifications d'une ou plusieurs gammes de fabrication, choix d'une stratégie de pilotage plus efficace, ...).

De ce point de vue, l'interactivité du niveau hiérarchique permet d'assurer aux différents utilisateurs une totale transparence des outils tout au long du cycle de vie du projet, qu'il s'agisse de conception ou d'exploitation. L'exemple qui illustre cette démarche a été réalisé sur un ensemble informatique constitué de trois TSX67 et d'un réseau TELWAY de la Télémécanique, ainsi que d'un IBM-AT dédié à la supervision et d'un BFM 186 animant un synoptique traduisant l'état courant de la cellule.

Après avoir présenté les différents constituants de la partie opérative et défini les notations retenues, nous détaillerons brièvement les différentes étapes de la méthodologie C.A.S.P.A.I.M. qui permettent d'aboutir au modèle développé et validé de la commande exprimé sous la forme de RdPSAC. Nous aborderons

ensuite la mise en oeuvre de l'implantation répartie et conjointement sa traduction en grafcet. Parallèlement à l'implantation, nous détaillerons les règles utilisées par le niveau hiérarchique qui autorisent la coopération entre les différents automates et traduisent le caractère flexible de la cellule.

II.2 - Présentation de la cellule [MAY 87]

La cellule flexible qui sert de support à notre exemple, est implantée dans le département Génie Mécanique de l'I.D.N. Ses différents constituants présentés figure 10 pour l'implantation physique et figure 11 pour la description fonctionnelle simplifiée sont les suivants :

(i) - 2 tables de transfert faisant office de tampons et notées **E** pour l'entrée des pièces et **S** pour la sortie.

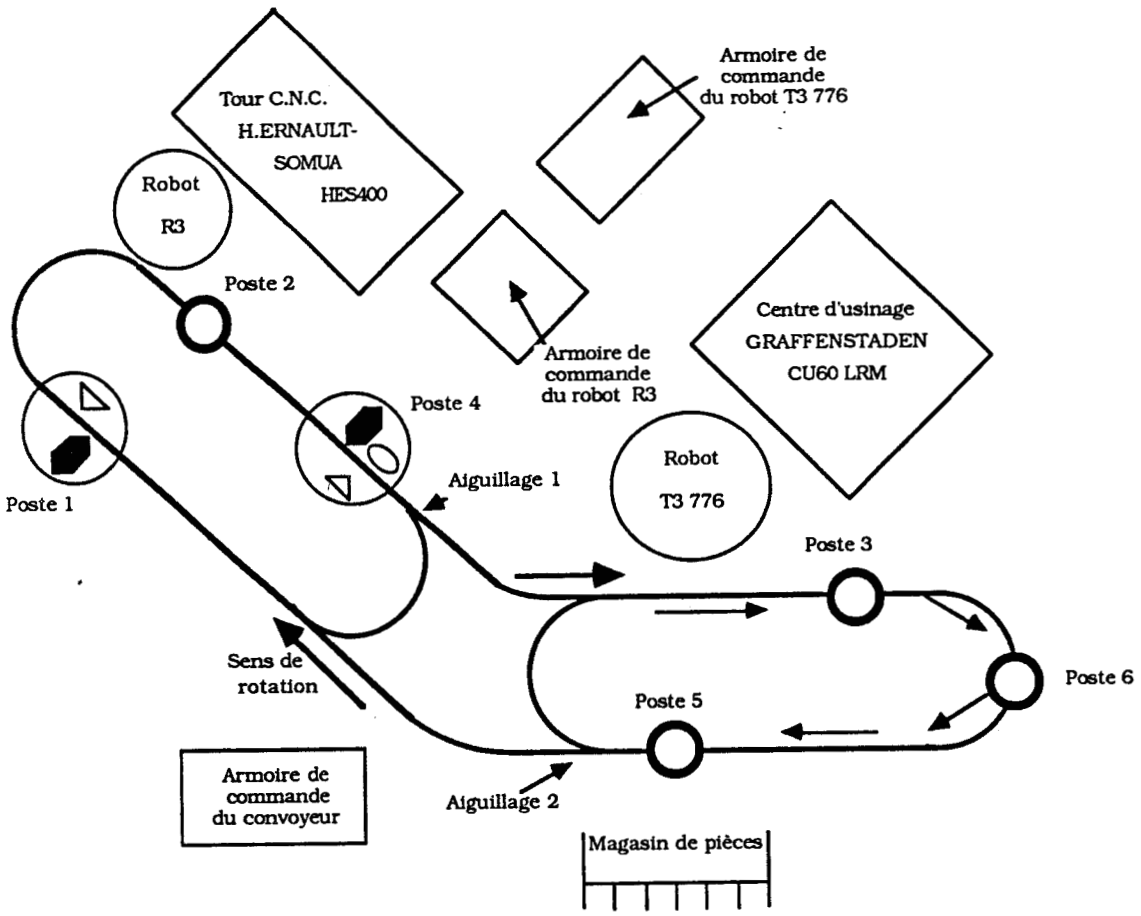
(ii) - 1 tour à commande numérique **T** (modèle HES400 de chez Hernault Somua) sans tampon d'entrée/sortie.

(iii) - 1 centre d'usinage à commande numérique **F** (modèle CU 60 LRM de chez Graffenstaden) et disposant de 2 tampons d'entrée/sortie : ES1 et ES2.

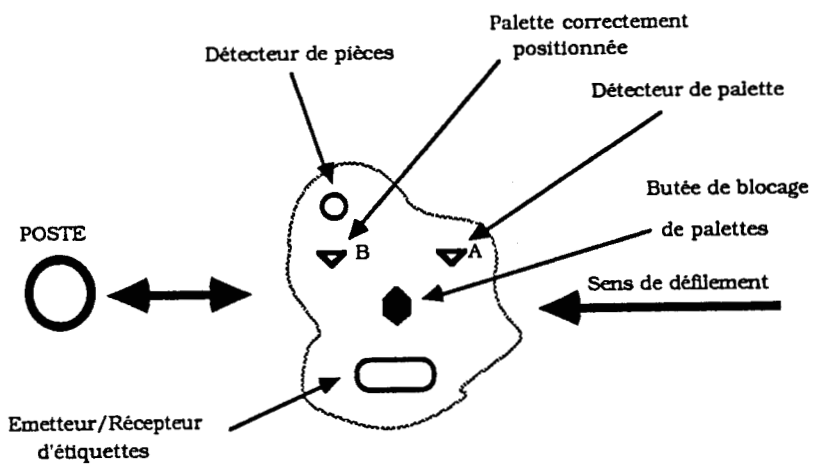
(iv) - 1 robot **R1** (AFMA de type R3 à 6 degrés de liberté) dédié exclusivement, de par sa localisation, à l'alimentation du tour.

(v) - 1 robot **R2** (T^3-776 de marce Cincinnati) qui alimente le centre d'usinage et assure les entrées/sorties des pièces.

(vi) - 1 convoyeur à chaine débrayable munie d'aiguillages et de butées de blocage qui assure les transferts des pièces palettisées entre les différents éléments de l'atelier.



Eléments pouvant entrer dans la composition d'un poste

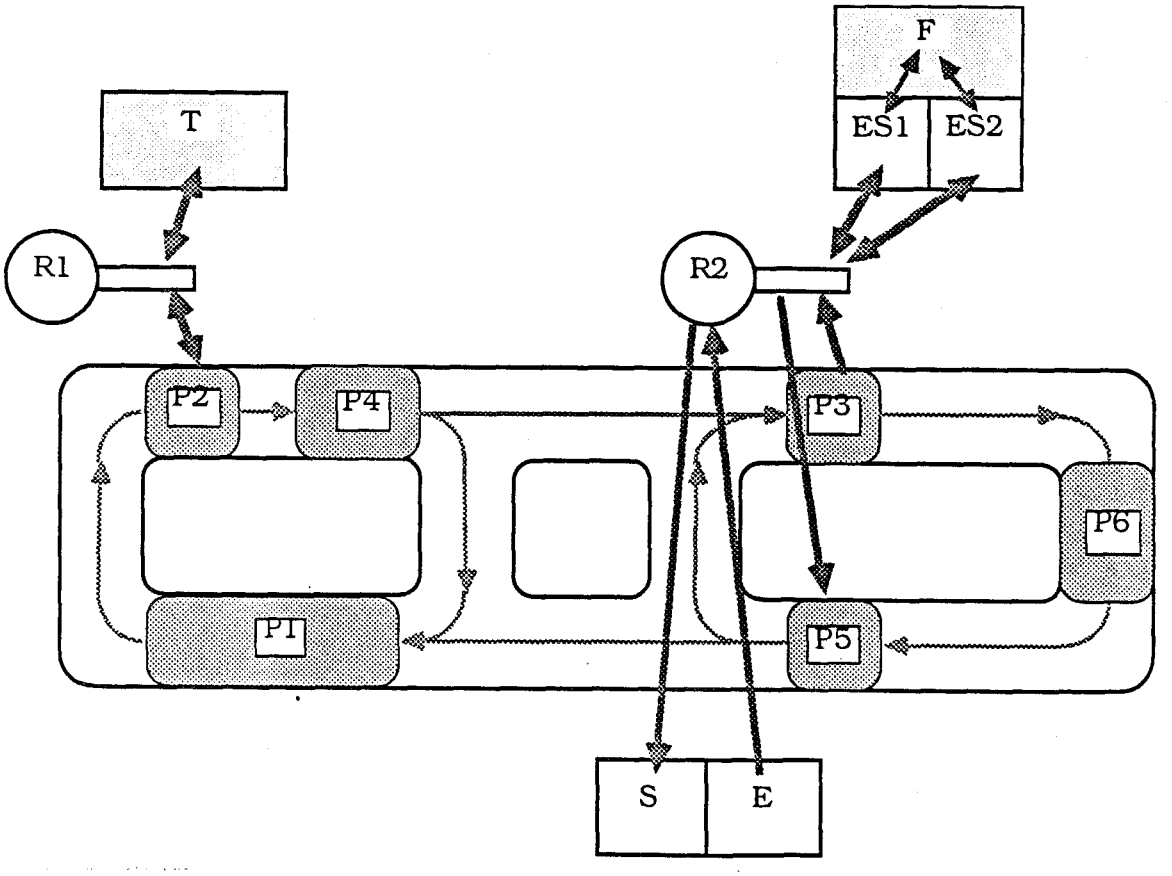


Implantation physique de la cellule

FIGURE 10

Six zones de stockage intermédiaires sont définies sur le convoyeur. Elles sont associées aux postes numérotés qui disposent de détecteurs, de positionneurs et de butées de blocage. Ces zones sont naturellement gérées en FIFO et leurs fonctions sont les suivantes :

- **P1** est une zone tampon de pièces destinées au tournage.
- **P2** est une zone de (dé)palettisation des pièces à tourner.
- **P3** est une zone de (dé)palettisation des pièces qui doivent, soit être usinées sur le centre, soit sortir de la cellule.
- **P4** est une zone tampon régulant la circulation des pièces dans la cellule.
- **P5** est une zone de palettisation des pièces brutes ou des pièces fraisées et dont la gamme opératoire n'est pas terminée (cf exemple ci-après d'une gamme Ft).
- **P6** est la zone de stockage des palettes vides.



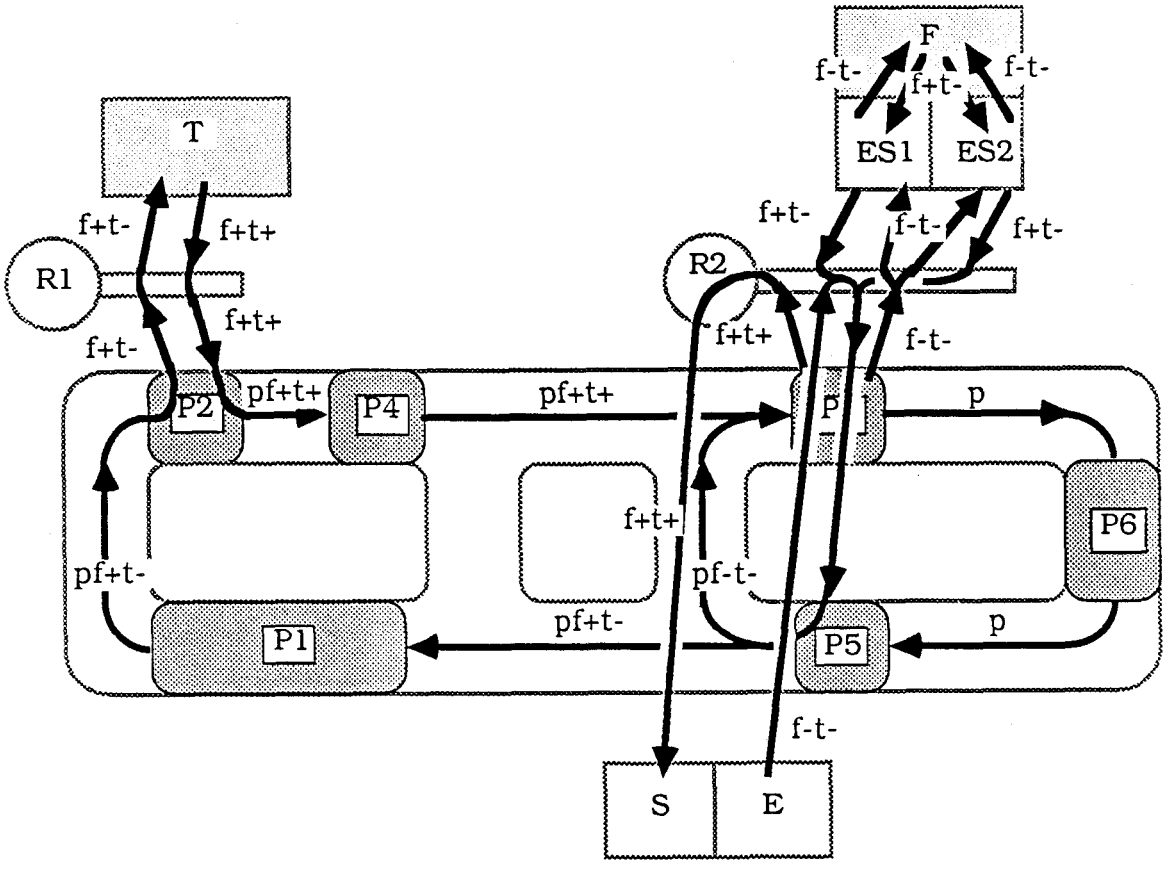
Description fonctionnelle de la cellule

FIGURE 11

On distingue deux parties dans le convoyeur. La première, plus particulièrement associée au tour comprend les zones tampons P1, P2 et P4 ; la seconde, liée au centre d'usinage et aux entrées/sorties regroupe les zones P5, P3, P6. Elles communiquent entre elles par deux aiguillages situés en aval de P4 et de P5.

