50376
1987
91

N° d'ordre : 132

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

Ahmed KARFIA

PRESENTATION D'UNE BASE DE CONNAISSANCES ADAPTEE A LA MODELISATION PAR RESEAUX DE PETRI STRUCTURES DU CONTROLE DES PROCESSUS DE PRODUCTION DISCRETISES

soutenue le 9 Juillet 1987 devant la Commission d'Examen

Membres du Jury :	M. TOULOTTE	Rapporteur
	M. PRUNET	Rapporteur
	M. GENTINA	Examinateur, Directeur de Thèse
	M. COMYN	Examinateur
	M. CORBEEL	Examinateur

PLAN GENERAL

AVANT-PROPOS

A V A N T - P R O P O S

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. Qu'ils trouvent ici le témoignage de toute ma reconnaissance.

Je tiens également à remercier :

Monsieur TOULOTTE, Professeur à l'U.S.T.L.F.A.,
Monsieur PRUNET, Professeur à l'U.S.T.L. de Montpellier,
Monsieur COMYN, Professeur à l'I.U.T. d'Informatique de Lille I,
Monsieur CORBEEL, Maitre de Conférence à l'I.D.N.,

pour l'honneur qu'ils me font en participant à mon jury de thèse.

Je suis très reconnaissant à Monsieur GENTINA, Monsieur CORBEEL et 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.

Enfin, je remercie les personnes qui ont assuré la réalisation matérielle de ce mémoire : Mademoiselle DENYS pour la dactylographie et Madame FERRAR pour la reprographie.

INTRODUCTION GENERALE

INTRODUCTION GENERALE

L'évolution actuelle des moyens de production est dominée par le double souci prioritaire d'accroîssement de la productivité et de limitation des coûts de fabrication.

Il n'est pas utile de rappeler ici les impératifs qui sont la cause de cette mutation, essentiellement liés aux contraintes du marché dans le contexte économique actuel. Les orientations prises favorisent l'introduction de toutes les techniques d'automatisation dans le monde de la production et conduisent à une importante transformation des moyens et structures de production.

Cette automatisation s'accompagne de la prise en compte de nouveaux concepts dont les principaux sont une grande flexibilité et disponibilité accrue.

Cette évolution est indissociable du développement technologique qui a affecté les différents niveaux de production tant au niveau machine, visant l'autonomie de son fonctionnement, qu'au niveau système, visant à améliorer les performances de l'atelier.

L'importance des investissements, l'enjeu économique des résultats escomptés, la complexité et la taille des moyens considérés ne permettent plus une approche expérimentale et empirique des problèmes de conception et de conduite.

Une démarche méthodologique est devenue indispensable à tous les niveaux du cycle de vie d'un projet d'automatisation des moyens de production.

Cette démarche apparaît essentielle pour la conception du système de commande des processus de production automatisée. En effet, les caractères de flexibilité et de disponibilité source d'une plus grande productivité reposent principalement sur les aptitudes du système de commande à prendre en charge de tels objectifs.

Dans ce contexte, différents modèles de représentation d'un système de commande sont proposés :

- le Grafcet,
- les réseaux de Petri (RdP),
- les règles de production,
- l'approche par objet.

Le Grafcet est l'un des premiers outils utilisés dans la conception et la représentation des automatismes industriels complexes. Il offre des possibilités de modélisation très large

dans le cadre d'une très forte interprétation de la sémantique du contrôle.

La nécessité de représenter formellement des systèmes de plus en plus complexes a conduit à l'utilisation d'un modèle plus adaptable : les réseaux de Petri et extensions associées.

Le modèle RdP se caractérise par la finesse, la rigueur de description ainsi qu'un fort pouvoir de modélisation. Ils peuvent s'adapter à la représentation d'une large classe de systèmes automatisés à événements discrets.

Un certain nombre d'études et de projets, tels que SIMULISP (SIMulation Utilisant le langage LISP pour la conduite des Systèmes de Production /BEL 85/), proposent d'utiliser les techniques d'intelligence artificielle pour concevoir des systèmes de conduite d'ateliers flexibles.

Dans ces systèmes, qui utilisent un language déclaratif, chaque unité de connaissance est exprimée indépendamment des autres sous forme d'une règle de production (RP).

Ainsi, un système de conduite de procédés industriels écrit à l'aide de RP exprime le fonctionnement du système modélisé.

Notons à ce titre que l'approche par RP n'est pas nécessairement incompatible avec le formalisme de type réseau de Petri. En effet, on peut mettre en évidence l'équivalence entre

l'énoncé d'une RP et la notion de transition dans le modèle réseau de Petri /COU 87/.

L'utilisation systèmatique de RP sacrifie les aspects graphiques et procéduraux du modèle RdP au profit du caractère déclaratif.

L'approche orientée objet adoptée par un certain nombre de projets, tels que MOOREA (Maquette Orientée Objet pour la Réalisation de l'Etude des Automatismes /FRA 87/) écrit dans le langage SMALLTALK, confond données et procédures dans une structure unique. Un objet est à la fois dans un certain état, défini par ses variables d'instances, capable d'effectuer certaines actions (méthodes) et communique avec l'extérieur par envoi de messages. Cette approche utilise deux principes fondamentaux : l'héritage (ou recopie virtuelle) et l'instanciation.

A ce titre, MOOREA repose sur une structure de données puissante dont la création et la validation dynamique sont faites dans une vision cohérente orientée objet avec des objets informationnels pour les données et des objets foncteurs pour les processus.

Autour de ces méthodologies de représentation ont été développés des outils de conception assistée.

Notons, en particulier, sans vouloir être exhaustifs, des travaux sur un poste de travail pour automaticien tel PIASTRE /PRU

85/, la nouvelle gamme d'automates programmables industriels TSX série 7 de TELEMECANIQUE /GUI 85/ et le progiciel OMEGA de la société 3IP /3IP 85/. Les méthodologies mises en oeuvre 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.

En outre, le concept de réseau homogène permettant une implémentation répartie de la commande est complètement intégré dans la série 7 de TELEMECANIQUE /BARI 85/. En revanche, Piastre permet une validation complète par simulation en autorisant la modélisation de la partie "procédé" à l'aide du même outil que pour la partie commande et ceci grâce à l'interprétation du Grafcet.

Une autre approche, utilisée pour l'implantation des systèmes de commande, est basée sur les systèmes d'exploitation temps réel tel que, par exemple, iRMX, système d'exploitaion multitâches multi-utilisateurs orienté vers le temps réel /COU 86/ développé par INTEL.

Dans ce type d'approche, la difficulté provient essentiellement de l'expression du modèle de représentation de l'automatisme dans le langage d'implémentation. Cette étape n'est pas systématique et constitue une contrainte relativement pénalisante.

La prise en compte des fondements de l'analyse et de la programmation structurées constitue l'approche la plus sûre permettant d'aborder la complexité de grands systèmes de processus industriels. Ainsi, de par la rigueur induite dans sa mise en oeuvre, la structuration introduite au niveau des réseaux de Petri, permet la définition d'une méthodologie de description de systèmes de commande par des réseaux de Petri structurés. C'est sur cette idée que nous avons développé notre recherche.

Nous présentons ici le logiciel d'aide à la conception et à la validation de graphes de commande décrits à l'aide de réseaux de Petri structurés.

Notre souci a été de proposer au concepteur une démarche progressive et modulaire lui permettant de valider chaque étape de modélisation en utilisant les résultats précédemment synthétisés. Dans ce sens, nous avons orienté notre travail dans le but de satisfaire les contraintes suivantes :

- intéractivité dans la réalisation du logiciel afin d'apporter une assistance dans la phase de conception ;

- ouverture du logiciel afin de permettre son extension et son intégration à d'autres outils développés au Laboratoire d'Automatique et d'Informatique Industrielle (LAII) de l'I.D.N. (Institut Industriel du Nord) ;

- validation des graphes de commande par utilisation de logiciels de validation et de simulation développés au LAII et écrits respectivement en Pascal et en Le_Lisp ;

- transparence pour l'utilisateur par une très forte intégration de ces outils.

Ce mémoire contient quatre chapitres. Le premier concerne les modèles utilisés pour la modélisation du graphe de commande de processus discrets. Le deuxième introduit le logiciel d'aide à la conception et à la validation. Le troisième porte sur la description de la base de connaissances de ce logiciel et présente en détail la base de données constituée. Enfin, le dernier chapitre traite du système de gestion de la base de connaissances du logiciel.

CHAPITRE I

CHAPITRE I

C H A P I T R E I

MODELISATION DE LA PARTIE COMMANDE D'UN SYSTEME DE PRODUCTION INDUSTRIELLE

MODELISATION DE LA PARTIE COMMANDE D'UN SYSTEME DE PRODUCTION INDUSTRIELLE

Nous présentons dans ce chapitre, les composantes d'un systèmes de production industrielle ; nous indiquerons ensuite le rôle de la modélisation dans la conception d'un tel système ainsi que les raisons du choix que nous avons fait des réseaux de Petri comme modèle. Nous présenterons enfin le modèle de base que nous avons adopté pour la conception et la validation de la commande d'un système de conduite de procédés industriels.

1 - SYSTEME COMPLET

Un système de conduite de processus industriels est un ensemble complexe comprenant généralement trois parties /BRA 83/, /CAS 87/ :

1. La partie "procédé" regroupe l'ensemble des dispositifs matériels (robots, convoyeurs, centres d'usinage, unités de stockage, etc.) où circulent et sont transformés les produits.

2. La partie commande coordonne l'ensemble des processus pilotant le procédé. C'est en général la partie informatique du système, implémentée sur un réseau d'automates programmables industriels ou tout autre dispositif permettant la gestion temps réel de processus physiques.

3. Enfin, la partie décisionnelle, dont l'importance varie selon le degré de flexibilité du système à piloter, paramètre la partie commande en fonction d'objectifs de production et génère les décisions nécessaires à l'évolution du système.

Le caractère de "bon fonctionnement" du système est issu de la coopération de ces trois sous-ensembles. L'interaction entre ces trois parties est décrite par le schéma conceptuel suivant :

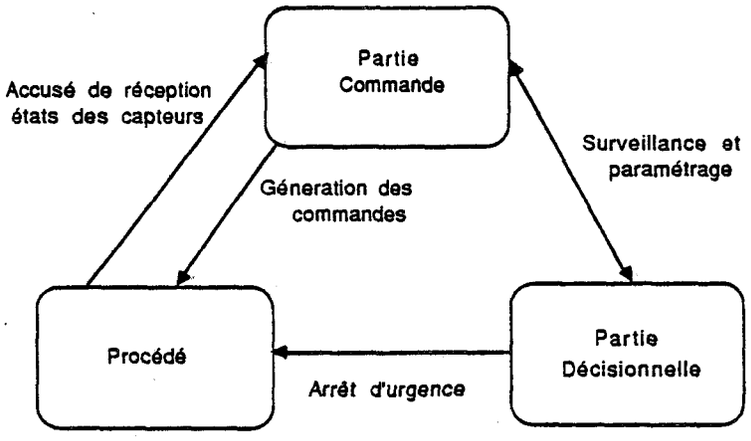


Figure I - 1

2 - PLACE DE LA MODELISATION DANS LA CONCEPTION D'UN SYSTEME DE PRODUCTION AUTOMATISEE.

La conception et la réalisation d'un système de production industrielle peuvent généralement être décomposées en cinq phases /GIR 84/ :

i) Analyse des besoins

Il s'agit de "mettre en forme" la demande de l'utilisateur en précisant les contraintes et les tâches à effectuer, en vue d'établir le cahier des charges.

ii) Préétude, ou définition de l'avant-projet

Le cahier des charges du projet est confié à des concepteurs, mécaniciens et automaticiens principalement, qui déterminent un certain nombre de solutions réalisables ; ces solutions sont analysées, prédimensionnées, évaluées et comparées entre elles ; les contraintes de coûts et de délais sont étudiées avec l'utilisateur ; la solution retenue permet de définir un cahier des charges d'étude.

iii) Conception de la partie mécanique du système

---

et

iv) Analyse de l'automatisation

Ces deux étapes sont interactives et parallèles ; les concepteurs mécaniciens doivent tenir compte des contraintes imposées par l'automatisation et vice-versa.

v) Mise au point et documentation

Il s'agit ici de mettre en évidence les possibilités de manoeuvre offertes par l'automatisme et ses performances, le cahier des charges de la commande et les algorithmes de gestion envisageables.

A tous ces niveaux, et même ultérieurement lors des phases de maintenance relevant du cycle de vie du projet, il est nécessaire de disposer d'outils permettant :

- la description initiale,
- les spécifications,
- l'évaluation des performances,
- l'analyse du fonctionnement

de l'automatisme à mettre en oeuvre, et assurant une bonne communication entre le concepteur et l'utilisateur.

Parmi les outils utilisés, on distingue :

- des méthodes de description et/ou de spécification : SADT, HIPO, SSAD, Grai, NBS, Réseau de Petri et Grafcet.

- des modèles formels de représentation : Langage Z, Qgert, Slam, Escl, Réseau de Petri et Grafcet.

On remarquera que les réseaux de Petri et le Grafcet peuvent être utilisés soit comme méthodologie de spécification, soit comme modèle formel de description de l'automatisme. Les autres méthodes de spécification sont dites semi-formelles, c'est-à-dire qu'elles utilisent simultanément un modèle graphique, un formalisme mathématique et le langage usuel. Parmi celles-ci, SADT est la plus employée ; elle utilise un ensemble structuré et hiérarchisé de diagrammes de deux types : diagrammes d'activités et diagrammes de données, et s'applique essentiellement à l'analyse de gestion.

HIPO est une méthode de documentation utilisant des diagrammes de structure hiérarchique à trois niveaux de description : niveau système, niveau programme et niveau module.

SSAD utilise les méthodes de spécification des systèmes d'information.

Grai est une méthode d'aide à la gestion de production ; elles s'inscrit dans cette ligne de modèles issus l'analyse systémique dont la plus connue est la méthode Merise ; cette méthode part du postulat selon lequel tout système de production automatisée se décompose en trois sous-systèmes (voir Figure I - 1) : physique, information et décision. C'est ce dernier système qui, avec les composantes des deux autres sous-systèmes qui lui sont liées, constitue le champ d'application de la méthode.

Ces différentes méthodes peuvent être combinées de la manière suivante :

- SADT, permet de modéliser, structurer, hiérarchiser l'information de manière standard ;
- Grai, produit une modélisation du système de décisions ;
- NBS, s'adapte à la représentation des fonctions spécifiques à une entreprise.

Les méthodes semi-formelles sont d'une utilisation facile, mais manquent de puissance d'analyse dans la mesure où elles utilisent le langage usuel. Aussi est-il intéressant de se tourner vers d'autres méthodes telles que le Grafcet et les réseaux de Petri.

Le Grafcet a été normalisé par l'Afcet et constitue de ce fait un outil standard par le plus grand nombre de concepteurs ; toutefois, ses possibilités de modélisation sont souvent plus limitées que celles des réseaux de Petri dans leur contexte le plus large.

3 - POURQUOI UTILISER LES RESEAUX DE PETRI ?

Pour répondre à cette question, nous donnerons d'abord des raisons d'ordre général, ensuite nous citerons quelques réalisations basées sur ce modèle.

a) Les réseaux de Petri (RdP), qui sont une représentation graphique condensée, permettent de modéliser des systèmes d'événements à haut degré de parallèlisme. Ce modèle prend en compte les interactions synchrones ou asynchrones /BRA 83/ entre les divers processus du système automatisé.

b) Du fait de leur caractère formel, les RdP rendent faciles la vérification des propriétés du système et l'évaluation des performances.

c) Depuis quelques années, les RdP et leurs dérivés (le Grafcet en particulier) sont utilisés pour modéliser des systèmes de production dans la phase de conception voire dans les phases d'implémentation et de suivi de production. Diverses études ont été réalisées, afin d'augmenter leur pouvoir de modélisation et de valider le "bon fonctionnement" des systèmes modélisés.

d) Les RdP permettent plusieurs niveaux de représentation, dans la mesure où un ensemble d'actions peut être regroupé par une seule place (macro-place) ; la modélisation peut donc procéder par une série de décompositions de plus en plus fines.

QUELQUES REALISATIONS :

Le groupe SYSECA propose le produit OVIDE, écrit en Fortran, comme outil de validation des RdP généralisés. Ce logiciel permet l'édition graphique du réseau, puis analyse le modèle en détectant les cas d'interblocage (dead lock), de famine, de bouclage, de non-réinitialisation. Il permet également la réduction du graphe et la recherche des invariants. Le système ARPEGE est un sur-ensemble d'OVIDE qui permettrait d'intégrer certaines extensions des RdP.

Le LAAS, en collaboration avec le GIE Renault Recherche Innovation, a mis au point PSI (Petri-net based SImulator) /ALA.84/. PSI a été développé en Pacal sur un micro 8 bits. Il permet de simuler des calculs statistiques aux places et aux transitions du réseau. Le projet SEDRIC, dans la continuation de PSI, permet d'intégrer les RdP colorés.

Citons également le logiciel de simulation développé par C. GIROD /GIR 84/, écrit dans le langage Pascal sur VAX 750, prenant en compte les RdP automodifiants interprétés. Ce programme fonctionne en mode conversationnel, à la manière d'un programme d'aide à la mise au point (Debugger). Il prévoit l'évolution du réseau dans le temps et permet de produire des statistiques de fonctionnement.

Enfin, le logiciel de simulation développé par E. Castelain /CAS 87/, écrit dans le langage Le_Lisp sur VAX 750, permet d'intégrer l'interprétation liée au graphe de commande, c'est-à-dire d'une part, les caractéristiques du procédé et d'autre part, les consignes de la partie décisionnelle.

Dans ce simulateur, le modèle de base retenu pour la partie commande est constitué des réseaux de Petri structurés adaptatifs et colorés.

La partie décisionnelle est représentée par un ensemble de règles de production, dont le but est de faciliter la maîtrise de la stratégie à adopter et le paramétrage de la commande sans modifier la structure du graphe de Petri sous-jacent.

L'intégration du procédé se fait en plusieurs niveaux. Tout d'abord, l'élaboration du graphe de commande à partir du graphe de transfert et de conditionnement, combinée à l'utilisation de la coloration permet d'intégrer naturellement, au niveau du graphe de commande, l'image opérative du procédé nécessaire à l'évolution de la commande. La modélisation proprement dite du procédé est alors rendue possible de deux manières différentes, par temporisation des places du réseau ou par utilisation du modèle spécifique basé sur l'utilisation de catégories génériques. La temporisation permet l'évaluation simple des performances du système. L'utilisation du modèle spécifique permet une description plus fine du procédé : calcul des statistiques de fonctionnement, l'étude de la

propagation des défauts, de leur détection, du traitement correspondant, etc.

D'autres solutions peuvent être proposées, à partir des outils récents mis au point par les fabricants d'automates industriels. Ces automates de haut niveau, associés à des consoles de programmation évoluées, autorisent une émulation plutôt qu'une simulation du système de commande.

Les réseaux de Petri sont utilisés dans d'autres domaines tels que :

- **L'informatique de base** : modélisation d'un système d'exploitation, représentation d'une architecture, allocation des ressources, files d'attente, systèmes répartis, etc.

- **L'intelligence artificielle** : représentation de la connaissance sous forme de réseaux de Petri et détection des contradictions en utilisant le cadre théorique et les outils logiciels développés autour des réseaux de Petri /PIP 85/

4 - MODELISATION DE LA PARTIE COMMANDE

4.1 - Introduction :

Pour résoudre le problème délicat de la modélisation de la partie commande d'un système de conduite de processus industriels, nous proposons non pas un seul outil qui ne manquerait pas d'être complexe, mais une gamme d'outils structurés en niveaux : chaque niveau supplémentaire permet d'augmenter les possibilités de représentation du modèle du niveau inférieur.

Le modèle de base utilise les réseaux de Petri structurés. Il permet une description modulaire du système de commande et prend en charge de façon sûre les interactions horizontales entre processus. La modélisation d'une hiérarchie de contrôle autorisant la flexibilité de fonctionnement du système (modes de marche, modes dégradés, etc.) nécessitera l'utilisation d'une extension du modèle précédent appelée réseaux adaptatifs. Un second niveau d'extension offre la possibilité d'identifier les classes d'objets qui circulent ou qui sont transformés sur la partie "procédé", il nécessite la coloration des marqueurs.

4.2 - Le modèle de base : les réseaux de Petri structurés

4.2.1 - Principe :

L'analyse structurée a apporté de nombreuses améliorations dans la spécification et la conception de systèmes. Elle constitue le principe de base de nombreuses méthodes de définition du cahier des charges.

La prise en compte des fondements de l'analyse et de la programmation structurées, au niveau des réseaux de Petri, constitue une approche sûre permettant d'aborder la représentation de la commande des systèmes industriels et se justifie d'autant plus que le graphe de commande conçu est destiné, dans sa finalité, à être implémenté sur les divers organes physiques de conduite du système. elle a conduit à la définition d'une classe particulière de RdP : les réseaux de Petri structurés /COR 79/.

Son intégration dans une méthodologie de spécification et de conception permet d'assurer une meilleure correspondance entre le modèle et son cahier des charges et de limiter les erreurs de conception. Le modèle RdP structuré défini en /COR 79/, /COR 83a/ s'appuie sur une décomposition fonctionnelle du système de commande en tâches élémentaires. En particulier, on s'interdit la mise en parallèle de tâches au sein d'un processus (la structure "Fork-join"). Une telle démarche préserve l'identité processus/processeur, d'où l'avantage en phase d'implantation.

4.2.2 - Méthodologie de représentation de systèmes de Processus

a) Représentaion modulaire

La méthodologie générale de définition du contrôle d'un système, qui s'appuie sur les RdP structurés, doit en premier lieu permettre une représentation modulaire du système.

La notion de module utilisée ici correspond à celle de processus, c'est-à-dire un ensemble de tâches parmi lesquelles une seule au plus peut être activée à un instant donné.

L'analyse qui permet la décomposition d'un système industriel en n processus, tient compte d'un certain nombre de critères parmi lesquels le regroupement des tâches ayant un lien fonctionnel, le niveau de parallèlisme et le temps de cycle de chaque unité fonctionnelle.

Chaque processus est alors décrit indépendamment des autres à l'aide d'un graphe de processus. Ce dernier est défini comme la combinaison séquentielle des trois structures élémentaires suivantes :

- l'action,
- l'alternative,
- la répétitive.

Notons à ce titre que la représentation modulaire a plusieurs intérêts :

1. La possibilité d'archiver les graphes de fonctionnement associés à des structures fonctionnelles utilisées à plusieurs reprises dans des applications différentes.

2. L'indépendance des graphes de fonctionnement qui implique que les modifications apportées à un graphe ne seront pas répercutées sur les autres graphes de processus.

3. La partition d'un système en sous-systèmes relativement indépendants, autorise une conception progressive du réseau de Petri structuré.

b) Définition des interactions entre les différents processus :

Les processus d'un système de production industrielle doivent coopérer. Afin de représenter les interactions entre les différents processus, trois types de liaisons primitives ont été définies /COR 79/ :

- liaison de synchronisation,
- liaison d'exclusion-mutuelle,
- liaison producteur/consommateur.

Ces interactions entre processus ne se font que par l'intermédiaire de bloc afin d'assurer une bonne construction du système de processus.

4.3 - Extensions du modèle de base

Les RdP structurés traduisent la modularité et la séquentialité des processus de la partie commande et facilitent donc la conception des grands systèmes.

Afin d'augmenter la puissance de modélisation des RdP structurés, deux extensions ont été définies : les RdP structurés adaptatifs (RdP SA) et les RdP structurés adaptatifs colorés (RdP SAC) /COR 84/, /COR 85a/.

4.3.1 - Réseaux de Petri structurés adaptatifs

Les RdP sont non déterministes et non flexibles, aussi de nombreux exemples de processus parallèles ne peuvent être modélisés par des RdP structurés. Dans ce sens, le modèle RdP peut être étendu, par exemple, en autorisant le test. A l'inverse, si l'on cherche à atteindre un objectif de validation formelle, il doit être restreint afin de limiter sa complexité lors de l'analyse des propriétés.

Une première étape dans cette direction, pourrait être la prise en compte d'un concept permettant l'augmentation notable de

la puissance d'expression et de calcul des RdP. Dans la littérature les extensions suivantes ont été proposées :

- Les arcs inhibiteurs /Hack 75a/,
- Les réseaux à priorités /Hack 75b/,
- Les réseaux automodifiants /Valk 78a/, /Valk 78b/, /Valk 81/.

Le L.A.I.I. a opté pour une extension nommée RdP structurés adaptifs qui inclut le modèle précédent.

4.3.2 - RdP structurés adaptatifs colorés

Face à la complexité croissante des systèmes de commande de processus industriels, les RdP structurés adaptatifs apportent des solutions qui, malgré leur puissance de modélisation, restent au niveau d'objets élémentaires. La lourdeur de cette approche apparaît dès l'instant où l'on prend en compte des objets faiblement distincts circulant dans la partie "procédé". Ainsi, si nous désirons distinguer au sein du marquage d'une même place des marques de types différents correspondants aux classes d'objets, les possibilités d'abréviation sont nombreuses. Une version élémentaire de réseaux à marques distinguées, encore dites colorées a été définie dans /COR 85b/.

Les définitions formelles de ces modèles sont données dans /COR 85a/, /COR 85b/.

4.4 - Conclusion :

Nous avons donc proposé dans cette partie trois classes de modèles présentés ici selon leur complexité croissante :

1. Banalisation du parallélisme et de communications.
2. Règlements de conflits (priorités), déconnexion/connexion.
3. Paramétrage.
4. Agrégation du modèle.
5. Suivi des types et de la phase d'évolution des objets pris en compte dans la partie "procédé".

CHAPITRE II

CHAPITRE II

C H A P I T R E I I

PRESENTATION DU SYSTEME DE CONCEPTION ASSISTEE DE LA COMMANDE D'UN SYSTEME DE PRODUCTION (C A P C O S)

PRESENTATION DU SYSTEME DE CONCEPTION ASSISTEE

DE LA COMMANDE D'UN SYSTEME DE PRODUCTION

(C A P C O S)

---

1. INTRODUCTION

CAPCOS est un outil d'aide à la conception et à la validation de la partie commande d'un système de production flexible. Cette partie commande est modélisée à l'aide d'un RdP structuré.

Il propose au concepteur une démarche progressive et structurée lui permettant de valider chaque étape de la modélisation et d'utiliser les résultats précédemment acquis (méta-graphe, sous-partie commande).

Dans le premier paragraphe de ce chapitre, nous présentons les éléments de base et la structure originale du système, qui

allie le traitement symbolique et le traitement numérique de l'information.

Le deuxième paragraphe sera consacré à la présentation des points forts et points faibles du système général Prolog, dans le cadre d'une application concernant la réalisation du système CAPCOS.

2. COMPOSANTS DE BASE ET ARCHITECTURE DU SYSTEME D'INFORMATION UTILISE DANS C A P C O S

2.1 - Composants de base :

Lorsqu'un expert humain cherche à résoudre un problème, il fait appel à sa compétence basée sur une connaissance approfondie du domaine concerné d'une part, et à sa capacité de raisonnement d'autre part /CHO 85/.

Cette division de comportement exprime la nécessité de représenter les connaissances séparément du programme qui les utilise /BON 84/.

Dans cette perspective, le système CAPCOS se compose essentiellement de deux parties, portant respectivement sur le traitement symbolique et le traitement numérique de l'information (voir figure II-1).

2.1.1 - Traitement symbolique

Le système de traitement symbolique est constitué de trois éléments :

(i) En premier lieu, la base de connaissances contenant l'ensemble des informations spécifiques aux outils de contrôle basés sur le modèle RdP structuré et adaptatif.

(ii) En second lieu, la base de faits peut jouer un double rôle. D'une part, en tant que mémoire de travail, elle contient l'ensemble de données propres au problème à traiter, et d'autre part, en tant que mémoire auxiliaire, elle mémorise tous les résultats intermédiaires en conservant une trace des raisonnements effectués. Elle peut donc être utilisée à la fois pour expliquer l'origine des informations déduites au cours d'une session et pour décrire le comportement du système.

(iii) Enfin, le moteur d'inférence utilise les méthodes et les heuristiques contenues dans la base de connaissances pour résoudre le problème spécifié par les données contenues dans la base de faits. Le moteur d'inférence sélectionne, valide et déclenche certaines règles afin d'arriver à la solution du problème posé.

2.1.2 - Traitement numérique

Le système de traitement symbolique est enrichi à la fois par un ensemble de modules de traitement numérique et par un ensemble d'interfaces tels que :

- Interface avec le module de validation des propriétés du RdP structuré ;

- Interface avec le logiciel de simulation ;

- Interface utilisateur ;

facilitant ainsi sa mise en oeuvre et son intégration à un ensemble plus complet de logiciels.

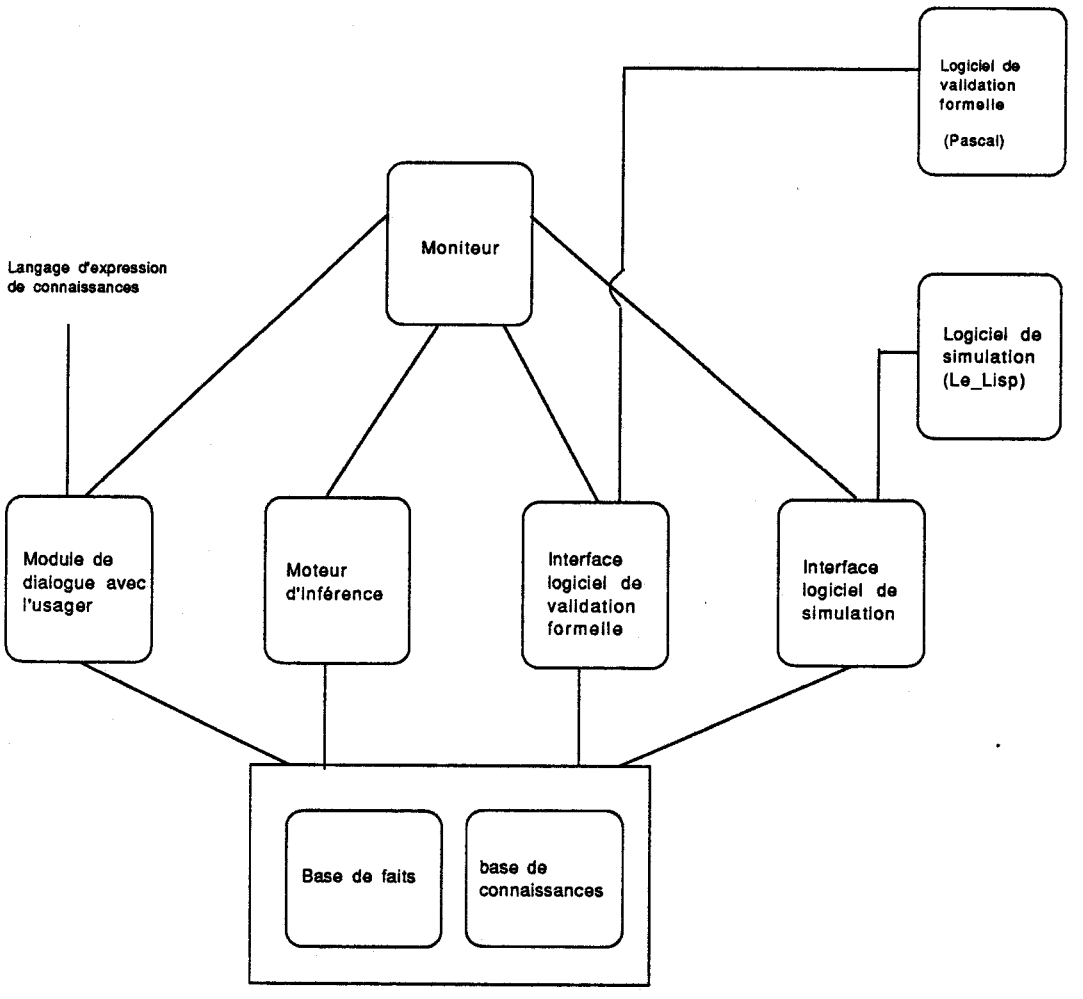


Figure II - 1 : Structure générale du système

2.2 - Architecture (voir Figure II-1)

De par son architecture, le système CAPCOS tente de répondre à quelques objectifs généraux :

1. **Il assiste tout d'abord l'expression et l'édition des connaissances** en facilitant l'expression la plus directe possible des règles, par l'utilisation d'un langage d'expression de connaissances proche de celui du concepteur. Il s'agit ici du langage de spécification basé sur l'utilisation des RdP structurés.

