N° d'ordre : 93

50376
1987
127

THÈSE

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

Emmanuel CASTELAIN

MODELISATION ET SIMULATION INTERACTIVE DE CELLULES DE PRODUCTION FLEXIBLES DANS L'INDUSTRIE MANUFACTURIERE

Soutenue le 26 Février 1987 devant la Commission d'Examen

Membres du Jury :	M. COURVOISIER	Rapporteur
	M. TOULOTTE	Rapporteur
	M. GENTINA	Examinateur, Directeur de Thèse
	M. CORBEEL	Examinateur
	M. COMYN	Examinateur
	M. THUREL	Invité
	M. LE GALLAIS	Invité

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 COURVOISIER, Professeur au L.A.A.S. de Toulouse,
Monsieur COMYN, Professeur à l'I.U.T. d'Informatique de LILLE I,
Monsieur CORBEEL, Maître de Conférence à l'I.D.N.,
pour l'honneur qu'ils me font en participant à mon Jury de Thèse.

Je suis flatté de la présence à ce Jury, de Monsieur THUREL et de Monsieur LE GALLAIS, de la Société TELEMECANIQUE et je tiens tout particulièrement à les en remercier.

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.

Je tiens à remercier le C.N.R.S. et la Région Nord/Pas-de-Calais pour la co-financement de la Bourse qu'ils m'ont attribuée pour la durée de cette étude.

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

S O M M A I R E

CHAPITRE I

"Modélisation des Systèmes de Production Flexible"

pages

o

o o

CHAPITRE II

"Simulation des Systèmes
de Production Flexible"

pages

o

o o

CHAPITRE III

"Application à un exemple"

INTRODUCTION GENERALE

L'atelier flexible est un concept original dont les premières réalisations datent des années 60 (Sundstrand Aviation à Rockford aux U.S.A. et IBM à Deptford en Grande-Bretagne en 1967). Le succès actuel des ateliers flexibles est dû, en grande partie, aux progrès technologiques des dernières années avec l'apparition de machines outils à commande numérique de plus en plus sophistiquées, de robots de plus en plus performants, de systèmes de convoyage de plus en plus adaptés (chariots filoguidés, convoyeurs et divers systèmes de palettisation).

Pourtant, 20 ans après ces premières réalisations et malgré les récents progrès techniques, le bilan actuel est loin d'être optimiste. La vème Conférence Internationale sur les ateliers flexibles s'est pratiquement soldée par un constat d'échec. Les ateliers flexibles promettent une réduction des coûts d'environ 50 % par rapport aux systèmes conventionnels, par réduction des encours et des délais de production, par augmentation des taux d'engagement des machines, ... Mais, dans la réalité, les grands ateliers flexibles d'usinage paraissent difficiles à rentabiliser. Certains même, ont failli techniquement et économiquement.

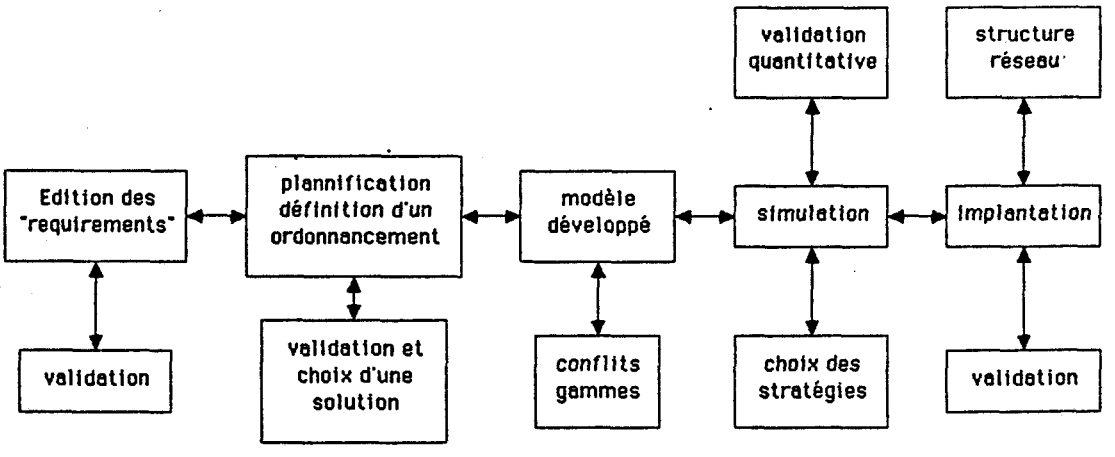
Le taux élevé de l'investissement nécessaire constitue le premier obstacle auquel se heurte l'automatisation de la production. Un atelier flexible coûte, au minimum 5 MF et en moyenne 50 MF avec un retour d'investissement en général supérieur à 3 ans. De plus, le coût de l'investissement est en général sous-estimé au départ. En effet, l'expérience acquise en ce domaine est insuffisante pour pouvoir anticiper les prix.

D'autre part, il n'existe pas de méthode d'approche globale permettant la mise au point d'un atelier flexible. Chaque réalisation est le résultat d'une étape particulière. La complexité du matériel utilisé, combinée au manque de standardisation des machines comme des logiciels induit une phase de développement considérable et un investissement en matière grise non négligeable. Il n'est pas rare, au Japon, de consacrer 100 000 heures d'étude pendant 2 ans pour concevoir un atelier flexible de 18 machines.

Plus particulièrement, la mise au point du système informatique de gestion de l'atelier flexible représente entre 25 et 40 % du coût total, ce qui est bien sûr un coût moyen puisque, par définition, il s'agit d'un système "sur mesure". Dans ce domaine également, les méthodologies de conception utilisées par les industriels ont évolué moins vite que leurs outils technologiques (systèmes de gestion temps réel, automates programmables à évolutions parallèles, ...).

En France, en particulier, le GRAFCET, qui s'est imposé dans les milieux industriels, depuis une dizaine d'années comme outil normalisé de conception, se révèle insuffisant dans le cadre des systèmes flexibles. Disposer d'un outil de représentation utilisable lors d'une démarche CAO apparaît donc comme un besoin bien réel.

Le projet CASPAIM (Conception Assistée de Systèmes de Production Automatisée en Industrie Manufacturière), développé au Laboratoire d'Informatique Industrielle de l'I.D.N., constitue un effort de synthèse qui permettra d'assister la démarche de conception des systèmes de production, depuis la définition du cahier des charges jusqu'à la phase finale d'implantation.



Le logiciel de simulation que nous présentons dans ce mémoire constitue l'une des charnières essentielles de ce projet. Son utilisation à plusieurs niveaux dans l'élaboration d'un système de production permet la conception et l'écriture des variantes stratégiques du niveau décisionnel, la validation du comportement d'ensemble, et par l'évaluation des performances des divers modèles retenus, fournit des critères de choix importants pour sélectionner la solution la plus adaptée. Dans ce sens, nous avons orienté notre travail afin de satisfaire les contraintes suivantes :

1. Cohérence de l'approche. L'outil proposé permet d'appréhender tous les aspects d'un système de production flexible :
   - la partie commande,
   - la partie décisionnelle,
   - le procédé.

2. Intéractivité dans l'exploitation du simulateur afin d'assister la démarche de conception.

3. Validation qualitative et quantitative du comportement (notamment par l'analyse des performances).

4. Homogénéïté de l'approche quel que soit le niveau de mise en œuvre lors des différentes étapes de la conception.

Ce mémoire est divisé en 2 parties. La première partie concerne les modèles nécessaires à la conception d'un système de production flexible. Ces modèles seront présentés plus particulièrement dans l'optique d'une simulation. La seconde partie concerne la description du simulateur et sa mise en œuvre sur les modèles définis précédemment.

CHAPITRE I

MODELISATION DES SYSTEMES DE PRODUCTION FLEXIBLE

INTRODUCTION

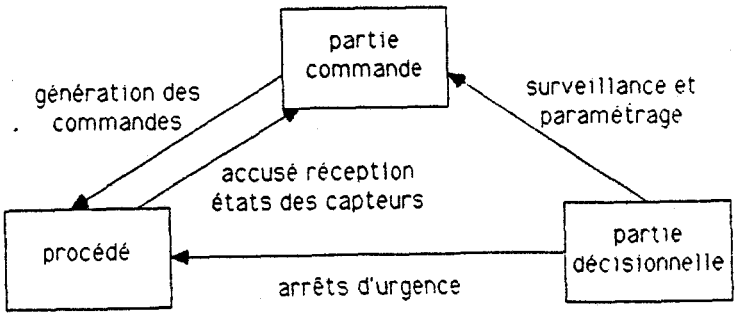
Un système de production flexible est un ensemble complexe que l'on peut scinder a priori en 3 parties :

1. Un ensemble de dispositifs matériels (robots, convoyeurs, centres d'usinage, unités de stockage, etc ...) où sont conditionnés les produits à usiner. Il s'agit du **procédé**.

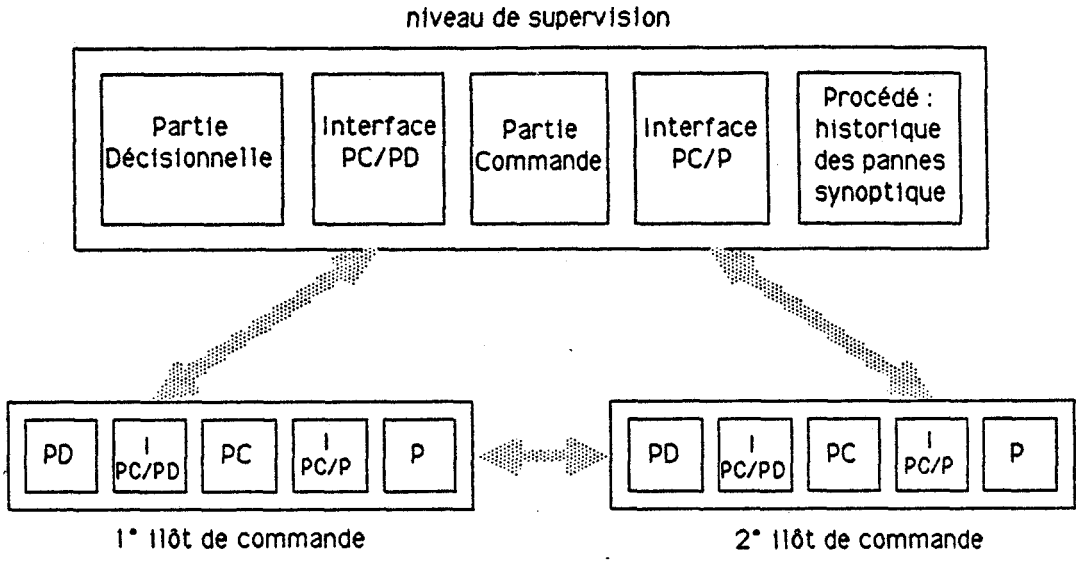
2. La **partie commande** est l'ensemble des processus pilotant le procédé. C'est en général la partie informatique du système, implantée sur automates programmables ou tout dispositif permettant la gestion temps réel de processus physiques.

3. La **partie décisionnelle**, dont l'importance est variable selon le degré de flexibilité du système, paramètre la partie commande en intégrant les objectifs de production, les consignes extérieures et, éventuellement, des algorithmes complexes dont le résultat est nécessaire à l'évolution du système.

Le "bon fonctionnement" du système réel de production provient de la coopération de ces 3 sous-ensembles. Nous avons décrit sur le **schéma conceptuel** ci-dessous, certaines interactions entre ces 3 parties.



La phase d'analyse et d'évaluation du système s'appuie sur ce schéma conceptuel. Cependant, afin de garantir, dès la phase de conception, l'adéquation entre le système conçu et le système à implémenter, il est intéressant, voire indispensable, de prévoir la décomposition fonctionnelle du système, en tenant compte en particulier de son architecture répartie. Dans cette optique, le schéma conceptuel est précisé selon un **schéma logique** qui sera, par exemple, le suivant.



Dans cet exemple, l'architecture du système est composé de 2 ilôts de production possédant chacun leur système de commande et reliés entre eux par un ensemble de liaisons de synchronisation nécessaire à leur coopération. Le niveau de supervision, dont le rôle est d'organiser la coopération entre ces 2 ilôts, assure par ailleurs, l'animation d'un synoptique et la mise à jour d'un historique des pannes du système, assimilés ici à un procédé particulier associé à ce niveau de supervision.

Enfin, la dernière phase avant implantation consiste à définir le **schéma physique** du système en précisant, en particulier, le matériel à mettre en œuvre et la nature des liaisons à réaliser : choix des machines, d'un réseau local, ...

Ces 3 étapes, qui s'inscrivent dans une démarche cohérente de conception, ne sont donc pas indépendantes, surtout si le cahier des charges impose des contraintes telles que : coût maximal du système, intégration d'un matériel déjà existant, ... Une bonne définition du

schéma conceptuel de base est fondamentale dans l'analyse du projet. D'où l'importance de la modélisation des 3 domaines de ce schéma conceptuel (partie commande, partie décisionnelle et procédé).

Une démarche classique de la modélisation consiste à privilégier l'un de ces 3 domaines par utilisation d'un modèle spécifique. Le système "OVIDE" du Groupe SYSECA, utilise les réseaux de Petri généralisés comme outil de représentation de la partie commande. Ce logiciel permet l'édition graphique puis l'analyse des propriétés essentielles du graphe. Le langage SLAM permet la simulation d'un système en vue de l'élaboration de statistiques de fonctionnement. Dans ce cas, c'est la description du procédé qui est privilégiée.

Dans la phase de conception d'un système, l'intérêt de tels outils est limité puisqu'ils ne permettent pas d'appréhender globalement les 3 aspects du problème.

Une 2ème approche consiste à étendre le pouvoir de description du modèle privilégié utilisé afin d'y intégrer les autres aspects du système envisagé. La temporisation des réseaux de Petri, par exemple, permet d'intégrer la dynamique induite par le procédé au sein du modèle de la partie commande /CHR 83/.

La 3ème solution consiste à interfacer plusieurs modèles spécifiques, chaque modèle décrivant un aspect particulier du système. Dans "Simulisp" (logiciel de **SIM**ulation **U**tilisant le langage **LI**sp pour la commande des **S**ystèmes de **P**roduction /BEL 85/), les parties commande et décisionnelle sont représentées par des règles de production alors que la description du procédé utilise un langage de simulation (SLAM, SIMAN) ou un simulateur dédié (PARSIFAL), ceci dans l'optique d'une validation par simulation, des systèmes de commande de "5ème génération".

L'approche que nous avons adoptée s'apparente à celle de Simulisp dans le cas plus classique où la partie commande est implantée sur des supports informatiques habituels (automates programmables, commandes numériques, ...). Nous utiliserons donc 3 modèles, chacun décrivant l'un des aspects du système. Le modèle de la partie commande consiste en une extension des réseaux de Petri, et celui de la partie décisionnelle, en un ensemble de règles de production. L'intégration

du procédé est réalisée à 2 niveaux, soit par temporisation du graphe de commande, soit grâce à un modèle spécifique basé sur l'utilisation de catégories génériques.

La première partie de ce mémoire décrit les modèles adoptés pour représenter un système de production flexible en vue d'une simulation, et les interfaces permettant la mise en correspondance de ces différents modèles. Le dénominateur commun de cet outil hybride de représentation d'un système (et de sa simulation) consiste en l'utilisation de Le_Lisp comme langage de base commun aux 3 modèles.

I - MODELISATION DE LA PARTIE COMMANDE

I.1 - Exemple introductif

Afin de préciser le domaine d'application considéré (la Productique) et les besoins du modèle nécessaire pour décrire et analyser de tels systèmes, considérons l'exemple de l'alimentation d'un tour à commande numérique par un robot manipulateur.

Le système est composé de 2 machines indépendantes possédant chacune leur propre système de commande et fonctionnant a priori en parallèle et de façon complètement asynchrone.

Prises indépendamment les unes des autres, les tâches du robot et du tour sont essentiellement séquentielles. Il s'agira pour le robot, de se positionner, d'ouvrir sa pince, d'assurer le chargement, puis d'opérer le trajet vers le lieu de destination.

On voit aisément que l'objectif fixé dépend de la coopération du robot et du tour. Avant de serrer sa poupée mobile, le tour doit attendre que le robot ait correctement positionné la pièce à usiner. Avant de déserrer sa pince, le robot doit attendre que la poupée mobile serre effectivement la pièce. Cette coopération n'est possible qu'au travers d'un mécanisme de synchronisation entre ces 2 processus initialement asynchrones.

Les réseaux de Petri (RdP) ont été développés afin de modéliser ces concepts d'actions asynchrones et concurrentes. Il est donc naturel de les utiliser comme outils de base pour modéliser les systèmes de production flexible. Cependant, cet outil est insuffisant pour traduire la complexité d'un système flexible. Après un bref rappel de la définition formelle des RdP, nous développerons, au travers de notre exemple introductif, les extensions existantes permettant d'accroître leur pouvoir de modélisation. Nous préciserons ensuite le modèle que nous proposons d'adopter.

I.2 - Les réseaux de Petri /BRA 83/ /TSI 85/

Ce modèle graphique représente le comportement dynamique d'un système discret composé de 2 ensembles finis : les places et les transitions. Les places sont associées au concept d'état du système et les transitions aux évènements dont l'occurence modifie l'état du système.

Définition :

Un RdP est un quadruplet R = <P,T,pré,post>
où P est l'ensemble fini des places,
T est l'ensemble fini des transitions (disjoint de P),
pré : P × T ⟶ $\mathbb{N}$ application d'incidence avant,
post : P × T ⟶ $\mathbb{N}$ application d'incidence arrière.

Au réseau R est associé le marquage M : P ⟶ $\mathbb{N}$. Ainsi, M(P) désigne le nombre de marques contenues dans la place P.

Les transitions permettent de modifier le nombre de marques contenues dans les places. La dynamique du réseau se définit comme suit.

I.2.1 - Activation d'une transition

Une transition t est **activable** pour un marquage M ssi :

$\forall$ p $\in$ P, M(p) $\geq$ pré(p,t).

C'est la précondition de franchissement de t;

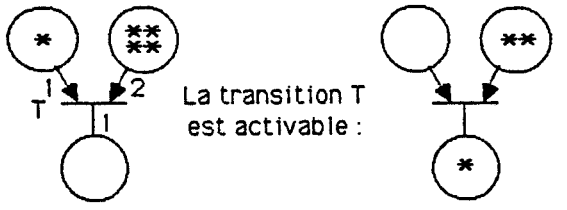
Le marquage M résultant du franchissement (ou **tir**) de la transition t (relation que l'on note M(t>M') se définit comme suit :

$\forall$ p $\in$ P M'(p) = M(p) - pré(p,t) + post(p,t)

En terme de graphe, cette dynamique peut s'interpréter de la façon suivante.

La transition t est activable si et seulement si chaque place p en amont de t contient un nombre de marques supérieur à la valuation de l'arc amont (p,t). Le franchissement de t consiste alors à enlever, de chaque place p en amont de t, un nombre de marques égal à la valuation de l'arc (p,t) et à ajouter, pour chaque place p' en aval de t, un nombre de marques égal à la valuation de l'arc (t,p').

Exemple :



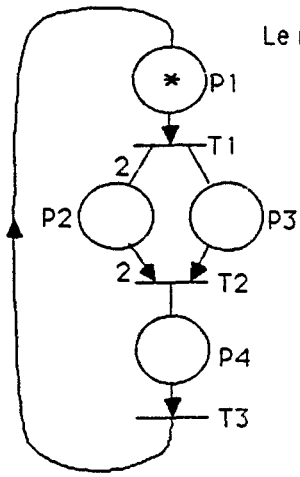
I.2.2 - Notations matricielles

Soit n_p le nombre de places du réseau et n_t le nombre de transitions.

Le marquage M peut être représenté par un vecteur de dimension n_p, pré et post par des matrices de dimension (n_p,n_t). On définit la matrice d'incidence C représentant l'application de :

$$P \times T \longrightarrow Z$$
$$(p,t) \longrightarrow C(p,t) = post(p,t) - pré(p,t)$$

Exemple :



Le marquage initial M0 = $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $\begin{matrix} P1 \\ P2 \\ P3 \\ P4 \end{matrix}$

La matrice d'incidence

$$C = \begin{bmatrix} -1 & 0 & +1 \\ +2 & -2 & 0 \\ +1 & -1 & 0 \\ 0 & +1 & -1 \end{bmatrix} \begin{matrix} P1 \\ P2 \\ P3 \\ P4 \end{matrix}$$

$\quad T1 \quad T2 \quad T3$

I.2.3 - Grammaire associée à un réseau de Petri

Le RdP est représenté par un système de réécriture sur l'alphabet des noms des places.

Le marquage initial constitue l'axiome de départ et chaque transition correspond à une ou plusieurs règles de réécriture.

Le système de réécriture de l'exemple présenté (§ I.2.2) est le suivant :

M_o : $\{p_1\}$

t_1 : $p_1 \longrightarrow p_2^2 p_3$

t_2 : $p_2^2 p_3 \longrightarrow p_4$

t_3 : $p_4 \longrightarrow p_1$

I.3 - Les RdP structurés

I.3.1 - La structuration des RdP : une limitation du modèle

L'apport de la structuration dans le cadre de la programmation a amélioré notablement la conception des grands programmes. Calquer cette démarche méthodique pour l'utilisation des RdP comme modèle de représentation de la commande des systèmes industriels, outre l'héritage des avantages de la structuration relatifs à l'appréhension des

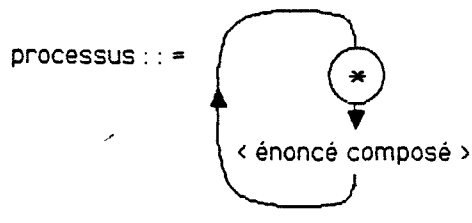
grands systèmes, se justifie d'autant plus que le graphe de commande conçu est destiné, dans sa finalité, à être implémenté sur les divers organes physiques de conduite du système. Or, cette implémentation s'opère grâce aux langages de programmation des supports informatiques de pilotage (armoire de commande numérique, automates programmables, ...). La structuration a priori du modèle RdP permettra donc une traduction plus aisée et plus fiable lors de l'écriture des programmes.

L'introduction de la structuration au niveau des RdP conduit à la définition d'une classe particulière et plus restrictive des RdP : les RdP structurés. Elle permet d'assurer une meilleure correspondance entre le modèle et son cahier des charges et de limiter les erreurs de conception. Le modèle de RdP structurés adopté /COR 79-80/ 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.

I.3.2 - Le modèle structuré

a) Représentation modulaire :

La notion de processus correspond ici au concept de "programme" en informatique, c'est-à-dire à un enchaînement de tâches.



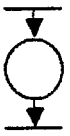
Un processus est défini comme la combinaison séquentielle de 3 structures élémentaires : l'action, l'alternative et la répétitive. En utilisant une notation de type Backus-Naur, on introduit récursivement la notion de **"bloc"**.

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

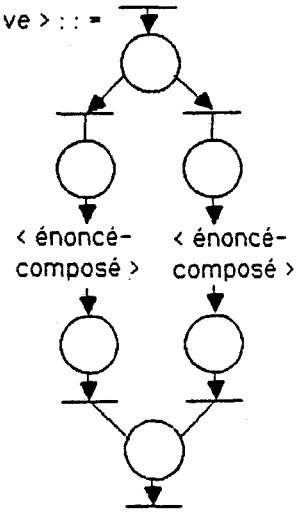
< énoncé-composé >

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

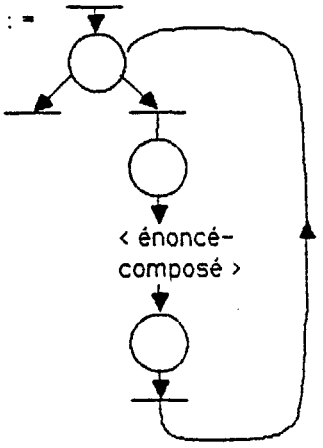
< action > : : =



< alternative > : : =



< répétitive > : : =



La notion de bloc permet de regrouper un enchaînement séquentiel de tâches et de lui associer un nom. Notons que la définition récursive des blocs introduit des contraintes sur l'imbrication des blocs : en particulier, 2 blocs ne peuvent pas se "chevaucher".

Un système de commande est composé de n processus parallèles. Le découpage de ce système en n processus dépend à la fois de contraintes de décentralisation fonctionnelle, visant à augmenter le parallèlisme du système, et de contraintes d'implantation répartie.

La représentation modulaire issue de cette décomposition en processus présente 3 avantages :

. La définition de structures fonctionnelles types, utilisées dans des applications différentes, autorise l'archivage de telles structures dans une base de données lors d'une démarche CAO de la conception du système. Cette approche est réalisée dans /KAR 86/.

. La modification d'un ou plusieurs processus reste localisée à ces processus, ce qui évite la redéfinition complète du système.

. La partition d'un système en sous-systèmes relativement indépendants, autorise une conception progressive du graphe de Petri. Cette démarche est également facilitée par la définition récursive des blocs composant les processus, puisqu'elle permet l'affinement progressif de la structure des blocs.

Exemple : Le bloc "usinage" d'un processus modélisant la commande d'une machine-outil opérant 2 types de traitements, peut être défini, dans une lère phase de spécification, comme un simple bloc "action" (Fig. 1), puis développé ultérieurement (Fig. 2).

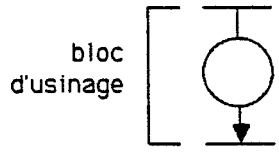


FIGURE 1

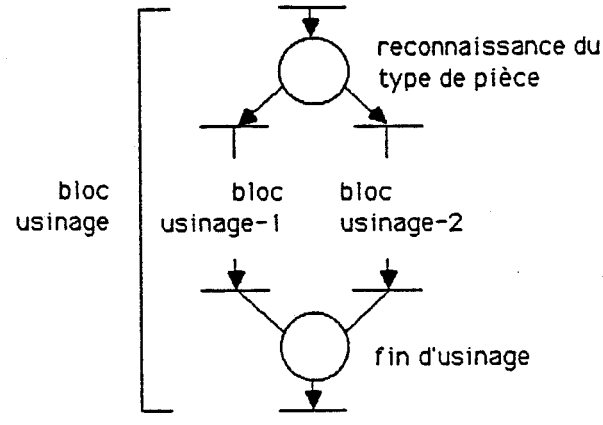
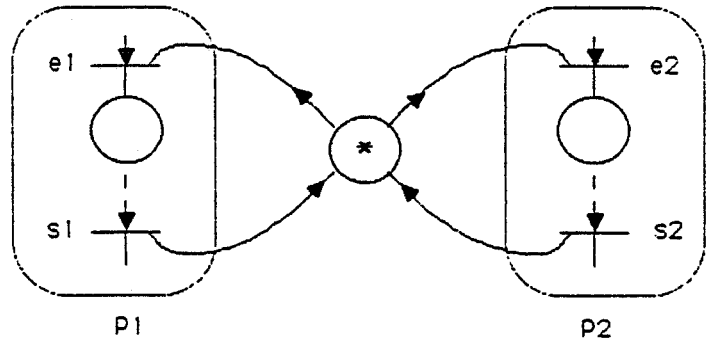


FIGURE 2

b) Définition des intéractions entre processus :

L'exemple introductif (§ I.1) a montré l'importance des liaisons nécessaires à la coopération des divers processus d'un même système. Ces structures de communication ont été banalisées. Les 3 types de liaisons retenus permettent de représenter la plupart des intéractions entre processus et minimisent les risques de blocage inhérents à leur définition.

*** L'exclusion mutuelle :**

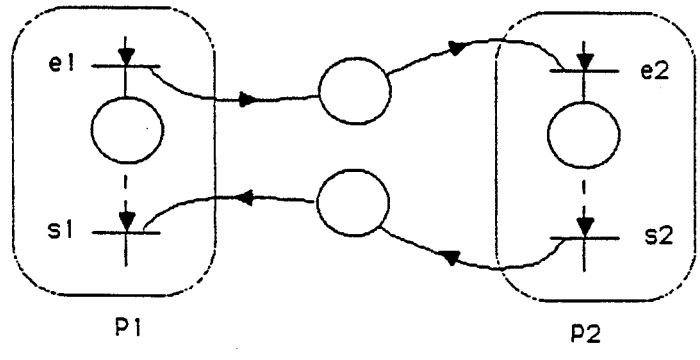


e_i : transition d'entrée dans la section critique

s_i : transition de sortie de la section critique

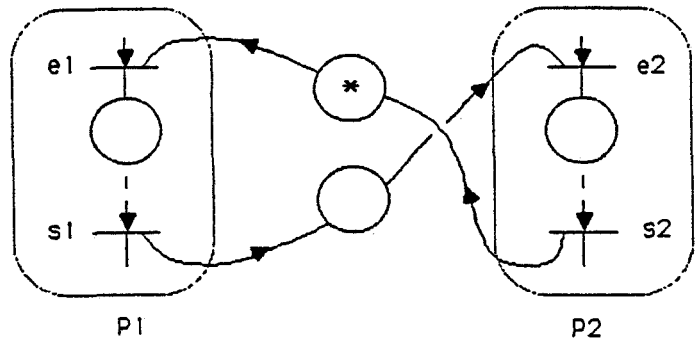
La structure d'exclusion mutuelle assure l'unicité d'accès à une section critique partagée par plusieurs processus. Dans le cadre des systèmes de production, une telle structure permet de préserver un et un seul mode de commande quand le pilotage d'un organe opératif peut être activé sélectivement par plusieurs processus de commande.

*** La synchronisation** (avec accusé de réception) :



Le processus P_1 synchronise le processus P_2.

*** Le producteur / consommateur :**

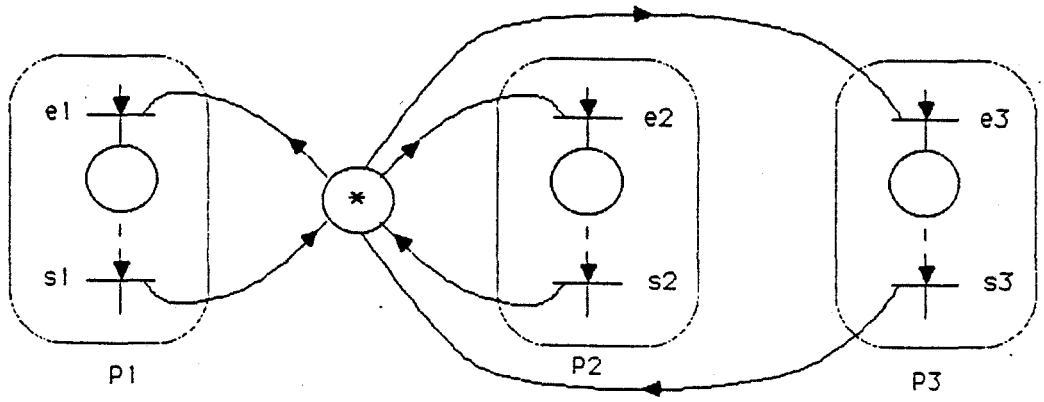


Le processus producteur P_1 produit des marques qui sont consommées par le processus consommateur P_2.

Dans le domaine qui nous intéresse (la productique), les processus envisagés modélisent le conditionnement d'une pièce circulant dans l'atelier. La liaison entre 2 processus consécutifs selon la gamme d'usinage d'une pièce est modélisée selon les cas par une structure de producteur / consommateur ou par un mécanisme de synchronisation.

Remarque : Par construction, les processus structurés sont **saufs**, c'est-à-dire que le marquage de toute place appartenant à un processus, est au plus égal à 1. Par contre, les places de liaison peuvent posséder éventuellement un marquage supérieur à 1.

Exemple : partage de 2 ressources entre 3 processus.



c) Localisation des liaisons :

Vue d'un processus, la définition d'une liaison fait apparaître un bloc de liaison délimité par les transitions d'entrée e_i et de sortie s_i. La définition des RdP structurés impose de **faire coïncider les blocs de liaisons avec les blocs de processus** afin d'éviter les blocages dus à une mauvaise localisation des liaisons /COR 81/.

Exemple :

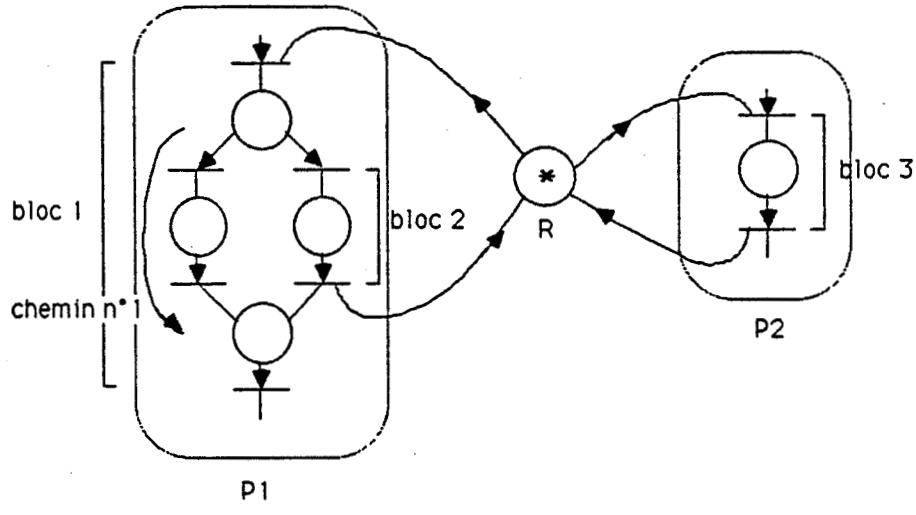


FIGURE 3

Si la ressouce est utilisée par la bloc alternative (bloc 1) et si le chemin parcouru est le chemin n° 1, la ressource R n'est pas restituée. Les processus P_1 et P_2 sont alors irrémédiablement bloqués.

Cependant, le respect de cette convention ne garantit pas l'absence de blocages (Fig. 4).

Exemple :

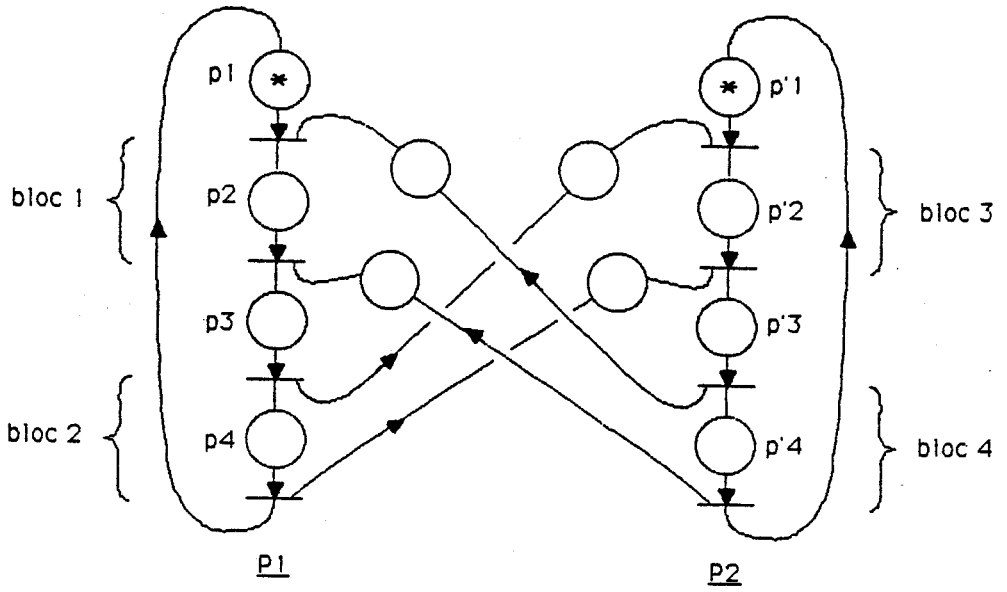


FIGURE 4

Les processus P_1 et P_2 sont initialement bloqués.

Par contre, elle interdit certaines configurations où les blocs de liaisons se chevauchent sans que cette situation ne génère de blocage (Fig. 5).

Exemple :

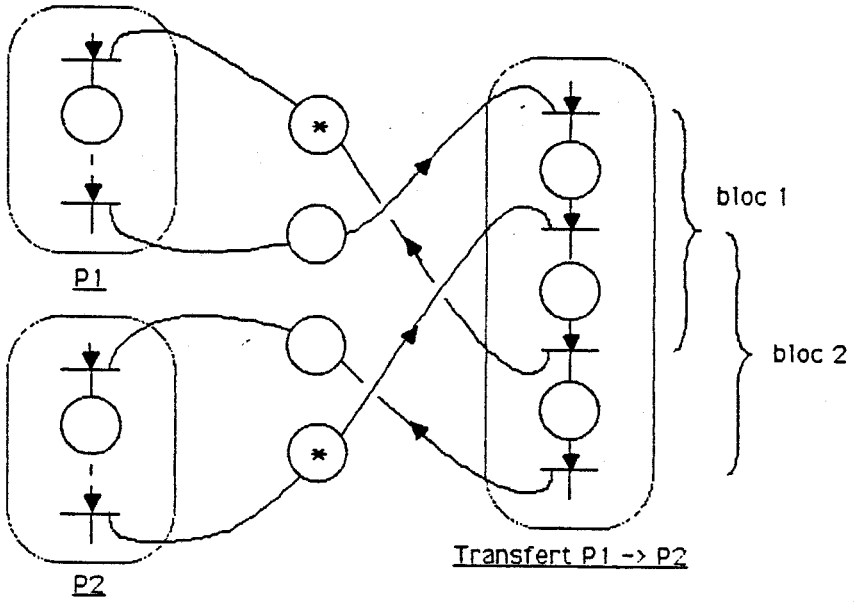


FIGURE 5

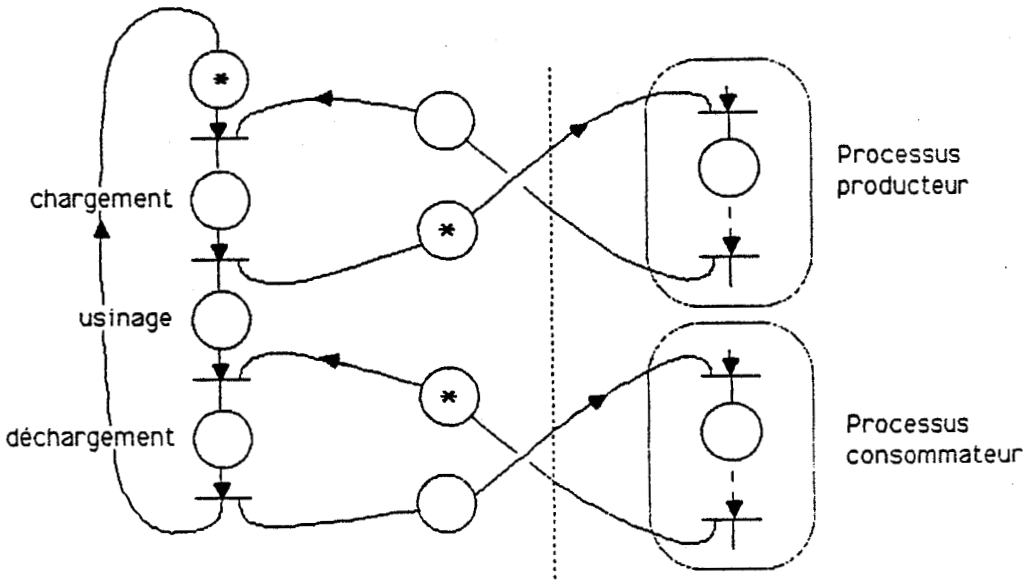
Les 3 exemples ci-dessus prouvent l'inadéquation entre le critère de localisation des blocs de liaison proposé dans /COR 81/ et la notion de blocage. En effet, le concept de blocage introduit ici de manière intuitive, est issu de la propriété de "vivacité" des RdP (ce point sera développé au Chapitre II, § I.1) dont l'étude dépend essentiellement du marquage initial. Or, le critère proposé repose uniquement sur la structure du graphe, indépendamment du marquage. Dans le cas de la Fig. 3, la mauvaise localisation de la structure d'exclusion mutuelle induit une configuration potentiellement bloquante quel que soit le marquage initial, ce qui revient implicitement à une condition sur le marquage. Dans le cas de la Fig. 4, le système initialement bloqué pour le marquage proposé devient tout à fait viable pour les marquages initiaux suivants : P_1' et P_3, P_1' et P_3', P_1 et P_3'.

Nous n'adopterons donc pas ce critère au niveau des modèles étudiés. La définition de toute nouvelle structure de liaison est sujette à caution. Notons que les blocages induits par ces liaisons, dont l'existence est simple à mettre en évidence si le blocage dépend d'un processus unique (Fig. 3) voire de 2 processus (Fig. 4), s'avère nettement plus problématique quand les liaisons s'échelonnent en cascade sur de multiples processus. Dans ce cas, aucune règle simple ne permet de détecter a priori le risque de blocage inhérent à l'ajout d'une nouvelle liaison.

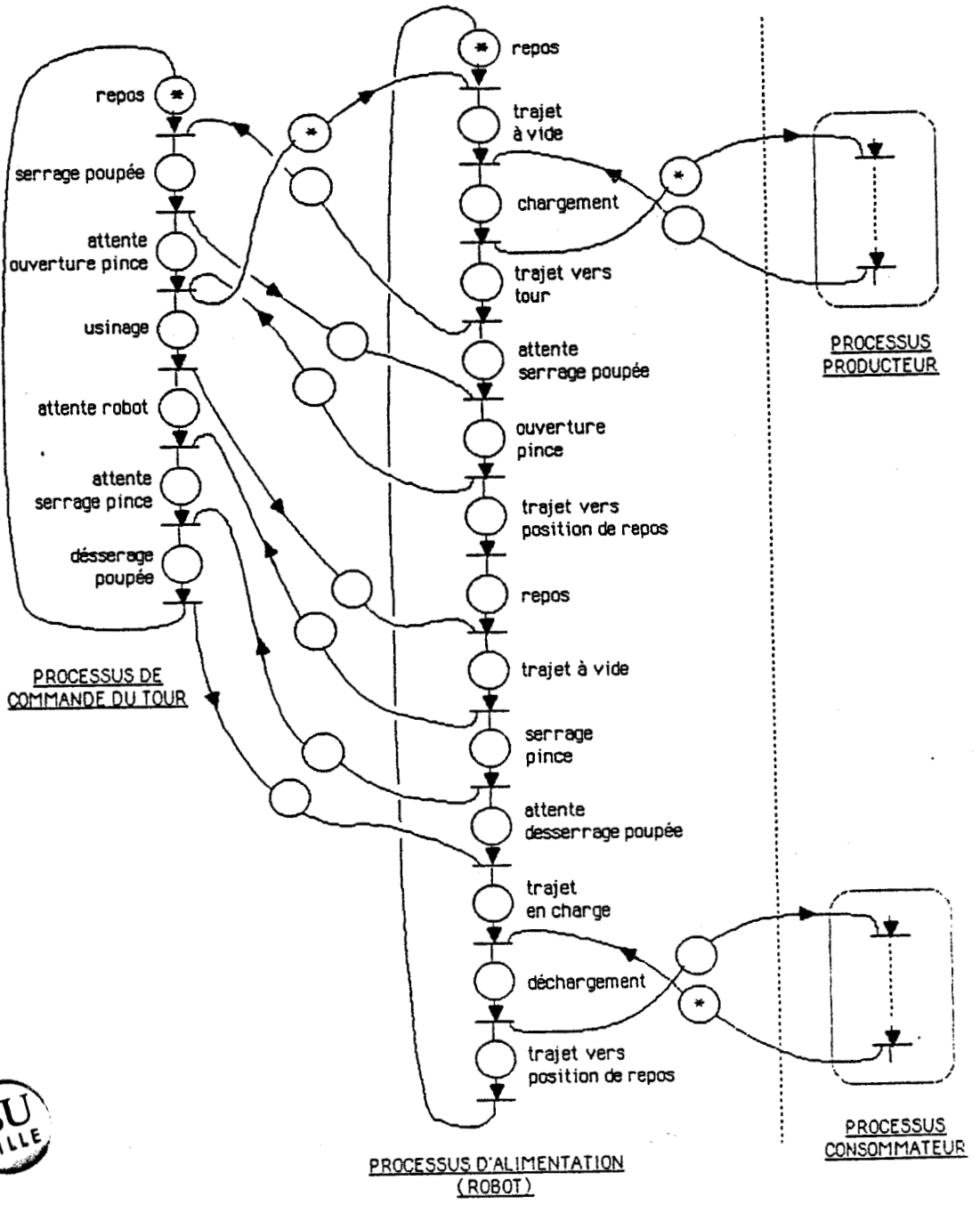
d) Un exemple de modélisation structurée :

Reprenons le sous-système présenté au § I.1 sur l'alimentation d'un tour à commande numérique par un robot manipulateur.

D'une première approche purement fonctionnelle, il apparaît qu'un processus unique permet de traduire correctement la commande de ce sous-système organisée séquentiellement en un cycle de 3 blocs : chargement, usinage et déchargement.



Par contre, la prise en compte des contraintes d'implantation répartie nous oblige à scinder ce sous-système en 2 processus : le processus d'usinage implanté sur la commande numérique du tour et le processus d'alimentation du tour implanté sur la commande numérique du robot. Le graphe résultant sera, par exemple, le suivant :



I.3.3 - Conclusion

Nous adopterons pour la modélisation et la simulation des systèmes flexibles, les hypothèses de la structuration afin de faciliter la conception des grands systèmes. La seule restriction envisagée concerne la localisation des structures de liaison.

I.4 - Les réseaux structurés adaptatifs

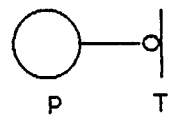
I.4.1 - Introduction

Les RdP sont non déterministes et non flexibles. Aussi, certaines opérations s'avèrent impossible à réaliser par RdP simples. C'est le cas de l'indéterminisme d'accès à une ressource critique partagée ou de l'interprétation des liaisons avec le procédé et le niveau hiérarchique de prise de décision, ou de la modélisation de la flexibilité relative aux divers modes de marche.

Afin d'augmenter le pouvoir de modélisation du modèle de base, diverses extensions ont été proposées.

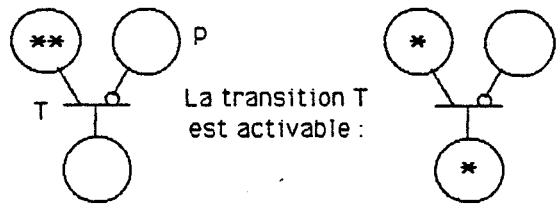
*** Les arcs inhibiteurs** /HAC 75/ :

Certains arcs amonts sont représentés graphiquement comme suit :

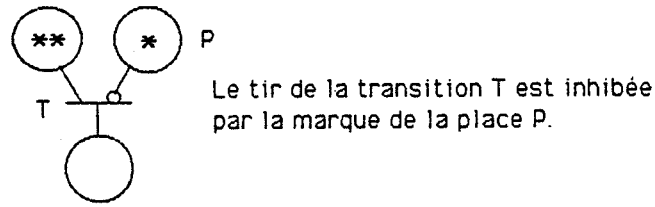


La condition d'activation de la transition T relativement à cet arc correspond à l'absence de marquage de la place P.