L'entrée des pièces en attente sur la table de transfert E se fait par l'intermédiaire du robot R2 via le tampon P5. Réciproquement, la sortie est assurée depuis P3 vers S par le même robot. Les pièces alimentant le tour sont issues de P2 en provenance de la zone P1 tandis que l'entrée du centre d'usinage provient de la zone P3. Sur cette cellule, circulent 5 types de pièces nécessitant pour leur fabrication un traitement sur le tour et (ou) sur le centre d'usinage. Les gammes opératoires sont donc tournage ou fraisage exclusif, tournage puis tournage ou fraisage et enfin fraisage puis tournage. Pour chaque type de pièce, les notations retenues sont t pour le tour, f pour le centre d'usinage ; le suffixe - représente une opération non encore effectuée et le suffixe + une opération réalisée.

La figure 12 présente le détail de la réalisation de la gamme opératoire fraisage puis tournage (la couleur p traduit la présence d'une palette dans le déroulement des opérations).



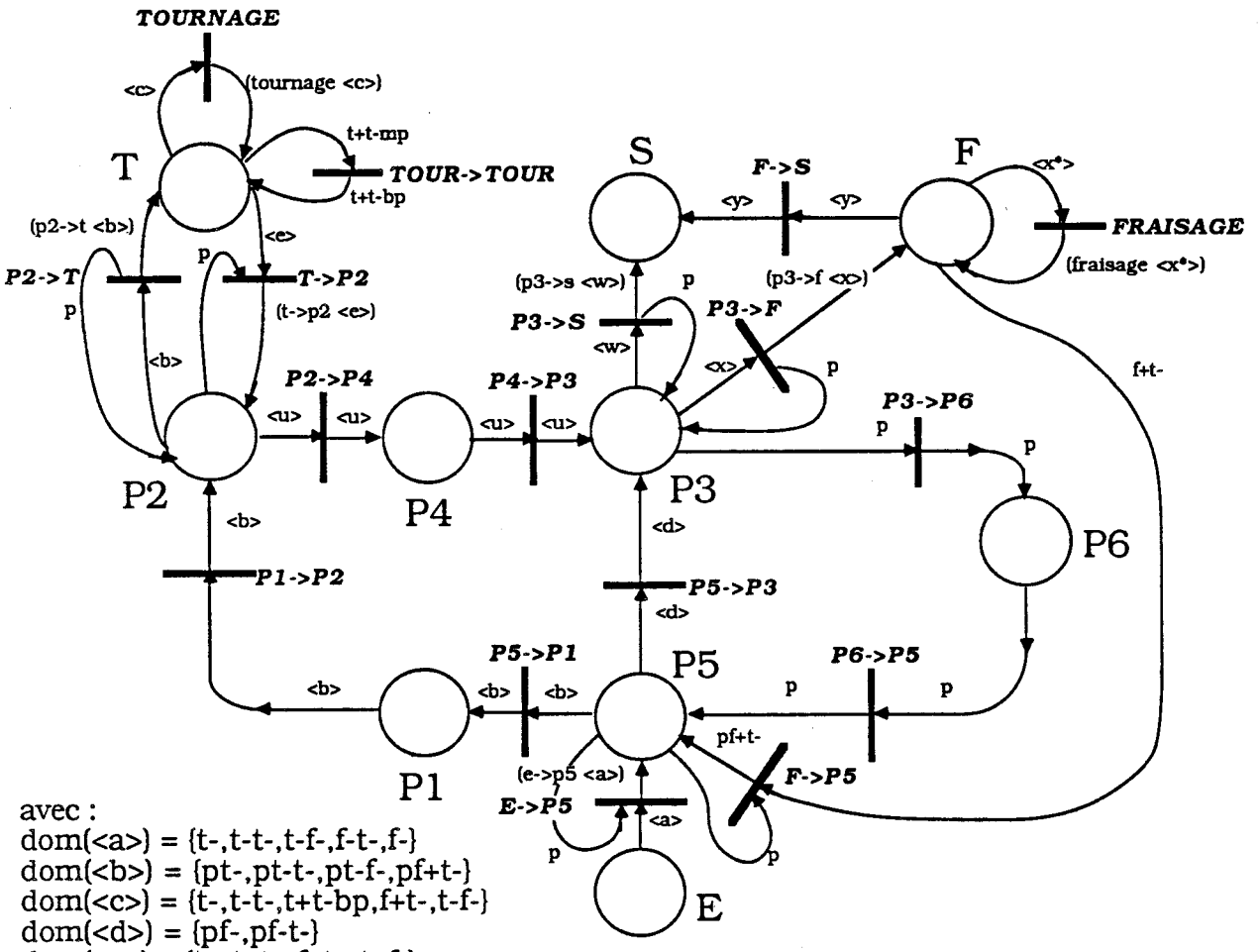
Détail de la gamme ft

FIGURE 12

II.3 - Application de la méthodologie

II.3.1 - Elaboration du prégraphe

La connaissance des gammes opératoires élémentaires ainsi que la détermination des lieux physiques minimaux nécessaires à leur réalisation permettent d'appréhender, à partir du cahier des charges, la réalisation du prégraphe (fig. 13). Ce dernier, après une analyse de cohérence et de complétude, valide formellement la description et sert de support pour le développement du graphe structuré.



avec :
dom(<a>) = {t-,t-t-,t-f-,f-t-,f-}
dom(<b>) = {pt-,pt-t-,pt-f-,pf+t-}
dom(<c>) = {t-,t-t-,t+t-bp,f+t-,t-f-}
dom(<d>) = {pf-,pf-t-}
dom(<e>) = {t+,t+t+,f+t+,t+f-}
dom(<u>) = {pt+,pt+t+,pt+f-,pf+t+}
dom(<w>) = {pt+,pt+t+,pf+t+}
dom(<x>) = {pf-,pf-t-,pt+f-}
dom(<x*>) = {f-,f-t-,t+f-}
dom(<y>) = {f+,t+f+}

Prégraphe de la cellule

FIGURE 13

Il est à signaler que la prise en compte des différents modes de marche ainsi que l'intégration de la notion de rebuts au cours d'une phase opératoire doivent être envisagées dès la conception du prégraphe. En effet, la définition d'un mode dégradé correspond à la création d'un ensemble de nouveaux chemins suivi par tout ou partie de la production. Il se traduit par l'apparition de nouveaux arcs reliant des lieux physiques non directement chainés en mode de marche normale ou par l'extension de domaines de couleurs. Si l'on suppose par exemple que le tour n'est plus opérationnel et qu'au niveau de la commande on souhaite pouvoir continuer l'opération de fraisage sans pour cela engorger jusqu'à saturation la cellule, il faut obligatoirement étendre la définition des domaines de coloration. Le domaine (<u>) doit ainsi inclure les couleurs de l'ensemble (<b>) ce qui permet de continuer à faire circuler une pièce non tournée jusqu'à la zone P3 où le domaine <(w)> subit la même modification pour laisser sortir les pièces dont la gamme a été incomplètement réalisée. Le concept des pièces rebuts relève de la même analyse : prendre en compte une pièce mal usinée revient à modéliser une couleur supplémentaire. Une opération de tournage défectueuse correspond donc à la création d'une couleur **t*** (* traduit l'opération mal réalisée) qu'il faudra véhiculer dans le prégraphe jusqu'à la sortie.

II.3.2. - Structuration

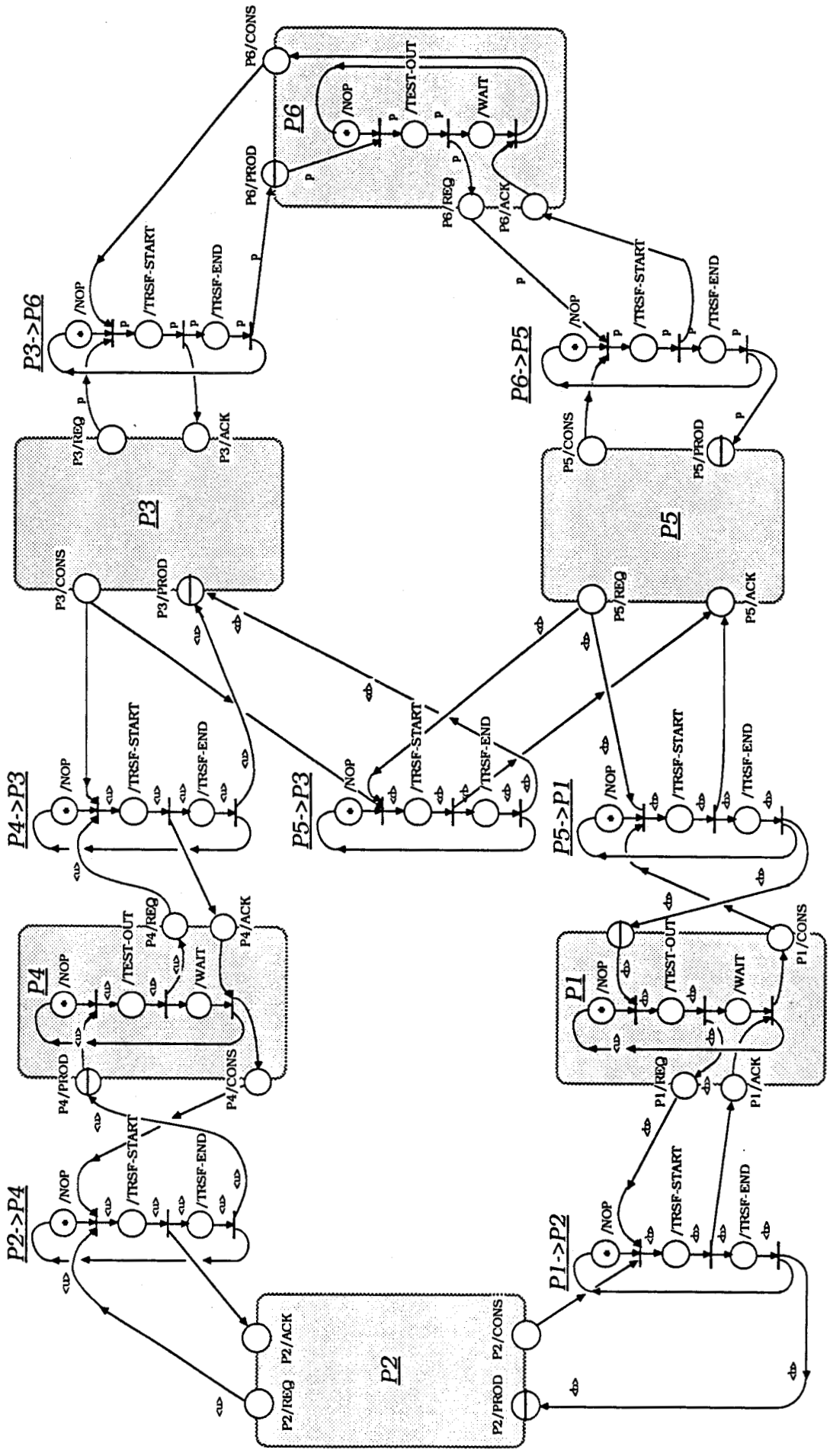
Cette étape consiste en un apport informationnel par une analyse fonctionnelle, associé à la définition des places et transitions du prégraphe. Les places, représentant principalement les lieux physiques, et les transitions, les actions effectuées sur les objets, sont développées en fonction de leurs spécificités propres. Ainsi, la place F modélisant le centre d'usinage est transformée pour aboutir à la commande traduisant les séquences des opérations élémentaires qui y sont réalisées. L'apport informationnel se situe dans la description physique du centre qui intègre à ce niveau la présence de deux tampons d'entrées/sorties ES1 et ES2. Similairement, les transitions sont développées fonction de l'opération élémentaire schématisée qui peut aussi bien être un transfert simple, une palettisation, un positionnement que tout autre opération. La structuration du modèle va ainsi dans le sens d'une augmentation de sa flexibilité. En effet, elle permet de reconfigurer automatiquement les organes de pilotage, action d'autant plus importante qu'un système de production n'est jamais figé mais sujet à des modifications en cours même d'exploitation.

Notre propos n'est pas ici de détailler la démarche complète de structuration. Nous allons donc nous contenter de donner à titre indicatif le développement en RdP Structurés de l'ensemble des processus assurant la gestion du convoyeur, de la place P3 chargé de l'alimentation du centre d'usinage et de la sortie des pièces de la cellule, et de la modélisation complète des processus caractérisant le fonctionnement du centre d'usinage. Cette dernière modélisation va nous servir de support pour présenter comment l'implantation est abordée et comment le niveau hiérarchique est utilisé aussi bien pour résoudre les problèmes de communication inter-automates que pour répondre à la mise en oeuvre de stratégies de fonctionnement complexes.

La figure 14 représente le développement en RdP structurés :

- des zones tampons P1, P4, P6,
- des différents transferts correspondant aux transitions du prégraphe P1—>P2, P2—>P4, P4—>P3, P3—>P6, P6—>P5, P5—>P1, P5—>P3.