2. **Il exploite l'ensemble des connaissances** en combinant et/ou chaînant des groupes de règles pour inférer de nouvelles connaissances, telles la génération automatique d'objets d'une classe représentant un module fonctionnel et ses propriétés.

3. **Il exploite aisément la mise à jour de l'ensemble des connaissances**, en offrant des facilités pour les adjonctions et suppressions de données et de règles, en utilisant le module de gestion de la base de connaissances.

3. CHOIX DU SYSTEME PROLOG

Le marché du logiciel comporte à ce jour un important échantillon de systèmes experts (voir les comptes rendus de réunions telles que les journées internationales d'Avignon /ADI 85/, /ADI 86/). Ces derniers peuvent être classés en deux grandes catégories :

1. Les systèmes experts spécialisés relatifs à des domaines variés d'applications ;

2. Les systèmes experts essentiels ou généraux. Un système général réunit un moteur d'inférence, un langage d'expression de connaissances et des structures et conventions de représentation de ces connaissances /FAR 85/. Un système général ne contient pas de connaissances spécifiques d'un domaine particulier.

S'il ne faut pas exclure a priori la réalisation d'un système expert spécifique dans le cadre d'une application donnée, il semble que le véritable poroblème qui se pose au concepteur d'un système expert soit le plus souvent celui du choix du logiciel le plus approprié aux caractéristiques de l'application développée.

De fait, les critères de choix d'un système général sont nombreux, l'importance relative de chacun d'eux varie en fonction des spécificités de l'application choisie.

A cette étape de l'analyse, nous pouvons considérer deux solutions possibles qui permettent la mise en oeuvre de la méthodologie de conception et de validation de la partie commande d'un système de conduite de procédés industriels s'appuyant sur le modèle RdP structuré.

La première solution consiste à choisir un programme commercialisé qui répond à nos besoins. Or, actuellement, il n'existe pas de tel programme opérationnel sur le marché des logiciels experts. Par conséquent, nous sommes amenés à envisager la deuxième solution qui consiste à écrire ce logiciel en utilisant un système plus général dans le cadre des outils de l'intelligence artificielle.

De ce point de vue, l'aptitude d'un système général à pouvoir fournir des outils de base permettant de représenter, de gérer et manipuler aisément les réseaux de Petri structurés nous est apparu comme critères essentiels du choix. Dans cette perspective, plusieurs aspects nous ont conduit à utiliser le système Prolog (/COL 83/, /GIA 85/, /CON 86/, /CLO 84/).

3.1 - Prolog : langage de spécification

Dans cette approche, la composante traitement symbolique du système CAPCOS est programmée directement en Prolog. Les connaissances sont représentées dans la logique des prédicats du premier ordre. Le raisonnement mis en oeuvre est celui fourni par

le langage lui-même ; avec la possibilité de programmer d'autres processus de déduction en utilisant, d'une part l'algorithme général d'unification et d'autre part la gestion automatique de l'arborescence de recherche avec retour-arrière (back tracking).

3.2 - Prolog : langage d'implémentation

Il est utilisé ici comme un langage de programmation. Cela conduit à définir un formalisme de représentation et d'exploitation des connaissances, et à l'implémenter en Prolog. Cette méthode présente l'avantage de permettre une implémentation très rapide en raison de la puissance et de la modularité de Prolog.

3.3 - Extensions de Prolog

Ce troisième aspect consiste à prendre acte des insuffisances de Prolog, notammant concernant le calcul scientifique, et à définir une extension du langage offrant des fonctionnalités nécessaires à la réalisation du système d'aide à la conception et à la validation des systèmes de production industrielle. Cette extension consiste, par exemple, à intégrer de nouvelles fonctions programmées dans le langage Pascal.

Néanmoins, Prolog (Prolog II v1.0) possède de nombreuses limites : celles qui sont liées à des choix théoriques initiaux et celles qui sont dues à la relative jeunesse du langage.

En effet, la structure de contrôle utilisée en Prolog est minimale, fixe et souffre de l'absence du mécanisme de retour-arrière intelligent limitant la combinatoire. En conséquence, on a recours non seulement au coupe-choix (cut), qui est déjà une entorse à la philosophie initiale, mais encore à des méta-règles pour exprimer certaines stratégies de contrôle d'une manière explicite.

Enfin, il est difficile d'exprimer sous forme clausale des traitements fondamentalement algorithmiques comme les calculs. De plus, on ne peut pas réutiliser des bibliothèques de programmes existants. L'introduction de l'attachement procédural, ou possibilité d'invocation de procédures dans une règle est parfois très utile pour augmenter les fonctionnalités du système.

Parmi les limites que nous associons à la relative jeunesse du langage, on trouve essentiellement la pauvreté du mécanisme d'entrées/sorties, la pauvreté d'aide à la mise au point et l'impossibilité d'utiliser les services du système d'exploitation hôte. Ces limites justifient l'intérêt de nombreux projets d'environnement de Programmation Logique sophistiqués.

4 - CONCLUSION

Nous avons présenté dans ce chapitre, les éléments de base du système d'aide à la conception et à la validation de la partie

commande d'un système de processus industriels (CAPCOS) et son architecture.

Dans un premier temps, nous avons distingué deux parties essentielles du système. L'une concerne le traitement symbolique et bénéficie par là de l'apport de l'intelligence artificielle dans le domaine de la conception assistée et notamment la technique Système Expert. L'autre porte sur le traitement numérique, qui comble l'insuffisance du système symbolique dans le domaine de calculs ; elle est constituée d'un ensemble de modules écrits dans le langage de programmation Pascal.

Dans le deuxième temps, nous avons exposé un ensemble de points portant sur le choix d'un logiciel qui doit satisfaire nos besoins. Nous nous sommes orientés vers le choix d'un système général, en l'occurence le langage Prolog, qui fournit un mécanisme d'exploitation de la connaissance.

Nous allons présenter dans le chapitre qui suit, la structure de la base de connaissances exploitée par Prolog.

CHAPITRE III

CHAPITRE III

C H A P I T R E I I I

STRUCTURE DE LA BASE DE CONNAISSANCES

STRUCTURE DE LA BASE DE CONNAISSANCES

1 - INTRODUCTION

Lors du processus de conception d'un système de contrôle de procédés industriels, le concepteur peut faire appel a priori à un ensemble de solutions déjà synthétisées. De ce fait, le système CAPCOS doit pouvoir restituer et/ou archiver les graphes de contrôle et les propriétés de ces solutions étudiées.

La caractéristique de conservation de modèles pré-étudiés nous conduit à scinder la base de connaissances du système CAPCOS en une base de données et une base de règles /KAR 85/, /KAR 86/, schématisées par la figure III-1.

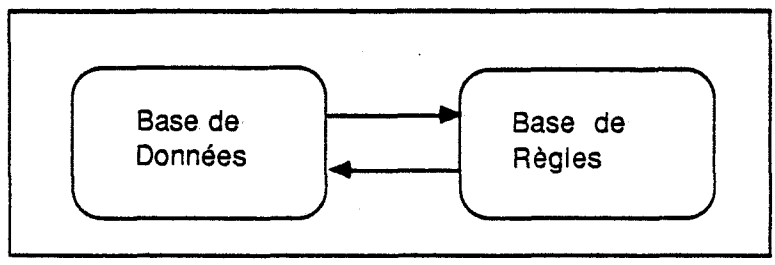


Figure III - 1 : Structure de la base de connaissances

Cette division de la base de connaissances, due à cette particularité de notre application, amène une différence de gestion entre les données et les règles. Ces dernières, dont la constitution représente la quasi-totalité du travail du réalisateur du programme expert, conduisent à la définition d'une base de règles relativement stable. La base de données, à l'inverse, doit être évolutive : on doit pouvoir consulter, insérer, modifier ou supprimer des données avec souplesse et transparence.

Malgré la distinction faite entre base de données et base de règles, l'homogénéité est conservée par l'utilisation d'un même modèle de représentation de la connaissance.

2 - MODELE DES CONNAISSANCES

La représentation de la connaissance reste un domaine clé de l'intelligence artificielle. Elle a pour but de développer des schémas pour l'incorporation de la connaissance du monde extérieur dans les systèmes de raisonnement /BAR 86/, /LAU 82/.

S'il n'existe pas, à ce jour, un langage universel de programmation, nous ne disposons pas encore d'un formalisme idéal

pour représenter les connaissances d'un système d'intelligence artificielle. Ce manque d'universalité explique la multiplicité des techniques et théories de représentation de la connaissance (logique, représentation procédurale, les systèmes de production, les réseaux sémantiques, les objets structurés, etc.) /BAR 86/, /BON 84/, /LAU 86/.

Du fait du choix du système Prolog pour réaliser notre application, le modèle des connaissances utilisé est basé naturellement sur les règles de production. Elles représentent un formalisme très général de représentation de la connaissance, et elles expriment par définition une connaissance factuelle, de manière purement déclarative, indépendantes des programmes d'utilisation, données globalement de façon totalement modulaire.

Une règle de production est une expression de la forme /LAU 86/ :

« situation ---> conclusion »

ou encore,

si	condition	alors	action
	hypothèse		conclusion

Ce qui signifie que chaque fois qu'une situation, représentée dans un formalisme approprié est reconnue (partie gauche de la règle), l'action est exécutée (partie droite de la règle).

Une règle de production, sous un formalisme un peu différent, contient la logique des prédicats de premier ordre.

A ce titre, les règles de production peuvent être de deux types :

1. Le premier est fondé sur la logique propositionnelle. Elle se définit :

- En premier lieu, par sa syntaxe qui régit l'ensemble des assertions exprimables dans le langage. Toute assertion proprement formulée, appelée proposition, est affectée de l'une des deux valeurs possibles vrai ou fausse ;

- En deuxième lieu, par ses règles d'inférence qui décrivent comment créer de nouvelles assertions à partir d'anciennes ;

- Et en dernier lieu, il est possible de représenter des propositions plus complexes en utilisant des connecteurs logiques [(et,$\wedge$), (ou,v), (non,$\neg$), (implique,$\Rightarrow$), (équivalent,$\equiv$) pour combiner de simples propositions.

Dans cette logique, les règles ne contiennent pas de variables et chaque fait ou proposition est retenu comme tel.

2. Le second formalisme est fondé sur la logique des prédicats de premier ordre. Le calcul de propositions est insuffisant pour

exprimer de nombreuses assertions utiles en intelligence artificielle. Dans l'intention de capturer, dans un formalisme adéquat, des connaissances à propos d'objets du monde réel, on a besoin non seulement d'être capable d'exprimer de vraies ou de fausses propositions, mais aussi d'être capable de particulariser les objets, de postuler des relations entre ces objets, et de généraliser ces relations sur des classes d'objets.

Le calcul des prédicats est une extension du calcul propositionnel. Il comporte en plus des notions de prédicats et symboles de quantification universelle et existentielle. Son intérêt principal par rapport au calcul propositionnel est l'introduction de la notion de variable.

Ainsi, le premier formalisme de représentation de la connaissance (logique d'ordre 0) est adapté à la représentation des entités et des objets dans la base de données du système CAPCOS ; tandis que le second formalisme (logique d'ordre 1) correspond d'une part à l'expression des règles de gestion des objets de la base de données, et d'autre part à l'expression des règles de déductions faites sur ces objets.

Ce type de représentation de la connaissance s'exprime aisément dans le formalisme Prolog. Ce dernier est caractérisé par l'uniformité de représentaion : faits, règles d'inférences, procédures complexes, structures de contrôle, etc., sont tous exprimés sous la forme unique de clause de Horn /COL 83/, /GIA 85/.

3 - DESCRIPTION DE LA BASE DE DONNEES

3.1 - Introduction :

La base de données est formée d'une collection d'objets. Un objet, décrit indépendamment des autres, représente l'ensemble des caractéristiques relatives à un réseau de Petri structuré. Ces caractéristiques peuvent être classées en trois catégories /KAR 85/, /KAR 86/.

Tout d'abord, les données structurales qui déterminent le schéma syntaxique des processus et leurs liaisons. Ensuite les données représentant les propriétés déduites ou issues de la validation syntaxique du modèle. Enfin, les données documentaires.

Pour l'implantation interne de ces différentes classes de données, nous sommes amenés à définir en premier lieu une structure de données les représentant. La construction d'une telle structure se fait en combinant judicieusement, pour une meilleure gestion, les structures de données de base de Prolog. En effet, Prolog offre des outils de base très puissants qui permettent d'élaborer et de manipuler des structures conceptuelles complexes. Nous présentons ensuite, les différents aspects de la représentation d'un RdP structuré dans la base de données.

3.2 - Structures de données de base : Arbres et Listes

(voir définition dans Annexe A).

3.3 - Description structurale d'un RdP structuré

Comme nous l'avons vu précédemment (chapitre I, § 4.2), un système de processus structurés se compose :

- de n processus : chacun d'eux décrit le contrôle d'un mécanisme physique déduit de la décomposition fonctionnelle du système étudié.

- d'un graphe de liaison traduisant les relations internes existantes entre les n processus.

La description structurale consiste à reproduire fidèlement dans la base de données, par le biais d'une structure de données appropriée, la structure générale du système de processus structurés.

Elle est spécifiée par trois types de faits ou relations définissant chacun un type d'informations.

3.3.1 - Relation "graphe"

Cette première relation de nom "graphe" renferme les caractéristiques des différents processus du système structuré. Elle possède un ensemble d'arguments portant sur :

i) l'identification du RdP structuré,

ii) la description syntaxique de ce réseau,

iii) le nombre de ses processus.

Le schéma de cette relation s'écrit :

graphe (identification du RdP structuré,
description syntaxique de réseau,
nombre de ses processus).

Description syntaxique du RdP structuré

Un RdP structuré est représenté dans la relation "graphe" par une liste non ordonnée d'arbres. Chaque arbre décrit un processus.

Cette liste s'écrit :

processus - 1 . processus - 2 . ... processus - n . nil

Chaque processus est alors défini indépendamment des autres par un graphe de processus.

Description d'un processus

Un processus est défini comme un enchaînement séquentiel de tâches et peut être décrit au moyen des trois structures de base : l'action, l'alternative, et la répétitive.

L'arbre associé à un processus, dans la base de données, est représenté par un n-uplet dont la structure est :

<identification du processus, graphe du processus,
nombre de ses places, nombre de ses transitions>

Graphe d'un processus

Le graphe d'un processus est complétement spécifié par une liste de triplets. Chaque triplet caractérise une transition, ses places d'entrées, ses places de sorties et s'écrit :

<liste des places d'entrées,transition,liste des places de sorties>

Une liste des places a la forme suivante :

el1 . el2 . ... . elk . nil

où un élément eli (i = 1, ..., k) est un doublet de la forme :

<pl , i>

dans lequel i est le numéro de la place "pl" dans le processus.

L'argument "transition" est représenté par le doublet :

<tr, j>

où j est le numéro de la transition "tr" dans le processus.

Exemple de description d'un processus

Considérons le schéma de processus suivant (Figure III-2) :

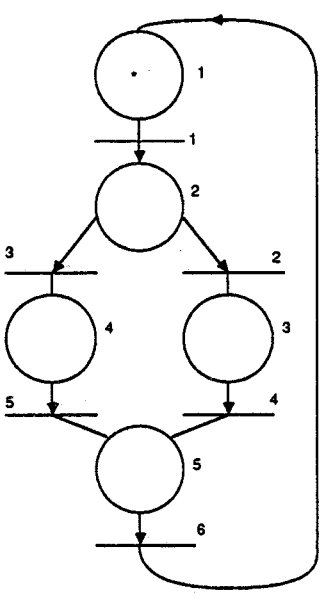


Figure III - 2 : Exemple d'un graphe de processus

La description qui lui est associée dans la base de données est :

<processus-exemple,

<<pl,1>.nil,<tr,1>,<pl,2>.nil>.

<<pl,2>.nil,<tr,2>,<pl,3>.nil>.

<<pl,3>.nil,<tr,3>,<pl,5>.nil>.

<<pl,2>.nil,<tr,4>,<pl,4>.nil>.

<<pl,4>.nil,<tr,5>,<pl,5>.nil>.

<<pl,5>.nil,<tr,6>,<pl,1>.nil>.nil,5,6>

3.3.2 - Relation "liaison" :

La prise en considération du parallélisme d'un système de production flexible, se traduit au niveau du modèle par l'introduction de plusieurs processus. L'évolution de chacun d'eux tient compte, en règle générale, de l'état des autres processus. Il est donc nécessaire de pouvoir représenter leurs relations par des liaisons cohérentes. Ces interactions permettent de définir le graphe de liaison, dont les extrémités appartiennent aux différents graphes de processus.

La représentation du graphe de liaison d'un RdP structuré dans la base de données se fait à travers la relation "liaison".

Elle contient un ensemble d'informations et d'indications portant sur les modes de composition et d'interaction entre processus.

Ses arguments sont :

i) identification du RdP structuré,

ii) description du graphe de liaison.

Son schéma relationnel est :

liaison (identification du RdP structuré, graphe de liaison)

Graphe de liaison

La structure de données choisie pour décrire le graphe de liaison est la structure de liste, dont chaque composant est un arbre qui caractérise une seule interaction entre processus. Cette liste s'écrit :

liaisons-1 . liaison-2 . ... . liaison-1 . nil

où liaison-i (i = 1, ..., 1) est un n-uplet de la forme :

<type de l'interaction, arguments de l'interaction>

Définitions des types de liaisons et structures de données associées

Les liaisons représentent les communications entre les divers processus. Elles sont de trois types :

i) liaison de synchronisation,

ii) liaison d'exclusion mutuelle,

iii) liaison producteur/consommateur.

i) Liaison de synchronisation

Elle permet d'indiquer dans quel ordre des opérations appartenant à des processus différents vont s'enchaîner. Elle est de deux types :

a) Rendez-vous avec accusé de réception, dont la structure de données associée est :

<sigacc, identification de l'émetteur, identification du récepteur>

b) Signal mémorisé sans accusé de réception, qui est représenté par :

<signal, identification de l'émetteur, identification du récepteur>

ii) Liaison d'exclusion mutuelle

Elle exprime le fait que les processus, dans leur déroulement respectif, nécessitent l'utilisation d'une ressource à accès exclusif.

La structure de données qui la représente est :

<sema, identification de la ressource, nombre d'exemplaires ou marques, liste d'utilisateurs de la ressource>

iii) Liaison producteur/consommateur

Elle permet à un ensemble de processus d'utiliser une ressource qui peut prendre deux états. Chacun de ces processus acquiert une ressource dans un état, et la restitue dans l'autre état. Parmi ces processus, il existe au moins un processus qui l'utilise dans un état et un processus qui l'utilise dans l'autre état.

Le n-uplet décrivant cette liaison est :

<pc, identification de la ressource, nombre d'exemplaires (marques) de la ressource dans l'état producteur, nombre de marques dans l'état consommateur, liste des processus producteurs, liste des processus consommateurs>

3.3.3 - Relation "référence"

Cette dernière relation de la description structurale associe à un RdP structuré un ensemble d'informations complémentaires sur la structure interne des blocs utilisés pour définir les interactions entre processus.

La notion de bloc a été introduite pour permettre tout d'abord de regrouper un enchaînement séquentiel de tâches et de lui associer un nom. Elle est également utilisée pour définir les sommets d'entrées et les sommets de sorties du graphe de liaison. On dira en effet que le bloc A du processus P1 synchronise le bloc B du processus P2, ou encore le bloc A de P1 et le bloc B de P2 se partagent en exclusion mutuelle la ressource R.

Les arguments de la relation "référence" sont :

i) identification du RdP structuré,

ii) table de correspondance.

Le schéma de cette relation s'écrit :

référence (identification du RdP structuré, table de correspondances)

Description de la table de correspondance

L'intérêt de la table de correspondances est de déterminer, pour un processus donné, l'ensemble de ses blocs. Elle est décrite à l'aide d'une liste dont le nombre d'éléments est identique au nombre de processus ayant la structure de blocs.

Cette liste s'écrit :

correspondance-1 . correspondance-2 . ... . correspondance-m . nil

où correspondance-i (i=1, ... , m) est un n-uplet de la forme suivante :

<identification du processus, caractéristiques de ses blocs>

Le deuxième argument de ce n-uplet est une liste, dont chaque élément caractérise un bloc.

Un bloc appartenant à un processus est représenté par un triplet, qui contient un point d'entrée et son point de sortie. Il s'écrit :

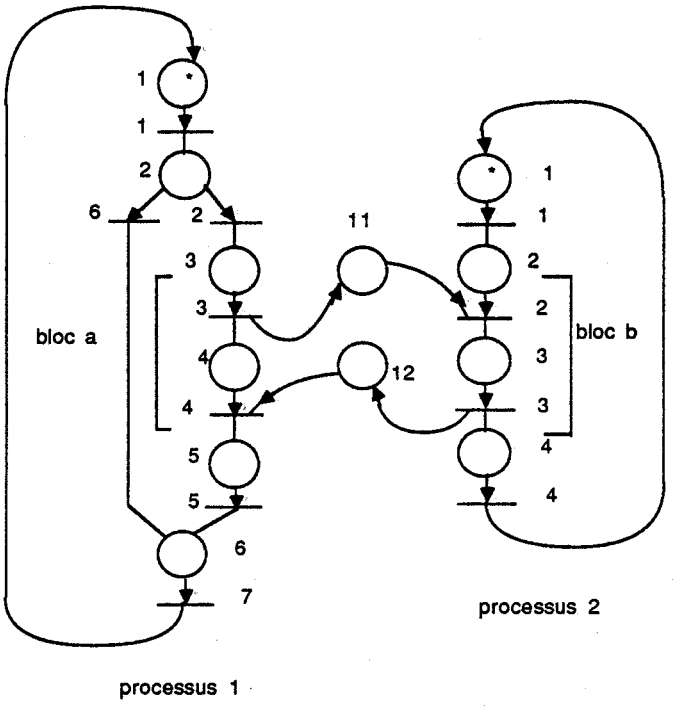
<identification du bloc, point d'entrée, point de sortie>

3.3.4 - Exemples

Nous allons présenter dans ce paragraphe, trois exemples de représentation d'un RdP structuré formé de deux processus reliés entre eux par les trois types de communication.

Remarque : pour une raison de commodité qui sera expliquée ultérieurement, nous avons comptabilisé les places de liaison au nombre total de places de processus.

i) Liaison de synchronisation de type "rendez-vous avec accusé de réception"



graphe (RdP-exemple-1,

<processus-1,<<pl,1>.nil,<tr,1>,<pl,2>,nil>.

<<pl,2>.nil,<tr,2>,<pl,3>.nil>.

<<pl,3>.nil,<tr,3>,<pl,4>.<pl,11>.nil>.

<<pl,4>.<pl,12>.nil,<tr,4>,<pl,5>.nil>.

<<pl,5>.nil,<tr,5>,<pl,6>.nil>.

<<pl,2>.nil,<tr,6>,<pl,6>.nil>.

<<pl,6>.nil,<tr,7>,<pl,1>.nil>.nil,8,7>.

<processus-2,<<pl,1>.nil,<tr,1>,<pl,2>.nil>.

<<pl,2>.<pl,11>.nil,<tr,2>,<pl,3>.nil>.

<<pl,3>.nil,<tr,3>,<pl,4>.<pl,12>.nil>.

<<pl,4>.nil,<tr,4>,<pl,1>.nil>.nil,6,4>.nil,

référence (Rd-exemple-1,

<processus-1,<bloc-a,3,4>.nil>.

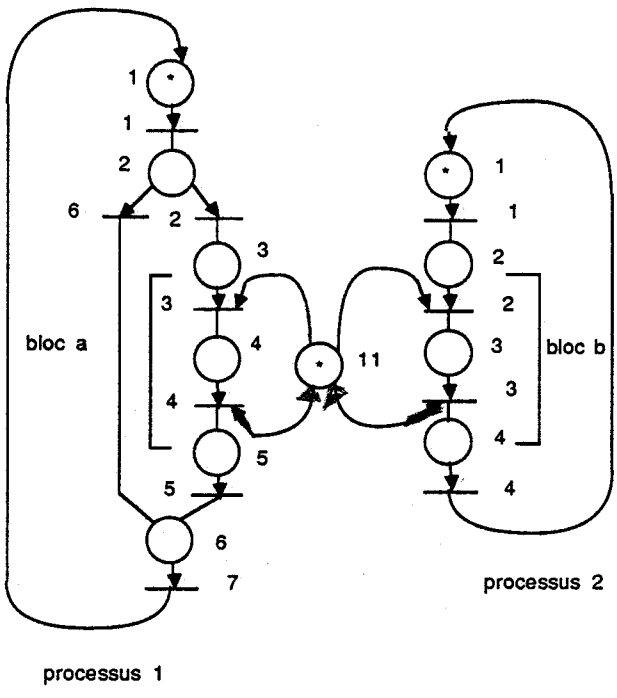
<processus-2,<bloc-b,2,3>.nil>.nil)

liaison (RdP-exemple-1,

(sigacc,<11,0>.<12,0>,<processus-1,bloc-a>,<processus-2,bloc-b>>.

nil)

ii) Liaison de type exclusion mutuelle



graphe (RdP-exemple-2,

<processus-1,<<pl,1>.nil,<tr,1>,<pl,2>.nil>.
<<pl,2>.nil,<tr,2>,<pl,3>.nil>.
<<pl,3>.<pl,11>.nil,<tr,3<,<pl,4>.nil>.
<<pl,4>.nil,<tr,4>,<pl,5>.<pl,11>.nil>.
<<pl,5>.nil,<tr,5>,<pl,6>.nil>.
<<pl,2>.nil,<tr,6>,<pl,6>.nil>.
<<pl,6>.nil,<tr,7>,<pl,1>.nil>.nil,7,7>.

<processus-2,<<pl,1>.nil,<tr,1>,<pl,2>.nil>.
<<pl,2>.<pl,11>.nil,<tr,2>,<pl,3>.nil>.
<<pl,3>.nil,<tr,3>,<pl,4>.<pl,11>.nil>.
<<pl,4>.nil,<tr,4>,<pl,1>.nil>.nil,5,4>.nil,

référence (Rd-exemple-2,

<processus-1,<bloc-a,3,4>.nil>.

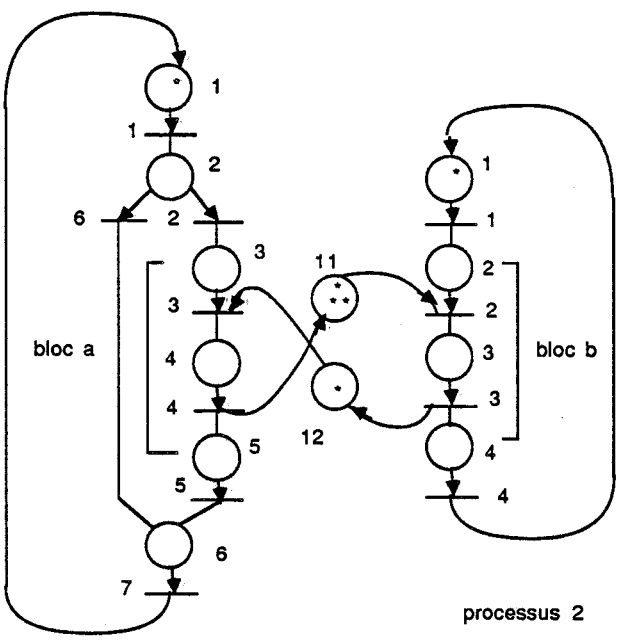
<processus-2,<bloc-b,2,3>.nil>.nil)

liaison (RdP-exemple-2,

<sema,ressource-exclusif,<11,1>,

<processus-1,bloc-a>,<processus-2,bloc-b>.nil>.nil)

iii) Liaison de type producteur / consommateur



graphe (RdP-exemple-3,

<processus-1,<<pl,1>.nil,<tr,1>,<pl,2>.nil>.
<<pl,2>.nil,<tr,2>,<pl,3>.nil>.
<<pl,3>.<pl,12>.nil,<tr,3>,<pl,4>.nil>.
<<pl,4>.nil,<tr,4>,<pl,5>.<pl,11>.nil>.
<<pl,5>.nil,<tr,5>,<pl,6>.nil>.
<<pl,2>.nil,<tr,6>,<pl,6>.nil>.
<<pl,6>.nil,<tr,7>,<pl,1>.nil>.nil,8,7>.

<processus-2,<<pl,1>.nil,<tr,1>,<pl,2>.nil>.
<<pl,2>.<pl,11>.nil,<tr,2>,<pl,3>.nil>.
<<pl,3>.nil,<tr,3>,<pl,4>.<pl,12>.nil>.
<<pl,4>.nil,<tr,4>,<pl,1>.nil>.nil,6,4>.nil,

référence (RdP-exemple-3,

<processus-1,<bloc-a,3,4>.nil>.

<processus-2,<bloc-b,2,3>.nil>.nil)

liaison (RdP-exemple-3,

<pc,ressource-pc,<11,3>.<12,1>,<<processus-1,bloc-a>.nil>,
<<processus-2,bloc-b>.nil>>.nil)

3.3.5 - Conclusion

La description structurale permet ainsi de définir complétement, d'une part les graphes de processus et leurs blocs et d'autre part le graphe de liaison qui forment globalement un RdP structuré.

3.4 - Propriétés du RdP structuré :

La prise en compte des propriétés d'un RdP structuré au niveau de la base de données sera expliquée ultérieurement (chapitre IV, § 4.3.7)

3.5 - Documentation concernant un RdP structuré :

Dans le but de disposer d'une documentation sur un RdP structuré modélisant un graphe de commande archivé dans la base de données, nous avons créé une relation de nom "commentaire", qui associe à chaque RdP structuré un ensemble d'informations, de remarques ou de mode d'emploi.

Elle possède les arguments suivants :

i) identification du RdP structuré,

ii) documentation.

Son schéma relationnel est :

commentaire (identification du RdP structuré, documentation)

4 - CONCLUSION

Dans ce chapitre, nous avons présenté la structure générale de la base de connaissances du système de conception assistée du graphe de commande. Dans ce sens, compte tenu de la particularité de notre application, nous avons décomposé la base de connaissances en une base de données et une base de règles, dont le modèle de représentation est basé sur les règles de production.

Cette démarche paraît particulièrement adaptée à la méthodologie de conception de la partie commande d'un système de production industrielle. En effet, la base de données, définie comme un ensemble structuré de connaissances permanentes et factuelles, collectionne à titre d'expertise un ensemble d'objets. Chaque objet, décrit à l'aide des cinq relations dont quatre ont été détaillées en donnant leurs schémas relationnels, représente soit une partie commande, soit un méta-graphe ou une sous-partie commande modélisés par un RdP structuré et validé.

La base de règles, quant à elle, gère la base de données en offrant les fonctions classiques d'un système de gestion de base de données. Elle permet d'établir le dialogue avec l'utilisateur et de mettre en oeuvre le mécanisme de déduction qui modélise la démarche du concepteur. Nous consacrerons justement le chapitre suivant à l'étude détaillée de la structure et fonctionnalités de la base de règles.

CHAPITRE IV

CHAPITRE IV

C H A P I T R E I V

SYSTEME DE GESTION DE LA BASE DE CONNAISSANCES

SYSTEME DE GESTION DE LA BASE DE CONNAISSANCES

1. INTRODUCTION

Nous présentons, dans ce chapitre, la base de règles du système d'aide à la conception et à la validation d'un graphe de commande. Elle est formée par un ensemble de règles de production et constitue la composante active de la base de connaissances. En ce sens, elle met en oeuvre un ensemble d'outils logiciels permettant au concepteur d'aborder par étapes le problème délicat que constitue la modélisation de la partie commande d'un système de conduite de processus à haut degré de parallélisme.

Ces outils constituent le système de gestion de la base de connaissances. C'est un logiciel multimodulaire comprenant un module de gestion de la base de données, un module d'aide à la conception et à la validation du graphe de commande, un module de gestion de la base de règles et un module superviseur assurant

l'interconnexion entre les différents éléments du système et d'interface avec l'utilisateur.

Nous présentons successivement :

- l'architecture du système de gestion de la base de connaissances ;
- les caractéristiques des différents modules constituant ce système.

2 - ARCHITECTURE

Le système de gestion de la base de connaissances propose au concepteur une démarche progressive lui permettant de valider chaque étape de la modélisation et d'utiliser des résultats précédemment acquis. Cette démarche est basée d'une part sur la mise en oeuvre de la méthodologie de description des systèmes de processus (/COR 79/, /VER 82/), et d'autre part sur l'utilisation de la base de données /KAR 86/. Ceci débouche sur la décomposition du système de gestion en un ensemble de modules donnant la configuration représentée sur la figure IV-1.