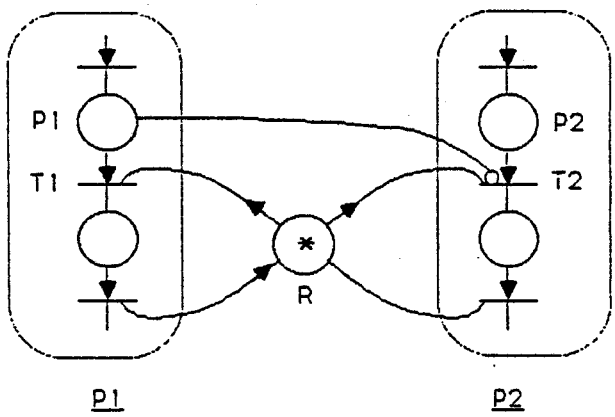
Exemple 1 :



Exemple 2 :



Une telle extension permet de résoudre par exemple l'indéterminisme d'accès à une ressource critique :



En cas de requête simultanée de la ressource R, le processus P_1 est prioritaire puisque la transition T_2 est inhibée par le marquage de la place P_1.

*** Les réseaux à priorités** /HAC 75/ :

On associe au RdP une relation d'ordre partiel strict entre les transitions potentiellement génératrices d'indéterminisme.

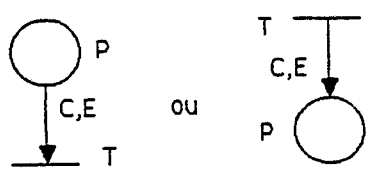
Ainsi, pour l'exemple ci-dessus, on définira la relation "$T_1 > T_2$" signifiant que la transition T_1 est plus prioritaire que la transition T_2 en cas d'indéterminisme résultant du marquage simultané des places P_1 et P_2.

*** Les réseaux automodifiants** /VAL 77/ :

Les arcs du RdP sont étiquetés par un doublet (C,E) où :

C est le poids de l'arc

E est le nom d'une place du réseau.



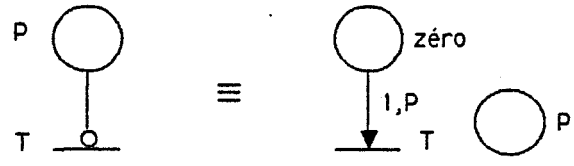
Les règles d'évolution du graphe sont identiques à celles définies pour les RdP simples, le poids de l'arc étant égal à C * M(E).

Cette définition entraîne les conséquences suivantes :

i) Le poids des arcs est fonction de l'état instantané du marquage du réseau.

ii) La définition de la place "UN" dont le marquage est constamment égal à 1 permet d'englober la définition classique des RdP. En effet, un réseau automodifiant dont tous les arcs sont étiquetés par la place UN est un RdP simple.

iii) La définition de la place "ZERO" dont le marquage est constamment nul permet d'englober la classe des RdP à arcs inhibiteurs.

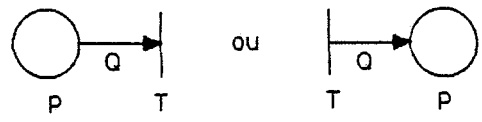


*** Les réseaux de Petri structurés adaptatifs** /COR 84/ :

Ils proposent une extension des RdP structurés dont la définition est proche des réseaux automodifiants de Valk.

I.4.2 - Les RdP structurés adaptatifs : Définition

Un RdP structuré adaptatif possède des arcs de la forme suivante, (où Q est le nom d'une place du graphe) :



Un arc adaptatif est donc un arc automodifiant dont le coefficient de pondération C est égal à 1.

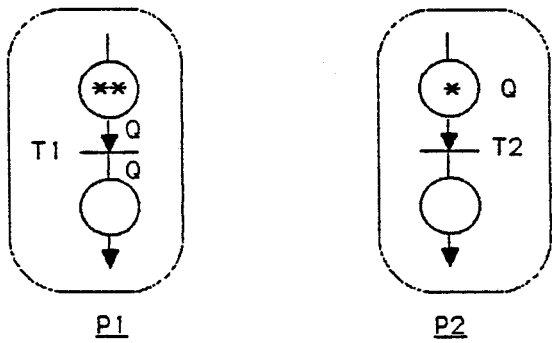
Pour un marquage M, si M(Q) ≠ 0 alors le poids de l'arc reliant P à T est égal à M(Q). Par contre, si M(Q) = 0 alors l'arc est supprimé.

Les RdP adaptatifs imposent une restriction sur l'activation des transitions par rapport au modèle original de Valk : une transition T possédant des arcs adaptatifs en amont, n'est activable que si le tir de la transition provoque une évolution du marquage amont. Ceci permet d'éviter la génération de marques par une transition source.

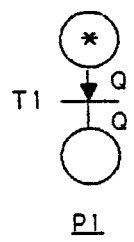
Deux conséquences résultent de cette définition.

i) Le tir d'une transition doit être considéré comme une opération indivisible puisque le poids des arcs adaptatifs peut évoluer entre le demi-déclenchement haut et le demi-déclenchement bas.

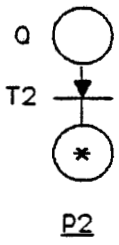
Exemple :



Le demi-déclenchement haut de la transition T_1 engendre la situation transitoire suivante :

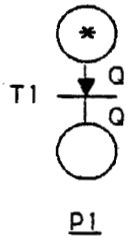


Si, avant le demi-déclenchement bas de la transition T_1, la marquage de la place Q a évolué (Fig. 6) alors, le demi-déclenchement bas de T_1 est décrit Fig. 7.



P2

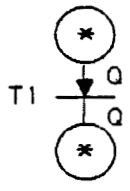
FIGURE 6



P1

FIGURE 7

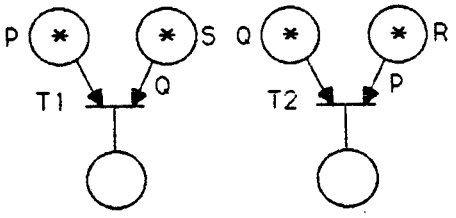
Afin d'éviter ce type d'évolution aléatoire, les demi-déclenchements haut et bas de la transition T_1 doivent être évalués en même temps :



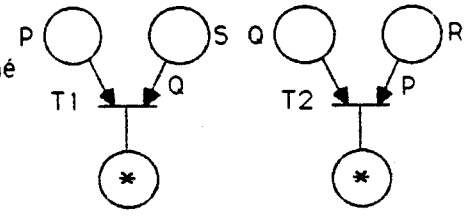
P1

ii) Le tir des transitions activables à un instant donné doit être simultané sinon l'état résultant du graphe peut dépendre de l'ordre de déclenchement.

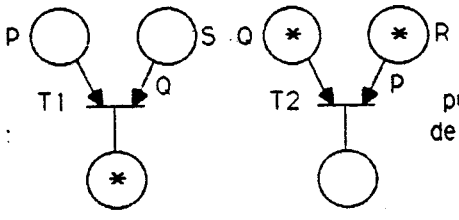
Exemple :



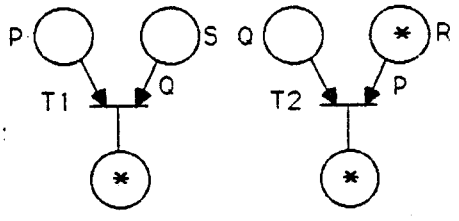
Déclenchement simultané de T1 et T2 :



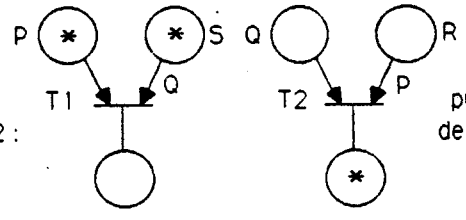
Tir de T1 :



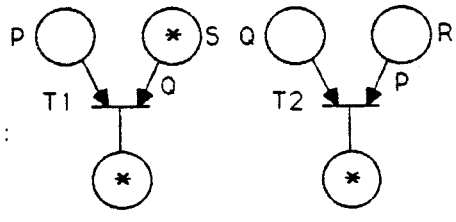
puis de T2 :



Tir de T2 :



puis de T1 :



Ces 2 remarques importantes introduisent les concepts d'indivisibilité et de cycle nécessairement respectés dans tout simulateur utilisant cette extension des RdP.

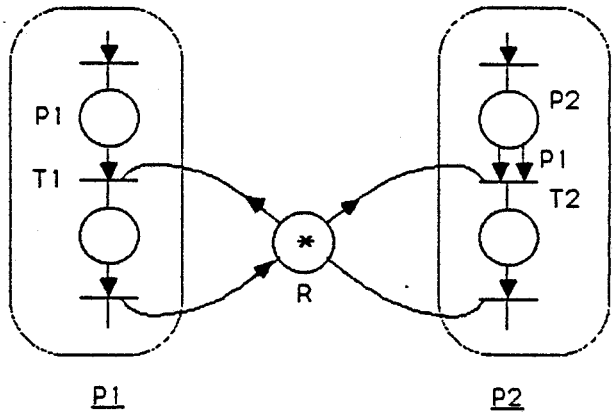
I.4.3 - Les extensions de la modélisation

L'utilisation des RdP structurés adaptatifs permet :

- de résoudre l'indéterminisme,
- de représenter différents modes de marche,
- d'intégrer les notions de "gel" de bloc ou de processus.

* **Résoudre l'indéterminisme** d'accès pour les ressources partagées ou les producteurs / consommateurs à accès multiples.

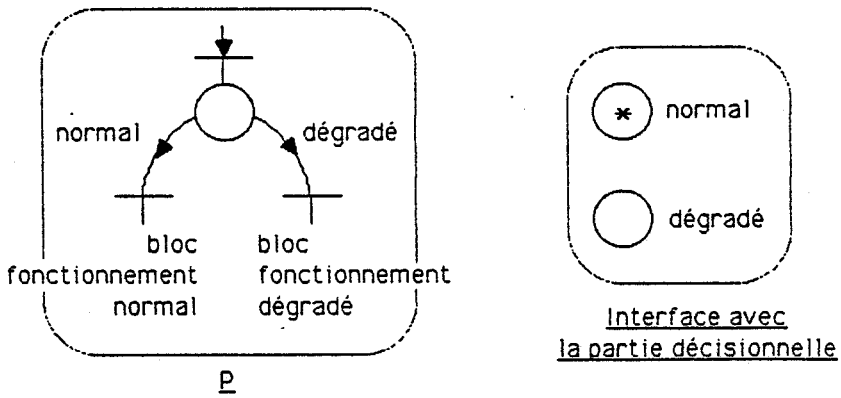
Exemple :



En cas de requête simultanée de la ressource R, le processus P_1 est prioritaire. Le "gel" de la marque présente dans la place P_2 est alors effectivement réalisé par l'adjonction de l'arc étiqueté par P_1 puisque, les processus P_1 et P_2 étant saufs, les places P_1 et P_2 ont leur marquage au plus égal à 1.

* **Représenter différents modes de marche** en permettant la connexion ou la déconnexion de sous-réseaux :

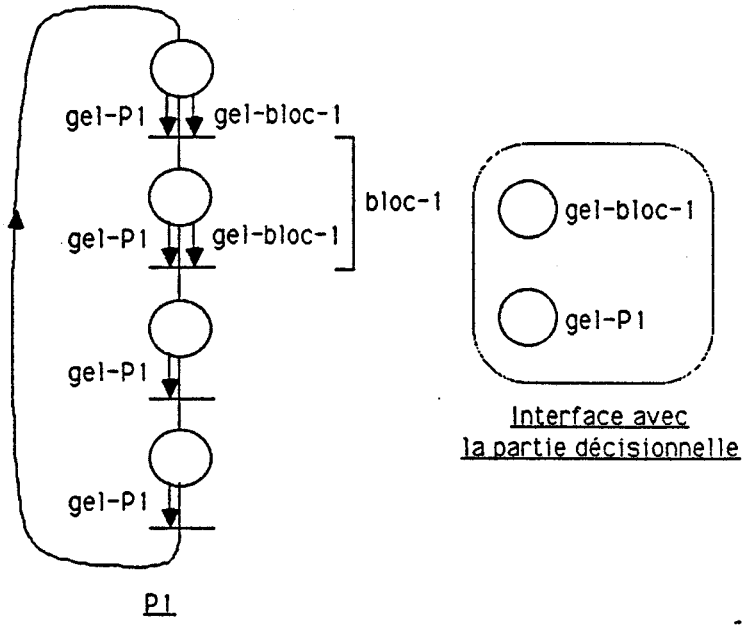
Exemple :



Le processus P est ici en mode de marche "normal". L'utilisation des places d'interface "normal" et "dégradé" et la gestion du changement de mode seront développés dans le paragraphe II.

* **Intégrer les notions de "gel" de bloc ou de processus,** mécanismes permettant le contrôle sélectif de l'évolution du graphe.

Exemple :



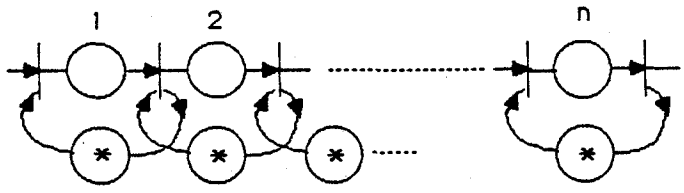
Le gel du processus P_1 permet d'interdire le tir de toute transition du processus P_1. Le gel du bloc-1 permet d'interdire toute évolution à l'intérieur du bloc-1.

I.5 - Les réseaux colorés

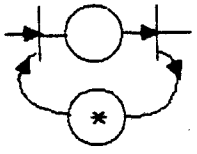
I.5.1 - Introduction à la coloration

Lors de la modélisation d'un système réel, la taille du réseau devient souvent très importante. Généralement, certaines structures identiques apparaissent plusieurs fois car elles sont utilisées dans des configurations différentes. Sans chercher à augmenter le pouvoir de modélisation du modèle, il peut être intéressant de chercher à "replier" le modèle. L'abréviation du réseau est donc issue d'une volonté de faciliter l'écriture et de permettre une plus grande concision du graphe. La notion de marque doit alors être enrichie afin de tenir compte de cette abréviation du graphe. La notion de coloration des marques a été introduite dans cet esprit /GEN 79/ /JEN 81/ /PET 80/.

La modélisation d'un mécanisme de transfert en FIFO par exemple, peut se représenter comme suit :



On conçoit aisément qu'un tel système peut se révéler prohibitif si la FIFO (First In First Out) représentée possède 40 places (n=40) sans que la représentation exhaustive des 80 places du sous-graphe concerné soit très enrichissante. On préfèrera agréger ce sous-système constitué de la répétition en 40 exemplaires de la même structure :



La couleur d'une marque du modèle replié traduira alors le rang de cette marque dans la file.

Dans cette optique, l'abréviation d'un graphe structuré présentant des redondances de processus introduira des structures ayant la morphologie des processus mais possédant plus d'une marque (pseudo-processus non saufs).

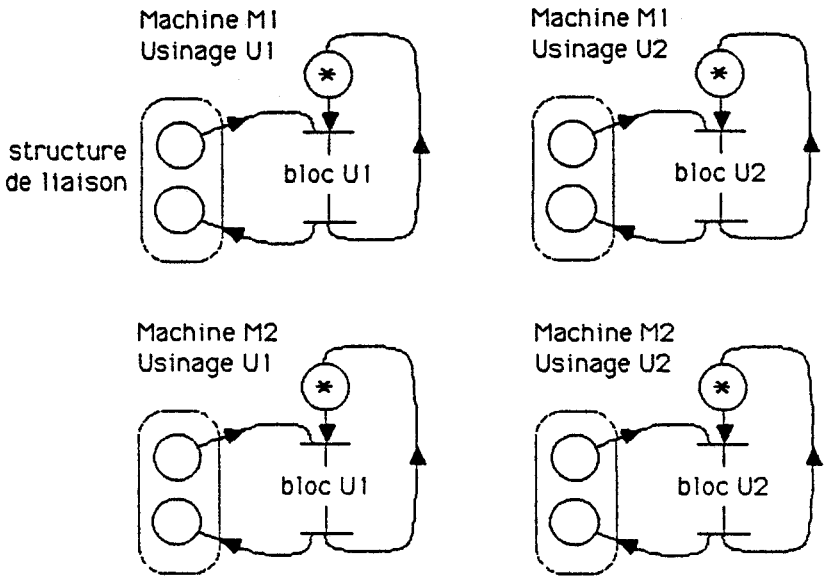
Si l'approche de la coloration que nous avons adoptée rejoint cette approche au niveau de la définition formelle, elle est tout autre au niveau des objectifs de modélisation.

I.5.2 - La coloration en productique

Les systèmes de production flexibles sont caractérisés par la diversité des pièces conditionnées par le système. En fonction de leur type, les pièces circulant dans l'atelier subissent des traitements spécifiques différents ou non.

Le critère de coloration que nous avons adopté résulte de la typologie des pièces et consiste en un regroupement **a priori** des **fonctions** du système. Au contraire, dans une autre approche, utilisée par exemple pour la modélisation des systèmes de communication, la coloration intervient **a posteriori** à partir d'un critère d'analogie **structurelle** des processus.

Prenons l'exemple de 2 machines identiques M1 et M2 opérant chacune selon 2 types d'usinages différents U1 et U2. Les 4 processus modélisant ces fonctionnements sont les suivants :



Un regroupement structurel différenciera les types d'usinages (Fig. 8).

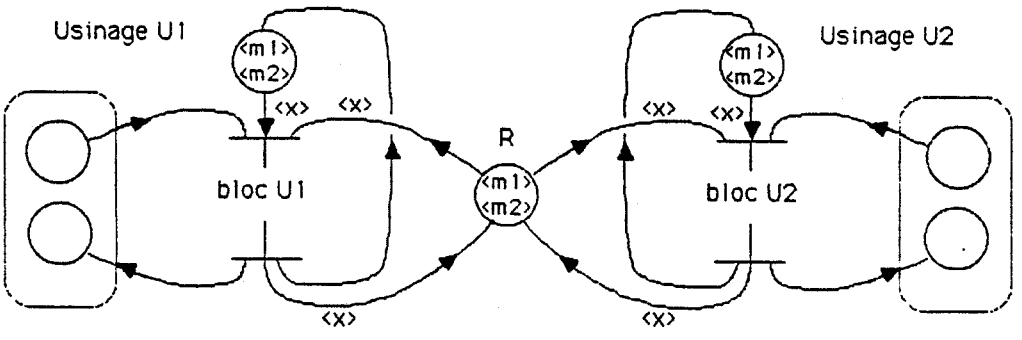


FIGURE 8

Les couleurs "m1" et "m2" identifient les machines. La ressource R assure l'unicité d'utilisation de la machine "M1" (respectivement "M2").

Le regroupement fonctionnel différencie les 2 machines en regroupant les usinages (Fig. 9).

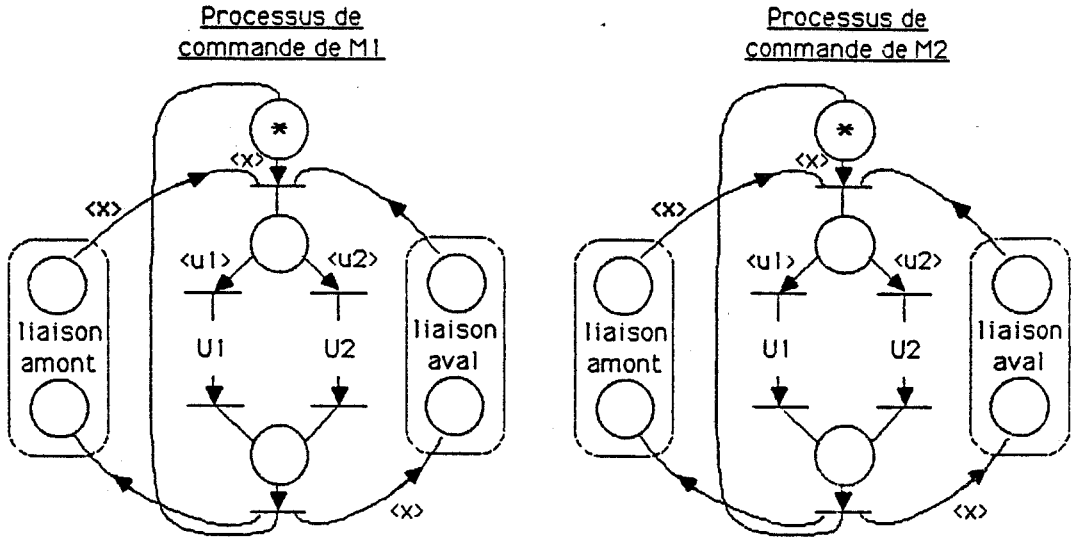


FIGURE 9

Les couleurs correspondent ici au type des pièces.

Notons que les processus agrégés résultant de la "coloration fonctionnelle" sont **saufs.**

I.5.3 - Définition formelle de la coloration

La définition initiale de la coloration adoptée, s'apparente à celle de Jensen. La seule restriction par rapport à ce modèle concerne l'étiquetage des arcs dans le cas des couleurs déterminées. Le modèle de Jensen autorise l'étiquetage d'un arc par un ensemble de couleurs déterminées alors que nous le limiterons à une seule couleur. Cette restriction s'inscrit naturellement dans le cadre des RdP structurés où les processus sont saufs par construction.

Soit $C = \{C_1, C_2, \ldots, C_n\}$ un ensemble fini de couleurs (chaque couleur correspond dans notre cas à un type de pièce circulant dans le système).

T désigne l'ensemble des transitions du réseau,
P désigne l'ensemble des places.

Le marquage M se définit comme une application de l'ensemble P dans l'ensemble des fonctions de C dans les entiers :

$$M : P \longrightarrow [\, C \longrightarrow \mathbb{N} \,]_f .$$

Ainsi, $M(P)(C_i)$ désigne le nombre de marques de couleur C_i au sein de la place P.

A chaque transition t de T, on associe :

(i) Un ensemble K de couleurs déterminées intervenant lors du tir de t (dans la suite de cet exposé, ces couleurs seront notées en majuscule).
(ii) Un ensemble X de couleurs variables (ces variables seront notées en minuscule). A chaque couleur x de X est associé un domaine dom(x) inclus dans C.
(iii) L'ensemble A-pré des arcs en amont de t.
(iv) L'ensemble A-post des arcs en aval de t.

Chaque arc est défini par la donnée d'un triplet (n,c,p) où :
n est le poids de l'arc ($n \in \mathbb{N}$),
c est la couleur transitant par cet arc ($c \in K$ ou $c \in X$),
p est la place reliée à t par l'arc considéré ($p \in P$).

* **La transition t est activable** depuis le marquage M ($M(t>.)$ ssi :

(i) $\forall\, x \in X, \exists\, S_x \in dom(x), \forall\, (n_i,c,p) \in$ A-pré

$$M(p)(c) \geq \sum_{(n_i,c,p) \in \text{ A-pré}} n_i.c$$

S_x est appelée couleur effective de x au marquage M pour la transition t.

(ii) $\sum_{(n,c,p) \in A\text{-pré}} n.c \geq 1$

Cette dernière condition impose une évolution du marquage amont lors du tir de t.

* **Le marquage M' résultant du tir de t** (M(t>M') est définit comme suit :

(i) Pour toute place P non connectée à t, M'(P) = M(P)

$\forall x \in X, \exists S_x \in dom(x)$ tq

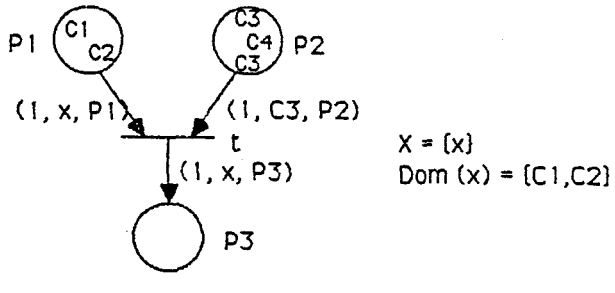
(ii) pour toute place P liée à t par des arcs amonts (n_i,c,p)

$$M'(P)(c) = M(P)(c) - \sum_{(n_i,c,p) \in A\text{-pré}} n_i.c$$

(iii) pour toute place P liée à t par des arcs avals (n_i,c,p)

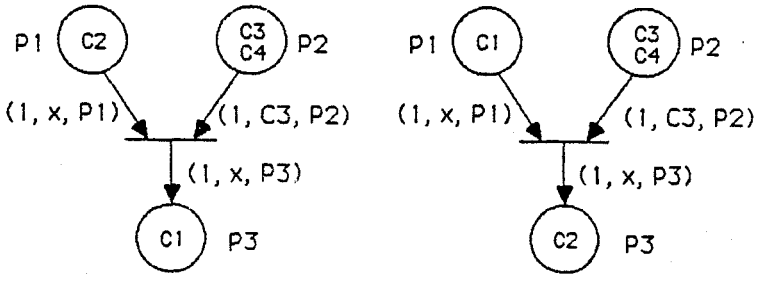
$$M'(P)(c) = M(P)(c) + \sum_{(n_i,c,p) \in A\text{-post}} n_i.c$$

Exemple :



La transition t est tirable pour

x = C1 ou x = C2



*** Remarque sur les variables libres :**

La définition d'une variable libre et de son domaine de variation est initialement locale à la transition considérée. Cependant, l'expérience montre qu'une séquence de transitions adjacentes posséde en général des variables libres analogues (c'est-à-dire dont les domaines sont identiques). Dans ce cas, nous conseillons d'associer à ces variables le même identificateur afin de faciliter l'écriture et la lisibilité du graphe. Nous définissons donc des variables libres **globales** et leur domaine de variation, avec la convention suivante :

> Si le domaine de variation d'une variable libre x, associée à la transition t est précisé au niveau de la définition de t, la variable x sera locale et globale sinon.

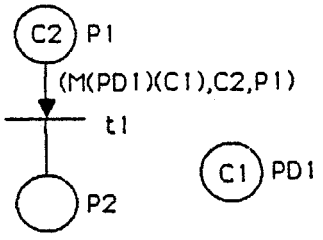
I.5.4 - Extension du modèle : les RdP structurés adaptatifs et colorés (RdP SAC)

La restriction due à la structuration et l'adjonction de l'aspect adaptatif définit notre modèle de base (les RdP SAC) que nous utiliserons pour la conception et la simulation de la partie commande des systèmes de production flexibles.

L'intégration du concept de structuration ne pose aucun problème supplémentaire du fait de la coloration.

La prise en compte de l'aspect adaptatif est rendu possible par l'extension de la définition du poids "n" des arcs (n,c,p) relatifs à une transition. Ce poids "n" doit pouvoir être conditionné par le marquage d'une place quelconque du réseau Q, relativement à une couleur C_Q. Le poids est défini par la donnée de $M(Q)(C_Q)$ où C_Q appartient à K (couleur fixée) où à X (couleur variable).

Exemple :



Le poids de l'arc amont est M(PD1)(C1) nombre de marques de couleur C1 dans la place PD1.

Ce modèle s'éloigne du modèle original proposé dans /COR 85/. les modifications apportées sont les suivantes :

(i) Le modèle original comme celui de Jensen, permet l'étiquetage d'un arc par un **ensemble** de couleurs constantes alors que nous le limitons à une seule.

(ii) Le modèle original impose l'identité de la couleur de test et de la couleur transitée par l'arc, c'est-à-dire que tous les arcs sont de la forme suivante :

(M(Q)(C),C,P)
└─┴─→ même couleur

I.6 - Conclusion sur la modélisation de la partie commande

Les trois modèles de base (RdP structurés, RdP adaptatifs et RdP colorés) proposent une gamme de modèles compatibles entre eux dont le pouvoir de modélisation est adaptable à la complexité du système étudié. Le modèle RdP SAC est suffisant pour représenter correctement un système flexible sans faire appel à des extensions plus puissantes telles les RdP prédicats / transition résultant de l'agrégation des RdP colorés. C'est ce modèle des RdP SAC qui a servi de base à l'élaboration de notre simulateur.

II - MODELISATION DE LA PARTIE DECISIONNELLE

II.1 - Introduction

Afin d'assurer une meilleure productivité, les systèmes de production flexibles intègrent généralement des notions de gestion de production et de fonctionnements dégradés. Définir une stratégie de production consiste en particulier à gérer les priorités dynamiques entre tâches concurrentes. On privilégiera par exemple la tâche la plus courte afin de réduire le temps moyen d'attente.

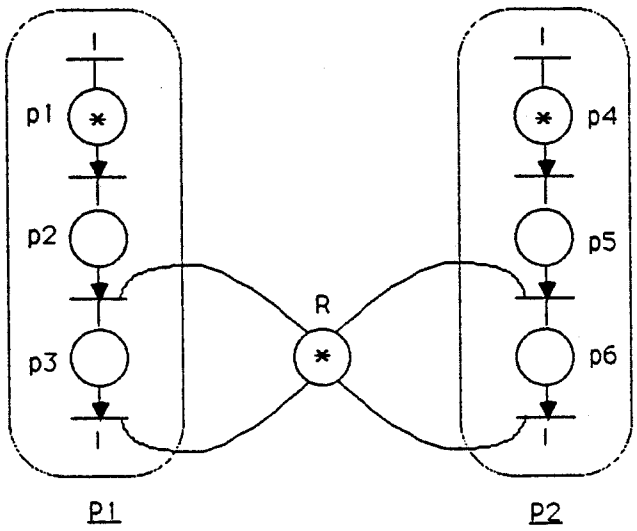
La prise en compte des défauts consiste à réorganiser la commande afin d'assurer un cadre de production réduit.

Il s'agit dans les 2 cas d'une prise de décision dont le résultat est de paramétrer le graphe de commande (cf § I.4.3). Ces mécanismes de décision sont généralement issus de critères divers et complexes et nous avons choisi, pour plusieurs raisons, de les regrouper dans ce que nous appellerons la partie décisionnelle.

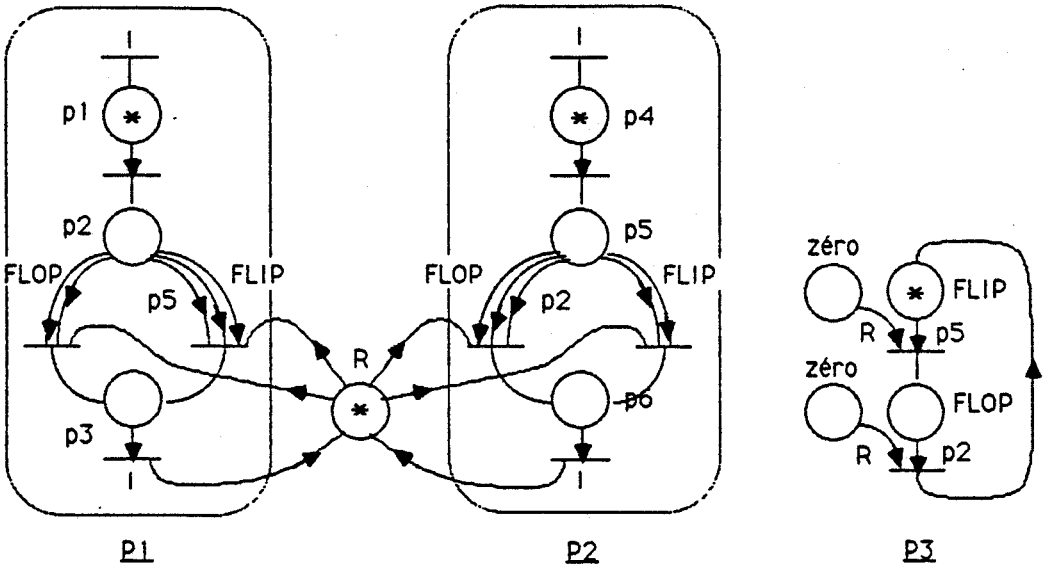
i) L'intégration des structures de contrôle au sein du graphe de commande se fait au détriment de la simplicité du réseau.

Exemple :

Paramétrage d'une priorité



Supposons que l'on veuille modéliser, par RdP adaptatifs, un mécanisme de priorité tournante en cas de requête simultanée des processus P_1 et P_2. Le graphe résultant est le suivant :



Le processus P_3 modélise la gestion de la priorité tournante. Si la place "FLIP" est marquée, priorité est donnée à P_1. Si la place "FLOP" est marquée, priorité est donnée à P_2. Ce mécanisme, s'il réalise bien la priorité tournante, complique considérablement la structure du graphe initial.

ii) Il se peut que le concepteur n'ait pas, a priori, la connaissance des contrôles à définir ou qu'il hésite entre diverses stratégies de contrôle. Grâce à la partie décisionnelle, l'ajout ou la modification peut être réalisée a posteriori sans devoir reconfigurer le graphe de commande.

iii) Les structures de contrôle nécessitent parfois des informations issues d'une vision globale du système et doivent donc être intégrées à un niveau suffisant de la hiérarchie du système.

Le passage d'une machine en mode dégradé par exemple, peut être décidé par un opérateur ou être déclenché par la détection de pièces anormalement usinées lors d'un contrôle de qualité ou par un organe décentralisé de surveillance (synoptique), et sa réalisation peut modifier le transit des pièces circulant dans tout l'atelier. En ce cas, le contrôle du mode dégradé ne peut pas être défini localement au niveau de l'organe de commande de la machine fautive.

II.2 - Description du modèle de la partie décisionnelle

Plutôt qu'utiliser un langage impératif, il s'avère intéressant d'utiliser un langage déclaratif comme outil de description de la partie décisionnelle. A une structure de contrôle sera associée une ou plusieurs règles de production. Nous avons donc défini, au niveau du simulateur, un moteur d'inférence spécifique apte à gérer un ensemble de règles de production dont le rôle est de piloter le graphe de commande.

L'intérêt d'un tel modèle est de donner à l'utilisateur la maîtrise de conception de sa stratégie car la modularité induite par cette démarche permet aisément d'enrichir ou de modifier les structures de contrôle.

La structure d'une règle de production est la suivante :

(i) un identificateur de règle,
(ii) un ensemble de variables libres et la donnée de leur domaine de variation ; ces variables libres permettent d'assurer le lien avec les couleurs du graphe de commande,
(iii) un énoncé du type : SI <condition> ALORS <action>.

La partie <condition> est constituée par une combinaison logique de prédicats testant l'état du graphe, et la partie <action> par une séquence des opérateurs "ajouter" et "enlever", définis comme suit :

Ajouter (n,c,p) : ajoute n marques de couleur c dans la place p,
Enlever (n,c,p) : retranche n marques de couleur c de la place p.

Note : Les prédicats et les opérateurs de base définis ici sont en nombre restreint. Cependant, les structures obtenues par leur composition permettent un paramétrage élaboré du graphe de commande. Certaines de ces structures, dont l'utilisation revient fréquemment, sont regroupées en primitives d'interface, point que nous développerons ultérieurement (§ II.6).

Les déclencheurs de règle sont de 3 types :

- la règle étudiée est déclarée activable,
- la règle possède des variables libres et, pour chaque variable, une unification est possible,
- la partie <condition> de l'énoncé est vraie.

Notons l'équivalence entre l'énoncé d'une règle de production et la notion de transition pour un formalisme de type RdP.

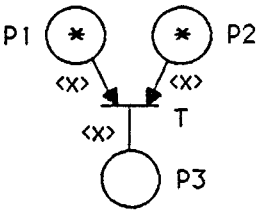
Exemple :

Règle R1

variable-libre = {x}
dom(x) précisé

énoncé :
SI (M(P1)(x) ≥ 1 ET M(P2)(x) ≥ 1)
ALORS (enlever (1,x,P1) ET
enlever (1,x,P2) ET
ajouter (1,x,P3))

<=>



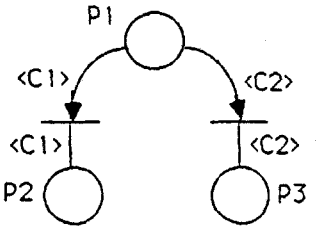
Règle R2

énoncé :
SI (M(P1)(C1) > 0)
ALORS (enlever (1,C1,P1) ET ajouter (1,C1,P2))

Règle R'2

énoncé :
SI (M(P1)(C2) > 0)
ALORS (enlever (1,C2,P1) ET ajouter (1,C2,P3))

<=>



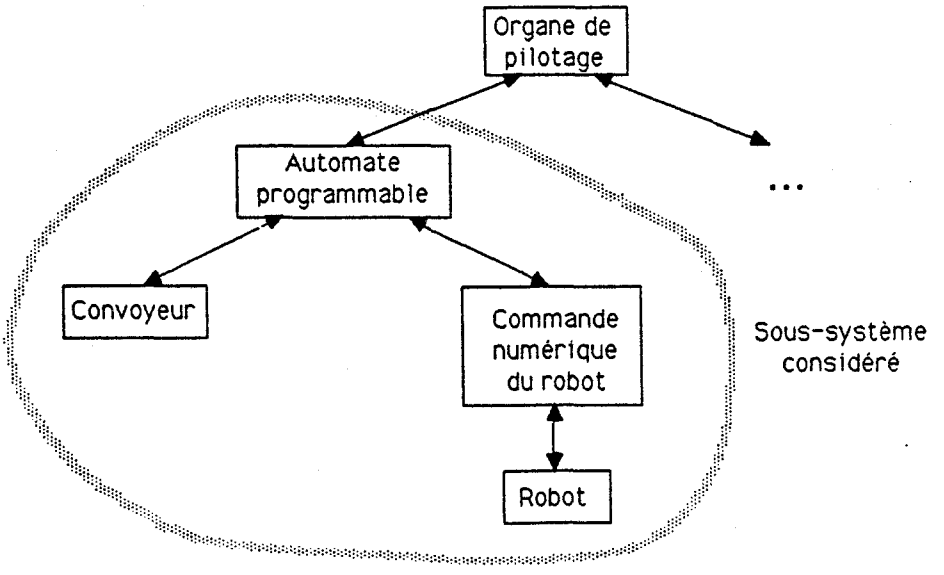
L'utilisation systématique des règles de production est réalisée dans le logiciel SYSTEL /BON 86/. On peut reprocher à une telle démarche de sacrifier l'aspect graphique du modèle RdP au profit de l'aspect déclaratif du modèle choisi. La démarche que nous adoptons est plus nuancée : seule la partie décisionnelle est représentée par un ensemble de règles de production.

II.3 - Critère de décision dans une architecture répartie

La partie décisionnelle est tributaire autant que le graphe de commande, de la structure matérielle implémentant le système de pilotage des diverses "machines" utilisées. Elle doit, elle aussi, satisfaire aux exigences de l'implantation répartie.

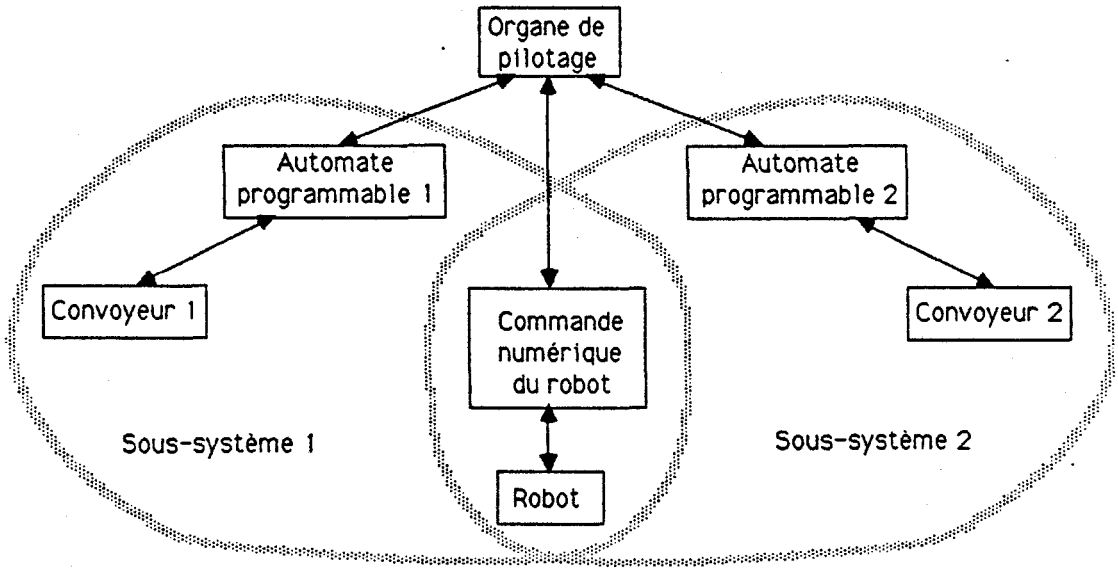
Du fait de la multiplicité des organes de commande, le système conçu est généralement organisé en une hiérarchie arborescente.

Considérons par exemple, le sous-système constitué d'un convoyeur et d'un robot d'alimentation. Le robot est employé à 2 tâches distinctes : le chargement des pièces brutes sur le convoyeur et le déchargement des pièces usinées. Ces 2 tâches induisent des commandes à la fois sur le robot (déplacement du bras, ouverture et fermeture de la pince, ...) et sur le convoyeur (gestion des butées de débrayage et éventuellement des embranchements, ...). Ce sous-système peut être implémenté comme suit :



La génération des commandes du robot se fait au niveau de l'automate programmable, outil de synchronisation entre le convoyeur et la commande numérique du robot. Si le robot est requis simultanément pour une tâche de chargement et pour une tâche de déchargement, la décision d'attribution peut être réalisée au niveau de l'automate programmable. Il s'agit d'une **décision locale** puisque prise au même niveau que la commande.

Par contre, si pour des raisons de rentabilité, ce même robot accomplit conjointement l'alimentation de deux convoyeurs, et gère les échanges entre 2 sous-systèmes disjoints, la structure implémentée pourra être la suivante :



Dans ce cas, l'activation des commandes du robot peut être requise simultanément par les 2 automates. La décision d'attribution du robot considéré comme ressource critique partagée par les 2 sous-systèmes, ne peut se faire qu'au niveau de l'organe de pilotage, ce qui nécessite un dialogue entre celui-ci et les automates. Ce mécanisme de requête / réponse nécessite l'utilisation du protocole de communication entre les divers organes de commande et induit donc une notion de temps de réaction qui n'existait pas dans le cas des décisions locales.

Pour traduire cette notion de décision locale ou répartie, nous avons scindé les règles de production du niveau décisionnel, en 2 classes : les règles de surveillance et les règles de requête.

II.4 - Les règles de surveillance

Elles permettent d'assurer le paramétrage du graphe issu d'une prise de décision locale et sont donc synchrones du cycle de base du mécanisme d'évolution du graphe de Petri. Les règles de surveillance observent l'état du graphe de commande au début de chaque cycle et génèrent sa paramétrisation grâce aux places d'interface introduites au § I.4.3.

Les règles de surveillance sont scindées en 2 classes : les règles d'initialisation et les règles de surveillance proprement dites (ou règles de paramétrisation).

*** Les règles d'initialisation :**

Elles réactualisent les places d'interface dont le marquage résulte de la paramétrisation due au cycle antérieur de l'évolution du graphe. Elles doivent donc être étudiées avant les règles de paramétrisation.

*** Les règles de paramétrisation :**

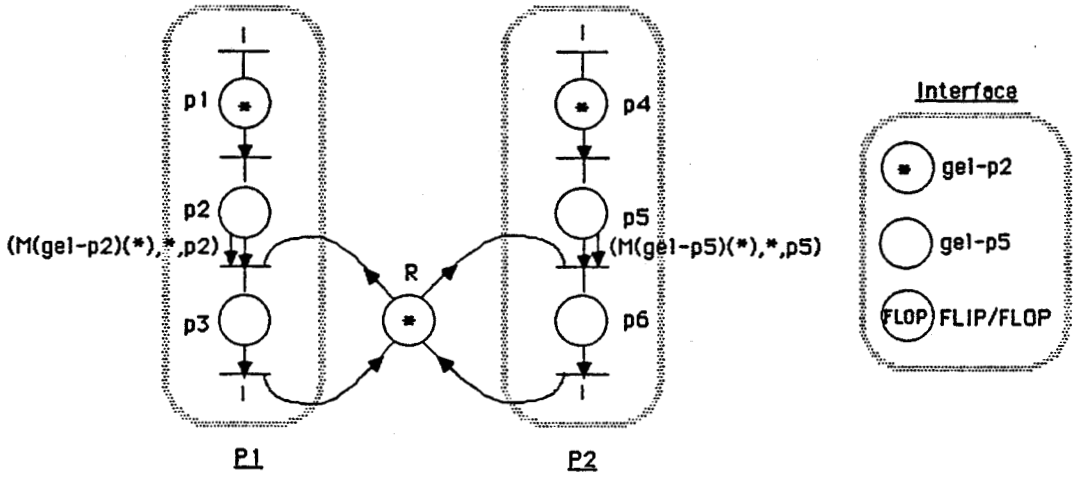
Elles assurent le contrôle du graphe (forçage des places d'interface) pour le cycle d'évolution envisagé.

Les règles de surveillance sont étudiées en séquence : d'abord les règles d'initialisation puis les règles de paramétrisation. Si le système de règles définies correspond à un contrôle cohérent du graphe, l'ordre des règles dans chacun de ces 2 groupes est indifférent. Pour des règles ayant des effets contradictoires, l'ordre d'étude de ces règles sera, bien sûr, déterminant.

Les places d'interface entre ces règles et le graphe de commande sont des places qui ne sont pas connectées au graphe, et dont le marquage est "forcé" uniquement par la partie <action> des règles de surveillance.

Exemple :

Paramétrage d'une priorité tournante (cf § II.1)



La place "FLIP/FLOP" joue le rôle de variable mémorisant le processus qui fut prioritaire lors de la requête simultanée précédente. Deux règles d'initialisation et 2 règles de paramétrisation peuvent représenter la décision en cas de requête simultanée.