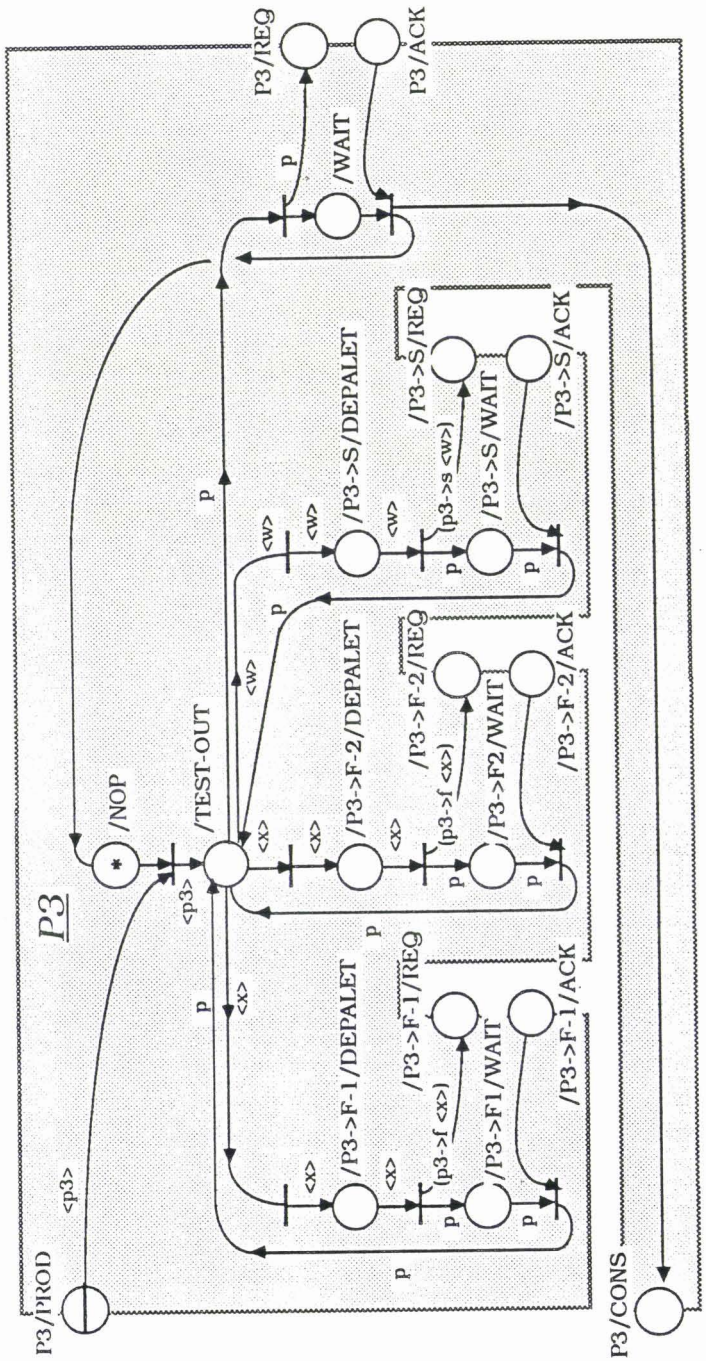
On remarque ici combien la communication entre les processus est banalisée et normalisée par l'utilisation alternative de producteur/consommateur et de requête avec accusé de réception. Cette standardisation va dans le sens d'une implantation automatisée où il est suffisant de simplement transcrire les processus en grafcet et de décrire les primitives de communication par une éventuelle utilisation du niveau hiérarchique.



Développement du convoyeur

FIGURE 14

La figure 15 représente le développement de la place P3 et des transitions de dépalettisation P3—>F-1, P3—>F-2, et P3—>S



dom(<p3>) = { pt+, pt+t+, pt+f-, pf+t+ , pf-, pf-t- }
dom(<x>) = { pt+f-, pf-, pf-t- }
dom(<w>} = { pt+, pt+t+, pf+t+ }

```
(de p3->f (x)
    (cond ((equal x 'pf-) 'f-)
          ((equal x 'pf-t-) 'f-t-)
          ((equal x 'pt+f-) 't+f-)))
```

```
(de p3->s (w)
    (cond ((equal w 'pt+) 't+)
          ((equal w 'pt+t+) 't+t+)
          ((equal w 'pf+t+) 'f+t+)))
```

Développement de P3

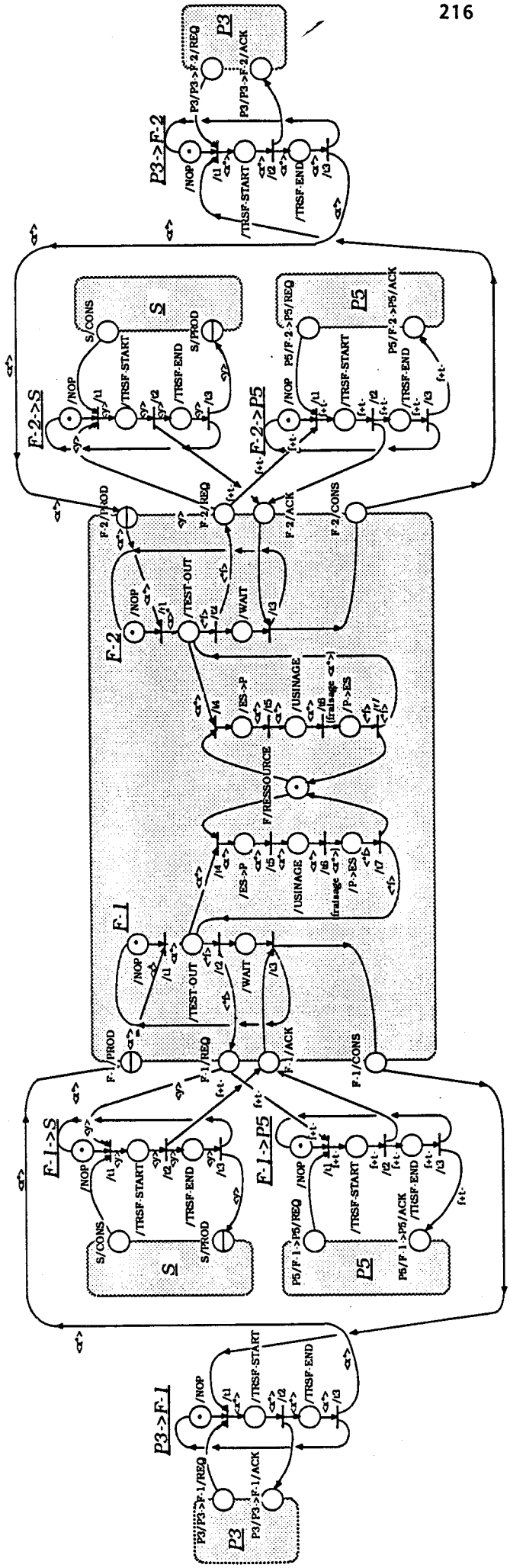
FIGURE 15

Ce développement a pour objet d'illustrer de quelle façon il est relié à l'ensemble des processus de fraisage qui seront principalement étudiés dans la suite de cette présentation. Ces derniers sont en fait connectés au processus P3 par l'intermédiaire des places P3/P3—>F-1/REQ (respectivement P3/P3—>F-2/REQ) qui correspondent à l'émission d'une demande de transfert donc d'usinage vers la commande du fraisage ; en retour, l'accusé de réception transite vers P3 via les places P3/P3—>F-1/ACK (resp. P3/P3—>F-2/ACK).

La figure 16 illustre la réalisation de la commande des processus de chargement et déchargement des tampons d'entrée/sortie ES1 et ES2 ainsi que celle de l'opération d'usinage nécessitant l'attribution d'une ressource exclusive. Le logiciel de structuration, prenant en compte les contraintes imposées par le cahier des charges, donne automatiquement la liste exhaustive des conflits générés par toutes les liaisons de type exclusion mutuelle. Afin de ne pas surcharger la figure, la ressource critique R2 n'est pas représentée. Néanmoins, elle concerne les 8 processus de transfert suivants :

E—>P5, F-1—>P5, F-2—>P5, P3—>S, P3—>F-1, P3—>F-2, F-1—>S, F-2—>S

Le résultat final de la structuration du prégraphe donne un graphe global composé de 215 places et 126 transitions formant 31 processus et leurs interconnexions. A ce stade de la conception, et avant d'envisager l'implantation définitive, il est nécessaire d'effectuer plusieurs simulations afin d'assurer que le comportement dynamique du graphe développé respecte les contraintes spécifiées dans le cahier des charges.



dom(<x*>) = { f-, f-t-,t+f- }
dom(<*f>) = { f+, f+t-, t+f+ }
dom(<y>) = { f+, t+f+ }

```
(de   fraisage (x)
      (cond  ((equal  x  'f-)  'f+)
             ((equal  x  'f-t-)  'f+t-)
             ((equal  x  't+f-)  't+f+)))
```

Le centre d'usinage

FIGURE 16

II.3.3 - Simulation

L'objectif de la simulation est double. Elle permet en premier lieu de valider une dernière fois les différentes gammes opératoires du système de production et de donner un premier ordre de grandeur des performances du système. D'autre part, en fonction principalement des algorithmes retenus pour l'introduction des pièces, elle fournit des résultats quantitatifs sur la productivité générale de la cellule.

Toute la difficulté liée à la simulation réside dans le nombre et le très fort degré de connexité des différents paramètres à dimensionner. Si les temporisations liées aux actions élémentaires sont imposées la plupart du temps et estimées ou relevées par chronométrage sur le site réel, le dimensionnement des zones tampons est par contre laissé à la discrétion de l'utilisateur (dans la limite d'une fourchette admissible). De façon analogue, la non-autonomie du modèle RdPSAC concernant en particulier les conflits et les indéterminismes directionnels impose la définition de règles évaluées par le simulateur et qui permettent de lever toute situation bloquante. Chacune de ces règles est un paramètre à part entière qui conditionne par sa définition, bonne ou mauvaise, les performances du système. Enfin, toute interprétation significative est étroitement liée aux séquences retenues pour l'introduction des pièces dans le système. Dans le cas présent, trois politiques complémentaires ont été choisies :

(i) - Simulation temporisée et introduction sélective d'une pièce à la fois.

(ii) - Simulation non temporisée et choix d'une politique d'entrée aléatoire et au plus tôt.

(iii) - Simulation temporisée et choix d'une politique d'entrée au plus tôt.

La première méthode met en avant la validité de la description des gammes opératoires et donne simultanément le temps minimal de conditionnement d'une pièce dans le système (en effet, la pièce étant seule, elle a donc accès immédiatement à toutes les ressources). Le deuxième point autorise une validation du comportement général du système par une analyse du fonctionnement des stratégies externes au modèle RdPSAC. Une simulation temporisée sélectivement (à titre d'exemple : les processus de transfert seulement ou encore uniquement

les opérations d'usinage) permettrait quant à elle de mettre en évidence les limites de la productivité en fonction de certains éléments du procédé. L'ajout d'une machine supplémentaire se révèlerait alors inutile si le principal goulot d'étranglement se situe en fait au niveau du convoyage. Le troisième et dernier point a pour but de déterminer, en fonctionnement normal, l'influence de la séquence d'entrée des pièces sur le système et de détecter d'éventuels cas de blocage (dead lock) en régime non autonome. En cas d'étreintes fatales et après analyse, plusieurs solutions s'offrent à l'utilisateur. La première et la plus dramatique revient à constater l'échec de la modélisation par son manque de flexibilité ; elle remet donc en cause tout le travail de conception qui doit être revu en intégrant par exemple un nouvel organe de transport. En second lieu, l'utilisateur peut, s'il arrive réellement à cerner et interpréter la nature du blocage, jouer sur la définition des stratégies d'allocation externes au modèle RdPSAC. Enfin, il peut simplement surveiller ses séquences d'entrée et interdir une suite bloquante spécifique en différant l'introduction de certaines pièces dont les gammes sont incompatibles.

Pour cet exemple, la simulation a permis d'établir la conformité de la commande par rapport aux spécifications du cahier des charges. Les différentes gammes sont effectuées en respectant les exigences imposées de productivité. De plus, deux situations bloquantes ont été mises en évidence lors de la simulation. La première provient de l'absence de palette vide alors que les pièces présentes sur les tampons du centre d'usinage sont en attente de palettes vides sur la zone P5. La solution consiste ici simplement à n'introduire en permanence dans le système que n-1 pièces si n est le nombre de palettes initialement disponibles. Le second blocage concerne 4 pièces qui forment un dead lock au niveau des zones P3 et P5. Néanmoins, toutes les informations nécessaires sont présentes dans la commande pour y remédier par la définition d'une stratégie de fonctionnement adaptée et réalisée au niveau hiérarchique.

II.4 - Implantation

II.4.1 - Introduction

L'implantation procède selon deux étapes. Tout d'abord, le graphe de commande initial décrit en RdPSAC est repris pour être traduit en grafcet. Cette transposition est systématique (cf chapitre 1) : l'ossature des places et transitions

est simplement respectée hormis toutes les liaisons de type ressource, producteur/ consommateur et requête avec accusé de réception. Cette première étape se termine par la programmation en grafcet de certains traitements essentiellement locaux à un graphe donné. La transformation d'une couleur après une opération d'usinage est par exemple directement intégrée dans une place d'un processus. En second lieu, une étude approfondie des primitives de communication prenant en compte les critères de répartition des processus sur les automates permet de définir les solutions à retenir concernant leur mise en oeuvre. Une communication entre tâches au sein d'un même automate fait l'objet d'une transposition simple en grafcet tandis qu'un échange inter-automates nécessite une modélisation intégrée au niveau hiérarchique. Parallèlement à ces définitions, les heuristiques de contrôle mises au point lors de la simulation sont reprises afin de pouvoir être utilisées par le niveau hiérarchique avec un interfaçage vers les modèles grafcet. Cette seconde étape consistue donc la réalisation concrète du niveau hiérarchique interfacé avec la commande et fonction de l'implantation retenue.

