50376
1993
131

N° d'ordre : 1141

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

Pour obtenir le titre de

DOCTEUR

en Productique : Automatique et Informatique Industrielle

par

Morad SOUAM
Ingénieur E.N.P

CONTRIBUTION A LA SPECIFICATION ET A LA MISE EN OEUVRE DE LA COMMANDE TEMPS-REEL DE CELLULES FLEXIBLES D'ASSEMBLAGE

Soutenue le 29 juin 1993 devant la commission d'examen :

MM.

P. Vidal	Président	Professeur à l'USTL
C. Vasseur	Directeur de recherche	Professeur à l'USTL
A. Lebrun	Rapporteur	Professeur à L'Université de Picardie
S. Tarasiewicz	Rapporteur	Professeur à l'Université de Laval
S. Maouche	Examinateur	Maître de conférences à l'USTL

à Mamiche

à mes chers Parents

à Ania

Remerciements

Le travail présenté dans ce mémoire a été réalisé au centre d'automatique de l'université de Lille I sous la direction de Monsieur le Professeur Christian VASSEUR.

Je tiens à présenter mes sincères remerciements à Monsieur le Professeur Pierre VIDAL pour m'avoir accepté dans son laboratoire et pour l'honneur qu'il me fait en présidant ce jury.

Que Monsieur le Professeur Christian VASSEUR, directeur de l'ENSAIT de Roubaix, trouve ici le témoignage de ma gratitude pour ces conseils et l'aide qu'il m'a apportée.

C'est un plaisir pour moi d'adresser mes chaleureux remerciements à Monsieur le professeur Stanislas TARASIEWICZ Professeur à l'Université de Laval (Canada) pour avoir bien voulu être le rapporteur de cette thèse et pour l'honneur qu'il me fait en participant à ce jury.

Je remercie tout aussi chaleureusement Monsieur André LEBRUN, Professeur à l'université de Picardie et directeur de l'INSSET de Saint-Quentin, pour l'intérêt qu'il a porté à ce travail et pour avoir accepter d'en être le rapporteur.

Je remercie également Monsieur Salah MAOUCHE, maître de conférences à l'IUT de Villeneuve d'Ascq pour avoir accepté de prendre part à ce jury.

Que ma famille et mes proches trouvent ici le témoignage de ma sincère reconnaissance pour le soutien qu'ils m'ont porté.

Je ne saurais clore ce volet sans penser à mon frère et ami Mamiche que dieu rappela auprès de lui, alors qu'il était précisément en phase de soutenir sa thèse. Que dieu ait pitié de son âme.

INTRODUCTION GENERALE.

Dans bon nombre de domaines de la technologie de fabrication, la part des coûts attachés aux procédés d'assemblage représente entre 20°/o et 50°/o du coût total de production avec une tendance à l'accroissement de ces pourcentages. L'assemblage constitue par conséquent le domaine de l'industrie manufacturière qui présente les potentialités les plus grandes pour la rationalisation (organisation, automatisation, gestion intégrée) de la production.

Si, actuellement, la fabrication des composants est largement automatisée, à l'inverse, l'assemblage reste essentiellement manuel. Lorsque l'automatisation existe, elle est limitée à quelques productions répétitives de très grandes séries réalisées par des machines spécialisées. En effet, dans le cas général, le coût d'investissement dans de telles machines est trop important par rapport au niveau de la production (moyennes et petites séries) et aux fluctuations du marché (débit maximal non garanti, durée de vie du produit incertaine). De plus, les coûts et délais d'étude puis de réalisation sont élevés, alors que la prise en compte initiale de nombreuses variantes du produit à assembler s'avère difficile et que les évolutions ultérieures du produit sont pratiquement interdites.

C'est pourquoi il convient maintenant de mettre au point des systèmes flexibles capables de réaliser, dans des conditions économiques, l'assemblage de productions faiblement répétitives.

Hier, les machines spéciales à grande capacité de production étaient adaptées à des marchés stables et à des productions mono-produit. Les productions moyennes étaient automatisées partiellement, souvent sans tenir compte des flux amont et aval.

Aujourd'hui, la notion de mono-produit est bouleversée par les effets de modes, qui contribuent à l'explosion de familles multi-produits et à la création de variantes. Ces nouveaux produits divisent les flux et demandent aux systèmes de production une adaptabilité aux changements qui s'exprime par le terme de flexibilité.

L'outil privilégié des systèmes flexibles d'assemblage est le robot industriel qui, par programmation, peut s'adapter à des modifications fréquentes de la production.

Parmi les freins au développement de systèmes flexibles d'assemblage, il y a évidemment ceux qui relèvent d'aspects financiers (investissement) ou sociaux (emploi). Mais un frein très fort se situe également au niveau des problèmes méthodologiques posés par la conception et la conduite de ce nouveau type de systèmes de production. Dans ce cas, la solution ne consiste pas en une adaptation triviale de solutions connues dans des domaines où l'automatisation est plus avancée, comme celui des ateliers d'usinage ou de parachèvement.

L'automatisation flexible d'ateliers d'assemblage implique la mise en oeuvre intégrée de plusieurs technologies avancées de base, telles que robots manipulateurs, vision artificielle, réseaux locaux, XAO, intelligence artificielle, que l'on peut qualifier de technologies diffusantes, car elles concernent transversalement de larges domaines industriels.

La conduite en temps-réel de cellules flexibles d'assemblage (lieu où les tâches d'assemblage sont effectivement réalisées) consiste à gérer en temps-réel les ressources de la cellule (robots, outils du magasin, espace, etc.) de façon à exécuter le plus efficacement possible le plan de fabrication élaboré par la partie gestion de la coordination globale de l'atelier. Le cas des ateliers d'assemblage sera en général plus complexe que celui des ateliers d'usinage, car les gammes d'assemblage laissent plus de degrés de liberté que celles d'usinage (ordres des opérations élémentaires) et posent de nombreux problèmes de synchronisation. Il faut, en effet, gérer des assemblages partiels de telle façon que les sous ensembles soient prêts simultanément pour être assemblés, sans stockage au niveau des cellules.

Cette conduite en temps-réel est un sujet relativement neuf en ce sens que rares sont les ateliers de fabrication flexibles complètement intégrés (depuis l'ordonnancement jusqu'à la commande locale), où la conduite se fait automatiquement et sans intervention humaine. En fait, il semble que deux écoles se détachent. L'une est influencée par l'utilisation de l'intelligence artificielle pour la planification et l'ordonnancement d'ateliers, l'autre cherche à étendre les méthodes de programmation élaborées pour la commande séquentielle au niveau local (utilisation du Grafcet et des Réseaux de Petri).

CHAPITRE I:

LA COMMANDE TEMPS-REEL

DANS LES ATELIERS FLEXIBLES

LA COMMANDE TEMPS-REEL

DANS LES ATELIERS FLEXIBLES

INTRODUCTION.

Après avoir été longtemps occultée par les secteurs de l'informatique (gestion, bureautique..), l'informatique temps-réel connaît actuellement, et depuis plusieurs années, un développement considérable.

En effet, les applications temps-réel sont de plus en plus présentes dans tous les grands secteurs industriels (transport, industries manufacturières, militaire...) et dans tous les grands projets d'envergure nationale (programme Hermès, centre de contrôle de satellite..).

Par ailleurs, l'ampleur de la problématique temps-réel ainsi que la diversité des domaines touchés font émerger de multiples questions sur les outils et les méthodes de spécification, de conception et de mise en oeuvre des systèmes temps-réel (STR).

Dans ce contexte, le premier chapitre expose, dans un premier temps, la problématique du temps-réel, en faisant apparaître une démarche universelle.

Ensuite il fait apparaître les différentes phases dans la vie d'un système temps-réel en mettant l'accent sur les qualités requises des outils et méthodes associés à ces diverses phases.

Enfin il précise les démarches puis les outils adaptés à la spécification de la commande en temps-réel des cellules flexibles d'assemblage.

I PROBLEMATIQUE DU TEMPS-REEL [ELL90][TEM88]

I-1 Application temps-réel.

Un système temps-réel est toute application mettant en oeuvre un système informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'état du procédé qui lui est connecté et dont il doit contrôler l'évolution. Ainsi le système informatique de contrôle doit être à l'écoute de tout changement d'état survenant au sein même du procédé ou dans son environnement. Les changements d'états se traduisent par des **événements asynchrones de l'évolution du système de contrôle** (figure 1.1).