Initialisation

1. Règle RAZ/gel-P2

énoncé : **SI** (M(gel-P2)(*) > 0) **ALORS** enlever (1,*,gel-P2)

2. Règle RAZ/gel-P5

énoncé : **SI** M(gel-P5)(*) > 0 **ALORS** enlever (1,*,gel-P5)

Paramétrage

3. Règle PRIO/P1

énoncé : **SI** (M(P2)(*) > 0 **et** M(P5)(*) > 0 **et**
M(FLIP/FLOP)(FLIP) > 0)
ALORS (ajouter (1,*,gel-P5) **et** enlever (1,FLIP,FLIP/FLOP)
et ajouter (1,FLOP,FLIP/FLOP))

4. Règle PRIO/P2

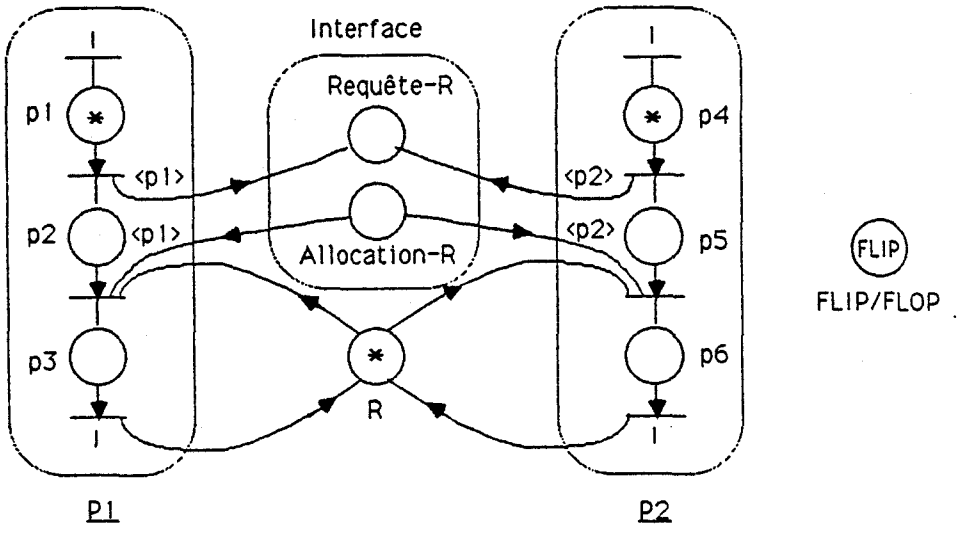
énoncé : **SI** (M(P2)(*) > 0 **et** M(P5)(*) > 0 **et**
M(FLIP/FLOP)(FLOP) > 0)
ALORS (ajouter (1,*,gel-P2) **et** enlever (1,FLOP,FLIP/FLOP)
et ajouter (1,FLIP,FLIP/FLOP))

II.5 - Les règles de requête

Elles sont principalement activées par la présence d'une marque au sein d'une place d'interface. Elles permettent de modéliser le dialogue nécessaire à la paramétrisation de la partie commande quand la décision est élaborée à un niveau hiérarchique supérieur dans l'architecture du système. Ce dialogue s'opère via des places d'interface permettant la liaison entre le modèle déclaratif des règles de requête et le modèle RdP de la partie commande.

Si l'organe élaborant la commande est différent de celui où est élaboré la décision, leurs évolutions sont nécessairement asynchrones. Le processus de commande ayant émis la requête doit alors se mettre en attente jusqu'à réception de la décision correspondante. Le mécanisme d'interfaçage est alors similaire à une structure de liaison de type synchronisation avec accusé de réception (cf § I.3.2.b).

Reprenons l'exemple du paramétrage d'une priorité tournante lors de l'accès à une section critique quand les 2 processus accédant à la ressource sont implantés sur des organes de commande différents.



A la place "requête-R" sont associées les règles de requête suivantes :

R1 : **SI** (M(requête-R)(P1) > 0 **et** M(requête-R)(P2) > 0 **et**
M(FLIP/FLOP)(FLIP) > 0)
ALORS (enlever (1,P1,requête-R) **et** ajouter (1,P1,allocation-R)
et enlever (1,FLIP,FLIP/FLOP) **et** ajouter (1,FLOP,FLIP/FLOP))

R2 : **SI** (M(requête-R)(P1) > 0 **et** M(requête-R)(P2) > 0 **et**
M(FLIP/FLOP)(FLOP) > 0)
ALORS (enlever (1,P2,requête-R) **et** ajouter (1,P2,allocation-R)
et enlever (1,FLOP,FLIP/FLOP) **et** ajouter (1,FLIP,FLIP/FLOP))

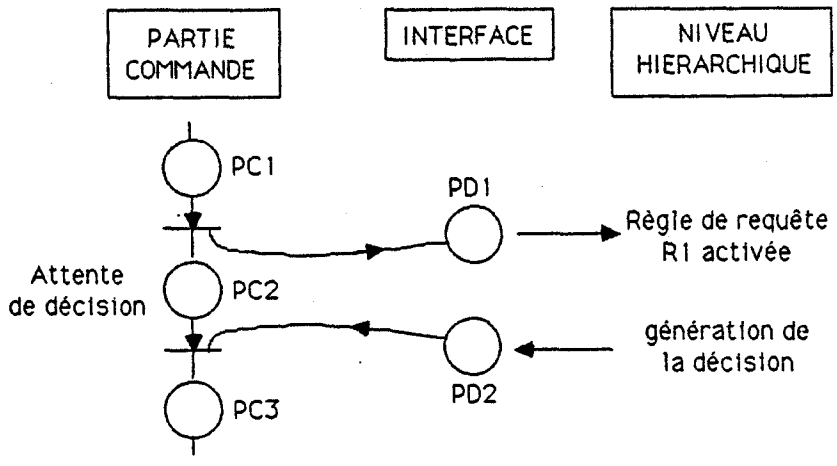
R3 : **SI** (M(requête-R)(P1) > 0 **et** M(requête-R)(P2) = 0)
ALORS (enlever (1,P1,requête-R) **et** ajouter (1,P1,allocation-R))

R4 : **SI** (M(requête-R)(P2) > 0 **et** M(requête-R)(P1) = 0)
ALORS (enlever (1,P2,requête-R) **et** ajouter (1,P2,allocation-R))

Ce mécanisme de requête / réponse est modélisé selon un formalisme dépendant de la localisation du processus opérant la requête par rapport au processus paramétré par la décision correspondante.

1er Cas : Requête et réponse interviennent au niveau du même processus, mais la décision est prise à un niveau supérieur dans la hiérarchie du système.

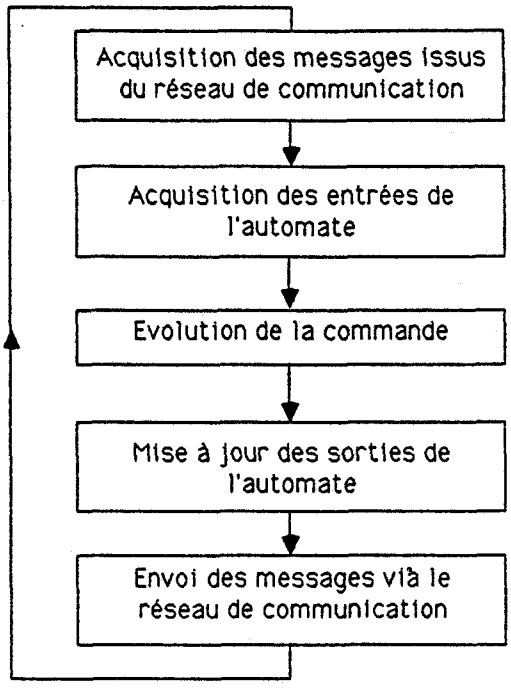
Le mécanisme de requête / réponse est typiquement celui de l'exemple précédent.



2ème Cas : La requête, la réponse, et éventuellement la décision, sont situées à des niveaux différents de la structure du système.

C'est typiquement le cas du paramétrage d'un fonctionnement dégradé après détection d'une défaillance d'une machine. La détection peut résulter de multiples conditions : intervention d'un opérateur détectant le dysfonctionnement ou d'un système décentralisé de surveillance (synoptique), autodiagnostic de la machine fautive (un capteur de contrainte, par exemple, peut détecter la casse de l'outil d'un tour), time-out associé à l'attente d'un accusé de réception après génération d'une commande.

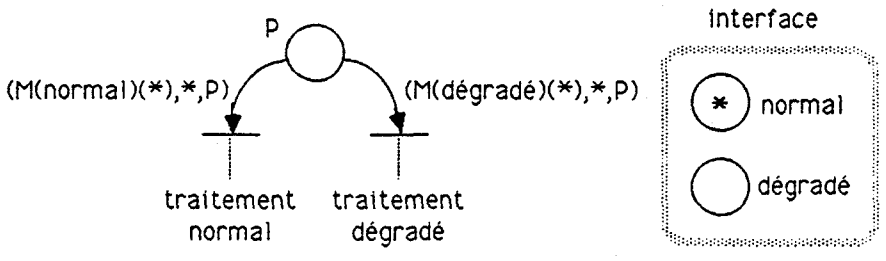
La caractéristique essentielle de ces mécanismes est l'asynchronisme des structures assurant la requête, la décision et la réponse, du fait de leur implantation sur des organes différents. Cependant, si ces structures sont fondamentalement asynchrones, leurs intéractions s'opèrent à des instants privilégiés de leur cycle respectif d'évolution puisqu'elles obéissent aux protocoles de communication sous-jacent dans le cadre d'une architecture répartie. Le cycle de base d'un automate programmable, par exemple, pourra être le suivant :



Dans ces conditions, le mécanisme de réalisation d'une décision peut être modélisé selon un formalisme similaire à celui des règles de surveillance.

Exemple :

Paramétrisation d'un passage en mode dégragé



Règle de requête : Passage en fonctionnement dégradé

SI <condition>

ALORS (enlever (1,*,normal)

et ajouter (1,*,dégradé))

II.6 - Les primitives de l'interface entre la partie commande et la partie décisionnelle

II.6.1 - Introduction

L'interface entre les parties commande et décisionnelle utilise des structures dont l'utilisation revient fréquemment lors de la modélisation d'un système de production flexible. Afin de faciliter la conception du système, nous avons défini des primitives banalisant les structures les plus courantes. Nous distinguons 2 types de primitives : les primitives de lecture de l'état du graphe de commande (dont la combinaison fournit une partie des prédicats utilisés dans la partie <condition> des règles de production), et les primitives d'action (correspondant à la réalisation de la décision élaborée dans la partie <action> des règles) /BOU 86/.

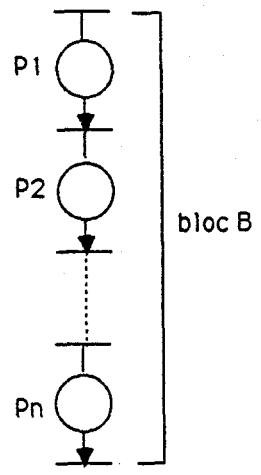
II.6.2 - Les primitives de lecture

Elles sont élaborées à partir d'une combinaison logique de tests sur l'état du graphe et retournent donc un résultat booléen. La fonction de marquage M est l'élément fondamental de la définition de ces primitives.

ETAT (<identificateur de processus>,<couleur>)
ETAT (<identificateur de bloc>[,<couleur>])
ETAT (<identificateur de place>[,<couleur>])

Ces 3 primitives retournent VRAI lorsque le processus, le bloc ou la place sont marqués avec éventuellement la couleur spécifiée.

Exemple : Considérons le bloc "B" constitué des places P_1, P_2, ..., P_n où transitent les marques de couleur C_1, ..., C_p.



La primitive ETAT (B,C_j) représente l'abréviation du test suivant :

$M(P_1)(C_j) > 0$ **ou** ... **ou** $M(P_n)(C_j) > 0$

La primitive ETAT (B) correspond à :

$(M(P_1)(C_1) > 0$ **ou** ... **ou** $M(P_n)(C_1) > 0)$

ou ⋮

$(M(P_1)(C_p) > 0$ **ou** ... **ou** $M(P_n)(C_p) > 0)$

Nous définissons également les primitives :

ACTIVITE (<identificateur de processus>)
ACTIVITE (<identificateur de bloc>)
ACTIVITE (<identificateur de place>)

qui retournent le booléen VRAI quand le processus, le bloc ou la place sont actifs (c'est-à-dire non gelés, voir § II.6.3).

II.6.3 - Les primitives d'action

Elles sont élaborées à partir des fonctions de base :

Ajouter (<entier>,<couleur>,<identificateur de place>)

et

Enlever (<entier>,<couleur>,<identificateur de place>).

Nous définissons ainsi les primitives suivantes :

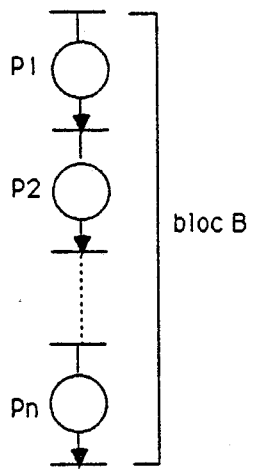
GELER (<identificateur de processus>[,<couleur>])
GELER (<identificateur de bloc>[,<couleur>])
GELER (<identificateur de place>[,<couleur>])

et

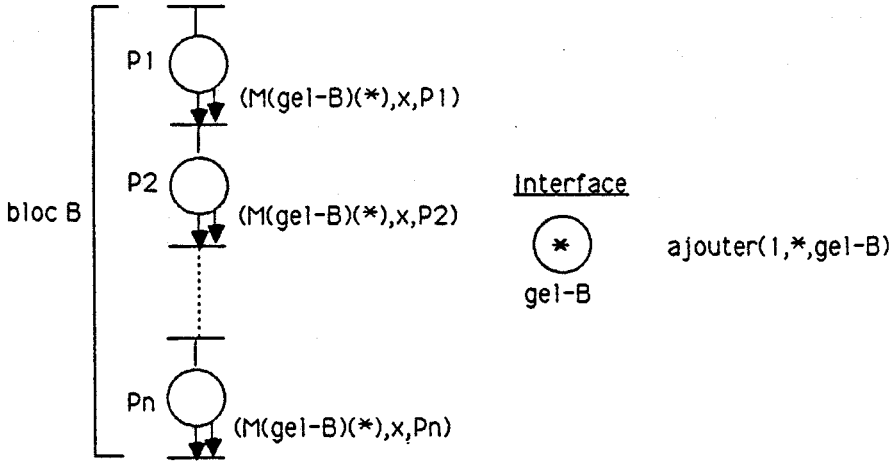
DEGELER (<identificateur de processus>[,<couleur>])
DEGELER (<identificateur de bloc>[,<couleur>])
DEGELER (<identificateur de place>[,<couleur>])

La primitive de gel permet de suspendre toute évolution du sous-graphe spécifié jusqu'à occurrence du dégel correspondant.

Exemple :



La primitive GELER (B,C_1) correspond à l'abréviation de la structure suivante :



Nous définissons également la primitive :

DOMAINE (<identificateur de variable libre globale>,
<liste de couleurs>)

qui remplace le domaine de la variable libre globale par l'ensemble des couleurs spécifiées.

Nous définissons également 3 primitives de recouvrement d'erreurs permettant de reconfigurer le graphe de commande après détection d'un dysfonctionnement de la partie opérative.

INIT-TOTALE : replace le système de commande dans son état initial

INIT (<identificateur de processus>)
et INIT (<identificateur de bloc>)
qui replacent le processus ou le bloc spécifié dans son état initial.

II.7 - Conclusion

La distinction entre décision locale et décision répartie est fondamentale puisqu'elle permet, dès la conception, de localiser le contrôle du système et de répertorier les informations nécessaires à l'élaboration d'une décision.

Intégrer a priori la notion d'architecture répartie assure l'adéquation entre le modèle conçu et sa réalisation matérielle.

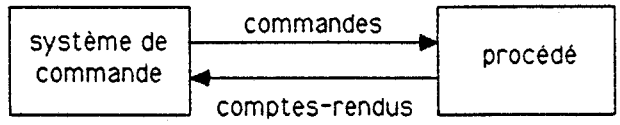
Le moteur d'inférence décrit pour la gestion des règles de surveillance est rudimentaire dans le sens où sa mise en œuvre s'apparente ici à une table de décision simple. Le choix de cette stratégie est bien sûr issu des exemples considérés ayant amené sa mise en œuvre. Ce point est obligatoirement amené à évoluer en fonction des spécifications futures. Si le paramétrage d'un mode dégradé nécessite une démarche basée sur le diagnostic, il est naturel d'envisager à cet effet la mise en œuvre d'un véritable système expert fonctionnant par exemple, en chaînage arrière. Dans le cas où la décision à prendre résulte d'un mécanisme de génération de plan, on pourra préférer un système expert fonctionnant en chaînage avant. Les problèmes qui se poseront lors d'une définition plus précise des fonctionnalités de la partie décisionnelle résulteront sans aucun doute de la nécessité de cohabitation, voire de communication entre diverses stratégies d'inférence, divers systèmes experts.

III - MODELISATION DU PROCEDE

III.1 - Introduction : La séparation entre la Partie Commande et le Procédé

La distinction entre la partie commande et le procédé est une notion apparaissant naturellement lors de l'observation d'un système automatisé. Rappelons que le procédé regroupe l'ensemble des dispositifs matériels alors que la partie commande désigne la partie informatique du système. Dans la suite de notre exposé, le terme de partie commande englobera, par extension, la partie décisionnelle décrite précédemment.

Ces 2 parties coopérent selon le schéma suivant :



Les ordres générés par la partie commande consistent à activer un interface de puissance (actionneurs et pré-actionneurs). Les compte-rendus du procédé regroupent les informations recueillies par tout un ensemble de capteurs.

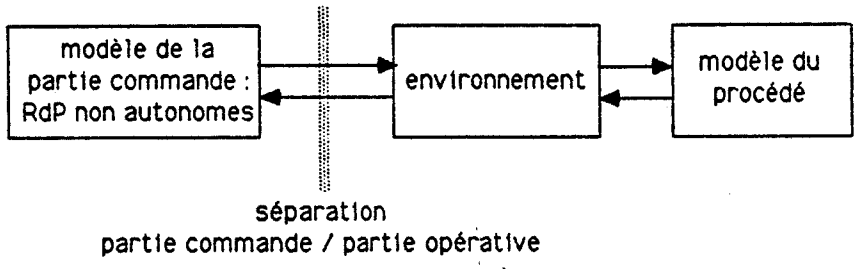
Cette décomposition a évidemment des répercussions sur la description du système. La partie commande est obligatoirement modélisée par **un réseau de Petri non-autonome** au sens où son évolution ne dépend plus uniquement de l'état du réseau mais également de l'état d'un environnement associé au procédé.

Cet environnement est alors représenté par un ensemble d'opérateurs et un ensemble de variables mémorisant l'état instantané du procédé. L'état de ces variables est modifiable par activation des opérateurs.

La partie commande gère le séquencement de ces opérateurs. Les places du graphe de Petri sont associées aux actions et les compte-rendus correspondent à des conditions supplémentaires de franchissement intervenant au niveau de l'interprétation des transitions /BRA 83/.

Le modèle du procédé gère les modifications des variables de l'environnement par activation des opérateurs associés à l'évolution de l'état des capteurs du système.

Globalement, le modèle du système peut être représenté comme suit :



A ce sujet, il existe dans la littérature, une confusion sémantique autour du terme "partie opérative". Chez certains auteurs, la "partie opérative" concerne uniquement l'environnement alors que d'autres y intègrent le mécanisme d'évolution induit par le procédé, c'est-à-dire le modèle du procédé. Afin d'éviter toute confusion, nous parlerons d'environnement et de modèle du procédé.

La séparation entre d'une part le modèle non autonome de la partie commande et d'autre part, l'environnement associé au modèle du procédé est communément appelée principe de séparation entre la partie commande et la partie opérative. Si ce principe est communément admis dans la littérature, la limite entre ces 2 domaines par contre, ne peut être précisée avec rigueur.

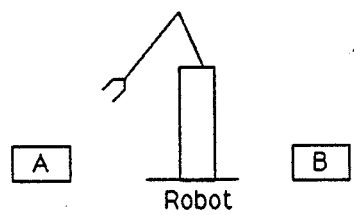
III.2 - Critère de séparation entre la partie commande et le procédé

A priori, la limite "naturelle" entre ces 2 parties (c'est-à-dire la localisation de l'environnement) peut être associée à l'interface constitué par les capteurs et les actionneurs, séparant les structures productives et logicielles du système. Ce qui est programmé constitue la commande et le reste, le procédé. Ce critère de séparation correspond à la réalité dans le cas d'automatismes simples (par exemple, commande d'un vérin jusqu'à détection du capteur fin-de-course). Dans le cas de machines plus complexes (robots, centre d'usinage, ...),

l'adjonction des armoires de commande numérique rend ce critère beaucoup moins significatif. Lors de la commande de déplacement du bras d'un robot 5 axes, par exemple, les informations émises par les codeurs associés aux axes n'ont individuellement aucune signification. Certains auteurs /ANA 86/introduisent le concept de "capteurs logiciels" permettant le prétraitement des informations brutes. Par dualité, les "actionneurs logiciels" permettent la décomposition des ordres en actions élémentaires. Ainsi, la donnée de points particuliers de la trajectoire du bras du robot est traduite en actions simples, éxécutables par les moteurs commandant les axes du robot.

Le critère de séparation entre la partie commande et le procédé (et donc la localisation et la définition de l'environnement) ne peut plus s'exprimer selon des considérations technologiques. En effet, les commandes numériques opèrent une intégration au niveau des ordres et des compte-rendus. Cette notion est intéressante puisque, étendue au concept de "macro-commande", elle permet d'éviter, dès la phase de conception, la définition détaillée des séquences de commandes élémentaires. Dans ces conditions, la limite avec le modèle du procédé dépend du niveau de complexité adopté pour la description du modèle.

Exemple : Considérons le sous-système composé d'un robot manipulant des pièces entre 2 points A et B appartenant à un système de convoyage quelconque.



Si l'architecture du système est organisée selon la structure décrite Fig. 10, le processus de commande de transfert d'une pièce de A vers B peut être modélisé selon le graphe de la Fig. 11.

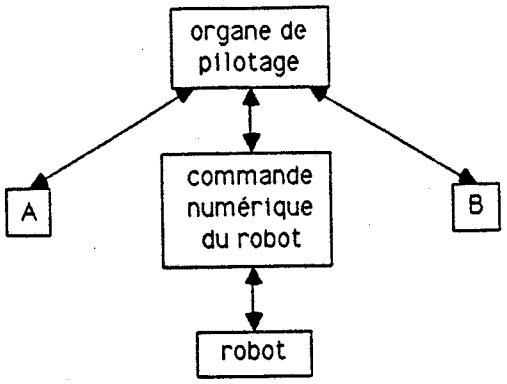
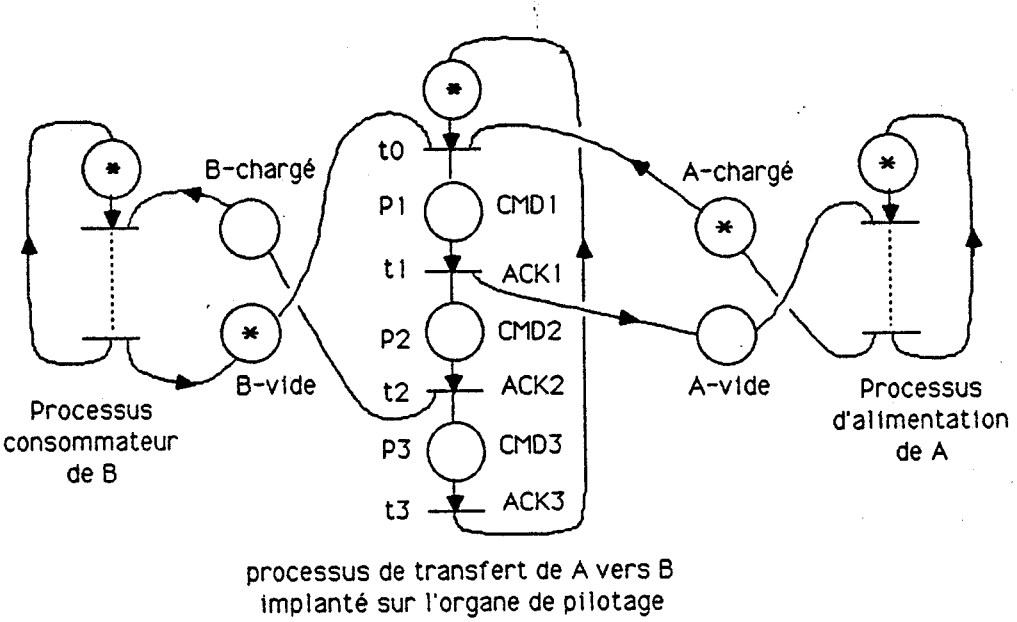


FIGURE 10



processus de transfert de A vers B implanté sur l'organe de pilotage

FIGURE 11

La commande CMD1 correspond à la séquence :

- trajet à vide, depuis la position de repos vers A,
- saisie de l'objet en A.

La commande CMD2 correspond à la séquence :

- trajet en charge de A vers B,
- dépôt de l'objet en B.

La commande CMD3 correspond au retour à la position de repos.

Le degré de description de la Fig. 11 est suffisant pour concevoir la commande globale du système puisque les intéractions avec le reste du système (les producteurs / consommateurs A et B) sont représentés. Les couples (CMD_i,ACK_i) sont considérés comme appartenant à l'environnement associé au procédé.

Une spécification plus précise en vue de l'implantation sur la commande numérique du robot, nécessite la définition des processus et des liaisons correspondants :

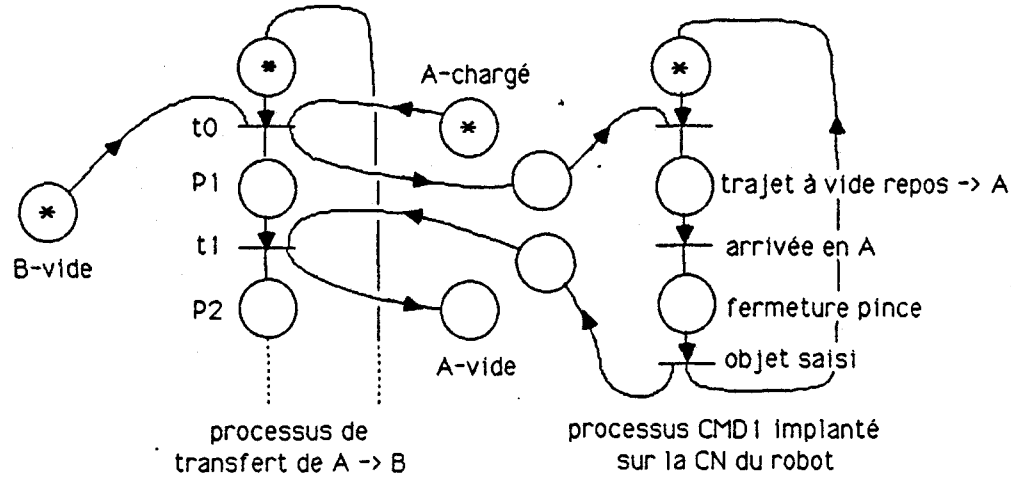


FIGURE 12

La définition des processus CMD1 (Fig. 12), CMD2 et CMD3 implantés sur la commande numérique du robot engendre une nouvelle localisation de l'environnement et donc de la limite avec le procédé.

En conclusion, le concept de procédé et donc d'environnement tel que nous l'entendons, est relatif au degré de spécification du système de commande, organisé en couples "(génération d'un ordre, accusé-réception de fin d'action)". Augmenter le degré de spécification atteint revient à définir de nouveaux processus selon un mécanisme proche de la notion de sous-programme informatique en architecture répartie.

III.3 - Etude des différents modèles théoriques de l'environnement et du procédé

III.3.1 - Introduction

La partie commande est modélisée, nous l'avons vu, par un graphe de Petri non autonome dont l'évolution dépend de l'environnement. Ce dernier est constitué des variables et opérateurs représentant le procédé. L'appréhension de cet environnement est nécessaire dans la phase de conception et d'évaluation des performances du système. La durée de vie de ce modèle est bien sûr limitée à cette phase de conception puisque l'environnement réel s'y substituera dans la phase d'implantation.

Il existe dans la littérature, 2 approches différentes pour appréhender cet environnement :

(i) l'extension du modèle RdP afin d'y intégrer la dynamique imposée par le procédé : les réseaux temporisés,

(ii) la création d'un modèle spécifique du procédé interfacé au modèle RdP interprétés.

III.3.2 - Les réseaux de Petri temporisés /RAM 73/ /MOA 76/ /SIF 77/ /CHR 83/

L'environnement fournit une référence de temps commune, et seule la durée des actions est prise en compte pour décrire le comportement dynamique du système. Dans de tels réseaux, les marques possèdent 2 états (disponible ou indisponible), ce qui génère des conditions supplémentaires pour l'activation des transitions.

Le procédé est alors réduit à un ensemble de compteurs référencés par l'horloge commune.

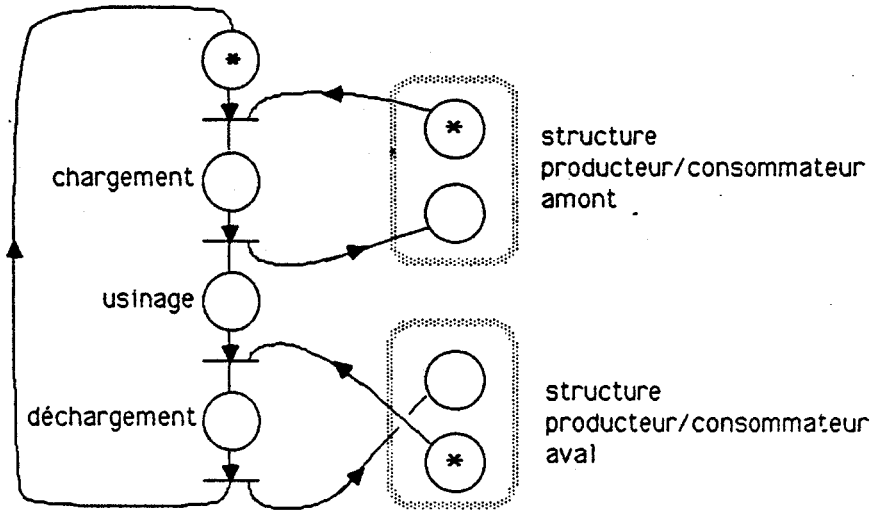
Nous ne donnerons pas ici la définition formelle des RdP temporisés. Il existe à ce sujet 2 écoles, selon que la temporisation est associée aux places ou aux transitions du réseau.

L'avantage de la temporisation est de permettre, dès la phase de conception, une évaluation quantitative des performances du système, tout en préservant la structure du graphe de commande. Cette solution

est simple, mais son utilisation s'appuie sur le critère suivant : «toute modification de l'état du procédé résulte directement de l'application d'une commande».

Ce critère s'applique effectivement aux processus de commande des **machines.**

Exemple : Processus de commande d'une machine-outil.



Associer des temporisations aux places "chargement", "usinage" et "déchargement" est suffisant pour représenter le procédé puisque l'évolution de l'environnement associé peut être partitionné (au sens mathématique du terme) en couples disjoints du type "(commande, accusé-réception de fin d'action)".

Par contre, lorsque cette partition n'existe pas, la temporisation s'avère être un outil mal adapté à la représentation du procédé.

Exemple :

Le transfert d'une pièce sur un convoyeur entre les points A et B de la Fig. 13, n'est la conséquence directe d'aucune commande et induit cependant une temporisation sur le système.

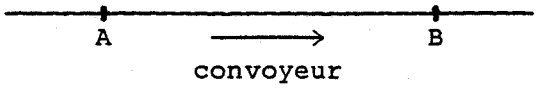


FIGURE 13

La quantification de la durée du trajet ne peut être associée simplement à aucun processus de commande. En effet, la gestion du flux de pièces entre A et B dépend d'un mécanisme passif : le convoyeur. La durée du trajet AB résulte effectivement du dépôt en A mais aussi de l'état et de la dynamique du convoyeur : arrêt du convoyeur, vitesse de transfert, existence de butées de débrayage intermédiaires, etc ...

III.3.3 - Utilisation d'un modèle spécifique du procédé

Supposons, dans un premier temps, défini le modèle du procédé pour nous attacher à son interfaçage avec le graphe de commande (définition de l'environnement). Ceci nous amène à utiliser le modèle des RdP interprétés. C'est ce modèle qui a largement inspiré le Groupe Systèmes Logiques de l'AFCET lors de ses réflexions sur la normalisation des outils de spécification du cahier des charges des automatismes logiques. Le modèle GRAFCET /GRA 77/, résultat de ses travaux, a connu une large diffusion dans les milieux industriels où il est de plus en plus utilisé comme outil de conception et d'implantation.

a) Les réseaux de Petri interprétés - Définition /BRA 83/ :

Soient :

D l'ensemble des états de l'environnement
OP un ensemble d'opérateurs ; OP_i : D ⟶ D
PR un ensemble de prédicats sur D ; $pr_i \in PR$: D ⟶ {Vrai,Faux}

A chaque place du réseau est associé un opérateur :

p : P ⟶ OP

A chaque transition sont associés un prédicat et un opérateur :

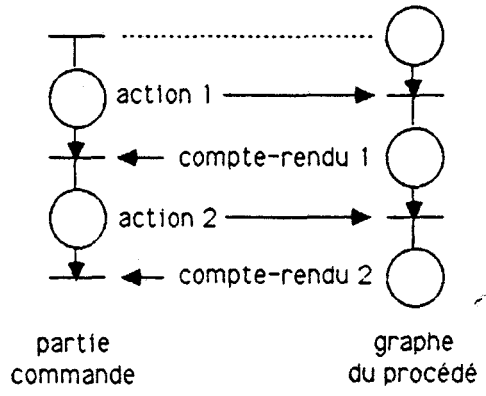
Ψ : T ⟶ PR × OP

Lors de l'étude de l'activation de la transition t, on teste en plus des conditions usuelles sur le marquage amont, si le prédicat $\Psi(t)$ qui lui est associé est vérifié par les valeurs présentes des variables de l'environnement.

Dès qu'un nouveau marquage est atteint, les opérateurs associés aux places nouvellement marquées sont activés.

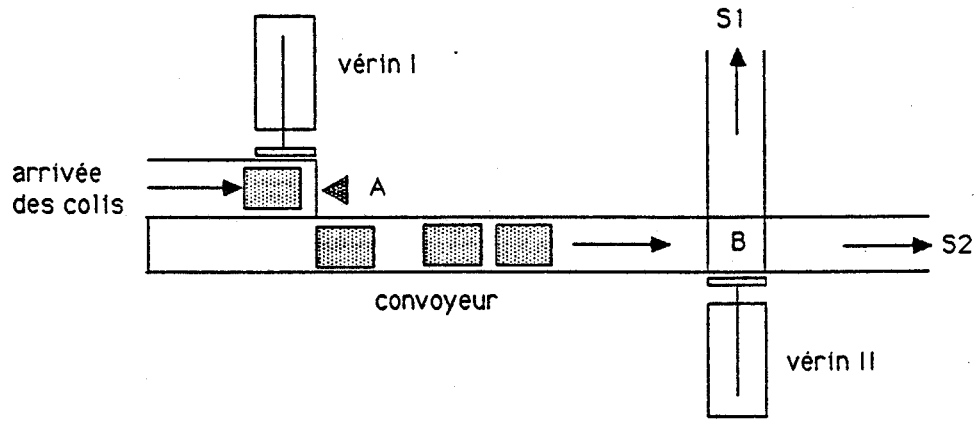
b) Représentation par dualité du procédé :

L'évolution du graphe de commande modélisé par RdP interprétés nécessite un mécanisme de gestion de l'environnement (c'est-à-dire la représentation du procédé). Une méthode purement intéractive où l'opérateur force les entrées n'est pas satisfaisante en raison du haut degré de parallèlisme des systèmes industriels qui sont généralement de grande dimension. La solution la plus réaliste consiste à utiliser un "simulateur" du procédé. Celui-ci peut alors être modélisé par un graphe dual de la partie commande et générant les compte-rendus.



Le système PIASTRE /CAR 83/ utilise cette méthode pour la conception des grafcets de commande, le grafcet dual étant généré à partir du grafcet de commande. Citons également les travaux du Centre d'Automatique de l'U.S.T.L. /DEF 86/ sur l'utilisation d'une famille d'automates d'états finis, pour décrire les évolutions de certaines grandeurs observées par l'intermédiaire des capteurs.

c) Un exemple de modélisation par RdP interprétés :



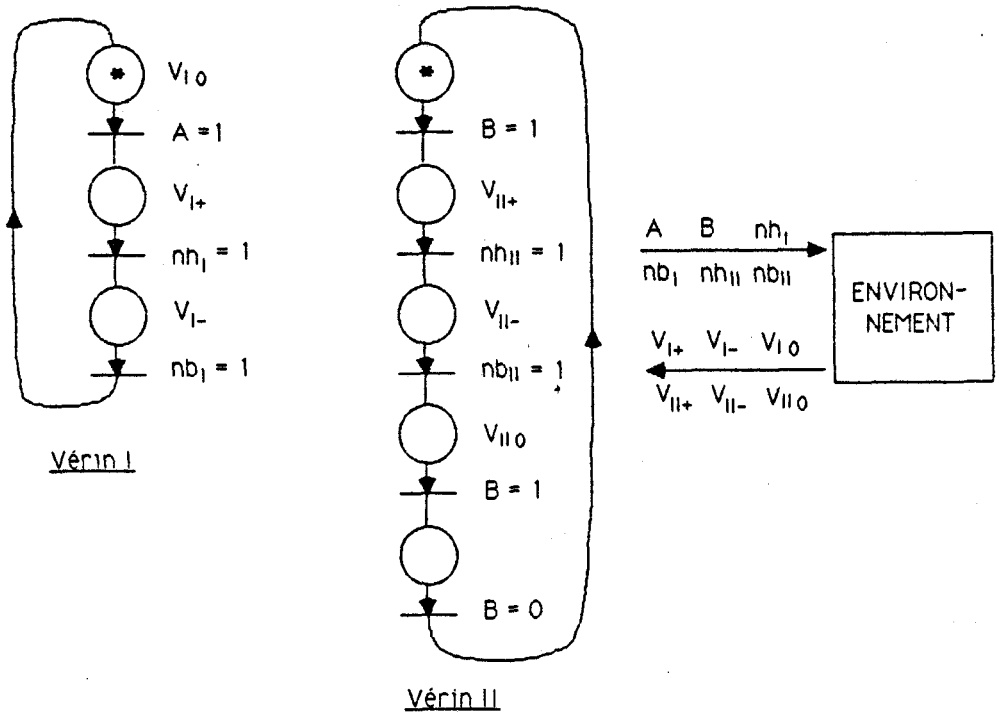
Le vérin I, après détection d'un colis (capteur A), pousse ce colis sur le convoyeur. Les colis sont alors amenés devant le capteur B où s'opère un "dispatching" vers les systèmes avals S1 et S2.

Appelons nh_i et nb_i, les indicateurs de fin de course haut et bas du vérin i (i = I ou II). Les commandes des vérins sont V_{i+}, V_{i-} et V_{io} (vérin au repos).

L'environnement est représenté par 6 variables : A, B, nh_I, nb_I, nh_{II} et nb_{II} associées aux différents capteurs et valant 1 en cas de détection et 0 sinon.

1er cas : Les colis, tous identiques, sont dispatchés alternativement vers S1 et S2.

Le graphe de commande peut alors être réalisé comme suit :



Ici, l'environnement du réseau interprété correspond exactement à l'état des capteurs du procédé et, par voie de conséquence, tous les opérateurs utilisés coïncident avec les commandes du système. Nous dirons que l'environnement est **"externe"**.

2ème cas : Supposons maintenant que les colis soient de 2 types S1 et S2 et le dispatching sélectif.

Pour reconnaître le type des colis, on remplace le capteur A par un dispositif de reconnaissance quelconque. La variable A associée à ce capteur prendra 3 valeurs : 0 (rien), 1 (colis de type S1) ou 2 (colis de type S2). Le capteur B par contre, reste un simple capteur de présence. Dans ces conditions, la nécessité d'identifier le type d'un colis arrivant devant B impose la mémorisation des colis passant devant A.

Nous définirons pour cela une structure linéaire "EN COURS" mémorisant la séquence ordonnée des colis entre A et B.

L'opérateur "ENTRER" correspondant à l'arrivée d'un nouveau colis en A, ajoute la valeur de la variable A au début de la structure "EN COURS".

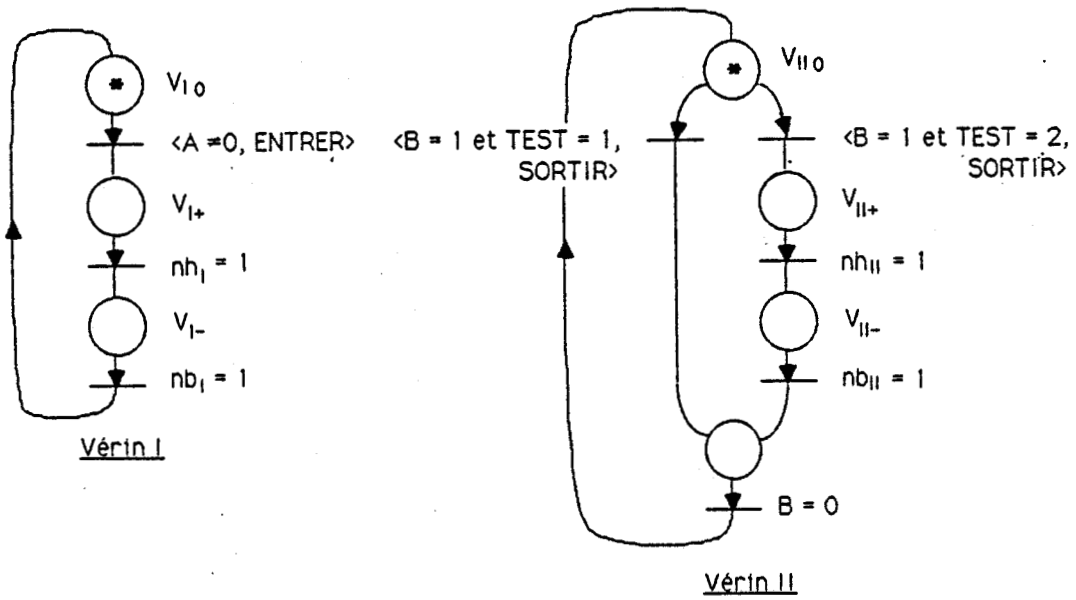
Exemple : EN COURS = (1,2,1,1) et A = 2
L'opérateur ENTRER aura pour effet de modifier la variable EN COURS :
EN COURS = (2,1,2,1,1).

Le prédicat "TEST" permet d'évaluer le type d'un colis détecté en B par lecture du dernier élément de la variable "EN COURS".

Exemple : **SI** EN COURS = (1,2,1,1) **ALORS** "TEST = 1" est vrai.

L'opérateur "SORTIR" correspond au départ d'un colis de B. Le dernier élément de la variable "EN COURS" est supprimé.

Exemple : EN COURS = (1,2,1,1)
L'effet de l'opérateur "SORTIR" est le suivant :
EN COURS = (1,2,1)



La flexibilité relative à la distinction du type des colis définit un environnement particulier dont le rôle est de fournir au graphe de commande ce que nous appellerons une "**image opérative**" du procédé.

Il s'agit ici d'observabilité d'état du modèle de commande. Cette "image opérative" constitue l'image suffisante de l'état du procédé qui est nécessaire à l'évolution de la commande.

III.3.4 - Conclusion

La temporisation et l'interprétation sont 2 méthodes intéressantes pour appréhender le procédé. la temporisation permet une approche simple mais suppose l'adéquation exacte entre commande et évolution du procédé. L'interprétation permet une conception très proche du système réel mais suppose l'existence d'un modèle spécifique du procédé. Dans les 2 cas, la modélisation se heurte au problème posé par les structures "passives" de communication entre les machines (en particulier les convoyeurs) dont l'existence nécessite la définition, pour permettre l'évolution de la partie commande, d'une "image opérative" du procédé, au sens de l'observabilité.

Nous proposons une méthodologie de structuration de la commande basée non seulement sur la définition des processus de commande, mais organisée également selon la structure des mécanismes de transfert du système physique. Sur cette base, la temporisation permettra l'évaluation simple des performances du système, et l'utilisation d'un modèle spécifique du procédé, l'élaboration de statistiques de fonctionnement et l'intégration des dysfonctionnements des organes productifs. Cette méthodologie donne par ailleurs, un cadre de modélisation systématique et structuré évitant la trop grande multiplicité des choix qui résultent de la mise en œuvre du concept large d'interprétation.

III.4 - Méthodologie d'une structuration orientée graphe de transfert

III.4.1 - Définition du graphe de transfert

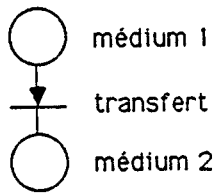
Un système de production flexible est organisé en fonction du conditionnement, éventuellement simultané, de plusieurs types de pièces ($C_1, C_2, \ldots, C_n$). Chaque pièce est caractérisée par une séquence d'opérations particulière (la gamme d'usinage par exemple). La réalisation de cette séquence d'opérations nécessite a priori un ordre de passage sur divers organes du procédé que nous appellerons **gamme de conditionnement.**

Exemple : La réalisation d'une pièce usinée nécessite des opérations de tournage et de fraisage. La gamme de conditionnement sera par exemple : tour/fraise/tour.

Pour réaliser sa gamme de conditionnement, une pièce doit accomplir une trajectoire particulière dans l'atelier comprenant évidemment les machines de sa gamme de conditionnement, mais aussi éventuellement des zones de stockage intermédiaires entre ces organes, les points d'entrée et de sortie de l'atelier, ...

Nous regrouperons les trajectoires de tous les types de pièces circulant dans le système sur un réseau de Petri particulier appelé **graphe de transfert et de conditionnement.** Chaque place de ce graphe ou **média** correspond à un lieu et un état précis (entrée, machine, zone de stockage, ...) et chaque transition à une évolution des pièces entre ces différents médias.

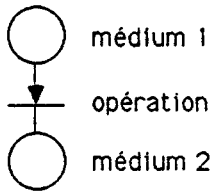
Exemple :



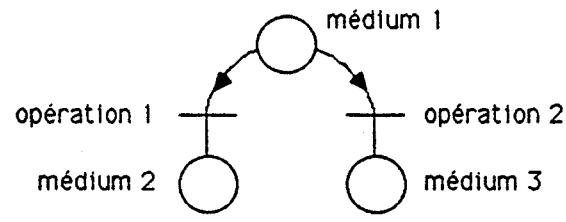
III.4.2 - Critères de modélisation du graphe de transfert

Les primitives permettant de construire le graphe de transfert sont les suivantes :

*** La séquence :**

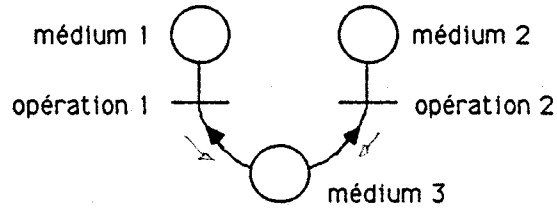


*** L'aiguillage sélectif :**

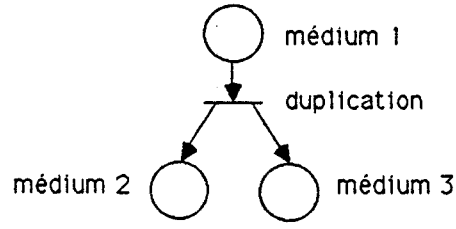


Le choix de la trajectoire depuis le médium 1 dépend, par exemple, du type de la pièce.