II.4.2 - Répartition des graphes de commande

Le matériel disponible pour l'implantation des graphes de commande se compose de trois automates TSX67 de la Télémécanique. Les processus ont donc été répartis sur ces trois unités en tenant compte de plusieurs critères. Nous avons en particulier essayé d'équilibrer la charge de l'ensemble de façon équitable entre les trois automates. En second lieu, nous avons évité de dissocier des processus dont les fonctionnalités correspondent à la commande d'une seule entité physique. Enfin et de façon complémentaire au point précédent, nous avons regroupé le plus possible les processus à fort degré de connexité afin de minimiser les communications et échanges inter-automates. En fonction de ces impératifs, la répartition des processus a été menée de la façon suivante :

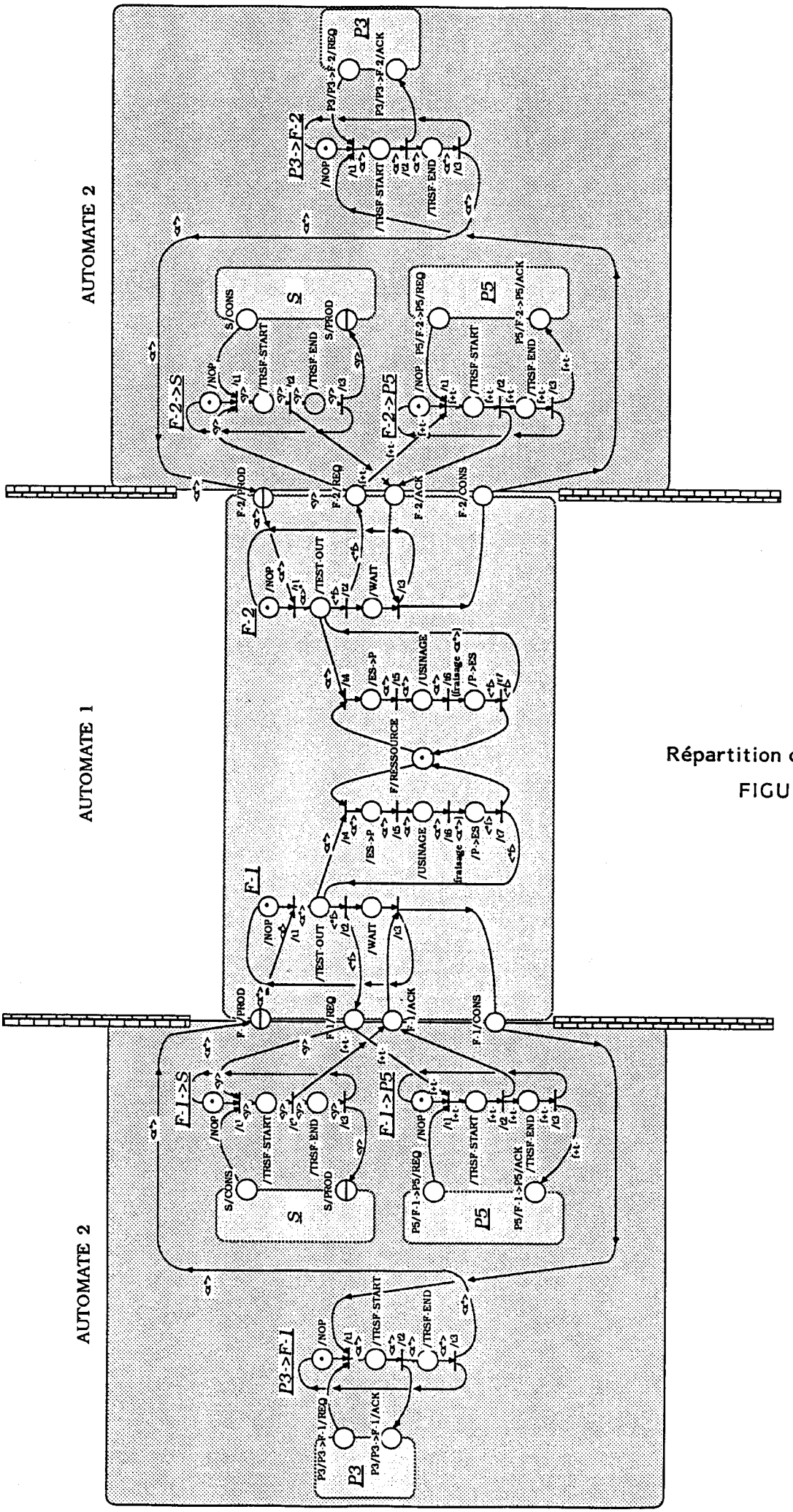
(i) - Sur l'automate 1 sont implantés la commande du tour et son processus associé P2 (zone de chargement/déchargement du tour). Les deuxième et troisième critères sont de la sorte respectés. L'automate 1, peu chargé, supporte également la modélisation de la commande du centre d'usinage.

(ii) - Sur l'automate 2 sont implantés les processus de chargement/ déchargement vers le centre d'usinage. Dans le cas présent, cette répartition (automate 1 : centre d'usinage, automate 2 : processus de

transfert) va à l'encontre des points 2 et 3 mais a été rendue nécessaire pour équilibrer la charge des différents automates. Elle va de plus permettre de mettre en oeuvre le niveau hiérarchique qui sera chargé de faire communiquer ces processus. Enfin, cet automate se charge des places P3 et P5 (zones de palettisation et dépalettisation) relativement autonomes.

(iii) - Sur l'automate 3 résident la zone de stockage P6 des palettes vides et l'ensemble des processus de transfert gérant le convoyeur hormis ceux précédemment cités. Le regroupement de la commande du convoyeur va tout à fait dans le sens d'une association fonctionnelle visant à limiter les communications.

L'implantation respecte ainsi les critères de répartition excepté pour le centre d'usinage et ses processus de transfert (fig. 17). La charge de l'ensemble est relativement équilibrée au détriment du regroupement fonctionnel d'une entité physique : c'est le compromis que nous avons retenu pour la réalisation de notre exemple.



Répartition des processus

FIGURE 17

II.4.3 - Progammation des grafcets et du niveau hiérarchique

Nous n'allons pas ici détailler la programmation de tous les graphes implantés sur les automates. Cette tâche serait fastidieuse et n'apporterait rien à la compréhension de l'exposé. Notre exemple de mise en oeuvre va principalement s'appuyer sur la réalisation effective des processus F-1 et F-2 du centre d'usinage avec utilisation du niveau hiérarchique pour banaliser les communications inter-automates. Nous montrerons dans un second temps comment la programmation du robot Cincinnati R2 utilise des heuristiques définies au niveau hiérarchique, donc indépendantes des grafcets, et modifiables à tout moment en cours d'exploitation.

Avant de présenter les processus F-1 et F-2, il nous faut introduire une extension définie par la Télémécanique et concernant les modes de fonctionnement d'un Grafcet. Le concept d'étape active a été étendu sur ce matériel. En fait, la programmation autorise trois types d'action :

(i) - Les actions à l'activation. Elles ne sont exécutées qu'une seule fois lorsque l'étape à laquelle elles sont associées est activée.

(ii) - les actions continues. Ces actions sont exécutées tant que l'étape à laquelle elles sont associées est active. Elles correspondent donc au fonctionnement normalisé du Grafcet.

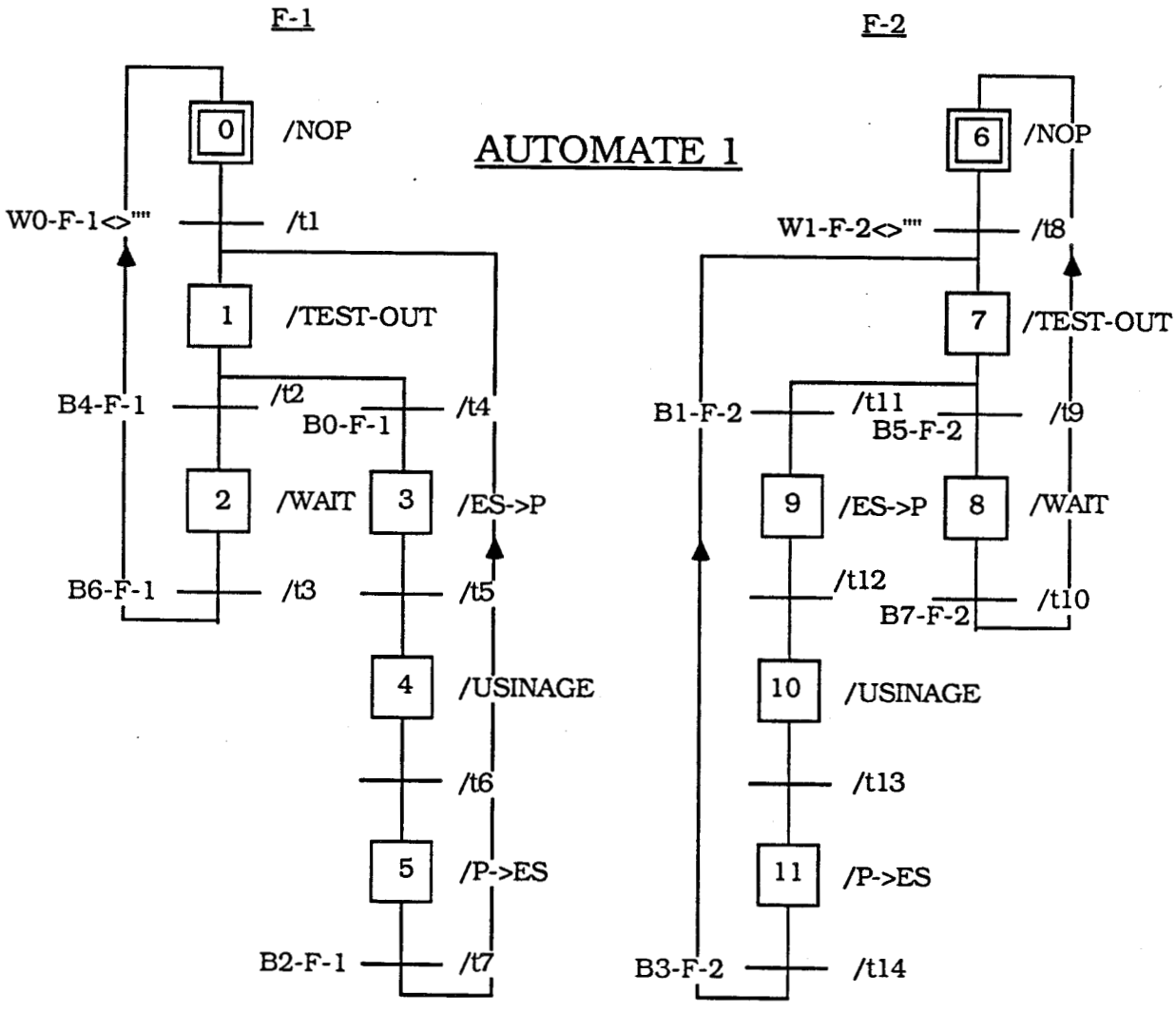
(iii) - Les actions à la désactivation. Elles sont le pendant des actions à l'activation et ne sont exécutées qu'une seule fois lorsque l'étape à laquelle elles sont associées passe de l'état actif à l'état repos.

Chaque fois qu'il sera nécessaire, nous spécifierons dans nos exemples la nature des actions programmées.

Exemple 1 : PROGRAMMATION DU PROCESSUS F-1

L'existence du graphe RdPSAC constitue un apport essentiel pour la réalisation de la programmation. Il intègre tout le séquencement élémentaire des actions nécessaires au fonctionnement du système et définit le cheminement du marquage en fonction de la coloration. Son ossature doit être reprise telle quelle pour la réalisation des grafcets. La figure 18 montre ainsi comment la

transposition des deux processus F-1 et F-2 reprend exactement le même nombre de places et de transitions en conservant un squelette identique. Nous avons utilisé pour des raisons de clarté les mêmes noms associés aux places et transitions et reporté sur certaines transitions des réceptivités conditionnées soit par des mots (W0-F-1<>"", W1-F-2<>""), soit par des bits (B0-F-1, B2-F-1,...) dont la signification sera détaillée par la suite.



Grafcets des processus d'usinage

FIGURE 18

Nous allons maintenant détailler le mécanisme général de fonctionnement du grafcet F-1.

* Place F-1/NOP :

Cette place est marquée à l'initialisation, elle ne contient aucune couleur et traduit simplement la disponibilité du processus F-1. Par hypothèse, la couleur utilisée par ce processus sera véhiculée dans le mot W0-F-1. On s'assure donc que ce mot est initialement vide par l'action suivante :

- action sur activation : SET W0-F-1<—"".

* Transition F-1/t1 : W0-F-1<>""

La réceptivité de cette transition est initialement fausse. Cela est assuré par l'action sur activation de la place F-1/NOP. Pour déclencher cette transition, le modèle RdPSAC indique qu'il faut une marque colorée dans le producteur/consommateur F-1/PROD. Comme ce dernier assure une liaison inter-automate imposée par l'implantation, nous décidons d'en effectuer la modélisation au niveau hiérarchique par l'intermédiaire d'un fait, qui est en réalité une structure de données, de nom F-1/PROD. La valeur du mot W0-F-1 changera ainsi suite au déclenchement de la règle R1 :

R1 : (égal W0-F-1 "") (1) (différent F-1/PROD vide) (2)
alors (get-fifo F-1/PROD W0-F-1) (3)

(1) caractérise indirectement le marquage de la place F-1/NOP
(2) caractérise la présence d'au moins une marque colorée dans F-1/PROD
(3) déclenche le tir de la transition F-1/t1 en transmettant la marque colorée dans la fifo F-1/PROD au mot W0-F-1.

* Place F-1/TEST-OUT :

Le marquage de cette place et sa couleur résoud l'indéterminisme directionnel sur t2 et t4. En effet les transitions F-1/t2 et F-1/t4 sont exclusives car leurs domaines de couleurs respectifs sur les arcs amonts sont disjoints. Nous allons tout d'abord étudier le mécanisme de déclenchement de la transition F-1/t4 ; nous reviendrons ultérieurement sur celui de la transition f-1/t2.

a) Premier mécanisme de déclenchement de la transition F-1/t4 avec modélisation interne à l'automate 1 de la ressource F/RESSOURCE.

Les conditions nécessaires et suffisantes pour le déclenchement de cette transition sont :

- place F-1/TEST-OUT marquée avec une couleur dans le domaine (<x*>).
- disponibilité de la ressource F/RESSOURCE caractérisant l'unicité de la zone de travail du centre d'usinage.

On gèle en premier lieu la transition /t4 d'où

place F-1/TEST-OUT, action sur activation : SET B0-F-1<—0

On détermine ensuite les conditions pouvant dégeler la transition

place F-1/TEST-OUT, action continue :

SI (|W0-F-1 = "f-"| ou |W0-F-1 = "f-t-"| ou |W0-F-1 = "t+f-"|)
et (F/RESSOURCE = 1)| alors SET F/RESSOURCE<— 0
SET B0-F-1<— 1

Ceci s'interprête comme suit : si la couleur correspond au domaine de validité et si la ressource est libre alors on prend la ressource et on autorise le franchissement de la transition. L'exclusivité de la prise de la ressource est garantie par le fonctionnement du scheduler de l'automate. En effet, l'évaluation d'une action est indivisible et ininterruptible.

b) Deuxième mécanisme de déclenchement de la transition F-1/t4 avec modélisation externe de la ressource.