Sous cet aspect, les systèmes temps-réel présentent les caractéristiques des systèmes pilotés par les signaux (occurrences d'événements) en provenance de l'environnement : les **systèmes réactifs** (aspect événementiel) [HAT85].

Le système de contrôle, en fonction des signaux émis ou reçus du procédé, doit évaluer les décisions à prendre et générer les actions appropriées afin de faire tendre l'évolution du procédé vers l'accomplissement de l'objectif souhaité. Ces diverses opérations nécessitent souvent des échanges riches en informations (archivage en bases de données, dialogue avec l'opérateur,...). Sous cet aspect propre aux systèmes pilotés par les données, les systèmes temps-réel présentent donc les caractéristiques des **systèmes transformationnels** (aspect fonctionnel) [HAT85].

Outre les nécessités de synchronisation, de communication et de partage de ressources relatifs au parallélisme et à la coopération, les instants de terminaison des actions déclenchées par le système de contrôle sont soumis à **des contraintes temporelles** imposées par le procédé.

La problématique du temps-réel nécessite la considération du temps non pas comme un facteur de performances, mais comme une contrainte logique de base. En effet le rôle du système de contrôle consiste à suivre l'état du procédé et à réagir à toute évolution significative du procédé, dans un laps de temps (temps de réaction) qui garantisse que le procédé reste totalement contrôlé par le système de contrôle.

Ces contraintes temporelles se traduisent alors par les **temps de réaction** de la part du système de contrôle relativement à la dynamique du procédé. On distingue deux types de contraintes temporelles:

- les **contraintes comportementales** que doit respecter l'environnement pour que le système évolue correctement, par exemple la fréquence maximale d'occurrence des événements externes.

- les **contraintes de performances** relatives à la dynamique interne du système commandé. C'est le cas notamment lorsque le non respect des échéances fixées peut amener le procédé dans un état jugé non contrôlable ou non tolérable.

Enfin un système temps-réel peut être **faiblement contraint** ou **fortement contraint.** La contrainte faible correspond au cas où le manquement d'une échéance entraîne une violation de spécification. Par contre, en contrainte forte, le manquement d'échéance est synonyme de panne, voire d'accident.

Lorsque le contrôle consiste en le **pilotage du procédé**, le rôle du système consiste en l'émission de commandes vers les actionneurs du procédé (Système de Contrôle Commande SCC). En revanche, lorsque le contrôle consiste en un **suivi du procédé**, le rôle du système de contrôle consiste en l'enregistrement et la signalisation des évolutions de son état.

Dans les industries manufacturières et les industries de production d'énergie le terme procédé peut désigner aussi bien un équipement isolé, qu'un ensemble de machines constituant un atelier ou qu'un ensemble de constituants d'un procédé continu (centrale thermique, unité de production chimique,...).

Quant au système informatique, il peut être constitué soit d'une machine unique (calculateur, automate), soit d'un ensemble d'équipements tels que des calculateurs reliés en réseau. C'est notamment le cas lorsque:

- le procédé est lui même constitué d'un ensemble d'équipements.

- les contraintes de temps imposent des exécutions d'actions en parallèle sur différentes machines.

- les contraintes de sûreté imposent la redondance de certains équipements de contrôle.

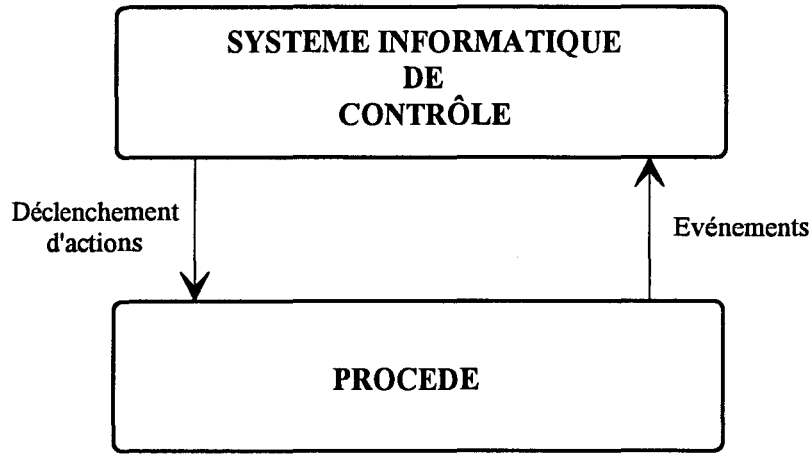


figure 1.1

I-2 Parallélisme et synchronisation.

L'expression du parallélisme traduit une double réalité:

- nécessité de décomposer une application complexe en processus parallèles moins complexes.
- nécessité de traiter simultanément plusieurs événements en provenance du procédé.

Lors de la phase d'implantation, le parallélisme se traduit sous deux formes:

- le **parallélisme vrai** sur un certain nombre fixé de processeurs réels.
- le **pseudo-parallélisme** lorsque l'exécution de plusieurs activités est réalisée sur un processeur réel unique, en respectant toutefois, pour chaque activité, les échéances de temps fixées par le cahier des charges.

Ce parallélisme ne doit pas cacher les temps de réaction exigés par l'application. Ces problèmes temporels s'ajoutent donc à ceux induits par l'algorithmique parallèle tels que: synchronisation, compétition pour l'accès à une ressource partagée, expression du parallélisme etc...

I-3 Ordonnancement.

Lorsque la décomposition en processus parallèles est effectuée et la synchronisation maîtrisée, il est impératif de concevoir des systèmes respectant les contraintes temporelles énoncées dans le cahier des charges. Ainsi l'ordonnancement dans les applications temps-réel constitue un problème central. Il intervient dans les situations suivantes:

- dans le cas du pseudo-parallélisme, l'exécution sur un processeur réel unique de plusieurs activités nécessite, la mise en oeuvre d'une politique d'allocation du processeur, respectant les contraintes suivantes:
  - *assurer la priorité de certaines tâches par rapport à d'autres.*
  - *respecter les échéances pour toutes les tâches.*
- dans le cas du parallélisme vrai, la répartition des tâches sur les processeurs disponibles doit garantir les contraintes temporelles. Ceci se produit notamment lorsque les tâches concernées par cette répartition peuvent être exécutées indifféremment sur plusieurs processeurs.
- lors de l'accès concurrent de plusieurs tâches à une ressource partagée de manière à satisfaire des temps de réponse imposés (contrainte forte) ou souhaités (contrainte faible).

Enfin il est nécessaire d'évaluer les performances d'un ordonnancement donné en utilisant un modèle de représentation du comportement temporel de l'application.

I-4 Sûreté de fonctionnement.

L'occurrence de fautes ou d'intrusions de signaux en provenance du procédé peut fausser l'image de l'état du procédé dont dispose le système de contrôle. Une telle situation peut laisser dériver le procédé vers des états inacceptables.

Les fautes peuvent provenir soit des instruments du procédé (capteurs, actionneurs), soit d'un comportement incorrect d'un constituant matériel du système de contrôle, soit encore d'une mauvaise conception ou programmation de l'application. Le dépassement d'échéance par une activité constitue un autre type de faute qui relève d'avantage de la surveillance. En effet un tel dépassement ne doit pas être détecté lors de son occurrence, car il est généralement trop tard pour tenter de le compenser, mais il doit être anticipé.

L'occurrence des fautes dans les applications temps-réel peut être prise en compte dès la phase de conception, en intégrant dans l'application des éléments tels que: redondance matérielle, redondance logicielle, reconfiguration automatique.

II CYCLE DE VIE D'UN SYSTEME TEMPS-REEL [PER90].

La vie de tout système informatique peut être décomposée en deux stades indépendamment de son environnement, de sa taille et de sa complexité: **son développement, son exploitation et maintenance**.

Le stade de développement est précédé d'une phase préalable de **spécification des besoins**. Celle-ci permet de formuler le plus clairement possible les besoins du client en termes d'objectifs, de performances et d'évolutions du produit etc.. Cette phase se traduit généralement par une rédaction libre en langage naturel.

Elle comporte également l'étude de faisabilité du système. Celle-ci porte sur les aspects économiques, techniques et sur l'environnement technique et humain du système. Pour les systèmes temps-réel, l'examen des disponibilités des ressources est effectué essentiellement en fonction des temps de réponse, vitesses de transfert requises etc..

II-1 Développement.

Le stade de développement comporte les phases de spécification, de validation, de conception, de construction, d'intégration et de validation: c'est le **cycle de développement** (figure 1.2).

II-1-1 Spécification.