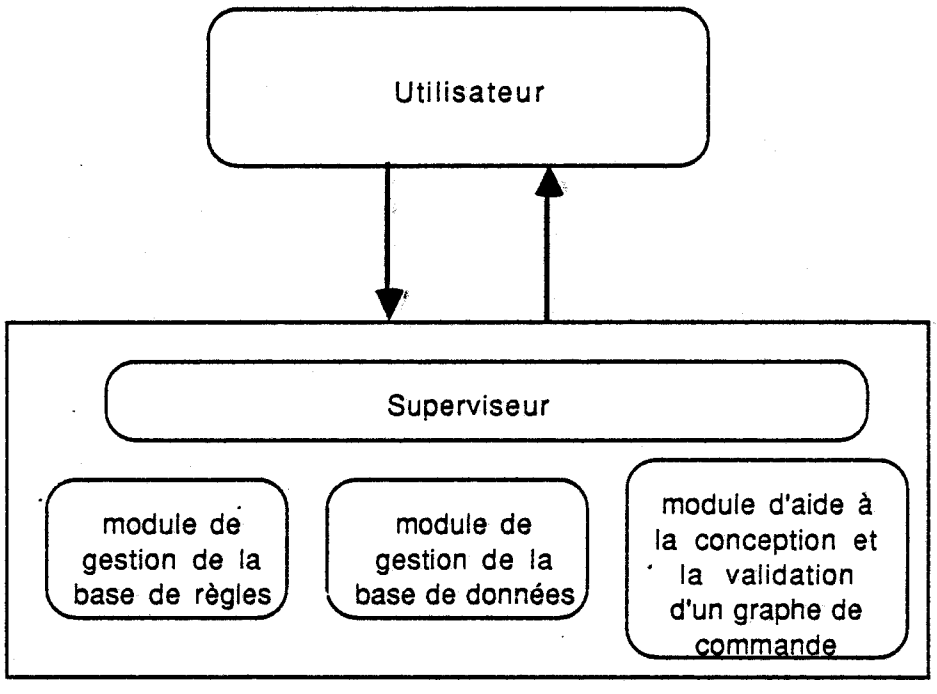


Figure IV - 1 : Configuration générale du système de gestion de la base de connaissances

Les quatre modules principaux de ce système sont :

i) le gestionnaire de la base de données ;

ii) le module d'aide à la construction et à la validation du graphe de commande ;

iii) le module de gestion de la base de règles ;

iv) le superviseur qui assure l'interface entre les trois premiers modules et avec l'utilisateur.

Le module de gestion de la base de règles effectue des traitements portant sur l'ajout, la suppression et la modification de règles d'inférences. Ce module utilise actuellement l'éditeur de règles du système Prolog.

Nous nous sommes intéressés plus particulièrement à l'étude des autres modules qui constituent des composantes fortement liées. Leur réalisation représente une phase importante dans l'élaboration du système CAPCOS. A cet égard, compte tenu du nombre important de ses fonctionnalités et de l'importance de sa taille, nous l'avons décomposé en un ensemble de méta-fonctions et avons dégagé les liens entre celles-ci. Sa structure générale est décrite par un graphe de macro-états représentée sur la figure IV-2.

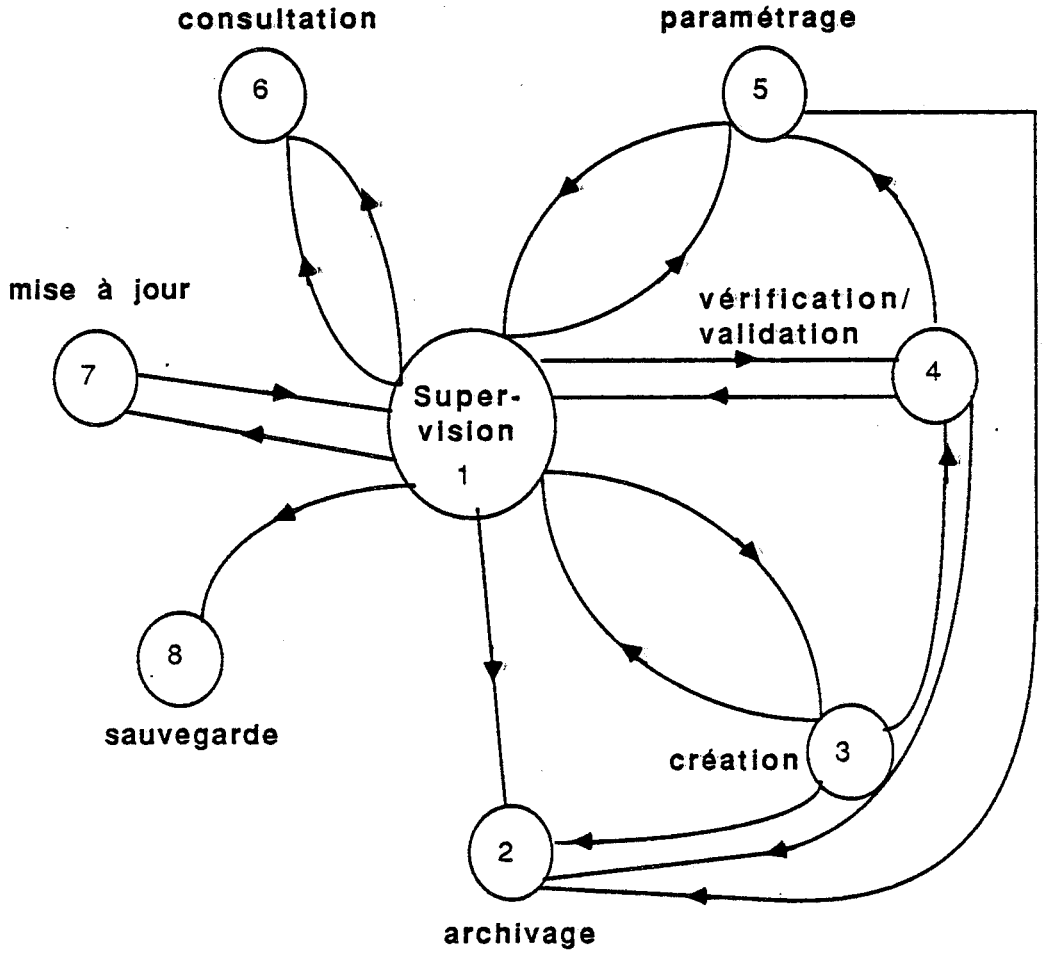


Figure IV - 2 : Décomposition des modules en méta-fonctions

Dans cette architecture (figure IV-2), on distingue six principales méta-fonctions. Elles permettent au concepteur de :

i) **créer un nouvel objet** en spécifiant la partie commande du système de production étudié ;

ii) **vérifier et valider** les spécifications ;

iii) préciser **l'interfaçage** de la partie commande avec les parties "procédés" et décisionnelle **(Paramétrage)** ;

iv et v) **consulter et mettre à jour** la base de données ;

vi) **enrichir la base de données** en archivant les résultats intermédiaires et/ou le résultat final issus des différentes méta-fonctions.

La méta-fonction "sauvegarde" a pour but d'assurer le transfert mémoire secondaire / mémoire active.

3 - PRESENTATION D'EXEMPLE D'APPLICATION

Afin d'illustrer notre approche à travers les fonctionnalités logicielles proposées, nous allons présenter l'exemple de cellule flexible de production schématisée sur la figure IV-3 /BOU 86/.

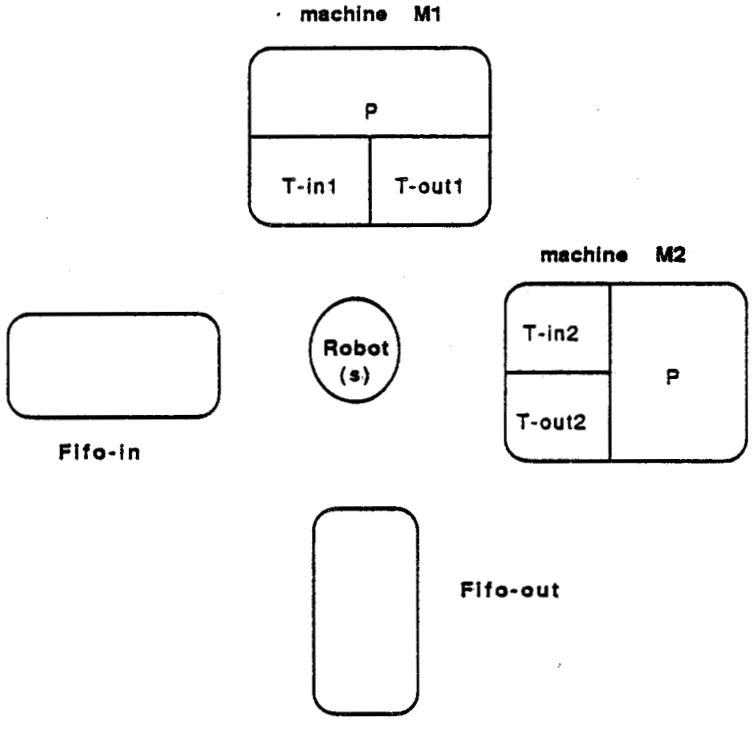


Figure IV - 3 : Exemple de cellule flexible

Cette cellule est composée de :

i) deux tables de transfert FIFO-IN et FIFO-OUT : FIFO-IN pour l'entrée de pièces et FIFO-OUT pour la sortie ;

ii) deux machines M1 et M2. Chaque machine est autonome et dispose d'un poste d'usinage P et de deux bancs de transfert : T-IN pour le chargement et T-OUT pour le déchargement.

iii) de 1 ou 2 robots qui assurent le transfert des pièces entre les différents éléments constituant la cellule.

Dans cette cellule, circulent quatre types de pièces. Ces pièces sont usinées :

- soit sur la station M1,
- soit sur la station M2,
- soit sur la station M1 puis la station M2,
- soit sur la station M2 puis la station M1.

A chaque pièce correspond son programme d'usinage.

La démarche que nous proposons par la suite consiste à présenter les méta-fonctions de chaque module du système de gestion de la base de connaissances, en illustrant leur mise en oeuvre dans le cadre de cette application.

4 - PREMIER MODULE : AIDE A LA CONSTRUCTION ET A LA VALIDATION DU GRAPHE DE COMMANDE D'UN SYSTEME DE CONDUITE DE PROCEDES INDUSTRIELS

4.1 - Introduction

La méthodologie de spécificaion et de conception décrite dans /COR 79/ et /VER 82/ permet de faire apparaître les différentes

unités fonctionnelles du système de conduite de procédés industriels. Elle organise ces unités dans un système de processus en respectant un certain nombre de critères comme :

- le parallélisme d'action,
- la cohérence fonctionnelle,
- la minimisation du nombre de processus.

Dans la phase de conception, lorsque la décomposition en processus a été réalisée, la modélisation s'effectue en deux étapes :

- la description de chaque processus exécutée indépendamment des autres processus ;

puis, lorsque tous les processus ont été définis,

- la description des interactions horizontales entre processus.

A cette phase d'étude, les réseaux de Petri peuvent servir autant comme méthode de spécification et de conception que comme outil de modélisation.

Le module d'aide à la modélisation et à la validation de graphes de commande proposé ici s'inscrit dans cette méthodologie générale de définition du contrôle d'un système de production flexible.

Il se compose de trois méta-fonctions permettant :

- la représentation modulaire du système (décomposition en processus) ;
- la définition des liaisons entre processus ;
- l'affinement du modèle ainsi que son abstraction ;
- la validation syntaxique du modèle et la vérification de certaines propriétés spécifiques du système modélisé ;
- le paramétrage.

4.2 - Première méta-fonction : construction du graphe de commande à l'aide de réseaux de Petri structurés

4.2.1 - Méthodologie de représentation de systèmes de processus

4.2.1.1 - Introduction

Un système de processus structurés peut être décrit de manière équivalente, soit à l'aide d'un langage de graphe, soit à l'aide d'un langage de spécification de haut niveau (langage littéral proche des langages de programmation évolués).

Dans les paragraphes qui suivent, nous nous intéressons à :

i) la description, dans les deux langages, de chaque composante du système de processus structurés ;

ii) la présentation du compilateur associé au langage littéral ;

iii) l'application de ceux-ci sur l'exemple d'illustration (cellule flexible) de la figure IV-3.

4.2.1.2 - Description d'un processus structuré

Un processus est défini comme un enchaînement séquentiel de tâches. Il est obtenu par composition de trois structures primitives : séquence, alternative et répétitive, que nous pouvons définir, de manière équivalente, soit par un langage de graphe (RdP structuré), soit par un langage de spécification évolué.

Afin de réduire la taille du réseau de Petri structuré modélisant un graphe de commande, nous avons opéré une réduction sur le nombre de places et de transitions appartenant aux différentes primitives initiales. Ceci constitue une modification par rapport au modèle de référence décrit dans /VER 82/. Ainsi, nous avons établi de nouvelles règles de transition de la grammaire du langage de graphe en conservant, d'une part la cohérence du modèle et d'autre part les propriétés du réseau initial.

Cette première étape de la réduction a l'avantage :

1. d'apporter un gain de places mémoires au niveau de stockage dans la base de données.

2. de faciliter la visualisation éventuelle, sur console graphique, du RdP structuré.

3. d'apporter un gain de temps d'exécution du logiciel de validation et de vérification des propriétés du RdP structuré.

A - Définition formelle d'un processus à l'aide d'une grammaire de graphe

Nous utilisons le formalisme Backus Normal Form (BNF) pour décrire les règles de transition de la grammaire du langage de graphe (figure IV-4).

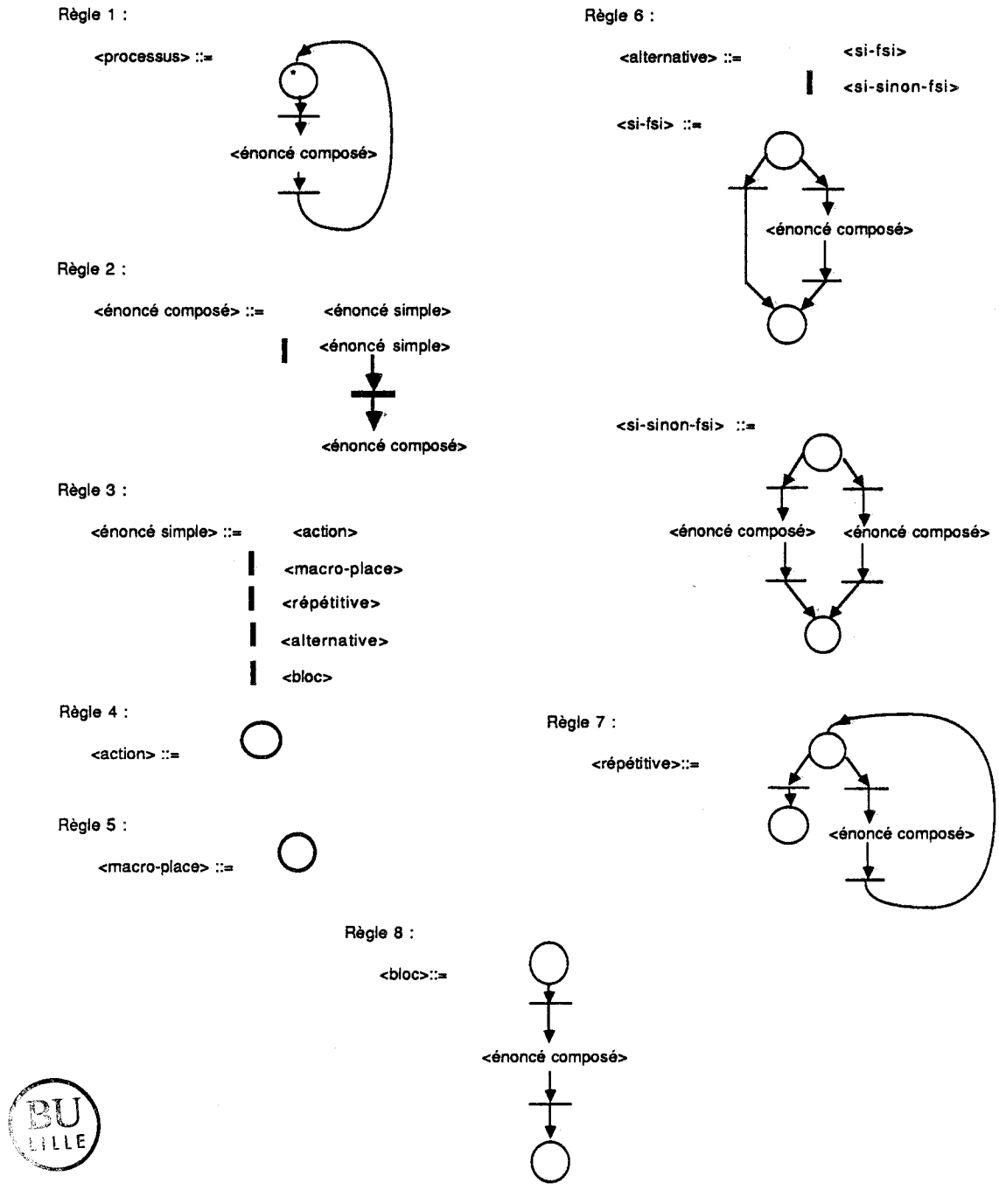


Figure IV - 4 : Les primitives de base

Résultat important :

- - - - - - - - - - -

Tout processus décrit à l'aide de ce langage de graphe, est un réseau de Petri borné, vivant et réinitialisable /VER 82/.

B - Définition équivalente d'un processus à l'aide d'un langage littéral évolué

Nous utilisons également la notation BNF pour énoncer les règles de réécriture de la grammaire du langage (figure IV -5).

<processus> ::= <en-tête du processus>
<corps du processus>
<fin du processus>

<en-tête du processus> ::= { PROC | proc } <nom du processus>

<fin du processus> ::= { FPROC | fproc }

<corps du processus> ::= <énoncé composé>

<énoncé composé> ::= <action>
| <macro-place>
| <alternative>
| <répétitive>
| <bloc>

<action> ::= {ACTION / action} <nom de l'action>

<macro-place> ::= {MACRO / macro} <nom de la macro-place>

<alternative> ::= <en-tête de l'alternative>
<énoncé composé>
<fin de l'alternative>

<en-tête de l'alternative> ::= {SI / si} <condition> {ALORS / alors}

<fin de l'alternative> ::= {FSI / fsi}
| {SINON / sinon} <énoncé composé> {FSI / fsi}

<répétitive> ::= <en-tête de la répétitive>
<énoncé composé>
<fin de la répétitive>

<en-tête de la répétitive>::= {TQ / tq} <condition> {FAIRE / faire}

<fin de la répétitive> ::= {FTQ / ftq}

<bloc> ::= <en-tête du bloc>
<énoncé composé>
<fin du bloc>

<en-tête du bloc> ::= {BLOC / bloc} <nom du bloc>

<fin de bloc> ::= {FBLOC / fbloc}

FIGURE IV-5

Pour dès raisons de sûreté de fonctionnement et de vérification des propriétés du RdP associé au système de processus structurés, la notion de bloc de synchronisation a été introduite. Les liaisons internes entre processus, se feront par l'intermédiaire des blocs de synchronisation. Par ailleurs, un bloc de synchronisation permet de nommer un groupement d'énoncés simples.

4.2.1.3 - Description des liaisons internes entre processus

Pour prendre en considération le parallélisme d'un système, il est nécessaire d'introduire plusieurs processus. L'évolution de chacun de ces processus tient compte, en règle général, de l'état des autres processus.

Il est donc nécessaire de pouvoir représenter leur interaction par des liaisons cohérentes.

Nous nous sommes intéressés à trois types de liaisons :

- liaison de synchronisation,
- liaison d'exclusion mutuelle,
- liaison producteur/consommateur.

A - Définition des liaisons dans le langage de graphe

a) Relation de synchronisation

Elle est de deux types :

- Signal mémorisé sans accusé de réception : un processus P2 est synchronisé par un processus P1, si et seulement si, le déroulement de P2 nécessite une ressource fournie par le déclenchement d'une des transitions de P1 (figure IV-6).

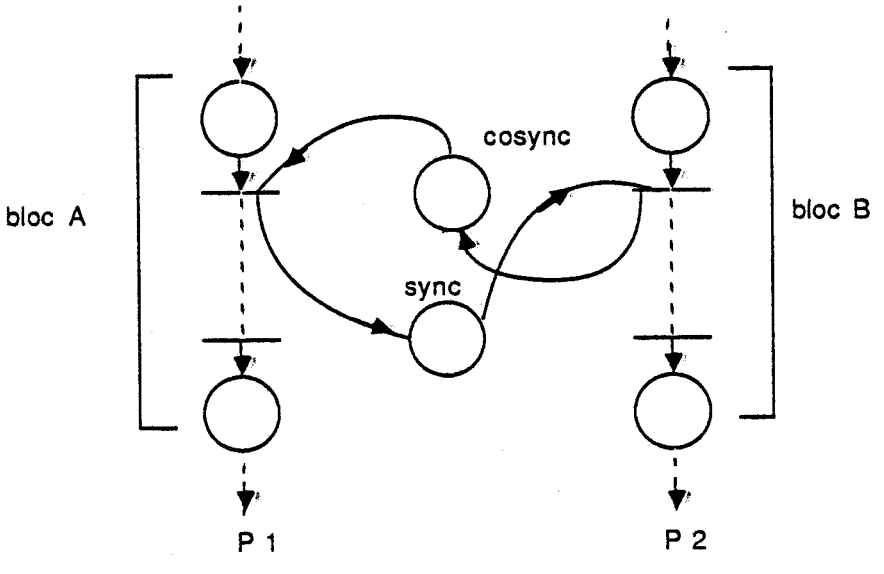


Figure IV - 6

- Rendez-vous avec accusé de réception : ce protocole de communication entre deux processus P1 et P2 est un cas particulier des relations de synchronisation. L'un des processus, P1,

synchronise le second processus P2, puis attend une ressource fournie par le déclenchement d'une des transitions de P2 : l'accusé de réception (figure IV-7).

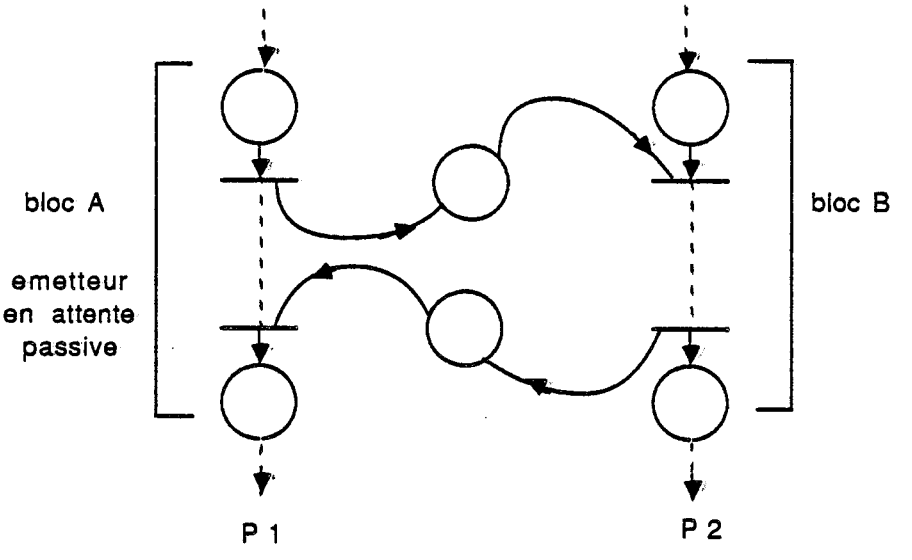


Figure IV - 7

b) Relation de partage de ressource de type exclusion mutuelle

- -

Deux processus P1 et P2 sont liés par une relation d'exclusion mutuelle Rexcl, si et seulement si, leur déroulement nécessite l'accès exclusif à une ressource non partageable. Cette relation est symétrique. La requête et la restitution doivent se faire respectivement au début et à la fin d'un bloc de synchronisation (figure IV-8).

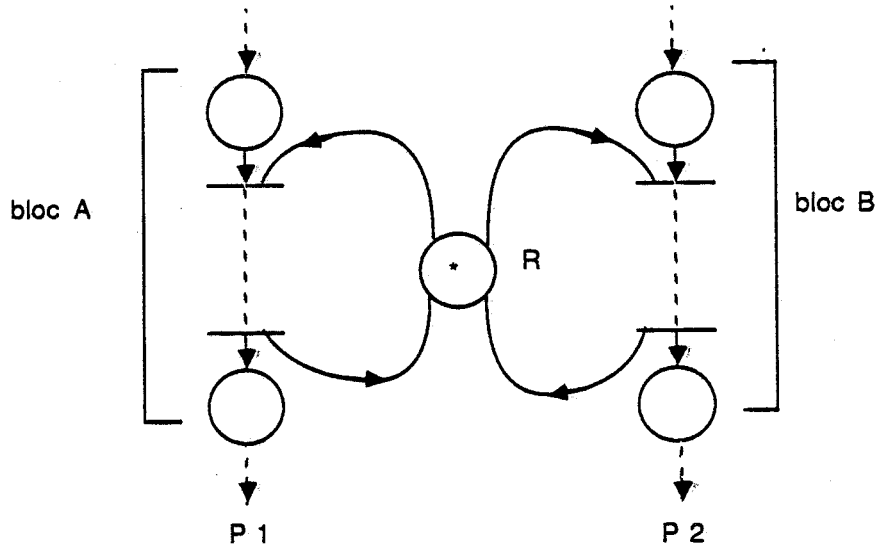


Figure IV - 8

La place R symbolise la ressource partagée et le marqueur la disponibilité de celle-ci. La notion de conflit structurel n'est pas prise en compte à ce niveau de description pour les transitions d'entrées des blocs A et B.

c) Relation producteur/consommateur

- - - - - - - - - - - - - - - - - -

Deux processus P1 et P2 sont liés par une relation producteur/consommateur Rpc, si et seulement si, leur déroulement nécessite l'accès à une ressource pouvant prendre deux états distincts noté E1 et E2 de la manière suivante :

i) Chaque processus requiert un exemplaire de la ressource se trouvant dans l'un des états E1 ou E2 et la restitue dans l'autre état ;

ii) L'un des processus a besoin de la ressource dans l'état E1 et l'autre dans l'état E2.

Le protocole de communication producteur/consommateur est instauré entre deux blocs de synchronisation appartenant à des processus distincts (figure IV-9).

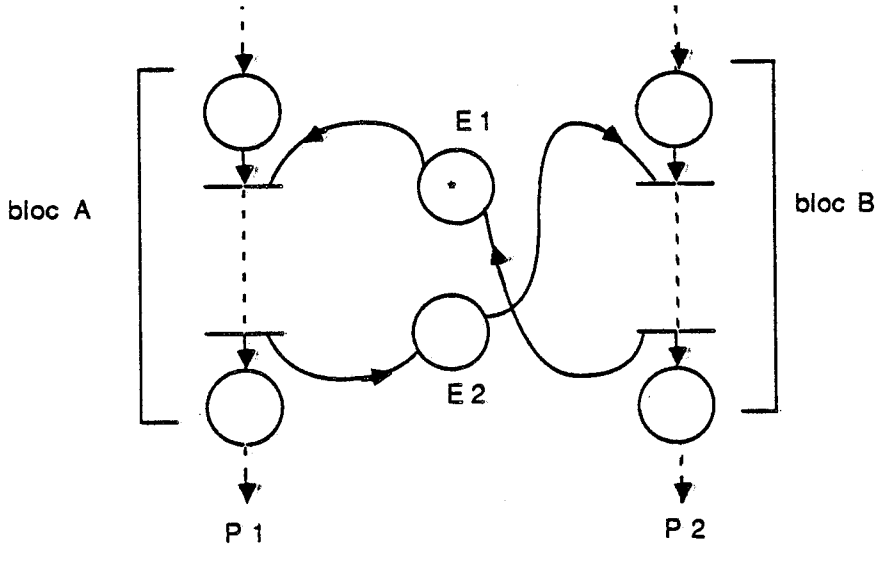


Figure IV - 9

B - Définition des liaisons internes entre processus dans le langage littéral évolué

Les règles grammaticales, décrites dans la notation BNF, associées à la définition des interactions entre processus sont (figure IV-10) :

<liaison entre processus> ::=

{SIGNAL / signal} <liaison signal mémorisé sans accusé de réception>

| {SIGACC / sigacc} <liaison rendez-vous avec accusé de réception>

| {SEMA / sema} <liaison exclusion mutuelle>

| {PC / pc} <liaison producteur/consommateur>

<liaison signal mémorisé sans accusé de réception> ::=

(<nom de l'émetteur>,<nom du récepteur>)

<liaison rendez-vous avec accusé de réception> ::=

(<nom de l'émetteur>, <nom du récepteur>)

<liaison exclusion mutuelle> ::=

(<nom de la ressource>, <nombre d'exemplaires de la ressource>,<liste d'utilisateurs de la ressource>)

<liaison producteur/consommateur> ::=

(<nom de la ressource>,<nombre d'exemplaires de la ressource dans l'état producteur>,<nombre d'exemplaires de la ressource dans l'état consommateur>,<liste des processus producteurs>, <liste des processus consommateurs>)

FIGURE IV - 10

Etant donné que les interactions entre processus se font par l'intermédiaire des blocs et qu'un processus peut en contenir plusieurs, nous avons introduit la notion de "nom composé" qui permet d'identifier :

- un émetteur ou un récepteur,
- un producteur ou un consommateur,
- un utilisateur d'une ressource partagée.

Sa syntaxe est :

<nom composé> ::= (<nom du processus>,<nom du bloc>)

Remarque : pour la définition complète du langage de spécification, voir Annexe B.

C - Remarques sur l'utilisation des liaisons

Le langage de spécification d'un système de processus structuré permet de minimiser dès l'analyse d'un problème, les défauts et les erreurs de conception dus à une mauvaise utilisation des liaisons. En effet, considérons l'exemple suivant :

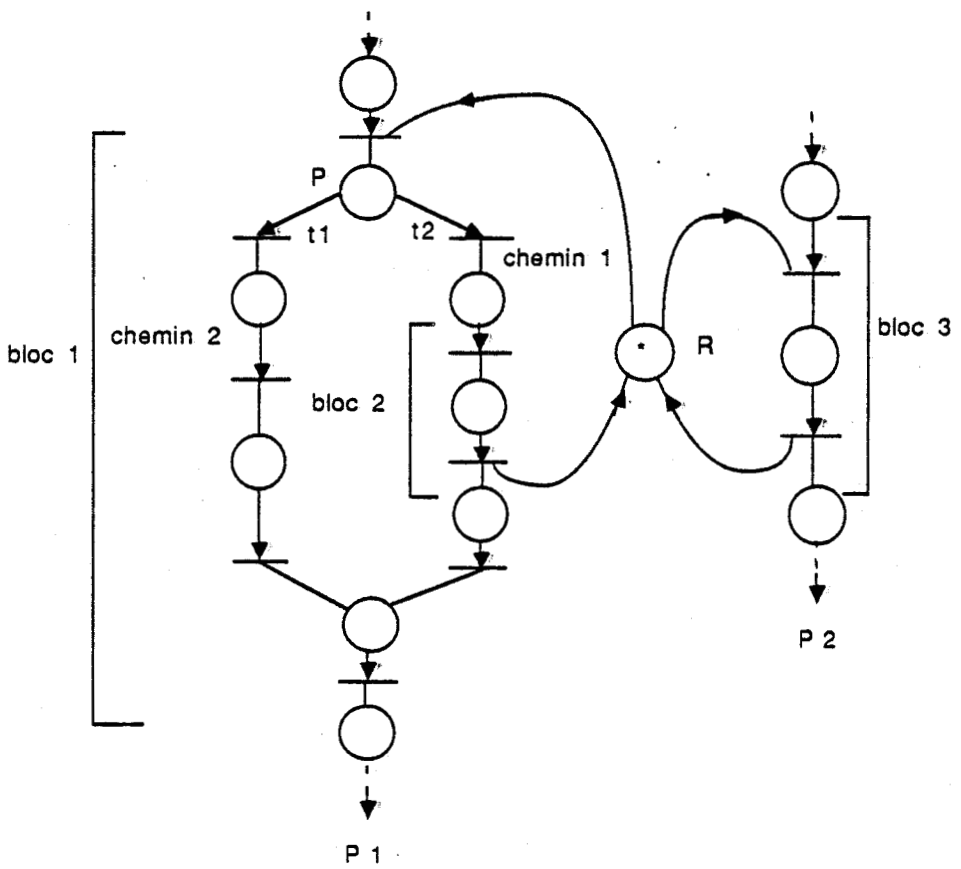


Figure IV - 11

Dans cet exemple (figure IV-11), supposons que le bloc 1 du processus P1 acquiert la ressource R. Deux utilisations de cette ressource sont possibles et correspondent sur le graphe aux deux chemins qui partent de la place P.