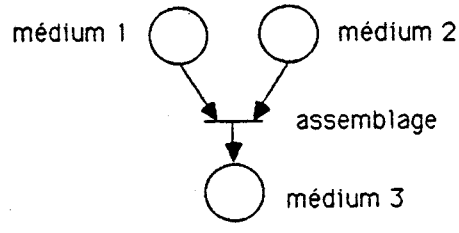
*** La conjonction :**



*** La duplication :**

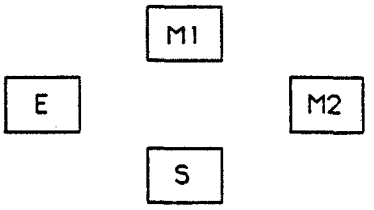


*** L'assemblage :**

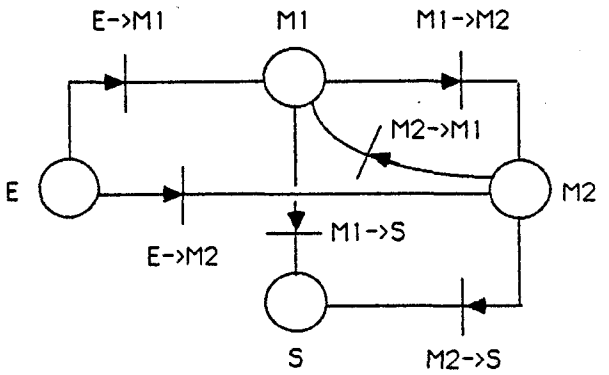


III.4.3 - Exemple de graphe de transfert

Considérons l'ilôt flexible qui sera développé par la suite (Chapitre III), composé de 2 machines M1 et M2, d'un point d'entrée E et d'un point de sortie S.



4 types de pièces circulent dans cet ilôt selon que'elles sont usinées exclusivement sur M1, exclusivement sur M2, sur M1 puis sur M2, ou sur M2 puis sur M1. Le graphe de transfert associé sera le suivant :



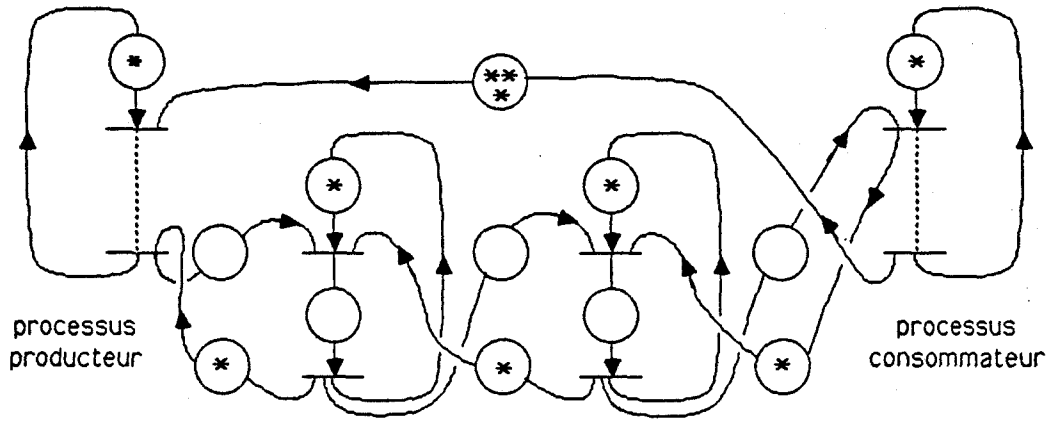
Notons qu'aucune hypothèse n'a été faite sur la nature du (ou des) mécanismes de transfert entre les différents organes. Le graphe de transfert est purement fonctionnel.

III.4.4 - Elaboration du graphe de commande à partir du graphe de transfert

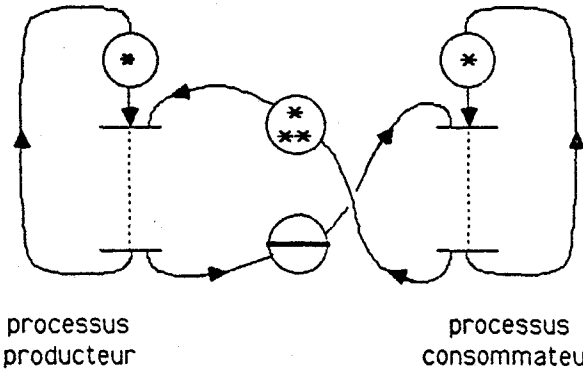
Après analyse du graphe de transfert fonctionnel, le concepteur choisit a priori, la nature des mécanismes de transports susceptibles d'assurer les transferts entre les différentes machines. Il peut également décider de dupliquer certains organes qu'il considère comme des

goulots d'étranglement, ou d'introduire des zones de stockage intermédiaires. Le graphe de transfert est alors modifié en conséquence.

Le graphe de transfert résultant n'est en général pas structuré. Le passage au modèle RdP SAC du graphe de commande constitue l'une des préoccupations actuelles du Laboratoire d'Informatique Industrielle de l'I.D.N.. Les processus du graphe de commande communiquent entre eux grâce aux structures de liaisons de type producteur / consommateur ou synchronisation. Afin d'améliorer les performances d'un système où les durées opératoires sont parfois importantes, il est intéressant d'introduire, à ce niveau, des zones tampons de stockages. Ceci nous amène à étendre la structure classique du producteur / consommateur en introduisant le concept de file d'attente. Ce mécanisme peut être modélisé par le réseau de Petri **structuré** suivant :



Cette file d'attente est représentée par la primitive graphique ⊖ (macro-place de type "FIFO" first-in, first-out).



III.4.5 - Conclusion

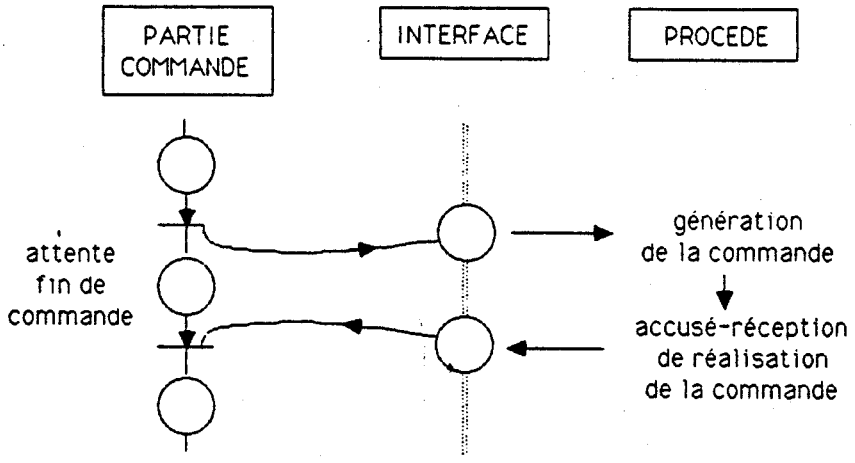
L'application de cette méthodologie basée sur le graphe de transfert fait apparaître des processus permettant de gérer le flux des pièces du procédé. Ces processus particuliers (par exemple ceux associés aux organes de transfert "passifs") n'existent pas lors d'une approche classique de modélisation puisqu'ils n'ont pas d'effet de commande bien qu'ils interviennent sur la dynamique d'état du système. Leur définition, associée au concept de coloration des marques du graphe permet de représenter aisément cette "image opérative" nécessaire à la commande que nous avons précédemment introduite. Elle constitue donc une première étape indispensable pour l'appréhension du procédé. Sur la base de ce modèle, nous pouvons définir les 2 approches permettant de représenter effectivement le procédé : la temporisation et l'utilisation d'un modèle spécifique.

III.5 - Interface du graphe de commande et du procédé

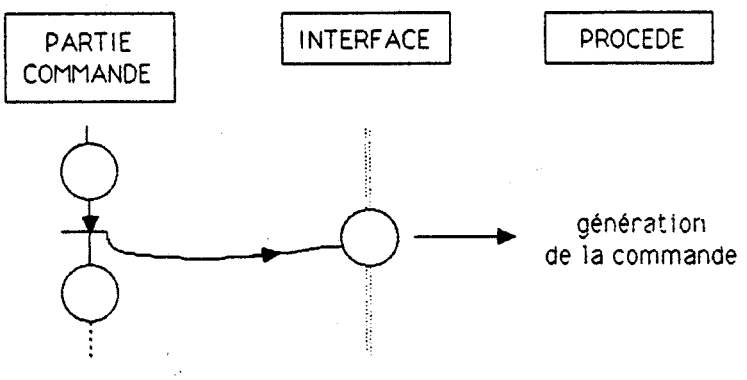
Le modèle du graphe de commande (RdP SAC) est interfacé au modèle du procédé au moyen d'un ensemble de places de type "po" dont le marquage modélise l'environnement associé au procédé. L'"image opérative" étant intégrée au graphe de commande (processus de transfert, macro-places FIFO, et coloration), cet environnement est essentiellement "externe". Aucun opérateur n'est donc nécessaire à cet interfaçage. Les prédicats sont réalisés grâce à l'aspect adaptatif du modèle de la partie commande. L'interprétation du graphe n'est donc pas fondamentalement nécessaire.

3 types de structures permettent cet interfaçage :

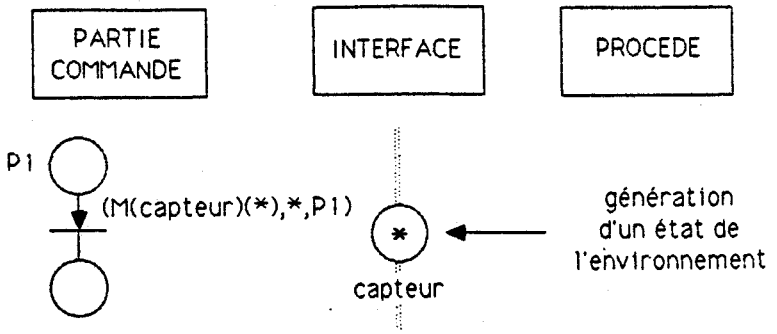
i) La **structure symétrique** : génération d'une commande, attente de l'accusé-réception correspondant.



ii) La **structure asymétrique** : la réalisation de la commande n'engendre aucun accusé-réception. C'est le cas, par exemple, pour l'ordre de fermeture de la pince d'un robot quand il n'y a pas de capteur de pression pour détecter la saisie de l'objet.



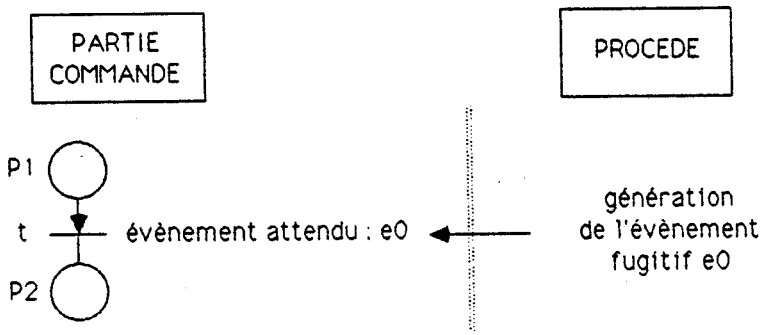
iii) La **structure duale de la précédente** : la modification de l'environnement associé au procédé ne dépend d'aucune commande particulière mais induit cependant une évolution du graphe de commande. C'est le cas, par exemple, des capteurs de présence sur un convoyeur.



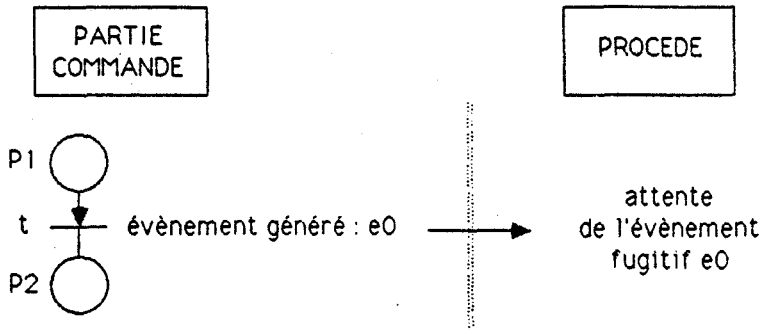
Le marquage des places d'interface (de type "po") est forcé dans ce cas par le modèle du procédé.

Remarque :

Le principe de communication sous-jacent à ces 3 structures est basé sur la mémorisation d'états. Certains systèmes cependant, communiquent par "évènements fugitifs" (ex : détection d'un front montant). Cette solution est rarement choisie dans le contexte de la productique, car facilement parasitée. Son intégration au modèle est réalisée par interprétation des transitions du graphe.



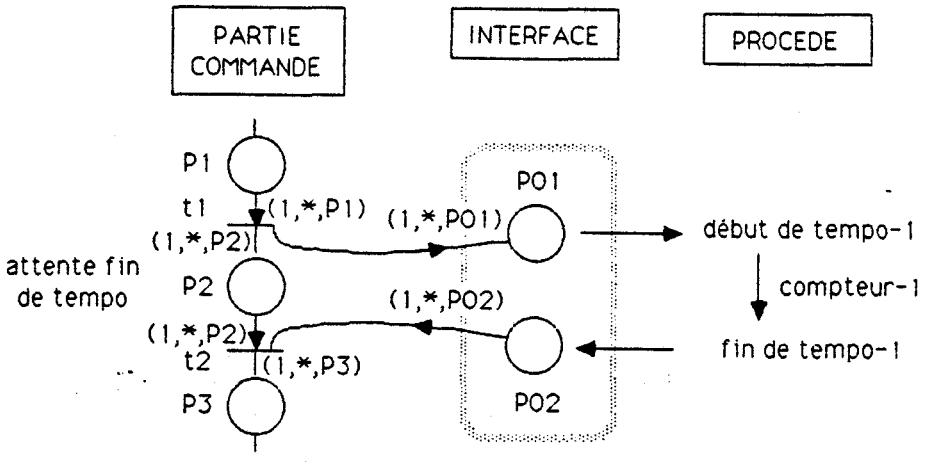
La présence de l'évènement e0 attendu par la transition t est une condition supplémentaire d'activation de la transition.



Le tir de la transition t génère l'évènement e0 dont l'occurence fugitive induit éventuellement une évolution du procédé.

III.6 - La temporisation

Le procédé est réduit à un ensemble de compteurs reliés au graphe de commande par la structure fondamentale suivante :

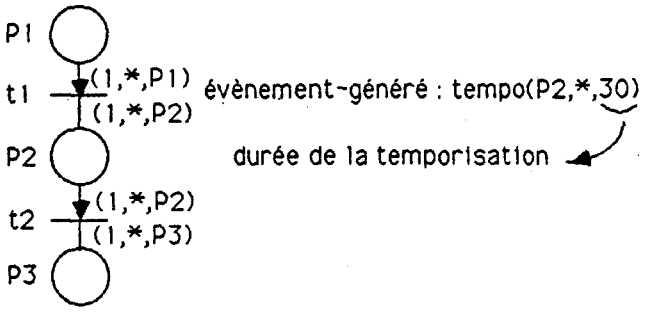


La présence d'une marque de couleur "*" dans la place PO1 active un compteur initialisé à la valeur définie pour cette temporisation, ce qui "consomme" la marque présente en PO1. Ce compteur est décrémenté, et son passage par zéro provoque l'ajout d'une marque dans la place PO2. La transition t2 est alors activable.

Afin de faciliter l'écriture du graphe, nous avons agrégé cette structure par définition d'une primitive de temporisation assimilée à un évènement généré par le tir de la transition t1 en amont. La syntaxe de cette primitive est la suivante :

TEMPO (<identificateur de place>,<identificateur de couleur>,<durée>).

La structure du graphe de commande reste donc inchangée :



Le tir de la transition t1 déclenche l'initialisation d'une temporisation de 30 secondes sur la marque de couleur "*" apparaissant dans la place P2. Pendant ces 30 secondes, la marque "*" sera "gelée" dans la place P2 et la transition t2 ne sera donc pas activable relativement à cette couleur. Par contre, si le processus n'est pas structuré (non sauf en particulier), la transition t1 peut être réactivée.

III.7 - Utilisation d'un modèle spécifique du procédé

III.7.1 - Introduction

L'intérêt de disposer d'un modèle spécifique du procédé est, nous l'avons vu, de se substituer aux organes physiques du système réel. Le degré de complexité du modèle à utiliser dépend essentiellement du degré de détail requis par l'utilisateur. En effet, ce modèle peut être employé pour plusieurs raisons :

i) L'élaboration de statistiques de fonctionnement lors d'une simulation du système : informations sur le taux d'engagement des machines, sur la durée des stockages intermédiaires, sur le temps de conditionnement des pièces, ... Notons que le recueil de statistiques associées aux pièces circulant dans l'atelier n'est pas supportable par un modèle de type RdP, même coloré, puisqu'il nécessite la reconnaissance individuelle des pièces.

ii) La vérification de l'adéquation entre le séquencement des commandes générées et la réaction des machines.

iii) La simulation des défaillances des machines afin d'en analyser les conséquences sur l'évolution de la commande, puis de valider les mécanismes prévus de recouvrements d'erreurs et les régimes transitoires lors du passage en mode dégradé, ...

Nous avons choisi de ne pas représenter la partie opérative par RdP. En effet, la puissance algorithmique du modèle RdP devrait être étendue pour pouvoir, par exemple, individualiser les pièces circulant dans l'atelier. Le modèle résultant ne pourrait plus être structuré puisque les intéractions entre les divers organes de production sont généralement complexes. Dans ces conditions, le degré de difficulté pour représenter le procédé risque d'être supérieur à celui de la modélisation de la partie commande, ce qui est paradoxal puisque la durée de vie du modèle décrivant le procédé est limitée à la phase de conception et d'évaluation du système.

Pour ces raisons, notre choix s'est porté sur une représentation du procédé par catégories génériques.

III.7.2 - Description par catégories génériques

a) Introduction sur les catégories génériques :

En Informatique, on distingue généralement la partie active d'un programme (les procédures) de la partie passive (les données sur lesquelles agissent les procédures). Le séquencement des instructions réalise l'appel des procédures en leur fournissant les données nécessaires à leur exécution. Dans les langages basés sur l'utilisation de catégories génériques, les données et les procédures ne sont plus traitées indépendamment, mais au contraire organisées en **entités** (ou **objets**) qui regroupent les aspects statisques et dynamiques au sein d'une même structure. Chaque objet est donc caractérisé par un ensemble de données appelées "**champs**" et un ensemble de procédures spécifiques appelées "**méthodes**".

Note : La programmation basée sur l'utilisation de catégories génériques décrites ici, s'apparente aux techniques de "Frame" et s'inscrit dans le contexte plus général des **langages objets** /BRA 86/ /HUL 84/.

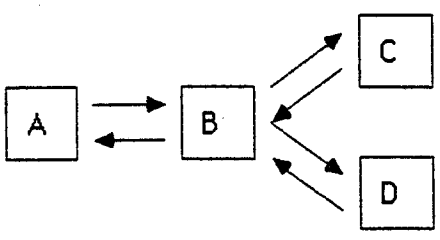
*** Les classes d'objets :**

Les objets possédant des comportements communs sont regroupés en **"classes"**. Chaque classe est le modèle général à partir duquel sont **instanciés** les représentants de cette classe. La définition de la classe générique "objet", qui est la schéma le plus général que l'on puisse envisager, permet d'organiser les diverses classes, sous-classes et instances d'objets en une hiérarchie arborescente. Le mécanisme de recopie virtuelle, encore appelé **héritage** permet alors, à chaque objet, d'hériter par défaut, en plus de ses champs et méthodes particuliers, des champs et méthodes de ses ancêtres.

*** La communication par envoi de messages :**

Arrivé à ce stade de définition, les objets apparaissent comme de entités autonomes capables d'évoluer indépendamment les unes des autres. Il est bien évident que les divers objets d'une même application (du même "programme") sont amenés à interagir en vue d'un objectif commun (les "résultats" du programme). La notion classique d'appel de procédure est remplacé ici, par la notion de communication par envoi de messages. Les objets peuvent communiquer entre eux par envoi de messages selon 2 mécanismes :

(i) La transmission :

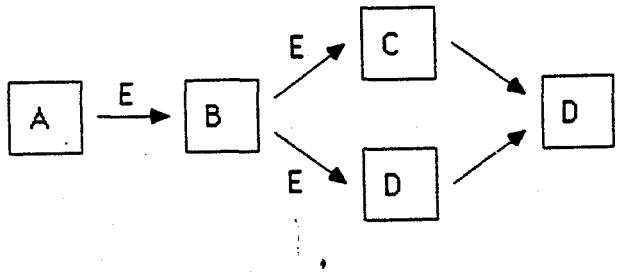


L'objet A émet un message vers B ce qui déclenche l'émission de messages vers C et D. Chaque objet recevant un message génère une réponse à l'émetteur.

(ii) La continuation :

Lors de l'émission d'un message, le destinataire de la réponse est précisé.

Exemple :



Les objets que nous utilisons pour représenter le procédé utilisent exclusivement le mécanisme de communication par transmission. Précisons également, que la partie commande communique avec les différents objets du procédé selon le même mécanisme (envoi de commandes). A ce titre, la partie commande, vue du procédé peut donc être assimilée elle aussi à un objet.

La syntaxe d'un message est la suivante :

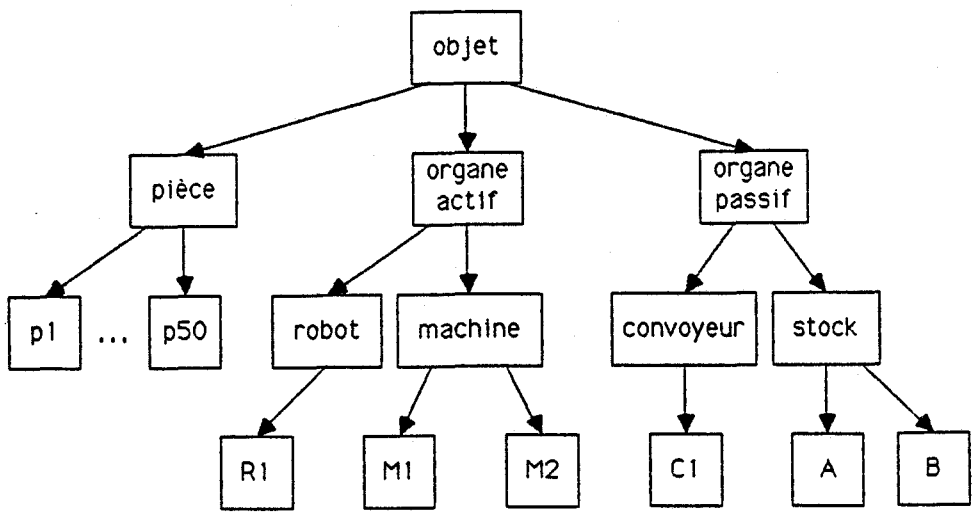
(⟹ <objet> <méthode> [<liste d'arguments>])

L'envoi de ce message consiste à activer la <méthode> de l'<objet> spécifié. Les valeurs éventuellement précisées dans la <liste d'arguments> correspondent au passage des paramètres nécessaires à l'activation d'une procédure dans le cas de la programmation classique.

b) Description du procédé par catégories génériques :

Décrire le procédé par catégories génériques présente un double avantage : d'une part la modularité inhérente à cette technique de programmation et, d'autre part, l'aspect descriptif lié au type déclaratif de l'outil proposé.

L'étude des différents organes présents dans un atelier flexible, fait apparaître un nombre restreint d'objets que l'on peut classer comme suit :



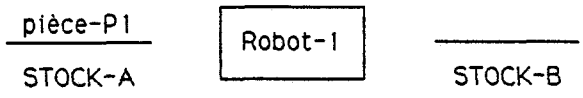
La définition du procédé spécifique à l'application envisagée consiste alors à décrire chaque objet, c'est-à-dire préciser ses caractéristiques (valeurs particulières de ses champs) et définir de façon locale ce qu'il sait faire (ses méthodes : un robot sait déposer une pièce, saisir ; une machine sait usiner, ...).

c) Exemple :

Nous allons présenter, par le biais d'un exemple, la mise en œuvre des techniques de programmation par catégories génériques pour décrire le procédé d'un système de production.

Nous nous attacherons plus particulièrement à définir la portée des méthodes associées aux différents objets, sans nous astreindre au formalisme d'un langage particulier. Une approche plus précise est réalisée dans le Chapitre III de ce mémoire où est traité un exemple complet de modélisation.

Exemple du transfert d'une pièce entre 2 lieux de stockage par l'intermédiaire d'un robot



* Définition de la classe STOCK :

STOCK	est un : objet	champs	
	occupation : <()>/<identificateur de pièce>		
	retrait : restitue la valeur du champ occupation et affecte le champ occupation à ()	méthodes	
	dépôt (pièce) : affecte la valeur pièce au champ occupation		

STOCK-A et STOCK-B sont 2 instances de la classe STOCK

STOCK-A	est un : stock	
	occupation : pièce-P1	
STOCK-B	est un : stock	
	occupation : () stock-B est initialement vide	

* Définition de la classe ROBOT :

ROBOT	est un : objet	champs	
	position : <stock-A>/<stock-B>/<repos>		
	occupation : <()>/<identificateur de pièce>		
	trajet (origine, destination) méthode de passage depuis la position "origine" vers la position "destination" affecte la valeur "destination" au champ "position"	méthodes	
	saisie : affecte au champ "occupation" la réponse au message (⟹ position retrait)		
	dépôt : génère le message (⟹ position dépôt occupation) affecte le champ "occupation" à la valeur ()		

ROBOT-1 est une instance de la classe ROBOT

ROBOT-1	est un : robot	
	position : repos occupation : ()	ROBOT-1 est initialement au repos et à vide

* Pièce-P1 est une instance de la classe PIECE que nous ne préciserons pas.

* La commande de transfert de la pièce-P1 depuis STOCK-A vers STOCK-B correspond à la séquence de messages suivante :

1) (⟹ robot-1 trajet (repos,stock-A))

Le champ "position" de ROBOT-1 passe de la valeur "repos" à la valeur "STOCK-A". Le robot est correctement positionné devant la zone de stockage STOCK-A.

2) (⟹ robot-1 saisie)

L'activation de la méthode "saisie", génère l'envoi du message :

(⟹ STOCK-A retrait)

qui restitue en réponse la valeur du champ "occupation", soit PIECE-P1 et affecte ce champ à ().

Le champ "occupation" du robot-1 est alors affecté à la valeur résultant de la réponse à ce message, soit PIECE-P1. Le robot-R1 a saisi la pièce-P1.

3) (⟹ robot-1 trajet (STOCK-A,STOCK-B))

Le champ "position" de ROBOT-1 passe de la valeur "STOCK-A" à la valeur "STOCK-B". Le robot a effectué le trajet en charge vers STOCK-B.

4) (⟹ robot-1 dépôt)

L'activation de la méthode "dépôt" génère l'envoi du message :

(⟹ STOCK-B dépôt (pièce-P1))

qui affecte le champ "occupation" de STOCK-B à "pièce-P1". Le champ "occupation" de "robot-1" est ensuite affecté à (). Le robot a déposé la pièce-P1 en STOCK-B.

5) (⟹ robot-1 trajet (STOCK-B,repos))

Le champ "position" de robot-1 passe de la valeur "STOCK-B" à la valeur "repos". Après déchargement, le robot est retourné à sa position de repos.

Suite à ce transfert, les instances des objets ont été modifiées de la manière suivante :

ROBOT-1	est un	: robot
	position	: repos
	occupation	: ()
STOCK-A	est un	: stock
	occupation	: ()
STOCK-B	est un	: stock
	occupation	: pièce-P1

La pièce-P1 a été transférée de STOCK-A vers STOCK-B

d) Intégration de l'aspect temporel :

Intégrer l'aspect temporel consiste à associer des durées d'action aux méthodes des différents objets. Chaque action est scindée en 2 méthodes : la méthode correspondant à la réception d'un message et instaurant un état transitoire des champs de l'objet, et la méthode correspondant à la fin de l'action résultant de la réception du message.

Pour temporiser, par exemple, la dynamique du robot décrit en (c), on étend le domaine du champ "position" en introduisant une nouvelle valeur possible "mouvement" correspondant au régime transitoire et on définit un nouveau champ afin de préciser la durée d'un trajet élémentaire.

La classe ROBOT est modifiée de la façon suivante :

ROBOT

est un : objet

position : <STOCK-A>/<STOCK-B>/<repos>/<mouvement>

occupation : <()>/<identificateur de pièce>

durée-trajet : 3Ø ,

trajet (origine, destination)
affecte le champ "position" à la valeur "mouvement"
génère après une durée de 30 secondes le message :
(⟹ robot-1 arrivée (destination))

arrivée (destination)
affecte le champ "position" à la valeur "destination"

Les méthodes "saisie" et "dépôt" supposées instantanées demeurent inchangées

La commande de déplacement depuis la position de repose vers STOCK-A

(⟹ robot-1 trajet (repos,STOCK-A))

se décompose comme suit :

1) Le champ "position" de robot-1 passe de la valeur "repos" à la valeur "mouvement".

2) Envoi différé de 30 secondes du message :
(⟹ robot-1 arrivée (STOCK-A)).

3) Au bout de 30 secondes, robot-1 reçoit ce message. Le champ "position" passe de la valeur "mouvement" à la valeur "stock-A".

L'intégration de l'aspect temporel consiste donc à instaurer un mode de continuation temporisée des messages générés.

e) Lien avec le modèle de la partie commande :

La liaison avec le graphe de commande est assurée par les places d'interface définies antérieurement (cf § III.5).

- Interface partie commande ⟶ procédé

  Aux places d'interface sont associées des messages. Le marquage de ces places génère l'envoi des messages correspondants.

- Interface procédé ⟶ partie commande

  L'activation de certaines méthodes provoque le forçage des places d'interface correspondant aux accusés réception de fin d'action, aux états des capteurs.

Reprenons l'exemple du transfert de pièces entre STOCK-A et STOCK-B sans tenir compte de l'aspect temporel. Pour réaliser l'interface "procédé ⟶ partie commande", on définit pour la classe robot de nouveaux champs afin de préciser les places d'interface à forcer pour accuser réception du dépôt d'une pièce, de la saisie d'une pièce et de l'arrivée à destination.

La classe robot est donc modifiée de la manière suivante :

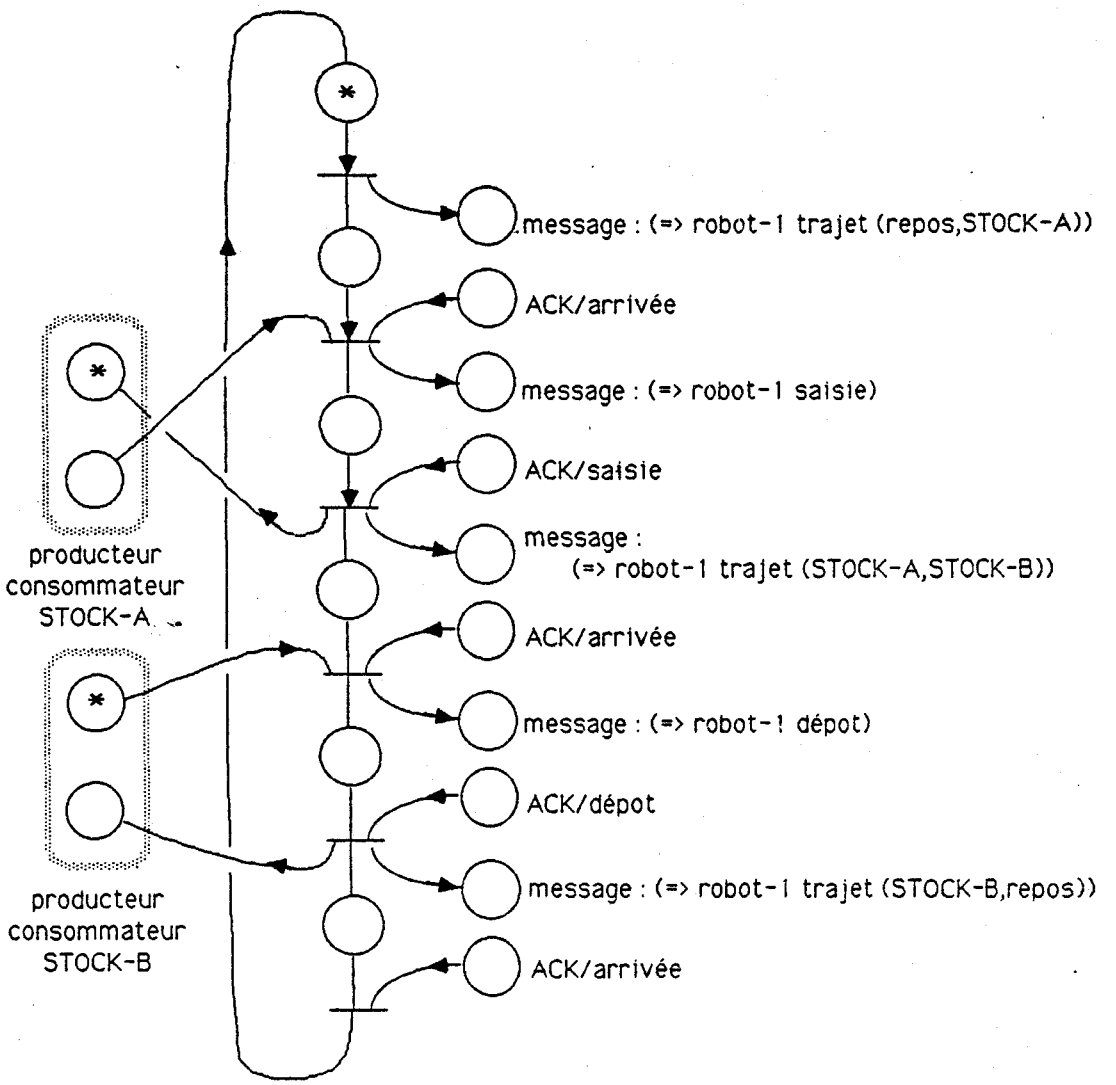
ROBOT		
	est un	: objet
	position	: <STOCK-A>/<STOCK-B>/<REPOS>
	occupation	: <()>/<identificateur de pièce>
	accusé-dépôt	: <identificateur de place>
	accusé-saisie	: <identificateur de place>
	accusé-arrivée	: <identificateur de place>
	trajet (origine, destination) : Affecte le champ "position" à la valeur "destination" Ajoute une marque dans la place spécifiée par le champ "accusé-arrivée"	
	saisie : Affecte au champ "occupation" la réponse au message (⟹ position retrait) Ajoute une marque dans la place spécifiée par le champ "accusé-saisie"	
	dépôt : Génère le message (⟹ position dépôt occupation) Affecte le champ "occupation" à () Ajoute une marque dans la place spécifiée par le champ "accusé-dépôt"	

Robot-1 est une instance de la classe ROBOT.

ROBOT-1		
	est un	: robot
	position	: repos
	occupation	: ()
	accusé-arrivée	: ACK/arrivée
	accusé-saisie	: ACK/saisie
	accusé-dépôt	: ACK/dépôt

La classe STOCK et ses 2 instances STOCK-A et STOCK-B ne sont pas modifiées puisqu'il s'agit d'organes passifs qui ne sont pas commandables et qui ne possèdent aucun capteur.

Le graphe de commande du processus de transfert de STOCK-A vers STOCK-B est le suivant :



III.7.3 - Conclusion

La modélisation du procédé par catégories génériques possède l'avantage d'être essentiellement modulaire. Elle permet, par utilisation d'une structure d'archivage d'objets types (robot, tour, centre d'usinage, ...), une description aisée et rapide du modèle particulier du procédé. Cette phase de conception consiste alors à choisir les classes d'objets utiles, d'en créer les instances nécessaires en précisant les valeurs des champs et en particulier les places d'interface utilisées pour le dialogue avec la partie commande, et d'associer les messages aux places générant les commandes.

CONCLUSION

Nous avons décrit, dans cette lère partie, les différents modèles permettant d'une part, la conception d'un système de production flexible et d'autre part, la simulation grâce au logiciel qui sera présenté dans le chapitre suivant.

Le modèle adopté pour représenter la partie commande est celui des RdP-SAC. La partie décisionnelle est modélisée par un ensemble de règles de production. Nous avons présenté également les grandes lignes d'une méthodologie permettant l'appréhension des grands systèmes, par une démarche progressive. Nous avons insisté, dans cette lère partie, sur l'avantage d'aborder, dès la phase de conception, l'architecture répartie du système réel, au niveau de la définition du graphe de commande et de la partie décisionnelle.

Notre objectif essentiel, pour la prise en compte du procédé, consiste à obtenir un compromis entre le fait de recueillir des résultats sur l'évolution dynamique du système et de préserver la simplicité du modèle initial. La temporisation permet d'intégrer le minimum d'informations concernant le procédé nécessaires à l'évaluation des performances du système.

La description par instantiation d'objets spécifiques permet d'obtenir simplement toutes les statistiques de fonctionnement voulues tout en restant proche du graphe de commande à implémenter. La plupart des logiciels de simulation de graphe de Petri qui fournissent des statistiques de fonctionnement (/GIR 84/ /ALA 84/) réalisent cet objectif en terme de graphe : nombre de fois où une place a été marquée, marquages moyens (dans le temps) et maximum, temps maximum, moyen et total pendant lequel une place est restée marquée, ... Les statistiques fournies par le modèle du procédé choisi, s'interprétent directement en terme de production : temps total d'usinage d'une machine, temps moyen de stockage en un lieu, temps moyen d'attente par pièce et par lieu, ...

CHAPITRE II

SIMULATION DES SYSTEMES DE PRODUCTION FLEXIBLE

INTRODUCTION

La simulation, utilisée depuis longtemps dans le domaine des processus continus, a fait plus récemment son entrée dans la Productique. Le rôle de la simulation est multiple.

Tout d'abord, les outils de simulation permettent d'envisager les configurations qui aboutiront à une productivité maximale grâce à un minimum de moyens de production. A un autre niveau, la simulation graphique permet de définir, dans une optique de CAO, la configuration géographique de l'atelier la mieux adaptée, ainsi que la forme, les dimensions et la localisation des robots à utiliser. Il existe également des systèmes de simulation à objectifs économiques permettant le calcul de rentabilité des différents postes de l'atelier.

Tous ces différents aspects de la simulation sont indispensables pour une conception réfléchie d'un atelier flexible. Quand on songe au volume de l'investissement financier mis en jeu, pour l'élaboration d'un atelier flexible, il est en effet souhaitable de minimiser les risques d'erreurs. Sans revenir sur l'importance de l'investissement en matière grise, nécessaire dans la phase de conception, qui a été développée dans l'introduction de ce mémoire, soulignons l'intérêt des résultats tant qualitatifs que quantitatifs fournis par la simulation pour confirmer et justifier, en préalable à toute implantation, la réalité du projet conçu.

Le logiciel de simulation que nous présentons ici, concerne plus particulièrement l'évaluation qualitative et quantitative de la dynamique du système. Afin de définir plus clairement les objectifs qui ont précédé sa réalisation, nous allons préciser ici, un bref "cahier des charges" du simulateur.

Le logiciel réalisé doit permettre l'édition, la simulation, la correction et l'archivage du modèle de production flexible.

Afin de permettre une définition progressive du système à concevoir, ces différentes parties ne doivent pas être indépendantes. L'utilisateur doit pouvoir simuler un modèle partiel et corriger son modèle après (ou en cours) de simulation. Le logiciel doit donc être essentiellement intéractif.

Le logiciel de simulation doit s'adapter à tout ou partie des différents modèles utilisés pour représenter le système étudié (RdP SAC pour la partie commande, règles de production pour la partie décisionnelle, temporisation ou description par catégories génériques pour le procédé). Si, à l'origine, le modèle de la partie commande était pratiquement défini par plusieurs années de recherche au sein du Laboratoire d'Informatique Industrielle de l'I.D.N. (les premiers travaux de J.C. Gentina et D. Corbeel sur les RdP structurés datent de 1979), la nécessité d'intégrer le procédé et la partie décisionnelle n'apparaissait pas clairement dans la phase de conception et d'évaluation de la partie commande d'un système flexible. L'utilisation d'un modèle hybride au niveau du simulateur a été réalisée parallèlement à une phase de réflexion commune sur les outils et les modèles requis pour appréhender ces divers aspects.

L'évaluation qualitative du système consiste à pouvoir détecter, analyser et résoudre les blocages et indéterminismes induits par une mauvaise conception ou par une définition insuffisante du graphe de commande.

L'évaluation quantitative consiste à pouvoir intégrer l'aspect temporel aux divers modèles utilisés afin d'analyser les performances du système et de les optimiser en dimensionnant éventuellement certains paramètres du système (taille des zones de stockage, nombre de ressources, etc ...) ou en modifiant la stratégie adoptée.

I - DE LA VALIDATION A LA SIMULATION

I.1 - La validation théorique

I.1.1 - Introduction

L'objectif de la validation est de garantir le caractère de bon fonctionnement du système modélisé. Cette analyse consiste à vérifier un ensemble de propriétés du RdP, et en particulier, les 3 propriétés fondamentales suivantes : la vivacité, le caractère borné et le caractère réinitialisable.

* La **vivacité** qui caractérise l'absence de blocages et la reproductibilité des opérations associées au réseau. En terme de graphe, la vivacité consiste à vérifier que chaque transition est franchissable depuis tout état du système.

* Le caractère **borné** qui exprime la finitude des états du réseau. Vérifier ce caractère borné revient à prouver, pour chaque place du graphe, l'existence d'une borne supérieure au nombre de marques pouvant s'accumuler dans cette place. Un réseau est qualifié de **sauf** si, pour chaque place, cette limite vaut 1.

* Le caractère **réinitialisable** qui caractèrise le fonctionnement cyclique du système. Il suppose l'existence d'un marquage d'accueil qui peut être reconstitué quelle que soit l'évolution du système.

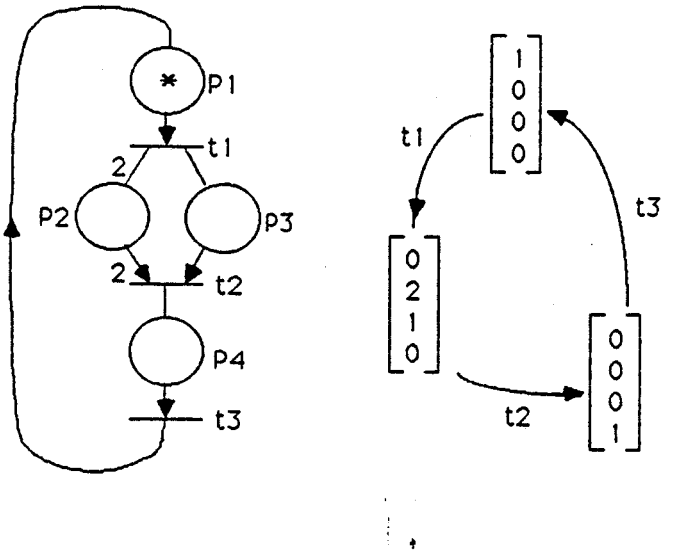
Il existe 2 types de méthodes permettant l'analyse de ces propriétés : l'étude du graphe de couverture et les méthodes d'algèbre linéaire basées sur la représentation matricielle du réseau.

I.1.2 - Les outils de validation

a) Etude du graphe de couverture :

L'étude du comportement du réseau marqué est réalisée par construction du graphe des marquages. On vérifie par exemple le caractère borné, si ce graphe de marquage est fini.

Exemple :



Le graphe ci-dessus est borné, vivant et réinitialisable.

Notons que, dans le domaine d'applications qui nous intéresse (la productique), la dimension importante des modèles est un premier obstacle à l'analyse du graphe des marquages, compte tenu de l'explosion combinatoire générée par les algorithmes mis en œuvre lors de l'informatisation de cette méthode.

b) Les méthodes d'algèbre linéaire :

Ces méthodes, basées sur la représentation matricielle du réseau (cf § I.2.2 du Chapitre I), permettent de vérifier, en plus des résultats dépendant du marquage, un ensemble de propriétés dites "structurelles", c'est-à-dire indépendantes du marquage /BER 79/. L'étude des ces propriétés structurelles consiste à rechercher les invariants de places et de transitions traduisant les relations invariantes sur le marquage ou sur les tirs de transitions lors d'une évolution quelconque du graphe. Ces invariants s'obtiennent par résolution de systèmes linéaires en nombres entiers /MEM 80/.

La matrice d'incidence est dimensionnées par la taille du réseau. Si l'on considère un système industriel modélisé par 100 places et 100 transitions, la matrices d'incidence est alors de dimension 100×100. De plus, le nombre de places en amont et en aval d'une transition est en général faible devant le nombre total de places. La matrice d'incidence est donc généralement "creuse". La représentation

matricielle d'un réseau propose une description simple du graphe, mais coûteuse à informatiser.

D'autre part, par la prise en compte de certaines extensions des RdP et en particulier l'aspect adaptatif, l'algèbre obtenue n'est plus linéaire. Dans ce cas, la recherche des invariants devient indécidable.

c) Réduction :

L'obstacle essentiel à la mise en œuvre des méthodes basées sur l'énumération des marquages et des méthodes algébriques, concerne la taille des réseaux. Pour y remédier, il est intéressant, avant d'engager la validation, de réduire la taille du réseau en utilisant des méthodes de réduction qui conservent les propriétés initiales du graphe /BER 83/et la structure générale du réseau /VER 82/.

I.1.3 - La validation des RdP structurés

L'approche structurée adoptée lors de la conception du graphe de commande facilite la validation du réseau. Les processus utilisés sont en effet, saufs, vivants et réinitialisables par construction. La définition des structures de liaisons permet de garantir le caractère 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 liaison entre divers processus (Chapitre I, § I.3.2.c) et nécessite la mise en œuvre des méthodes de validation.

I.1.4 - Les limites des outils de validation dans le cadre de l'analyse des systèmes flexibles de production

L'étude des propriétés du graphe de commande d'un système de production flexible se heurte, nous l'avons vu, à 2 obstacles : la taille du modèle et l'indécidabilité des propriétés structurelles due au caractère adaptatif du modèle choisi.