La phase de **spécification** s'opère généralement en deux étapes:

- la **spécification semi-formelle** qui permet d'analyser et d'affiner les désirs exprimés en décomposant le système initial en sous systèmes. Ainsi cette étape doit détailler la sémantique, le rôle et le comportement de chaque sous système ainsi que les interactions entre les sous systèmes. Des retours à la phase de spécification non formelle sont envisageables dans le cas où des incohérences ou des omissions apparaissent.

- la **spécification formelle** dont le but final est de générer automatiquement les programmes. Cette étape doit utiliser un outil permettant des spécifications claires, précises, non ambiguës, complètes et favorisant le dialogue. Les spécifications concernent le contrôle (communication, synchronisation), le parallélisme et les contraintes temporelles. **L'outil utilisé doit permettre d'obtenir un modèle qui autorise la validation des spécifications.**

II-1-2 Validation.

Cette phase a un triple objectif:

- valider la **représentation**, c'est à dire sa conformité aux spécifications du client.

- vérifier la **cohérence** de la spécification: garantir que des situations indésirables telles que **étreinte mortelle**, **ordres contradictoires** ne se produisent pas.

- vérifier la **complétude** de la spécification.

Pour cela deux méthodes sont envisageables:

- la **validation formelle ou a priori.** C'est le cas par exemple de la validation structurelle dans les réseaux de Petri (RdP), par analyse de leurs propriétés.

- la **validation par simulation à événements discrets.** Celle-ci permet de dérouler toutes les situations possibles pour la modélisation et par conséquent de déterminer les états du système sources de problèmes. Toutefois les simulations sont généralement partielles et contingentes des données initiales. La simulation peut apporter une aide appréciable pour l'estimation et le dimensionnement des paramètres.

I-1-3 Conception.

La **conception** a pour but de répondre à la question "comment faire?". On peut y distinguer deux étapes:

- la **conception préliminaire**: On effectue d'abord les choix d'allocation des fonctions aux matériels et/ou aux logiciels puis le choix du matériel et du logiciel conséquents. Ensuite cette étape consiste à produire l'architecture globale du système et à déterminer une façon de l'intégrer.

- la **conception détaillée**: Celle-ci établit une façon de procéder, séparément, pour le logiciel et le matériel.

II-1-4 Construction, Intégration et Validation.

Suivant les résultats de la conception détaillée, la **construction** des divers éléments composant le système est effectuée. Pour le logiciel, il s'agit du codage, des tests unitaires puis de l'intégration du logiciel. Pour le matériel, il s'agit de la fabrication du prototype puis des tests et mesures.

L'**intégration** du système concerne la phase d'unification du logiciel et du prototype matériel en présence du procédé à conduire. Enfin la **validation** consiste à établir que le produit est conforme au besoin exprimé par le client.

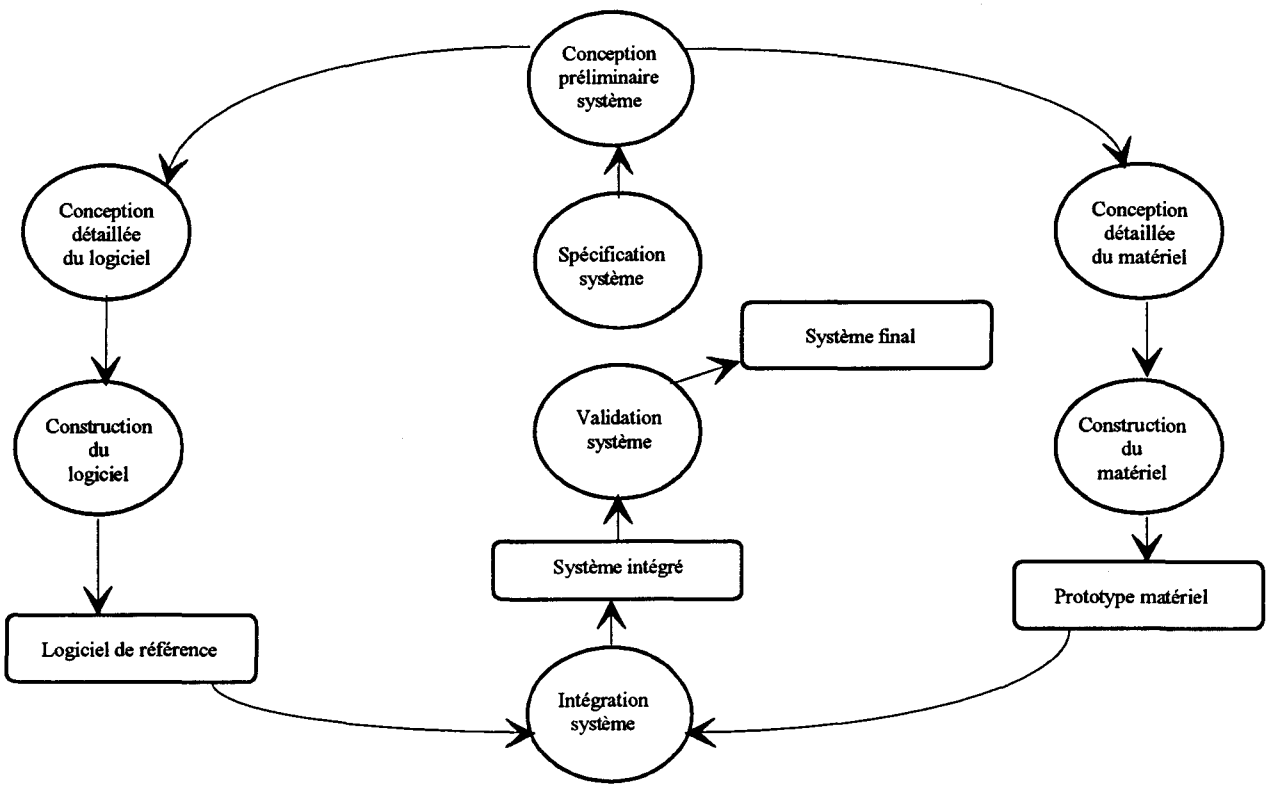


figure 1.2

II-2 Exploitation et Maintenance.

Ce second stade concerne la mise en oeuvre, **l'exploitation** et **la maintenance** du système temps-réel sur le site. Celles-ci inscrivent le système dans la durée et révèlent la qualité de son développement. Les caractéristiques de fiabilité et de maintenance y sont alors très appréciées. Pour obtenir de bons résultats, il est nécessaire d'intégrer, en particulier, des politiques de tolérance aux fautes dès la phase de conception. En outre ce stade doit assurer la conformité de la réalisation par rapport aux spécifications.

Actuellement aucune méthode concernant la spécification et la conception préliminaire ne s'est imposée comme standard de fait, dans le domaine du temps-réel. L'objet en particulier de notre travail, ainsi que nous le verrons dans les paragraphes suivants, est de concevoir une méthode pour le domaine spécifique de la commande en temps-réel des cellules flexibles d'assemblage par robots.

III LE CARACTERE TEMPS-REEL DES ATELIERS FLEXIBLES.

Les ateliers flexibles sont habituellement structurés en 4 niveaux (figure 1.3):

- Gestion-Planification et Ordonnancement prévisionnel.
- Décision et Ordonnancement temps-réel.
- Supervision-Coordination et Commande Globale.
- Commande Locale.

A partir d'un plan de charges établi pour un horizon assez lointain (quelques mois), le niveau **Gestion-Planification** élabore un ordonnancement prévisionnel sur plusieurs semaines.

Les ordres de fabrication sont ensuite envoyés vers le niveau **Décision** qui possède deux types de fonctions:

- prendre en compte l'état de l'atelier et remettre en cause l'ordonnancement temps-réel chaque fois que l'environnement est perturbé par des pannes ou des commandes urgentes.
- décider du lancement des produits et de l'affectation des tâches aux machines afin de satisfaire un critère d'optimisation imposé.

Lorsque le choix est arrêté, les ordres conséquents sont transmis vers le niveau **Supervision-Coordination**. Chaque **Commande Globale** génère et coordonne alors les différents ordres à envoyer au niveau **Commande Locale** pour atteindre l'objectif.

Finalement le niveau **Commande Locale** assure les fonctions de bas niveau telles que automatismes réflexes, commandes directes, etc...

Les deux premiers niveaux rassemblent ce qu'il est convenu d'appeler les activités de **Gestion-Conduite**, tandis que les deux autres niveaux intègrent les activités de **Commande**. Les contraintes temps-réel sont naturellement les plus sévères dans les niveaux les plus bas.