- Si le chemin parcouru est le chemin numéro 1, alors le fonctionnement du système des processus semble correct.

- Par contre, si c'est le chemin numéro 2 qui est utilisé, alors le bloc 1 se termine sans restituer la ressource, par conséquent le processus P2 est totalement bloqué.

Pour éviter ce type de blocage dû à une mauvaise utilisation de ressources, nous avons mis en oeuvre la règle suivante :

« Une ressource est allouée au début de bloc et restituée à la fin de ce même bloc ».

Dans ce cas, les deux configurations suivantes numérotées (1) et (2) de la figure IV-12 sont possibles séparément.

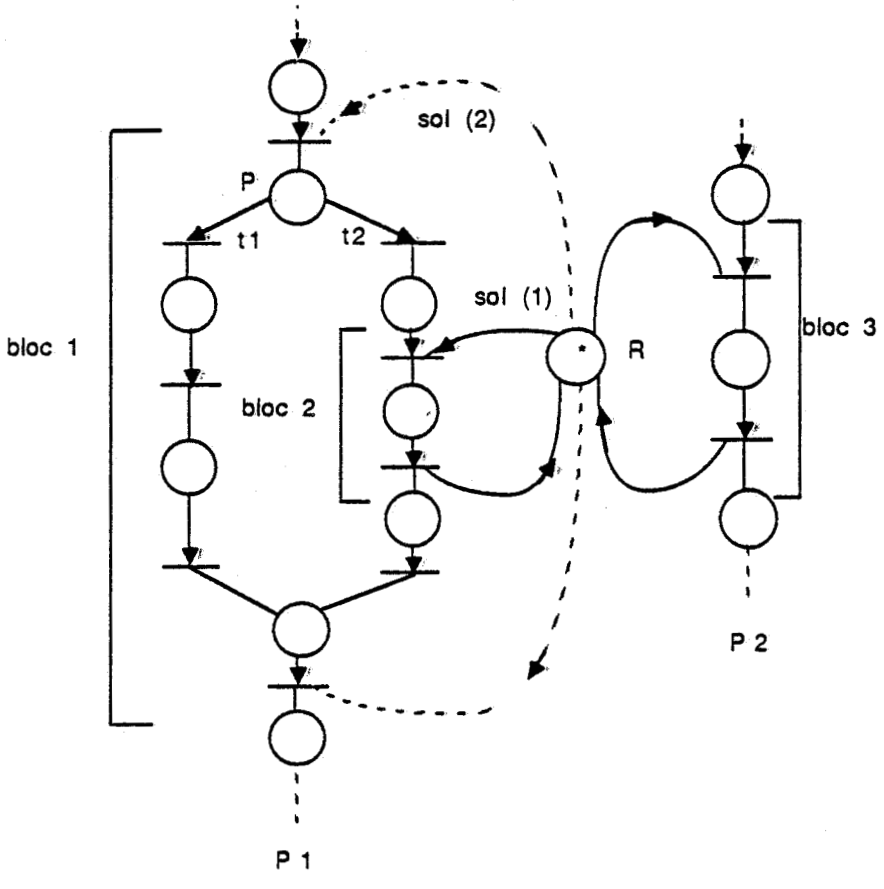


Figure IV - 12

Néanmoins, le respect de cette règle ne garantit pas l'absence de blocages. En effet, considérons deux processus P1 et P2 reliés entre eux par deux liaisons de synchronisation de type rendez-vous avec accusé de réception (figure IV-13).

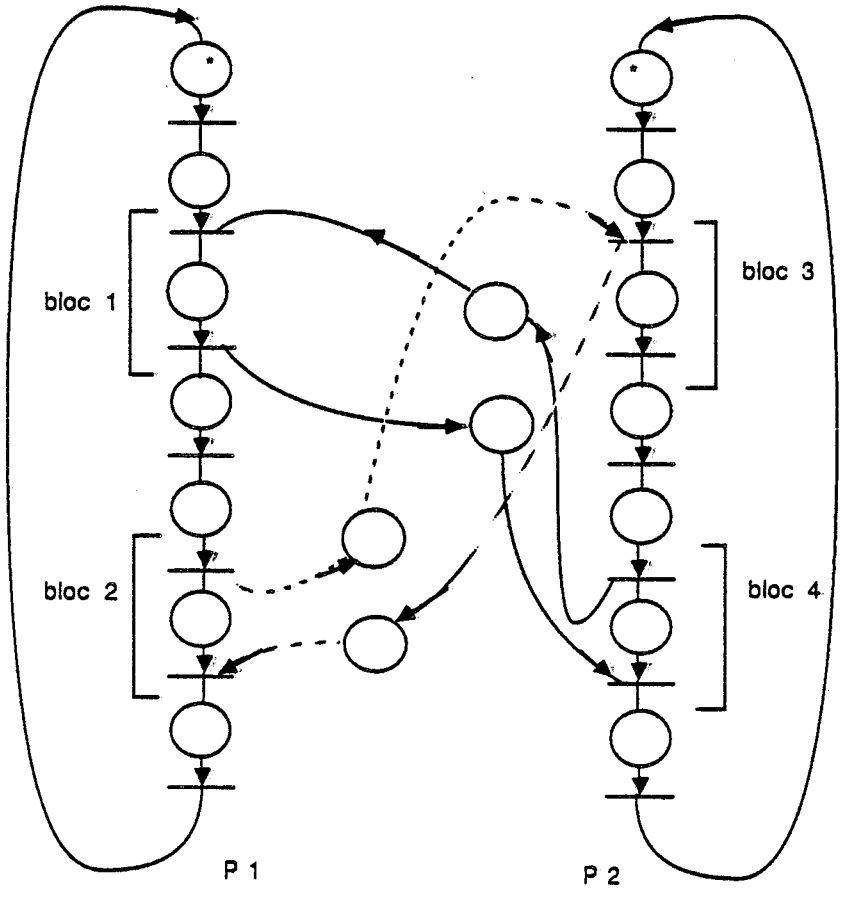


Figure IV - 13

Les processus P1 et P2 sont initialement bloqués.

4.2.1.4 - Exemple de modélisation structuré

La première phase du travail de conception consiste à spécifier l'application. Pour mener à bien cette phase, une approche descendante dans laquelle on commence à spécifier les fonctions principales et leurs interactions, semble la mieux appropriée.

Ces fonctions sont ensuite modélisées par des processus structurés et leurs interactions par les trois types de liaisons.

Dans l'exemple d'illustration présenté sur la figure IV-3 (cellule flexible), on peut identifier d'une part un certain nombre de processus tels que :

- le processus de transfert de la machine M1 vers la machine M2 (M1 --> M2) ;

- le processus de transfert de la machine M2 vers la machine M1 (M2 --> M1) ;

- le processus de transfert de la table de transfert FIFO-IN vers M1 (FIFO-IN --> M1) ;

- le processus de transfert de M1 vers la table de transfert FIFO-OUT (M1 --> FIFO-OUT) ;

- le processus de transfert de FIFO-IN vers M2 (FIFO-IN --> M2) ;

- le processus de transfert de M2 vers FIFO-OUT (M2 --> FIFO-OUT) ;

- le processus modélisant M1 ;

- le processus modélisant M2.

Et d'autre part, un ensemble de liaisons entre ces processus, tels que :

- la liaison d'exclusion mutuelle Robot. Il est partagé par les processus suivants :

  . FIFO-IN --> M1,

  . FIFO-IN --> M2,

  . M1 --> M2,

  . M2 --> M1,

  . M1 --> FIFO-OUT,

  . M2 --> FIFO-OUT.

- les liaisons du type producteur/consommateur. Elles sont au nombre de quatre :

a) T-IN1 :

processus producteurs :

. FIFO-IN --> M1,

. M2 --> M1.

processus consommateur :

. M1

b) T-IN2

processus producteurs :

. FIFO-IN --> M2,

. M1 --> M2.

processus consommateur :

. M2

c) T-OUT1

processus producteur :

. M1

processus consommateurs :

. M1 --> M2,

. M1 --> FIFO-OUT.

d) T-OUT2

processus producteur :

. M2

processus consommateurs :

. M2 --> M1,

. M2 --> FIFO-OUT.

La description de cette cellule flexible dans le langage de spécification structuré donne le programme suivant (figure IV-14) :

```
proc FIFO-IN-VERS-M1
    bloc TRANSFERT
        action WAIT-R
        bloc CS
            action FIFO-IN-VERS-R
            action R-VERS-T_IN1
        fbloc
    fbloc
fproc

proc FIFO-IN-VERS-M2
    bloc TRANSFERT
        action WAIT-R
        bloc CS
            action FIFO-IN-VERS-R
            action R-VERS-T_IN2
        fbloc
    fbloc
fproc
```

```
proc M1-VERS-M2
    bloc TRANSFERT
        action WAIT-R
        bloc CS
            action T-OUT1-VERS-R
            action R-VERS-T-IN2
        fbloc
    fbloc
fproc

proc M2-VERS-M1
    bloc TRANSFERT
        action WAIT-R
        bloc CS
            action T-OUT2-VERS-R
            action R-VERS-T-IN1
        fbloc
    fbloc
fproc

proc M1-VERS-FIFO-OUT
    bloc TRANSFERT
        action WAIT-R
        bloc CS
            action T-OUT1-VERS-R
            action R-VERS-FIFO-OUT
        fbloc
    fbloc
fproc
```

```
proc M2-VERS-FIFO-OUT
    bloc TRANSFERT
        action WAIT-R
        bloc CS
            action T-OUT2-VERS-R
            action R-VERS-FIFO-OUT
        fbloc
    fbloc
fproc

proc M1
    bloc TRANSFERT-ENTREE
        action T-IN1-VERS-M1
    fbloc
    bloc USINAGE
        action MACHINING-1
    fbloc
    bloc TRANSFERT-SORTIE
        action WAIT
        bloc TRANSFERT
            action M1-VERS-T-OUT1
        fbloc
    fbloc
fproc
```

```
proc M2
    bloc TRANSFERT-ENTREE
        action T-IN2-VERS-M2
    fbloc
    bloc USINAGE
        action MACHINING-2
    fbloc
    bloc TRANSFERT-SORTIE
        action WAIT
        bloc TRANSFERT
            action M2-VERS-T-OUT2
        fbloc
    fbloc
fproc

sema (Robot, 1, ( (FIFO-IN-VERS-M1,CS) ,
                  (FIFO-IN-VERS-M2,CS) ,
                  (M1-VERS-M2,CS)      ,
                  (M2-VERS-M1,CS)      ,
                  (M1-VERS-FIFO-OUT,CS),
                  (M2-VERS-FIFO-OUT,CS) ))

pc(T-IN1, ( (FIFO-IN-VERS-M1,TRANSFERT),
            (M2-VERS-M1,TRANSFERT) )
        , ( (M1,TRANSFERT-ENTREE)  )
        , 1 , 0 )
```

```
pc (T-IN2, ( (FIFO-IN-VERS-M2,TRANSFERT),
             (M1-VERS-M2,TRANSFERT) )
         , ( (M2,TRANSFERT-ENTREE ) )
         , 1 , 0 )

pc (T-OUT1, ( (M1,TRANSFERT-SORTIE) )
          , ( (M2-VERS-M2,TRANSFERT),
              (M1-VERS-FIFO-OUT,TRANSFERT) )
          , 1 , 0 )

pc (T-OUT2, ( (M2,TRANSFERT-SORTIE) )
          , ( (M2,VERS-M1,TRANSFERT),
              (M2-VERS-FIFO-OUT,TRANSFERT) )
          , 1 , 0 )
```

FIGURE IV-14

4.2.2 - Intégration de la base de données dans la définition du langage de spécification

La description d'un graphe de commande dans le langage de spécification structuré, dont nous venons de préciser les caractéristiques, consiste à définir complètement et explicitement tous les processus du système modélisé et leurs interactions. Ce travail doit donc être effectué en une seule fois pendant une même session.

Dans la pratique, cette démarche est contraignante. En effet, ce langage ne permet pas de mettre en oeuvre une démarche progressive qui consisterait à définir le réseau de Petri structuré représentant le graphe de contrôle global par étapes.

Une étape peut être assimilée à une ou plusieurs sessions de travail dans lesquelles le concepteur peut définir une structure de commande (sous-partie commande, méta-graphe, graphe de commande type, ...) et d'archiver ce résultat dans la base de données(voir chapitre III, § 3) . Dans le même sens, il apparaît que ce langage de spécification ne possède pas de primitives de gestion de la base données construite permettant au concepteur d'utiliser d'une part, des structures synthétisées et archivées en vue de les intégrer dans un graphe de commande plus global, et d'autre part, de reprendre un graphe de contrôle incomplet dans le but de l'enrichir.

Pour combler ces insuffisances, nous avons défini une extension du langage de spécification offrant les fonctionnalités nécessaires à l'utilisation de la base de données dans le cadre d'une conception progressive qui peut faire référence à des structures de contrôle partielle prédéfinies.

L'idée de base que nous avons retenue est l'extension du langage vers une représentation restreinte de la notion d'objet. Ce concept s'est avéré à la fois puissant et efficace pour la représentation et la manipulation des éléments de la base de données.

Les raisons d'une telle extension tiennent au fait que :

1. La représentation orientée objet permet de décrire très efficacement des informations de nature prototypiques, c'est-à-dire qui sont caractéristiques d'un type d'individu. Dans ce contexte, nous avons considéré que tout élément de la base de données est un objet structuré qui est soit un prototype, soit une instance.

Un prototype sert à représenter :

- une structure de commande synthétisée et prête à être intégrée dans différentes instances ;

- un graphe de contrôle incomplètement spécifié et qui sera éventuellement complété par des structures de premier type ;

- un méta-graphe ;

- etc.

De par la nature des données qu'il représente, le modèle prototype que nous avons adopté ne dispose pas de la notion d'attachement procédural. C'est-à-dire qu'il ne possède pas de procédures qui sont invoquées lors de la réception d'un message.

Une instance est une structure de données qui représente la description complète d'un graphe de contrôle. Elle est :

Cas 1 : soit issue de prototypes et dans ce cas, la relation prototype-instance est définie par les mécanismes d'héritage et d'instanciation. Ces derniers permettent notamment d'éviter la recopie et l'édition inutiles de prototypes partagés par différentes instances.

Cas 2 : soit spécifiée complètement et explicitement sans faire appel à des éléments prédéfinis (Pour un exemple d'instance de ce type, voir la figure IV-14).

2. La volonté de permettre au concepteur d'application de faire référence à la base de données en intégrant ses éléments lors de la phase de spécification d'un graphe de commande.

Le langage de spécification structuré que nous avons défini est suffisamment complet pour décrire un prototype ou une instance

(cas 2). Notons cependant que la représentation d'une instance fait appel à des graphes prédéfinis (cas 1), il suffit donc d'introduire des règles grammaticales mettant en oeuvre le mécanisme d'instanciation (que nous confondons, par abus de langage, avec celui de l'héritage dans la mesure où un réseau de Petri structuré peut être considéré comme "héritant" son comportement du prototype dont il est instance). Ces règles, écrites dans la notation BNF, sont : (figure IV-15)

<héritage d'un réseau prototype> ::=

    <création du lien prototype-instance>
    <renommer les processus du réseau prototype>

<création du lien prototype-instance> ::=

{ HERITE / hérite } <nom de l'instance> { DE / de } <nom du prototype>

<renommer les processus du réseau prototype> ::=

    <renommer un processus>
    <renommer les processus du réseau prototype>
  | vide

<renommer un processus> ::=

```
      <redéfinir le nom du processus>
      <reféfinir les noms des blocs du processus>

<redéfinir le nom du processus>  ::=

  { RENAME-PROC  } (<ancien nom du processus    prototype>,
  { rename-proc  }  <nouveau nom du processus   instance> )

<redéfinir les noms du processus ::=

  { RENAME-BLOC  } (<renommer les blocs>)
  { rename-bloc  }

  | vide

<renommer les blocs>  ::=

      <renommer un bloc>
    | <renommer un bloc>,<renommer les blocs>

<renommer un bloc>  ::=

      (<ancien nom du bloc "  prototype">,
      <nouveau nom du bloc "  instance">)
```

FIGURE IV-15

L'intérêt de renommer les processus et leurs blocs réside dans le fait que plusieurs instances d'un même réseau prototype peuvent être intégrées dans un graphe de contrôle. Par conséquent, il est nécessaire d'attribuer des noms différents à des processus identiques.

Pour établir les communications entre les processus, il est nécessaire de se donner les moyens de pouvoir :

1. créer de nouvelles liaisons, ceci est pris en compte dans le langage (voir figure IV-10) ;

2. connecter de nouveaux processus à des liaisons pré-existantes de type producteur/consommateur ou exclusion mutuelle.

Pour réaliser ce deuxième point, nous avons adjoint au langage les règles grammaticales suivantes (figure IV-16) :

<définition d'une liaison prédéfinie> ::=

{SEMA-OLD / sema-old} <déclaration de sémaphore>

| {PC-OLD / pc-old} <déclaration du producteur/consommateur>

<déclaration de sémaphore> ::=

(<nom de la ressource partagée>,
<nouvelle valeur initiale> ,
<liste des nouveaux processus utilisateurs>)

<déclaration du producteur/consommateur> ::=

(<nom de la ressource prod-cons>,
<nouvelle valeur initiale de la place prod.>,
<nouvelle valeur initiale de la place cons.>,
<liste des nouveaux processus prod.>,
<liste des nouveaux processus cons.>.

FIGURE IV-16

Le langage de spécification structuré ainsi défini, apporte à l'utilisateur un éventail de possibilités permettant la construction du RdP structuré modélisant la partie commande du système de procédés industriels étudié.

Exemple d'application

Nous avons proposé, sur la figure IV-14, une première solution pour spécifier le graphe de commande de la cellule flexible (figure IV-3) qui constitue notre exemple d'illustration.

Nous allons présenter maintenant une autre solution qui consiste à construire le même graphe en plusieurs étapes.

La première étape a pour objet la définition des réseaux prototypes. La deuxième sert à spécifier une partie du graphe de contrôle et enfin, la dernière étape permet de compléter la description de ce graphe.

Première étape : définition des prototypes

Dans cet exemple, on distingue deux prototypes qui représentent respectivement un procesus de transfert et un processus modélisant le fonctionnement d'une machine.

La description de ces prototypes dans le langage de spécification structuré donne le programme suivant : (figure IV-17-a)

```
rdp PROTOTYPE-MACHINE
    proc MACHINE
        bloc TRANSFERT-ENTREE
            action BANCS-TRANSFERT-VERS-MACHINE
        fbloc
        bloc USINAGE
            action MACHINING
        fbloc
        bloc TRANSFERT-SORTIE
            action WAIT
            bloc TRANSFERT
                action MACHINE-VERS-BANCS-TRANSFERT
            fbloc
        fbloc
    fproc
frdp

rdp PROTOTYPE-TRANSFERT
    proc COMMUNICATION
        bloc TRANSFERT
            action WAIT-R
            bloc CS
                action ENTREE-VERS-RESSOURCE
                action RESSOURCE-VERS-SORTIE
            fbloc
        fbloc
    fproc
frdp
```

FIGURE IV-17-a

Deuxième étape : spécification incomplète du graphe de contrôle.

Dans cette phase, nous allons construire une partie du graphe de contrôle en créant d'une part, des processus par le biais des mécanismes d'héritage et d'instanciation et d'autre part, des liaisons entre ces processus (figure IV-17-b) :

```
rdp CELLULE-FLEXIBLE-1
      hérite INSTANCE-M1 de PROTOTYPE-MACHINE
            rename-proc (MACHINE, MACHINE-1)

      hérite INSTANCE-FIFO-IN-M1 de PROTOTYPE-TRANSFERT
            rename-proc (COMMUNICATION, FIFO-IN-VERS-MACHINE-1)

      hérite INSTANCE-M1-FIFO-OUT de PROTOTYPE-TRANSFERT
            rename-proc (COMMUNICATION, MACHINE-1-VERS-FIFO-OUT)

      hérite INSTANCE-M1-M2 de PROTOTYPE-TRANSFERT
            rename-proc (COMMUNICATION, MACHINE-1-VERS-MACHINE-2)

sema (ROBOT,1,
     ( (INSTANCE-FIFO-IN-M1,FIFO-IN-VERS-MACHINE-1,CS),
       (INSTANCE-M1-FIFO-OUT,MACHINE-1-VERS-FIFO-OUT,CS),
       (INSTANCE-M1-M2,MACHINE-1-VERS-MACHINE-2,CS) ) )
```

```
pc (T-IN1,
   ( (INSTANCE-FIFO-IN-M1,FIFO-IN-VERS-MACHINE-1,TRANSFERT) ),
   ( (INSTANCE-M1,MACHINE-1,TRANSFERT-ENTREE) ),
   1 , 0 )

pc (T-OUT1,
   ( (INSTANCE-M1,MACHINE-1,TRANSFERT-SORTIE) ),
   ( (INSTANCE-M1-M2,MACHINE-1-VERS-MACHINE-2,TRANSFERT),
     (INSTANCE-M1-FIFO-OUT,MACHINE-1-VERS-FIFO-OUT,TRANSFERT) ),
   1 , 0 )

frdp
```

FIGURE IV-17-b

Troisième étape : description complète du graphe de contrôle.

Dans cette dernière phase, nous allons compléter le réseau de Petri structuré de nom "CELLULE-FLEXIBLE-1" en effectuant les opérations suivantes :

- créer une instance du sous-graphe "CELLULE-FLEXIBLE-1" de la figure IV-17-b ;

- créer de nouveaux processus ;

- créer de nouvelles liaisons ;

- connecter les processus à des liaisons pré-éxistantes.

Celles-ci donnent le programme suivant : (figure IV-17-c).

```
rdp CELLULE-FLEXIBLE

      hérite SOUS-RESEAU de CELLULE-FLEXIBLE-1

      hérite INSTANCE-M2 de PROTOTYPE-MACHINE
            rename-proc (MACHINE,MACHINE-2)

      hérite INSTANCE-FIFO-IN-M2 de PROTOTYPE-TRANSFERT
            rename-proc (COMMUNICATION,FIFO-IN-VERS-MACHINE-2)

      hérite INSTANCE-M2-M1 de PROTOTYPE-TRANSFERT
            rename-proc (COMMUNICATION,MACHINE-2-VERS-MACHINE-1)

      hérite INSTANCE-M2-FIFO-OUT de PROTOTYPE-TRANSFERT
            rename-proc (COMMUNICATION, MACHINE-2-VERS-FIFO-OUT)

pc (T-IN2,
   ( (INSTANCE-FIFO-IN-M2,FIFO-IN-VERS-MACHINE-2,TRANSFERT),
     (SOUS-RESEAU,MACHINE-1-VERS-MACHINE-2,TRANSFERT) ),

   ( (INSTANCE-M2,MACHINE-2,TRNSFERT-ENTREE) ),

   1 , 0 )
```

```
pc (T-OUT2,
   ( (INSTANCE-M2,MACHINE-2,TRANSFERT-SORTIE) ),

   ( (INSTANCE-M2-M1,MACHINE-2-VERS-MACHINE-1,TRANSFERT),
     (INSTANCE-M2-FIFO-OUT,MACHINE-2-VERS-FIFO-OUT,TRANSFERT) ),
   1 , 0 )


sema-old (ROBOT, 1,
         ( (INSTANCE-FIFO-IN-M2,FIFO-IN-MACHINE-2,CS),
           (INSTANCE-M2-M1,MACHINE-2-VERS-MACHINE-1,CS),
           (INSTANCE-M2-FIFO-OUT,MACHINE-2-VERS-FIFO-OUT,CS) ))


pc-old (T-IN1,
       ( (INSTANCE-M2-M1,MACHINE-2-VERS-MACHINE-1,TRANSFERT) ),
       (     )
       1 , 0 )


frdp.
```

FIGURE IV-17-c

4.2.3 - Compilateur du langage

4.2.3.1 - Introduction

L'écriture de compilateurs est particulièrement intéressante en utilisant les techniques et les formalismes de la programmation logique. En effet, la programmation logique peut être utilisée aussi bien pour l'analyse syntaxique d'un langage et l'évaluation sémantique de la phrase analysée, que pour assurer la représentation du formalisme utilisé. Il en résulte une grande homogénéité conceptuelle, une grande facilité de manipulation et enfin une implantation aisée /CON 86/.

Dans Prolog, connaissances et règles de raisonnements sont intégrées en un seul formalisme, ce qui permet de ne pas faire entre elles de distinction a priori : de ce point de vue, une règle peut être considérée comme une connaissance, éventuellement d'un degré assez général. D'un autre point de vue, Prolog peut être considéré comme un simple langage de programmation et, à ce titre, il est indépendant de tout présupposé linguistique /GIA 85/. Notons enfin que l'analyse d'une phrase peut être considérée comme un processus de déduction. Dans ce sens, l'emploi de variables logiques dans les règles permet de représenter de façon symbolique des informations complexes qui n'ont pas encore été rencontrées dans la phrase, et donc de formuler des contraintes sur ces dernières. Il en résulte une plus grande flexibilité et facilité de description de la syntaxe et de la sémantique d'une phrase /CLO 84/, /CLA 82/.

4.2.3.2 - Caractéristiques de la grammaire du langage de spécification structuré.

La grammaire du langage de spécification structuré définie précédemment, est un ensemble de règles qui spécifient quelles sont les suites de mots acceptables pour former une expression de ce langage. Elle précise comment les mots sont groupés et dans quel ordre peuvent se trouver ces groupes. En partant de la grammaire du langage, nous pouvons examiner n'importe quelle suite de mots et décider alors si elle possède tous les critères qui en font une expression correcte. Si une suite de mots forme une expression acceptable, il est évident que le processus qui en a décidé a aussi trouvé de quels groupes de mots elle est formée et dans quel ordre ils se trouvent, c'est-à-dire qu'il a analysé et validé la structure grammaticale sous-jacente de l'expression.

La grammaire du langage de spécification évolué appartient à la famille des grammaires dites "hors-contexte" ("context-free" en anglais).

Une grammaire "hors-contexte" est donnée par un ensemble de règles de la forme :

A ::= a1 a2 ... an

où A est non-terminal et a1,a2, ..., an sont des terminaux ou des non terminaux.

Le compilateur du langage de spécification contient quatre phases :

1 - L'analyse lexicale transforme une liste de caractères en une liste de symboles (identificateurs, nombres, mots-clés) appelés "atomes Prolog" ou "unités lexicales". Pour sa mise en oeuvre, nous avons écrit le prédicat "lire" qui lit une expression entrée au terminal et la transforme en une liste d'unités lexicales.

2 - L'analyse syntaxique transforme ensuite cette liste d'unités lexicales en une liste d'arbres représentatifs de la structure syntaxique du RdP structure étudié. Chaque arbre syntaxique, décrit dans un code appelé "code intermédiaire", représente le graphe d'un processus appartenant au RdP structuré analysé.

3 - La génération de code est décomposée en deux phases distinctes selon que l'on s'intéresse à la réalisation des mécanismes d'héritage et d'instanciation ou à la génération du code interne des processus nouvellement créés.
Dans le premier cas, il faut chercher dans la base de données les prototypes demandés par l'utilisateur et créer les instances correspondantes.
Dans le second cas, il faut, pour chaque arbre syntaxique issu de la phase 2, effectuer le passage d'une représentation en code interne tel qu'il a été défini dans le chapitre III, § 3.

4 - L'édition de liens est la dernière phase de compilation. Elle consiste à établir entre les processus définis séparément, les

liaisons demandées par l'utilisateur.

Pour améliorer la lisibilité et la sécurité de l'écriture du compilateur, nous avons séparé ces différentes phases. Nous proposons de les présenter ci-après.

4.2.3.3 - Analyse syntaxique

i) Introduction

- - - - - - -

Le problème général de l'analyse syntaxique est d'établir le rapport entre le texte et la grammaire. Si, en principe, il suffit de déterminer si le texte est conforme ou pas, en pratique, dans un compilateur, on souhaite aussi découvrir la structure du programme, c'est-à-dire construire l'arbre abstrait correspondant.

En outre, il est fondamental pour la compilation que la syntaxe du langage ne comporte pas d'ambiguïté sémantique, c'est-à-dire qu'en général une suite de mots ne puisse avoir qu'une seule signification

Pour illustrer notre propos, considérons le fragment de grammaire relative à la définition d'un processus (voir description sur figure IV-5). Cette grammaire est écrite dans le formalisme BNF. Ce dernier a une interprétation procédurale qui fixe la stratégie d'analyse. La grammaire est traduite en clauses Prolog.

Il en résulte que la stratégie est imposée par l'interpréteur Prolog : analyse descendante en profondeur d'abord.

ii) Analyse descendante avec retour-arrière

L'analyse construit l'arbre syntaxique à partir de la racine qui est l'axiome de la grammaire, ici <processus>. Il cherche la première règle qui s'applique à l'axiome et remplace ce symbole non-terminal par sa partie droite, c'est-à-dire la règle :

<processus> ::= <en-tête du processus>
<énoncé composé>
<fin de processus>

puis la règle :

<en-tête du processus> ::= { PROC / proc } <nom du processus>

Lorsqu'il atteint un symbole terminal, l'analyseur vérifie que ce terminal est bien de type attendu. L'analyse se poursuit alors à partir du dernier non-terminal apparu dans l'arbre. La technique d'analyse est descendante : l'arbre syntaxique est construit de la racine vers les feuilles. Cette stratégie est due évidemment à la stratégie d'évaluation des clauses qui résultent de la traduction de la grammaire en Prolog.

Quand l'analyseur doit remplacer un symbole non-terminal ayant plus d'une partie droite, il doit pouvoir faire le "bon choix". L'analyseur du langage de spécification structuré est non-déterministe en ce sens qu'il ne dispose pas de moyens pour faire ce choix. Il faut qu'il essaie les autres possibilités restantes ouvertes. Pour ce faire, l'analyse émet à chaque étape des hypothèses sur les structures susceptibles d'être reconnues. Si la règle courante ne s'avère pas satisfaisante, l'algorithme revient en arrière pour en essayer une autre. La stratégie utilisée est celle dite en profondeur d'abord : un seul chemin est exploré à la fois, en cas d'échec, on effectue un retour en arrière. Cette technique n'est pas limitée à l'analyse syntaxique, elle est classique dans la résolution de problèmes non déterministes.