Les outils théoriques de validation fournissent des résultats intéressants sur le "bon fonctionnement" du graphe de commande mais ne constituent qu'une première étape, souvent "nécessaire", dans la validation des systèmes de production flexible. En particulier, de

tels outils sont insuffisants lorsqu'il s'agit de valider des comportements, notamment de régime transitoire inhérent à la flexibilité du système, d'évaluer les performances en terme de production, ou d'évaluer différentes stratégies de pilotage. Dans ce cas, la simulation est un outil indispensable de validation dont l'intérêt primordial est de proposer un outil homogène à la fois pour l'analyse, la synthèse et la validation du système étudié.

I.2 - La simulation : Outil de validation dans le contexte des systèmes de production flexible

I.2.1 - Introduction

La simulation et les méthodes de validation jouent donc des rôles complémentaires dans la vérification du caractère de bon fonctionnement du système. En effet, si la validation concerne les RdP **autonomes**, la simulation permet d'intégrer l'interprétation liée au graphe, c'est-à-dire d'une part, les caractéristiques du procédé et d'autre part, les consignes de la partie décisionnelle. Elle intervient après la validation et permet :

1) de prévoir le comportement réel du système et d'évaluer en particulier ses performances,
2) d'évaluer le mode transitoire entre 2 régimes établis lors du passage à un mode de marche dégradé, par exemple.

I.2.2 - Les outils de simulation

Nous distinguons 2 types d'outils de simulation selon le degré de représentation du procédé.

Les premiers utilisent des logiciels dédiés à la simulation des processus physiques. C'est le cas du langage de simulation SLAM. Ces langages ne satisfaisant pas au principe de séparation entre les parties commande et opérative, fournissent des évaluations précises sur le comportement du système, mais les résultats obtenus ne permettent pas l'élaboration directe des processus de commande à implanter.

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

Citons 2 exemples de telles réalisations : PSI (Petri-net based SImulator) mis au point par le L.A.A.S. en collaboration avec le G.I.E. Renault Recherche Innovation /ALA 84/ et le logiciel de C. Girod /GIR 84/. Ces 2 logiciels permettent de produire des calculs statistiques associés aux places et aux transitions du réseau.

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 très évoluées, autorisent une émulation plutôt qu'une simulation du système de commande. C'est le cas, en particulier, des automates de la Société Télémécanique, associés aux consoles de programmation T 607 /GUI 85/.

I.2.3 - Conclusion

Dans la plupart des logiciels de simulation existants, le niveau décisionnel n'est, en général, que peu abordé. Ces logiciels réalisent au mieux, la séparation entre la partie commande et le procédé, ce qui est une contrainte indispensable pour permettre l'adéquation entre la commande réalisée et son implémentation. De même, l'étude d'exemples d'ateliers flexibles montre l'intérêt de séparer, au niveau du modèle, les parties commande et décisionnelle afin de permettre, dans la phase de conception, une meilleure maîtrise des stratégies à définir pour paramétrer la commande.

II - PRESENTATION GENERALE DU SIMULATEUR

II.1 - Introduction

Le simulateur que nous avons réalisé permet d'élaborer la modélisation d'un système de production en distinguant les parties commande, décisionnelle et le procédé.

Le modèle de base retenu pour la partie commande est constitué des Réseaux de Petri Structurés Adaptatifs Colorés (RdP SAC) décrits dans la lère partie. Le choix de ce modèle repose sur l'utilisation d'une **méthodologie de conception** du graphe qui sera assistée, voire complétement automatisée par la suite. L'intérêt de cette démarche est de permettre la maîtrise du parallélisme et donc de faciliter l'appréhension des grands systèmes. Cependant, si cette démarche est conseillée, nous avons laissé à l'utilisateur toute liberté sur la modélisation de sa commande. Ainsi, les limites imposées par la structuration peuvent ne pas être respectées ; notre logiciel est prévu pour simuler un graphe de Petri quelconque. De même, l'abréviation permise par la coloration, peut être étendue facilement jusqu'à atteindre le pouvoir de modélisation des RdP à prédicats/transitions au sens de Genrich /GEN 79/.

La partie décisionnelle est représentée par un ensemble de règles de production dont le but est, rappelons le, 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 à plusieurs niveaux. Tout d'abord, l'élaboration du graphe de commande à partir du graphe de transfert et de conditionnement (cf lère Partie), 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 2 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 de

statistiques de fonctionnement, étude de la propagation des défauts, de leur détection, du traitement correspondant, ...

Notons également la possibilité d'associer une interprétation aux transitions du réseau. A chaque transition, l'utilisateur peut définir des évènements attendus ou générés par le tir de cette transition.

II.2 - Intérêt d'une approche déclarative

La plupart des systèmes de validation et de simulation cités précédemment sont écrits en langages impératifs compilés (OVIDE en Fortran, le simulateur de C. Girod en Pascal sur VAX 750, SLAM en Fortran, ...) et bénéficient donc de la rapidité d'analyse inhérente à l'utilisation de ces langages.

Cependant, la convivialité de ces systèmes est relativement restreinte. La simulation d'un système SLAM, par exemple, débute par la compilation combinée des primitives du modèle et du langage SLAM. L'utilisateur n'accède donc à son modèle qu'en phase d'édition.

Lors de la conception d'un système de production flexible, il est indispensable d'adopter une démarche structurée dans le sens d'une analyse par niveaux croissants de complexité. Il s'agira, par exemple, de concevoir d'abord le mode de marche automatique, puis de définir les stratégies à adopter au niveau décisionnel, d'intégrer les différents modes de marche dégradés prévus, d'optimiser les performances, ... Cette démarche, généralement effectuée par une méthode d'essai/erreur, nécessite une convivialité telle qu'il soit possible de compléter ou de corriger aisément le modèle conçu. L'avantage des langages fonctionnels est de proposer une intéractivité plus importante. C'est pourquoi nous avons choisi Le_Lisp comme langage de réalisation du simulateur. En effet, Le_Lisp, même compilé, permet à tout instant d'accéder à l'intégralité de l'univers de travail, c'est-à-dire à la fois les données relatives au modèle et la façon de les exploiter (le "programme"). Le temps d'exécution est certes plus important que pour un langage impératif compilé ; cependant, l'intéractivité obtenue compense cette relative perte de rapidité.

II.3 - Implantation du simulateur

Le simulateur est réalisé en Le_Lisp, en interprété sur micro-calculateur compatible IBM-PC (BFM 186) et en compilé sur VAX 750. Le choix du langage Le_Lisp est le résultat de la démarche prospective suivante :

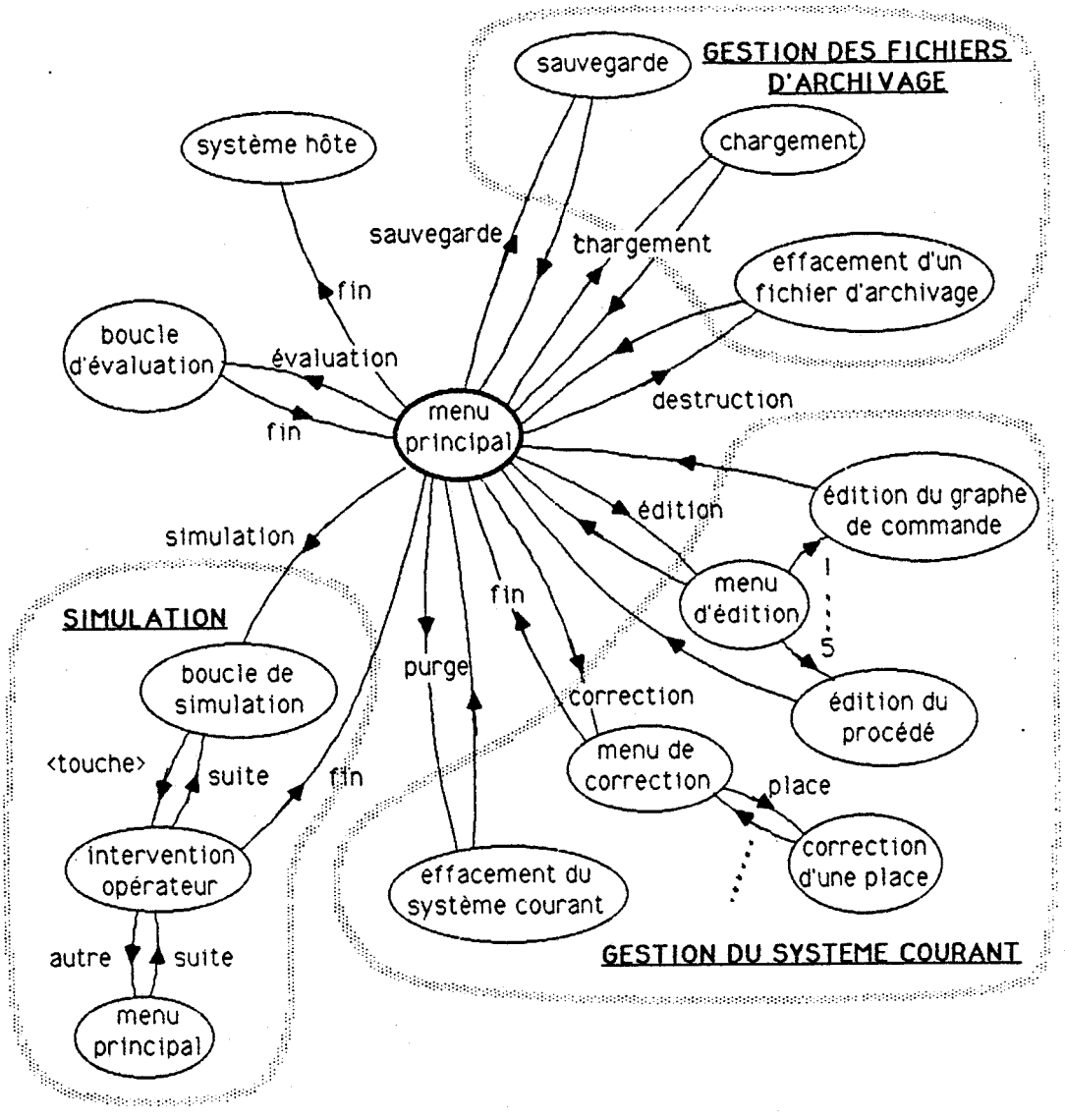
Notre première approche fut de réaliser un logiciel de validation de RdP, permettant l'analyse des propriétés fondamentales par construction du graphe des marquages. Cet interpréteur fut réalisé en Prolog II sur VAX 750 afin de bénéficier des avantages du "retour-arrière" (ou backtracking) lors de l'élaboration du graphe de couverture. Ce logiciel a donné entièrement satisfaction pour des réseaux de petite taille. Cependant, l'exploration arborescente du graphe des marquages, provoquait, dans le cas de réseaux plus importants, une explosion combinatoire responsable d'un temps d'exécution généralement assez long, voire d'un débordement des piles du système /CAS 85/. Ce logiciel était donc incapable de traiter des réseaux de taille industrielle.

L'étape suivante fut la réalisation d'un simulateur de RdP adaptatifs et interprétés en LOGO sur Apple II. Les résultats obtenus traduisaient une vitesse d'exécution et une intéractivité satisfaisantes. Cependant, la taille du réseau simulé était limitée par l'espace disponible (à peine 13 k Octets pour les programmes , les données du réseau à simuler et l'espace utile de travail). Afin de résoudre ce problème d'espace, nous avons retranscrit ce simulateur en VAX-LISP compilé sur VAX 750. Cette version, bien que permettant d'appréhender des réseaux de taille industrielle (100 places, 100 transitions) mit en évidence certaines faiblesses du langage VAX-LISP. Notre choix final pour l'intégration de la coloration et l'ouverture vers la partie décisionnelle et le procédé, s'est porté sur Le_Lisp, également compilable sur VAX et beaucoup plus souple d'utilisation que VAX-LISP.

Notons cependant que les 3 versions de notre simulateur existant actuellement en Le_Lisp (en interprété sur BFM, en interprété sur VAX et en compilé sur VAX) sont légèrement différentes les unes des autres. En effet, il est apparu à l'usage que les fonctions de contrôle gérant la pile du système Le_Lisp doivent être utilisées différemment dans les 3 cas.

II.4 - Organisation du logiciel

Le logiciel de simulation est scindé en 29 modules permettant l'édition, la correction, l'archivage et la simulation du système de production conçu. Son utilisation est régie par un ensemble de menus dont l'organisation de base est la suivante :



La caractéristique principale de notre simulateur est l'existence de l'interruption programmée "Intervention opérateur" permettant à tout instant de suspendre la simulation, de revenir au menu principal tout en assurant la reprise ultérieure de la simulation en cours. Cette interruption, sur laquelle nous reviendrons, permet un gain considérable dans l'intéractivité du système et dépend essentiellement de l'utilisation du langage déclaratif Le_Lisp.

III - STRUCTURE ET EDITION D'UN SYSTEME DE PRODUCTION

III.1 - Introduction

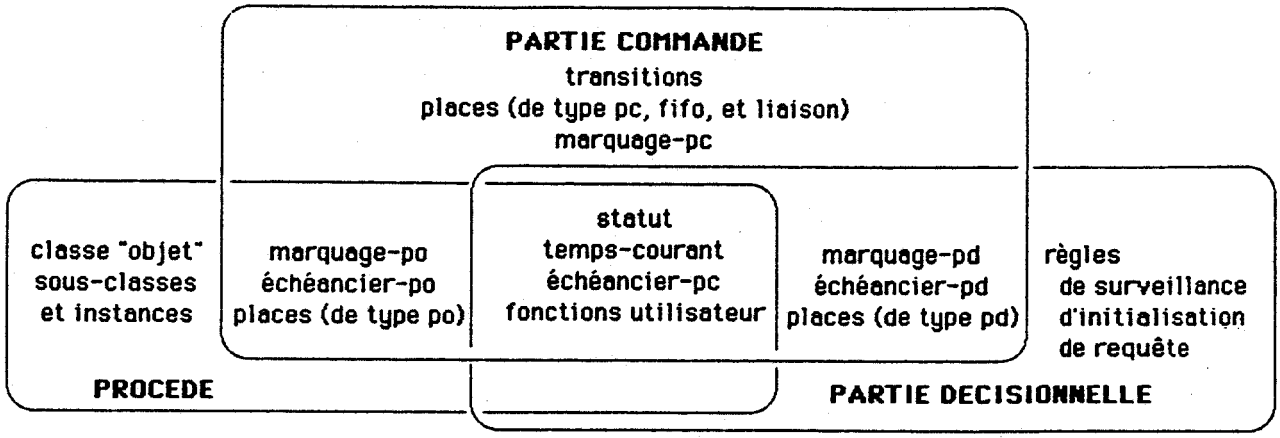
Rappelons que notre travail s'inscrit dans le cadre d'un projet plus général, développé au Laboratoire d'Informatique Industrielle de l'I.D.N., sur la conception des systèmes de production flexibles. Dans cette optique, l'élaboration du modèle résulte de la définition précise du cahier des charges du système à concevoir et des moyens choisis a priori pour atteindre l'objectif fixé. Le modèle est donc un outil intermédiaire permettant la conception, l'analyse puis l'implémentation du système. A ce titre, la phase d'édition du modèle doit être suffisamment transparente pour éviter au concepteur de se perdre dans le formalisme d'un RdP démesuré (les RdP modélisant les ateliers flexibles avoisinent souvent plus de 100 places et 100 transitions).

Le projet développé au laboratoire prévoit donc une approche CAO pour l'élaboration du modèle, basée sur l'utilisation d'une base de données de RdP.

Les modules permettant l'édition de la partie commande du modèle n'ont donc un intérêt que purement ponctuel, cette démarche devant être automatisée par la suite. Ces modules ont été conçus afin de présenter actuellement, notre simulateur comme un outil autonome.

III.2 - Structure du système

La structure du système est saisie sous forme de quelques variables globales regroupant les informations relatives au système considéré et principalement sous forme de p-listes (listes de propriétés) associées aux éléments constitutifs. Cette structure est scindée en 3 parties : le graphe de commande (de type réseau de Petri), la partie décisionnelle (de type règles de production) et le procédé (décrit par catégories génériques). Le croquis ci-dessous rassemble toutes les données relatives à un système.



* Le **statut** d'un système traduit le stade de spécification atteint dans la définition du système, à savoir :

  - son nom,
  - définition ou non du graphe de commande,
  - définition ou non de la partie décisionnelle,
  - représentation ou non du procédé par catégories génériques,
  - définition ou non de temporisation,
  - définition ou non de time-out,
  - définition ou non de macros-places de type FIFO.

  Le statut du système est modifié lors de la spécification croissante du système et utilisé lors de la configuration du simulateur (ce point sera développé ultérieurement).

* La variable "marquage-pc" (respectivement "marquage-pd", marquage-po") mémorise la liste des places de type "pc" (respectivement "pd", "po") possédant un marquage actif.

a) Le **réseau de Petri** est saisi sous forme de p-listes associées aux places et aux transitions. Pour chaque transition, sont précisées la liste éventuelle des variables libres et leur domaine de variation s'il s'agit de variables libres locales à la transition, la liste des arcs amonts et celle des arcs avals et éventuellement les évènements attendus et générés par cette transition. La p-liste associée à une place contient les caractéristiques statiques de cette place : son type (pc, po, pd, fifo, ou rien s'il s'agit d'une place de liaison), la liste des transitions en aval, et ses caractérisques dynamiques : la liste des marques actives et des marques "gelées" par temporisation ou par décision du niveau hiérarchique,

les indications nécessaires pour gérer d'éventuels time-out associés au marquage.

b) Pour chaque **règle de production** de la partie décisionnelle, la p-liste associée contient la liste éventuelle des variables libres et leurs domaines, le type de la règle (initialisation, surveillance ou requête) et l'énoncé qui est une expression évaluable en Lisp et restituant intentionnellement un résultat booléen.

c) Pour chaque **objet** du procédé, la p-liste associée regroupe les champs et les méthodes relatifs à cet objet.

Les échéanciers ("échéancier-pc", "échéancier-pd" et "échéancier-po") sont constitués de la liste ordonnée de couples (date, évènement) dont l'occurence conditionne l'évolution du système. L'échéancier de la partie commande gère les évènements extérieurs prévus par l'opérateur ou générés par la partie décisionnelle et par le procédé, et les évènements internes générés par le simulateur (fin de temporisation, time-out dépassé, ...).

III.3 - Edition et correction du système

Réalisée en dehors d'une approche de type CAO, l'édition d'un système est une tâche ingrate principalement à cause de la taille importante du graphe de Petri modélisant la commande d'un îlot de production flexible. Afin de rendre cette édition moins astreignante, nous avons rendu cette phase progressive en la découpant en 5 modules d'édition proprement dite et en 9 modules de correction.

Les 5 premiers modules permettent l'édition sélective du graphe de commande, des règles de la partie décisionnelle, des temporisations, des time-out et du procédé. Les modules de correction permettent l'ajout ou la modification d'une structure particulière : une place, une transition, une règle, une structure de mutuelle exclusion, etc ...

Les pages suivantes décrivent de manière plus précise ces modules d'édition et de correction et leur utilisation. A ce sujet, le lecteur se reportera à l'Annexe A où nous avons développé, sur un exemple simple mais relativement complet, leur mise en œuvre effective.

III.4 - La saisie du graphe de Petri

La saisie s'opère transition par transition. Pour chaque transition, on précise les variables libres concernées, et éventuellement leur domaine de variation. Ce domaine peut être explicité exhaustivement ou être le résultat de l'évaluation d'une expression Lisp. Lisp ne présentant aucune différence syntaxique entre la donnée d'une liste ou celle d'une expression, on précise, au moment de la définition du domaine, s'il s'agit d'une liste ou d'une expression.

Le logiciel opère ensuite la saisie des arcs en amont puis en aval de la transition considérée. Pour chaque arc, on précise :

- Le poids de l'arc : un entier donné ou résultant de l'évaluation d'une expression.

  Exemple : poids = 1

  poids = (m 'Cl 'PDl) pour M(PDl)(Cl)
  où Cl est une couleur constante
  (Cl ∈ K)

  poids = (m x 'PDl) pour M(PDl)(x)
  où x est une couleur variable
  (x ∈ X)

- La couleur transitant par cet arc qui peut être une couleur déterminée, une couleur variable ou le résultat de l'évaluation d'une expression.

- La place reliée à la transition par l'arc considéré. Si cette place est le résultat de l'évaluation d'une expression et si l'arc considéré est un arc amont, le logiciel demande la liste des places possibles résultant de cette évaluation afin de pouvoir mettre à jour la liste des transitions successeurs de ces places. Notons que cette mise à jour, automatique, s'opère parallèlement à la saisie des transitions.

Quand la saisie des transitions du réseau est achevée, le logiciel interroge l'utilisateur sur le type des places répertoriées lors de cette première phase. Une place est déclarée de type "pc" si elle appartient à un processus, de type "po" s'il s'agit d'une place d'inter-

face vers le procédé, de type "pd" pour une place de l'interface avec la partie décisionnelle. Le type n'est pas précisé pour les places de liaisons (mutuelle exclusion, producteur/consommateur et synchronisation) sauf s'il s'agit d'une macro-place modélisant une fifo, auquel cas elle est déclarée de type "fifo".

Après édition de la structure, suit la phase de définition du marquage initial actif puis gelé. On précise les places initialement marquées, la couleur et le nombre de ces marques. Aux marques non colorées est attribuée par défaut la couleur "*".

III.5 - Saisie de la partie décisionnelle

Cette saisie s'opère en deux temps : la saisie des règles de surveillance et la saisie des règles de requête.

Pour la saisie des règles de surveillance, on précise pour chaque règle, s'il s'agit d'une règle d'initialisation ou non.

Pour la saisie des règles de requête, on définit pour chaque règle, la place (de type "pd") opérant la requête. A ce sujet, le simulateur impose une restriction par rapport au modèle présenté dans la première partie (cf Chapitre I, § II.5). A une place d'interface de type "pd", on ne peut associer qu'une seule règle de requête. L'utilisation de la conditionnelle généralisée (le "COND" de Le_Lisp) permet aisément d'agréger les diverses règles associées à une place en une seule.

(COND ((condition 1) (action 11) ... (action $1n_1$))
((condition 2) (action 21) ... (action $2n_2$))
⋮
((condition p) (action p_1) ... (action pn_p))

Ainsi, pour l'exemple (présenté au Chapitre I, § II.5) sur le paramétrage d'une priorité tournante, on associe à la place "requête-R" l'unique règle de requête suivante :

```
R (COND ((M(requête-R)(P1) > 0 et M(requête-R)(P2) > 0 et
          M(FLIP/FLOP)(FLIP) > 0)
         (enlever (1,P1,requête-R)) (ajouter (1,P1,allocation-R))
         (enlever (1,FLIP,FLIP/FLOP)) (ajouter (1,FLOP,FLIP/FLOP)))
        ((M(requête-R)(P2) > 0 et M(requête-R)(P2) > 0 et
          M(FLIP/FLOP)(FLOP) > 0)
         (enlever (1,P2,requête-R)) (ajouter (1,P2,allocation-R))
         (enlever (1,FLOP,FLIP/FLOP)) (ajouter (1,FLIP,FLIP/FLOP)))
        ((M(requête-R)(P1) > 0 et M(requête-R)(P2) = 0)
         (enlever (1,P1,requête-R)) (ajouter (1,P1,allocation-R)))
        ((M(requête-R)(P2) > 0 et (M(requête-R)(P1) = 0)
         (enlever (1,P2,requête-R)) (ajouter (1,P2,allocation-R)))
```

Pour les règles de requête comme pour les règles de surveillance, l'édition s'opère de la manière suivante : définition des variables libres associées, de leur domaine puis de l'énoncé de la règle.

III.6 - Saisie du procédé

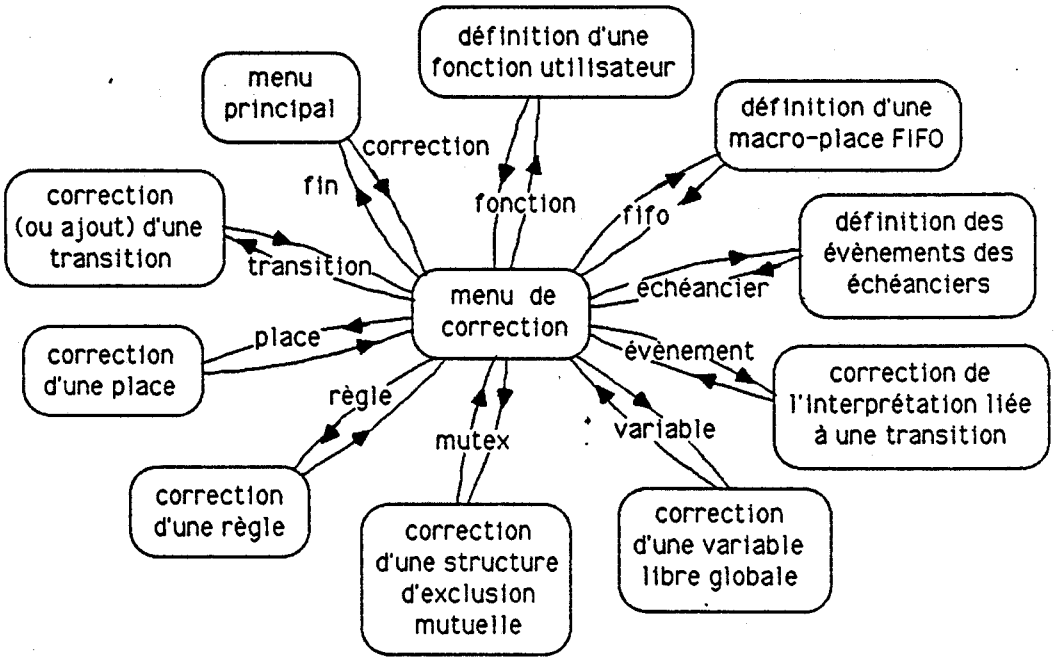
Cette phase de l'édition n'est actuellement pas supportée directement par notre logiciel. Une telle démarche nécessite à notre sens, l'existence d'une base de données de modèles spécifiques aux machines et organes de transports utilisés et au degré de description requis (élaboration de statistiques, étude des dysfonctionnements, ...) (cf Chapitre I, § III.7.3). Les modèles actuellement existants, créés au cas par cas pour les besoins d'une simulation donnée, ne constituent pas un panorama suffisamment exhaustif pour justifier cette édition systématique. Le logiciel se contente de créer la classe générique "objet" permettant de construire les sous-classes et instantiations nécessaires.

La définition de l'interface avec les places de type "po" du graphe de commande est réalisée par le module de correction des places. Si la place est de type "po", le logiciel s'enquiert de la variable libre éventuellement nécessaire, de son domaine d'évolution et du message émis vers un objet du procédé. La syntaxe de ce message est la suivante (cf Chapitre I, § III.7) :

(⟹ <instance d'objet> <méthode> [<argument 1> ... <argument n>]

III.7 - Les modules de correction

Le menu de correction est organisé de la façon suivante :



Nous ne nous étendrons pas sur la description de chacun des modules de correction, mais seulement sur l'utilité de la commande "fonction" permettant de définir des fonctions-utilisateurs. L'intérêt de ces fonctions dépend du degré d'utilisation d'expressions Lisp au niveau de la description du modèle. Certaines fonctions prédéfinies sont intégrées au simulateur. C'est le cas des fonctions M(P)(C) fournissant le nombre de marques de couleur "C" au sein de la place "P", dom(P) restituant la liste des couleurs des marques présentes dans la place P.

Prenons par exemple le problème de l'entrée de nouvelles pièces dans le système. (Ce point sera développé au § VII.2 de ce chapitre). La détermination du type de pièce entrant dans le système peut être réalisée aléatoirement ou issue d'une séquence préfixée ou laissée à l'initiative de l'opérateur, ... Il est donc utile de créer une fonction "entrée" gérant la stratégie choisie pour l'alimentation simulée d'un système de production selon des lois paramétrables.

III.8 - Conclusion

La structure du système est saisie principalement sous forme de p-listes associées aux identificateurs des éléments constitutifs (voir Annexe A, la structure générée pour l'exemple développé). L'avantage d'un tel choix est double.

A priori, la structure d'enregistrement (ou "record") semble plus économique en espace mémoire qu'une structure de liste de propriétés. Cependant, l'analyse des systèmes étudiés montre que les places sont statistiquement rarement marquées, les transitions rarement interprétées, ... Dans ces conditions, la structure plus adaptative des p-listes est au moins aussi compacte que des enregistrements vides pour la plupart.

D'autre part, l'accès aux données, enregistrées sur ces p-listes est d'autant plus facile pour l'utilisateur que ces données ne sont pas codées dans un ordre déterminé plus ou moins transparent puisqu'imposé par le logiciel de simulation.

Achevons cet exposé relatif à l'édition d'un système par une remarque sur le degré de compétence en Lisp requis pour utiliser notre simulateur. L'édition d'un RdP coloré ne demande aucune connaissance spéciale de Lisp. les données sont alors directement prises en charge par le logiciel pour générer la structure de données exploitable par le simulateur. Par contre, la définition de l'énoncé des règles de la partie décisionnelle ou des fonctions-utilisateurs, nécessite, de la part de l'utilisateur, une certaine connaissance de Le_Lisp. Le niveau de compétence en Lisp nécessaire est évidemment lié à la complexité des expressions Lisp utilisées. Plus le modèle s'écarte du cadre simple des RdP colorés, plus il faut considérer notre logiciel comme un sur-langage de Le_Lisp dédié à la simulation.

IV - SIMULATION D'UN SYSTEME DE PRODUCTION

IV.1 - Modularité du logiciel de simulation

Le logiciel de simulation proprement dit est scindé en 12 modules. Chaque module gère un aspect particulier de la simulation : l'évolution de la partie commande, de la partie décisionnelle, du procédé, la temporisation, les time-out, la gestion des macro-places de type fifo. L'intérêt d'un tel découpage est double.

Le choix du modèle dépend de la complexité du système étudié et du degré de spécification atteint dans la phase de conception. En suivant une démarche structurée, dans le sens d'une analyse par niveau croissant de complexité, le concepteur s'attache à définir le graphe de commande avant d'intégrer la partie décisionnelle puis d'évaluer les performances du système conçu. Il est donc intéressant, par souci d'efficacité, d'adapter le logiciel de simulation au degré de définition du modèle simulé. La structure modulaire adoptée permet une telle approche. Si le système modélisé ne comporte pas de time-out, par exemple, il est inutile d'intégrer la gestion des time-out au niveau du simulateur.

Le deuxième avantage est de permettre, a posteriori, une simulation partielle du système considéré. Il peut être intéressant par exemple, de simuler un graphe de commande où l'indéterminisme est résolu par un ensemble de règles du niveau décisionnel, sans tenir compte de la partie décisionnelle afin de mettre en évidence cet indéterminisme puis d'émuler l'évolution du niveau décisionnel pour étudier la façon dont cet indéterminisme a été résolu a priori.

C'est pourquoi l'utilisateur, avant toute simulation, définit le cadre de simulation qu'il souhaite. Nous appelons cette phase, la configuration du simulateur.

IV.2 - La configuration du simulateur

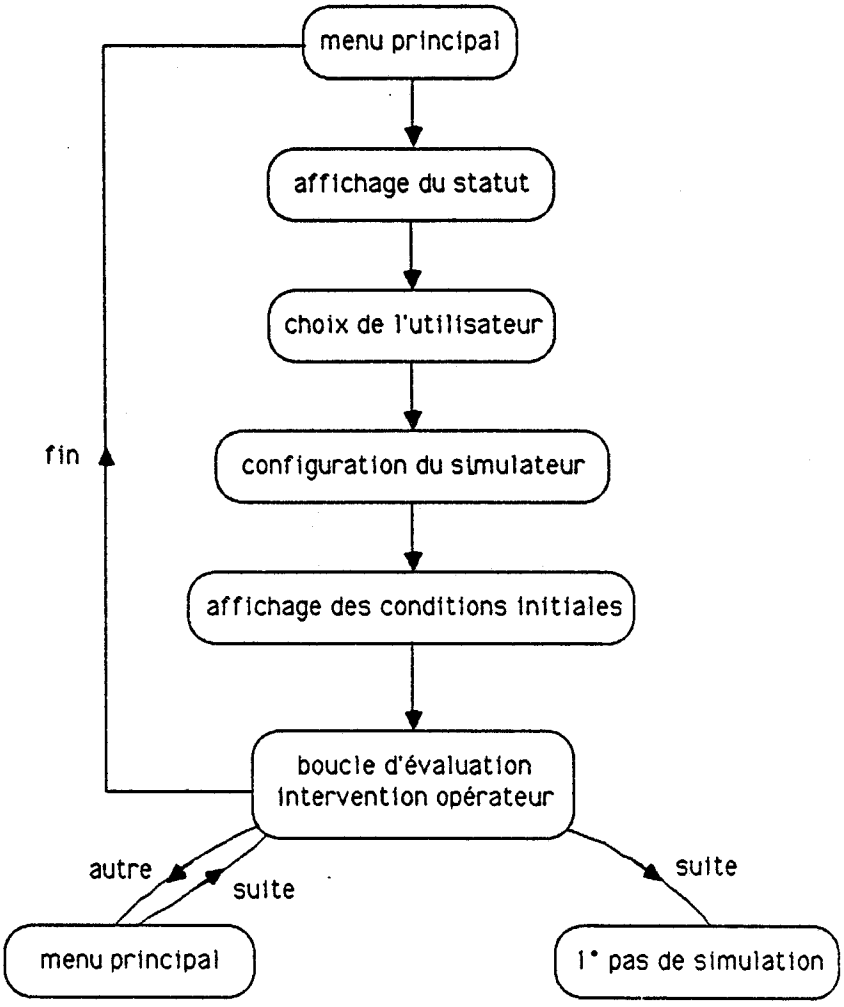
Rappelons que le **statut** du système regroupe les informations fondamentales relatives au système considéré, à savoir si le graphe de commande est défini, si le niveau décisionnel est défini, ... En fonction de ce statut et des choix dictés par l'opérateur, les divers modules de simulation sont assemblés, afin de définir le cadre de simulation.

De manière générale, pour toute caractéristique inexistante du statut, le module correspondant n'est pas intégré. Par contre, pour toute caractéristique définie dans le statut, le module correspondant est pris en compte mais inhibé ou non selon la décision de l'opérateur.

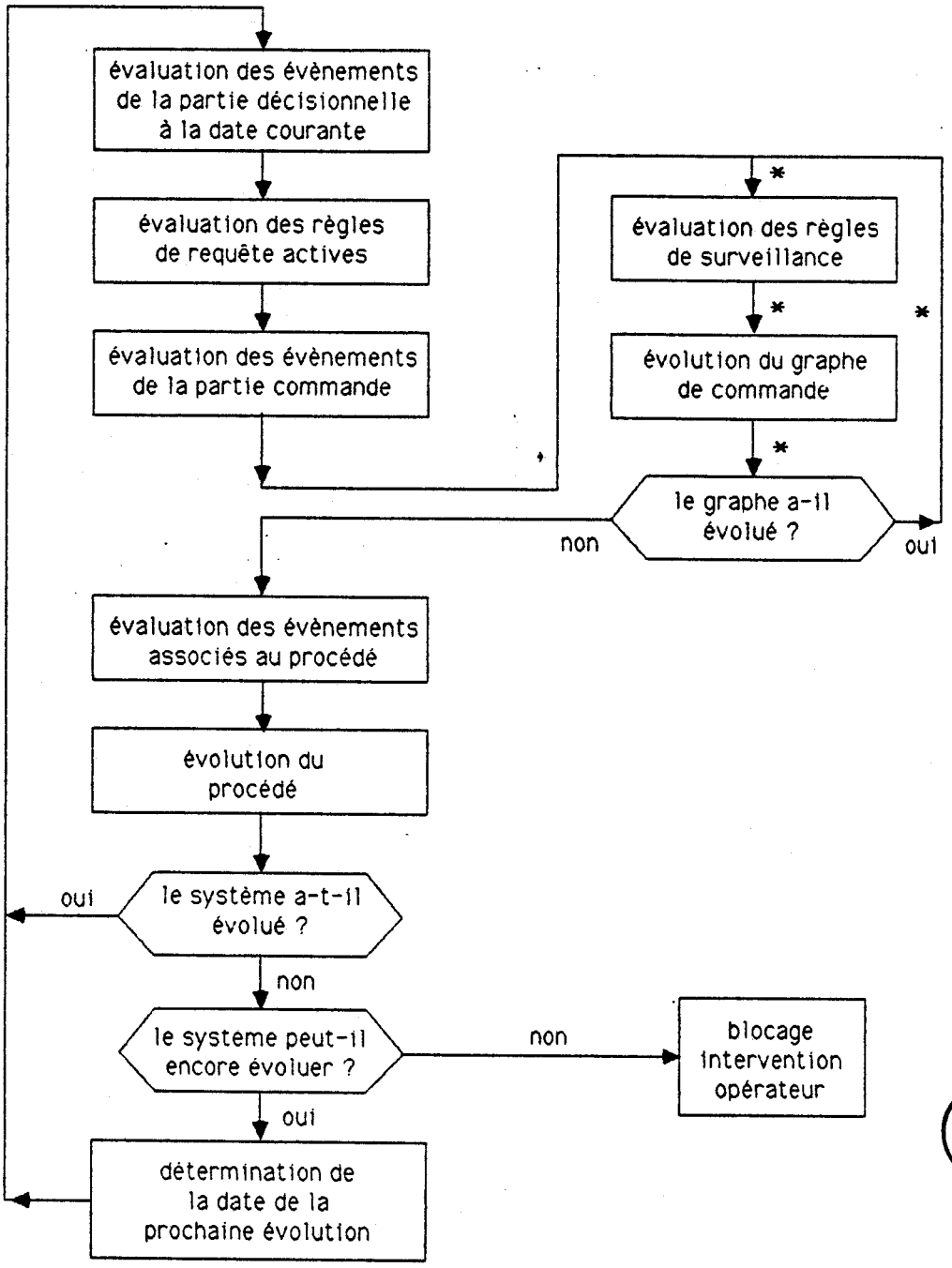
Les choix proposés à l'utilisateur sont les suivants :

- Faut-il simuler ou non le graphe de commande ?
- L'indéterminisme d'agrégation est-il permis ou non ? (ce point sera développé ultérieurement)
- Faut-il simuler ou non la partie décisionnelle ?
- Faut-il simuler ou non le procédé ?
- Faut-il temporiser ou non la simulation ?
- Faut-il prendre en compte ou non les time-out ?

Cette phase de début de simulation est organisée selon le schéma suivant :



IV.3 - Cycle de simulation du système complet



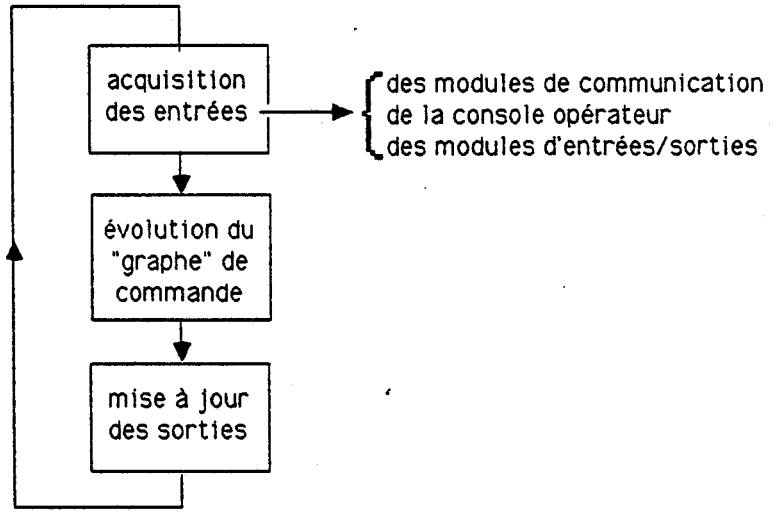
Cet organigramme traduit le cycle de simulation (ou pas élémentaire) dans le cas le plus général. Il dépend évidemment de la configuration du simulateur.

Ce cycle de simulation combine 2 approches courantes en simulation : **"par activité"** et **"par évènement"**.

Le cycle interne (étiqueté par une étoile) correspond au cycle par activité. L'étude de l'état du graphe de commande permet de déterminer l'évolution du graphe puis, si l'état résultant le permet, on réitère cette recherche. Ces cycles de simulation par activité s'opèrent donc à temps de simulation constant.

Le cycle externe, par contre, est un cycle par évènement, c'est-à-dire que l'évolution du système est régi par l'évaluation ou la présence d'évènements présents dans les divers échéanciers. Le mécanisme d'avance dans le temps, en particulier, dépend de cette approche par évènement.

La définition d'un tel cycle de simulation au niveau du simulateur ne recrée évidemment pas l'**évolution réelle** du système puisqu'en réalité, la partie commande, la partie décisionnelle et le procédé évoluent en parallèle. Par contre, ce cycle réalise une bonne approximation de l'**évolution du système vue de la partie commande.** En effet, si la partie décisionnelle et le procédé sont fondamentalement asynchrones du graphe de commande, la prise en compte de leurs évolutions s'opère à des instants privilégiés du cycle d'évolution de la commande. C'est ce que nous avons voulu traduire dans notre simulateur en nous inspirant du "cycle automate" suivant :



La structuration du cycle de simulation (et en particulier l'existence du cycle interne à temps-constant) résulte également de la définition à donner au concept de "temps". En effet, dans les applications industrielles, le "cycle de scrutation" du graphe de commande est, par exemple, variable entre la milliseconde et le dixième de seconde alors que l'évolution du procédé est, en général, beaucoup plus lente.

IV.4 - Arrêt de la simulation

La simulation s'interrompt dans les cas suivants :

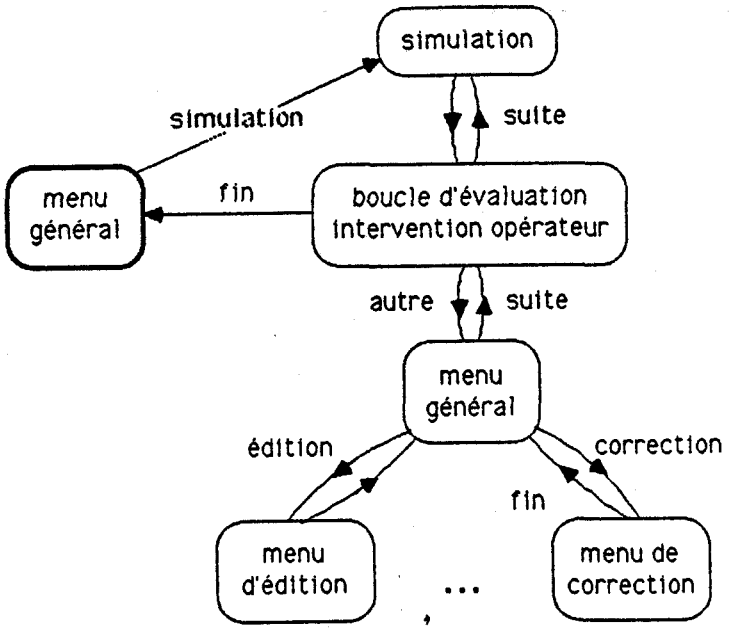
- Aucune transition, aucune règle n'est activable et les échéanciers sont vides. Le système est alors bloqué.

- Un point d'arrêt a été atteint. Les points d'arrêts sont définis par l'opérateur et peuvent être associés à l'échéance d'une date fixée, au franchissement d'une transition, au marquage d'une place, à l'activation d'une règle, ou plus généralement, intégrée à n'importe quelle expression Lisp.

- Un indéterminisme a été détecté (nous expliciterons plus précisèment ce point dans la suite de notre exposé).

- L'intervention programmée "Intervention Opérateur" est déclenchée.

Dans tous les cas, la simulation est suspendue et le contrôle est rendu à l'opérateur dans une boucle d'évaluation similaire au "Top-level" mais sans dépiler les blocs de contrôle des environnements antérieurs de la pile du système Lisp. Cela signifie concrètement que la simulation peut être éventuellement reprise au point exact où elle fût interrompue.



Cela signifie en particulier, que le système modélisé peut être complété ou corrigé en cours de simulation.

IV.5 - Conclusion

Plutôt que d'étudier séparément les caractéristiques de simulation de la partie commande, de la partie décisionnelle et du procédé, nous allons orienter cette présentation selon les approches "par activité" et "par évènement" du cycle de simulation du système complet. A ce sujet, le lecteur est invité à se reporter à l'Annexe A où est décrit une cession de simulation correspondant à l'exemple précédemment édité.

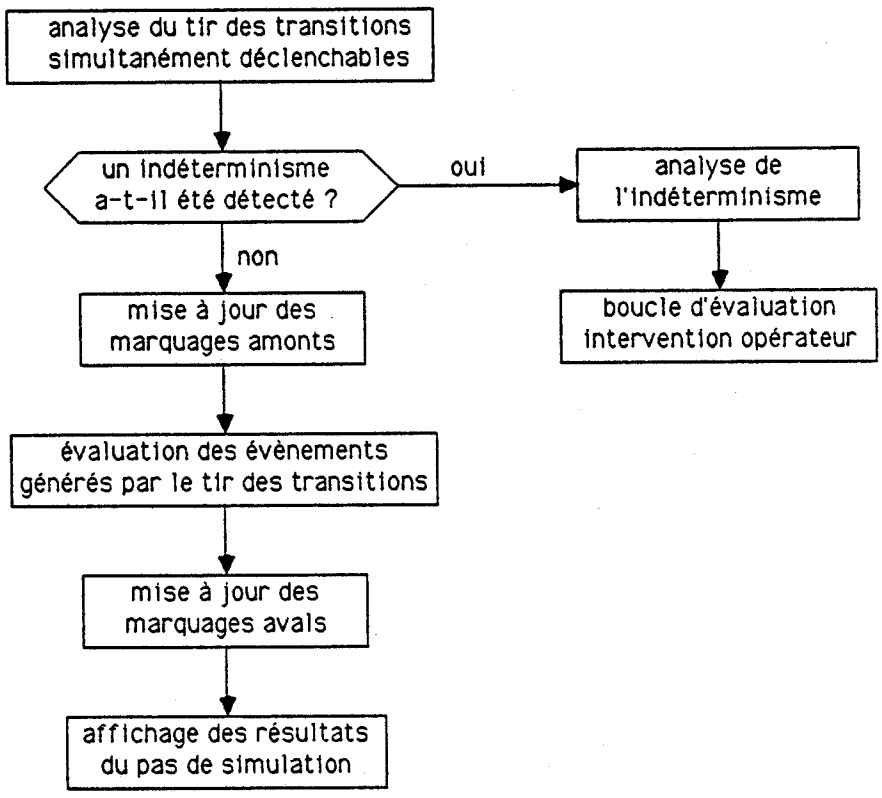
V - CYCLE DE SIMULATION PAR ACTIVITE

V.1 - Introduction

Le cycle de simulation "par activité" constitue le cœur du simulateur de la commande du système modélisé. Il concerne l'évolution instantanée du graphe de Petri et de l'ensemble des règles de surveillance modélisant le contrôle local de ce graphe de commande.