On décide cette fois d'attribuer la ressource en cas de conflit entre les processus F-1 et F-2 à l'opération d'usinage la moins longue. Cette durée est liée à la couleur de la pièce présente dans la place F-1/TEST-OUT ou F-2/TEST-OUT. On utilise dans ce cas le niveau hiérarchique car il est par hypothèse le seul à connaître la correspondance entre la couleur et le temps d'usinage. L'action sur activation est identique à celle décrite au point a). Par contre, la transition est débloquée par l'évaluation des règles du niveau hiérarchique suivantes :

```
R3 : (DE demande-ressource-f-1 ()
        (COND
        ((AND
              (egal f/ressource libre)
              (egal f-1/test-out marque)
              (OR  (egal w0-f-1  "f-")
                   (egal w0-f-1  "f-t-")
                   (egal w0-f-1  "t+f-")))
              (faits-int '(affecte demande-ressource-f-1  vrai)) t)
        (t  ())))

R4 : (DE demande-ressource-f-2  ()
     (COND
          ((AND
                (egal  f/ressource libre)
                (egal  f-2/test-out marque)
                (OR  (egal w1-f-2  "f-")
                     (egal w1-f-2  "f-t-")
                     (egal w1-f-2  "t+f-")
                (faits-int  '(affecte demande-ressource-f-2  vrai))  t)
          (t  ()))))

R5 : (DE attribution-f/ressource  ()
          (COND
               ((AND
                     (egal  f/ressource libre)
                     (egal  demande-ressource-f-1 vrai)
                     (egal  demande-ressource-f-2 vrai))
                     (faits-int   '(affecte f/ressource occupe))
                     (IF    (<=   (tpfrai  'w0-f-1)  (tpfrai  'w1-f-2))
                            (progn
                                  (faits-int
                                        '(affecte demande-ressource-f-1  faux))
                                  (faits-ext  '(affecte b0-f-1  1)))
                            (faits-int  '(affecte demande-ressource-f-2  faux))
                            (faits-ext  '(affecte b1-f-2  1)))  t)
```

```
((AND
      (egal f/ressource libre)
      (egal demande-ressource-f-1  vrai))
      (faits-int  '(affecte f/ressource occupe))
      (faits-int  '(affecte demande-ressource-f-1  faux))
      (faits-ext  '(affecte b0-f-1  1))  t)
((AND
      (egal f/ressource libre)
      (egal demande-ressource-f-2  vrai))
      (faits-int  '(affecte f/ressource occupe))
      (faits-int  '(affecte demande-ressource-f-2  faux))
      (faits-ext  '(affecte b1-f-2  1))  t)
(t  ())))

(DE  tpfrai  (w)
     (COND
          ((egal  w  "f-")  temps1)
          ((egal  w  "f-t-")  temps2)
          ((egal  w  "t+f-"  temps3)))
```

Le principe de résolution est simple : les règles R3 et R4 se contentent de vérifier si les processus F-1 ou F-2 sont en attente de la ressource tandis que la règle R5 attribue la ressource si au moins un processus la demande.

* Transition F-1/t4 :

Le bit B0-F-1 conditionne la réceptivité de cette transition. Sa valeur est définie en interne (hypothèse a)) ou par le niveau hiérarchique (hypothèse b)).

* Place F-1/P—>ES :

Après l'opération d'usinage, nous traduisons le changement de nature de la pièce dans sa zone activation :

action sur activation :

Si |W0-F-1 = "f^-"| alors SET W0-F-1<— "f^+"

Si |W0-F-1 = "f^-t^-"| alors SET W0-F-1<— "f^+t^-"

Si |W0-F-1 = "t^+f^-"| alors SET W0-F-1<— "t^+f^+"

Ces conditions correspondent à la traduction de la fonction fraisage (x) donnée figure 16.

Si la ressource F/RESSOURCE est gérée exclusivement par les automates, il suffit alors de la rendre de nouveau disponible à ce niveau.

action sur désactivation : SET F/RESSOURCE<— 1

Par contre, si la ressource est gérée par le niveau hiérarchique, il est nécessaire de geler la transition F1/t7 afin de s'assurer que ce dernier puisse prendre en compte la sortie de la section critique. Cela est réalisé par le bit B2-F-1 et la règle R6

action sur activation : SET B2-F-1<— 0

R6 :
```
(DE restitution-f/ressource ()
    (IF (egal f/ressource occupe)
        (COND
            ((egal f-1/p—>es marque)
                (faits-int '(affecte f/ressource libre))
                (faits-ext '(affecte b2-f-1 1)) t)
            ((egal f-2/p—>es marque)
                (faits-int '(affecte f/ressource libre))
                (faits-ext '(affecte b3-f-2 1)) t)
            (t t)) ()))
```

Nous avons ici utilisé une forme LISP pour restituer la ressource. Nous aurions tout aussi bien pu écrire deux règles de production sous la forme suivante :

R6 bis : (egal F/RESSOURCE occupe) (egal F-1/p—> ES marque)
alors (affecte F/RESSOURCE libre)
(affecte B2-F-1 1)

R6 ter : (egal F/RESSOURCE occupe) (egal F-2/P—>ES marque)
alors (affecte F/RESSOURCE libre)
(affecte B3-F-2 1)

* Transition F-1/t7 :

gestion interne de la ressource : réceptivité = 1

gestion externe de la ressource : réceptivité = B2-F-1

A ce stade, nous avons terminé la programmation de la gestion de l'usinage. La marque se retrouve dans la place F-1/TEST-OUT avec une nouvelle couleur et va nécessiter l'émission d'une requête pour être prise en compte par un des processus de transfert.

* Place F-1/TEST-OUT :

Nous complétons la gestion de cette place pour assurer le bon fonctionnement de la requête inter-automate F-1/REQ. Nous nous contentons èn fait de geler la transition F-1/t2.

Action sur activation : SET B4-F-1<— 0

* Transition F-1/t2 :

Pour valider cette transition, il faut que la place F-1/TEST-OUT comporte la bonne couleur.

De plus son franchissement, conjointement lié à l'émission d'une requête, doit faire transiter la couleur courante du processus F-1 vers le processus F-1—>S ou F-1—>P5. Ce mécanisme inter-automates nécessite l'utilisation d'une nouvelle règle R7 ainsi que la définition d'une variable interne au niveau hiérarchique qui modélisera la place F-1/REQ

```
R7 : (DE émission-f-1/req ()
          (COND
               ((AND
                    (egal f-1/test-out  marque)
                    (egal b4-f-1 0)
                    (OR  (egal w0-f-1  "f+")
                         (egal w0-f-1  "f+t+")
                         (egal w0-f-1  "t+f+")))
                    (faits-int  '(affecte f-1/req w0-f-1))
                    (faits-ext  '(affecte b4-f-1  1))  t)
               (t  t)))
```

Réceptivité de la transition f-1/t2 = B4-F-1

* Place F-1/WAIT :

Cette place se contente de geler la transition F-1/t3 dans l'attente d'un accusé de réception émis par des processus externes à l'automate 1.

action sur activation : SET B6-F-1<— 0

* Transition F-1/t3 :

Cette transition nécessite la présence d'un accusé de réception dans la place F-1/ACK ; son franchissement doit de plus ajouter une marque dans la variable F-1/CONS. Comme tous ces mécanismes concernent des communications inter-automates, nous définissons simplement une nouvelle règle de production.

R9 : (egal F-1/ACK 1) (egal F-1/WAIT marqué) (egal B6-F-1 0)
alors (affecte F-1/ACK 0) (ajoute F-1/CONS 1) (affecte B6-F-1 1)

A ce stade, la programmation des processus d'usinage F-1 et F-2 se trouve réalisée sur l'automate 1 (nous n'avons pas complètement détaillé le processus F-2 car il est complètement symétrique avec le processus F-1). Les interfaces de communication avec les autres processus implantés sur d'autres automates sont complètement banalisées et prises en charge par le niveau hiérarchique. En annexe est présentée une description complète des processus F-1 et F-2 ainsi que la gestion du fonctionnement assurée par le niveau hiérarchique sous la forme textuelle éditée par notre logiciel.

Exemple 2 : GESTION DU ROBOT R2

Dans ce deuxième exemple, nous proposons de décrire la solution retenue pour la gestion du robot CINCINNATI R2. Celui-ci assure les entrées/sorties des pièces ainsi que le chargement/déchargement du centre d'usinage F. A ce titre, il constitue une ressource critique susceptible d'être attribuée à 8 processus différents : gestion du tampon d'entrée/sortie ES1 (P3—>F-1, F-1—>S, F-1—>P5), gestion du tampon d'entrée/sortie ES2 (P3—>F-2, F-2—>S, F-2—>P5), gestion de la zone P3 (P3—>S), gestion de l'entrée (E—>P5). Chaque processus n'est pas susceptible d'entrer en conflit avec les 7 autres. En effet, des invariants de marquage protègent et interdisent certains conflits. Par exemple, un seul des deux processus F-1—>S et F-1—>P5 peut être actif à un instant donné car le contenu de la place F-1/REQ détermine quel processus doit consommer la requête.

Ce faisant, il s'octroie la ressource qu'il ne rendra qu'après l'opération de transfert terminée. La méthode d'attribution de la ressource conserve sensiblement la même technique que celle utilisée pour la gestion de la zone d'usinage du centre. En premier lieu, un ensemble de règles détermine constamment les processus dont le marquage courant est gelé et n'attend plus que la ressource pour continuer à évoluer. En second lieu, une règle se chargera d'attribuer cette ressource en fonction des critères de priorité retenus par l'utilisateur. Cette méthode a pour avantage de dissocier nettement la détermination des conflits de leur résolution. La détermination est immuable. Par contre la résolution peut être modifiée a posteriori en fonction de nouveaux critères stratégiques. Seule une règle spécifique est alors concernée et pourra être revue.

La stratégie retenue vise dans le cas présent à privilégier systématiquement les pièces dont la gamme opératoire est la plus avancée. On évite ainsi un engorgement de la cellule en faisant d'abord sortir les pièces terminées, puis circuler les pièces dont la gamme est déjà avancée et enfin en introduisant en dernier les pièces brutes. L'ordre de priorités décroissantes est donc le suivant :

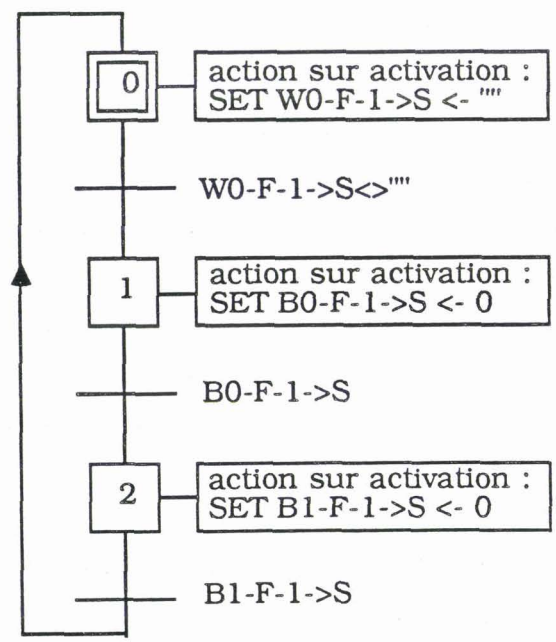
P3—>S	R11 : demande-R2-P3—>S
F-1—>S	R12 : demande-R2-F-1—>S
F-2—>S	R13 : demande-R2-F-2—>S
F-1—>P5	R14 : demande-R2-F-1—>P5
F-2—>P5	R15 : demande-R2-F-2—>P5
P3—>F-1	R16 : demande-R2-P3—>F-1
P3—>F-2	R17 : demande-R2-P3—>F-2
E—>P5	R18 : demande-R2-E—>P5

Nous ne détaillerons pas ici le contenu de toutes les règles, mais simplement celles qui concernent le processus F-1—>S dont le grafcet d'implantation est donné figure 19. La règle R12 qui détermine si le processus F-1—>S nécessite l'utilisation de la ressource est la suivante :

```
R12 : (de demande-r2-f-1—>s ()
          (cond
               ((and (egal r2 libre)
                     (different s/cons vide)
                     (egal f-1—>s/nop marque)
                     (or   (egal f-1/req "f+")
                           (egal f-1/req "t+f+")))
                     (faits-int '(affecte demande-r2-f-1—>s vrai)) t)
               (t ())))
```

La règle R19 se contente d'attribuer la ressource en fonction de la priorité des processus demandeurs. Elle utilise simplement le COND lisp dont la structure correspond à un IF en cascade :

```
R19 : (de attribution-r2 ()
        (if   (egal r2 libre)
              (cond
                   ((egal demande-r2-p3—>s vrai)
                        (faits-int '(affecte r2 "p3—>s"))
                        (faits-int '(affecte demande-r2-p3—>s faux)) t)
                   ((egal demande-r2-f-1—> vrai)
                        (faits-int '(affecte r2 "f-1—>s"))
                        (faits-int '(affecte demande-r2-f-1—>s faux)) t)
                                .
                                .
                                .
                   ((egal demande-r2-e—>p5 vrai)
                        (faits-int '(affecte r2 "e—>p5"))
                        (faits-int '(affecte demande-r2-e—>P5 faux)) t)
                   (t ())) ()))
```



Grafcet du processus F-1—>S

FIGURE 19

A ce stade, la ressource est virtuellement allouée à un processus. Il reste donc à définir les règles qui permettent de dégeler effectivement la transition amont de la section critique. Cela donne pour le processus F-1—>S :

R20 : (egal R2 "F-1—>S") (egal W0-F-1 "")
alors (ajoute S/CONS -1) (affecte W0-F-1—>S F-1/REQ)

R21 : (egal F-1—>S/TRSF-START marqué)
alors (affecte F-1/REQ "")

La restitution de la ressource sera simplement assurée lors du marquage de la place 2 de nom F-1—>S/TRSF-END. Elle devra conjointement assurer l'émission de la valeur du marquage dans le producteur/consommateur S/PROD. La règle qui assure ce traitement est ainsi la suivante :

R22 : (egal F-1—>S/TRSF-END marqué) (egal B1-F-1—>S 0)
alors (put-fifo S/PROD W0-F-1—>S)(affecte R2 libre)(affecte B1-F-1—>S 1)

II.4.4 - Intérêt des métarègles

L'utilisation des métarègles est à la fois simple et limitée. Leur principal objectif consiste pour des raisons de performances évidentes, à minimiser la taille de la base de règles courantes. En effet, chaque règle ou ensemble de règles, est définie pour répondre à un problème particulier correspondant à un marquage donné. Il est facile de déterminer, en analysant le comportement séquentiel des graphes de commande, un intervalle de temps où un ensemble de règles est susceptible de répondre à un problème précis. A l'exception de cet intervalle, il est tout à fait inutile de conserver ces règles dans la base courante. Leur évaluation, qui nécessairement se terminerait par l'interprétation d'une prémisse retournant un résultat faux, ne ferait que ralentir le cycle de fonctionnement général du moteur d'inférence. Si l'on considère par exemple le fonctionnement du centre d'usinage, trois règles sont évaluées en permanence pour déterminer l'attribution de la ressource F/RESSOURCE (sous l'hypothèse d'une gestion assurée par le niveau hiérarchique : règles R3, R4, R5) tandis que la règle R6 est dédiée à sa restitution. En tenant compte de l'importance du temps d'usinage par rapport au cycle d'inférence du niveau hiérarchique, on constate que les règles d'attribution vont être évaluées inutilement un grand nombre de fois pendant l'opération d'usinage. La définition de deux métarègles est ici pleinement justifiée.