Pour expliquer cette méthode, nous allons, à présent, donner en Prolog une illustration du programme qui analyse la grammaire de la figure IV-5. Cet analyseur possède en entrée une liste d'unités lexicales et fournit en résultat un arbre syntaxique décrit dans le code intermédiaire que nous avons défini. L'analyse est schématisée par (figure IV-18) :

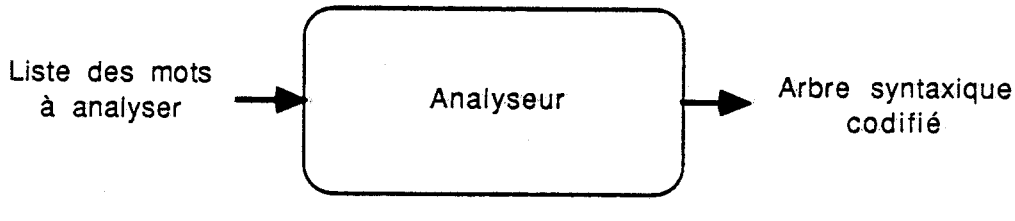


Figure IV - 18

"Définition d'un processus"

```
processus (u, v, <proc.n,c-orps-proc,fproc>) -->
        en-tête-proc (u, w, n)
        énoncé-copmposé (w, t, c-orps-proc)
        fin-proc (t, v) ;
```

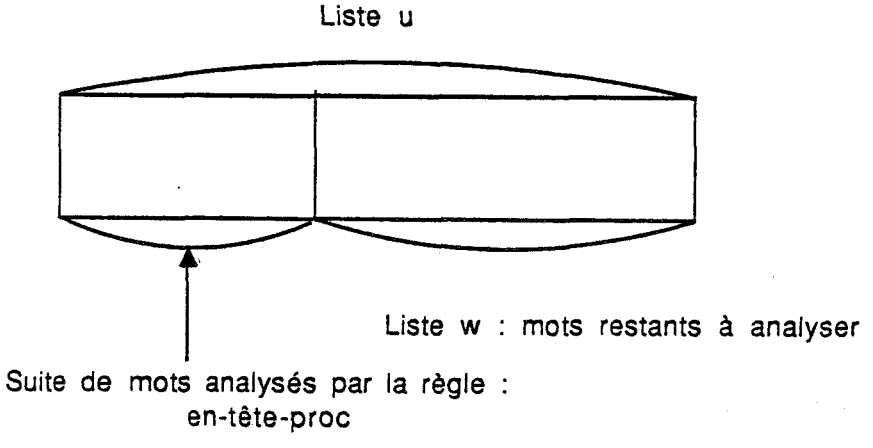
Signification des différents arguments :

u : argument d'entrée constitué de la liste des mots à analyser.

v : argument initialisé à "nil", évolue au fur et à mesure de l'analyse de u.

v, w, t contiennent, après inférence de la règle correspondante, la liste des mots restants à analyser :

exemple :



<proc.n, c-orps-proc, fproc> : est l'arbre syntaxique

codifié sous forme d'un n-uplet, dont :

"proc" est le mot clé déclarant le début d'un processus ;

"n" est le nom du processus ;

"c-orps-proc" est une variable qui va représenter la structure syntaxique du corps du processus ;

"fproc" est le mot clé déclarant la fin du processus.

```
en-tête-proc (m-ot-clé . n-om . u , u , n-om)  -->
      terminal (m-ot-clé)
      type-donnée (n-om) ;

fin-proc (m-ot-clé . u, u )  -->
      terminal (m-ot-clé) ;


      "énoncé composé"

énoncé-composé (u, v, <séquence, é-noncé-1, é-noncé-2>) -->
      énoncé-simple  (u, w, é-noncé-1)
      énoncé-composé (w, v, é-noncé-2) ;

énoncé-composé (u, v, é-noncé) -->
      énoncé-simple  (u, v, é-noncé) ;
```

```
      "énoncé simple"

énoncé-simple (u, v, s-contrôle)  --> action      (u,v,s-contrôle) ;
énoncé-simple (u, v, s-contrôle)  --> macro-place(u,v,s-contrôle) ;
énoncé-simple (u, v, s-contrôle)  --> alternative(u,v,s-contrôle) ;
énoncé-simple (u, v, s-contrôle)  --> répétitive (u,v,s-contrôle) ;
énoncé-simple (u, v, s-contrôle)  --> bloc-RdP    (u,v,s-contrôle) ;

      "action"


action (m-ot-clé . n-om . u, u, action) -->
      terminal (m-ot-clé)
      type-donnée (n-om) ;

      "macro-place"

macro-place (m-ot-clé . n-om . u, u, macro) -->
      terminal (m-ot-clé)
      type-donnée (n-om) ;

      "alternative"

alternative (u, v, <si, é-noncé-1, é-noncé-2>) -->
      en-tête-alternative (u, w)
      énoncé-composé (w, t, é-noncé-1)
      fin-alternative (t, v, é-noncé-2) ;
```

```
en-tête-alternative (m-ot-clé-1 . c . m-ot-clé-2 . u, u) -->
      terminal (m-ot-clé-1)
      terminal (m-ot-clé-2)
      condition (c) ;
fin-alternative (m-ot-clé . u, u, fsi) -->
      terminal (m-ot-clé) ;
fin-alternative (m-ot-clé . u, v, <sinon, é-noncé, fsi>) -->
      terminal (m-ot-clé)
      énoncé-composé (u,m-ot-clé-1 . v, é-noncé)
      terminal (m-ot-clé-1) ;

      "répétitive"
répétitive (u, v, <tq, é-noncé, ftq >) -->
      en-tête-répétitive (u, w)
      énoncé-composé (w, t, é-noncé)
      fin-répétitive (t, v) ;
en-tête-répétitive (m-ot-clé-1 . c . m-ot-clé-2 . u, u) -->
      terminal (m-ot-clé-1)
      terminal (m-ot-clé-2)
      condition (c) ;
fin-répétitive (m-ot-clé . u, u) -->
      terminal (m-ot-clé) ;

      "bloc"
bloc-RdP (u, v, <dbloc . n, é-noncé, fbloc>) -->
      en-tête-bloc (u, w, n)
      énoncé-composé (w, t, é-noncé)
      fin-bloc-RdP (t, v) ;
```

```
en-tête-bloc (m-ot-clé . n . u, u, n) -->
      terminal (m-ot-clé)
      type-donnée (n) ;
fin-bloc-RdP (m-ot-clé . u, u) -->
      terminal (m-ot-clé) ;

      "Terminaux"
terminal (PROC)   --> ;
terminal (proc)   --> ;
terminal (FPROC)  --> ;
terminal (fproc)  --> ;
terminal (ACTION) --> ;
terminal (action) --> ;
terminal (MACRO)  --> ;
terminal (macro)  --> ;
terminal (SI)     --> ;
terminal (si)     --> ;
terminal (ALORS)  --> ;
terminal (alors)  --> ;
terminal (SINON)  --> ;
terminal (sinon)  --> ;
terminal (FSI)    --> ;
terminal (fsi)    --> ;
terminal (TQ)     --> ;
terminal (tq)     --> ;
terminal (FAIRE)  --> ;
terminal (faire)  --> ;
terminal (FTQ)    --> ;
```

```
terminal (ftq)    --> ;
terminal (BLOC)   --> ;
terminal (bloc)   --> ;
terminal (FBLOC)  --> ;
terminal (fbloc)  --> ;
```

FIGURE IV-19

Remarque : pour garder une certaine lisibilité de ce programme Prolog, nous avons volontairement omis de typer les terminaux.

iii) Exemple d'application :

- - - - - - - - - - - -

Considérons le processus de transfert tiré de notre exemple d'illustration (figure IV-17-a).

Après analyse de cette description du processus de nom "PROTOTYPE-TRANSFERT", on obtient l'arbre syntaxique codifié sous la forme suivante (figure IV-20) :

```
<proc.COMMUNICATION, <dbloc . TRANSFERT,
    <séquence, action, <dbloc . CS, <séquence, action, action>,
        fbloc> >, fbloc>, fproc>
```

FIGURE IV-20

---

iv) Récupération d'erreurs :

- - - - - - - - - - - -

L'analyseur syntaxique tel quel ne sait qu'échouer en cas d'erreurs, sans donner aucune information sur la cause d'échec ni possibilité de la récupérer. Une méthode standard est celle des débuteurs et des terminateurs /CON 86/. Elle consiste à tester, pour chaque notion, les conditions suivantes :

- le symbole terminal qui suit peut effectivement être l'un des premiers symboles qui commence la notion suivante (terminateurs implicites). Si ce n'est pas le cas, on saute les symboles jusqu'à un terminateur effectif.

- le symbole terminal courant est bien un des terminaux par lesquels la notion peut commencer (débuteurs implicites). Si ce n'est pas le cas, on émet un message d'erreur et on saute les symboles jusqu'à ce qu'on puisse reprendre l'analyse, c'est-à-dire jusqu'à un débuteur effectif ou un terminateur.

Ce mécanisme a été implanté /JAZ 86/ par l'introduction d'un paramètre supplémentaire pour propager le contexte d'erreurs.

4.2.3.4 - Génération de code

Ainsi que nous l'avons présenté précédemment, la génération

de code est scindée en deux parties distinctes :

- la première permet de générer le code interne d'un processus en partant de son arbre syntaxique codifié fourni par l'analyseur. Rappelons brièvement que le code interne consiste à représenter chaque transition du graphe de processus par un n-uplet. Ce dernier caractérise les places d'entrées et les places de sorties de cette transition.

Considérons, par exemple, le prototype de transfert "COMMUNICATION" décrit sur la figure IV-17-a. En partant de son arbre syntaxique codifié (figure IV-20), on obtient le code interne suivant (figure IV-21-a) :

```
<COMMUNICATION, <<pl,1> . nil, <tr,1>, <pl,2> . nil> .
                <<pl,2> . nil, <tr,2>, <pl,3> . nil> .
                <<pl,3> . nil, <tr,3>, <pl,4> . nil> .
                <<pl,4> . nil, <tr,4>, <pl,5> . nil> .
                <<pl,5> . nil, <tr,5>, <pl,6> . nil> .
                <<pl,6> . nil, <tr,6>, <pl,7> . nil>.
                <<pl,7> . nil, <tr,7>, <pl,8> . nil> .
                <<pl,8> . nil, <tr,8>, <pl,1> . nil > . nil,
                8, 8 >
```

FIGURE IV-21-a

Ceci correspond au graphe de Petri suivant (Figure IV-21-b)

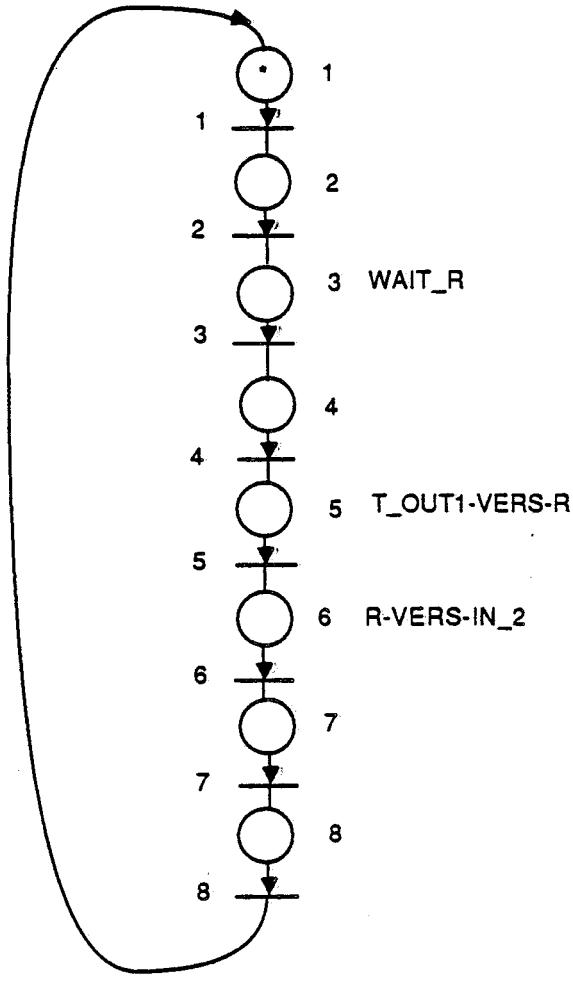


Figure IV - 21 - b

- le second volet de la génération de code consiste à mettre en oeuvre les mécanismes d'héritage et d'instanciation en créant des instances de prototypes archivés dans la base données. Ils ont été présentés § 4.2.2, figure IV-15.

4.2.3.5 - Edition de liens entre processus

Cette dernière étape de la compilation consiste à réaliser les connexions entre processus en créant les arcs correspondant aux liaisons précisées par l'utilisateur. Ceci se fait en exécutant les opérations suivantes :

1. Création des places de liaisons (2 pour une liaison de type producteur/consommateur ou synchronisation, 1 pour une liaison de type exclusion mutuelle). Pour déterminer leur numéro, on évalue la relation suivante :

numéro de la place générée := somme des nombres de places des processus utilisateurs + 1 (ou + 2). (1)

2. Selon le type de la liaison effective, on connecte les places de liaisons aux points d'entrées et de sorties des blocs de synchronisation.

Pour illustrer ce qui précède sur un exemple, considérons le processus de transfert M1-VERS-M2, le processus M2 modélise le fonctionnement de la machine 2 (figure IV-14). Ces processus communiquent entre eux par une liaison de type producteur/consommateur définie comme suit :

```
pc(t-in2, ( (M1-VERS-M2,TRANSFERT) ),
          ( (M2, TRANSFERT-ENTREE) ), 1, 0 )
```

Pour éviter de décrire complètement les processus, nous allons nous intéresser uniquement aux modifications provoquées par cette liaison, en commençant tout d'abord par dessiner le réseau de Petri structuré :

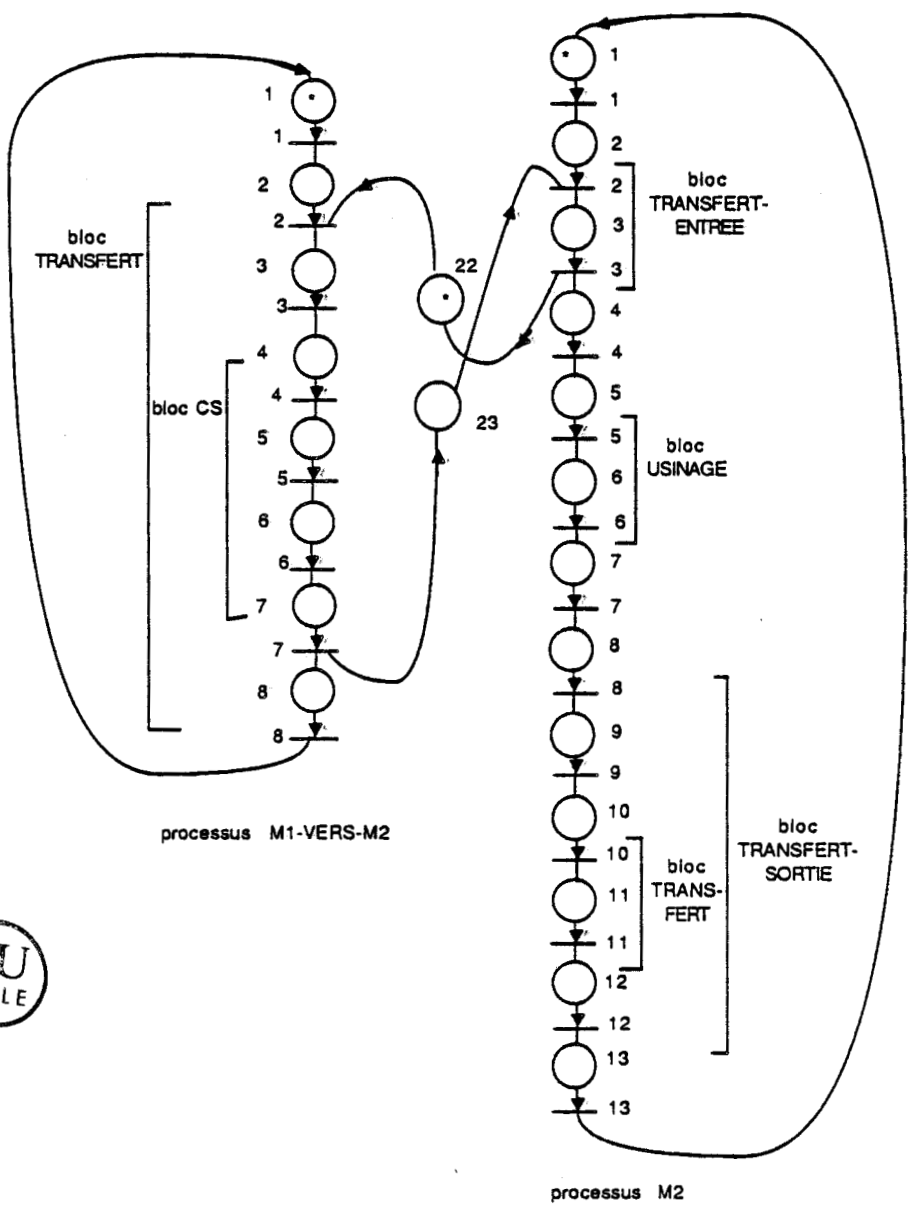


Figure IV - 22

Les modifications portent sur les points d'entrées et de sorties des blocs de synchronisation. En effet, pour le processus M1-VERS-M2, les points d'entrée/sorties du bloc TRANSFERT deviennent respectivement après traitement :

<<pl,2> . <pl,22> . nil, <tr,2> , <pl,3> . nil>

et

<<pl,7> . nil, <tr,7>, <pl,8> . <pl,23> . nil>

De même pour le processus M2, les points d'entrées/sorties du bloc TRANSFERT-ENTREE deviennent respectivement :

<<pl,2> . <pl,23> . nil, <tr,2>, <pl,3> . nil>

et

<<pl,3> . nil, <tr,3>, <pl,4> . <pl,22> . nil>

Remarque : les places de liaison producteur/consommateur ont respectivement les numéros 22 et 23. Leur calcul s'effectue selon la relation (1). Ces places sont introduites automatiquement au niveau du code interne par l'éditeur de liens.

4.2.3.6 Conclusion

Nous avons présenté ici des méthodes permettant de traiter d'une façon rigoureuse et homogène à la fois l'analyse syntaxique des programmes écrits dans le langage de spécification des RdP structurés, leur représentation dans le code interne identique à

celui de la base de données et la réalisation effective des communications entre processus à l'aide des techniques de la programmation logique.

Nous avons, au fur et à mesure de leur représentation, appliqué ces méthodes sur des exemples.

La distinction et la séparation des phases du compilateur et l'écriture en Prolog de ce dernier le rendent facilement maintenable et extensible.

4.2.4 - Conclusion de la première méta-fonction

Nous avons, dans ce sous-chapitre, rappelé les principales définitions et caractéristiques du modèle réseau de Petri structuré, qui concilie le caractère graphique de description des réseaux de Petri et la rigueur de l'analyse structurée.

La structure de ce modèle a permis une traduction aisée en langage structuré, qui supprime toute interprétation erronée lors de la construction du graphe de commande.

L'intégration d'un tel modèle dans une méthodologie de conception et de spécification permet :

- d'assurer la cohérence syntaxique du modèle représenté ;

- d'en retirer les avantages qui découlent habituellement de la structuration, notamment pour assurer une bonne sûreté de fonctionnement de l'ensemble et permettre ultérieurement une implémentation aisée du système de commande.

Ce langage permet aussi de remplir une autre fonction, à savoir, l'édition du modèle. En effet, cette phase doit être suffisamment transparente pour éviter au concepteur de gérer la complexité effective du formalisme adopté, qu'il soit graphique ou littéral, pour des systèmes de contrôle de taille industrielle (les RdP modélisant les ateliers flexibles avoisinent souvent plus de 100 places et 100 transitions).

Nous avons ensuite présenté le compilateur de ce langage, qui à partir de la description littérale d'un RdP structuré génère sa structure de données interne. Le résultat de la compilation peut être :

- soit archivé dans la base de données ;

- soit soumis à la deuxième meta-fonction portant sur la validation et la vérification des propriétés

- soit communiqué au logiciel de simulation /CAS 87/.

4.3 - Deuxième méta-fonction : validation et vérification des propriétés des réseaux de Petri structurés

4.3.1 - Introduction

La validation d'un modèle constitue une étape importante dans le processus de raffinement des spécifications. Elle représente une condition essentielle permettant à ce processus de se réitérer, en cas de non validation, afin de proposer un nouveau modèle. Ce dernier sera ensuite archivé dans la base de données pour intervenir plus tard dans la synthèse de modèles plus complexes.

D'une manière générale, la méthode proposée procède en deux étapes :

- La première est une vérification syntaxique, elle intervient lors du processus de création d'un nouveau RdP structuré. Cette étape consiste à vérifier la cohérence du modèle en utilisant le compilateur du langage de spécification structuré.

- La seconde étape de validation a pour objectif d'étudier le caractère dynamique du système modélisé en déterminant certaines caractéristiques nécessaires à son "bon fonctionnement". Elle procède de deux manières complémentaires /COR 83b/ :

* la première consiste à vérifier les propriétés structurelles du modèle indépendamment de son interprétation.

* la seconde permet de vérifier que certaines spécificités du système sont respectées (propriétés d'invariance).

Dans le contexte des travaux effectués sur le plan de la validation, les propriétés fondamentales d'un réseau concernent la finitude, la vivacité et la réinitialisation du modèle. La finitude traduit la capacité nécessairement finie du marquage des places d'un modèle de type RdP. La vivacité des transitions d'un RdP signifie que tous les événements interprétant les transitions ont toujours une incidence sur l'évolution du système modélisé, quel que soit son état et assurent en particulier l'absence de blocages ou états dans lesquels le système ne peut plus évoluer. La réinitialisation enfin, traduit l'existence d'un état stable dans le fonctionnement cyclique du système modélisé /VER 82/.

Deux classes de méthodes ont été développées pour l'analyse de ces propriétés : les méthodes classiques et les méthodes structurelles.

4.3.2 - Méthodes de validation

4.3.2.1 - Méthodes classiques

Elles nécessitent l'énumération de tous les marquages possibles obtenus à partir d'un marquage initial. Cette énumération permet de construire le graphe de couverture à partir duquel les propriétés du réseau peuvent être vérifiées /BRA 83/.

L'application de ces méthodes à des modèles complexes constitue un obstacle à l'analyse du graphe de marquage.

En effet, la dimension du graphe de commande d'un système industriel conduit très souvent à une taille mémoire et un temps de calcul important. Ceci est dû à la nature fortement combinatoire des algorithmes d'énumération.

4.3.2.2 - Méthodes structurelles

Ces méthodes se basent essentiellement sur la structure du graphe de Petri. Le marquage est considéré comme un paramètre.

Elles ont pour but de faire apparaître un ensemble de propriétés structurelles, dont l'étude consiste à rechercher les composantes consistantes et conservatrices traduisant les relations invariantes sur le marquage ou sur les tirs de transitions lors d'une évolution quelconque du graphe /BRA 83/.

La recherche de ces composantes nécessite généralement la résolution de systèmes linéaires obtenus à partir de la matrice d'incidence du réseau.

4.3.2.3 - Réduction

Pour les deux méthodes que nous venons de décrire, il est souvent nécessaire de réduire la taille du graphe en utilisant les règles de transformation qui conservent les propriétés du graphe analysé.

Ces règles de réduction générales sont en nombre de trois /BER 83/ :

- Règle 1 : substitution d'une place,
- Règle 2 : Simplification d'une place implicite,
- Règle 3 : Suppression des transitions identités et transitions identiques

4.3.3 - L'analyse des RdP structurés

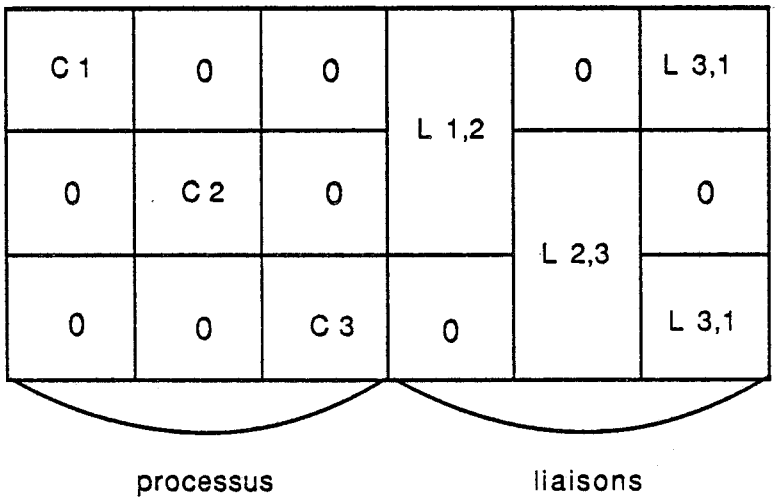
L'approche structurée mise en oeuvre lors de la conception du graphe de commande facilite l'analyse du réseau. Cette dernière peut se faire en plusieurs étapes. La première est immédiate : en effet, chaque graphe de processus est borné, vivant et réinitialisable par construction. Les propriétés internes du RdP structuré dépendent donc de la bonne conformité des liaisons. Le

respect des règles de construction des liaisons qui ont été définies au § 4.2.1.3, figure IV-10, permet de garantir la propriété borné du graphe et limite la recherche des invariants à l'intérieur du graphe de liaison. La conservation de la propriété de vivacité, par contre, ne peut être assurée lors de l'ajout de structures de liaisons entre processus et nécessite la mise en oeuvre des méthodes de validation. Les méthodes de validation générales ont été adaptées pour tenir compte des spécificités des RdP structurés /VER 82/. C'est-à-dire :

i) Le nombre de processus correspond au degré de parallélisme du système et, compte tenu de la commutativité des tirs de transitions de processus différents, il est possible de diminuer la taille du graphe de marquage.

ii) La matrice d'incidence du réseau peut être partitionnée en blocs. Chaque bloc représente la matrice d'incidence d'un graphe de processus, ou celle d'un sous-ensemble du graphe de liaison.

Exemple :



Ceci est très intéressant pour les opérations de calcul sur la matrice d'incidence, ainsi que pour le rangement en mémoire de cette matrice.

iii) La réduction d'un RdP structuré peut être interprétée comme une procédure d'abstraction de la structure du modèle.

Afin d'exploiter les caractéristiques de ce modèle, il est nécessaire d'adapter les règles générales de réduction. A ce titre, les règles de réduction qui ont été considérées sont /VER 82/, /COR 83/ :

- Règle 1 : fusionnement de places et de transitions,
- Règle 2 : simplification de transitions et de places identiques,
- Règle 3 : simplification de transitions et de places identitées.

Toutes ces règles peuvent être exécutées indépendamment les unes des autres. Cependant, l'analyse par réduction du modèle a été scindée en deux phases /VER 82/ :

- La première phase permet de conserver le graphe de liaison. Les places de liaisons symbolisent généralement une ou plusieurs ressources, elles peuvent également introduire le concept de synchronisation entre tâches. Une analyse structurelle sur le modèle partiellement réduit permet alors de vérifier les propriétés relatives aux contraintes de liaisons (assertions de liaisons).

- La seconde phase consiste en une réduction maximale du réseau. Dans le cas où celui-ci se réduit à une seule place ou à une seule transition, il sera alors possible de conclure que le réseau initial était borné, vivant, sinon il est nécessaire d'utiliser une autre méthode d'analyse appliquée au réseau réduit pour conclure.

Ces règles de réduction présentent l'avantage d'être efficaces et simples à mettre en oeuvre (elles ne nécessitent pas en particulier l'utilisation de logiciels lourds tels que le logiciel de programmation linéaire en nombres entiers).

iv) Dans le cadre de la modélisation structurée de la commande de procédés industriels, les invariants de places et de transitions déduits de l'analyse structurelle sont nécessaires ; néanmoins, ils ne constituent pas de preuve de "bon fonctionnement" du système modélisé. Nous allons montrer, sur la figure IV-23, un contre-exemple de réseau structuré possédant un invariant total de places et un invariant total de transitions sans pour autant qu'il vérifie une réelle propriété de vivacité (au sens d'absence de dead-lock).

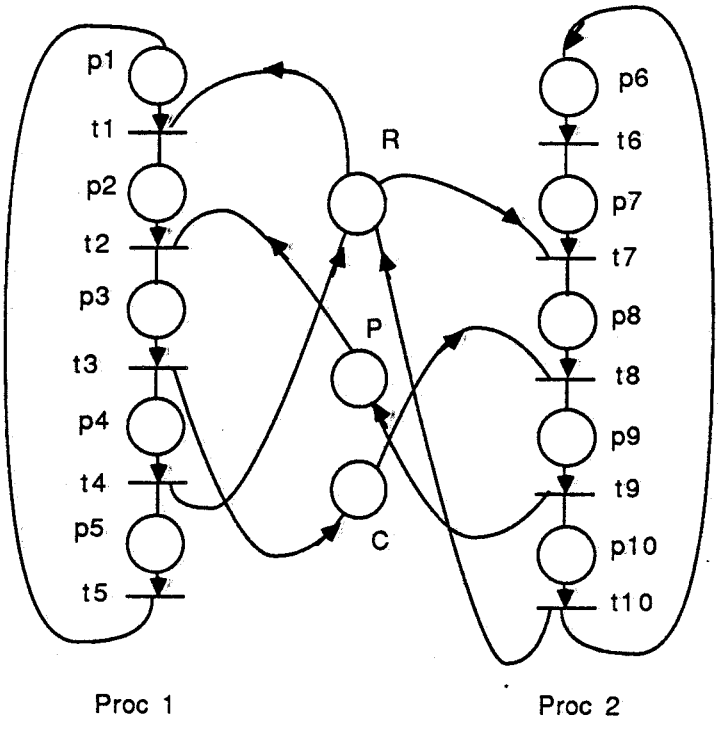


Figure IV - 23

La matrice d'incidence de ce réseau est :

	p1	p2	p3	p4	p5	R	P	C	p6	p7	p8	p9	p10
t1	-1	1	0	0	0	-1	0	0					
t2	0	-1	1	0	0	0	-1	0					
t3	0	0	-1	1	0	0	0	1			0		
t4	0	0	0	-1	1	1	0	0					
t5	1	0	0	0	-1	0	0	0					
t6						0	0	0	-1	1	0	0	0
t7						-1	0	0	0	-1	1	0	0
t8			0			0	0	-1	0	0	-1	1	0
t9						0	1	0	0	0	0	-1	1
t10						1	0	0	1	0	0	0	-1

- L'invariant total de transition It :

$$It = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^T$$

- Les invariants de places :

  - pour le processus proc 1 Ip

$$Ip = [1\ 1\ 1\ 1\ 1]$$

qui définit l'invariant de marquage :

$$M(p1) + M(p2) + M(p3) + M(p4) + M(p5) = 1$$

- pour le processus proc 2 Ip

$$Ip = [\ 1\ \ 1\ \ 1\ \ 1\ \ 1]$$

qui définit l'invariant de marquage :

$$M(p6) + M(p7) + M(p8) + M(p9) + M(p10) = 1$$

- pour le graphe de liaisons, on a les invariants de marquage suivants :

$$M(R) + M(p2) + M(p3) + M(p4) + M(p8) + M(p9) + M(p10) = 1$$

$$M(P) + M(C) + M(p3) + M(p9) = 1$$

- l'invariant total des places Ip :

$$Ip = [\ 1\ 2\ 3\ 2\ 1\ 1\ 1\ 1\ 1\ 1\ 2\ 3\ 2\ ]$$

p1 ... p5 R P C p6 ... p10

Or ce réseau présente un dead-lock, si on considère le marquage suivant :

M (p1), M (R), M (p6), M(C).

L'existence d'invariants totaux sur les places et les transitions prouve seulement qu'il existe au moins un marquage initial assurant la vivacité du graphe :

M (p1), M (R), M(P), M(p6).

Les méthodes structurelles présentent donc un intérêt relatif pour l'étude des propriétés essentielles des RdP. En effet, ces méthodes donnent une condition nécessaire mais non suffisante du fait qu'elles ne tiennent pas nécessairement compte du marquage initial du réseau.

Par ailleurs, si on prend en compte certaines extensions de RdP et en particulier l'aspect adaptatif, l'algèbre obtenu n'est plus linéaire. Dans ce cas, la recherche des invariants devient indécidable.

4.3.4 - Limites des outils de validation

La validation permet de détecter a priori les risques de fonctionnement défectueux, en particulier les blocages ou l'impossibilité pour le système de boucler, sans tenir compte de l'interprétation. Si le réseau est non vivant, non borné et non réinitialisable, il y a généralement un défaut soit dans la conception du système, soit au niveau de la modélisation.

Les outils de validation fournissent donc des résultats intéressants sur le "bon fonctionnement" du graphe de commande,

mais ne constituent qu'une première étape nécessaire dans la validation des systèmes de production industrielle. En particulier, de tels outils se heurtent, comme nous l'avons vu, à plusieurs obstacles tels que d'une part, la taille du modèle, l'insuffisance des méthodes structurelles et l'indécidabilité des propriétés structurelles due à l'aspect adaptatif. D'autre part, ils sont insuffisants lorsqu'il s'agit d'évaluer les performances en terme de production, et d'évaluer différentes stratégies de pilotage. Dans ce cas, la simulation est un outil indispensable de validation et joue dans ce sens un double rôle permettant de :

i) Prévoir le comportement réel du système, vérifier qu'il est conforme aux spécifications, évaluer ses performances, en particulier par la production de statistiques de fonctionnement.

ii) Assurer l'évaluation du mode transitoire entre deux régimes établis (démarrage ou arrêt d'une installation par exemple).

4.3.5 - Les outils de simulation

On distingue différents types d'outils de simulation suivant le degré d'interprétation du graphe de commande.

Les premiers utilisent des logiciels dédiés à la simulation des processus physiques. tel est le cas du langage de simulation SLAM. Ces langages ne permettent pas cependant de satisfaire le principe de séparation entre la partie commande et le procédé. Ils

fournissent des évaluations précises sur le comportement du système, mais les résultats obtenus ne permettent pas l'élaboration directe des processus de commande à implémenter.

Les deuxièmes réalisent une simulation du réseau de Petri modélisant la partie commande du système. Le procédé est intégré au modèle par temporisation et interprétation du graphe.

Citons deux réalisations de ce type : PSI (Petri-net based SImulator) développé au LAAS en collaboration avec le GIE Renault Recherche Innovation /ALA 84/ ; et le logiciel développé par C. GIROD /GIR 84/. Ces deux logiciels permettent de produire des calculs statistiques associés aux places et aux transitions du réseau.

Enfin, le logiciel de simulation développé plus récemment au L.A.I.I. de l'I.D.N. par E. CASTELAIN /CAS 87/, permet d'intégrer l'interprétation liée au graphe de commande, c'est-à-dire d'une part, les caractéristiques du procédé et d'autre part, les consignes de la partie décisionnelle (voir chapitre I, § 3 pour plus de détails).