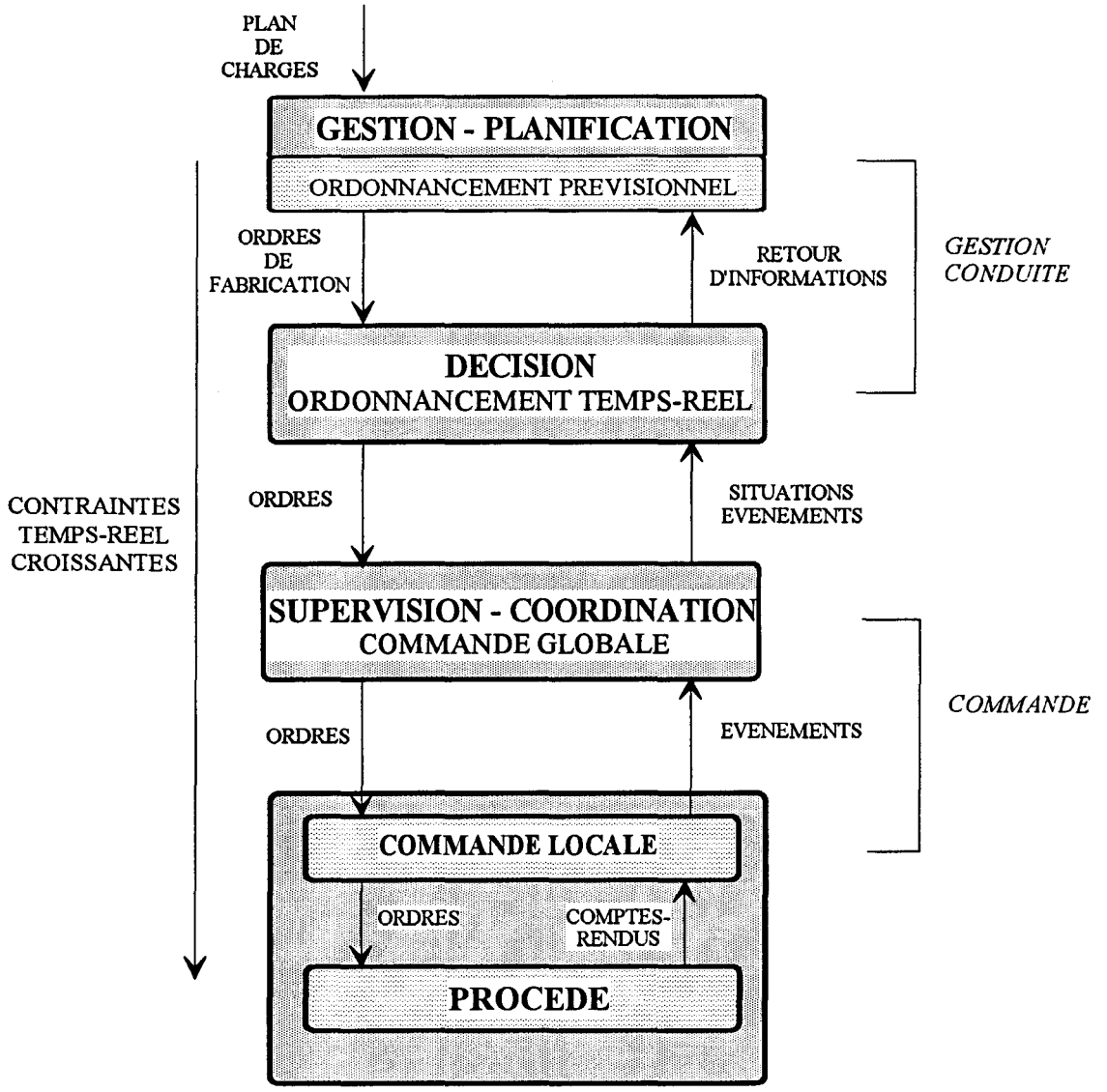


figure 1.3

En résumé deux flux d'informations apparaissent:

- un **flux descendant** associé aux activités de **conduite** et de **commande**. Des décisions sont prises en temps-réel en tenant compte de l'état instantané du système et des requêtes venant des niveaux supérieurs.

- un **flux ascendant** associé aux activités de **surveillance**. Chaque niveau de décomposition collecte des informations élémentaires qu'il combine et fait remonter au niveau supérieur. Il s'agit en particulier de faire remonter les erreurs irréparables détectées. Dans l'optique ateliers flexibles les niveaux ont la faculté de corriger eux mêmes certains dysfonctionnements.

Toutes ces opérations (commande, surveillance) sont soumises à des contraintes temporelles. A ce titre, elles rassemblent les caractères temps-réel des ateliers flexibles.

L'étude qui suit se situe essentiellement sur le plan de la commande et plus particulièrement sur celui de la commande globale des activités au sein d'une cellule flexible d'assemblage.

IV LA COMMANDE DANS LES SYSTEMES DE PRODUCTION.

Deux démarches principales ont été proposées pour la structuration de la commande dans les systèmes de production:

- une démarche hiérarchique.
- une démarche intégrée proposée dans le cadre du projet CASPAIM [BOU88-a].

IV-1 La démarche hiérarchique.

Elle est conforme au schéma de la figure 1.3. En effet elle correspond à une structure de décomposition de la commande en différents niveaux d'abstraction. C'est la démarche adoptée dans le système de commande A.D.C.S (Autonomous Decentralized Control System) [KOM84], dans la méthodologie A.M.R.F (Automated Manufacturing Research Facility) [ALB82], ainsi que dans le projet SECOIA (Spécification Emulation et COnception Informatisée d'Ateliers) [COU83] [COU84] du LAAS, qui intègre non seulement une méthodologie de commande mais aussi un support matériel et logiciel.

Dans le projet SECOIA la commande se décompose en trois niveaux d'abstraction [SAH87] (figure 1.4):

- la commande des machines ou **commande locale**.
- la **coordination** des sous ensembles.
- l'ordonnancement en temps-réel et le **pilotage global**.

La commande locale des machines correspond soit à des commandes numériques de machines outils, soit à des armoires de commande de robots soit à des automatismes logiques.

La coordination des sous-ensembles est essentiellement logique et correspond à la maîtrise de la coopération entre les robots et les divers équipements.

Quant au dernier niveau qui relève plus du domaine de la conduite que de celui de la commande, il a pour fonction de prendre une décision chaque fois qu'un événement survient au niveau global de l'atelier. Ses fonctionnalités sont:

- l'ordonnancement temps-réel
- le calcul d'itinéraire
- la gestion de stock
- la coordination globale
- le suivi et la supervision.

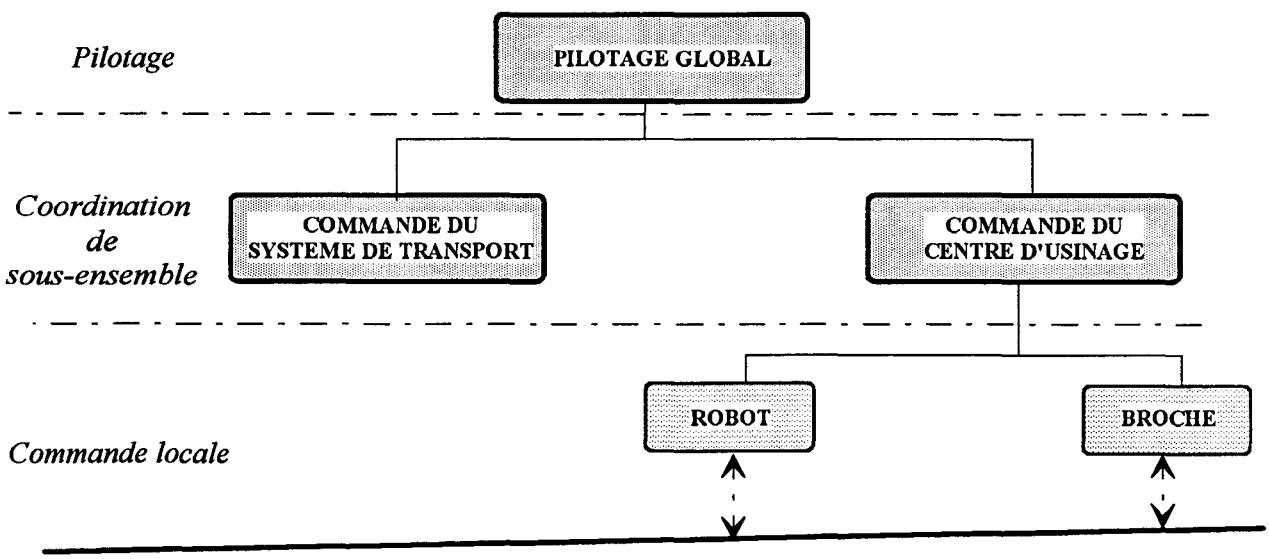


figure 1.4

IV-2 La démarche intégrée.