La première va, si la ressource F/RESSOURCE est occupée, soustraire de la base de règles courantes les règles d'attribution devenues désormais inutiles. La seconde métarègle possède un comportement dual de la précédente : la ressource est de nouveau libre, il faut réintroduire les règles concernant sa gestion et enlever celle qui s'occupe de sa restitution. Les deux métarègles sont donc les suivantes :

Méta 1: (egal F/RESSOURCE occupe)
alors (retrait-règle R3 R4 R5) (ajout-règle R6)

Méta 2 : (egal F/RESSOURCE libre)
alors (retrait-règle R6) (ajout-règle R3 R4 R5)

L'évaluation de ces deux métarègles est immédiate. Leurs prémisses et conséquents ne concernent que des faits internes ou des manipulations de listes ; elles permettent ainsi de minimiser la taille de la base de règles courante et d'en améliorer de façon notable les performances. Le compromis classique à définir entre taille de la base de métarègles et taille de la base de règles n'est pas ici significatif. En effet, l'évaluaton d'une règle comportant des faits externes est toujours pénalisante pour les performances du système. La définition d'une métarègle pour enlever une règle au moment opportun est toujours intéressante et diminuera de façon perceptible le temps nécessaire à un cycle d'inférence complet.

De façon plus significative encore l'utilisation de métarègles pour la gestion du robot R2 permet de supprimer au minimum les 9 règles dédiées à son attribution (R11 à R19) lorsqu'il n'est pas disponible. Méta 3 et Méta 4 sont les deux règles qui permettent de basculer d'un état à l'autre :

Méta 3: (different R2 libre)
alors (retrait-règle R11 R12 R13 R14 R15 R16 R17 R18 R19)
(ajout-règle R22bis*)

Méta 4 : (egal R2 libre)
alors (retrait-règle R22bis*) (ajout-règle R11 R12 R13 R14 R15
R16 R17 R18 R19).

* la règle R22bis correspond à une extension de la règle R22 qui normalement ne se préoccupe que de la restitution de la ressource

par le processus f-1—>S. Ici R22bis est une règle générale qui permet de récupérer la ressource quel que soit le processus qui la possède :

R22bis :
```
(de restitution-ressource-R2 ()
    (COND
        ( ... )
        .
        .
        .
        ((AND (egal F-1—>S/TRSF-END marque)
              (egal B1-F-1—>S 0))
         (faits-int '(put-fifo S/PROD W0-F-1—>S))
         (faits-int '(affecte R2 libre))
         (faits-ext '(affecte B1-F-1—>S 1)) t))
        .
        .
        .
        (t t)))
```

II.4.5 - Bilan de mise en oeuvre de l'exemple d'implantation

La commande de la cellule flexible a été implantée dans sa totalité sur les 3 automates TSX et régulée par le niveau hiérarchique installé sur IBM/AT. L'ossature des réseaux de Petri ayant été préservée, les grafcets comportent donc au total le même nombre de places et de transitions que le modèle initial à l'exception des places de liaisons et des ressources dont la modélisation et la gestion sont assurées par le niveau hiérarchique. Ce dernier, à titre indicatif, comporte 54 règles de production ou fonctions LISP, 14 métarègles et 167 variables internes ou externes modélisant des bits, des mots, des places, ... ainsi que des structures de données telles les fifos. Enfin, l'état du procédé est dynamiquement visualisé sur un synoptique représentant les différents constituants significatifs et indiquant les pièces à travers la cellule en traduisant, par des couleurs, l'état d'avancement de leurs gammes opératoires.

Les performances de l'ensemble, à travers les solutions techniques retenues, sont sensiblement bridées par l'évaluation du niveau hiérarchique. En effet, un cycle d'inférence complet, incluant les requêtes minimales nécessaires pour connaître la valeur des faits externes, nécessite un temps de l'ordre de 15 à 20

secondes. Cela signifie qu'un processus est susceptible d'attendre 20 secondes que le niveau hiérarchique prenne conscience de son marquage et dégèle, si les conditions le permettent, sa transition aval. Deux facteurs essentiels limitent la rapidité de l'évaluation :

(i) La communication des requêtes transite à partir de l'IBM à travers un premier automate servant de passerelle pour accéder au réseau, puis à travers le réseau pour atteindre un second automate ; le chemin suivi en retour par la réponse est identique (cf. fig. 3). On peut ainsi estimer le temps d'une seule requête à 0,1 seconde. Cette lenteur est atténuée par le fait que la plupart des requêtes retourne un ensemble d'informations (valeur de 8 bits, marquage courant de l'ensemble des grafcets d'un automate, ...) plutôt qu'une information unitaire.

(ii) L'IBM PC/AT ne permet sous DOS de ne gérer que 640 Koctets de mémoire vive. Or, nous avons dû installer de façon résidente tout le protocole de communication. La mémoire restante disponible pour Le-LISP est insuffisante de par la taille de ses zones vierges. Il s'ensuit des "garbage collectors" fréquents qui grèvent les performances de l'évaluateur LISP.

Les temps de réponse peuvent cependant rester compatibles avec les actions assurées par la commande. Si, comme dans notre exemple, les opérations d'usinage et de transfert sont de l'ordre de plusieurs minutes, les 20 secondes d'un cycle d'inférence sont admissibles. Par contre, si les temps opératoires sont inférieurs à ceux du niveau hiérarchique, une autre mise en oeuvre devra être envisagée (supervision exclusivement locale assurée par les automates eux-mêmes par exemple).

II.5 - CONCLUSION

L'exemple présenté ici a utilisé tout au long de sa définition la démarche méthodologique définie dans le projet C.A.S.P.A.I.M. Il en résulte, outre un gain de temps appréciable pendant la conception, une fiabilité importante, résultant d'une approche rigoureuse et progressive. Le modèle RdPSAC validé est transcrit systématiquement en grafcet dans un contexte multi-automates tandis que les stratégies de fonctionnement, dissociées de la commande des automatismes, sont réalisées au niveau hiérarchique. Cette répartition de la commande au sens large en augmente à la fois la maintenabilité et la flexibilité.

L'approche normalisée concernant les modèles, associée à une structuration rigcureuse, constitue un apport essentiel concernant la clarté et la lisibilité des programmes. En assurer une reprise par des modifications éventuelles peut désormais être envisagée sereinement tant la mise en oeuvre est limpide.

La flexibilité doit être perçue à deux niveaux. Le premier concerne les programmes et rejoint en quelque sorte le point précédent. Les primitives de communication banalisées ainsi que la structuration utilisée permettent de modifier simplement les réseaux définis afin d'y adjoindre de nouvelles séquences opératoires. Le second point implique le niveau hiérarchique dissocié de la commande. Sa définition sous forme de règles, ainsi que l'utilisation d'un mode "espion" pour l'acquisition de ses données, autorisent à tout moment la modification dynamique des stratégies de fonctionnement. On dispose ainsi non seulement d'une commande flexible régulée par le niveau hiérarchique, mais encore d'un niveau hiérarchique flexible susceptible d'évoluer en fonction des impératifs de la production.

CONCLUSION

La mise en oeuvre d'un exemple à caractère industriel a permis de valider la réalisation tout en dégageant ses faiblesses. La flexibilité obtenue dans la commande par l'utilisation d'une supervision non procédurale est effectuée aujourd'hui au détriment des performances de l'ensemble du système de production. Il conviendra de rechercher un meilleur compromis entre rapidité/ flexibilité. L'évolution prévisible des nouveaux matériels et surtout leur ouverture vers le monde extérieur devrait logiquement à terme atténuer l'importance de ce compromis. L'apport d'architecture à processeurs dédiés améliorera de façon importante les communications entre machines sans pour autant ralentir leur tâche essentielle : l'évaluation de leur code programmé. L'utilisateur pourra cette fois se consacrer totalement à la réalisation d'une commande optimale où aucune concession ne sera faite à la flexibilité.

CONCLUSION GENERALE

CONCLUSION GENERALE

La démarche proposée dans le cadre du projet C.A.S.P.A.I.M. a consisté à étudier la faisabilité d'une génération quasi automatique et systématique du système de commande répartie de cellules flexibles de production en Industrie Manufacturière. Dans ce cadre, nous nous sommes attachés à la présentation des principaux résultats qui permettent de conclure à la faisabilité de l'automatisation de l'implantation.

Déduit du modèle de conception utilisé dans la démarche C.A.S.P.A.I.M., nous avons réalisé le modèle d'implantation en assurant :

(i) : La transposition systématique et rigoureuse du modèle RdPSAC en grafcet. Les extensions, arcs adaptatifs et coloration, gérées explicitement par programmation garantissent un fonctionnement identique entre le modèle généré lors de la conception et celui mis en oeuvre pour l'implantation.

(ii) : La généralisation de la mise en oeuvre de la communication entre automates distincts par une banalisation des primitives de communication au niveau hiérarchique. Cette réalisation facilite ainsi la répartition des processus sur les différents organes informatiques en standardisant leurs modes d'échange.

(iii) : La définition d'un niveau hiérarchique hétérogène et asynchrone vis à vis de la commande. Il a pour objet de superviser le modèle de la commande tout en augmentant sa flexibilité dans un contexte de fonctionnement non-autonome.

(iv) : La mise en oeuvre effective de ce niveau hiérarchique interfacé avec des automates programmables sur lesquels sont implantés les grafcets. Un moteur d'inférence utilisant des règles de production a été retenu pour sa réalisation. Cette solution favorise une définition modulaire, dynamique et souvent a posteriori des stratégies de fonctionnement. L'interactivité de l'interpréteur de règles alliée à l'aspect "déclaratif" de la description des stratégies augmente très sensiblement le caractère d'adaptabilité de la commande.

Cette implantation permet de dissocier nettement les tâches de fonctionnalités différentes. La commande exprimée sous la forme de grafcet caractérise le séquencement naturel des actions à réaliser et traduit notamment les circulations des différentes gammes opératoires dans le système. Le niveau hiérarchique, basé sur l'utilisation de règles déclaratives, permet facilement d'intégrer les spécificités de paramétrage et de règlements stratégiques de production conduisant à faciliter la recherche des solutions les plus adaptées par simple comparaison des performances. Cette implantation rend l'ensemble flexible, non seulement du fait de l'architecture matérielle de la cellule, mais également au niveau de la commande qui loin d'être figée, est à même d'être adaptée aux impératifs de production les plus divers et d'évoluer dans le temps vers la recherche d'une optimisation des performances.

Enfin, l'approche proposée garantit la parfaite conformité entre le modèle d'implantation et le modèle de conception. Les simulations, réalisées lors de la conception et visant tout aussi bien à lever les blocages et les indéterminismes qu'à obtenir des résultats statistiques sur la production garantissent ainsi le bon fonctionnement du modèle d'implantation. C'est dans ce sens que nous avons proposé une transposition rigoureuse conservant les propriétés et caractéristiques de comportement dynamique du modèle de conception. Une telle méthodologie à l'avantage d'annuler tout à la fois les erreurs de transposition et de réduire le temps consacré à ce travail. Elle augmente en conséquence la maintenabilité du produit final ; la standardisation du modèle, le nombre restreint des primitives utilisées et la banalisation de l'interfaçage avec le niveau hiérarchique facilite la lisibilité, donc la compréhension du système de commande.

A partir des travaux d'ensemble présentés dans ce mémoire, il apparaît plusieurs champs prospectifs.

Le premier concerne l'intérêt d'une structuration du niveau hiérarchique. En effet, si une stratégie de chainage avant est parfaitement adaptée à la résolution des indéterminismes et/ou conflits de la commande, il conviendrait par contre de proposer une stratégie de chainage arrière en vue du traitement des défaillances [ATA 87]. La structuration hiérarchisée des règles du niveau hiérarchique devrait ainsi permettre d'assurer les regroupements fonctionnels tels que la supervision normale, la gestion des pannes, la planification, ... et d'associer à chaque niveau le mode de résolution le plus adéquat.

Un deuxième champ d'investigation concerne la puissance de description des modèles utilisés et le meilleur choix de compromis à proposer entre d'une part la complexité des outils et d'autre part la facilité de mise en oeuvre et/ou d'assistance logicielle.

Une première démarche consiste, pour adapter la puissance de description du modèle RdP à la complexité du problème traité, à utiliser si besoin les RdP à prédicats |GEN 79| ou encore les RdP à structure de données |SIB 88|. La résolution des stratégies de commande nécessite toutefois la mise en oeuvre d'une structure de décision.

Dans le cadre du projet C.A.S.P.A.I.M., il nous semblerait plus adapté de conserver au niveau commande un modèle basé sur les RdPSAC sans prendre en compte à ce niveau d'informations supplémentaires qui ne seraient en fait justifiées que pour une prise de décision de niveau supérieure. A notre sens, le niveau hiérarchique doit assurer lui-même la gestion de structures de données complémentaires lui permettant de disposer des informations nécessaires aux prises de décision. Le niveau hiérarchique et le graphe de commande disposeraient ainsi respectivement d'une image procédé suffisante pour assurer chacun à leur niveau les "prises de décision" et l'"élaboration des commandes".