4.3.6 - Interface avec les logiciels de validation et de simulation

Pour valider le graphe de commande du système de production industrielle, nous proposons l'utilisation des outils de validation et de simulation développés au Laboratoire d'Automatique et

d'Informatique Industrielle de l'I.D.N. Celle-ci sera effectuée selon deux approches complémentaires :

- La première consiste à développer et à étudier le graphe de marquage du RdP structuré. Ce dernier, de par ses caractéristiques (structuration), permet de notables simplifications dans la mise en oeuvre des moyens d'analyse (la finitude, les invariants totaux de places et de transitions sont implicites).

- La seconde approche a pour but de simuler le comportement réel du système de commande modélisé et d'évaluer ses performances.

Dans le premier cas, on utilise le logiciel de validation écrit en Pascal et développé par C. Vercauter /VER 82/. On lui fournit la matrice d'incidence et le marquage initial du réseau à analyser, en opérant la transformation représentation symbolique/représentation matricielle du réseau.

Dans le second cas, on utilise le logiciel de simulation écrit dans le langage Le_Lisp et développé par E. CASTELAIN /CAS 87/. On lui fournit également la structure de données décrivant le RdP structuré.

4.3.7 - Archivage des propriétés d'un RdP structuré dans la base de données

La prise en compte des propriétés, issues de la validation d'un RdP structuré, au niveau de la base de données se fait par la création de la relation "propriétés". Elle admet les arguments suivants :

i) identification du RdP structuré,
ii) propriétés issues de l'analyse classique,
iii) réduction.

Le schéma relationnel est :

propriétés (identification du RdP structuré,
propriétés issues de l'analyse classique,
réduction).

Avec ce cinquième prédicat, nous avons donc complété la description complète d'un RdP structuré.

4.4 - Conclusion

La finalité du module, que nous avons présenté ici, est de proposer un ensemble de fonctionnalités permettant :

- la conception interactive et progressive du graphe de commande d'un système de production industrielle et la génération du RdP structuré correspondant.

- L'étude des propriétés et l'évaluation des performances du modèle en utilisant d'une part, le logiciel de validation par analyse du graphe de marquage et d'autre part, le logiciel de simulation. Celle-ci a permis une intégration horizontale entre le module d'aide à la conception et les outils de validation et de simulation développés au L.A.I.I. de l'I.D.N.

Les perspectives de développements de ce premier module concerne principalement l'écriture de la troisième méta-fonction : paramétrage.

Elle a pour but d'une part d'enrichir le modèle de base (RdP structuré), afin d'augmenter son pouvoir de modélisation (les extensions proposées sont : RdP structurés adaptatifs et RdP structurés adaptatifs colorés), en utilisant un logiciel d'édition graphique, et d'autre part, de valider ces modèles par simulation.

5 - DEUXIEME MODULE : GESTION DE LA BASE DE DONNEES

5.1 - Logique mathématique et système de base de données

L'unification de la logique des prédicats du premier ordre et la théorie des bases de données a fait l'objet de nombreuses publications (/GAL 78/, /CLA 82/, /LLO/, /ASI 85/), et a donné lieu à de nombreuses réalisations de système de gestion "intelligent" /GAL 78/. C'est ainsi que Prolog a été utilisé :

- soit comme outil de modélisation d'un univers de connaissances ;

- soit comme outil permettant d'étendre les fonctionnalités des systèmes de gestion de base de données (SGBD).

Les applications de Prolog aux SGDB peuvent être réparties en deux catégories :

- la première consiste à établir la coopération entre Prolog et un SGDB existants (ou plusieurs) ;

- la seconde utilise Prolog pour réaliser un SGDB relationnel et l'étendre à de nouvelles fonctionnalités (possibilités déductives notamment).

5.2 - Coopération entre Prolog et SGDB

Il existe de nombreuses possibilités de faire coopérer Prolog et des SGBD. Elles peuvent être regroupées en deux classes correspondants respectivement à deux stratégies différentes. Ces deux stratégies traduisent une intégration de plus en plus étroite entre Prolog et des SGBD existants.

La première consiste à enrichir le SGBD existant avec les extensions que peut lui apporter Prolog. La seconde stratégie se propose d'apporter à Prolog, considéré comme le noyau d'un système de gestion des connaissances, les fonctionnalités propres aux SGBD /LI 84/.

5.2.1 - SGBD étendu par Prolog

C'est la stratégie qui est actuellement la plus développée. Elle permet à court terme d'enrichir notablement les fonctionnalités d'un SGBD existant selon deux approches complémentaires /CON 86/ :

- la première permet d'unifier le langage hôte et le langage de manipulation des données (LMD) du SGBD existant. Cette coopération peut être représentée par le schéma suivant /DEV 84/ (figure IV-24) :

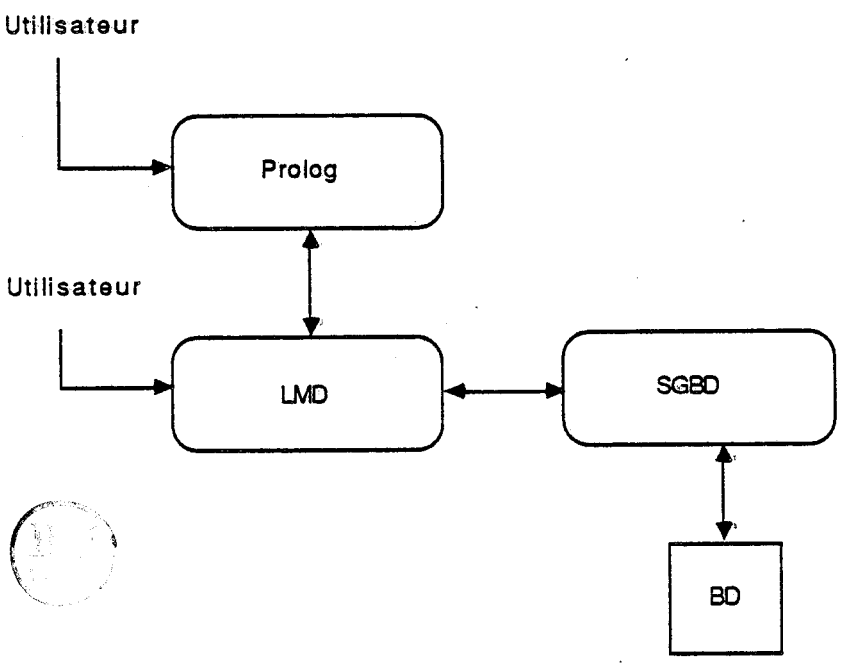


Figure IV - 24

- la seconde approche va plus loin en permettant d'ajouter à un SGBD des possibilités déductives. Le schéma général de cette coopération est celui de la figure IV-25 suivant :

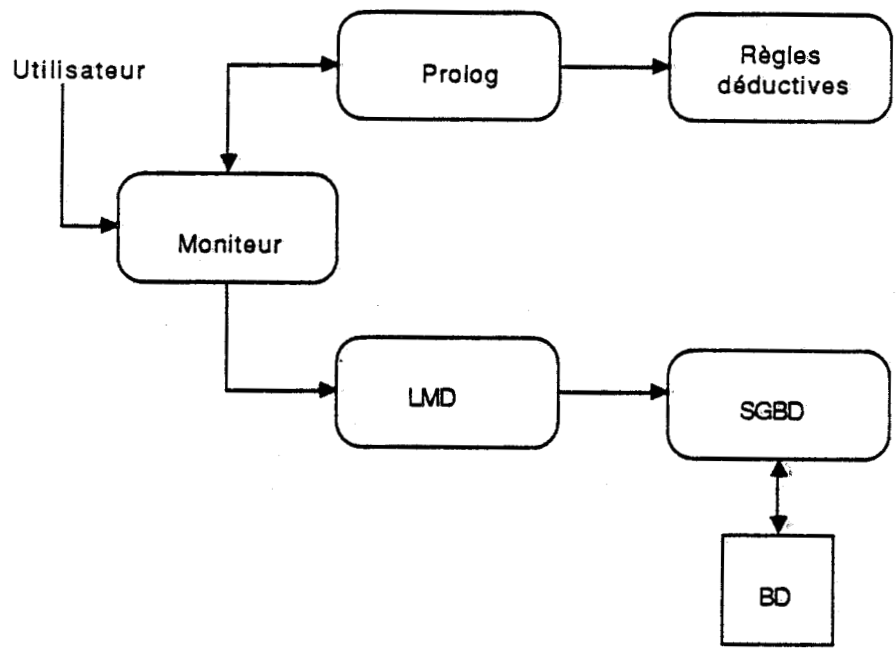


Figure IV - 25

Principe : le moniteur reçoit la requête de l'utilisateur et est chargé de la diriger vers le LMD si elle est simple, ou vers l'interpréteur Prolog et sa base de règles de déduction si elle fait intervenir des relations virtuelles. Prolog, en interprétant les règles de déduction, construit une nouvelle requête ne faisant intervenir que des relations effectivement stockées dans la base de données et la transmet au moniteur qui la dirige vers le LMD.

Des travaux de recherche sont en cours et des prototypes sont développés. Citons comme exemples /NGU 84/, BDGEN /NIC 83/.

5.2.2 - Prolog assisté par des SGBD

Cette deuxième catégorie part de l'hypothèse que Prolog représente un bon noyau pour un SBGD étendu à de nouvelles fonctions. On lui adjoint donc la possibilité de s'adresser à des SGBD existants /LI 84/, ce qui permet :

- de déléguer aux SGBD les fonctions d'accès et de stockage de grands ensembles d'informations ;

- de rentabiliser les investissements faits sur les SGBD existants.

Le schéma général pour cette catégorie est : (figure IV-26)

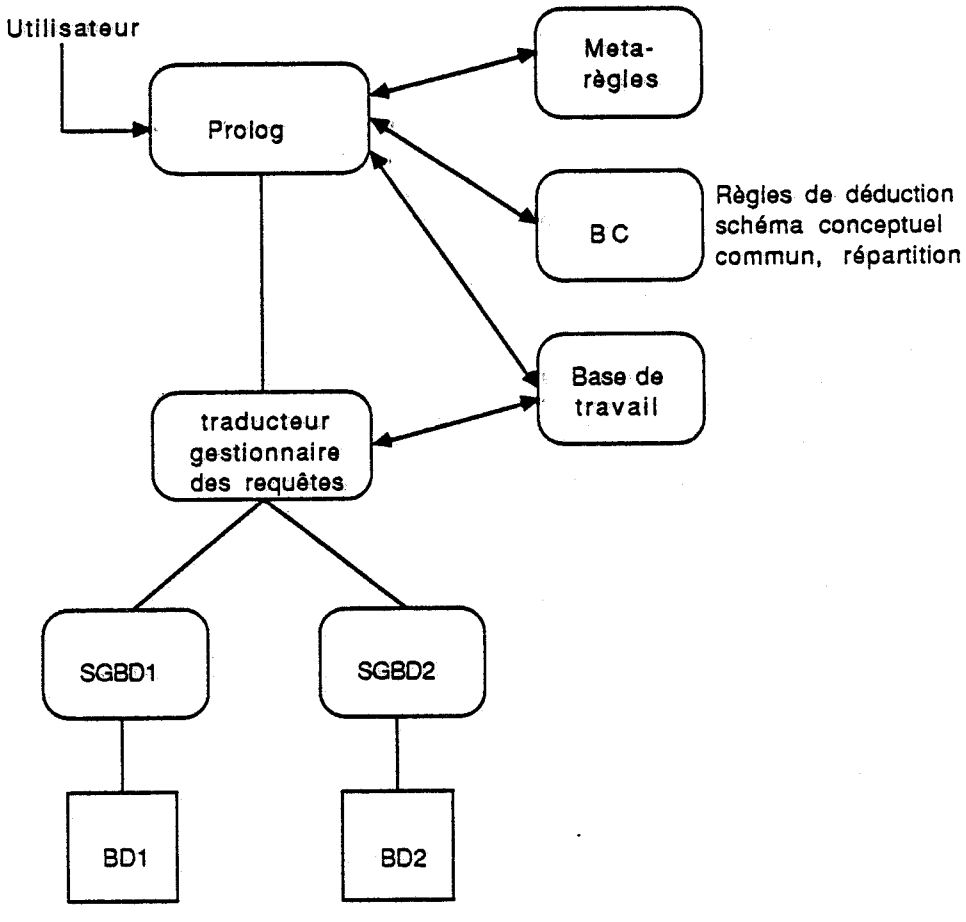


Figure IV - 26

Principe : la partie haute du schéma peut être en particulier un système expert gérant la communication entre les utilisateurs et les SGBD. Elle se compose de :

- Prolog assimilé à un moteur d'inférence de base ;

- un ensemble de méta-règles permettant de contrôler le fonctionnement du moteur d'inférence ;

- une base de connaissances (BC) contenant le schéma conceptuel (résultat du processus de modélisation du monde réel) commun aux différents SGBD, les règles de déduction, les contraintes de cohérence globales et éventuellement des informations sur la répartition ;

- une base de travail, alimentée par le traducteur gestionnaire de requêtes, qui contient les faits extraits des différents SGBD en fonction des requêtes soumises.

La partie basse du schéma a pour but d'alimenter la base de travail de Prolog. Elle comporte :

- le traducteur gestionnaire de requêtes qui prend en charge les demandes de Prolog, les optimise et les dirige vers les SGBD concernés. Il renvoit les résultats en provenance des SGBD dans la base de travail ;

- les SGBD existants.

Ce type d'approche autorise une utilisation autonome des différents SGBD, moyennant la maintenance de la cohérence globale du système /LI 84/.

5.2.3 - Conclusion

Nous avons présenté ici les différentes approches concernant la coopératon entre Prolog et des SGBD existants. La complémentarité Prolog-SGBD permet dans l'immédiat de contourner les défauts actuels de Prolog d'une part, et d'enrichir les fonctionnalités des SGBD d'autre part /CON 86/. Néanmoins, cette coopération nécessite encore de surmonter quelques difficultés mal ou non résolues jusqu'à maintenant telles que :

- la possibilité en Prolog de manipuler des variables globales ;

- l'optimisation des requêtes adressées par Prolog aux SGBD ;

- la stratégie de communication entre Prolog et les SGBD, et son contrôle.

- la possibilité d'effectuer des retours-arrière (backtraking) lors des mises à jour réalisées par le SGBD (problème de sauvegarde des états de la base de données).

5.3 - Gestion de la base de données du système CAPCOS

5.3.1 - Réalisation de SGBD en Prolog

Dans cette approche, Prolog est considéré comme le noyau d'un SGBD relationnel muni des possibilités déductives. Il permet en effet de représenter et manipuler des connaissances, d'en assurer la cohérence et l'intégrité, de définir des interfaces orientées vers l'utilisateur.

Deux raisons justifient le choix de cette approche pour développer le module de gestion de la base de données du système d'aide à la conception de graphes de commande :

- la première est qu'il est difficile et complexe de représenter le graphe de commande d'un système de production industrielle dans un SGBD relationnel classique (absence de structures de données complexes). Celle-ci exclue l'adoption de la première approche présentée au paragraphe précédent (§5.2).

- la seconde raison résulte de notre volonté de maintenir une certaine homogénéité des outils utilisés globalement pour notre projet et pour le développement d'un tel module. Nous avons, par conséquent évité d'alourdir le système en lui ajoutant d'autres interfaces.

5.3.2 - Fonctions du module de gestion de la base de données (MGBD)

Le MGBD se compose de quatre méta-fonctions qui sont :

- la consultation de la base de données (BD) ;
- la modification d'objets de la BD ;
- la suppression d'objets de la BD ;
- l'archivage d'objets dans la BD.

Ces méta-fonctions permettent principalement de représenter et d'organiser les connaissances ; de fournir des procédures de gestion qui ont pour but d'offrir à l'utilisateur une interaction avec la base de données sous-forme d'un dialogue pour rechercher, sélectionner, modifier et supprimer des données ; de définir des règles permettant de maintenir l'intégrité et la cohérence de la base de données.

Nous allons donc présenter ci-après, chacune de ces fonctionnalités.

5.3.2.1 - Représentation et organisation des connaissances

i) Représentation

- - - - - - - - -

Nous avons montré, au chapitre III, que la représentation d'un graphe de commande dans la base de données se fait à l'aide de

cinq relations ou prédicats, Ces derniers sont de la forme suivante :

P (arg 1, arg 2, ..., arg n)

où

- P est le nom du prédicat (P = "graphe" ou "liaison" ou "référence" ou "propriétés" ou "commentaire").

- arg 1, arg 2, ... , arg n sont les noms des constituants ou arguments du prédicat. Ils peuvent être :

  . soit de type simple (entier, caractère, identificateur, chaîne de caractères, etc.)

  . soit de type complexe (arbre, listes, etc.)

- n est l'arité du prédicat

ii) Organisation

- - - - - - -

L'implantation actuelle du système Prolog II (/COL 83/, /GIA 85/), ne permet pas de manipuler directement les clauses en mémoire secondaire. Par conséquent, l'ensemble des objets de la base de données doit être présent en mémoire active. Dans ce contexte, l'organisation des connaissances en mémoire centrale de l'ordinateur est une étape primordiale. Elle repose sur deux notions : modularité et protection.

ii-a) Modularité

- - - -

D'une manière générale, le développement d'applications de grande taille impose une découpe des programmes et des données en entités. Ceci implique que le langage possède des concepts de structuration.

Au plan de la modularité, on distingue essentiellement trois approches :

- absence de concepts : C-Prolog /CLO 84/, D-Prolog /DON 84/, Micro-Prolog /CLA 85/.

- réseaux de modules : Prolog/P/BARB 83/.

- hiérarchie de modules ou mondes : Prolog II /COL 83/.

Dans le modèle réseau de modules, comme celui présenté sur la figure IV-27, les règles de visibilité doivent être explicites.

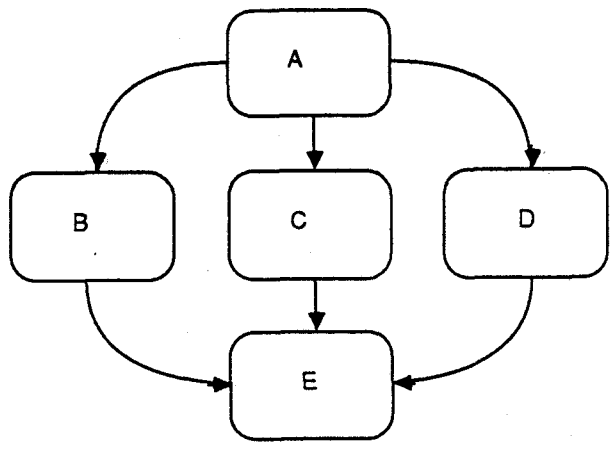


Figure IV - 27 : Organisation de modules en réseau

Ce modèle, comme dans Prolog/P, s'appuie sur les principes suivants :

- étanchéité a priori des modules : implicitement, un module ne connaît rien de l'extérieur et ne laisse rien voir à l'extérieur (concept proche de celui utilisé dans Modula II et Ada) ;

- l'opération EXPORTER permet de rendre visible à l'extérieur les prédicats désignés du module courant ;

- l'opération IMPORTER permet de référencer un prédicat défini dans un autre module à la condition qu'il soit visible.

Dans le modèle hiérarchique, qui est adopté par Prolog II, il s'agit d'une structuration de l'espace des prédicats en une hiérarchie arborescente de mondes. Un monde lègue à ses descendants tous les noms qu'il connaît lui-même (c'est-à-dire tous les prédicats définis dans ces ancêtres et ceux définis dans le monde même). Il n'y a pas à expliciter les règles de visibilité : celles-ci sont révélées par la structure sous-jacente.

L'application de cette méthode de structuration, imposée par le langage Prolog, est facilitée par la décomposition de la description d'un graphe de commande en cinq relations.

En effet, nous avons associé un monde à chaque type de relation et avons organisé l'ensemble sous-forme d'une hiérarchie donnant le schéma suivant (figure IV-28) :

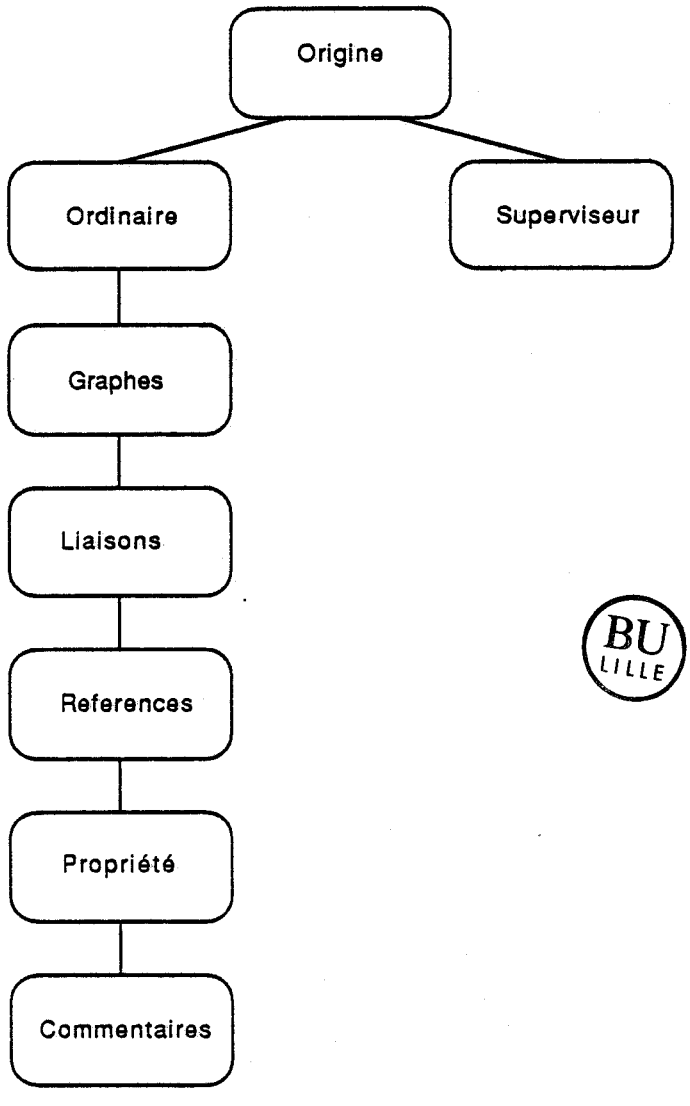


Figure IV - 28 : Organisation de la base de données en mémoire active

Le monde "superviseur" contient tous les appels aux règles prédéfinies de Prolog ; le monde "origine" contient tous les accès à ces règles et qui permet d'isoler les primitives du "superviseur" de celles de l'utilisateur ; le monde "ordinaire" est celui dans lequel l'utilisateur se trouve normalement et qui contient les prédicats évaluables de l'utilisateur ; et enfin, les modules "graphes", "liaisons", "références", "propriété" et "commentaires" contiennent respectivement l'ensemble des relations de la base de données de noms "graphe", "liaison", "référence", "propriétés" et "commentaire".

ii-b) Protection

- - - -

La modularité ainsi explicitée pourra servir ensuite à différencier les "règles de gestion et de manipulation" des "connaissances de la base de données", en associant une certaine forme de protection interdisant l'alteration des prédicats. C'est aussi un élément concourant à la fiabilité.

Au delà de la protection "interne" purement dynamique, il pourra être introduit une protection "externe" de nature statique interdisant la "décompilation" (donc la connaissance par l'utilisateur des "sources" Prolog de l'application). Ceci permet d'assurer la confidentialité qui accompagne généralement tout développement industriel de logiciel. Cet aspect est en général traité par les langages compilés, il est rarement pris en compte par les langages interprétés.

5.3.2.2 - Procédures de gestion des connaissances

i) Introduction

- - - - - - -

La manipulation des connaissances se fait à l'aide de primitives de gestion permettant la consultation, l'ajout, la suppression et la modification d'objets de la base de données, en tenant compte de la représentation et de l'organisation des connaissances en mémoire active.

Ces méta-fonctions mettent en oeuvre un ensemble de techniques en utilisant :

- les prédicats prédéfinis de Prolog chargés de la gestion de clauses ;

- les opérateurs relationnels de base tels que la projection le produit cartésien, l'union, la jointure, le filtrage, etc.

- des règles plus complexes constituées des deux premières ;

- des méta-règles définissant des stratégies d'ordonnancement des règles dans le module de gestion de la base de données.

ii) Opérations de base

- - - - - - - - - - -

Elles permettent de générer d'autres relations et s'expriment de façon naturelle en Prolog.

1. La projection de la relation R suivant sa deuxième composante C2 est une relation P dont les n-uplets sont obtenus par élimination des valeurs des arguments de R n'appartenant pas à P :

```
P (C1, C3) --> R (C1, C2, C3) ;
```

2. Le produit cartésien de deux relations R et P est une relation S ayant pour arguments la concatenation de ceux de R et P, et dont les n-uplets sont toutes concatenations d'un n-uplet de R à un n-uplet de P :

```
S (C1, C2)  -->  R (C1)
                 P (C2) ;
```

3. L'union de deux relations P et R de même schéma relationnel, est une relation S de même schéma contenant l'ensemble des n-uplets de R ou de P ou aux deux relations :

```
S (C1, C2) --> P (C1, C2) ;
S (C1, C2) --> R (C1, C2) ;
```

4. La jointure de deux relations P et R selon une qualification multi-arguments q est l'ensemble des n-uplets de P x R satisfaisant la qualification q :

S (q, C2, C3) --> P (q, C2)
R (q, C3) ;

5. La relation d'un ensemble de n-uplets de la relation R selon un critère de sélection sur son deuxième argument s'écrit :

S (C1, C3) --> R (C1, C2, C3)
Critère (C2) ;

Parmi ces outils, on distingue une opération fondamentale commune aux méta-fonctions, et qui consiste à rechercher et sélectionner un RdP structuré de la base de données.

iii) Recherche dans la base de données

La recherche, selon un certain critère, consiste à retrouver les caractéristiques et les propriétés de modules fonctionnels archivés dans la base de données.

Les critères de recherche sont de deux types correspondant respectivement au nom et au graphe du RdP structuré.

iii-1) Premier critère de recherche : nom du RdP structuré

- - - - - - - - - - - - - - - - - - - -

/KAR 85/

L'algorithme de recherche dans la base de données, selon ce critère, consiste à comparer le nom (constante de type identificateur), fourni par l'utilisateur au module de gestion, avec l'argument "identification du RdP structuré" de la relation "graphe".

Si cette comparaison aboutit à un résultat positif (valeur Vraie), les caractéristiques demandées par l'utilisateur, concernant le module fonctionnel trouvé, seront donc extraites ; s'il y a échec, on en déduit alors que l'objet recherché n'est pas archivé dans la base de données.

Cette opération de consultation, utilisant l'algorithme d'unification de Prolog, a l'avantage d'être rapide à mettre en oeuvre.

iii-2) Second critère de recherche : graphe du RdP structuré

- - - - - - - - - - - - - - - - - - - -

/KAR 85/, /KAR 86/

Dans ce cas, la consultation de la base de données repose sur la comparaison de deux graphes de Petri structurés. L'un appartient à la base de données, il est représenté, comme nous l'avons vu au (chapitre III, § 3), sous forme d'une liste d'arbres décrite à

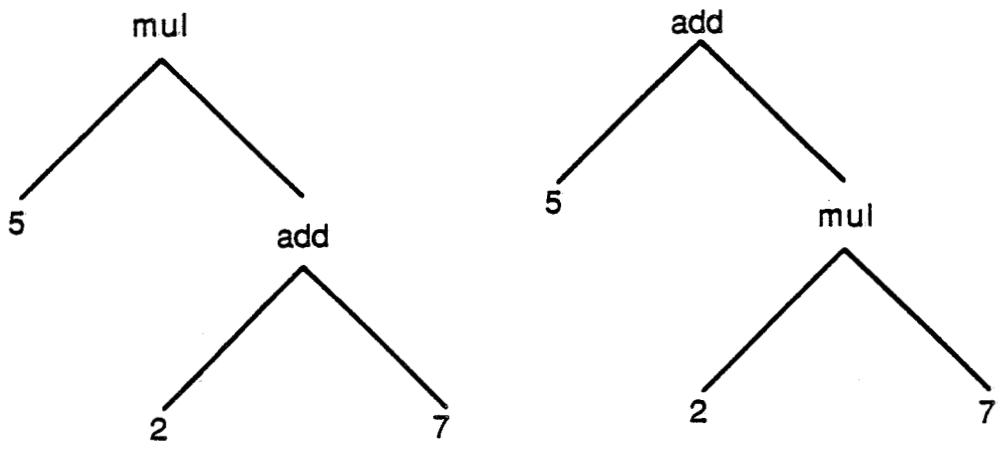
l'aide de constantes (identificateur, nombre entier, etc.) et nous notons **"graphe-bd"**.

L'autre, fourni par l'utilisateur, est décrit dans le langage de spécification structuré. Partant de cette description litérale, le compilateur génère la structure interne correspondante et qui est compatible avec la base de données, c'est-à-dire une liste d'arbres décrite à l'aide de constantes.

Dans ce contexte, la comparaison de deux graphes de Petri repose sur l'unification de leurs structures internes respectives.

Etant donné que les listes d'arbres sont décrites à l'aide de constantes, l'algorithme d'unification consiste uniquement à vérifier leur égalité. Ceci découle du principe de l'algorithme d'unification proposé par Prolog. Celui-ci, pour que l'unification de deux arbres (une liste est un cas particulier d'arbre), dont les noeuds sont des constantes, fournisse un résultat positif, vérifie d'une part que leurs structures (schéma d'arbre) coincident et d'autre part que leurs noeuds respectifs sont identiques.

contre-exemple :



Arbre 1 Arbre 2

deux arbres non unifiables

Vu la particularité de notre application, cette démarche s'avère insuffisante et contraignante du fait qu'elle ne permet pas de conclure sur l'égalité de deux graphes de Petri identiques dont les places et/ou les transitions sont numérotées différemment.

En effet, l'unification de deux graphes de Petri repose sur l'idée de base suivante :

"Deux RdP structurés sont identiques, si leurs schémas de graphe coincident et ceci indépendamment des valaurs des noeuds (numéros des places et transitions)".

Cette remarque nous amène à proposer la solution suivante :

"Deux RdP sont identiques, s'il existe une substitution permettant de faire coincider leurs représentations internes (listes d'arbres)".

La mise en oeuvre de cette technique suppose que l'une des deux structures internes ne contienne que des noeuds variables. Et à ce titre, nous avons enrichi le compilateur, qui en partant d'un RdP structuré fourni par l'utilisateur et décrit dans le langage de spécification structuré, génère, dans ce cas, une liste d'arbres dont les noeuds sont représentés par des variables. Cette représentation est notée **"graphe-var"**.

La technique de comparaison utilise le principe de la méthode d'unification de Prolog.

Exemple de description à l'aide de variables (figure IV-29) :

```
proc EXEMPLE
    si condition alors
        action 1
    sinon
        action 2
    fsi
fproc
```

La description de ce processus en utilisant les variables (p1, p2, ..., pour les places ; t1, t2, ..., pour les transitions), donne le n-uplet suivant :

<EXEMPLE, <p1.nil, t1, p2.nil> . <p2.nil, t2, p3.nil> .
<p3.nil, t3, p5.nil> . <p2.nil, t4, p4.nil> .
<p4.nil, t5, p5.nil> . <p5.nil, t6, p1.nil> .
nil, 5, 6 >

Ceci correspond au schéma suivant :

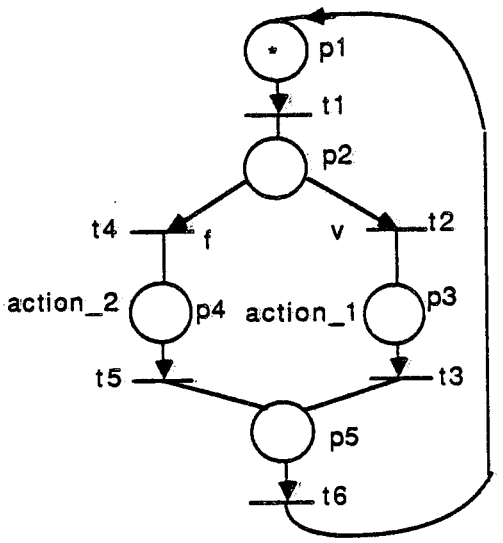


Figure IV - 29

Cette solution proposée ne résoud pas tout le problème d'unification deux graphes de Petri. En effet, deux cas majeurs ne sont pas pris en compte lors de la comparaison des deux représentations graphe-bd et graphe-var :

- d'une part, le rangement des éléments (un élément est un n-uplet décrivant un processus) dans la structure de données

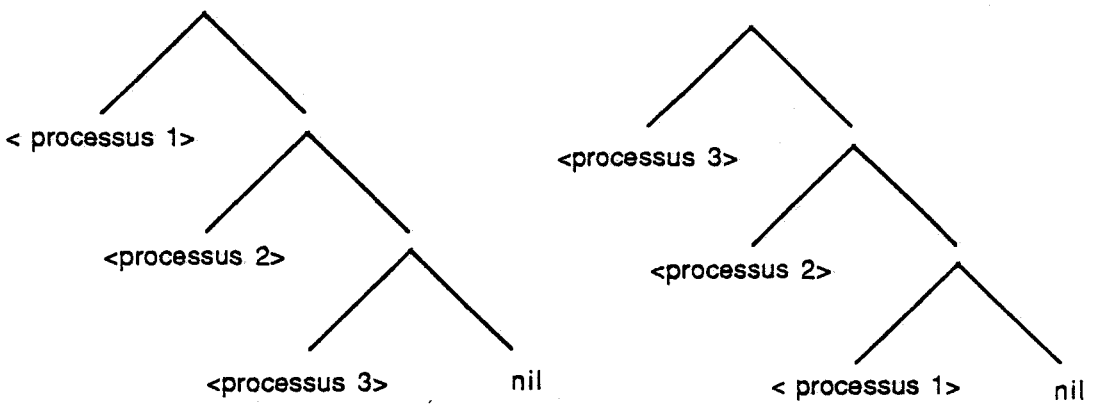
graphe-var n'est pas ordonné, c'est-à-dire qu'il n'est pas forcément identique à celui de graphe-bd ;

- d'autre part, la description de chaque graphe de processus à l'aide de variables, permet une numérotation arbitraire sur les places et sur les transitions.