Celle-ci consiste à "mettre à plat" les différentes fonctions du système de commande et à ne considérer que deux niveaux de décomposition: un niveau hiérarchique et la partie commande (figure 1.5). Le niveau hiérarchique a alors pour fonction d'envoyer à la partie commande un ensemble de paramètres précisant les objectifs de production ainsi que les fonctions de supervision. Ceci revient à intégrer le niveau Décision de la figure 1.3 dans le niveau Supervision-Coordination

C'est la démarche adoptée dans le cadre du projet CASPAIM (Conception Assistée des Systèmes de Production Automatisée en Industrie Manufacturière) de l'Ecole Centrale de Lille [BOU88-b].

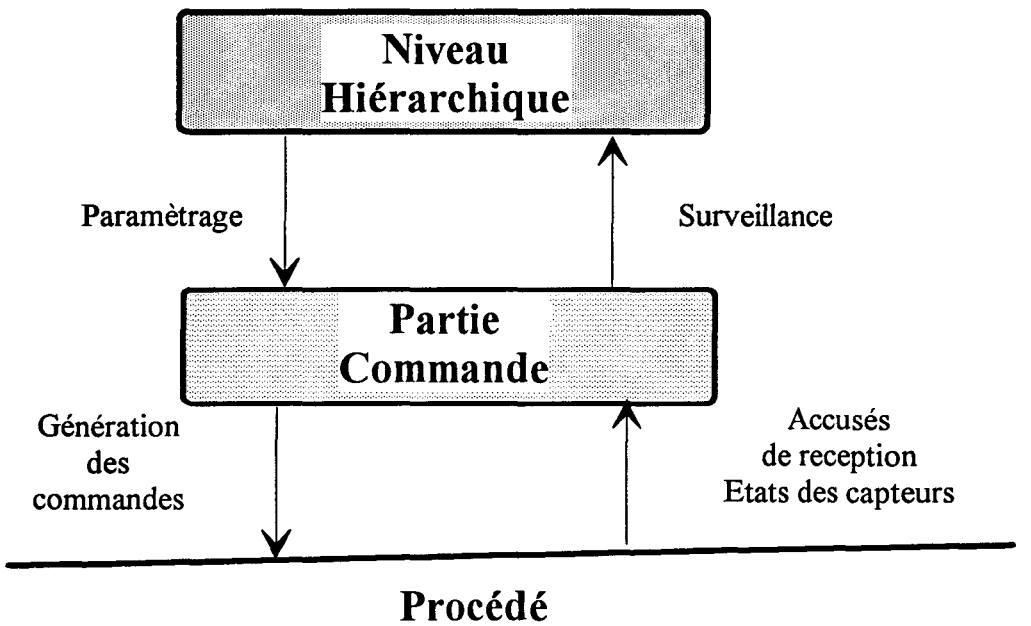


figure 1.5

V L'ASSEMBLAGE [BOU84].

La fabrication d'un produit fini est accomplie à partir d'un préproduit constitué de composants qui sont:

- soit des composants élémentaires $\{c_1, c_2, \quad, c_m\}$, tels que deux composants c_i et c_j sont liés par une ou plusieurs liaisons mécaniques.
- soit des sous-ensembles préassemblés en amont appelés sous-assemblages.

L'assemblage de deux objets consiste à créer une ou plusieurs liaisons mécaniques entre eux . On dit alors qu'ils sont liés par une liaison fonctionnelle unique. La réalisation d'une liaison fonctionnelle nécessite l'accomplissement d'un ensemble d'actions dit action fonctionnelle. Une action fonctionnelle vise à modifier l'état des deux objets à relier (propriétés physiques, forme, etc...), afin de les faire passer dans un nouvel état matériel.

L'ensemble des actions fonctionnelles à mettre en oeuvre pour aboutir au produit fini peut être représenté par un graphe, dit graphe des liaisons fonctionnelles, où les m sommets représentent les composants élémentaires et où les n arcs sont associés aux n liaisons fonctionnelles. La figure 1.7 illustre ce concept dans le cas du stylo à bille représenté figure 1.6.

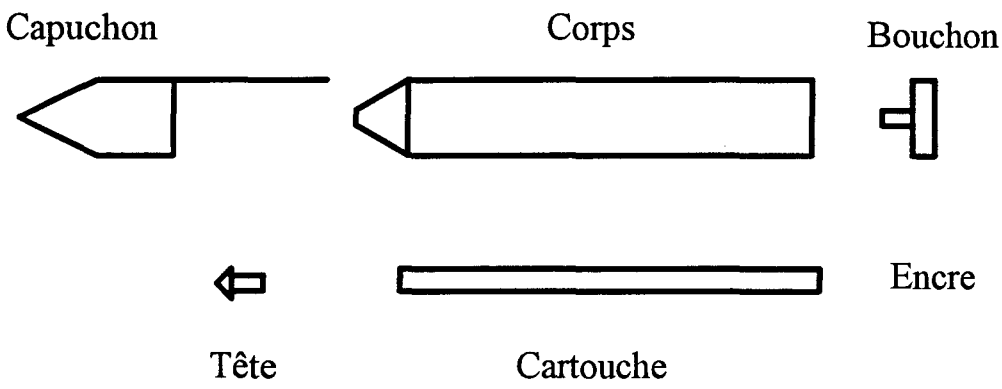


figure 1.6

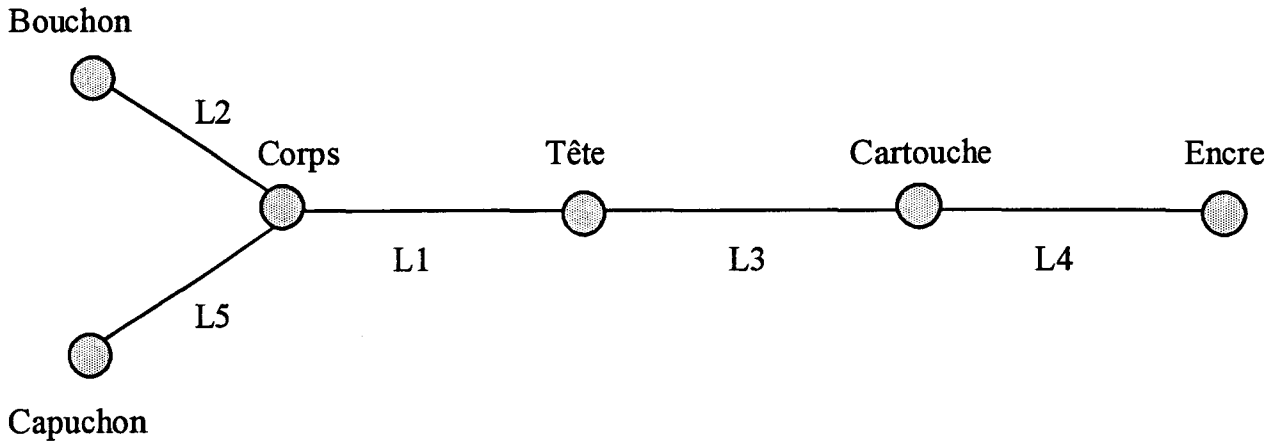


figure 1.7

En règle générale il existe plusieurs façons d'assembler un produit fini à partir des composants élémentaires, chaque façon correspondant à ce que l'on nomme **gamme d'assemblage**. Celle-ci doit satisfaire un ensemble de contraintes inhérentes au produit même. Ces contraintes, appelées **contraintes de réalisation**, traduisent la nécessité de tenir compte, lors de la réalisation d'une liaison fonctionnelle, de la présence ou de l'absence d'autres liaisons fonctionnelles. Les contraintes de réalisation se traduisent par les relations n-aires suivantes:

- une relation binaire de type R entre deux liaisons fonctionnelles L_i et L_j s'exprime par le prédicat: " il est impossible de réaliser L_i lorsque L_j est établie ". On note $R(L_i, L_j)$.