BIBLIOGRAPHIE

[DEF 85] M. Defaux, A. Loréal
"Atelier flexible : commencez petit."
L'usine nouvelle, pp. 47 à 59, Avril 1985

[MOR 88] G. Morel, M. Roesch, M. Veron
"Génie Productique, Génie -X."
Congrés AFCET Automatique, Grenoble, Novembre 1988

CHAPITRE I

Introduction

[PRU 87] F. Prunet, J.L. Sturlese, C. Cazalot, E. Ginestet, D. Panaget, G. Dechenaux, P. Llorca
"Méthodologie et implantation automatique de commande d'automatisme à l'aide de la chaîne PIASTRE."
APII, vol 21, n° 4, pp. 299 à 322, 1987

[RHE 88] G. Rhemes
"Automaticiens : voici votre atelier de génie logiciel !"
Electronique Industrielle, n° 142, pp. 49 à 53, 1988

[TIX 88] J.M. Tixador, P. Lhoste
"Du GRAFCET à l'approche objet."
Forum AFCET : "Logiciels pour la conception assistée des systèmes de commande.", Nancy, Avril 1988

Description du modèle

[BAR 88] J. M. Barbez, E. Craye, J. C. Gentina, J. Mayet
"Hierarchical level and implementation for analysis and synthesis of control and reliability of flexible manufacturing systems."
Congres IMACS n° 12, Vol 3, pp. 552 à 558, Paris, Juillet 1988

[BOU 88] J.P. Bourey
"Structuration de la partie procédurale du système de commande de cellules de production flexibles dans l'industrie manufacturière."
Thèse de Docteur de l'Université de Lille,1988

[BRA 83] G. W. Brams
"Réseaux de Pétri : théorie et pratique."
Tome 2, Editions Masson 1983

[COR 79] D. Corbeel
"Schéma de cablage et schéma de contrôle. Application à la simulation et à la gestion des processus industriels."
Thèse de Doct. de spécialité, LILLE, 1979

[COR 80] D. Corbeel
"Formal description of processes systems and exception handling."
Mini & Micro, Proc. pp. 335 à 339, BUDAPEST, Septembre 1980

[COR 85] D. Corbeel, C. Vercauter, J.C. Gentina
"Application of an extension of Petri nets to modelization of control and production processes."
Sixth European Workshop on application and theory of Petri nets, pp. 53 à 74, Juin 1985

[HAC 75a] M. Hack
"Petri net language."
MIT Computation Structure Group, Memo 124, 1975

[HAC 75b] M. Hack
"Decision problems for Petri nets and vector addition systems."
MIT, MAC Tech. Memo 59, 1975

[MAR 88] J. Martinez, P. R. Muro, M. Silva, S. F. Smith, J. L. Villarroel
"Merging artificial intelligence techniques and Petri nets for real time scheduling and control of production systems."
Congres IMACS n° 12, Vol 3, pp. 528 à 531, Paris, Juillet 1988

[VAL 78] R. Valk
"Self-modifying nets, a natural extension of Petri nets."
ICALP, Lect. Notes in Computer Sc., n° 62, pp. 464 à 476, Springer, Berlin, 1978

[VAL 88] R. Valette, J. Cardoso, H. Atabakhche, M. Courvoisier, T. Lemaire
"Petri nets and Production rules for Decision levels in FMS control."
Congres IMACS n° 12, Vol 3, pp. 522 à 524, Paris, Juillet 1988

Présentation du projet CASPAIM

[BAL 87] G. Balbo, G. Chiola, G. Francceschinis, G. Molinar Roet
"Generalized stochastic Petri nets for the performance evaluation of F.M.S."
IEEE, International conference on robotics and automation, pp. 1013 à 1018, USA , Avril 1987

[BOU 87] J.P. Bourey, J.C. Gentina
"Computer aided design for structuration and representation of control of flexible manufacturing systems."
8th European Workshop on application and theory of Petri-nets, pp. 117 à 135, Espagne, Juin 1987

[CAS 85] E. Castelain, D. Corbeel, J.C. Gentina
"Comparative simulations of control processes described by Petri-nets."
COMPINT'85, Montreal, Septembre 1985

[CAS 87] E. Castelain
"Modélisation et simulation interactive de cellules de production flexibles dans l'industrie manufacturière."
Thèse de Docteur de l'Université de Lille, 1987

[CAS 88] E. Castelain, J.C. Gentina
"Petri-nets and artificial intelligence in the context of simulation and modelling of manufacturing systems."
IMACS, International symposium on System modelling and simulation, Italie, Septembre 1988

[CHA 86] J. Chailloux
"Le_LISP, Manuel de référence."
I.N.R.I.A., Mai 1986

[COL 86] A. Collongues, J. Huges, B. Laroche
"MERISE : méthode de conception."
Ed. Dunod Informatique, 1986

[DES 85] B. Descotes-Genon, P. Ladet
"Outils graphiques pour la modélisation et la simulation d'applications de commande séquentielle."
Congrès AFCET, pp. 559 à 570, Toulouse, octobre 1985

[KAP 87] M. Kapusta, J.C. Gentina
"Introduction to a first step of the aided design of the control system of flexible manufacturing cells."
IEEE Montech'87-compint'87, pp. 258 à 262, Canada, Novembre 1987

[KAP 88] M. Kapusta
"Génération assistée d'un graphe fonctionnel destiné à l'élaboration structurée du modéle de la partie commande pour les cellules de production flexibles dans l'industrie manufacturière."
Thèse de doctorat de l'université. Université des sciences et techniques de Lille, Décembre 1988

[THO 85] J.P. Thomesse
"Les réseaux locaux industriels."
Ed. E.T.A., collection Novotique, 1985

CHAPITRE II

Introduction

[BAR 85] G. Barillier
"Automates multifonctions pour une approche structurée des automatismes."
Conférences Automation 1985

[BON 85] R. Bonetto
"Les ateliers flexibles de production."
Editions Hermes 1985

[CIN 87] F. Cinare
"FIP : un réseau local pour l'acquisition de données dans l'usine."
Minis et Micros n°283, 1987

[GAL 87] Y. Le Gal, M. Mitenne
"Un réseau local sûr de fonctionnement."
Minis et Micros n°277, 1987

[MIN 87] P. Minet
"MAP : un réseau local pour un environnement industriel automatisé."
T.S.I. vol. 6, n°2, 1987

[VER 86] M. Veron, E. Bajic, J. Richard
"Programmation of flexible manufacturing cell with integrated quality control."
8th International Conference on Industrial Robot Technology, Brussels, Belgique, Septembre 1986

Le fonctionnement en mode de marche normale

[BLA 85] B. Le Blanc
"Evolution de la fonction dialogue."
Conférences Automation 1985

[BOU 84] A. Bourjault
"Contribution à une approche méthodologique de l'assemblage automatisé : Elaboration automatique des séquences opératoires."
Thèse d'état, Université de Besançon, 1984

[BOU 88] J.P. Bourey
"Structuration de la partie procédurale du système de commande de cellules de production flexibles dans l'industrie manufacturière."
Thèse de Docteur de l'Université de Lille, 1988

[BRA 83] G.W. Brams
"Réseaux de Pétri : théorie et pratique."
Tome 1, Editions Masson 1983

[CAM 87] J.P. Campagne, C. Caplat
"Elaboration automatique de gammes d'assemblage."
2ème conférence internationale INRIA, Systèmes de production, pp. 537 à 550, Paris, Avril 1987

[CAS 87] E. Castelain
"Modélisation et simulation interactive de cellules de production flexibles dans l'industrie manufacturière."
Thèse de Docteur de l'Université de Lille, 1987

[CRO 75] CROCUS : J. BELLINO et al.
"Systèmes d'exploitation des ordinateurs : Principes de conception."
Editions Dunod 1975

[FRO 84] B. Froment, J.J. Lesage
"Productique : Les techniques de l'usinage flexible."
Editions Dunod 1984

[GIR 84] C. Girod
"Conception et réalisation d'un logiciel de simulation de réseaux de Pétri."
Thèse de Doct. Ing., I.D.N., Lille, 1984

[GUI 85] G. Guidez
"Le Grafcet : Macro-représentation et structuration du traitement des automatismes."
Conférences Automation 1985

[JEN 81] K. Jensen
"Coloured Petri nets and invariant method."
Theorical computer science n°14, pp. 317 à 336, North Holland Pub. Comp., 1981

[LAU 86] J.L. Lauriere
"Intelligence Artificielle : résolution des problémes par l'homme et la machine."
Editions Eyrolles 1986

[PLO 87] N. Plouzeau, M. Raynal, J.P. Verjus
"Producteurs/Consommateur : quelques solutions réparties."
T.S.I. vol. 6, n°3, pp. 231 à 241, 1987

[ROU 87] F. Roubellat, V. Thomas
"Une méthode et un logiciel pour l'ordonnancement en temps réel d'ateliers."
2ème conférence internationale INRIA, Systèmes de production, pp. 87 à 101, Paris, Avril 1987

[SAH 87] A. Sahraoui, H. Atabakhche, M. Courvoiser, R. Valette
"Joining Petri nets and knowledge based systems for monitoring purposes."
IEEE, pp. 1160 à 1165, 1987

Sureté de fonctionnement

[ATA 87] H. Atabakhche, D. Simonetti Barbalho, R. Valette, M. Courvoisier
"Commande d'ateliers : un compromis est-il possible entre une approche graphique et une approche intelligence artificielle?"
APII, vol 21, n° 4, pp. 377 à 394, 1987

[BAR 88] J.M. Barbez, E. Craye, J.C. Gentina, J. Mayet
"Hierarchical level and implementation for analysis and synthesis of control and reliability of flexible manufacturing systems."
Congres IMACS n° 12, Vol 3, pp. 552 à 558, Paris, Juillet 1988

[BOU 86] J.P. Bourey, D. Corbeel, E. Craye, J.C. Gentina
"Adaptive and coloured structured Petri nets for description, analysis and synthesis of hierarchical control and reliability of flexible cells in manufacturing systems."
1rst European Workshop on fault diagnostics, reliability and related knowledge-based approaches, ile de Rhodes, Septembre 1986
Vol 1, pp. 281 à 295, D. Reidel Publ. Comp., 1987

[BOU 87] D. Bouteille et al.
"Les automatismes programmables."
Cepadues-Editions 1987

[LAP 85] J.C. Laprie
"Sûreté de fonctionnement des systèmes informatiques et tolérance aux fautes : concepts de base."
T.S.I. vol 4, n° 5, pp. 419 à 429, 1985

[MAR 87] J. Martinez, P. Muro, M. Silva
"Modeling, validation and software implementation of production systems using high level Petri nets."
IEEE, pp. 1180 à 1185, 1987

[MAR 88] D. Martin
"Système expert d'aide au diagnostic technique et à la maintenance d'un réseau d'éclairage public."
Conférences Automation 1988

[SAH 87] A.E.K. Sahraoui
"Contribution à la surveillance et à la commande d'atelier."
Doctorat de l'université Paul Sabatier, Toulouse, 1987

[SOU 85] C. Sourisse
"Incidence de la sécurité et de la disponibilité sur les équipements pilotés par API."
Electronique Industrielle, n° 90, 1985

Réalisation du niveau hiérarchique

[AYE 86] M. Ayel, E. Pipard, M.-C. Rousset
"Le controle de cohérence dans les bases de connaissances."
PRC-GRECO Intelligence artificielle, pp. 171 à 186, Novembre 1986

[BEL 85] G. Bel, D. Dubois
"Modélisation et simulation de systèmes automatisés de production."
APII, vol. 19, n° 1, pp. 3 à 43, 1985

[BEU 86] P. Le Beux, D. Fontaine
"Un système d'acquisition des connaissances pour systèmes experts."
T.S.I., vol. 5, n° 1, pp. 7 à 20, 1986

[CHA 87] B. Chaib Draa, P. Millot, D. Willaeys
"Architecture pour les systèmes d'intelligence artificielle distribuée."
IEEE Montech'87, pp. 64 à 69, Novembre 1987

[FAR 85] H. Farreny
"Les systèmes experts : principes et exemples."
CEPADUES-EDITIONS 1985

[GIA 85] F. Giannesini, H. Kanoui, R. Pasero, M. Van Caneghem
"Prolog."
InterEditions 1985

[HAU 86] F. Hautin, A. Vailly
"La coopération entre systèmes experts."
PRC-GRECO Intelligence artificielle, pp. 187 à 199, Novembre 1986

[HOS 84] F. Hoste
"Les réseaux locaux d'entreprises : marchés et technologies."
édiTESTS 1984

[HUL 84] J.M. Hullot
"Programmer en CEYX."
Rapport de l'INRIA, 1984

[KNI 85] C.G. Knickerbocker, R.L. Moore, L.B. Hawkinson, M.E. Levin
"The PICON expert system for process control."
Congrés Avignon sur systèmes experts, 1985

[KOD 85] Y. Kodratoff, J.G. Ganascia
"Demonstration automatique de théorèmes et systèmes experts."
Congrés Avignon sur systèmes experts, 1985

[LAU 82] J.L. Laurière
"Représentation et utilisation des connaissances."
T.S.I., vol 1, n° 1, pp. 25 à 42, n° 2, pp. 109 à 133, 1982

[LAU 86] J.L. Laurière
"Un langage déclaratif : SNARK."
T.S.I., vol 5, n° 3, pp. 141 à 172, 1986

[VOY 87] R. Voyer
"Moteurs de systèmes experts. Première partie : fondements."
Editions Eyrolles 1987

[ZHA 85] Zhao Xi
"La logique temporelle et l'analyse du système séquentiel automatisé."
Mémoire de D.E.A., I.D.N., Lille, 1985

Conclusion

[BON 86] A. Bonnemay
"Systel : un logiciel de quatrième génération pour la conduite d'installations industrielles."
Convention automatique, productique. Paris, Mai 1986

[BOU 87] J.P. Bourey, D. Corbeel, E. Craye, J.C. Gentina
"Utilisation des réseaux de Petri structurés adaptatifs colorés dans l'analyse et la synthèse du contrôle hiérarchisé de processus discontinus. Partie A : les modèles de description."
APII, vol. 21, n° 4, pp. 343 à 362, 1987

CHAPITRE III

Réalisation effective du logiciel

[APT 85] APTOR
"Factor : an Open Local Area Network."
Manuel de présentation, 1985

[RES 85] Réseau Telway 7
"Utilisation, Mise en œuvre."
Manuel Télémécanique, Juin 1985

[TRI 84] J.M. Trio
"Microprocesseurs 8086-8088 : Architecture et programmation."
Editions Eyrolles, 1984

Exemple de réalisation

[MAY 87] J. Mayet
"Sur la conception d'un atelier flexible de fabricatioN mécanique."
Mémoire Ing. C.N.A.M., Centre Régional de Lille, Juin 1987

CONCLUSION GENERALE

[ATA 87] H. Atabakhche
"Utilisation conjointe de l'intelligence artificielle et des réseaux de Petri : Application au controle d'exécution d'un plan de fabrication."
Doctorat de l'Université Paul Sabatier en Informatique, Toulouse,1987

[GEN 79] H. Genrich, K. Lautenbach, P. Thiagarajan
"Elements of general net theory."
Proc. of the advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979

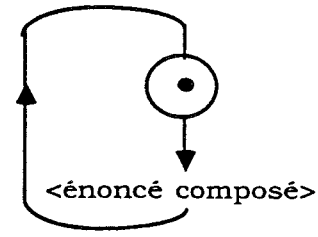
[SIB 88] C. Sibertin-Blanc
"L'utilisation des réseaux de Petri à objets dans le domaine des systèmes d'information."
Séminaire sur les réseaux de Petri, IIE, Octobre 1988

Annexe A:

Syntaxe associée à la définition d'un graphe de processus.