Ceux-ci nous conduisent à avoir :

- différentes configurations possibles d'un même système de processus, qui se traduit par différents schémas équivalents d'une même structure graphe-var.

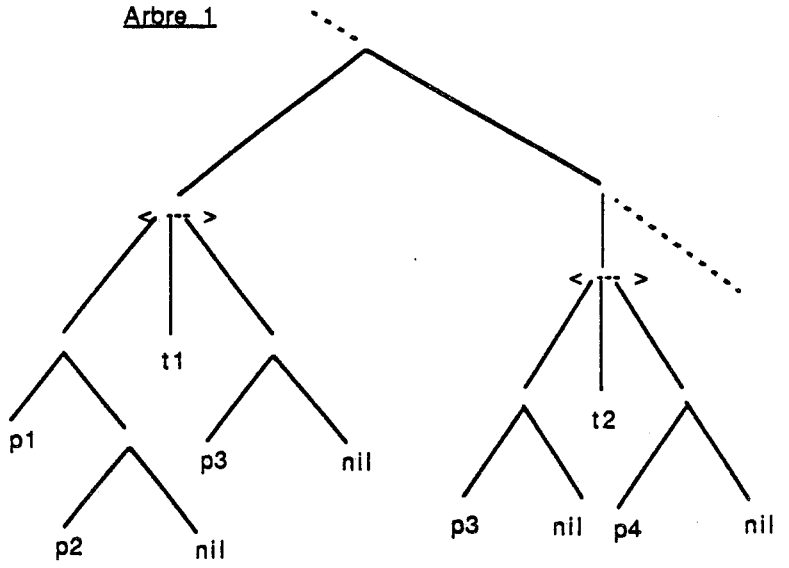
Exemple :

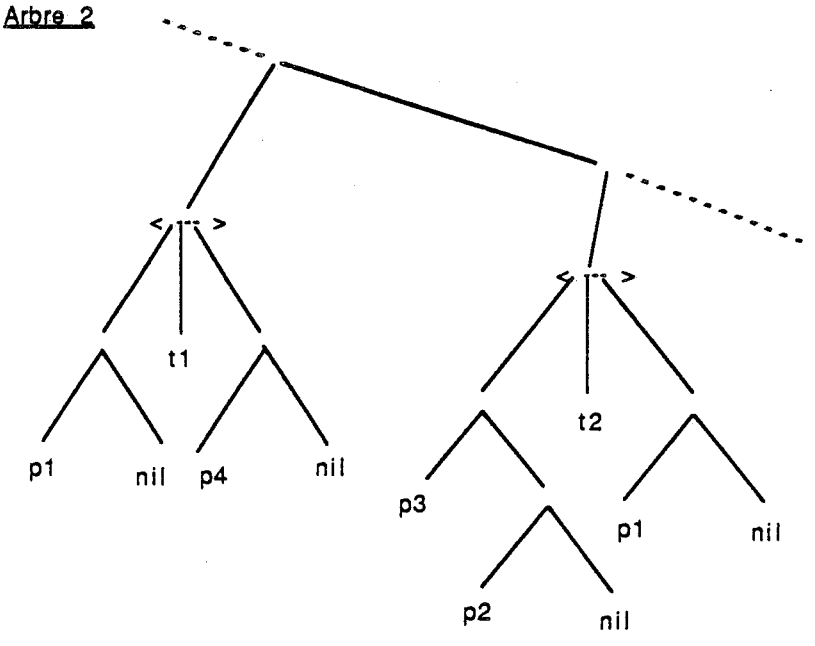


Deux configurations possibles d'un même système de processus.

- différentes configurations possibles de graphe d'un même processus, c'est-à-dire différents schémas d'arbres équivalents représentant le même processus.

Exemple : considérons une portion d'un processus





Deux schémas d'arbres d'un même processus

Par conséquent, l'ordre de rangement, soit celui des processus dans la structure de données graphe-var, soit celui des triplets de l'arbre représentant un processus, diffère d'une configuration à une autre et, aboutit à rendre l'algorithme d'unification au sens de Prolog inefficace car, il ne permet à lui seul de conclure sur l'égalité de deux RdP structurés.

Nous avons donc été amenés à développer un algorithme original d'unification de graphes de Petri. Nous le présentons ci-après.

Algorithme de comparaison de graphes de Petri

Le cylce de base de cet algorithme s'écrit comme suit (figure IV-30) :

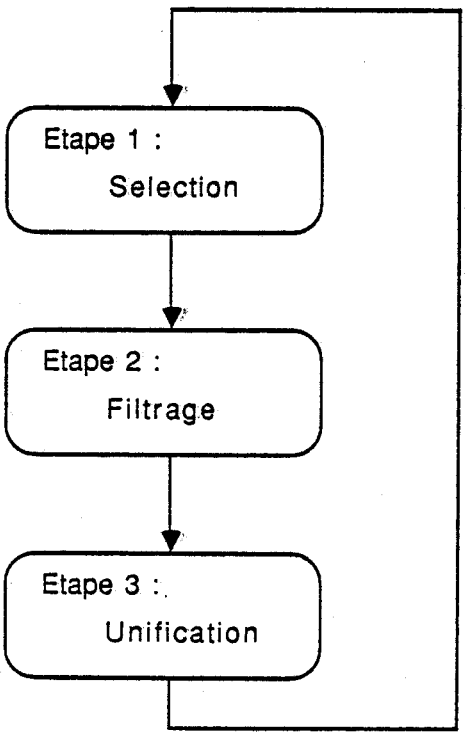


Figure IV - 30 : Algorithme général

Etape 1 : Sélection

Cette première étape consiste à déterminer, à partir de la base de données, une relation de type "graphe" contenant la structure de données "graphe-bd", qui va être comparée lors de l'étape filtrage qui suit.

La stratégie de parcours utilisée dans cette phase est identique à celle mise en oeuvre par Prolog, c'est-à-dire que l'espace renfermant l'ensemble des relations de type "graphe" est balayé séquentiellement de haut en bas.

Etape 2 : Filtrage

Pendant cette deuxième étape, l'algorithme de contrôle compare les structures de données graphe-bd (issue de l'étape de sélection) et graphe-var (graphe du RdP structuré décrit à l'aide de variables à instancier), selon trois critères combinés de la manière suivante :

si 1er critère : (nombre de processus du graphe-bd =
| nombre de processus du graphe-var)
| et
| 2ème critère : (nombre total des places du graphe-bd =
| nombre total des places du graphe-var)
| et
| 3ème critère : (nombre total de transitions du graphe-bd=
| nombre total de transitions du graphe-var)
|
ALORS
|
| La structure graphe-bd mérite d'être comparée avec graphe-var
| lors de l'étape de l'unification qui suit.
|
SINON
|
| Il y a échec
|
FSI.

Dans le cas d'échec, l'algorithme de contrôle opère un retour-arrière qui met en cause le choix courant de la relation "graphe" issue de l'étape de sélection. Cette technique de retour-arrière (à ne pas confondre avec chaînage-arrière) est réalisable grâce aux primitives fournies par Prolog et qui sont utilisables dans le corps des règles pour commander explicitement des retours-arrière.

L'avantage de cette méthode par filtres est qu'elle est simple, facile à mettre en oeuvre et peu coûteuse. Elle permet en très peu de temps de diminuer l'effort nécessaire pour rechercher un RdP structuré dans la base de données.

Etape 3 : Unification

Lors de cette dernière étape, l'algorithme de contrôle tente de comparer les deux structures de données graphe-bd et graphe-var.

Cette procédure consiste à effectuer des demi-unifications successives du graphe-bd (liste d'arbres dont les noeuds sont des constantes) avec graphe-var (liste d'arbres dont les noeuds sont des variables) ; on appelle demi-unification une unification de deux termes lorsque l'un des deux est instancié /CORD 86/.

En pratique, cette procédure tente de trouver, si elles existent, les valeurs que doivent prendre les variables présentes dans graphe-var pour que celle-ci coincide avec la structure graphe-bd. L'ensemble des égalités de la forme :

{p = <p1;l> ; t =<tr, k>}

qui rendent les deux listes d'arbres égaux est appelé substitution.

Sa mise en oeuvre est fondée sur le principe d'effacement d'arbres /COL 83/. En effet, cette méthode va tenter d'effacer la liste des n-uplet (un n-uplet représente un arbre codifié ou plus

précisément un processus) dans l'ordre où ils sont rangés dans la structure de données graphe-var, en transportant et en complétant, au fur et à mesure qu'elle avance dans ce travail, la substitution des variables de type {p ,t } qui permet cet effacement. En cas de succès, cette substitution constituera la réponse de cette procédure d'unification.

Pour expliquer ce mécanisme, considérons les deux structures de données graphe-bd et graphe-var décrivant chacun un système de n processus :

graphe-var = proc-var-1 . proc-var-2 . ... . proc-var-n . nil

graphe-bd = proc-bd-1 . proc-bd-2 . ... . proc-bd-n . nil

où

proc-var-i est un n-uplet dont les éléments sont des variables et représente le processus i ;

proc-bd-j est un n-uplet (arbre) dont les éléments (noeuds) sont des constantes et représente le processus j.

L'algorithme d'unification de deux graphes, basé sur le principe d'effacement, s'écrit comme suit :

pour unifier graphe-var et graphe-bd, il faut :

effacer (proc-var-1)

puis effacer (proc-var-2)

puis .

.

.

puis effacer (proc-var-n).

Cet algorithme possède deux conditions d'arrêts :

- soit tous les n-uplets proc-var-i (i = 1, ..., n) sont effacés : c'est un succès et la solution est matérialisée par la substitution qui a permis l'effacement.

Lorsque cette condition d'arrêt est réalisée, il est alors inutile de poursuivre la recherche dans la base de données et par conséquent, on provoque systématiquement l'arrêt du cycle de base de l'algorithme de recherche dans la base de données (celui de la figure IV-30).

- soit on bute sur un n-uplet qu'on ne peut effacer : c'est un échec. On en déduit alors que graphe-var et graphe-bd ne sont pas unifiables. En effet, partant de l'idée de base que : l'effacement d'un n-uplet proc-var-i n'est jamais remis en cause, l'algorithme d'unification de deux graphes n'opère pas de retour-arrière dès qu'il y a échec, pour remettre en cause les effacements des n-uplets précédents. On dit que cette stratégie est de type irrévocable (irrévocable contro régime) /NIL 80/.

Lorsque cette condition d'arrêt est vérifiée, l'algorithme de recherche dans la base de données revient systématiquement à l'étape 1 (Sélection).

Cet algorithme d'unification de deux graphes de Petri est présenté sur le schéma de la figure IV-31 :

```
i <--- 0 :
Répéter
   |  ▪ i <--- i + 1 ;
   |  ▪ prendre le n-uplet proc-var-i de la structure de
   |    données graphe-var ;
   |  ▪ effacer (proc-var-i, échec) ;
   |
-------
jusqu'à (i = n) ou (échec = vrai) ;
```

FIGURE IV-31 : algorithme d'unification de deux graphes de Petri

Algorithme d'effacement d'un n-uplet "proc-var-i"

Il est décomposé a son tour en quatre phases donnant le cycle de base suivant (figure IV-32) :

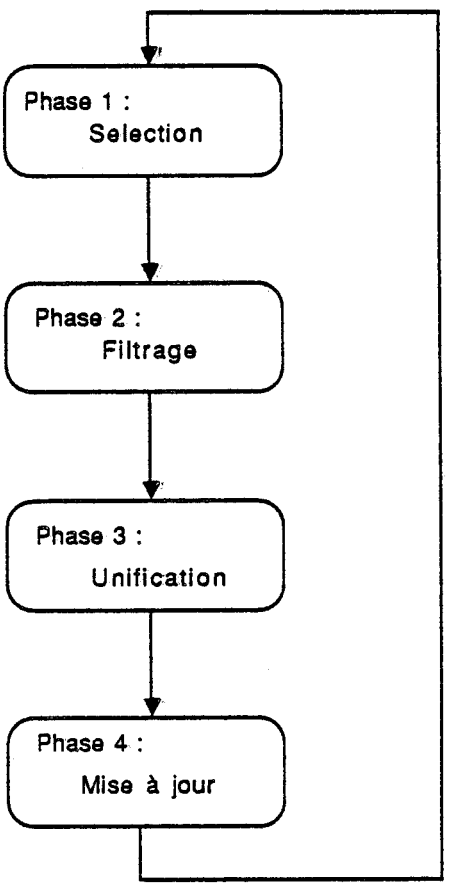


Figure IV - 32 : Effacement d'un n-uplet (processus)

Phase 1 : Sélection

------- ---------

Cette première phase consiste à déterminer, à partir de la structure de données graphe-bd, un n-uplet proc-bd-j qui a priori

mérite d'être traité lors de la phase 2 (filtrage) ; sinon, elle renvoie le résultat (échec = vrai) à l'algorithme d'unification de deux graphes (figure IV-31) lorsqu'il n'existe pas de processus proc-bd-j unifiable avec le processus proc-var-i.

La stratégie de parcours utilisée dans cette phase est basée sur le balayage séquentiel dans l'ordre de rangement des n-uplets dans la liste graphe-bd.

Phase 2 : Filtrage

Etant donné qu'un n-uplet décrivant un processus est de la forme suivante :

<nom du processus, description du graphe de processus sous-forme d'une liste de triplets, nombre de ses places, nombre de ses transitions>

l'étape de filtrage consiste à comparer proc-var-i et proc-bd-j selon deux critères décrits comme suit :

SI 1er critère : (nombre de places du processus proc-var-i =
| nombre de transitions du processus proc-bd-j)
| et
| 2ème critère : (nombre de transitions du processus proc-var-i
| =nombre de transitions du processus proc-bd-j)
|
ALORS
|
| Le n-uplet proc-bd-j peut être comparé avec le n-uplet
| proc-var-i lors de la phase 3.
|
SINON
|
| Il y a échec et, dans ce cas, l'algorithme (figure IV-32)
| opère un retour-arrière vers la phase 1 (sélection).
|
FSI

Phase 3 : Unification

Nous avons vu (au chapitre III, § 3), que le graphe d'un processus est représenté sous-forme d'une liste de triplets. Par conséquent, cette troisième phase a pour but d'unifier la liste "triplets-var-i" (description du graphe du processus proc-var-i) avec la liste triplets-bd-j (description du graphe du processus proc-bd-j).

La méthode que nous avons préconisée tente de trouver une permutation des triplets de la liste triplets-var-i déterminant ainsi une nouvelle configuration unifiable avec la liste triplets-bd-j.

Ce traitement nous conduit à tirer deux conclusions posssibles :

- soit que la construction de la nouvelle configuration de la liste triplets-var-i est achevée et qu'elle s'unifie avec la liste triplets-bd-j, on déduit alors l'égalité des deux processus proc-var-i et proc-bd-j matérialisée par la substitution trouvée.

Lorsque ce cas est réalisé, l'algorithme passe à la phase 4.

- ou bien, il n'existe pas de configuration susceptible de représenter la liste triplets-var-i unifiable avec la liste triplet-bd-j, par conséquent, on conclut que les deux processus proc-var-i et proc-bd-j ne sont pas identiques.

Dans ce cas, l'algorithme met en cause le choix de proc-bd-j et opère un retour-arrière vers la phase 1 (sélection).

Phase 4 : Mise-à-jour

Cette dernière phase de l'algorithme d'effacement de processus consiste à mettre à jour la structure de données graphe-bd en supprimant le processus proc-bd-j qui a été unifié

avec le processus proc-var-i.

En outre, cette étape, à la fin de son exécution, redonne le contrôle à l'algorithme d'unification de graphes (figure IV-32) et lui retourne le résultat (échec=faux).

Variante de l'algorithme de comparaison de graphes de Petri

Pour chercher un RdP structuré dans la base de données, par application de l'algorithme d'unification présenté sur la figure IV-30, l'utilisateur doit fournir au système toutes les composantes de son graphe de commande (processus et graphe de liaisons).

Dans le cadre de la modélisation de la commande de systèmes industriels de grande taille, cet algorithme sera caractérisé par un coût prohibitif en temps de recherche dû à sa nature combinatoire et en occupation de places mémoire de l'ordinateur.

Aussi est-il intéressant de proposer une variante de cet algorithme qui permet une recherche rapide et une occupation optimisée de l'espace mémoire. Cette variante s'appuie, non pas sur le RdP structuré complet, mais uniquement une partie restreinte de ce réseau qu'on appelle "signature du réseau".

Cette dernière, fournie par l'utilisateur et que nous notons "graphe-var-bis", peut être considérée comme un sous-réseau qui caractérise d'une manière spécifique un RdP structuré (concept proche de la notion de "clé unique" dans un fichier à organisation séquentielle indexée).

Pour mettre en oeuvre cette technique, nous avons principalement modifié la phase filtrage de l'algorithme de comparaison de graphes de Petri (présenté sur la figure IV-30).

Cette étape se réécrit comme suit :

SI 1er critère : (nombre de processus du graphe-var-bis ≤
| nombre de processus du graphe-bd)
| et
|
| 2ème critère : (nombre de places du graphe-var-bis ≤
| nombre de places du graphe-bd)
| et
|
| 3ème critère : (nombre de transitions du graphe-var-bis ≤
| nombre de transitions du graphe-bd)
|
ALORS
|
| La structure de données graphe-bd, issue de l'étape de sélection
| peut être comparée avec graphe-var-bis lors de l'étape
| unification
|
SINON
|
| C'est un échec
|
FSI

Remarque

L'algorithme d'unification de deux processus présenté sur la figure IV-31, peut être appliqué à des graphes de Petri généraux.

Vers l'amélioration des performances de l'algorithme de recherche dans la base de données.

Nous avons vu précédemment que l'algorithme de recherche d'un RdP structuré (figure IV-30) possède une stratégie de parcours qui lui permet de comparer successivement un RdP structuré fourni par l'utilisateur avec chaque élément appartenant à l'ensemble des relations de type "graphe".

Cette stratégie de parcours séquentiel qui caractérise cet algorithme conduit, lorsque la base de données est importante, à des temps d'exécution prohibitifs.

En effet, le temps moyen de recherche d'un élément est proportionnel au nombre de réseaux archivés dans la base de données.

Pour améliorer les performances de cet algorithme, nous envisageons de développer une technique qui permet d'effectuer des recherches uniquement dans un espace restreint de relations de type "graphe".

Cette technique consiste à concevoir une représentation interne efficace de l'espace mémorisant les relations de type "graphe".

Cette représentation interne a pour effet de structurer cet ensemble de relations en les groupant en classes.

Pour effectuer cette structuration, nous proposons deux solutions possibles développées ci-après.

Première solution :

- - - - - - - - -

Elle est exécutée lors du processus de création d'un RdP structuré et permet d'intégrer ce dernier dans une classe de relations selon le résultat d'une certaine fonction d'évaluation et de sélection.

Chaque classe est caractérisée par un nom de relation qui est différent de celui des autres classes.

L'ensemble des relations d'une même classe est représenté en mémoire par un paquet.

L'ensemble des paquets forme le même module ou monde en mémoire active de la machine.

Les critères d'évaluation et de sélection de la fonction peuvent être a priori identiques à ceux de l'étape filtrage de l'algorithme de la figure IV-30. Par conséquent, on peut comparer cette fonction d'évaluation à une fonction "H-coding" qu'on trouve en Informatique classique.

Pour illustrer plus clairement notre propos, imaginons la structuration de l'espace renfermant des relations de type "graphe" suivante (figure IV-33) :

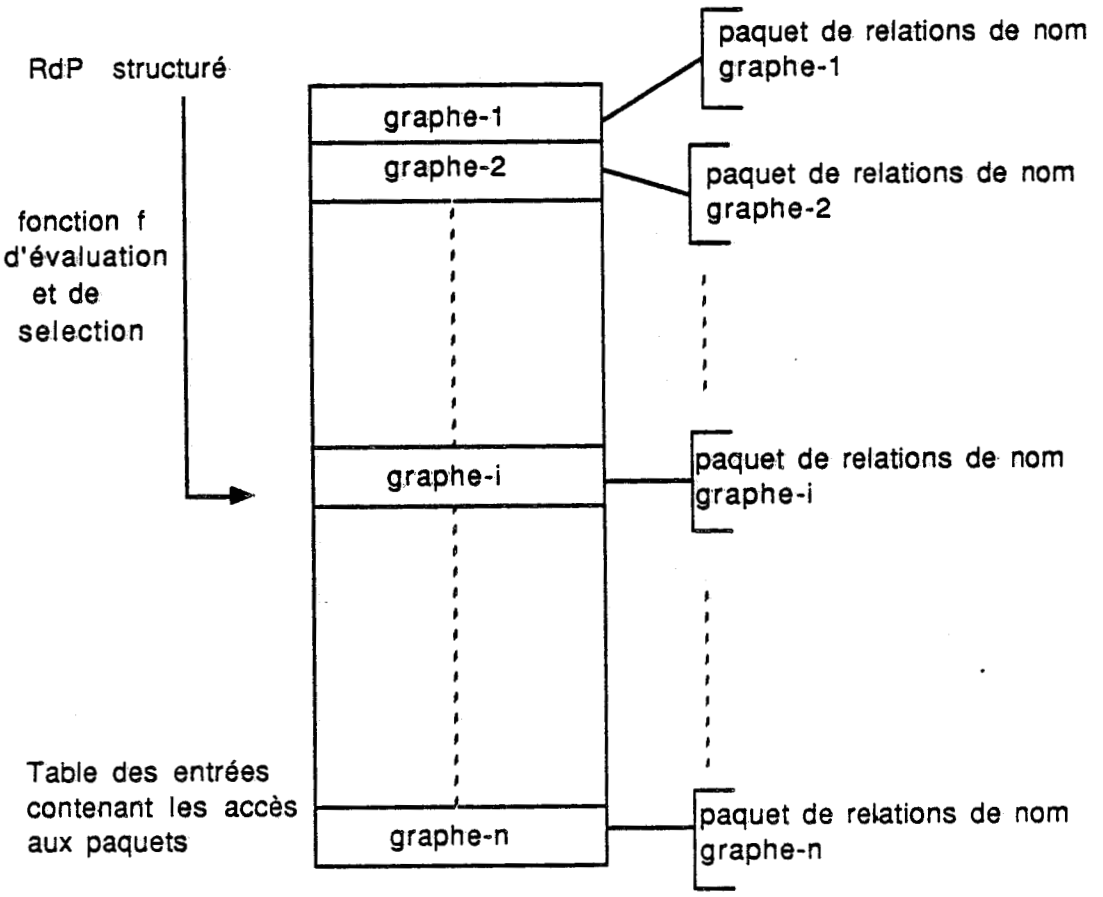


Figure IV - 33 : Structuration de l'espace des relations en mémoire active

Dans cette première solution, l'algorithme de recherche d'un RdP structuré "graphe-var" dans cet espace structuré (figure IV-33) consiste :

- En premier lieu, à exécuter la fonction d'évaluation et de sélection sur la structure de données "graphe"-var". Son résultat permet de sélectionner le nom du paquet en question.

- En second lieu, à parcourir séquentiellement ce paquet, qui représente un espace restreint de réseaux, en comparant la structure de données "graphe-var" avec chacun de ses éléments.

Il est clair que cette solution, qui est sujette à des améliorations, permet un gain de temps de recherche très appréciable (temps équivalent aux méthodes de recherche par H-coding).

Seconde solution :

- - - - - - - - -

A l'inverse de la première, cette seconde solution ne segmente pas l'espace des relations de type "graphe" en paquets ; elle le garde totalement homogène, c'est-à-dire un seul paquet (toutes les relations ont le même nom "graphe") dans un même module ou monde.

Mais, pour améliorer les performances de l'algorithme de recherche, elle utilise un arbre de classification.

Cet arbre possède la structure suivante (fiugure IV-34) :

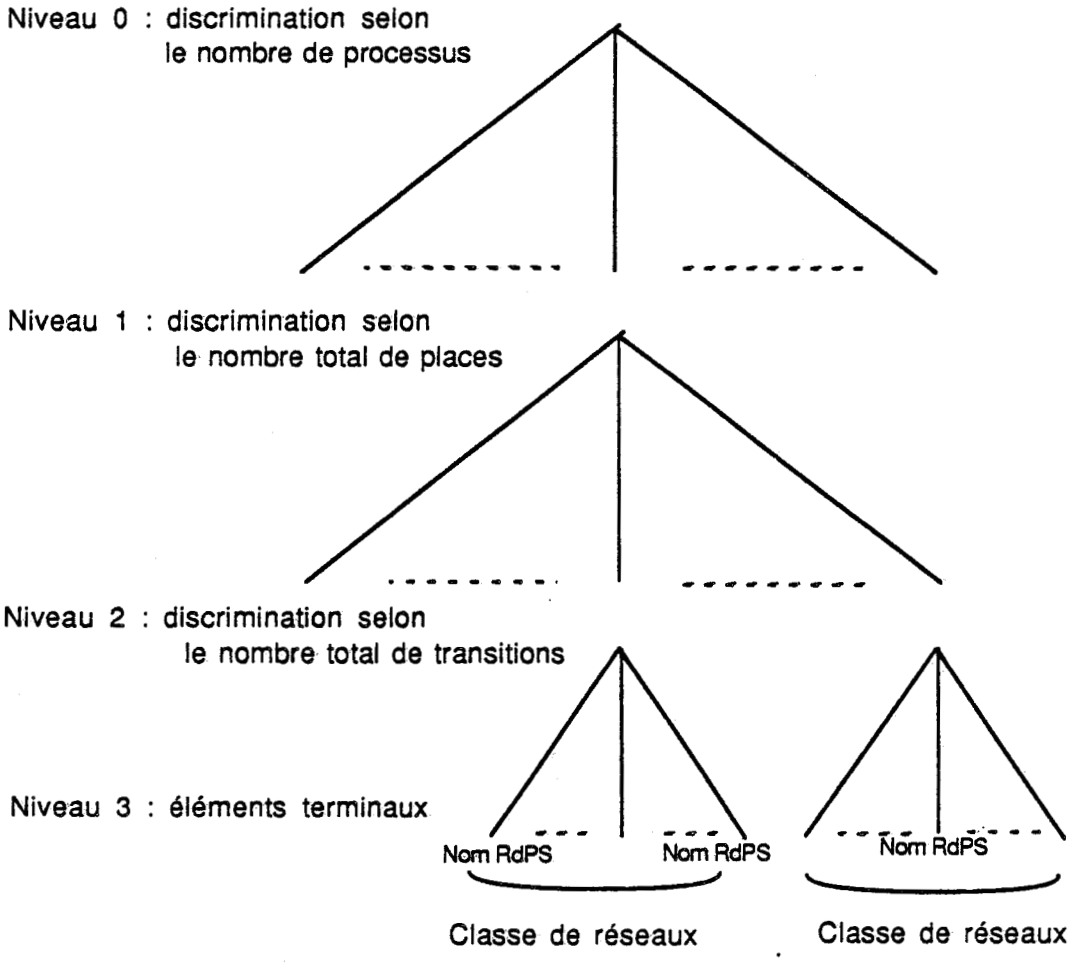


Figure IV - 34 : Arbre de classification

Une classe de réseaux est une liste d'accès dont chaque élément est un nom d'un RdP structuré appartenant au module "graphes" (monde présenté sur la figure IV-28).

Cet arbre de classification est construit, enrichi au fur et à mesure que l'utilisateur crée un nouveau RdP structuré, en

intégrant le nom de celui-ci dans une classe bien précise qui est caractérisée par les résultats des différents critères de discrimination.

Dans ce type de structuration, l'algorithme de recherche d'un RdP structuré "graphe-var" consiste :

- En premier lieu, à rechercher, dans l'arbre de classification, la classe où en principe le RdP structuré graphe-var doit appartenir. Ce traitement qui est identique à un algorithme classique de recherche dans un arbre, consiste à sélectionner les différentes branches (ou niveaux) de l'arbre selon les résultats de l'évaluation des critères de discrimination sur la structure de données graphe-var.

- En second lieu, lorsque sa classe est déterminée, à balayer séquentiellement le sous-arbre représentant cette classe en traitant chacun de ses éléments.

L'amélioration des performances de l'algorithme de recherche selon cette seconde proposition est liée essentiellement à la structure de l'arbre de classification. En effet, le temps de recherche se détériore si la dispersion des réseaux dans les différentes classes est mal équilibrée, c'est-à-dire qu'il existe une grande disparité en nombre d'éléments entre les classes.

Pour éviter ces inconvénients, il faut donc que les critères de discrimination permettent d'uniformiser et d'équilibrer les

classes de l'arbre de classification.

Cette remarque concernant l'amélioration des performances est également valable pour la première solution.

5.3.2.3 - Intégrité

Plus la masse d'informations enregistrées dans la base de données est grande, plus le risque que la donnée enregistrée soit erronée par rapport à la réalité est grand. Pour diminuer ce risque, le module de gestion de la base de données doit mettre en oeuvre des règles qui permettent de maintenir l'intégrité de la base de données.

Ces règles sont appelées contraintes, elles peuvent être classées en deux catégories : les contraintes statiques (Intégrité) qui permettent de valider un état de la base de données, et les contraintes dynamiques (Cohérence) qui permettent de valider les changements d'état de la base de données /ASI 85/, /CON 86/.

Les contraintes d'intégrité correspondent à des propriétés qui doivent toujours être vérifiées dans la base de données quelles que soient les RdP structurés enregistrés. Ces règles sont intégrées dans le compilateur du langage de spécification structuré. Elles sont de deux types :

- les règles qui définissent les domaines des arguments des relations de la base de données ;

- les règles qui spécifient les liens existants entre les arguments des cinq relations de la base de données.

Etant donné qu'un objet de la base de données représente une entité indépendante des autres, les opérations de création, suppression et modification appliquées à cet objet n'ont pas d'effets de bord sur les autres objets. Par conséquent, ces opérations préservent l'intégrité et la cohérence de la base de données.

5.4 - Conclusion

Le module de gestion de la base de données du système d'aide à la conception et à la validation de la commande d'un système de production industrielle, que nous avons présenté ici, est décomposé en quatre méta-fonctions :

- consultation de la base de données (BD) ;
- suppression de réseaux de la BD ;
- modification de réseaux de la BD ;
- archivage de réseaux dans la BD.

Les trois premières méta-fonctions ont en commun une opération fondamentale qu'est l'accès ou la recherche d'un RdP structuré dans la base de données. L'amélioration des performances de l'algorithme de recherche aura incontestablement des effets notables sur les performances générales du MGBD. Mais, ces améliorations prendront une autre dimension lorsque les problèmes suivants seront résolus :

1. Les performances actuelles des interpréteurs Prolog ne sont pas très bonnes ;

2. Les implantations actuelles de Prolog, à quelques exceptions près /DON 84/, ne permettent pas de manipuler directement des clauses en mémoire secondaire. L'ensemble des relations de la base de données (voir figure IV-28) et le module de gestion associé, doivent être résidant en mémoire active de la machine. Ceci est évidemment une grave lacune pour l'utilisation industrielle de Prolog. Des extensions sont donc nécessaires telles que /CON 86/ :

- l'enrichissement des primitives de gestion de clauses ;

- l'implantation de mécanismes d'accès efficaces à des clauses en mémoire secondaire (modularité, accès associatif et indexé).

6 - TROISIEMES MODULE : SUPERVISEUR

L'objet de ce dernier module est d'établir le dialogue entre l'utilisateur et les système de gestion de la base de connaissances.