- une relation binaire de type S entre deux liaisons fonctionnelles L_k et L_i s'exprime par le prédicat: " il est impossible de réaliser L_i lorsque L_k n'est pas établie ". On note $S(L_k, L_i)$.

- une relation p-aire de type R_p entre les liaisons fonctionnelles L_1 et $(L_2,.., L_p)$ s'exprime par le prédicat: " il est impossible de réaliser L_1 lorsque L_2, L_3,....et L_p sont établies ". On note R_p $(L_1, L_2,.., L_p)$.

- une relation p-aire de type S_p entre les liaisons fonctionnelles L_1 et $(L_2,.., L_p)$ s'exprime par le prédicat: " il est impossible de réaliser L_1 lorsque ni L_2 ni L_3....ni L_p ne sont établies ". On note $S_p(L_2,.., L_p, L_1)$.

A partir de ces éléments la recherche des gammes d'assemblages peut s'effectuer de deux manières:

- en réalisant une modélisation mathématique [BOU86], dont l'exploitation permet de trouver toutes les suites croissantes représentant les différents états intermédiaires que peut prendre en théorie le produit (sans intégration des contraintes de réalisation), depuis l'état initial ou aucune liaison fonctionnelle n'est établie, jusqu'à l'état final ou toutes les liaisons fonctionnelles sont établies. Ces suites peuvent êtres représentées par un ensemble d'arbres, où chaque arbre représente les séquences temporelles possédant la même liaison au départ.

  Ensuite, l'intégration des conditions n-aires permet de supprimer les arbres et les branches correspondant à des états intermédiaires interdits.

- en traduisant chaque liaison fonctionnelle par un RdP et en intégrant les contraintes n-aires dès le départ [BOU87-a]. L'ensemble des gammes opératoires est alors constitué des séquences complètes de franchissements possibles.

Dans le cas du stylo à bille, l'utilisation de l'une ou l'autre des méthodes aboutit à douze (12) gammes opératoires représentées sur la figure 1.8.

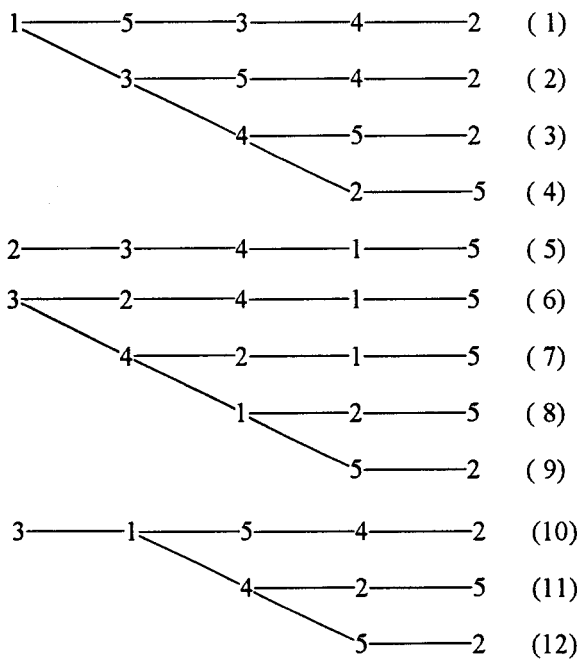


figure 1.8

Les séquences obtenues par les méthodes citées sont des séquences séries, pour lesquelles le passage d'un état intermédiaire à un autre, se fait par l'établissement d'une seule liaison fonctionnelle.

Il existe en outre des séquences, appelées processus série-parallèle, pour lesquels certaines actions sont effectuées en parallèle. La mise en évidence de ces processus peut se faire à partir des graphes des séquences séries. On cherche alors dans chaque séquence si, pour q actions réalisées en série, l'état intermédiaire atteint est le même que celui atteint lorsque ces mêmes actions sont réalisées simultanément.

En particulier, les actions permutables (aux rangs t et t+1 dans deux séquences (k) et (l)), c'est à dire les actions pour lesquelles les deux séquences (k) et (l) sont identiques sauf aux rangs t et t+1, où l'ordre est inversé, conduisent à des processus série-parallèle.

A titre d'exemple, les liaisons L_2 et L_4 des séquences (6) et (7) de la figure 1.8 vérifient cette propriété et conduisent au processus série-parallèle illustré par le graphe de la figure 1.9.

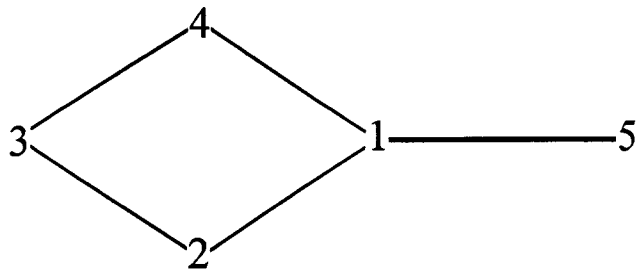


figure 1.9

Les actions permutables procurent une **flexibilité locale** au processus d'assemblage.

Lorsque le nombre de composants à assembler est important il est intéressant de mettre en oeuvre des sous-assemblages, réalisés **indépendamment les uns des autres**.

Formellement un sous-assemblage s'identifie à un sous graphe noté SG(p, r) à p sommets et r liaisons, issu du graphe des liaisons fonctionnelles et possédant les propriétés suivantes:

- le sous graphe est connexe.
- l'établissement des r liaisons est possible d'une façon au moins.
- lorsque les r liaisons sont établies il est toujours possible d'effectuer les (n-r) restantes.

A titre d'exemple le sous graphe extrait de la figure 1.7 constitué des 3 sommets: cartouche, encre et tête (p=3) et des liaisons: L_3 et L_4 (r=2) constitue un sous-assemblage car il vérifie les trois propriétés énoncées précédemment.

La mise en évidence des sous-assemblages peut s'effectuer de deux manières [BOU87-b]:

- a priori par décomposition menée à partir du graphe des liaisons fonctionnelles.
- a posteriori à partir des graphes représentant les séquences temporelles.

Il résulte de ce qui précède qu'un sous-assemblage est soumis aux mêmes contraintes qu'un composant, ce qui se traduit par les propriétés d'indépendance suivantes:

- deux sous-assemblages ne peuvent pas avoir de constituants communs.
- le produit fini peut être obtenu en assemblant, deux à deux, des sous-assemblages réalisés en parallèle.

Par exemple, les deux sous-assemblages de la figure 1.7 constitués respectivement de (tête, cartouche, encre) et (corps, bouchon) constituent deux sous-assemblages exécutables en parallèle.

Les modes opératoires et les équipements à mettre en oeuvre peuvent être très différents selon les cas envisagés (processus série-parallèle, sous-assemblage,...). Dès lors, la recherche de toutes les séquences opératoires possibles représente toutes les hypothèses de travail engendrant différentes stratégies de mise en oeuvre.

VI ANALYSE DES SYSTEMES D'ASSEMBLAGE [OLI86].

La structure d'un centre de production se présente sous forme de plusieurs départements. Parmi ceux-ci figurent: l'approvisionnement, la maintenance, l'outillage et la fabrication. Le département fabrication réalise des tâches telles que: usinage, moulage, assemblage,...etc.

Selon une décomposition qui tient compte de la technologie mise en oeuvre et de la nature des produits fabriqués, le département de fabrication est constitué d'îlots reliés par des réseaux de transport aboutissant éventuellement à des zones de stockage.

Parmi les différents îlots, l'îlot d'assemblage est un ensemble d'équipements assurant l'assemblage complet soit d'un produit déterminé Pr, soit d'une famille de produits. Une famille de produits peut, elle même, appartenir à l'une des classes suivantes [LHO88]:

- **famille à pseudo variantes**: c'est une famille dont les membres admettent tous la même géométrie et les mêmes modes de solidarisation interne. Les différences portent sur l'aspect de certains constituants (couleur..) ou sur certaines propriétés (nature du matériau..).

- **famille à variantes substitutives**: les membres peuvent différer sur les formes et les dimensions mais les interfaces avec les autres produits restent identiques.

- **famille dimensionnelle**: les membres prennent des valeurs géométriques selon une échelle.

- **famille fonctionnelle**: les membres ont les mêmes fonctionnalités mais des matérialisations différentes par le nombre, la forme des composants et la nature des liaisons.

Les flux entrant dans l'îlot sont des flux discrets de composants fabriqués dans d'autres îlots du même département ou en provenance de l'extérieur. Les flux sortant sont constitués de produits finis ou de recyclage d'outils ou de rebuts.