V.2 - Simulation du graphe de commande

L'organigramme détaillé d'un pas de simulation du graphe de commande est le suivant :



La liste des transitions activables est déterminée par l'état du réseau de Petri en début de pas. Afin de limiter cette recherche, seules les transitions en aval des places marquées de type "pc" sont étudiées. Cette première sélection s'opère très rapidement puisque d'une part, la variable "marquage-pc" mémorise la liste des places de type "pc" contenant des marques actives et d'autre part, chaque place possède sur sa p-liste la liste des transitions en aval. Notons

que cette démarche est particulièrement intéressante dans le cas des RdP structurés puisque de tels réseaux ne possèdent, par construction, qu'une place marquée par processus. Dans ce cas, cette première sélection élimine environ 4/5 des transitions à étudier.

Chacune de ces transitions est ensuite étudiée individuellement, c'est-à-dire que les préconditions de franchissement sont analysées.

Les évolutions du marquage amont des transitions effectivement tirables sont alors analysées globalement afin de détecter d'éventuels indéterminismes.

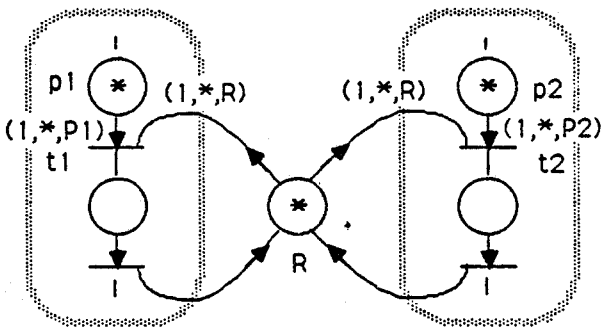
Note : Si la transition est interprétée et possède une condition supplémentaire d'activation référencée par la propriété "évènement-attendu" sur sa p-liste, cette précondition est testée au plus tôt, c'est-à-dire, si possible avant évaluation des préconditions sur le marquage amont, ou après si cette précondition nécessite l'instantiation des variables libres associées à la transition. Ceci, afin de minimiser le temps de traitement.

V.3 - La détection des indéterminismes

Nous distinguons 2 types d'indéterminismes : l'indéterminisme structurel et l'indéterminisme d'agrégation.

V.3.1 - L'indéterminisme structurel

Il apparait lorqu'au moins 2 transitions simultanément déclenchables ne peuvent être simultanément tirées du fait de l'insuffisance globale du nombre de marques disponibles. C'est le cas, par exemple, pour un partage de ressource exclusif où aucune priorité d'accès n'a été définie.

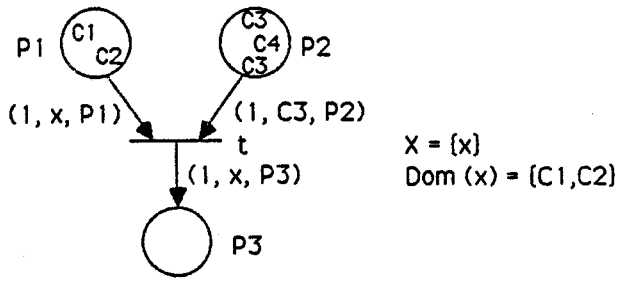


t1 et t2 sont en conflit structurel lors de l'accès à la ressource "R" quand les places "P1" et "P2" sont simultanément marquées.

Le simulateur récupère et analyse les indéterminismes structurels amenant des situations conflictuelles en affichant la liste des transitions en conflit, des marques en nombre insuffisant (ici la marque de type "*" dans la place "R") et l'évolution indépendante de chacune des transitions en conflit (cf Annexe A, page 202, pour un exemple).

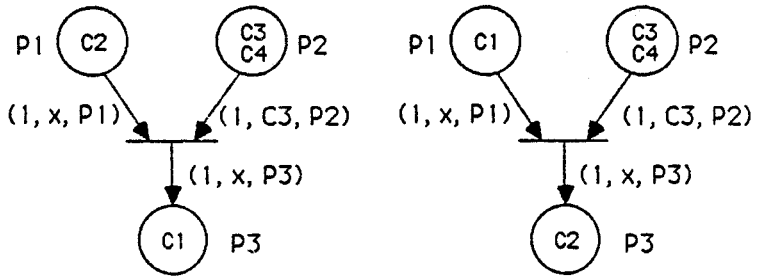
V.3.2 - L'indéterminisme d'agrégation

Prenons l'exemple suivant :



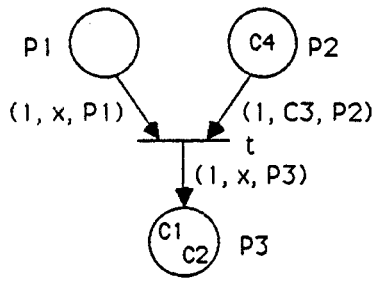
Deux interprétations sont alors possibles :
La transition t est tirable pour

x = C1 ou x = C2



La transition t est tirable simultanément pour

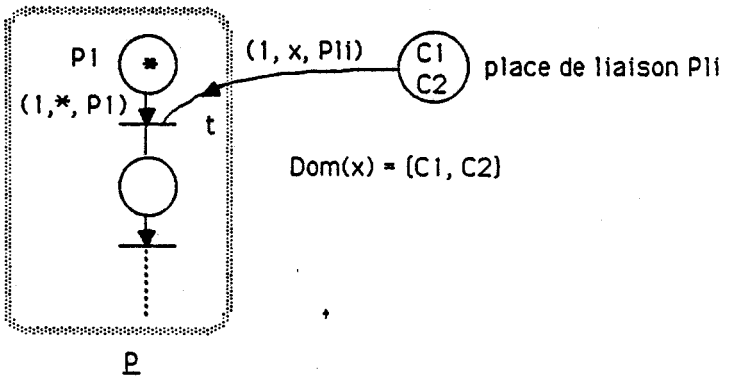
x = C1 et x = C2



Nous appellerons cette configuration, un indéterminisme d'agrégation. Si l'on considère les RdP colorés comme l'agrégation de RdP simples, une telle situation n'est pas anormale. Il suffira de tirer 2 fois la transition "t" dans le même pas de simulation. Permettre ou non l'indéterminisme d'agrégation est laissé en option à l'utilisateur selon son interprétation (cf Configuration du simulateur).

Dans le cas particulier des RdP structurés, une telle configuration est impossible à obtenir puisque les processus sont saufs. Tout indéterminisme d'agrégation qui ne peut être du qu'au marquage non sauf des places de liaison, engendre inévitablement une situation conflictuelle (un indéterminisme structurel).

Exemple :



La transition t est en conflit structurel avec elle-même.

Si l'indéterminisme d'agrégation n'est pas permis par l'opérateur et si un indéterminisme d'agrégation n'amenant pas une situation conflictuelle a été détecté, le simulateur récupère et analyse cet indéterminisme.

V.3.3 - Traitement de l'indéterminisme

Après détection et analyse d'un indéterminisme, le simulateur restitue le graphe dans l'état précédent la mise en évidence de cet indéterminisme et se place dans la boucle d'évaluation "Intervention opérateur". L'opérateur peut alors corriger son modèle, en particulier en ajoutant une règle de décision adéquate avant de poursuivre la simulation à partir du point où elle a été suspendue.

V.4 - Simulation des règles de surveillance

Au début de chaque cycle d'évolution par activité, il y a exploration des règles de surveillance. Cette évaluation s'opère en 2 temps :

- Evaluation des règles d'initialisation,

puis

- Evaluation des règles de surveillance proprement dites (ou règles de paramétrage).

Dans chacun des 2 cas, les règles sont étudiées en séquence. Cependant, l'ordre des règles qui dépend de l'ordre dans lequel elles ont été définies, n'intervient pas au niveau du paramétrage résultant dans la mesure où leurs définitions sont cohérentes (des règles cohérentes sont des règles dont les effets ne sont pas contradictoires). Le simulateur ne réalise aucun contrôle de cohérence lors de l'évaluation des règles.

V.5 - Conclusion

Le cycle de simulation par activité constitue la phase d'exécution qui pénalise le plus le temps d'étude malgré les techniques d'accélération réalisées. En effet, toute transition ou toute règle de surveillance étudiée n'est pas forcément active contrairement à l'approche par évènement où toute occurence d'évènement engendre obligatoirement une évolution du système. C'est pourquoi, nous avons cherché à minimiser le temps consacré à cette recherche d'une part en limitant le nombre des transitions à étudier et d'autre part, en évaluant au plus tôt les évènements-attendus associés aux transitions.

VI - CYCLE DE SIMULATION PAR EVENEMENT

VI.1 - Introduction

Ce cycle de simulation "par évènement" regroupe les mécanismes d'évolution du modèle décrivant le procédé et des règles de requête de la partie décisionnelle. Le terme "évènement" est pris ici dans un sens large. Il ne s'applique pas seulement aux occurences d'évènements gérés par les échéanciers mais il est étendu au marquage des places d'interface vers le procédé, auxquelles est associé l'émission d'un message, et au marquage des places d'interface vers la partie décisionnelle décentralisée, auxquelles est associée une règle de requête.

VI.2 - Simulation du modèle décrivant le procédé

L'évolution du modèle décrivant le procédé s'opère en 2 temps :

* Evaluation des messages présents dans l'échéancier associé au procédé (échéancier-po) à la date courante, ce qui correspond à la propagation des messages (ou commandes) émis antérieurement.

* Evaluation des messages associés aux places d'interface de type "po" marquées, ce qui correspond à la prise en compte des commandes nouvellement élaborées par le graphe de commande.

Aucune analyse de cohérence n'est réalisée par le logiciel de simulation. Les messages sont évalués dans l'ordre où ils sont mémorisés dans l'échéancier ou dans l'ordre de déclaration des places d'interface correspondantes, mémorisée dans la variable "marquage-po". Cette analyse de cohérence doit être effectuée au moment où l'on définit les méthodes associées aux objets concernés.

Exemple : Si l'on s'intéresse à l'éventualité de commandes concurrentes pour l'utilisation d'un robot, on définira sur la p-liste des caractéristiques du robot, un champ supplémentaire intitulé "activation" mémorisant la participation du robot à un ensemble de tâches. Pour toute nouvelle commande du robot,

correspondant à l'activation d'une méthode particulière, on testera, au niveau de cette méthode, le champ "activation".

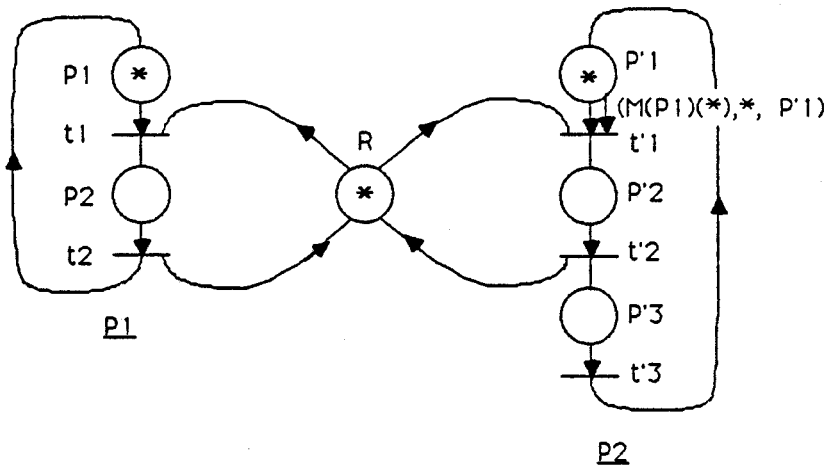
VI.3 - Simulation des règles de requête

L'évaluation des règles de requête dépend du marquage des places d'interface de type "pd" auxquelles elles sont associées. Là aussi, l'ordre d'activation des règles dépend de l'ordre des places mémorisées dans la variable "marquage-pd" et l'analyse de cohérence de l'ensemble des règles reste à la charge du concepteur.

VI.4 - Le concept de blocage et la notion de time-out

Le simulateur détecte un blocage quand le système simulé ne peut plus évoluer et que les échéanciers sont vides. Ces blocages concernent donc l'ensemble des évolutions du système : aucune transition n'est tirable, aucune règle n'est activable et aucun objet n'est susceptible de recevoir de message. Plus précisèment, il est difficile, par simulation, de détecter un blocage partiel (autrement appelé "famine") où un ou plusieurs processus ne peuvent plus évoluer sans que le système ne soit globalement bloqué.

Considérons le système composé des 2 processus P1 et P2.



L'arc adaptatif en aval de la place P1' règle le conflit d'accès à la ressource R en donnant priorité au processus P1 en cas de requête simultanée. Si l'on observe l'évolution du processus P1, on remarque qu'il possède 2 états : soit la place P2 est marquée et P1 utilise la ressource R, soit la place P1 est marquée et la priorité statique adoptée alloue la ressource au processus P1. Le processus P2 n'accède donc jamais à la ressource R et, par voie de conséquence, il ne peut jamais évoluer. C'est un cas de blocage partiel (on dit également qu'il y a famine) bien que globalement, le système ne soit jamais bloqué.

Cette notion de famine est liée à la propriété de vivacité d'un graphe. En effet, les transitions t'_1, t'_2 et t'_3 de notre exemple ne sont même pas quasi-vivantes /BRA 83/. Le graphe ne peut donc pas être vivant. Si cette propriété de famine peut être mise en évidence par des méthodes de validation, elle demeure par contre indétectable par simulation.

Un deuxième cas de blocage partiel provient de l'interprétation liée au graphe par adjonction du modèle spécifique décrivant le procédé. Il est possible, au niveau des objets, de simuler les pannes du procédé soit directement par l'interruption "Intervention Opérateur", soit en intégrant dans les caractéristiques des objets les lois de défaillances adéquates. L'occurence d'une panne sur une machine provoque en général le blocage du processus de commande de cette machine. Celui-ci est alors bloqué dans l'attente d'un accusé-réception de fin de tâche qui n'est pas généré puisque la tâche correspondante est inhibée par la panne.

C'est pourquoi, nous avons décidé d'intégrer dans notre simulateur, une primitive spéciale apte à aborder ce genre de problème : le **time-out**.

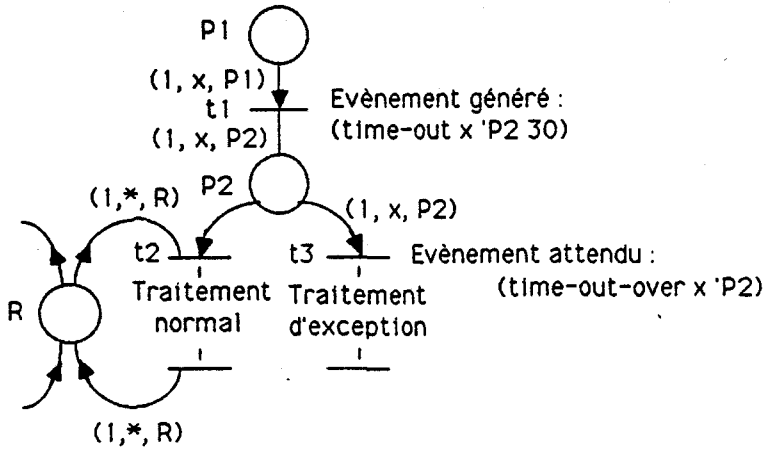
Cette primitive, dont la syntaxe est la suivante :

(time-out <couleur> <place> <durée>)

peut être activée directement par l'opérateur ou définie comme un évènement généré par une transition ou intégrée au sein d'une expression Lisp (en particulier, l'énoncé d'une règle).

L'activation de cette primitive amorce un time-out (ou "chien de garde" ou "temps enveloppe") associé à la <couleur> de la <place> précisée. Si, pendant le temps correspondant à la <durée> définie, une marque de cette <couleur> quitte la <place> soit consécutivement au tir d'une transition en aval, soit par "forçage" du niveau decisionnel ou du procédé, le time-out est désarmé. Par contre, si au bout de ce laps de temps, le time-out n'a pas été désarmé, un évèment "fin-de time-out dépassé" est généré. Cet évènement provoque l'affichage d'un message et peut être récupéré comme évènement attendu par le tir d'une transition, par exemple.

Exemple : Time-out et traitement d'exception.



Le tir de la transition tl amorce un time-out de 30 secondes pour la marque de couleur <x> dans la place P2. Si au bout de ces 30 secondes, cette marque en attente de la ressource R n'a pas pu évoluer (tir de la transition t2) alors la transition t3 est activable et le traitement d'exception est déclenché.

VI.5 - Conclusion

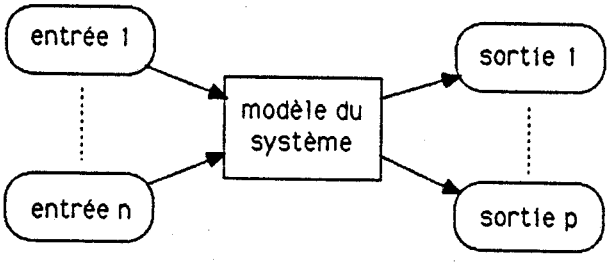
L'intégration du modèle décrivant le procédé et du niveau hiérarchique de prise de décision ne pénalise pas outre mesure les performances du simulateur puisque cette approche est fondamentalement évènementielle.

Si la durée des communications physiques entre d'une part, la partie commande réelle et d'autre part, le procédé et le niveau hiérarchique de prise de décision, est un facteur important dans le comportement du système étudié, les échanges entre procédé, partie décisionnelle et partie commande seront temporisés afin de tenir compte des délais induits.

VII - VERS UNE METHODOLOGIE DE LA SIMULATION

VII.1 - Introduction

Très globalement, un système de production flexible peut être représenté par le schéma suivant :



Au niveau des blocs d'entrée apparaissent des pièces qui pénètrent dans le système. Elles sont conditionnées sélectivement selon leur type et selon l'état du système puis acheminées vers les organes de sortie.

L'état du système dépend essentiellement du flux des pièces qui y sont conditionnées. La gestion de ce flux est externe au fonctionnement du graphe modélisant la partie commande (RdP non autonome). Elle engendre un nombre d'états en général très important dans le cas des systèmes de taille industrielle. Considérons un système capable de conditionner simultanément jusqu'à 10 pièces de 10 types différents, chaque pièce passant sélectivement par 10 étapes différentes. Un calcul approximatif montre que la taille du "graphe d'états" correspondant est de l'ordre de 10^{20} soit :

$$(\text{nombre de type} \times \text{nombre d'étapes})^{\text{nombre de pièces}}$$

$$= (10 \times 10)^{10} = 10^{20}.$$

L'un des objets essentiels de la simulation concerne l'étude du flux des pièces circulant dans le système : temps de transfert, débit, ordre des opérations et, en particulier, la détection des blocages. On peut reprocher à la simulation, un manque de rigueur dans la détermination des blocages. C'est en effet une caractéristique fondamentale de la simulation de ne valider que les comportements simulés. Ne pas exhiber de blocages ne prouve pas l'absence de configurations bloquantes ! Notons cependant que le concept de blocage étudié ici, s'il s'apparente à la notion de vivacité, dépasse le cadre des outils de validation mal adaptés au traitement de graphes d'état de très grandes tailles.

Pour répondre au mieux à ces contraintes, nous proposons dans la phase de conception et d'évaluation de la commande d'adopter une démarche structurée organisée en diverses étapes, chaque étape validant un aspect du système conçu et susceptible de mettre en évidence un type de blocage éventuel.

VII.2 - Emulation des entrées de pièces

VII.2.1 - Introduction

L'ordre et la fréquence d'entrée des pièces dans le système est un facteur déterminant de son évolution. L'apparition d'un blocage, par exemple, résulte généralement de cette séquence d'entrée et son origine relève plutôt de considérations d'ordonnancement que de la notion de vivacité en terme de validation de RdP. En effet, les méthodes de validation actuelles s'appliquent au réseaux de Petri **autonomes** et prennent surtout en considération la structure du graphe. Ici, en plus de la structure, interviennent la séquence et la fréquence d'entrée.

C'est pourquoi, l'émulation des entrées est un point particulièrement important dans l'évaluation d'un système de production. Nous avons déjà justifié (cf Chapitre I, § III.7) l'intérêt qu'il y a de pouvoir disposer d'une fonction "entrée" assurant par ailleurs une réelle flexibilité dans la gestion des entrées.

Le niveau de spécification de la fonction d'entrée dépend de l'étape atteinte dans la démarche progressive que nous proposons. Outre sa définition, cette fonction est également caractérisée par la façon dont elle est intégrée au modèle.

VII.2.2 - Spécification de la fonction d'entrée

La spécification de la fonction d'entrée dépend de la **séquence** des pièces pénétrant dans le système. Nous distinguons 3 types de séquencements.

*** La séquence intéractive :**

L'opérateur décide, à l'instant où une pièce pénètre dans le système, du type de cette pièce. Cette saisie s'opère donc en "temps réel". Une telle fonction peut être définie en Le_Lisp de la manière suivante :

```
(de entrée ()
    (prog (rep)
          (print "Type de la pièce :")
          (setq rep (read))
          (ajouter 1 rep 'place-entrée)))
```

où "place-entrée" est la place du RdP modélisant l'entrée du système.

*** La séquence prédéfinie :**

L'opérateur construit, avant la simulation, la liste des pièces fournissant la séquence d'entrée.

```
(setq réserve '(m1 m1m2 m2 ... m2m1 m1))
(de entrée ()
    (ajouter 1 (nextl réserve 'place-entrée))
```

*** La séquence aléatoire :**

Le type de la pièce est défini aléatoirement parmi les différents types de pièces possible. Si le domaine d'entrée des pièces dans le système est (m1,m2,m1m2,m2m1), on pourra définir la fonction d'entrée de la manière suivante :

```
(de entrée ()
    (ajouter 1 (nth (random 0 4) '(m1 m2 m1m2 m2m1))
               'place-entrée))
```

VII.2.3 - Intégration de la fonction d'entrée au modèle

La fonction d'entrée peut être associée à une règle de surveillance, à une règle de requête, à un évènement, à l'étiquetage du poids d'un arc ou géré par un objet spécifique du modèle décrivant le procédé, ... Le choix du point d'intégration dépend essentiellement de la fréquence d'entrée de pièces désirée. Nous distinguons 3 types d'itérations.

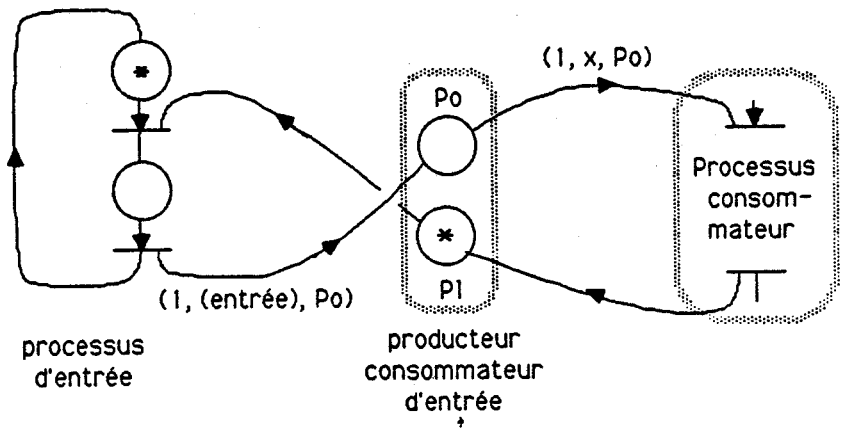
*** L'itération intéractive :**

L'opérateur gère en cours de simulation (en "temps réel") l'entrée des pièces dans le système par le biais de l'interruption "intervention opérateur".

```
INTERVENTION OPERATEUR > (entrée)
Type de la pièce : m1
INTERVENTION OPERATEUR > suite
```

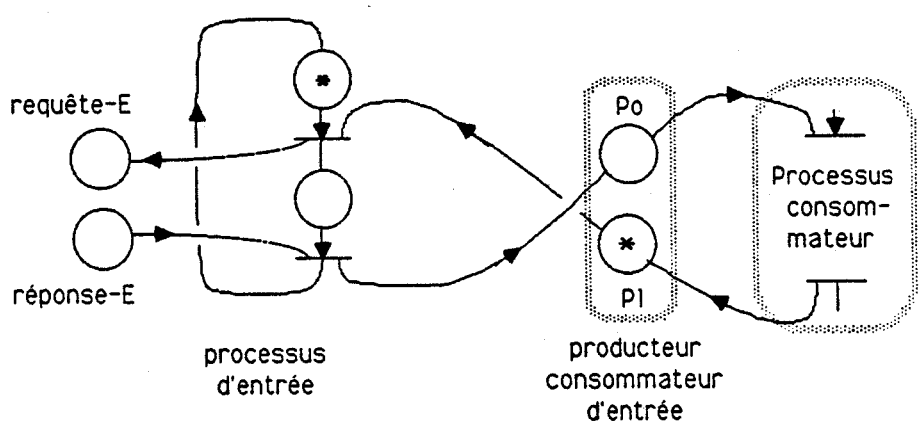
*** L'itération au plus tôt :**

Elle modélise un flux continu de pièces en entrée.




```
(de entrée () (nextl réserve))
(setq réserve '(m1 ... m2)
```

L'émulation de l'entrée peut également être réalisée par une règle de requête.

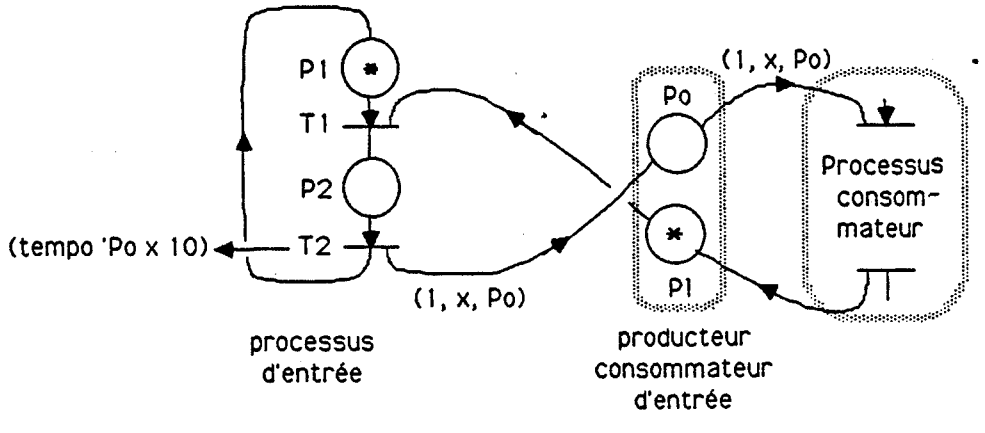



```
(plist 'requête-E
       '(type pd
         règle-activée requête-entrée))
(plist 'requête-entrée
       '(type requête
         énoncé (if ((m 'requête-E '*) > 0)
                    (progn (enlever 1 '* 'requête-E)
                           (ajouter 1 (entrée) 'réponse-E)))))
```

*** L'itération évènementielle :**

L'occurence d'entrée des pièces est liée au cycle "par évènement" du simulateur. Ceci peut être réalisé pratiquement de plusieurs façons : les évènements d'entrée de pièces sont prévus initialement dans l'échéancier ou plus simplement générés par temporisation de la structure émulant l'entrée.

Exemple :




```
(plist 'T2
       '(var-libre (x)
         dom-x (entrée)
         arc-pré ((1 '* 'P2))
         arc-post ((1 '* 'P1) (1 x 'P0))
         evt-généré (tempo x 'P0 10)))
```

VII.2.4 - Conclusion

Le choix de la séquence d'entrée (définition de la fonction d'entrée) et de sa fréquence (le mode d'intégration au modèle) permettent de définir la politique d'entrée adéquate aux résultats de simulation désirés :

- Génération aléatoire au plus tôt,
- Génération intéractive au plus tôt,
- Génération aléatoire évènementielle,
- etc ...

VII.3 - Les étapes de la simulation

Les étapes de simulation que nous proposons sont adaptées à la définition progressive du système.

a) Définition du graphe de Petri coloré réalisant a priori les diverses fonctionnalités du système.

La simulation correspondante consiste à introduire sélectivement une seule pièce à la fois dans le système afin de vérifier l'adéquation du conditionnement réalisé et de la gamme prévue.

b) Définition des règles de décision et des structures de mutuelle exclusion apte à résoudre a priori les indéterminismes du système.

La phase de simulation correspondante consiste en un jeu d'essais où les pièces sont générées aléatoirement au plus tôt afin de vérifier, par une méthode essai/erreur, l'adéquation des règles adoptées.

c) Temporisation des places correspondant aux divers usinages.

Toutes les autres durées opératoires et en particulier les temps de transferts sont assimilés à zéro. De cette façon, les points d'usinages sont considérés comme des goulots d'étranglement.

La simulation d'un tel graphe permet d'atteindre rapidement un état de saturation du à l'engorgement des structures en amont des différentes machines. On vérifie ainsi que les processus des structures de transport sont capables de résorber cette saturation. Cette situation est en effet généralement génératrice de blocages, s'il en existe.

Une démarche complémentaire consiste à désactiver sélectivement (par "gel" par exemple) les divers processus d'usinage. Le résultat des simulations correspondantes fournit des informations intéressantes pour la définition d'éventuels modes dégradés. En effet, si la commande du système est relativement connexe, l'arrêt d'une machine provoque le blocage du système complet. Ce problème devra être résolu si l'on définit un mode de marche dégradé correspondant à la panne de cette machine.

d) Temporisation des opérations de transfert uniquement.

Cette démarche, duale de la précédente, assimile les organes de transferts à des goulots d'étranglement.

e) Temporisation de toutes les places caractéristiques du graphe ou définition d'un modèle temporisé du procédé.

La simulation s'opère alors en 2 temps :

- Entrée sélective d'une seule pièce par essai de simulation afin d'évaluer le temps minimum de conditionnement de chaque pièce.

- Génération d'un flux de pièces en entrée afin d'évaluer globalement les performances du système.

f) Comparaison des performances de diverses configurations de l'atelier correspondant au dimensionnement des ressources (taille des zones de stockage intermédiaire, par exemple) ou à la définition de diverses stratégies de décision.

Il est intéressant d'utiliser dans ce cas un échantillon assez important de pièces à entrer comme étalon des divers jeux d'essai. Le temps global de conditionnement de l'échantillon complet est alors un critère déterminant de comparaison des performances des diverses solutions envisagées.

VII.4 - Conclusion

La liste des étapes de simulation proposées est loin d'être exhaustive. Elle s'est révélée à l'usage intéressante dans le phase de conception et d'évaluation de la commande d'un système flexible car elle permet de mettre en évidence et de résoudre les blocages potentiels et, éventuellement, de dimensionner le système.

CONCLUSION

Le logiciel de simulation présenté, plus qu'un simple outil de simulation, s'est révélé, à l'usage, un véritable outil de conception assistée. En définissant à la fois la structure de donnée (le modèle du système) et le cadre de simulation (la configuration du simulateur), l'utilisateur oriente l'analyse en fonction du point précis qu'il désire étudier. L'intéractivité du logiciel facilite la maîtrise complète d'une telle étude.

Les résultats fournis par la simulation (mise en évidence d'un indéterminisme, d'un blocage, recueil de statistiques, ...) sont exploités et interprétés de manière intéractive par l'utilisateur. Ils permettent, dans un premier temps, une meilleure compréhension de la dynamique du système étudié. Ils conduisent, dans une seconde phase, à déduire des "expériences enregistrées", les modifications à apporter et le paramétrage à définir afin d'atteindre le fonctionnement requis. L'intéractivité et les informations ou "messages" délivrés par le simulateur, soit à la demande de l'utilisateur, soit par le simulateur lui-même dans différents contextes (blocages, indéterminismes, ...) "assistent" véritablement le concepteur du système flexible de production étudié, notamment dans l'élaboration des règles de résolution de conflits.

C H A P I T R E III

APPLICATION A UN EXEMPLE

INTRODUCTION

Dans ce dernier chapitre, nous proposons d'illustrer sur un exemple, les modèles et les méthodes décrits antérieurement et leur mise en œuvre à partir de l'outil de simulation présenté dans ce mémoire.

L'exemple choisi /BOU 86/ constitue une cellule flexible de production considérée ici comme autonome, mais supposée intégrée dans une chaîne de production, ce que nous traduisons par la définition de l'émulation des pièces en entrée de la cellule.

La démarche adoptée ici répond à un double objectif. Le premier consiste à résoudre l'indéterminisme du système de commande par une définition du niveau décisionnel qui réponde aux spécifications. On cherche ensuite à optimiser les performances du système en dimensionnant certains paramètres liés à la configuration matérielle de la cellule.

I - PRESENTATION DE LA CELLULE FLEXIBLE

La cellule flexible (Fig. 14) est composée de :

* Un organe d'entrée FIFO-IN,
* Un organe de sortie FIFO-OUT,
* 2 Machines M1 et M2. Chaque machine est autonome et dispose d'un poste d'usinage P et de 2 bancs de transfert T-IN (pour le chargement) et T-OUT (pour le déchargement),
* 1 ou 2 Robots qui assurent le transfert des pièces entre les divers organes constitutifs du système.

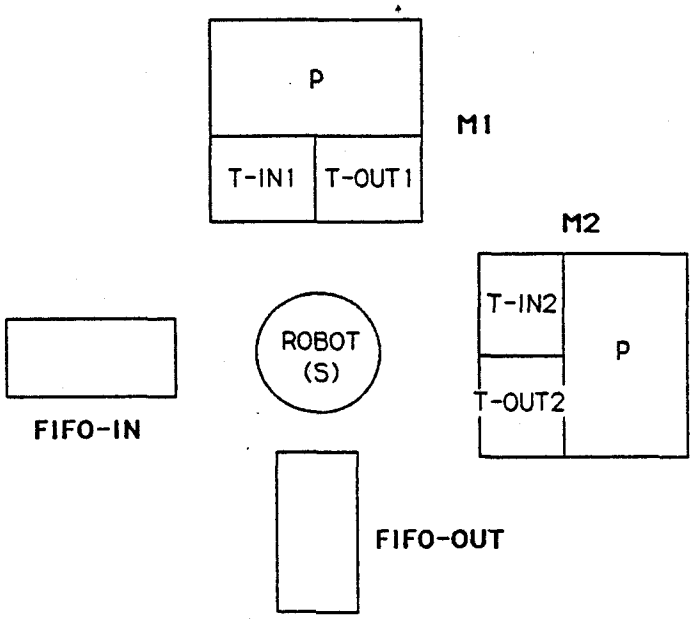


FIGURE 14

Dans cet atelier circulent 4 types de pièces. Ces pièces sont usinées :

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

II - MODELISATION DE LA PARTIE COMMANDE

Le graphe de transfert de cet ilôt flexible a été présenté au Chapitre I, § III.4.3. Le passage au graphe de commande (Fig. 15) s'opère globalement de la manière suivante : chaque place (média) et chaque transition du graphe de transfert est développé en un processus particulier, les liaisons entre processus étant réalisées ici par des structures producteur/consommateur. Seules les médias "FIFO-IN" et "FIFO-OUT" gérant respectivement l'entrée et la sortie des pièces n'ont pas été représentés, car l'ilôt est considéré ici comme autonome. La gestion de l'entrée, en particulier, sera réalisée, par une fonction d'entrée.

Dans ce graphe de commande, on distingue :

- les processus d'usinage M1 et M2,
- les processus de transfert :
  * FIFO-IN ⟶ M1 et FIFO-IN ⟶ M2 pour les transferts de l'organe d'entrée vers les centres d'usinages,
  * M1 ⟶ M2 et M2 ⟶ M1 pour les transferts entre centres,
  * M1 ⟶ FIFO-OUT et M2 ⟶ FIFO-OUT pour les transferts des centres d'usinages vers l'organe de sortie.

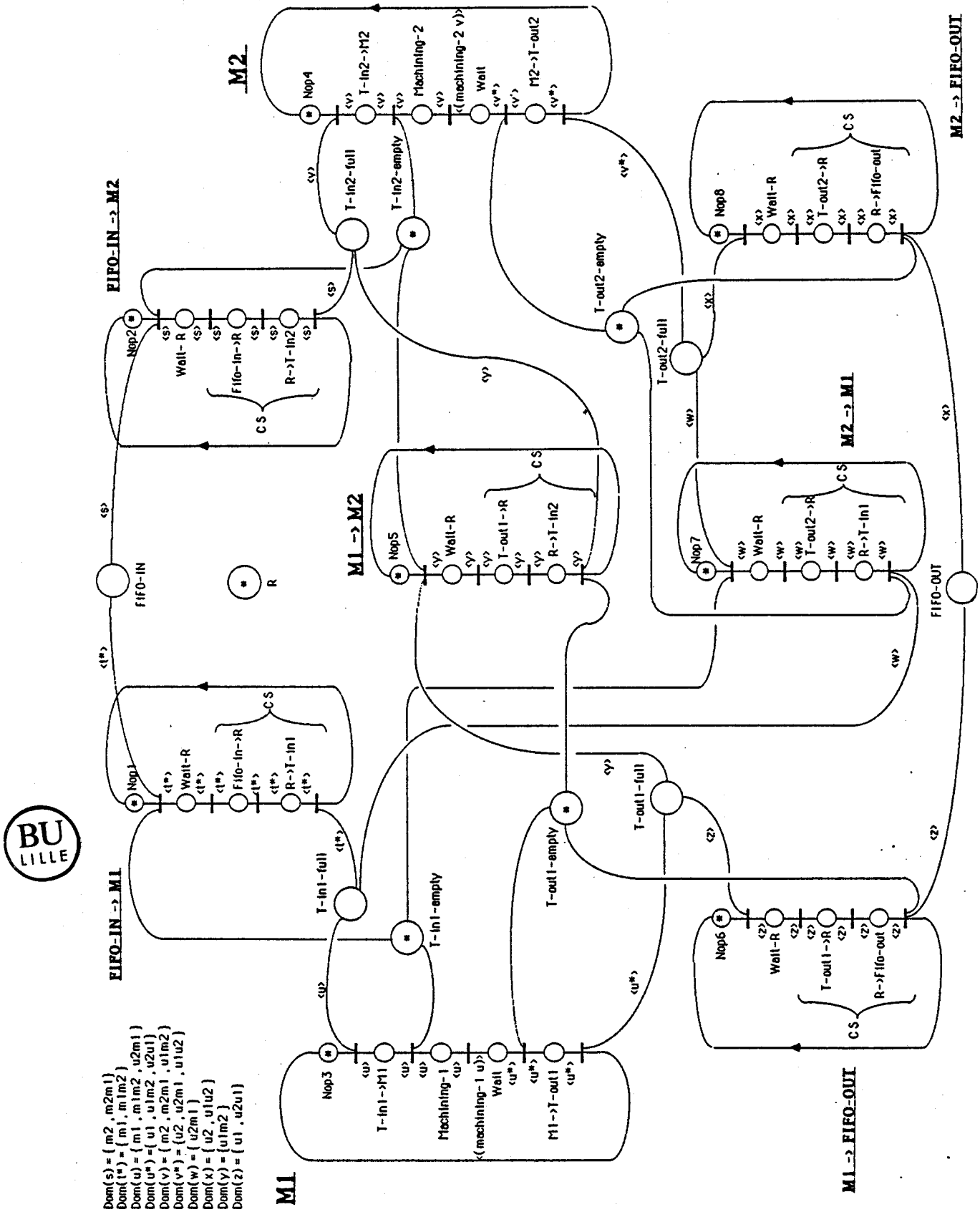
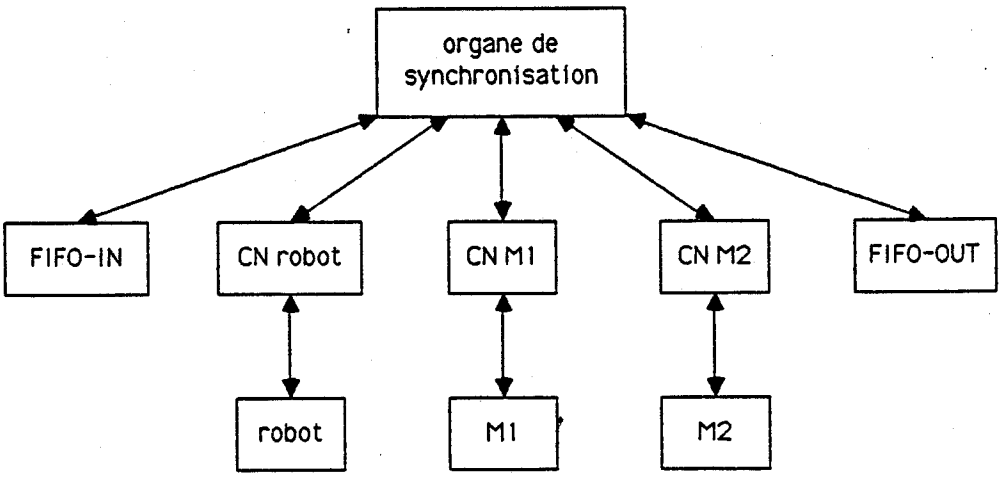

M2
FIFO-IN -> M2
M1 -> M2
M2 -> M1
M2 -> FIFO-OUT
FIFO-IN -> M1
M1 -> FIFO-OUT
M1
FIFO-IN
FIFO-OUT
R
Nop1
Nop2
Nop3
Nop4
Nop5
Nop6
Nop7
Nop8
Wait-R
Fifo-in->R
R->T-in1
R->T-in2
T-in1-full
T-in1-empty
T-in2-full
T-in2-empty
T-in1->M1
T-in2->M2
Machining-1
(machining-1 u)
Wait
M1->T-out1
M2->T-out2
T-out1-R
T-out2-R
R->fifo-out
T-out1-full
T-out1-empty
T-out2-full
T-out2-empty
CS
Dom(s) = { m2 , m2m1 }
Dom(i*) = { m1, m1m2 }
Dom(u) = { m1 , m1m2 , u2m1 }
Dom(u*) = { u1 , u1m2 , u2u1 }
Dom(v) = { m2 , m2m1 , u1m2 }
Dom(v*) = { u2 , u2m1 , u1u2 }
Dom(w) = { u2m1 }
Dom(x) = { u2 , u1u2 }
Dom(y) = { u1m2 }
Dom(z) = { u1 , u2u1 }

FIGURE 15

Afin de permettre la coopération entre les divers organes actifs (les machines M1 et M2, et le robot) on adopte l'architecture répartie suivante pour le système de commande :



Le graphe de la Figure 15 correspond aux processus implantés au niveau de l'organe de synchronisation. Les commandes numériques des machines M1, M2 et du robot sont considérées ici, comme appartenant au modèle du procédé (cf Chapitre I, § III.2).

La couleur d'une marque correspond au programme d'usinage de la pièce considérée et à son état d'avancement dans sa gamme d'usinage. Ainsi la couleur "m2m1" caractérise une pièce devant être usinée sur M2 puis sur M1. Après usinage sur M2, cette couleur sera transformée en "u2m1" puis, après passage sur M1, en "u2u1". les transformations de couleur sont assurées au niveau du modèle par les fonctions "machining-1" et "machining-2" associées aux arcs avals des transitions successeurs des places "machining-1" et "machining-2" des processus d'usinage.

Ces fonctions sont définies de la manière suivante :

```
(de machining-1 (x)
    (cond ((eq x 'm1) 'u1)
          ((eq x 'm1m2) 'u1m2)
          ((eq x 'u2m1) 'u2u1)))
(de machining-2 (x)
    (cond ((eq x 'm2) 'u2)
          ((eq x 'm2m1) 'u2m1)
          ((eq x 'u1m2) 'u1u2)))
```

La ressource R représente la disponibilité du ou des robots assurant les divers transferts. Elle est initialisée à 1 dans le cas d'un seul robot et à 2 dans le cas de 2 robots. Les arcs reliant la place R aux divers processus utilisant l'organe de transfert (blocs "SC" du graphe) n'ont pas été représentés afin de préserver la lisibilité du graphe.

III - OBJECTIFS DE LA SIMULATION

La première étape consiste à introduire sélectivement une pièce à la fois dans la place "FIFO-IN" afin de vérifier la bonne définition du graphe (définition des variables libres, de l'étiquetage des arcs, des fonctions utilisées, ...). On vérifie ainsi l'adéquation du conditionnement réalisé et de la gamme prévue, pour chaque type de pièce.

Les étapes suivantes s'articulent selon deux directions différentes : la résolution de l'indéterminisme et l'optimisation des performances.

IV - RESOLUTION DE L'INDETERMINISME

Deux types de conflits apparaissent lors de l'analyse du graphe.

* Les processus FIFO-IN ⟶ M1 et M2 ⟶ M1 peuvent demander l'accès simultané au tampon d'entrée de la machine M1 (T-IN1-EMPTY). De même, les processus FIFO-IN ⟶ M2 et M1 ⟶ M2 peuvent demander l'accès simultané au tampon d'entrée de la machine M2 (T-IN2-EMPTY).

* Les processus de transfert (FIFO-IN ⟶ M1, FIFO-IN ⟶ M2, M1 ⟶ M2, M2 ⟶M1, M1 ⟶ FIFO-OUT et M2 ⟶ FIFO-OUT) peuvent demander l'accès simultané à la section critique "SC" représentant l'exclusion mutuelle lors de l'accès au robot.

Afin de régler ces 2 familles de conflits potentiels, nous avons choisi une politique de prudence visant à éviter le blocage du système par engorgement des organes de transfert et des tampons de stockages intermédiaires. Ainsi, c'est l'état d'avancement des pièces par rapport à leur gamme d'usinage qui détermine la pièce prioritaire en cas de conflit. Si une requête est déclarée moins prioritaire, la marque correspondante sera "gelée" pour le pas de simulation "par activité" considéré.

Pour régler la 1ère famille de conflits, la priorité est donnée aux transferts de pièces entre machines par rapport à l'introduction d'une nouvelle pièce dans le système. Cela se traduit par les 2 règles de surveillance suivantes :

Règle PRIO / M1 ⟶ M2 / FIFO-IN ⟶ M2 :

```
(type       surveillance
 var-libre  (y)
 énoncé     (if (and (eq (m y 'T-OUT1-FULL) 1)
                     (eq (m '* 'T-IN2-EMPTY) 1))
                (geler '* 'nop2)))
```

Cette règle s'interprète de la manière suivante :

"Si une pièce de type "y" (variable libre globale dont le domaine est {ulm2}) est susceptible d'être transférée sur M2 et si le tampon d'entrée de M2 est libre, alors on gèle la marque "*" dans la place "nop2" du processus FIFO-IN ⟶ M2 afin d'empêcher toute nouvelle entrée de pièce depuis FIFO-IN".