Ce dialogue est établi grâce à un système de menus.

Pendant son utilisation, le système se trouve dans un des modes suivants :

- création,
- vérification/validation,
- paramétrage,
- consultation,
- suppression,
- archivage.

Chacun de ses modes correspond aux fonctionnalités d'une méta-fonction appartenant aux deux modules décrits précédemment (voir figure IV-2).

On accède à ces modes soit par le menu principal apparaissant au début de l'exécution, soit en cours d'exécution à l'aide de commandes.

Ce système de menus est décrit par le schéma suivant (figure IV-35) :

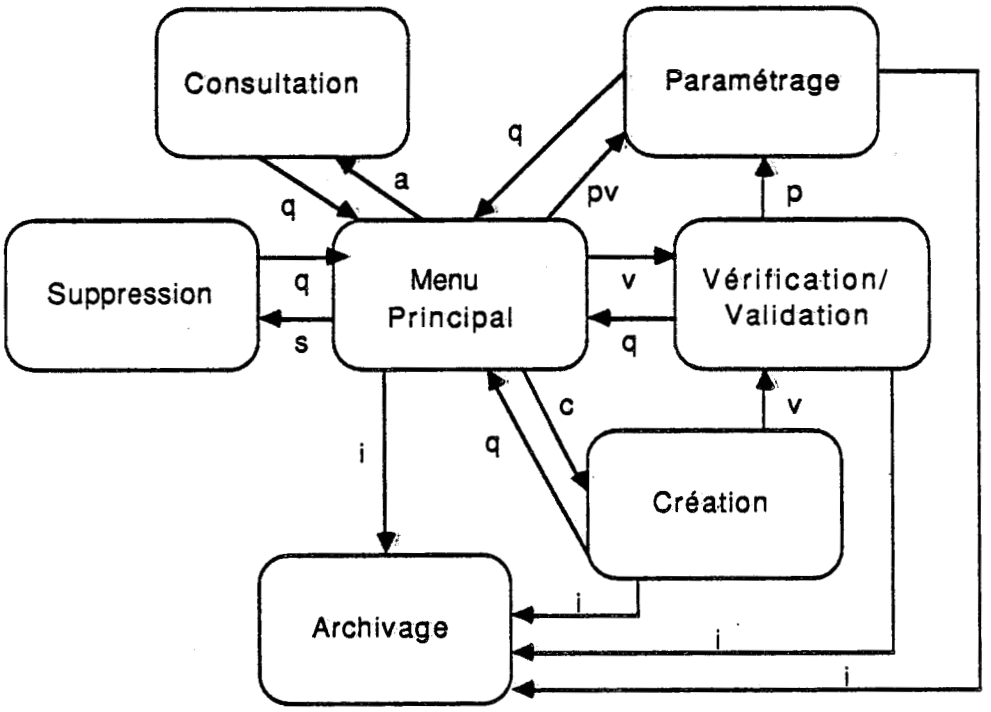


Figure IV - 35 : Structuration en menus

Chaque menu dispose de deux types de commandes :

1- les commandes internes au bloc qui après exécution offrent à nouveau le même menu ;

2- les commandes externes qui sont des sorties ou des retours vers d'autres blocs. Leur exécution entraîne donc l'apparition d'un nouveau menu correspondant au bloc d'arrivée.

A titre d'exemple, examinons les blocs suivants :

Menu principal

Ce menu permet d'effectuer l'enchaînement des opérations suivantes :

- transférer la base de données de la mémoire secondaire vers la mémoire active de l'ordinateur ;

- proposer à l'utilisateur les différents menus par sélecion, en tapant la commande correspondante ;

- et enfin, sauvegarder la base de données en la transférant de la mémoire active vers la mémoire secondaire de la machine.

Mode Création

C'est le mode choisi à partir du menu principal en tapant la lettre "C" (ou "c") pour activer le menu associé à la méta-fonction "Construction du graphe de commande". Dans ce mode, l'utilisateur, après avoir construit le RdP structuré modélisant ce graphe de commande, peut alors choisir une des possibilités proposées par le menu "Création".

Ces possibilités sont :

- le caractère "C" (ou"c") est une commande interne permettant à l'utilisateur de construire un nouveau RdP structuré ;

- le caractère "V" (ou "v") qui permet de passer au "vérification/validation" qui a pour but d'étudier les propriétés du RdP structuré construit ;

- le caractère "I" (ou "i") qui permet de passer au mode "archivage" consistant à archiver dans la base de données le RdP structuré nouvellement créé.

- et enfin, le caractère "Q" (ou "q") pour "quit" qui permet de passer au menu principal.

7 - CONCLUSION

Nous avons présenté, dans ce chapitre, la base de règles du système d'aide à la conception et à la validation de la partie commande d'un système de production industrielle.

Cette base de règles représente la partie action du système, elle fournit un ensemble d'outils permettant d'une part, de gérer la base de données et d'autre part, de mettre en oeuvre une démarche d'aide à la conception progressive de graphes de commande.

Par conséquent, de ces fonctionnalités découle la structure multimodulaire de cette composante active, c'est-à-dire sa décomposition en trois modules :

- modules d'aide à la construction et à la validation du RdP structuré qui modélise le graphe de commande ;

- module de gestion de la base de données ;

- module de supervision.

Pour illustrer notre démarche, nous avons, au cours de ce chapitre, appliqué cette méthodologie d'aide à la conception à un exemple illustratif de cellule flexible mettant en évidence une partie des possibilités offertes par le système.

CONCLUSION GENERALE

CONCLUSION GENERALE

Nous avons présenté dans ce mémoire, un outil d'aide à la conception et à la validation de réseaux de Petri structurés modélisant la commande de processus de production automatisée.

Cet outil possède deux composantes complémentaires portant respectivement sur le traitement symbolique et le traitement numérique.

Le système de traitement symbolique réalisé dans le langage Prolog, utilise les techniques informatiques de l'intelligence artificielle afin de proposer au concepteur une démarche progressive et modulaire d'aide à la conception des système de commande. Cette assistance à la modélisation repose sur :

- la définition d'un langage de spécification structuré (proche de l'utilisateur) associé au modèle graphique RdP structuré et assurant la cohérence du modèle ;

- la réalisation du compilateur de ce langage éléminant a priori une large gamme d'erreurs de conception ;

- une aide à la documentation ;

- la création et la gestion de la base de connaissances, qui archive l'expertise du concepteur, en utilisant principalement les techniques issues des systèmes de gestion de bases de données ;

- la réalisation d'algorithmes originaux d'unification de RdP structurés.

Le système de traitement numérique interfacé à la composante symbolique réalise la validation et la simulation de RdP structurés.

Ce logiciel qui a été conçu de manière interactive, permet de prendre en compte les développements futurs envisagés dans le cadre du L.A.I.I. de l'I.D.N. Dans cette perspective, nous travaillons actuellement sur les points suivants :

- prise en compte des extensions du modèle de base développées au L.A.I.I. :
  . les réseaux de Petri structurés adaptatifs,
  . les réseaux de Petri structurés adaptatifs colorés.

- la validation de ces modèles par simulation.

Concernant ce dernier point, nous avons engagé le développement d'un module d'édition et d'enrichissement graphique. Ce module constituera l'interface entre la méta-fonction paramétrage du logiciel d'aide à la conception et le simulateur développé par E. Castelain /CAS 87/. Il permet ainsi une intégration horizontale entre le simulateur et le système d'aide à la conception de graphes de commande de processus de production discrétisée.

ANNEXES

ANNEXE A

STRUCTURES DE DONNEES DE BASE

1 - Les arbres

Outre les constantes représentées par leurs noms, la structure la plus générale des objets manipulés est la structure d'arbre.

Plus généralement, un arbre A est constitué d'un noeud R appelé racine et d'un ensemble, éventuellement vide, ordonné d'éléments $A_1, A_2, \ldots, A_n$ qui sont eux mêmes des arbres (voir Figure 1 - a). Les racines de $A_1, A_2, \ldots, A_n$ sont les descendants de R. Tout noeud sans descendant est dit terminal ou feuille. Les arbres peuvent représenter toutes sortes de connaissances (voir exemples Figures 1 - b et 1 - c).

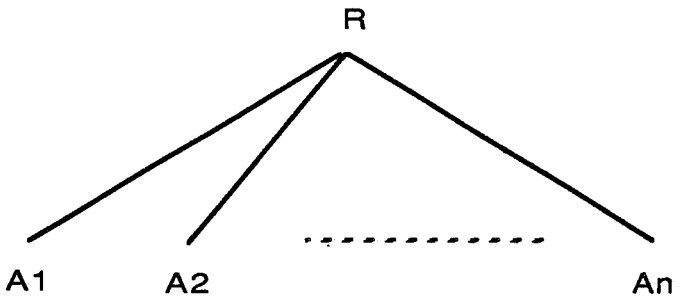
Si les arbres ont été choisis comme structures de données, c'est qu'ils sont suffisamment riches pour représenter des informations complexes, organisées hiérarchiquement. Ils sont également suffisamment simples à manipuler, aussi bien du point de vue informatique qu'algébrique.

Les arbres, représentés sous leur forme graphique, présentent des avantages indéniables de lisibilité, mais ne sont pas toujours d'un maniement trés aisé. Il est ainsi pratique d'utiliser une notation parenthésée dans laquelle tout arbre est codé en faisant suivre le nom de chaque noeud par la liste, entre parenthèses, de ses branches, séparées par des virgules (il s'agit en pratique de la notation préfixée).

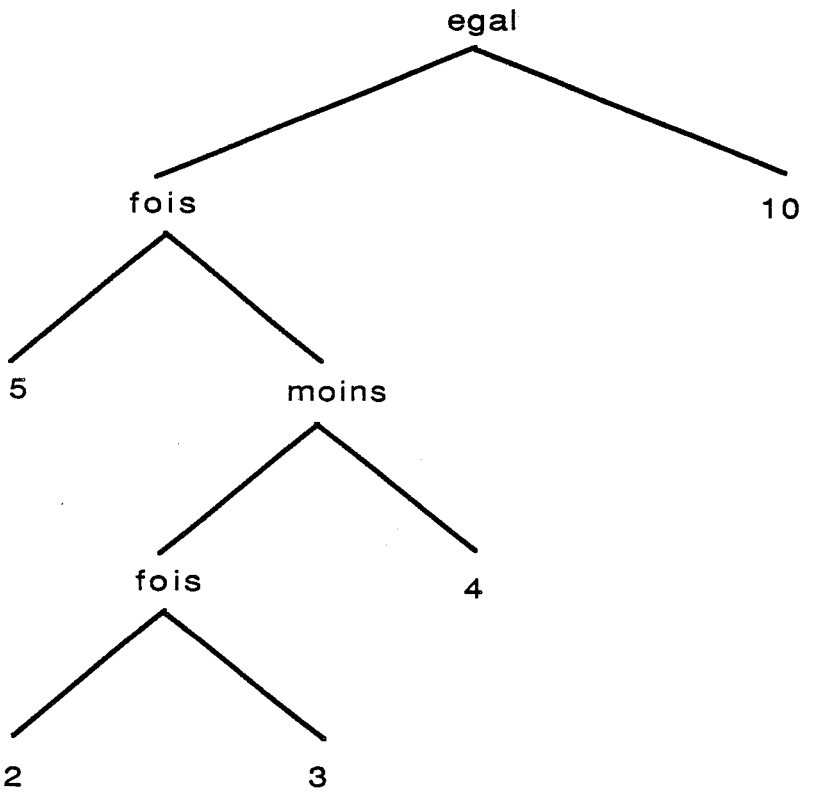
Ainsi, les arbres b et c de la figure 1 s'ecrivent-ils respectivement :

(b) egal (fois (5 , moins (fois (2 , 3) , 4)) , 10)

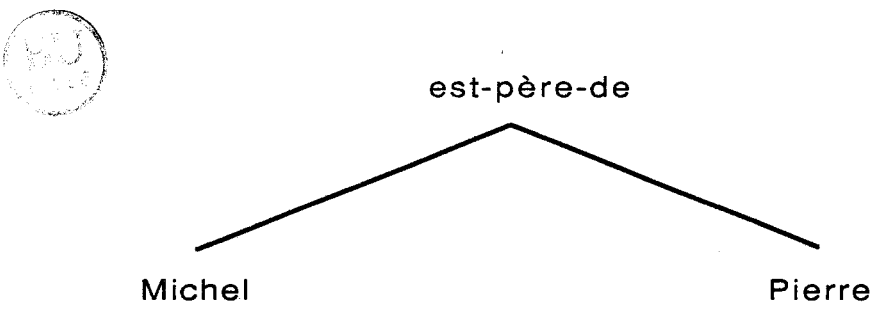
(c) est-père-de (Michel , Pierre)


R
A1
A2
An

(a)


egal
fois
10
5
moins
fois
4
2
3

(b)


est-père-de
Michel
Pierre

(c)

Figure 1

2 - Les listes

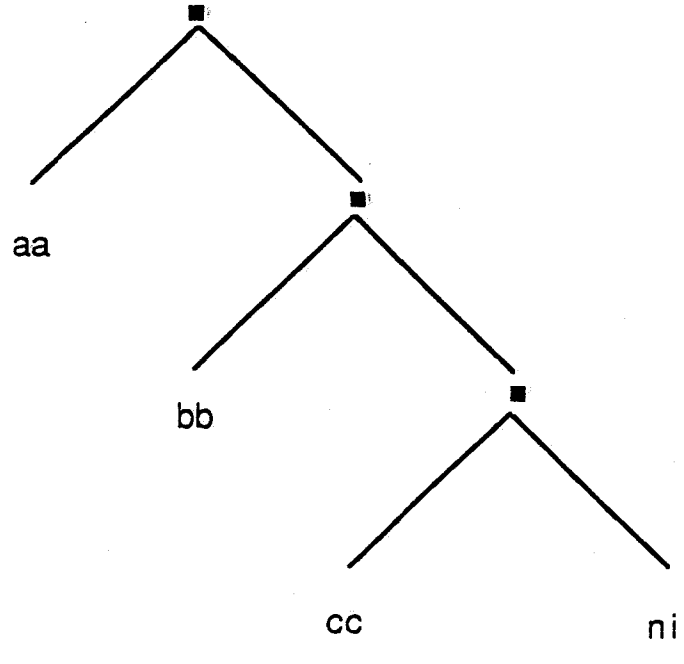
La liste est une structure de données très classique en programmation non numérique. Elle est constituée d'une suite ordonnée d'éléments de longueur quelconque.

Chaque élément de la liste peut être une constante, une variable, une structure ou une liste. De plus, les listes peuvent servir à représenter presque toutes les sortes de structures complexes que l'on peut souhaiter manipuler, elles sont très largement utilisées pour représenter des arbres, des grammaires, des graphes, etc.

Les listes peuvent se représenter comme un cas particulier d'arbre. Une liste est soit une liste vide, qui n'a pas d'éléments, notée **"nil"**, ou une structure à deux composantes : la tête et la queue sont reliées par le symbole fonctionnel **"."**, pour lequel l'association se fait de droite à gauche. Ainsi, une liste est formée de trois éléments aa, bb et cc peut s'écrire comme suit :

aa.bb.cc.nil

et se représente par :



ANNEXE B

Grammaire du langage de spécification des réseaux de Petri structurés

<réseau de Petri structuré> ::= <en-tête du RdpS>
<corps du RdPS>
<documentation sur le RdPS>
<fin du RdPS>

<en-tête du RdPS> ::= { **rdp** , **RdP** } <nom du RdPS>

<fin du RdPS> ::= { **frdp** , **FRDP** }

<documentation sur le RdPS> ::=

{ **commentaire** , **COMMENTAIRE** } <chaîne de caractères>

<corps du RdPS> ::=
<définition de nouveaux processus>
<suite éventuelle de définition-composition>
| <définition-composition de structures>

<définition de nouveaux processus> ::= <liste de processus>

<liste de processus> ::= <processus>
| <processus>
<liste de processus>

<processus> ::= <en-tête du processus>
<corps du processus>
<fin du processus>

<en-tête du processus> ::= { **proc** , **PROC** } <nom du processus>

<fin du processus> ::= { **fproc** , **FPROC** }

<corps du processus> ::= <énoncé composé>

<énoncé composé> ::= <action>
| <macro-place>

| <alternative>
| <répétitive>
| <bloc>

<action> ::= { **action** , **ACTION** } <nom de l'action>

<macro-place> ::= { **macro** , **MACRO** } <nom de la macro-place>

<alternative> ::= <en-tête de l'alternative>
<énoncé composé>
<fin de l'alternative>

<en-tête de l'alternative> ::= { **si** , **SI** }
<condition>
{ **alors** , **ALORS** }

<fin de l'alternative> ::= { **fsi** , **FSI** }
| { **sinon** , **SINON** }
<énoncé composé>
{ **fsi** , **FSI** }

<répétitive> ::= <en-tête de la répétitive>
<énoncé composé>
<fin de la répétitive>

<en-tête de la répétitive> ::= { **tq** , **TQ** }
<condition>
{ **faire** , **FAIRE** }

<fin de la répétitive> ::= { **ftq** , **FTQ** }

<bloc> ::= <en-tête du bloc>
<énoncé composé>
<fin du bloc>

<en-tête du bloc> ::= { **bloc** , **BLOC** } <nom du bloc>

<fin du bloc> ::= { **fbloc** , **FBLOC** }

<suite éventuelle de définition-composition> ::=

<définition-composition de structures>

| <composition de structures>
| <vide>

<définition-composition de structures> ::=

<définition de structures prédéfinies>
<composition de structures>

<définition de structures prédéfinies> ::=

<héritage d'un réseau prototype>
| <héritage d'un réseau prototype>
<définition de structures prédéfinies>

<héritage d'un réseau prototype> ::=

<création du lien prototype-instance>
<renommer les processus du réseau prototype>

<création du lien prototype-instance> ::=

{ **herite** , **HERITE** } <nom de l'instance> { **de** , **DE** } <nom du prototype>

<renommer les processus du réseau prototype> ::=

<renommer un processus>
<renommer les processus du réseau prototype>
| vide

<renommer un processus> ::=

<redéfinir le nom du processus>
<redéfinir les noms des blocs du processus>

<redéfinir le nom du processus> ::=

{ **rename-proc** , **RENAME-PROC** }
(<ancien nom du processus prototype> ,
<nouveau nom du processus instance>)

<redéfinir les noms des blocs du processus> ::=

{ **rename-bloc** , **RENAME-BLOC** } (<renommer les blocs>)
| vide

<renommer les blocs> ::=

<renommer un bloc>
| <renommer un bloc> , <renommer les blocs>

<renommer un bloc> ::= (<ancien nom du bloc "prototype"> ,
<nouveau nom du bloc "instance">)

<composition de structures> ::=

<liste de liaisons entre structures>

<liste de liaisons entre structures> ::=

<liaison entre processus>
| <liaison entre processus>
<liste de liaisons entre structures>

<liaison entre processus> ::=

{**signal**, **SIGNAL**} <liaison signal mémorisé sans accusé de réception>
| { **sigacc** , **SIGACC** } <liaison rendez-vous avec accusé de réception>
| { **sema** , **SEMA** } <liaison exclusion mutuelle>
| { **pc** , **PC** } <liaison producteur/consommateur>
| { **sema-old** , **SEMA-OLD** } <déclaration de sémaphore-old>
| { **pc-old** , **PC-OLD** } <déclaration du producteur/consommateur-old>

<liaison signal mémorisé sans accusé de réception> ::=

(<nom de l'emetteur> , <nom du récepteur>)

<liaison rendez-vous avec accusé de réception> ::=

(<nom de l'emetteur> , <nom du récepteur>)

<liaison exclusion mutuelle> ::=

(<nom de la ressource> , <nombre d'exemplaires de la ressource> ,
<liste d'utilisateurs de la ressource>)

<liaison producteur/consommateur> ::=

(<nom de la ressource> ,
<nombre d'exemplaires de la ressource dans l'etat producteur> ,
<nombre d'exemplaires de la ressource dans l'etat consommateur> ,
<liste des processus producteurs> ,
<liste des processus consommateurs>)

<déclaration de sémaphore-old> ::=

(<nom de la ressource partagée> , <nouvelle valeur initiale> ,
<liste des nouveaux processus utilisateurs>)

<déclaration du producteur/consommateur-old> ::=

(<nom de la ressource prod/cons> ,
<nouvelle valeur initiale de la place dans l'etat producteur> ,
<nouvelle valeur initiale de la place dans l'etat consommateur> ,
<liste des nouveaux processus producteurs> ,
<liste des nouveaux processus consommateurs>)

BIBLIOGRAPHIE

BIBLIOGRAPHIE

/ADI 85/ Agence De l'Informatique
"Les systèmes experts et leurs applications"
5ème Journées Internationales, Avignon 13-15 Mai.

/ADI 86/ Agence De l'Informatique
"Les systèmes experts et leurs applications"
6ème Journées Internationales, Avignon 28-30 Avril.

/ALA 84/ ALANCHE P., BENZAKOUR K., DOLLE F., GILLET P., RODRIGUEZ P., VALETTE R.
"PSI : a Petri-net Based Simulator for Flexible Manufacturing Systems".
5th workshop of Petri-nets, Aarhus, Juin 84.

/ASI 85/ ASIRELLI P., DE SANTIS M., MARTELLI M.,
"Integrity Constraints in Logic Databases".
The Journal of Logic Programming, 1985:3, pp 221-232.

/BARB 83/ BARBERYE G., JOUBERT T., MARTIN M.
"Manuel d'utilisation de Prolog/P"
Note technique NT/PAA/CLC/LSC/1058.

/BARI 85/ BARILLIER G.,

"Automates multifonctions : pour une approche structurée des automatismes"

Conference Automation 85, Avril 1985, Paris.

/BAR 86/ BARR A., FEIGENBAUM E.A.

"Le manuel de l'intelligence artificielle" Tome 1.

Eyrolles.

/BEL 85/ BEL G., BERRADA M., CORREGE M., DUBOIS D.

"SIMULISP : un banc d'essai de systèmes experts pour la commande des ateliers de production"

Afcet, Congrès Automatique, Toulouse, Octobre 1985.

/BER 83/ BERTHELOT G.,

"Transformations et analyses de réseaux de Petri. Applications aux protocoles".

Thèse d'Etat, Paris, 1983.

/BON 84/ BONNET A.

"L'intelligence artificielle, promesses et réalités"

Inter-Editions.

/BOU 86/ BOUREY J.P., CORBEEL D., CRAYE E., GENTINA J.C.
"Adaptative and Coloured Structured Petri-nets for Description, Analysis and Synthesis of Hierarchical Control and Reliability of Flexible Cells in Manufacturing Systems"
First European Workshop on "Fault Diagnostics, Reliability and Related Knowledge-based Approaches" Ile de Rhodes (Grèce).

/BRA 83/ BRAMS G.W.
"Réseaux de Petri : théorie et analyse"
Tome 1, Masson.

/BRA 83/ BRAMS G.W.
"Réseaux de Petri : théorie et pratique"
Tome 2, Masson.

/CAS 85/ CASTELAIN E., CORBEEL D., GENTINA J.C.
"Comparative Simulations of Control Processes Described by Petri-nets"
COMPINT' 85, Montreal (Canada), pp 530-532.

/CAS 87/ CASTELAIN E.
"Modélisation et simulation interactive de cellules de production flexibles dans l'industrie manufacturière"
Thèse de Docteur de l'Université, Lille.

/CHO 85/ CHOURAQUI E.
"Modèles information de raisonnements"
Bulletin du CID, n° 2, pp 35-46.

/CLA 82/ CLARK K.L., TARNLUND S.A.
"Logic Programming" Academic Press.

/CLA 85/ CLARK K.L., Mc CABE F.G.,
"Micro-Prolog" : Programme en Logique"
Eyrolles.

/CLO 84/ CLOCKSIN W.F., MELLISH C.S.
"Programmer en Prolog"
Eyrolles.

/COL 83/ COLMERAUER A., KANOUI H., VAN CANEGHEM M.
Prolog : bases théoriques et développements actuels". TSI, vol 2, n°4.

/CON 86/ CONDILLAC M.,
"Prolog, fondements et applications"
Dunod Informatique

/COR 79/ CORBEEL D.
"Schéma de cablage et schéma de contrôle. Application à la simulation et à la gestion de processus industriels"
Thèse de Doctorat de spécialité, Lille.

/COR 83a/ CORBEEL D., VERCAUTER C., GENTINA J.C.
"Spécifications et conception des systèmes de conduite de processus temps réels".
Proc. of MIMI' 83, Lugano (Suisse), pp. 36-39.

/COR 83b/ CORBEEL D. VERCAUTER C., GENTINA J.C.
"Réduction et propriétés structurelles des réseaux de Petri structurés".
Proc. of MIMI' 83, Lugano (Suisse), pp 40-43.

/COR 83c/ CORBEEL D., VERCAUTER C., GENTINA J.C.
"Généralisation des réseaux de Petri"
A.I. 83, Lille, Actes vol. III, pp 201-206.

/COR 84/ CORBEEL D., VERCAUTER C., GENTINA J.C.
"Adaptative Petri-nets for Real Time Applications"
Proc. of Digitec' 84, IMACS, Patras (Grèce).

/COR 85a/ CORBEEL D., VERCAUTER C., GENTINA J.C.
"Application of an Extension of Petri-nets to Modelization of Control and Production System"
6th European Workshop on App. and Theory of Petri-nets, Finland, pp 53-74.

/COR 85b/ CORBEEL D., VERCAUTER C., GENTINA J.C.
"Modéllisation homogène du graphe de contrôle d'un système de conduite de processus industriels".
AFCET, Congrès Automatique 1985, Toulouse, pp 517-533.

/CORD 86/ CORDIER M.O., FALLER B., ROUSSER M.C.
"Optimisation de l'opération «Pattern Matching» dans les systèmes experts"
TSI, Vol 5, n° 3.

/COU 86/ COURVOISIER M., VALETTE R.
"Systèmes d'exploitation des micro-ordinateurs concepts et systèmes dominants"
Dunod.

/COU 87/ COURVOISIER M., ATABAKHCHE H., SIMONETTI BARBALHO D., VALETTE R.,
"Commande d'ateliers : un compromis est-il possible entre une approche graphique et une approche intelligence artificielle". APII à paraître.

/DEL 82/ DELOBEL C., ADIBA M.
"Bases de données et systèmes relationnels"
Dunod Informatique

/DEV 84/ DEVIENNE P.
"Interface entre la base de données IDS II et Prolog"
Journées d'études sur les systèmes experts et leurs applications, ADI, Avignon 84.

/DON 84/ DONZ Ph., HURTADO R.,
"Le langage D. Prolog, initiation au langage de la 5ème génération"
Edi-tests.

/ESE 86/ Ecole Supérieure d'Electricite
"Systèmes experts : automatisation et maintenance industrielle"
Journées d'études à Gif-sur-Yvette, 26 Novembre 86.

/FAR 85/ FARRENY H.

"Les systèmes experts, principes et exemples"

CEPADUES- Editions.

/FRA 87/ FRACHET J.P., POUGET J.P., SAIDI A.

"Modeling the Know-how of the Automation Expert in a Context of a C.A.O. of Automatic Industrial Machines for Discontinuous Production".

IMACS ; AI, Expert Systems and Languages in Modelling and Simulation.

Barcelone 2-4 Juin 1987.

/GAL 78/ GALLAIRE H., MINKER J.

"Logic and Database" Plenum Press

/GAN 85/ GANASCIA J.G.

"La conception des systèmes experts"

La Recherche n° 170, Octobre 1985.

/GIA 85/ GIANNESINI F., KANOUI H., PASERO R., VAN CANEGHEM M.

"Prolog" Inter-Editions

/GIR 84/ GIROD C.

"Sur la conception et la réalisation d'un logiciel de simulation de réseaux de Petri".

Thèse de Docteur-ingénieur, I.D.N., Lille.

/GRI 86/ GRIFFITHS M.

"Intelligence artificielle, techniques algorithmiques"

Hermes Publishing.

/GUI 85/ GUIDEZ G.

"Le Grafcet : macro-représentation et structuration du traitement des automatismes"

Conférence AUTOMATION' 85, Paris, Avril 1985.

/HACK 75a/ HACK M.

"Petri-net Language"

MIT Computation Structure Group, Memo 124.

/HACK 75b/ HACK M.

"Decision Problems for Petri-nets and Vector Addition Systems"

Mac. Tech. Memo 59, MIT.

/HOL 85/ HOLLINGER D.,

"Utilisation pratique des réseaux de Petri dans la conception des systèmes de production"

TSI, Vol. 4, n° 6.

/KAR 85/ KARFIA A. CORBEEL D., GENTINA J.C.

"Consultation d'une base de données de réseaux de Petri"

Intelligencia, 21-24 Avril 1985, Paris.

/KAR 86/ KARFIA A., CORBEEL D., GENTINA J.C.
"Conception automatisée d'un système de conduite de procédés industriels".
Convention Automatique Productique, 28-30 Mai 1986, Paris.

/LAUR 84/ LAURENT J.P.
"La structure de contrôle dans les système experts"
TSI, vol 3, n° 3.

/LAU 82/ LAURIERRE J.L.
"Représentation et utilisation des connaissances"
TSI, vol 1, n° 1 et 2

/LAU 86/ LAURIERE J.L.
"Intelligence artificielle, résolution de problèmes par l'homme et la machine"
Eyrolles.

/LI 84/ LI D.,
"A Prolog Database System"
Research Studies Press Ltd.

/LLO 85/ LLOYD J.W., TOPOR R.W.
"A Basis for Deductive Database Systems"
The Journal of Logic Programming, 1985:2, pp 93-109.

/NGU 84/ NGUYEN G.T., OLIVARES J., WENNINGER P.
"Coopération entre Prolog et d'un SGBD généralisé : principes et applications".
Actes du séminaire sur la programmation en Logic,
CNET Lannion, Avril 1984.

/NIC 83/ NICOLAS J.M., YAZDANIAN K.
"Un aperçu de BDGEN : un SGBD déductif"
Conférence IFIP' 83.

/NIL 80/ NILSSON N.
"Principles of Artificial Intelligence"
Palo-Alto, CA : Tiogo-Publishing Company.

/PIP 85/ PIPARD E.,
"Détection des contradictions dans les bases de connaissances" ADI,
5ème journées Internationales sur "les systèmes experts et leurs applications"
Avignon, 13-15 Mai 1985.

/PRU 85/ PRUNET F., LLORCA P., STURLESE J.L., CAZALOT C., ALANCHE P., SALVI.P.
"Méthodologie et outil de conception assistée de spécification progressive d'un automatisme complexe en Grafcet étendu"
Afcet, Congrès Automatique, Toulouse, Octobre 85.

/TSI 85/ Technique et Science Informatique
"Spécial Réseaux de Petri"
Vol. 4, n° 1.

/VALK 78b/VALK R.
"On the Computational Power of Petri-nets"
MFCS, Lect. Notes in Computer Sc. n° 64, Springer, Berlin, pp 526-535.

/VALK 81/ VALK R.
"Generalization of Petri-nets"
MFCS, Lect. Notes in Computer Sc. n° 118, Springer, Berlin, pp 140-155.

/VALR 78a/VALR R.
"Self-modifying Nets, a Natural Extension of Petri-nets"
IGALP, Lect. Notes in Computer Science, n° 62, Springer, Berlin, pp 464-476.

/VER 82/ VERCAUTER C.
"Sur un ensemble d'outils d'aide à la spécification et à la conception des systèmes industriels".
Thèse de troisième cycle, Lille.

/JAZ 86/ JAZMI R.
"Traitement d'erreurs dans la phase d'édition d'un Réseau de Petri structuré".
Rapport de D.E.A. de Productique, I.D.N., Lille.

/3IP 85/ OMEGA, Notice de présentation, Société pour l'Innovation, l'Informatique Industrielle et la Productique, Paris.

RESUME

La prise en compte des fondements de l'analyse et de la programmation structurées 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 oeuvre, la structuration introduite au niveau des réseaux de Petri, permet la définition d'une méthodologie efficace de description des systèmes de commande.

Le travail présenté dans ce mémoire a pour but la conception et la réalisation en langage Prolog, d'un système à base de connaissances. Ce dernier est un outil d'aide à la conception et à la validation de la partie commande d'un système de processus industriels décrite par réseaux de Petri structurés.

Le logiciel, fonctionnant en mode interactif, propose à l'utilisateur une démarche progressive et modulaire lui permettant de valider chaque étape de modélisation en utilisant les structures fonctionnelles déjà synthetisées et archivées dans la base de données.

MOTS-CLEFS :

BASE DE CONNAISSANCES
BASE DE DONNEES
BASE DE REGLES
LANGAGE PROLOG
PRODUCTION FLEXIBLE
PARTIE COMMANDE
RESEAUX DE PETRI