La décomposition hiérarchique descendante d'un îlot d'assemblage fait apparaître les cellules d'assemblage reliées les unes aux autres par des stocks tampons ou par des convoyeurs (figure 1.10).

La cellule d'assemblage assemble un produit intermédiaire ou fini à partir de constituants et de produits intermédiaires émanant d'autres cellules. Au même instant plusieurs produits sont assemblés (ou présents) dans la cellule. Une cellule est caractérisée par:

- son temps de cycle Tc séparant les apparitions successives de deux produits.
- son temps de traitement Tt représentant la durée de séjour d'un produit à l'intérieur de la cellule.

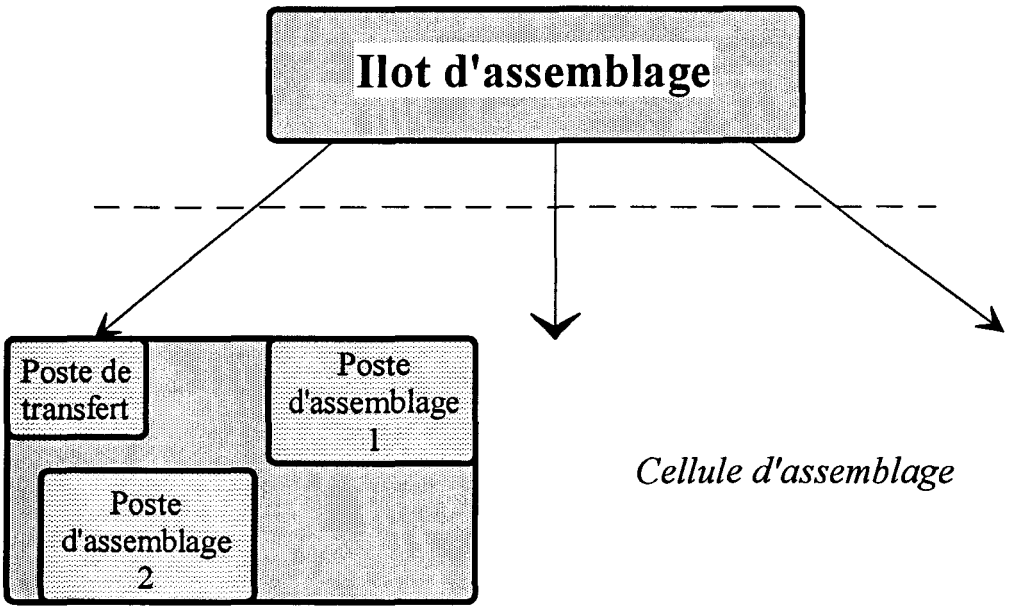


figure 1.10

Le poste d'assemblage est l'ensemble des équipements contribuant au montage d'un produit intermédiaire et un seul à la fois. A l'entrée d'un poste se trouvent des constituants isolés mis à disposition par des moyens de transport (convoyeur ou autre). A l'interface poste-cellule se situent des stocks d'entrée ou de sortie. C'est au sein des postes d'assemblage que doit s'exprimer, s'il y a lieu, la **flexibilité locale**. Sous cet aspect, le problème du choix de la gamme ainsi que de la distribution des tâches aux équipements, que ce soit en phase de conception ou de reprogrammation d'une cellule, est primordial.

Le problème doit être formulé dès le départ en étudiant et en évaluant les performances de l'atelier:

- fréquence de sortie ou temps de cycle de chaque cellule.
- durée d'utilisation des machines.
- dimension des stocks tampons.
- temps de traitement des postes d'assemblage.

VII APPROCHE SYSTEMES A EVENEMENTS DISCRETS.

Les problèmes de spécification rencontrés dans la conception et la commande des systèmes industriels discontinus sont à l'opposé de ceux traités habituellement dans le domaine des processus industriels continus. Cette opposition se concrétise essentiellement par la nature même des produits manipulés. Dans l'industrie chimique, par exemple, les flots de matière manipulée sont issus de processus de transformations continus. Par contre, dans l'industrie manufacturière, les flots se traduisent par des séries de pièces issues de processus discontinus.

Cette différence se traduit et se formalise par la nature même des variables associées aux produits et aux machines de ces processus. Aux processus continus sont associées des variables analogiques dont l'évolution est continue dans le temps: mesure de débit, taux d'acidité. En revanche les variables représentant les processus discontinus sont de nature booléenne et à évolution discontinue: actions tout ou rien, fins de course.

Le fonctionnement d'une machine dans ses différents modes et la manipulation d'une pièce sont des fonctions dites logiques. Ces fonctions sont de deux types:

- fonction combinatoire: les sorties dépendent exclusivement des entrées.
- fonction séquentielle: les sorties à un instant t dépendent non seulement des entrées à cet instant, mais aussi de l'état antérieur.

Le temps est alors appréhendé à travers un ensemble d'instants caractéristiques de l'évolution du processus représentant la réalisation de conditions et/ou l'apparition

d'événements. Pour un état donné et pour un instant caractéristique de cet état il y a franchissement d'une transition traduisant l'évolution ou le changement d'état du processus séquentiel. A la suite de ces états correspond une suite d'actions associées à ceux-ci. Les systèmes automatisés de production sont ainsi appréhendés en tant que Systèmes à Evénements Discrets (S.E.D).

CONCLUSION.

Une méthode de spécification se compose:

- d'un ou de plusieurs **outils de modélisation** et de règles d'assemblage,
- d'une démarche ou **stratégie de spécification** qui permet de construire, de façon suffisamment formalisée, le modèle du système

Cette méthode doit pouvoir exprimer: le caractère réactif, le caractère transformationnel et les contraintes temporelles.

Ces diverses descriptions exigent de l'outil utilisé:

- de décrire le système d'une façon claire et précise.
- de se prêter aux méthodes d'analyse et de validation.
- d'être un moyen de dialogue et d'échange, entre le demandeur et le réalisateur, entre les membres d'une équipe de développement, entre les développeurs et le personnel de maintenance.
- de permettre une mise en oeuvre directe avec garantie de la conformité de l'exécution par rapport à la spécification.

Une approche purement fonctionnelle n'est pas satisfaisante du fait qu'elle ne permet pas d'exprimer l'aspect événementiel et les contraintes temporelles.

Les méthodes issues de l'Analyse Structurée [YOU78] étendue au temps-réel: Analyse Structurée/Temps-Réel (Structured Analysis/real Time) telles que HP [HAT87][HAT91], WM [WAR85][WAR87], DARTS [GOM84][GOM86], sont des méthodes qui s'adressent aussi bien aux systèmes continus qu'aux systèmes discrets.

Cette exhaustivité diminue leurs capacités dans la représentation des systèmes à événements discrets. En particulier elles ne prennent pas en charge la spécification des gestions des stockages de données (file d'attente). Par ailleurs les outils utilisés ne sont pas toujours des plus familiers.

D'autres méthodes existent dans la littérature: le modèle de Calvez [CAL82], Paisley [ZAV82], Toocatta [SZY88]. Mais soit qu'elles sont mal adaptées aux S.E.D soit qu'elles ne vérifient pas la totalité des exigences citées ci-dessus.

Une approche équationnelle de type langage synchrone [BEN90] (synchrone dans le sens où la réponse à tout événement extérieur est supposée être simultanée avec son occurrence, Lustre [CAS91] ou Signal [LEG91]), est très bien adaptée pour décrire le caractère réactif et les contraintes temporelles. Toutefois, dans cette approche, les actions qui durent ne s'expriment pas naturellement.

Pour ce qui concerne les outils, les extensions des RdP: RdP temporels, RdP temporisés [RAM83], RdP synchrones [MOA85], RdP colorés synchrones [ILI90] permettent de considérer le temps, donc de spécifier le niveau réactif. Ainsi la règle qui consiste à imposer qu'une transition soit franchie dès qu'elle est validée a été rendue nécessaire pour ces extensions. Toutefois la plupart des propriétés mathématiques du modèle de base sont perdues (les propriétés relatives aux composantes conservatrices restent valables).

Le Grafcet, outil formel et graphique très répandu (voir Annexe 1), sert souvent d'outil de spécification du niveau commande locale (automatismes réflexes..) [PIC87] [PRU87] [MOS89]. Il a également fait l'objet d'utilisations éphémères pour la simulation et la commande d'ateliers flexibles [DAL84]. Pour ce dernier cas le RdP avec ses diverses extensions a pris le pas sur le Grafcet.