Règle PRIO / M2 ⟶ M1 / FIFO-IN ⟶ M1 :

```
(type      surveillance
 var-libre (w)
 énoncé    (if (and (eq (m w 'T-OUT2-FULL) 1)
                    (eq (m '* 'T-IN1-EMPTY) 1))
               (geler '* 'nop1)))
```

Notes : - La couleur "*" correspond aux marques non colorées du graphe (places "nop1", ..., 'nop8", "R", etc ...).
- La fonction "geler" utilisée ici correspond à l'implémentation Lisp de la primitive "geler" présentée dans la 1ère partie.

On définit également les 2 règles d'initialisation correspondantes :

Règle DEGEL-NOP2 :

```
(type   initialisation
 énoncé (if (dom-gelé 'nop2) (dégeler '* "nop2")))
```

La fonction "dom-gelé" est une primitive dont l'appel restitue la liste des couleurs gelées dans la place précisée en argument.

L'interprétation de cette règle est la suivante :

"Si la marque "*" de la place "nop2" est gelée (par activation de la règle "PRIO / M1 ⟶ M2 / FIFO-IN ⟶ M2" au pas de simulation précédent), alors on la dégèle afin de lui donner la possibilité d'évoluer au pas considéré.

Règle DEGEL-NOPl :

```
(type   initialisation
 énoncé (if (dom-gelé 'nopl) (dégeler '* 'nopl)))
```

Pour règler la 2ème famille de conflits, nous résoudrons les accès simultanés possibles des 6 processus de transfert concernés, 2 par 2, ce qui résout a fortiori les conflits où 3, 4, 5 ou 6 processus requièrent l'accès à la section critique "SC". Nous dénombrons donc C_6^2 = 15 conflits à résoudre. Sur ces 15 conflits, 10 sont potentiellement effectifs. En effet, le conflit entre :

FIFO-IN ⟶ M1 et FIFO-IN ⟶ M2

est impossible, de même que les conflits :

M2 ⟶ M1 avec M2 ⟶ FIFO-OUT et

M1 ⟶ M2 avec M1 ⟶ FIFO-OUT.

Les conflits :

M1 ⟶ M2 avec FIFO-IN ⟶ M2 et

M2 ⟶ M1 avec FIFO-IN ⟶ M1

sont réglés par les règles :

PRIO / M1 ⟶ M2 / FIFO-IN ⟶ M2 et

PRIO / M2 ⟶ M1 / FIFO-IN ⟶ M1.

Les 4 règles de surveillance suivantes règlent les conflits restants (la première en résout 2, la deuxième 1, la troisième 6 et la quatrième 1).

Règle PRIO / M* ⟶ M* / FIFO-IN ⟶ M* :

```
(type       surveillance
 var-libre  (t y)
 dom-t      '(m1 m2 m1m2 m2m1)
 dom-y      '(u1m2 u2m1)
 énoncé     (if (and (eq (m y 'wait-R) 1)
                     (eq (m t 'wait-R) 1))
                 (geler t 'wait-R)))
```

Cette première règle traduit la priorité du transfert inter-machines par rapport à l'entrée d'une nouvelle pièce. Si une marque "t" (non usinée) et une marque "y" (partiellement usinée) requièrent simultanément la ressource "R", la marque "t" est gelée.

Règle PRIO / M1 ⟶ M2 / M2 ⟶ M1 (priorité du transfert de M1 vers M2 par rapport au transfert de M2 vers M1) :

```
(type       surveillance
 var-libre  (y w)
 énoncé     (if (and (eq (m y 'wait-R) 1)
                     (eq (m w 'wait-R) 1))
                 (geler w 'wait-R)))
```

Règle PRIO / * ⟶ FIFO-OUT / * ⟶ * (priorité à la sortie de pièce par rapport à tout autre transfert) :

```
(type       surveillance
 var-libre  (x w)
 dom-x      '(u1 u2 ulu2 u2ul)
 dom-w      '(u2m1 ulm2 m1 m1m2 m2 m2m1)
 énoncé     (if (and (eq (m x 'wait-R) 1)
                     (eq (m w 'wait-R) 1))
                 (geler w 'wait-R)))
```

Règle PRIO / M2 ⟶ FIFO-OUT / M1 ⟶ FIFO-OUT (priorité de la sortie depuis M2 par rapport à la sortie depuis M1) :

```
(type       surveillance
 var-libre  (x z)
 énoncé     (if (and (eq (m x 'wait-R) 1)
                     (eq (m z 'wait-R) 1))
                 (geler z 'wait-R)))
```

On définit la règle d'initialisation correspondante à ces 4 règles.

Règle DEGEL-WAIT-R :

```
(type       initialisation
 var-libre  (z)
 dom-z      (dom-gelé 'wait-R)
 énoncé     (dégeler z 'wait-R))
```

Afin de valider les règles de décision adoptées, nous avons réalisé un jeu d'essais de simulation où les pièces sont générées aléatoirement au plus tôt. L'ensemble des règles définies résout effectivement les indéterminismes pressentis.

V - EVALUATION ET OPTIMISATION DES PERFORMANCES DU SYSTEME

V.1 - Première Etape

Les places correspondant aux usinages sont temporisées : 3000 secondes pour "machining-1" et 2000 secondes pour "machining-2". Toutes les autres durées opératoires (temps de transferts) sont nulles afin d'assimiler les points d'usinages à des goulots d'étranglement (cf Chapitre II, § VII.3.c). On vérifie ainsi que les commandes de transfert adoptées (processus de transferts) sont capables de résorber la saturation engendrée.

En désactivant sélectivement le processus M1 (respectivement M2) en gelant la marque de couleur "*" dans la place "nop3" (respectivement "nop4"), on observe le blocage du système. Si, par exemple, la machine M1 est désactivée, la présence d'une marque de type "m1" ou "m1m2" dans la place d'entrée "FIFO-IN" bloque complètement le système si la place "T-IN1-FULL" est occupée. La définition des modes de marche dégradés correspondant aux pannes des machines M1 et M2 (que nous ne traiterons pas ici) devra prendre en compte l'évacuation des pièces ne pouvant pas être usinées.

V.2 - Deuxième Etape

Toutes les places d'usinage et de transfert sont temporisées. Les temps de transfert à l'intérieur des machines M1 et M2 au niveau des places "T-IN1 ⟶ M1", "M1 ⟶ T-OUT1", "T-IN2 ⟶ M2" et "M2 ⟶ T-OUT2" sont de 100 secondes ainsi que tous les temps de transfert utilisant le robot (FIFO-IN ⟶ R, R ⟶ T-IN1, R ⟶ T-IN2, T-OUT1 ⟶ R, T-OUT2 ⟶ R et R ⟶ FIFO-OUT). Afin de pouvoir comparer diverses configurations du système, nous utiliserons un échantillon de 50 pièces qui servira de séquence de référence aux divers essais (cf Chapitre II, § VII.3.f). L'émulation de l'entrée a été définie de la manière suivante :

La variable "réserve" contient la séquence des pièces à usiner. Cette variable est initialisée à (m2m1 m1 m2 m2 ... m2m1), soit 14 pièces de type "m1", 14 pièces "m2", 12 pièces "m1m2" et 10 pièces "m2m1". La fonction "entrée" est définie comme suit :

```
(de entrée () (nextl réserve))
```

L'entrée au plus tôt d'une pièce dans le système est réalisé au moyen d'une règle de surveillance :

Règle ENTREE-PIECE :

```
(type   surveillance
 énoncé (if (and (not (dom 'FIFO-IN))
                 (dom 'nop1)
                 (dom 'nop2)
                  réserve)
            (ajouter 1 (entrée) 'FIFO-IN)))
```

1er Essai : Le système dispose d'un unique robot pour assurer les divers transferts (le marquage de la place "R" est initialisé à 1).

Le temps total de conditionnement de l'échantillon complet est de 120 200 s. Notons que le temps de calcul effectif est d'environ 5 mn pendant lesquelles 1525 pas de simulation "par activité" ont été effectués à 798 dates différentes. Le nombre de transitions étudiées est de 11 226 (soit environ 1 transition sur 5) pour 848 transitions effectivement tirées. Le nombre de règles étudiées est de 15 250 pour 104 effectivement déclenchées.

2ème Essai : Le système dispose de 2 robots permettant d'assurer tous les transports.

Le temps de conditionnement de l'échantillon est rigoureusement le même qu'au 1er essai, soit 120 200 s. Augmenter le nombre de robot n'améliore en aucune façon les performances du système. L'analyse de ce résultat sera faite ultérieurement.

3ème Essai : Dimensionnement des bancs de transfert en entrée des machines M1 et M2.

Les tampons d'entrée des machines M1 et M2 sont dimensionnés à 2 emplacements gérés en FIFO. Les modifications à apporter au graphe sont les suivantes : le marquage des places T-IN1-EMPTY et T-IN2-EMPTY est initialisé à 2 et les places T-IN1-FULL et T-IN2-FULL sont déclarées de type FIFO. Le temps total de conditionnement est alors de 115 800 s soit un gain global de 3,7 %. par contre, si l'on surdimensionne encore les tampons d'entrée (3 places au lieu de 2), on n'améliore plus les performances du système.

4ème Essai : Dimensionnement des bancs de transfert en sortie des machines M1 et M2.

Le surdimensionnement des tampons de sortie n'améliore en aucune façon les performances globales.

V.3 - CONCLUSION

Les performances optimales sont obtenues pour la configuration minimale suivante :

- 1 seul robot pour assurer les divers transferts,
- Les tampons d'entrée des machines M1 et M2 dimensionnés à 2 emplacements gérés en FIFO.

VI - MODELISATION DU PROCEDE EN VUE DE L'ELABORATION DE STATISTIQUES DE FONCTIONNEMENT

VI.1 - Introduction

L'intérêt de définir un modèle décrivant le procédé est de permettre l'élaboration de statistiques de fonctionnement, non seulement sur les diverses machines (durée d'usinage, durée de transfert, ...) mais aussi sur les pièces (temps de conditionnement moyen, temps moyen d'attente, ...). Une telle approche est impossible par simple temporisation du graphe de commande puisque les pièces circulant dans la cellule ne sont pas identifiables individuellement.

Afin de permettre l'intégration du modèle décrivant le procédé, le graphe de commande initial (Fig. 15) a été modifié (Fig. 16). Les places d'interface correspondant aux couples (génération d'une commande, réception de fin d'action) ont été ajoutées sur le graphe. L'émulation de l'entrée est maintenant directement prise en compte par le modèle décrivant le procédé. Le processus "ENTREE" a été défini afin de permettre le lien entre la partie commande et le procédé au niveau de la détection des pièces présentes dans l'organe d'entrée.

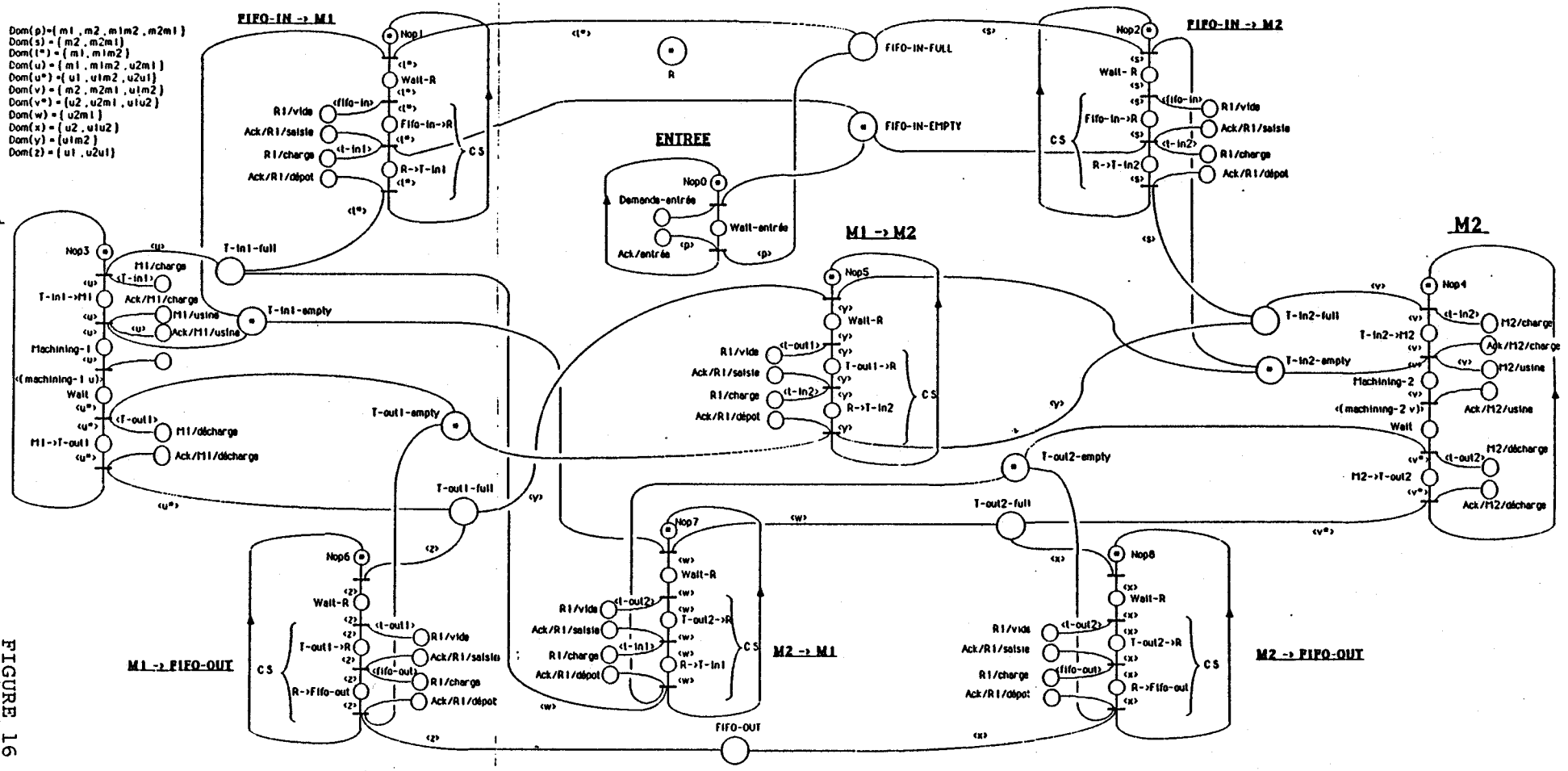
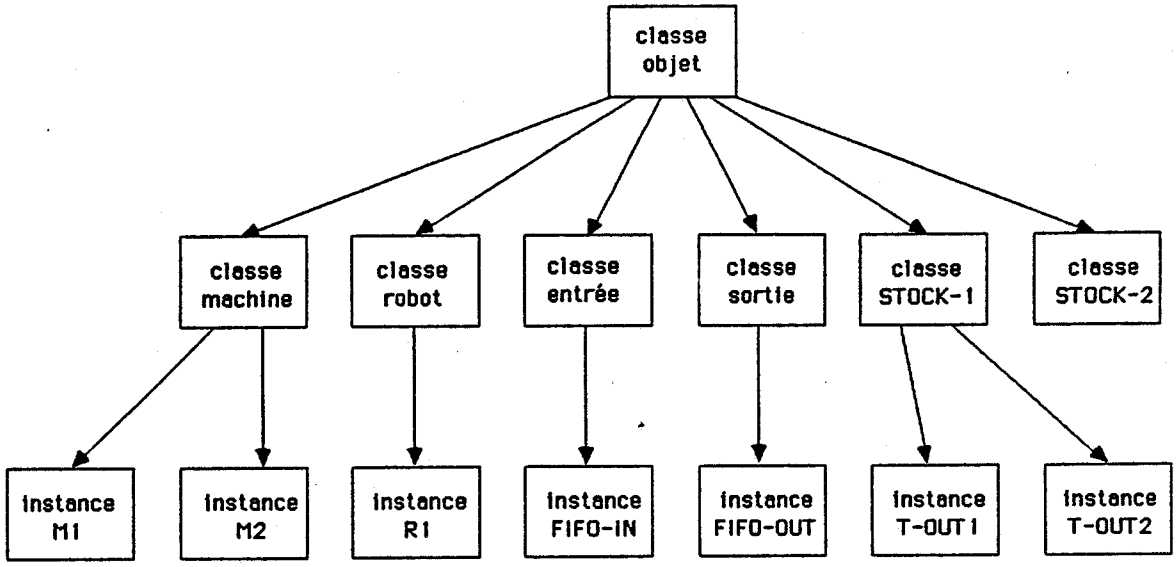

Dom(p)={ m1 , m2 , m1m2 , m2m1 }
Dom(s) = { m2 , m2m1}
Dom(t*) = { m1 , m1m2 }
Dom(u) = { m1 , m1m2 , u2m1 }
Dom(u*) ={ u1 , u1m2 , u2u1}
Dom(v) = { m2 , m2m1 , u1m2 }
Dom(v*) = {u2 , u2m1 , u1u2 }
Dom(w) = { u2m1 }
Dom(x) = { u2 , u1u2 }
Dom(y) = {u1m2 }
Dom(z) = { u1 , u2u1}
M1
FIFO-IN -> M1
Nop1
Wait-R
R1/vide
Ack/R1/saisie
R1/charge
Ack/R1/dépot
Fifo-in->R
R->T-in1
CS
R
FIFO-IN-FULL
FIFO-IN-EMPTY
ENTREE
Nop0
Demande-entrée
Ack/entrée
Wait-entrée
FIFO-IN -> M2
Nop2
Wait- R
Fifo-in->R
R->T-in2
M1 -> M2
M2
Nop3
T-in1-full
M1/charge
T-in1->M1
Ack/M1/charge
M1/usine
Ack/M1/usine
T-in1-empty
Machining-1
Wait
M1->T-out1
M1/décharge
Ack/M1/décharge
T-out1-empty
T-out1-full
NopS
T-out1->R
R->T-in2
T-in2-full
T-in2-empty
Nop4
T-in2->M2
M2/charge
Ack/M2/charge
M2/usine
Ack/M2/usine
Machining-2
M2->T-out2
M2/décharge
Ack/M2/décharge
T-out2-empty
T-out2-full
Nop6
M1 -> FIFO-OUT
Nop7
T-out2->R
R->T-in1
M2 -> M1
FIFO-OUT
Nop8
R->Fifo-out
M2 -> FIFO-OUT

FIGURE 16

VI.2 - Description des objets

L'arborescence des objets utiles est la suivante :



La classe "STOCK-1" caractérise des zones de stockages à 1 emplacement et la classe "STOCK-2" des zones à 2 emplacements gérés en FIFO. Les tables d'entrée "T-IN1" et "T-IN2" sont des instances de STOCK-1 ou de STOCK-2 selon les cas envisagés.

Les méthodes associées aux objets sont temporisées. La fonction permettant d'écrire un évènement à une date précise dans un échéancier s'appelle de la façon suivante :

(générer <date> <évènement> <échéancier>).

La classe générique "OBJET" possède les méthodes suivantes :

- CREER (<nom><p-liste>) permettant de créer une sous-classe ou une instance de l'objet récepteur du message de création dont les valeurs spécifiques des champs sont précisées sur la p-liste en argument.
- ⟵ (<objet><champ>) permettant de lire la valeur <champ> de l'<objet> précisé.
- ⟶ (<champ><valeur>) permettant d'affecter le <champ> de l'objet courant à <valeur>.

Rappelons (cf Chapitre I, § III.7.2.a) qu'un message a la syntaxe suivante :

(⟹ <objet> <méthode> [<argument 1> ... <argument n>])

Au niveau de la description des différents objets, nous adopterons la notation suivante :

La variable "moi" représente l'identificateur de l'objet (classe, sous-classe ou instance particulière) récepteur du message.

*** Classe MACHINE :**

Champs

est-un	objet	
occupation	()	
durée-usinage	0	⟶ durée d'un usinage (par défaut)
durée-transfert	100	⟶ durée d'un transfert (T-IN ⟶ P ou P ⟶ T-OUT)
temps-usinage	0	⟶ durées cumulées d'usinage
temps-transfert	0	⟶ durées cumulées de transfert
type-usinage	()	Les types d'usinage et les places d'interface dont précisés au niveau des instances
accusé-charge	()	
accusé-usinage	()	
accusé-décharge	()	
nombre-usinage	0	

Méthodes

```
Chargement (origine)
  (générer (+ temps-courant (⟵ moi 'durée-transfert))
           (⟹ moi 'fin-chargement origine)
           échéancier-po)

Fin-chargement (origine)
  (⟶ 'temps-transfert (+ (⟵ moi 'durée-transfert)
                         (⟵ moi 'temps-transfert)))
  (⟶ 'occupation (⟹ origine 'retrait))
  (ajouter 1 '* (⟵ moi 'accusé-charge))

Déchargement (destination)
  (générer (+ temps-courant (⟵ moi 'durée-transfert))
           (⟹ moi 'fin-déchargement destination)
           échéancier-po)

Fin-déchargement (destination)
  (⟶ 'temps-transfert (+ (⟵ moi 'durée-transfert)
                         (⟵ moi 'temps-transfert)))
  (⟹ destination 'dépôt (⟵ moi 'occupation))
  (⟶ 'occupation ())
  (ajouter 1 '* (⟵ moi 'accusé-décharge))

Usinage (type)
  (générer (+ temps-courant (⟵ moi 'durée-usinage))
           (⟹ moi 'fin-usinage type)
           échéancier-po)

Fin-usinage (type)
  (⟹ (⟵ moi 'occupation)
     'écrire
     'type
     (cassq type (⟵ moi 'type-usinage)))
  (⟶ 'temps-usinage (+ (⟵ moi 'durée-usinage)
                       (⟵ moi 'temps-usinage)))
  (ajouter 1 '* (⟵ moi 'accusé-usinage))
  (⟶ 'nombre-usinage (+ 1 (⟵ moi 'nombre-usinage)))
```

*** Instance M1 :**

est-un	machine
durée-usinage	3000
type-usinage	((m1 . u1)(m1m2 . u1m2)(u2m1 . u2u1))
accusé-charge	ack/M1/charge
accusé-décharge	ack/M1/décharge
accusé-usinage	ack/M1/usine

*** Instance M2 :**

est-un	machine
durée-usinage	2000
type-usinage	((m2 . u2)(m2m1 . u2m1)(u1m2 . u1u2))
accusé-charge	ack/M2/charge
accusé-décharge	ack/M2/décharge
accusé-usinage	ack/M2/usine

Nous ne commenterons pas ici de façon précise, tous les objets décrits et leurs instances, chose faite déjà par ailleurs (cf Chapitre I, § III.7.2, pour la description des robots et des stocks). Nous nous limiterons à la description de la classe "machine" en insistant tout d'abord sur la façon dont est réalisé le recueil de statistiques de fonctionnement.

Le champ "durée-usinage" mémorise les durées cumulées d'usinage pour chaque machine. Initialement nulle, sa valeur est incrémentée de la valeur du champ "durée-usinage" à chaque activation de la méthode "usinage". De même, les durées cumulées de transfert sont mémorisées dans le champ "durée-transfert" et le nombre cumulé d'usinages opérés dans le champ "nombre-usinage".

La méthode "chargement (origine)" et sa continuation temporisée "fin-chargement (origine)" décrivent l'opération de transfert entre le banc de transfert en entrée "T-IN" et le point d'usinage "P" de la machine considérée. Le passage effectif de la pièce transférée est réalisé par évaluation de :

```
(⟶ 'occupation (⟹ origine 'retrait))
                  ──────
                  T-IN1 ou T-IN2
```

De même, la méthode "déchargement (destination)" et sa continuation temporisée "fin-déchargement (destination)" décrivent le transfert de la pièce usinée depuis P jusqu'au banc de transfert en sortie "T-OUT1" ou "T-OUT2".

Les méthodes "usinage" et "fin-usinage" assurent la transformation du type de la pièce usinée :

```
(⟹ (⟵ moi 'occupation) 'écrire
    ──────────────────
       pièce usinée

'type (cassq type (⟵ moi 'type-usinage)))
      ──────────────────────────────────
          type de la pièce après usinage
```

Au niveau des instances M1 et M2 sont précisés les usinages que savent réaliser ces machines (champ "type-usinage") et leur durée (champ "durée-usinage"). La durée des transferts de "T-IN" vers "P" et de "P" vers "T-OUT" est précisée par défaut au niveau de la classe "machine".

*** Classe ROBOT :**

Champs

```
est-un              objet
position            repos
occupation          ()
durée-trajet        100   ⟶ durée d'un trajet
temps-occupation    0     ⟶ durée cumulée
                            d'occupation (initiale)

accusé-saisie       ()    ┐ Les places d'interface sont
accusé-dépôt        ()    ┘ précisées au niveau de
                            l'instance R1

nombre              0     ⟶ nombre de pièces transférées
```

Méthodes

```
Trajet-à-vide (destination)
  (⟶ 'position 'mouvement)
  (générer (+ temps-courant (⟵ moi 'durée-trajet))
           (⟹ moi 'arrivée-saisie destination)
           échéancier-po)

Arrivée-saisie (destination)
  (⟶ 'position destination)
  (⟶ 'temps-occupation
       (+ (⟵ moi 'temps-occupation)
          (⟵ moi 'durée-trajet)))
  (⟶ 'occupation (⟹ (⟵ moi 'position) 'retrait))
  (ajouter 1 '* (⟵ moi 'accusé-saisie))

Trajet-en-charge (destination)
  (⟶ 'position 'mouvement)
  (générer (+ temps-courant (⟵ moi 'durée-trajet))
           (⟹ moi 'arrivée-dépôt destination)
           échéancier-po)

Arrivée-dépôt (destination)
  (⟶ 'position destination)
  (⟶ 'temps-occupation
       (+ (⟵ moi 'temps-occupation)
          (⟵ moi 'durée-trajet)))
  (⟹ (⟵ moi 'position) 'dépôt (⟵ moi 'occupation))
  (⟶ 'occupation ())
  (ajouter 1 '* (⟵ moi 'accusé-dépôt))
  (⟶ 'nombre (+ 1 (⟵ moi 'nombre)))
```

*** Instance R1 :**

```
est-un            robot
accusé-saisie     ack/R1/saisie
accusé-dépôt      ack/R1/dépôt
```

La méthode "trajet-à-vide (destination)" et sa continuation temporisée "arrivée-saisie (destination)" décrivent le trajet à vide vers le lieu de chargement et l'opération de chargement. On définit de la même façon la méthode "trajet-en-charge (destination)" et sa continuation temporisée "arrivée-dépôt (destination)".

*** Classe PIECE :**

Champs

```
est-un       objet
type          ()
date-entrée    0
```

Les instances de pièces sont créées par l'objet "FIFO-IN".

*** Classe ENTREE :**

Champs

```
est-un         objet
occupation      ()
numéro           1
accusé-entrée   ()
durée            0    ⟶ cumul des durées d'attente
date-entrée      0
```

Méthodes

```
Retrait
  (⟶ 'durée (+ (⟵ moi 'durée)
                (- temps-courant (⟵ moi 'date-entrée))))
  (⟶ 'date-entrée 0)
  (progl (⟵ moi 'occupation)
         (⟶ 'occupation ()))

Demande-entrée
  (let ((type (entrée))
        (pièce (concat "p" (⟵ moi 'numéro))))
       (⟶ 'numéro (+ 1 (⟵ moi 'numéro)))
       (⟹ 'pièce 'créer pièce
            'date-entrée temps-courant
            'type        type)
       (⟶ 'occupation  pièce)
       (ajouter 1 type (⟵ moi 'accusé-entrée))
       (⟶ 'date-entrée temps-courant))
```

*** Instance FIFO-IN :**

```
est-un          entrée
accusé-entrée   ack/entrée
```

L'activation de la méthode "demande-entrée" provoque la création d'une nouvelle pièce. L'identificateur de cette pièce est de type "p_i" (p_1 pour la lère pièce, p_2 pour la seconde, ...). Le numéro d'ordre "i" est mémorisé dans le champs "numéro". Le type de la pièce est généré par la fonction "entrée".

*** Classe SORTIE :**

Champs

```
est-un       objet
occupation   ()
```

Méthode

```
Dépôt (pièce)
  (⟶ 'occupation (nconcl (⟵ moi 'occupation)
                         (list pièce
                               (⟹ pièce 'lire 'type)
                               (⟹ pièce 'lire 'date-entrée)
                               temps-courant)))
  (remob pièce)
```

Le champ "occupation" contiendra donc la liste ordonnée selon les dates de sortie de quadruplets :

(<identificateur de pièce>,<type>,<date d'entrée>,<date de sortie>).

Les pièces apparaissant dans un organe de sortie sont ensuite détruites physiquement (fonction "remob").

*** Instance FIFO-OUT :**

```
est-un      sortie
```

*** Classe STOCK-1 :**

Champs

```
est-un      objet
occupation   ()
nombre        0    ⟶ nombre de pièces stockées
                      (initialisé à 0)
durée         0    ⟶ durées cumulées de stockage
date-entrée   0    ⟶ date de début de stockage
                      de la pièce présente
```

Méthodes

```
Dépôt (pièce) (⟶ 'occupation pièce)
              (⟶ 'date-entrée temps-courant)

Retrait
  (⟶ 'durée (+ (⟵ moi 'durée)
               (- temps-courant (⟵ moi 'date-entrée))))
  (⟶ 'date-entrée 0)
  (⟶ 'nombre (+ 1 (⟵ moi 'nombre)))
  (progl (⟵ moi 'occupation)(⟶ 'occupation ()))
```

*** Instance T-OUT1 :**

```
est-un      stock-1
```

*** Instance T-OUT2 :**

```
est-un      stock-2
```

*** Classe STOCK-2 :**

Champs

```
est-un        objet
occupation     ()
nombre          0
durée           0
date-entrée     0
```

Méthodes

```
Dépôt (pièce)
  (⟶ 'occupation (appendl (⟵ moi 'occupation)
                           pièce))
  (⟶ 'date-entrée (appendl (⟵ moi 'date-entrée)
                           temps-courant))

Retrait
  (⟶ 'durée (+ (⟵ moi 'durée)
               (- temps-courant
               (car (⟵ moi 'date-entrée)))))
  (⟶ 'date-entrée (cdr (⟵ moi 'date-entrée)))
  (⟶ 'nombre (+ 1 (⟵ moi 'nombre)))
  (progl (car (⟵ moi 'occupation))
         (⟶ 'occupation (cdr (⟵ moi 'occupation))))
```

La classe "STOCK-2" décrit un lieu de stockage à 2 emplacements, géré en FIFO (First-In, First-Out). Le champ "occupation" mémorise la (ou les) pièce(s) présente(s), dans une liste de la forme :

(<identificateur de la 1ère pièce> <identificateur de la 2ème pièce>).

VI.3 - Première Simulation

La 1ère simulation correspond au 1er essai de la 2ème étape (cf § V.2). Le système dispose d'un unique robot et les bancs de transferts en entrée des machines (T-IN1 et T-IN2) sont dimensionnés à 1. Les objets "T-IN1" et "T-IN2" sont donc des instances de la classe "STOCK-1". On retrouve, bien sûr, le même temps total de conditionnement soit 120 200 s. Dans ces conditions, le temps d'engagement de la machine M1 est réparti en :

89,9 % pour l'usinage
8,5 % pour les transferts (entre T-IN1 et P d'une part, et entre P et T-OUT1 d'autre part)

⎫ Engagement net à 95,9 %

Pour la machine M2 :

60 % pour l'usinage
6 % pour les transferts

⎫ Engagement net à 66 %

Le robot a assuré les 122 transferts nécessaires en 24 400 s. Son rendement est seulement de 20,3 %.

Chaque pièce reste en moyenne 9 510 s dans le système :

8650 s pour les pièces de type m1

5614 s pour les pièces de type m2

13392 s pour les pièces de type m1m2

11510 s pour les pièces de type m2m1

Le temps de conditionnement d'une pièce est réparti en :

37,3 % d'usinage effectif

8,5 % de transfert

54,2 % d'attente

Le temps d'attente est réparti comme suit :

42,4 % dans l'organe d'entrée FIFO-IN

30,7 % en T-IN1

3,5 % en T-OUT1

17,7 % en T-IN2

5,7 % en T-OUT2

A titre indicatif, les résultats de simulation ayant aboutis à ces statistiques, sont fournis en Annexe B.

L'interprétation de ces statistiques explique les critères de dimensionnement optimaux adoptés. C'est la machine M1 dont le temps d'usinage est le plus long, qui impose son rythme au système. Comme le montrent les durées d'attente dans les organes d'entrée T-IN1 et T-IN2, un seul robot est suffisant pour assurer l'alimentation des machines. Le robot n'est alors employé qu'à 20 % de ses possibilités. Dans ces conditions, sous-employer un robot à 20 % de ses possibilités ou 2 robots à 10 % n'améliore en rien les performances du système. Ajouter un 2ème robot ne serait efficace que si le temps d'usinage était comparable aux temps de transferts.

Nous avons simulé un 2ème jeu d'essais répondant à ce critère. Dans ce cas, effectivement, l'ajout d'un robot améliore les performances de 35 % environ. Les points d'usinage des machines ne sont plus alors assimilés à des goulots d'étranglements. Les points d'usinage sont plus souvent inactifs car un seul robot ne peut pas assurer conjointement l'alimentation des tables d'entrée T-IN1 et T-IN2. Disposer de 2 robots assure une meilleure continuïté dans l'alimentation des pièces et, par voie de conséquence, minimise les temps d'attente entre 2 usinages consécutifs.

VI.4 - Deuxième Simulation

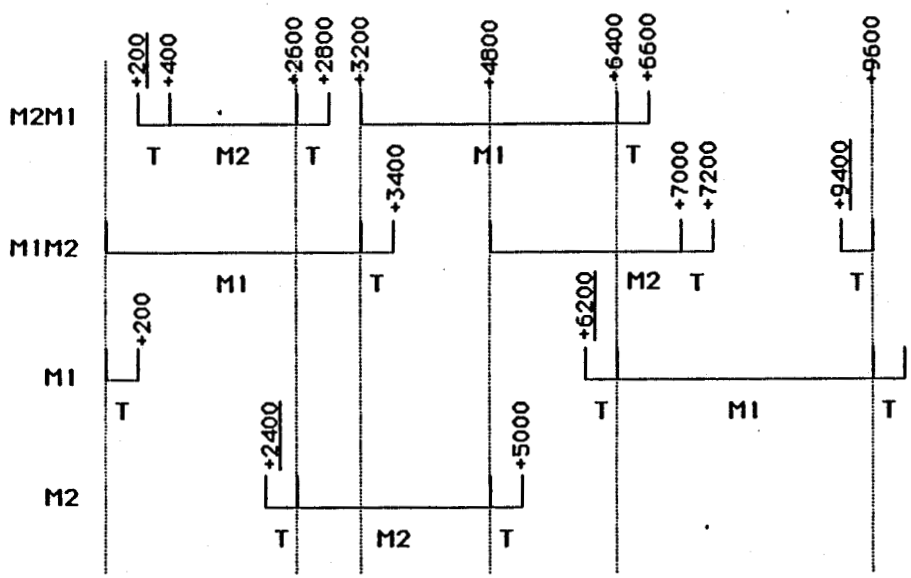
En regard des résultats obtenus par cette lère simulation, nous pouvons aisèment calculer, pour l'échantillon considéré, le temps minimum de conditionnement global. La machine M1 opère 36 usinages dont la durée unitaire est de 3200 s

(100 s de transfert T-IN1 ⟶ M1,
3000 s d'usinage et
100 s de transfert M1 ⟶ T-OUT1).

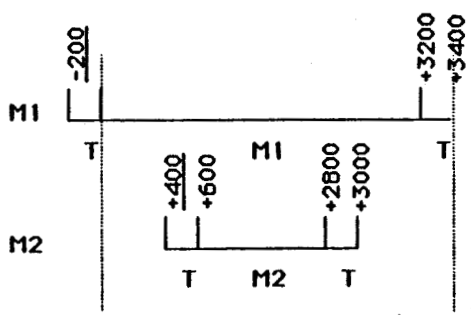
Il faut, de plus, 200 s pour alimenter initialement T-IN1 et 200 s pour le transfert de T-OUT1 vers FIFO-OUT de la dernière pièce. Le temps total minimum est donc de :

$$200 + (36 \times 3200) + 200 = 115\ 600 \text{ s}$$

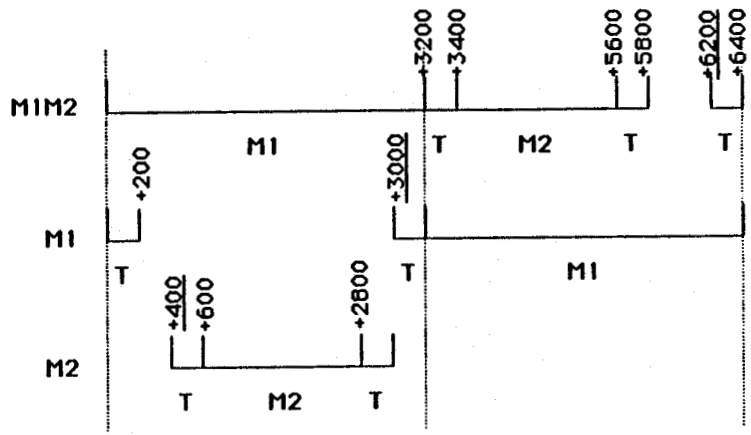
Le premier moyen d'atteindre cette durée minimale est d'organiser la séquence et la fréquence d'entrée afin d'obtenir un **ordonnancement** au sens des techniques de **gestion de production**, à la restriction près que l'ordonnancement requis concerne les pièces prises individuellement alors que la gestion de production classique repose sur le traitement par lots. Nous avons donc simulé un jeu d'essai reposant sur l'ordonnancement de la Figure 17. Le temps total de simulation est bien sûr de 115 600 s.



10 cycles



2 cycles



2 cycles

Afin de produire 12 pièces de type m1m2
10 pièces de type m2m1
14 pieces de type m2
14 pieces de type m2

nous avons scindé ce lot en 14 cycles de production.
Les 10 premiers cycles durent chacun 9600s, pendant lesquelles sont usinées une pièce de chaque type.
Les 2 cycles suivant de 6400 secondes chacun, conditionnent chacun une pièce m1m2, m1 et m2.
Les 2 derniers cycles de 3400 secondes conditionnent les pièces m1 et m2 restantes.
Chaque cycle a été conçu en vue d'une utilisation optimale de la machine M1.

Note: Les dates soulignées correspondent à la date d'entrée de la pièce considérée dans l'organe d'entrée FIFO-IN.

FIGURE 17

Cette solution, bien que donnant de bons résultats, nécessite une connaissance et une planification complètes et a priori du nombre et de la répartition des pièces à produire. Or, le concept de flexibilité suppose à l'inverse l'adaptation de la production aux besoins immédiats. La solution à adopter nécessite donc, dans notre cas, une optimisation des performances quelles que soient la séquence et la fréquence d'entrée des pièces. L'objectif à atteindre est d'approcher les 100 % pour le taux d'engagement de la machine M1, quelles que soient la séquence et la fréquence d'entrée.

La solution que nous avons adoptée consiste à surdimensionner les bancs de transfert en entrée des machines (T-IN1 et T-IN2). Outre les modifications mineures à apporter au graphe de commande (cf § V.2, 3ème Essai), les objets "T-IN1" et "T-IN2" sont maintenant des instances de la classe "STOCK-2". Le temps total de conditionnement est alors de 115 800 s et est donc très proche du temps minimal de 115 600 s.

La machine M1 usine pendant 93,2 % du temps (soit un gain de + 3,3 %) et assure les transferts pendant 6,2 % du temps (+ 0,2 %). Le taux d'engagement de M1 est donc de 99,4 % (+ 3,5 %).

Pour la machine M2 :

62,2 % pour l'usinage (+ 2 %)
6,2 % pour les transferts (+ 0,2 %)

] engagement net à 68,4 % (+ 2,4 %)

Le rendement du robot est alors de 21,1 % (+ 0,8 %).

Globalement, le gain est de 3,7 %.

Chaque pièce reste en moyenne 11 996 s dans le système, soit 26 % de plus :

12 279 s pour les pièces m1,
5 707 s pour les pièces m2,
15 950 s pour les pièces m1m2,
15 660 s pour les pièces m2m1.

Le temps de conditionnement d'une pièce est réparti en moyenne en :

30,5 % d'usinage,
7,1 % de transfert,
62,4 % d'attente.

Le temps d'attente est réparti comme suit :

29,0 % dans l'organe d'entrée FIFO-IN,
44,5 % en T-IN1,
0,9 % en T-OUT1,
20,1 % en T-IN2,
5,5 % en T-OUT2.

En comparant ces résultats à ceux obtenus lors de la première simulation, on constate que le temps d'attente dans les tables d'entrée des machines est supérieur de plus de 2 % alors que le pourcentage d'attente en sortie est réduit d'autant. Le dimensionnement adopté permet donc une meilleure alimentation (quasi-continue pour la machine M1) des 2 machines.

CONCLUSION

Divers essais portant sur le même échantillon de 50 pièces, organisé de manières différentes (séquences variées) ont mis en évidence une fluctuation maximale de + 0,8 % par rapport au temps global de conditionnement de l'échantillon complet. Le dimensionnement adopté et les règles de priorité définies autorisent une quasi-indépendance des performances du système par rapport à la séquence d'entrée.

Dans cet exemple, le logiciel de simulation nous a permis, tout d'abord, de vérifier le bien-fondé du système de commande conçu, c'est-à-dire de valider qualitativement la structure du graphe de commande et le comportement induit, et de vérifier l'adéquation des règles de surveillance définies pour résoudre l'indéterminisme du graphe.

Dans la phase de prédimensionnement du système, nous avons vu l'intérêt d'un outil permettant l'analyse quantitative des performances du système. Les statistiques de production obtenues ont permis, après analyse, une meilleure compréhension de la dynamique du système de commande. A ce niveau, les divers essais de simulation ont permis de répondre aux questions essentielles suivantes :

- Que faut-il modifier pour optimiser les performances ?
- Comment faut-il modifier ?
- Quelles sont les conséquences de ces modifications ?

CONCLUSION GENERALE

Nous avons présenté dans ce mémoire, un ensemble de modèles et d'outils homogènes adaptés à la description cohérente de tous les aspects d'un système de production flexible aussi bien en phase de conception qu'en phase d'évaluation.

Pour cela, nous utilisons 3 types de modèles, décrivant chacun l'un des aspects du système flexible considéré. La partie commande est modélisée par Réseaux De Petri Structurés Adaptatifs Colorés. La partie décisionnelle, dont le rôle est de paramétrer le graphe de commande est représentée par un ensemble de règles de production. Le procédé est décrit grâce à des catégories génériques multiniveaux. L'utilisation du langage déclaratif Le_Lisp est le dénominateur commun de ces différents modèles au niveau de leur implantation dans le cadre du logiciel de simulation présenté.

Notre objectif principal pour la réalisation de ce simulateur a été d'accroître son intéractivité afin d'assister véritablement la démarche de conception des systèmes de production flexible.

Dans ce sens, ce logiciel a été conçu de manière suffisamment générale pour que les évolutions futures puissent facilement être prises en compte. C'est le cas, par exemple, d'approches complémentaires visant à introduire de manière plus précise la problématique des fonctionnements dégradés. Leur intégration au modèle a constitué l'un des objectifs qui ont guidé la réalisation du logiciel de simulation. Par contre, l'étude de ces fonctionnements dégradés n'a pas été réalisée de façon assez systématique pour affirmer la suffisance des primitives définies (gel, forçage, interface, ...).

Dans le contexte plus général du projet développé au Laboratoire d'Informatique Industrielle de l'I.D.N., ce travail permettra de disposer d'un outil cohérent d'évaluation qualitative et quantitative à travers toutes les phases de spécifications d'un système de production flexible.

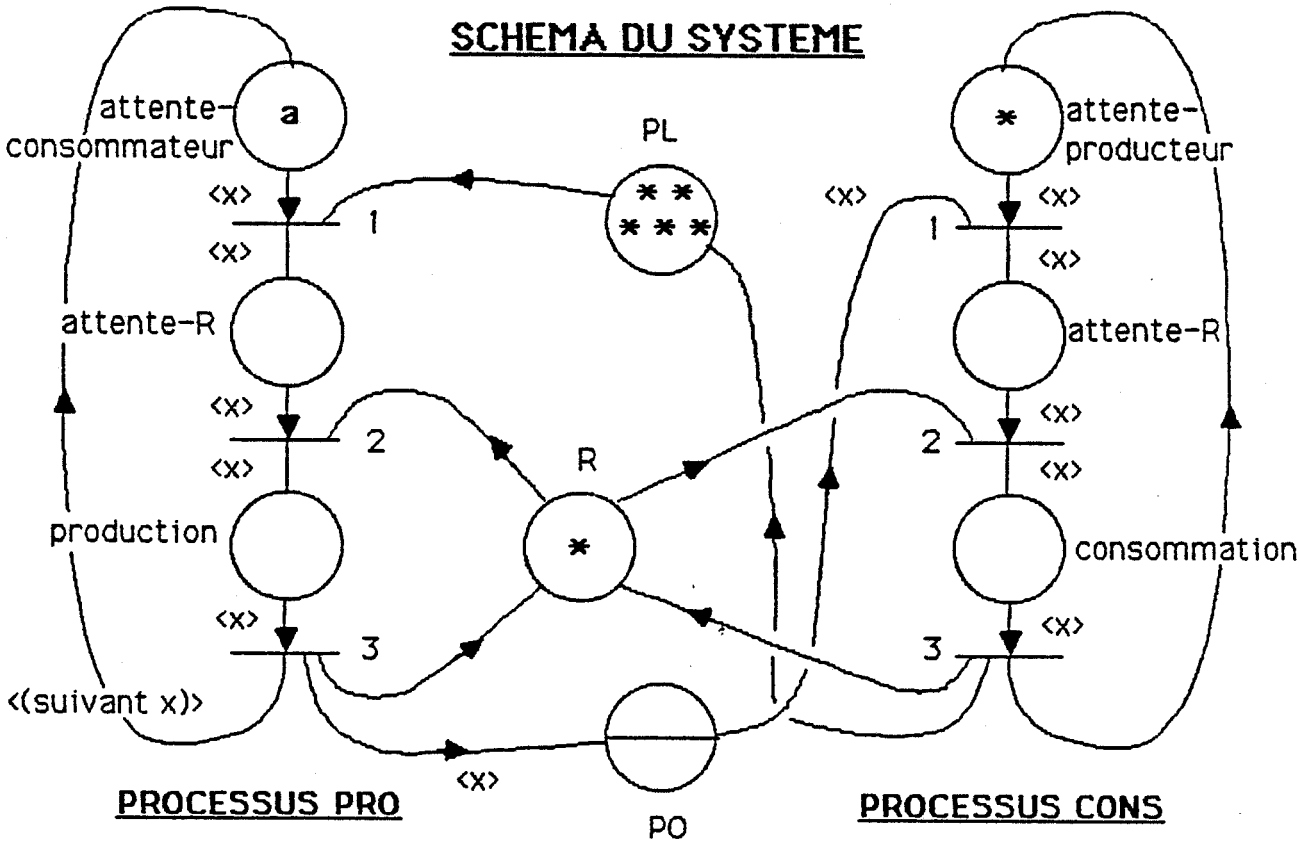
A un autre niveau, nous pensons qu'il serait intéressant d'utiliser le modèle original de description du procédé hors du contexte de la simulation. Considéré comme le miroir du système réel en phase d'implantation et d'exécution, il permettrait, par comparaison avec l'évolution du système réel de détecter les erreurs ou les défaillances des organes de commande. Un tel élargissement du mode de description du procédé, constituera vraisemblablement l'un des axes de nos recherches futures.