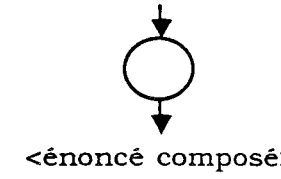
PROCESSUS

<processus> ::=



ENONCE COMPOSE

<énoncé composé> ::= <énoncé simple> / <énoncé simple>

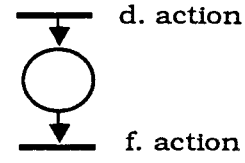


ENONCE SIMPLE

<énoncé simple> ::= <action> / <alternative> / <répétitive>

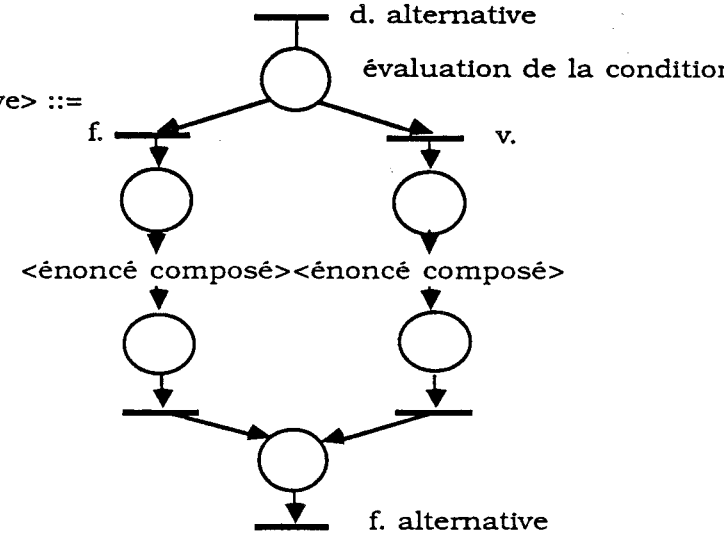
ACTION

<action> ::=



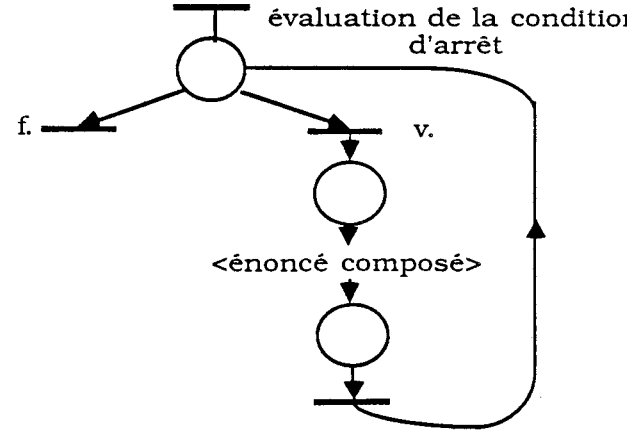
ALTERNATIVE

<alternative> ::=



REPETITIVE

<répétitive> ::=



Annexe B:

Contenu des bases de faits et de règles pour l'exemple du processus d'usinage.

Gestion des processus du centre d'usinage.

```
;    * * * * * * * * * * * * * * * * *
;    *                               *
;    *      BASE DES FAITS           *
;    *                               *
;    * * * * * * * * * * * * * * * * *
```

```
(SETQ basedesbitsinternes '(f-2/ack f-1/ack occupe faux
    demande-ressource-f-2 vrai demande-ressource-f-1 marque libre
    f/ressource vide))

(PLIST 'f-2/ack '(nature bit nature-bis interne valeur 0))
(PLIST 'f-1/ack '(nature bit nature-bis interne valeur 0))
(PLIST 'occupe '(nature bit nature-bis interne valeur 0))
(PLIST 'faux '(nature bit nature-bis interne valeur 0))
(PLIST 'demande-ressource-f-2 '(nature bit nature-bis interne valeur 0))
(PLIST 'vrai '(nature bit nature-bis interne valeur 1))
(PLIST 'demande-ressource-f-1 '(nature bit nature-bis interne valeur 0))
(PLIST 'marque '(nature bit nature-bis interne valeur 1))
(PLIST 'libre '(nature bit nature-bis interne valeur 1))
(PLIST 'f/ressource '(nature bit nature-bis interne valeur 1))
(PLIST 'vide '(nature bit nature-bis interne valeur 0))


(SETQ basedesmotsinternes '(f-2/cons f-1/cons f-2/req f-1/req))

(PLIST 'f-2/cons '(nature mot nature-bis interne valeur 1))
(PLIST 'f-1/cons '(nature mot nature-bis interne valeur 1))
(PLIST 'f-2/req '(nature mot nature-bis interne valeur 0))
(PLIST 'f-1/req '(nature mot nature-bis interne valeur 0))


(SETQ basedesfifosinternes '(f-2/prod f-1/prod))

(PLIST 'f-2/prod '(nature fifo nature-bis interne valeur ()))
(PLIST 'f-1/prod '(nature fifo nature-bis interne valeur ()))


(SETQ basedesbitsexternes '(b7-f-2 b6-f-1 b5-f-2 b4-f-1 b3-f-2
    b2-f-1 b1-f-2 b0-f-1))

(PLIST 'bitsranges '(bit10 (b0-f-1 b1-f-2 b2-f-1 b3-f-2 b4-f-1 b5-f-2
    b6-f-1 b7-f-2)))
```

```
(PLIST 'b7-f-2
    '(numero-bit 7 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b6-f-1
    '(numero-bit 6 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b5-f-2
    '(numero-bit 5 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b4-f-1
    '(numero-bit 4 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b3-f-2
    '(numero-bit 3 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b2-f-1
    '(numero-bit 2 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b1-f-2
    '(numero-bit 1 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))
(PLIST 'b0-f-1
    '(numero-bit 0 nature bit nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-bit-externe
    fonction-restitution restitution-bit-externe))


(SETQ basedesmotsexternes '(w1-f-2 w0-f-1))

(PLIST 'w1-f-2
    '(numero-mot 1 nature mot nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-mot-externe
    fonction-restitution restitution-mot-externe))
(PLIST 'w0-f-1
    '(numero-mot 0 nature mot nature-bis externe station 1 valeur
    non-encore-calculee fonction-acquisition acquisition-mot-externe
    fonction-restitution restitution-mot-externe))
```

```
(SETQ basedesplacesexternes '(f-2/wait f-1/wait f-2/p->es f-1/p->es
    f-2/test-out  f-1/test-out))

(PLIST  'f-2/wait
    '(num-macro 0 num-etape 8 type-place 0 nature place nature-bis
    externe station 1 valeur non-encore-calculee fonction-acquisition
    acquisition-place-externe  fonction-restitution  (***)))
(PLIST  'f-1/wait
    '(num-macro 0 num-etape 2 type-place 0 nature place nature-bis
    externe station 1 valeur non-encore-calculee fonction-acquisition
    acquisition-place-externe  fonction-restitution  (***)))
(PLIST  'f-2/p->es
    '(num-macro 0 num-etape 11 type-place 0 nature place nature-bis
    externe station 1 valeur non-encore-calculee fonction-acquisition
    acquisition-place-externe  fonction-restitution  (***)))
(PLIST  'f-1/p->es
    '(num-macro 0 num-etape 5 type-place 0 nature place nature-bis
    externe station 1 valeur non-encore-calculee fonction-acquisition
    acquisition-place-externe  fonction-restitution  (***)))
(PLIST  'f-2/test-out
    '(num-macro 0 num-etape 7 type-place 0 nature place nature-bis
    externe station 1 valeur non-encore-calculee fonction-acquisition
    acquisition-place-externe  fonction-restitution  (***)))
(PLIST  'f-1/test-out
    '(num-macro 0 num-etape 1 type-place 0 nature place nature-bis
    externe station 1 valeur non-encore-calculee fonction-acquisition
    acquisition-place-externe  fonction-restitution  (***)))

(PLIST 'placesrangees
    '(place1  (f-1/test-out  f-2/test-out  f-1/p->es  f-2/p->es  f-1/wait
    f-2/wait)))
```

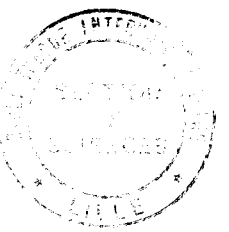
```
;    * * * * * * * * * * * * * * * * *
;    *                               *
;    *      BASE DES REGLES          *
;    *                               *
;    * * * * * * * * * * * * * * * * *
```

```
(SETQ compteurregle 10)

(SETQ baseabsoluedesregles '(r10 r9 r8 r7 r6 r5 r4 r3 r2 r1))

(SETQ basedesregles '(r10 r9 r8 r7 r6 r5 r4 r3 r2 r1))

(PLIST 'r10
     '(type production consequents-externes ((affecte b7-f-2 1))
     consequents-internes ((ajoute f-2/cons 1) (affecte f-2/ack 0))
     premisses-externes ((egal b7-f-2 0) (egal f-2/wait marque))
     premisses-internes ((egal f-2/ack 1))))
(PLIST 'r9
     '(type production consequents-externes ((affecte b6-f-1 1))
     consequents-internes ((ajoute f-1/cons 1) (affecte f-1/ack 0))
     premisses-externes ((egal b6-f-1 0) (egal f-1/wait marque))
     premisses-internes ((egal f-1/ack 1))))
(PLIST 'r8 '(enonce (emission-f-2/req) type libre))
(PLIST 'r7 '(enonce (emission-f-1/req) type libre))
(PLIST 'r6 '(enonce (restitution-f/ressource) type libre))
(PLIST 'r5 '(enonce (attribution-f/ressource) type libre))
(PLIST 'r4 '(enonce (demande-ressource-f-2) type libre))
(PLIST 'r3 '(enonce (demande-ressource-f-1) type libre))
(PLIST 'r2
     '(type production consequents-externes
          ((get-fifo f-2/prod w1-f-2))
     premisses-internes ((different f-2/prod vide))
     premisses-externes ((egal w1-f-2))))
(PLIST 'r1
     '(type production consequents-externes
          ((get-fifo f-1/prod w0-f-1))
     premisses-externes ((egal w0-f-1))
     premisses-internes ((different f-1/prod vide))))


(SETQ basedesmetaregles '(meta2 meta1))

(PLIST 'meta2
     '(consequents-internes ((ajout-regle r3 r4 r5) (retrait-regle r6))
     premisses-internes ((egal f/ressource libre))))
```

```
(PLIST 'meta1
    '(consequents-internes ((ajout-regle r6) (retrait-regle r3 r4 r5))
    premisses-internes ((egal f/ressource occupe))))
```

```
;     * * * * * * * * * * * * * * * * * * * * *  *
;     *                                          *
;     *        BASE DES FONCTIONS                *
;     *                                          *
;     * * * * * * * * * * * * * * * * * * * * *  *

(DE  demande-ressource-f-1  ()
     (COND
          ((AND
               (egal  f/ressource  libre)
               (egal  f-1/test-out  marque)
               (OR  (egal  w0-f-1  "f-")
                    (egal  w0-f-1  "f-t-")
                    (egal  w0-f-1  "t+f-")))
               (faits-int  '(affecte  demande-ressource-f-1  vrai))  t)
          (t  ())))

(DE  demande-ressource-f-2  ()
     (COND
          ((AND
               (egal  f/ressource  libre)
               (egal  f-2/test-out  marque)
               (OR  (egal  w1-f-2  "f-")
                    (egal  w1-f-2  "f-t-")
                    (egal  w1-f-2  "t+f-")))
               (faits-int  '(affecte  demande-ressource-f-2  vrai))  t)
          (t  ())))
```

```
(DE attribution-f/ressource ()
    (COND
        ((AND
            (egal f/ressource libre)
            (egal demande-ressource-f-1 vrai)
            (egal demande-ressource-f-2 vrai))
            (faits-int '(affecte f/ressource occupe))
            (IF (<= (tpfrai 'w0-f-1) (tpfrai 'w1-f-2))
                (progn
                    (faits-int
                        '(affecte demande-ressource-f-1 faux))
                    (faits-ext '(affecte b0-f-1 1)))
                (faits-int '(affecte demande-ressource-f-2 faux))
                (faits-ext '(affecte b1-f-2 1))) t)
        ((AND
            (egal f/ressource libre)
            (egal demande-ressource-f-1 vrai))
            (faits-int '(affecte f/ressource occupe))
            (faits-int '(affecte demande-ressource-f-1 faux))
            (faits-ext '(affecte b0-f-1 1)) t)
        ((AND
            (egal f/ressource libre)
            (egal demande-ressource-f-2 vrai))
            (faits-int '(affecte f/ressource occupe))
            (faits-int '(affecte demande-ressource-f-2 faux))
            (faits-ext '(affecte b1-f-2 1)) t)
        (t ())))

(DE tpfrai (w)
    (COND
        ((egal w "f-") temps1)
        ((egal w "f-t-") temps2)
        ((egal w "t+f-") temps3)))

(DE restitution-f/ressource ()
    (IF (egal f/ressource occupe)
        (COND
            ((egal f-1/p->es marque)
                (faits-int '(affecte f/ressource libre))
                (faits-ext '(affecte b2-f-1 1)) t)
            ((egal f-2/p->es marque)
                (faits-int '(affecte f/ressource libre))
                (faits-ext '(affecte b3-f-2 1)) t)
            (t t)) ()))
```



```
(DE  emission-f-1/req  ()
     (COND
          ((AND
               (egal  f-1/test-out  marque)
               (egal  b4-f-1  0)
               (OR  (egal  w0-f-1  "f+")
                    (egal  w0-f-1  "f+t+")
                    (egal  w0-f-1  "t+f+")))
               (faits-int  '(affecte  f-1/req  w0-f-1))
               (faits-ext  '(affecte  b4-f-1  1))  t)
          (t  t)))

(DE  emission-f-2/req  ()
     (COND
          ((AND
               (egal  f-2/test-out  marque)
               (egal  b5-f-2  0)
               (OR  (egal  w1-f-2  "f+")
                    (egal  w1-f-2  "f+t+")
                    (egal  w1-f-2  "t+f+")))
               (faits-int  '(affecte  f-2/req  w1-f-2))
               (faits-ext  '(affecte  b5-f-2  1))  t)
          (t  t)))
```

RESUME

Nous présentons dans ce mémoire une méthodologie d'implantation permettant la mise en œuvre effective de la commande de systèmes flexibles de production discontinue.

Notre démarche assure la génération d'une commande exprimée en Grafcet alors que le modèle initial de conception utilise les Réseaux de Petri Structurés, Adaptatifs et Colorés. Le Niveau Hiérarchique, dont le role est de paramétrer le graphe de commande en vue d'une flexibilité accrue, est décrit par un ensemble de régles de production associé à un ensemble de fonctions LISP; un moteur d'inférence en assure l'évaluation. Notre objectif a été de respecter rigoureusement les données issues de la conception afin d'obtenir un modèle d'implantation qui ne remette pas en cause les validations antérieures, notamment les propriétés du modèle de conception. La systématisation d'une telle approche apporte un gain de temps non négligeable dans la réalisation de l'implantation tout en augmentant la fiabilité assurée par la cohérence entre les modèles de conception et d'implantation.

Nos travaux ont abouti à l'implantation effective de la commande vers un ensemble d'automates programmables industriels supervisés de façon asynchrone par le niveau hiérarchique installé sur un micro-ordinateur. Un exemple de dimension industrielle illustre les principaux résultats du mémoire.

MOTS-CLEFS :

SYSTEMES DE PRODUCTION FLEXIBLE
MODELISATION
IMPLANTATION
RESEAUX DE PETRI
GRAFCET
NIVEAU HIERARCHIQUE
LANGAGE LE_LISP