A N N E X E A

Un exemple d'édition et de simulation

Remarques : * Toutes les commandes et informations définies par l'utilisateur sont notées en caractère gras.

* Les commentaires sont écrits en caractère "Los-Angelès".



TEMPORISATION : INTERPRETATION DES TRANSITIONS

pro/2 -> evt-genere : (tempo x 'pro/production (duree-production x))
cons/2 -> evt-genere : (tempo x cons/consommation (duree-consommation x))

VARIABLE LIBRE GLOBALE

```
Variable : x
Dom-x  : (a b c)
```

FONCTIONS UTILISATEUR

```
(de suivant (x)
   (cond ((eq x 'a) 'b)
         ((eq x 'b) 'c)
         ((eq x 'c) 'a)))

(de duree-production (x)
   (cond ((eq x 'a) 10)
         ((eq x 'b) 20)
         ((eq x 'c) 30)))

(de duree-consommation (x)
   (cond ((eq x 'a) 40)
         ((eq x 'b) 30)
         ((eq x 'c) 20)))
```

REGLES DE SURVEILLANCE

```
REGLE degel-producteur/attente-r
  type : initialisation
  var-libre (x)
  dom(x) = (dom-gele 'pro/attente-r)
  enonce : (degeler x 'pro/attente-r)

REGLE degel-cons/attente-r
  type : initialisation
  var-libre (x)
  dom(x) = (dom-gele 'cons/attente-r)
  enonce : (degeler x 'cons/attente-r)

REGLE priorite
  var-libre (x y)
  type : surveillance
  dom(x) = (dom-actif 'pro/attente-r)
  dom(y) = (dom-actif 'cons/attente-r)
  enonce : (if (<= (duree-consommation y) (duree-production
             (geler x 'pro/attente-r)
             (geler y 'cons/attente-r))
```

EDITION DU SYSTEME

**********<COMMANDES POSSIBLES>**********
EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
FIN - (AFFICHAGE - SUITE)
****

COMMANDE > **EDITION**

Nom du systeme a editer : **PRO-CONS**

EDITION DU GRAPHE DE COMMANDE : 1
EDITION DE LA PARTIE DECISIONNELLE : 2
EDITION DES TEMPORISATIONS : 3
EDITION DE TIME-OUT : 4
EDITION DU PROCEDE : 5
RETOUR AU MENU : AUTRE

Votre choix : **1**

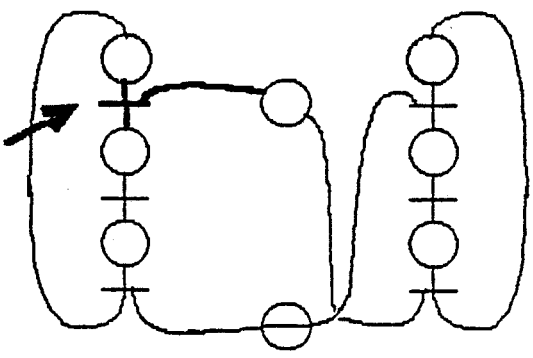
SAISIE DES VARIABLES LIBRES GLOBALES

Variable : **X**
Domaine (liste) : **(A B C)**

Variable : **<RC>**

SAISIE DES TRANSITIONS

IDENTIFICATEUR DE LA TRANSITION : **PRO/1**



VARIABLES LIBRES

Variable : **X**
Domaine (liste) : **<RC>**
Domaine (fonction) : **<RC>**
Variable libre globale

Variable : **<RC>**

ARCS EN AMONT

Poids de l'arc : **1**
Couleur concernee : **X**
Place concernee : **PRO/ATTENTE-CONSOMMATEUR**

Poids de l'arc : **1**
Couleur concernee : *****
Place concernee : **PL**

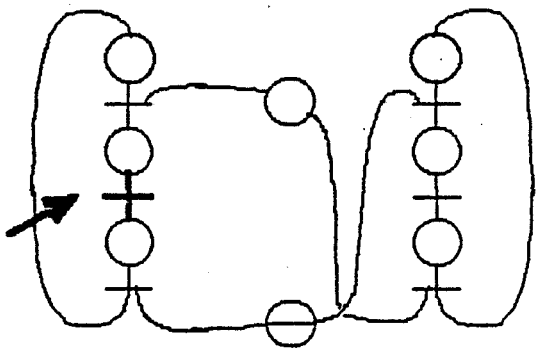
Poids de l'arc : **<RC>**

ARCS EN AVALS

Poids de l'arc : **1**
Couleur concernee : **X**
Place concernee : **PRO/ATTENTE-R**

Poids de l'arc : **<RC>**

IDENTIFICATEUR DE LA TRANSITION : **PRO/2**



VARIABLES LIBRES

Variable : **X**
Domaine (liste) : **<RC>**
Domaine (fonction) : **<RC>**
Variable libre globale

Variable : **<RC>**

ARCS EN AMONT

Poids de l'arc : **1**
Couleur concernee : **X**
Place concernee : **PRO/ATTENTE-R**

Poids de l'arc : **<RC>**

ARCS EN AVALS

Poids de l'arc : **1**
Couleur concernee : **X**
Place concernee : **PRO/PRODUCTION**

Poids de l'arc : **<RC>**

... SAISIE DES TRANSITIONS PRO/3, CONS/1, CONS/2, CONS/3...

IDENTIFICATEUR DE LA TRANSITION : **<RC>**

SAISIE DU TYPE DES PLACES (PC/PD/PO/FIFO)

Type de la place pro/attente-consommateur : **PC**
Type de la place pl : **<RC>**
Type de la place pro/attente-r : **PC**
Type de la place pro/production : **PC**
Type de la place po : **FIFO**
Type de la place cons/attente-producteur : **PC**
Type de la place cons/attente-r : **PC**
Type de la place cons/consommation : **PC**

PLACES OUBLIEES (PLACES NON CONNECTEES)

Nom de la place : **<RC>**

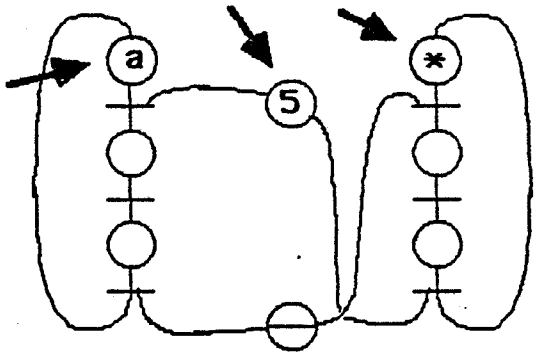
SAISIE DU MARQUAGE

Place : **PRO/ATTENTE-CONSOMMATEUR**
Couleur : **A**
Nombre : **1**

Place : **CONS /ATTENTE-PRODUCTEUR**
Couleur : *
Nombre : 1

Place : **PL**
Couleur : *
Nombre : 5

Place : **<RC>**



Saisie de fonctions utilisateurs (O/N) : **O**

Definition de la fonction
? **(DE SUIVANT (X)**
? **(COND ((EQ X 'A) 'B)**
? **((EQ X 'B) 'C)**
? **((EQ X 'C) 'A)))**

Definition de la fonction
? **FIN**

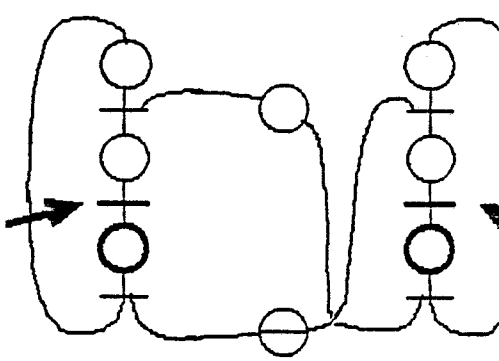
**********<COMMANDES POSSIBLES>**********
EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
FIN - (AFFICHAGE - SUITE)
****

COMMANDE > **EDITION**

Nom du systeme : pro-cons
Graphe de commande defini
Le graphe de commande possede des macros places FIFO

EDITION DU GRAPHE DE COMMANDE : 1
EDITION DE LA PARTIE DECISIONNELLE : 2
EDITION DES TEMPORISATIONS : 3
EDITION DE TIME-OUT : 4
EDITION DU PROCEDE : 5
RETOUR AU MENU : AUTRE

Votre choix : **3**



TRANSITION DECLENCHANT LA TEMPORISATION > **PRO/2**

VARIABLES LIBRES ASSOCIEES A LA TR ANSITION pro/2

Variable : X
Variable libre globale

Evenement genere : **(TEMPO X 'PRO/PRODUCTION (DUREE-PRODUCTION X))**

TRANSITION DECLENCHANT LA TEMPORISATION > **CONS/2**

VARIABLES LIBRES ASSOCIEES A LA TR ANSITION cons/2

Variable : X
Variable libre globale

Evenement genere : **(TEMPO X 'CONS/CONSOMMATION (DUREE-CONSOMMATION X))**

TRANSITION DECLENCHANT LA TEMPORISATION > **<RC>**

**********<COMMANDES POSSIBLES>**********
EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
FIN - (AFFICHAGE - SUITE)
****

COMMANDE > **CORRECTION**

**********<CORRECTIONS POSSIBLES>**********
TRANSITION - PLACE - VARIABLE - EVENEMENT - ECHEANCIER
FIFO - MUTEX - REGLE - FONCTION - FIN
****

CORRECTION > FONCTION

FONCTIONS DEJA DEFINIES :

(de suivant (x) (cond ((eq x 'a) 'b) ((eq x 'b) 'c) (eq x 'c) 'a)))
OK (o/n) : **<RC>**

Definition de la fonction
? **(DE DUREE-PRODUCTION (X)**
? **(COND ((EQ X 'A) 10)**
? **((EQ X 'B) 20)**
? **((EQ X 'C) 30)))**

Definition de la fonction
? **(DE DUREE- CONSOMMATION (X)**
? **(COND ((EQ X 'A) 40)**
? **((EQ X 'B) 30)**
? **((EQ X 'C) 20)))**

Definition de la fonction
? **FIN**

**********<CORRECTIONS POSSIBLES>**********
TRANSITION - PLACE - VARIABLE - EVENEMENT - ECHEANCIER
FIFO - MUTEX - REGLE - FONCTION - FIN
****

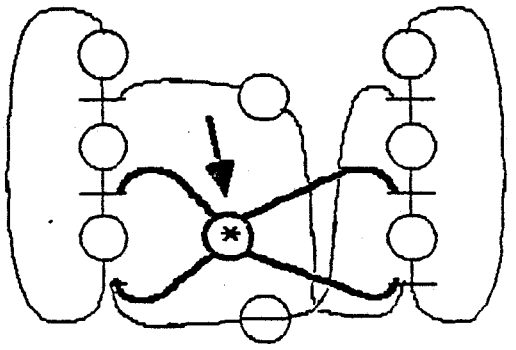
CORRECTION > **MUTEX**

MUTEX > **R**

ARCS EN ACCES A LA RESSOURCE r

Transition concernee : **PRO/2**
Poids de l'arc : 1
Couleur concernee : *

Transition concernee : **CONS/2**
Poids de l'arc : **1**



```
Couleur concernee   : *

Transition concernee : <RC>

ARCS RESTITUANT LA RESSOURCE r

Transition concernee : PRO/3
Poids de l'arc       : 1
Couleur concernee    : *

Transition concernee : CONS/3
Poids de l'arc       : 1
Couleur concernee    : *

Transition concernee : <RC>

MARQUAGE ACTIF DE LA RESSOURCE r

Couleur : *
Nombre : 1

Couleur : <RC>

MARQUAGE GELE DE LA RESSOURCE r

Couleur : <RC>

MUTEX > <RC>

**********<CORRECTIONS POSSIBLES>**********
  TRANSITION - PLACE - VARIABLE - EVENEMENT - ECHEANCIER
        FIFO - MUTEX - REGLE - FONCTION - FIN
**************************************************

CORRECTION > FIN

**********<COMMANDES POSSIBLES>**********
 EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
 SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
              FIN - ( AFFICHAGE - SUITE )
**************************************************

COMMANDE > SAUVEGARDE

NOM DU FICHIER : PRO-CONS

**********<COMMANDES POSSIBLES>**********
 EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
 SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
              FIN - ( AFFICHAGE - SUITE )
**************************************************

COMMANDE >
```

STRUCTURE EDITEE

```
(plist 'statut '(nom pro-cons pc t tempo t fifo t))
(setq places '(pro/attente-consommateur
               pl
               pro/attente-r
               pro/production
               po
               cons/attente-producteur
               cons/attente-r
               cons/consommation
               r))
(setq transitions '(pro/1 pro/2 pro/3 cons/1 cons/2 cons/3))
(setq var-libre '(x))
(setq regles '())
(setq regles-surveillance '())
(setq marquage-pc '(pro/attente-consommateur cons/attente-producteur))
(setq marquage-pd '())
(setq marquage-po '())
(setq echeancier-pc '())
(setq echeancier-pd '())
(setq echeancier-po '())
(setq temps-courant '0)
(setq fonctions-utilisateur '(suivant duree-production duree-consommation))
(plist 'var-libre '(dom-x '(a b c)))
(plist 'pro/attente-consommateur
       '(marquage-actif ((a . 1))
         type pc
         successeur (pro/1)))
(plist 'pl
       '(marquage-actif ((* . 5))
         successeur (pro/1)))
(plist 'pro/attente-r
       '(type pc
         successeur (pro/2)))
(plist 'pro/production
       '(type pc
         successeur (pro/3)))
(plist 'po
       '(type fifo
         successeur (cons/1)))
(plist 'cons/attente-producteur
       '(marquage-actif ((* . 1))
         type pc
         successeur (cons/1)))
(plist 'cons/attente-r
       '(type pc
         successeur (cons/2)))
(plist 'cons/consommation
       '(type pc
         successeur (cons/3)))
(plist 'r
       '(marquage-actif ((* . 1))
         successeur (pro/2 cons/2)))
```

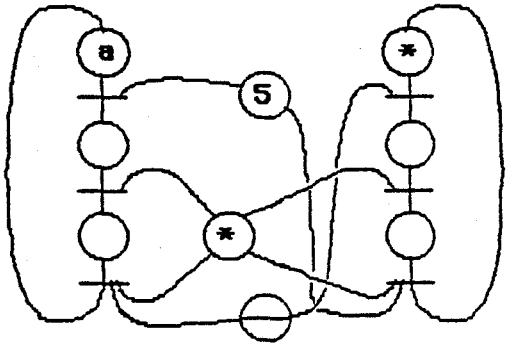
```
(plist 'pro/1
       '(arc-post ((1 x 'pro/attente-r))
         arc-pre ((1 x 'pro/attente-consommateur) (1 '* 'pl))
         var-libre (x)))
(plist 'pro/2
       '(evt-genere (tempo x 'pro/production (duree-production x))
         arc-post ((1 x 'pro/production))
         arc-pre ((1 x 'pro/attente-r) (1 '* 'r))
         var-libre (x)))
(plist 'pro/3
       '(arc-post ((1 (suivant x) 'pro/attente-consommateur) (1 x 'po) (1 '* 'r))
         arc-pre ((1 x 'pro/production))
         var-libre (x)))
(plist 'cons/1
       '(arc-post ((1 x 'cons/attente-r))
         arc-pre ((1 '* 'cons/attente-producteur) (1 x 'po))
         var-libre (x)))
(plist 'cons/2
       '(evt-genere (tempo x 'cons/consommation (duree-consommation x))
         arc-post ((1 x 'cons/consommation))
         arc-pre ((1 x 'cons/attente-r) (1 '* 'r))
         var-libre (x)))
(plist 'cons/3
       '(arc-post ((1 '* 'cons/attente-producteur) (1 '* 'pl) (1 '* 'r))
         arc-pre ((1 x 'cons/consommation))
         var-libre (x)))
(de suivant (x)
    (cond ((eq x 'a) 'b)
          ((eq x 'b) 'c)
          ((eq x 'c) 'a)))
(de duree-production (x)
    (cond ((eq x 'a) 10)
          ((eq x 'b) 20)
          ((eq x 'c) 30)))
(de duree-consommation (x)
    (cond ((eq x 'a) 40)
          ((eq x 'b) 30)
          ((eq x 'c) 20)))
```

CESSION DE SIMULATION

**********<COMMANDES POSSIBLES>**********
EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
FIN - (AFFICHAGE - SUITE)
****

COMMANDE > **SIMULATION**

Les marques gelées sont en majuscules et les marques actives en minuscules.



STATUT DU RESEAU

Nom du reseau : pro-cons
Graphe de commande defini
Le graphe est temporise
Le graphe de commande possede des macros places FIFO

Simulation partie commande (o/n) : **0**
Indeterminisme d'agregation permis (o/n) : **0**
Simulation temporisee (o/n) : **0**

CONFIGURATION DU SIMULATEUR

MARQUAGE INITIAL A LA DATE 0
De la partie commande : (pro/attente-consommateur cons/attente-producteur)
De la partie operative : ()
De la partie decisionnelle : ()

**********<COMMANDES POSSIBLES>**********
SUITE - FIN - AUTRE
****

INTERVENTION OPERATEUR > **SUITE**

Début du cycle par évènements à la date 0.

PARTIE COMMANDE A LA DATE 0
Transitions tirees : (pro/1) *cycle par activités*
Marquage obtenu : (pro/attente-r cons/attente-producteur)

PARTIE COMMANDE A LA DATE 0
Transitions tirees : (pro/2) *cycle par activités*
Marquage obtenu : (cons/attente-producteur)

Début du 2° cycle par évènements à la date 10.

PARTIE COMMANDE A LA DATE 10
Transitions tirees : (pro/3)
Marquage obtenu : (pro/attente-consommateur cons/attente-producteur)

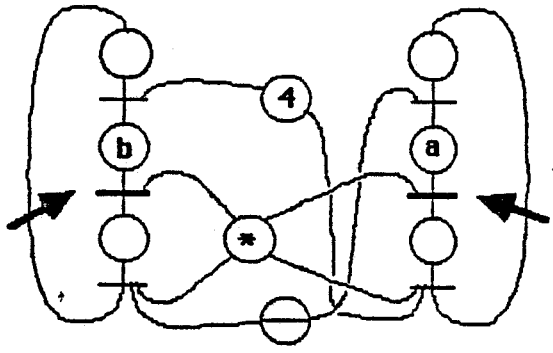
PARTIE COMMANDE A LA DATE 10
Transitions tirees : (cons/1 pro/1)
Marquage obtenu : (pro/attente-r cons/attente-r)

L'indéterminisme détecté résulte de l'absence de règles de priorité capables de résoudre l'indéterminisme d'accès à la ressource R. Cet indéterminisme est récupéré et analysé par la simulateur :

```
** ** : INDETERMINISME STRUCTUREL : ** ** A LA DATE 10
LORS DU TIR DE (cons/2 pro/2)
au niveau des marques
    de couleur * dans la place r

TIR DE LA TRANSITION cons/2
Pour les liaisons
x = a
      - ((1 a cons/attente-r) (1 * r))
      + ((1 a cons/consommation))

TIR DE LA TRANSITION pro/2
Pour les liaisons
x = b
      - ((1 b cons/attente-r) (1 * r))
      + ((1 b pro/production))
```




```
****************<COMMANDES POSSIBLES>***************
                  SUITE - FIN - AUTRE
***************************************************

INTERVENTION OPERATEUR > FIN

FIN DE SIMULATION A LA DATE 10
** ** : FIN DE SIMULATION : ** ** .

****************<COMMANDES POSSIBLES>***************
 EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
 SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
                FIN - ( AFFICHAGE - SUITE )
***************************************************

COMMANDE > EDITION

Nom du reseau : pro-cons
Graphe de commande defini
Le graphe est temporise
Le graphe de commande possede des macros places FIFO

EDITION DU GRAPHE DE COMMANDE    : 1
EDITION DE LA PARTIE DECISIONNELLE : 2
EDITION DES TEMPORISATIONS       : 3
EDITION DE TIME-OUT              : 4
EDITION DU PROCEDE               : 5
RETOUR AU MENU                   : AUTRE

Votre choix                      : 2
```

Définition des règles du niveau décisionnel capables de résoudre l'indétermisme détecté.

SAISIE DES VARIABLES LIBRES GLOBALES

Variable : **<RC>**

SAISIE DES REGLES DE SURVEILLANCE

IDENTIFICATEUR DE LA REGLE : **PRIORITE**
Regle d'initialisation (n/o) : **N**

VARIABLES LIBRES

Variable : **X**
Domaine (liste) : **<RC>**
Domaine (fonction) : **(DOM-ACTIF 'PRO/ATTENTE-R)**

Variable : **Y**
Domaine (liste) : **<RC>**
Domaine (fonction) : **(DOM-ACTIF 'CONS/ATTENTE-R)**

Variable : **<RC>**

ENONCE
```
(IF (<= (DUREE-CONSOMMATION Y) (DUREE PRODUCTION X))
    (GELER X 'PRO/ATTENTE-R)
    (GELER Y 'CONS/ATTENTE-R))
```

IDENTIFICATEUR DE LA REGLE : **DEGEL-PRO/ATTENTE-R**
Regle d'initialisation (n/o) : **O**

VARIABLES LIBRES

Variable : **X**
Domaine (liste) : **<RC>**
Domaine (fonction) : **(DOM-GELE 'PRO/ATTENTE-R)**

Variable : **<RC>**

ENONCE
```
(DEGELER X 'PRO/ATTENTE-R)
```

IDENTIFICATEUR DE LA REGLE : **DEGEL-CONS/ATTENTE-R**
Regle d'initialisation (n/o) : **O**

VARIABLES LIBRES

Variable : **X**
Domaine (liste) : **<RC>**
Domaine (fonction) : **(DOM-GELE 'CONS/ATTENTE-R)**

Variable : **<RC>**

ENONCE
```
(DEGELER X 'CONS/ATTENTE-R)
```

IDENTIFICATEUR DE LA REGLE : **<RC>**

SAISIE DES REGLES DE REQUETE

IDENTIFICATEUR DE LA REGLE : **<RC>**

**********<COMMANDES POSSIBLES>**********
EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
FIN - (AFFICHAGE - SUITE)
****

Reprise de la simulation après correction.

COMMANDE > **SIMULATION**

STATUT DU RESEAU

Nom du reseau : pro-cons
Graphe de commande defini
Partie decisionnelle definie
Le graphe est temporise
Le graphe de commande possede des macros places FIFO

Simulation partie commande (o/n) : **0**
Indeterminisme d'agregation permis (o/n) : **0**
Simulation partie decisionnelle (o/n) : **0**
Simulation temporisee (o/n) : **0**

CONFIGURATION DU SIMULATEUR

MARQUAGE INITIAL A LA DATE 10
De la partie commande : (pro/attente-r cons/attente-r)
De la partie operative : ()
De la partie decisionnelle : ()

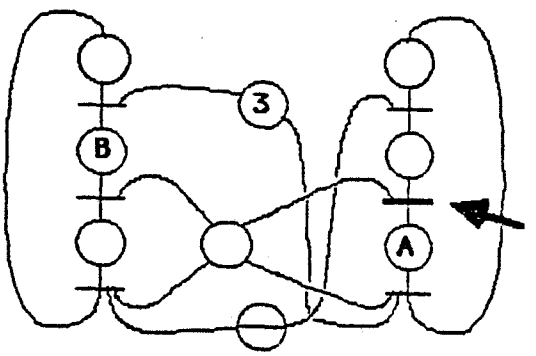
**********<COMMANDES POSSIBLES>**********
SUITE - FIN - AUTRE
****

INTERVENTION OPERATEUR > **SUITE**

PARTIE DECISIONNELLE A LA DATE 10
Regles activees : (priorite)
Marquage obtenu : ()

PARTIE COMMANDE A LA DATE 10
Transitions tirees : (pro/2)
Marquage obtenu : ()

PARTIE DECISIONNELLE A LA DATE 10
Regles activees : (degel-cons/attente-r)
Marquage obtenu : ()



<TOUCHE> *Interruption programmée 'Intervention Opérateur'*

**********<COMMANDES POSSIBLES>**********
SUITE - FIN - AUTRE
****

INTERVENTION OPERATEUR > **AUTRE**

```
**********<COMMANDES POSSIBLES>**********
 EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
 SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
            FIN - ( AFFICHAGE - SUITE )
************************************************
```

COMMANDE > **CORRECTION**

```
**********<CORRECTIONS POSSIBLES>**********
   TRANSITION - PLACE - VARIABLE - EVENEMENT - ECHEANCIER
        FIFO - MUTEX - REGLE - FONCTION - FIN
************************************************
```

CORRECTION > **ECHEANCIER**

Définition d'un point d'arrêt à la date 200.

ECHEANCIER (PC/PD/PO) : **PC**

EVENEMENTS DE L' ECHEANCIER DE LA PARTIE COMMANDE

Date : 30
Evenement : (degeler 'b 'pro/production)
OK (o/n) : **O**

Date : **200**
Evenement : **(POINT-ARRET)**

Date : **<RC>**

ECHEANCIER (PC/PD/PO) : **<RC>**

```
**********<CORRECTIONS POSSIBLES>**********
   TRANSITION - PLACE - VARIABLE - EVENEMENT - ECHEANCIER
        FIFO - MUTEX - REGLE - FONCTION - FIN
************************************************
```

CORRECTION > **FIN**

```
**********<COMMANDES POSSIBLES>**********
 EVALUATION - SIMULATION - EDITION - CORRECTION - LECTURE
 SAUVEGARDE - CHARGEMENT - PURGE - DESTRUCTION - STATUT
            FIN - ( AFFICHAGE - SUITE )
************************************************
```

COMMANDE > **SUITE**

INTERVENTION OPERATEUR > **SUITE**

PARTIE COMMANDE A LA DATE 30
Transitions tirees : (pro/3)
Marquage obtenu : (pro/attente-consommateur cons/attente-r)

... SIMULATION JUSQU'A LA DATE 200 ...

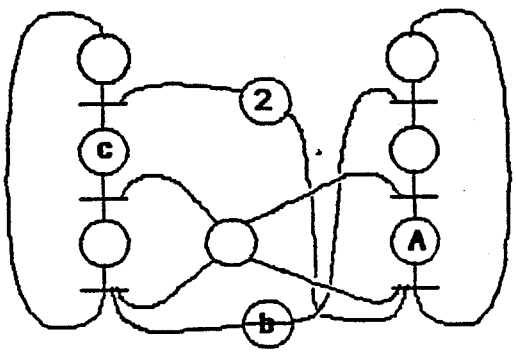
PARTIE COMMANDE A LA DATE 180
Transitions tirees : (cons/2 pro/1)
Marquage obtenu : (pro/attente-r)

** ** POINT D'ARRET ** ** A LA DATE 200

**********<COMMANDES POSSIBLES>**********
SUITE - FIN - AUTRE
****

INTERVENTION OPERATEUR > **FIN**

... RETOUR AU MENU GENERAL ...



COMPLEMENT SUR LA STRUCTURE EDITEE : PARTIE DECISIONNELLE

```
(plist 'statut '(nom pro-cons pc t pd t tempo t fifo t))
(setq regles '(priorite degel-pro/attente-r degel-cons/attente-r))
(setq regles-surveillance '(degel-cons/attente-r degel-pro/attente-r priorite))
(plist 'priorite
       '(type surveillance
         var-libre (x y)
         dom-x (dom-actif 'pro/attente-r)
         dom-y (dom-actif 'cons/attente-r)
         enonce (if (<= (duree-consommation y) (duree-production x))
                    (geler x 'pro/attente-r)
                    (geler y 'cons/attente-r))))
(plist 'degel-pro/attente-r
       '(type initialisation
         var-libre (x)
         dom-x (dom-gele 'pro/attente-r)
         enonce (degeler x 'pro/attente-r)))
(plist 'degel-cons/attente-r
       '(type initialisation
         var-libre (x)
         dom-x (dom-gele 'cons/attente-r)
         enonce (degeler x 'cons/attente-r)))
```

A N N E X E B

Recueil de statistiques
par les objets du procédé

1° SIMULATION :

Les tampons d'entrée "T-IN1" et "T-IN2" sont des instances de STOCK-1

```
(setq temps-courant '120200.)

(plist 'm1
       '(est-un machine
         duree-usinage 3000.
         type-usinage ((m1 . u1) (m1m2 . u1m2) (u2m1 . u2u1))
         occupation ()
         temps-usinage 108000.
         temps-transfert 7200.
         nombre-usinage 36.))

(plist 'm2
       '(est-un machine
         duree-usinage 2000.
         type-usinage ((m2 . u2) (m2m1 . u2m1) (u1m2 . u1u2))
         occupation ()
         temps-usinage 72000.
         temps-transfert 7200.
         nombre-usinage 36.))

(plist 'r1
       '(est-un robot
         position fifo-out
         occupation ()
         temps-occupation 244 00.
         nombre 122.))

(plist 'fifo-in
       '(est-un entree
         occupation ()
         numero 50
         duree 111500.
         date-entree 0))

(plist 'fifo-out
       '(est-un sortie
         occupation ((p2  u1   0.100E+03 0.380E+04)
                     (p3  u2   0.300E+03 0.480E+04)
                     (p4  u2   0.500E+03 0.700E+04)
                     (p1  u2u1 0.000E+00 0.720E+04)
                     (p6  u1   0.490E+04 0.102E+05)
                     (p7  u1   0.510E+04 0.134E+05)
                     (p5  u2u1 0.270E+04 0.166E+05)
                     (p10 u2   0.167E+05 0.194E+05)
                     (p8  u2u1 0.730E+04 0.198E+05)
                     (p11 u2   0.169E+05 0.216E+05)
                     (p12 u1   0.172E+05 0.262E+05)
                     (p9  u1u2 0.750E+04 0.299E+05)
                     (p15 u1u2 0.217E+05 0.321E+05)
                     (p13 u2u1 0.199E+05 0.326E+05)
                     (p14 u2u1 0.201E+05 0.358E+05)
```

```
(p16 u1u2 0.230E+05 0.414E+05)
(p17 u1   0.327E+05 0.422E+05)
(p20 u2   0.423E+05 0.450E+05)
(p18 u1   0.359E+05 0.454E+05)
(p19 u1   0.391E+05 0.486E+05)
(p23 u2   0.487E+05 0.514E+05)
(p21 u1   0.425E+05 0.518E+05)
(p22 u1   0.455E+05 0.550E+05)
(p24 u1u2 0.489E+05 0.606E+05)
(p25 u1u2 0.519E+05 0.638E+05)
(p26 u2u1 0.551E+05 0.646E+05)
(p27 u1u2 0.553E+05 0.702E+05)
(p28 u1   0.615E+05 0.710E+05)
(p29 u1   0.647E+05 0.742E+05)
(p30 u1   0.679E+05 0.774E+05)
(p33 u2   0.716E+05 0.782E+05)
(p31 u2u1 0.711E+05 0.806E+05)
(p32 u2u1 0.713E+05 0.838E+05)
(p34 u1u2 0.738E+05 0.894E+05)
(p35 u1   0.807E+05 0.902E+05)
(p37 u2   0.871E+05 0.916E+05)
(p36 u1   0.839E+05 0.934E+05)
(p40 u2   0.903E+05 0.960E+05)
(p41 u2   0.917E+05 0.982E+05)
(p38 u2u1 0.873E+05 0.998E+05)
(p42 u2   0.939E+05 0.100E+06)
(p39 u1u2 0.895E+05 0.103E+06)
(p44 u2   0.966E+05 0.105E+06)
(p46 u2   0.101E+06 0.107E+06)
(p43 u1u2 0.961E+05 0.109E+06)
(p45 u1u2 0.101E+06 0.111E+06)
(p47 u2   0.103E+06 0.114E+06)
(p48 u1u2 0.109E+06 0.118E+06)
(p50 u2u1 0.110E+06 0.119E+06)
(p49 u1u2 0.110E+06 0.120E+06))))
```



```
(plist 't-out1
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 10300.
         date-entree 0))

(plist 't-out2
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 15300.
         date-entree 0))

(plist 't-in1
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 92300.
         date-entree 0))
```

```
(plist 't-in2
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 47000.
         date-entree 0))
```

2° SIMULATION :

Les tampons d'entrée "T-IN1" et "T-IN2" sont des instances de STOCK-2

```
(setq temps-courant '115800.)

(plist 'm1
       '(est-un machine
         duree-usinage 3000.
         type-usinage ((m1 . u1)(m1m2 . u1m2)(u2m1 . u2u1))
         occupation ()
         temps-usinage 108000.
         temps-transfert 7200.
         nombre-usinage 36.))

(plist 'm2
       '(est-un machine
         duree-usinage 2000.
         type-usinage ((m2 . u2)(m2m1 . u2m1)(u1m2 . u1u2))
         occupation ()
         temps-usinage 72000.
         temps-transfert 7200.
         nombre-usinage 36.))

(plist 'r1
       '(est-un robot
         position fifo-out
         occupation ()
         temps-occupation 244 00.
         nombre 122.))

(plist 'fifo-in
       '(est-un entree
         occupation ()
         numero 50
         duree 103300.
         date-entree 0))

(plist 'fifo-out
       '(est-un sortie
         occupation ((p2  u1   0.100E+03 0.380E+04)
                     (p3  u2   0.300E+03 0.480E+04)
                     (p4  u2   0.500E+03 0.700E+04)
                     (p1  u2u1 0.000E+00 0.720E+04)
                     (p6  u1   0.270E+04 0.102E+05)
                     (p7  u1   0.290E+04 0.134E+05)
                     (p10 u2   0.730E+04 0.139E+05)
                     (p11 u2   0.750E+04 0.161E+05)
                     (p9  u1u2 0.490E+04 0.190E+05)
                     (p5  u2u1 0.700E+03 0.198E+05)
                     (p8  u2u1 0.390E+04 0.230E+05)
                     (p12 u1   0.920E+04 0.262E+05)
                     (p15 u1u2 0.171E+05 0.318E+05)
                     (p13 u2u1 0.167E+05 0.326E+05)
                     (p14 u2u1 0.169E+05 0.358E+05)
```

```
                (p16 u1u2 0.199E+05 0.414E+05)
                (p17 u1   0.295E+05 0.422E+05)
                (p20 u2   0.391E+05 0.436E+05)
                (p18 u1   0.327E+05 0.454E+05)
                (p23 u2   0.455E+05 0.482E+05)
                (p19 u1   0.359E+05 0.486E+05)
                (p21 u1   0.393E+05 0.518E+05)
                (p22 u1   0.423E+05 0.550E+05)
                (p24 u1u2 0.457E+05 0.606E+05)
                (p25 u1u2 0.487E+05 0.638E+05)
                (p26 u2u1 0.519E+05 0.646E+05)
                (p27 u1u2 0.521E+05 0.702E+05)
                (p28 u1   0.583E+05 0.710E+05)
                (p29 u1   0.615E+05 0.742E+05)
                (p30 u1   0.647E+05 0.774E+05)
                (p33 u2   0.683E+05 0.779E+05)
                (p34 u1u2 0.703E+05 0.830E+05)
                (p31 u2u1 0.679E+05 0.838E+05)
                (p37 u2   0.839E+05 0.866E+05)
                (p32 u2u1 0.681E+05 0.870E+05)
                (p35 u1   0.711E+05 0.902E+05)
                (p40 u2   0.871E+05 0.910E+05)
                (p41 u2   0.873E+05 0.932E+05)
                (p36 u1   0.807E+05 0.934E+05)
                (p42 u2   0.875E+05 0.954E+05)
                (p44 u2   0.935E+05 0.976E+05)
                (p39 u1u2 0.843E+05 0.998E+05)
                (p38 u2u1 0.841E+05 0.100E+06)
                (p46 u2   0.967E+05 0.102E+06)
                (p47 u2   0.969E+05 0.104E+06)
                (p43 u1u2 0.888E+05 0.106E+06)
                (p45 u1u2 0.937E+05 0.111E+06)
                (p48 u1u2 0.977E+05 0.113E+06)
                (p49 u1u2 0.100E+06 0.115E+06)
                 (p50 u2u1 0.103E+06 0.116E+06))))

(plist 't-out1
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 4000.
         date-entree 0))

(plist 't-out2
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 19000.
         date-entree 0))

(plist 't-in1
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 203800.
         date-entree 0))
```

```
(plist 't-in2
       '(est-un stock-1
         occupation ()
         nombre 36.
         duree 69000.
         date-entree 0))
```

B I B L I O G R A P H I E

INTRODUCTION A LA MODELISATION

/CHR 83/ CHRETIENNE P.
"Les réseaux de Petri temporisés"
Thèse d'Etat, Université Pierre & Marie Curie, Paris, 1983.

/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.

MODELISATION DE LA PARTIE COMMANDE

/BRA 83/ BRAMS G.W.
"Réseaux de Petri. Théorie et pratique"
Tome 1 : Théorie et analyse, Ed. Masson, 1983.

/TSI 85/ TSI Spécial RdP
Vol. 4, n° 1, Janvier 1985.

/COR 79/ CORBEEL D.
"Schéma de cablage et schéma de contrôle. Application à la simulation et à la gestion des processus industriels"
Thèse de Doct. de Spécialité, Lille, 1979.

/COR 80/ CORBEEL D.
"Formal description of processes systems and exception handling"
Mini & Micro, Budapest, Proc. pp. 335-339, 9-11 Septembre 1980.

/KAR 86/ KARFIA A., CORBEEL D., GENTINA J.C.
"Conception automatisée d'un système de conduite de procédés industriels"
Convention Automatique et Productique, Paris, Mai 1986.

/COR 81/ CORBEEL D., GENTINA J.C., VERCAUTER C.
"Méthodologie de description des systèmes de processus et de gestion d'erreurs"
IASTED Modelling, Identification and Control, Davos, 1981.

/HAC 75/ HACK M.
"Petri nets languages"
Computation Structure Group, Memo 124, Juin 1975.

/VAL 77/ VALK R.
"Self modifying nets"
Bericht 34, Pachberei Informatik, Hamburg, 1977.

/COR 84/ CORBEEL D., GENTINA J.C., VERCAUTER C.
"Adaptive Petri nets for real-time applications"
IMACS Digitech'84, Patras, 1984.

/GEN 79/ GENRICH H., LAUTENBACH K., THIAGARAJAN P.
"Elements of general net theory"
Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979.

/JEN 81/ JENSEN K.
"Coloured Petri nets and the invariant method"
Theorical Computer Science, n° 14, pp. 317-336, North Holland Pub. Comp., 1981.

/PET 80/ PETERSON J.L.
"A note on coloured Petri nets"
Information Processing Letters, Vol. 11, n° 1, pp. 40-43, Août 1980.

/COR 85/ CORBEEL D., GENTINA J.C., VERCAUTER C.
"Modélisation homogène du graphe de contrôle d'un système de conduite de processus industriels"
Congrès Automatique AFCET, Toulouse, pp. 517-534, 23-25 Octobre 1985.

MODELISATION DE LA PARTIE DECISIONNELLE

/BON 86/ BONNEMAY A.
"SYSTEL : un logiciel de 4ème génération pour la conduite d'installations industrielles"
Moyens d'Automatisation dans les Industries Manufacturières, Convention Automatique Productique, Paris, Mai 1986.

/BOU 86/ BOUREY J.P., CORBEEL D., CRAYE E., GENTINA J.C.
"Adaptive and coloured Petri nets for description, analysis and synthesis of hierarchical control and reliability of flexible cells in manufacturing systems"
1st European Workshop on Fault Diagnostic, Reliability and Related Knowledge-Based Approaches, Rhodes, Septembre 1986.

MODELISATION DU PROCEDE

/BRA 83/ BRAMS G.W.
"Réseaux de Petri. Théorie et pratique"
Tome 2 : Modélisation et Applications, Ed. Masson, Paris, 1983.

/ANA 86/ ANAKOK Y., PLAGNOL J.L., FRACHET J.P.
"Etude et implémentation des modes opératifs de systèmes de production automatisée"
Convention Automatique Productique, Proc. pp. 159-164, Paris, Mai 1986.

/CHR 83/ CHRETIENNE P.
"Les réseaux de Petri temporisés"
Thèse d'Etat, Université Pierre & Marie Curie, Paris, 1983.

/RAM 73/ RAMCHANDANI C.
"Analysis of asynchronous concurrent systems by timed Petri nets"
Ph. D. Thesis, M.I.T., Septembre 1973.

/SIF 77/ SIFAKIS J.
"Use of Petri nets for performance evolution"
Measuring Modelling and Evaluating Computer Systems, North Holland Pub. Comp., 1977.

/MOA 76/ MOALLA M.
"L'approche fonctionnelle dans la vérification des systèmes informatiques. Proposition d'un ensemble de méthodologies"
Thèse de Doct. Ing., INPG-ENSIMAG, Grenoble, 1976.

/GRA 77/ "Rapport final de la Commission AFCET : normalisation de la représentation du cahier des charges d'un automatisme logique"
Revue Automatique et Informatique Industrielle, n° 61-62, Novembre-Décembre 1977.

/CAR 83/ CARRIERE B., CAZALOT C., DUMAS J.M., GROSJEAN P.M., LEROY P., PRUNET F.
"Un système de C.A.O. pour la commande de processus reposant sur un standard"
IFIP, 1983.

/DEF 86/ DEFRENNE J., TOULOTTE J.M., HACHEMANI R.
"Validation des logiciels de commande des systèmes industriels à évolutions simultanées"
Convention Automatique Productique, Paris, Proc. pp. 154-158, Mai 1986.

/BRA 86/ BRAD J. COX
"Object oriented programming. An evolutionary approach"
Addison-Wesley Publishing Company, 1986.

/HUL 84/ HULLOT J.M.
"Programmer en CEYX"
Rapport de l'INRIA, 1984.

CONCLUSION MODELISATION

/GIR 84/ GIROD C.
"Conception et réalisation d'un logiciel de simulation de réseaux de Petri"
Thèse de Doct. Ing., I.D.N., Septembre 1984.

/ALA 84/ ALANCHE P., BENZAKOUR K., DOLLE F., GILLET P., RODRIGUE P., VALETTE R.
"PSI : a Petri net based simulator for flexible manufacturing systems"
5ème Workshop sur les RdP, Aarhus, Juin 1984.

DE LA VALIDATION A LA SIMULATION

/BER 79/ BERTHOMIEU B.
"Analyse structurelle des réseaux de Petri. Méthodes et outils"
Thèse de Doct. Ing., Toulouse, 1979.

/MEM 80/ MEMMI G., ROUCAIROL G.
"Linear algebra in net theory"
Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, Springer 1980.

/BER 83/ BERTHELOT G.
"Transformations et analyse de réseaux de Petri. Applications aux protocoles"
Thèse d'Etat, Paris, 1983.

/VER 82/ VERCAUTER C.
"Sur un ensemble d'outils d'aide à la spécification et à la conception des systèmes industriels"
Thèse de Doct. 3ème Cycle, Lille, 1982.

/ALA 84/ ALANCHE P., BENZAKOUR K., DOLLE F., GILLET P., RODRIGUE P., VALETTE R.
"PSI : a Petri net based simulator for flexible manufacturing systems"
5ème Workshop sur les RdP, Aarhus, Juin 1984.

/GIR 84/ GIROD C.
"Conception et réalisation d'un logiciel de simulation de Réseaux de Petri"
Thèse de Doct. Ing., I.D.N., Septembre 1984.

/GUI 85/ GUIDEZ G.
"Le Grafcet : macro-représentation et structuration du traitement des automatismes"
Confèrence AUTOMATION'85, Paris, Avril 1985.

PRESENTATION GENERALE DU SIMULATEUR

/GEN 79/ GENRICH H., LAUTENBACH K., THIAGARAJAN P.
"Predicate / transition nets"
Elements of General Net Theory, pp. 76-92.

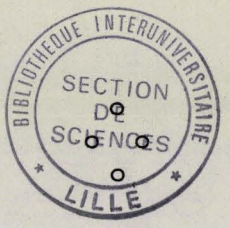
/CAS 85/ CASTELAIN E., CORBEEL D., GENTINA J.C.
"Interpréteur et analyse de processus modélisés par RdP"
IASTED MIC'85, Grindelwald, Février 1985.

CYCLE DE SIMULATION PAR EVENEMENT

/BRA 83/ BRAMS G.W.
"Réseaux de Petri. Théorie et pratique"
Tome 1 : Théorie et Analyse, Ed. Masson, 1983.

INTRODUCTION A L'EXEMPLE

/BOU 86/ BOUREY J.P., CORBEEL D., CRAYE E., GENTINA J.C.
"Adaptive and coloured Petri nets for description, analysis and synthesis of hierarchical control and reliability of flexible cells in manufacturing systems"
1st European Workshop on Fault Diagnostic, Reliability and Related Knowledge-Based Approaches, Rhodes, Septembre 1986.



RESUME

Nous présentons dans ce mémoire, un ensemble de modèles et d'outils homogènes adaptés à la description cohérente de tous les aspects d'un système de production flexible. Leur mise en œuvre est abordée dans le cadre d'un logiciel de simulation interactif dans le but d'assister la démarche de conception et de permettre l'évaluation des performances du système conçu.

Pour cela, nous utilisons 3 types de modèles, décrivant chacun l'un des aspects du système flexible considéré. La partie commande est modélisée par Réseaux de Petri Structurés Adaptatifs Colorés. La partie décisionnelle, dont le rôle est de paramétrer le graphe de commande, est représentée par un ensemble de règles de production. Le procédé est décrit par des catégories génériques multiniveaux. L'utilisation du langage déclaratif Le_Lisp est le dénominateur commun de ces différents modèles au niveau de leur implantation dans le cadre du logiciel de simulation présenté.

Un exemple industriel illustre l'approche proposée.

MOTS-CLEFS :

FLEXIBILITE DE PRODUCTION
MODELISATION
SIMULATION
PARTIE COMMANDE
PARTIE DECISIONNELLE
LANGAGE LE_LISP
RESEAUX DE PETRI
LANGAGE OBJET