On assiste toutefois actuellement à un regain d'intérêt pour le Grafcet, particulièrement de la part des chercheurs sur les langages synchrones [AND92-a] [MAR92]. Cet intérêt est suscité d'une part par les extensions du Grafcet: Grafcets partiels et macroactions (voir Annexe 2), et d'autre part par le fait que le Grafcet est également un **outil synchrone**.

Il est vrai que le Grafcet peut aboutir très vite à des représentations touffues. Cet inconvénient peut être compensé par une **démarche structurée de la spécification.** Ceci peut être obtenu notamment en dégageant des motifs renouvelables. Ainsi pour des petites applications, du type cellule flexible d'assemblage, le Grafcet est intéressant à exploiter. En outre il peut être utilisé aussi bien comme outil de **spécification** que comme outil de **mise en oeuvre** (commande). Pour toutes ces raisons, le Grafcet est l'outil que nous avons exploité dans cette étude.

Dans ce contexte (figure 1.11), le second chapitre expose les approches utilisées pour la spécification des caractères réactif (ou événementiel) et transformationnel (ou fonctionnel). Le troisième chapitre présente l'intégration des outils dégagés dans les processus. Enfin le quatrième chapitre présente l'ensemble de la stratégie de spécification de la commande temps-réel de cellules flexibles d'assemblage. Celle-ci intègre naturellement la spécification des contraintes temporelles grâce notamment aux notions de hiérarchie et de macroactions.

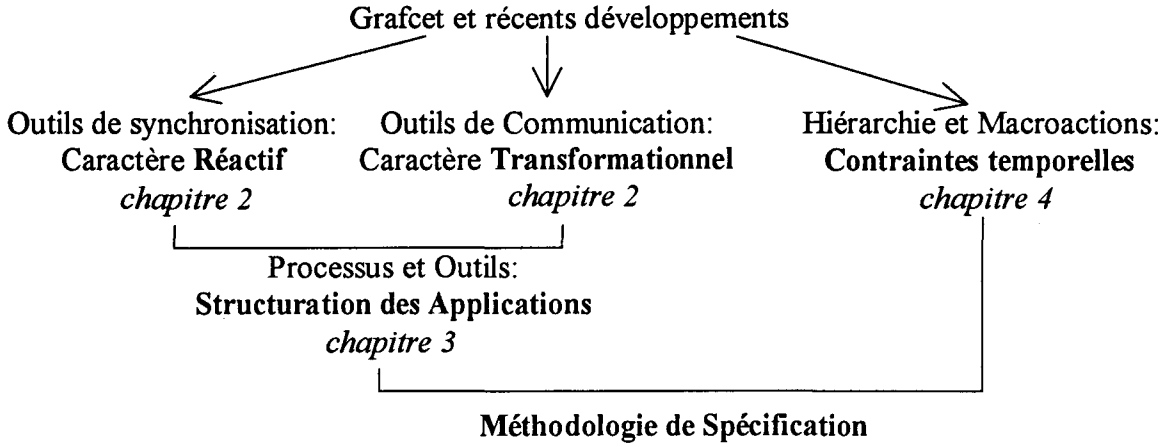


figure 1.11

Le cinquième chapitre concerne la phase de conception. Il introduit d'abord les algorithmes d'interprétation nécessaires aux développements associés au Grafcet. Il enchaîne ensuite en cernant les conditions requises pour la mise en oeuvre de la commande dans un contexte centralisé.

Enfin le sixième chapitre récapitule l'application de l'ensemble de la méthode pour un montage de moteurs sur une cellule flexible d'assemblage.

CHAPITRE II:

SPECIFICATION DE LA SYNCHRONISATION

ET DE LA COOPERATION

SPECIFICATION DE LA SYNCHRONISATION

ET DE LA COOPERATION

INTRODUCTION.

" La hiérarchie est la forme que l'intelligence finie doit prendre pour s'adapter à la complexité " [SIM80].

En effet devant tout problème complexe, l'intuition humaine, pour mieux appréhender cette complexité, tend à décomposer le problème en sous problèmes. Si la complexité persiste ces derniers peuvent à leur tour être décomposés et ainsi de suite jusqu'à aboutir à une structure pyramidale.

Les applications de commande en temps-réel tout comme les systèmes informatiques n'échappent pas à cette règle. On parle alors de décomposition en processus plus ou moins indépendants qui évoluent en parallèle en interagissant entre eux et avec le procédé pour accomplir l'objectif commun souhaité [WAL84].

Pour cela ils doivent coopérer:

- d'une part pour se partager l'accès à des ressources communes, qui sont soit de nature informationnelle (variable, base de données..), soit de nature physique (chariot, outil..).

- d'autre part pour respecter l'ordre d'exécution défini par l'application et pour se communiquer des informations.

Lorsque les processus accèdent en exclusion mutuelle à une ressource partagée, on dit qu'ils sont en **concurrence** ou en **compétition** vis à vis de cette ressource.

Si la coopération ne concerne que l'échange d'informations, les processus sont dits en **communication**.

Enfin la coopération peut se restreindre à l'ordre d'exécution, il s'agit alors de la **synchronisation**.

Ces trois cas font référence au caractère **transformationnel** ou **fonctionnel** de l'application.

La décomposition de l'application, ainsi que nous l'avons signalé au chapitre précédent, peut être rendue nécessaire du fait de l'existence de divers événements en provenance de l'environnement.

Il s'agit alors pour l'application, donc pour ces diverses tâches, d'être réceptives aux signaux issus de l'environnement et du procédé, c'est l'aspect **réactif** ou **événementiel**. Celui-ci s'exprime également en termes de **synchronisation**.

Certains auteurs [PET74] ne font pas de distinction entre l'acte de communication et l'acte de synchronisation. En réalité, ceci dépend du niveau où on se place (spécification ou implantation de la commande).

En effet, au niveau implantation, toute communication suppose une synchronisation et inversement. Il n'en demeure pas moins que dans beaucoup de cas de spécification, il faut faire la distinction entre synchronisation et communication.

Toutefois cette distinction n'est pas toujours effective, même au niveau spécification de la commande. En effet, dans le cas de la relation producteur/consommateur par exemple, la **communication est du type synchrone** et s'accompagne nécessairement d'une synchronisation, puisque tout objet produit doit être consommé (structure flot de données).

En revanche dans le cas d'une communication asynchrone, il n'y a pas de synchronisation car la consommation n'a pas lieu pour toutes les données produites et peut même se produire plusieurs fois pour la même donnée.

Dans la littérature, les primitives décrivant les mécanismes de la synchronisation, de la communication et de la compétition sont nombreuses et sont souvent spécifiques du domaine d'application. En outre elles sont rarement supportées par une méthodologie d'approche des problèmes soulevés.

Afin de couvrir l'ensemble des applications que nous envisageons, nous avons retenu:

- pour la synchronisation l'approche opérée par Ladet [LAD77].

- pour la compétition et la communication essentiellement l'éventail des mécanismes cités par Thomesse [THO80].

Ce chapitre fixe dans un premier volet le vocabulaire utilisé pour les notions de processus et celles s'y rattachant. Dans un second volet, il expose l'approche utilisée dans la synchronisation. Celle-ci est valable aussi bien pour la synchronisation entre tâches que pour l'évolution de celles-ci par rapport à l'environnement. Enfin, on termine par les primitives de coopération retenues. On s'attache, tout le long de ce chapitre, à représenter les diverses primitives par l'**outil synchrone** choisi: le **Grafcet**.

I NOTION DE PROCESSUS.

On appelle processus un enchaînement d'états jalonné par des évolutions ou changements d'états. A ces divers états sont associées des actions représentant les sorties ou fonctions du processus.

Par convention, on représente un processus par un Grafcet qui traduit un enchaînement d'étapes auxquelles peuvent être associées des actions. Ces différentes étapes sont séparées, s'il y a lieu, par des mécanismes de synchronisation/communication, soit avec le procédé ou l'environnement, soit avec d'autres Grafcets évoluant en parallèle.

Lorsque, pour un processus donné, les directives de synchronisation/communication relatives à une succession d'étapes, ne se font qu'avec le procédé, alors on peut regrouper cette succession d'étapes dans une macroétape.

Pour illustrer ces notions, la figure 2.1-a présente un convoyeur équipé de deux capteurs: cap1 disposé en A et cap2 disposé en B. Le convoyeur achemine une pièce à la fois, du point A vers le point B. La figure 2.1-b donne une première représentation possible du graphe de commande de ce convoyeur. Pour cette représentation la description de l'enchaînement des étapes (et actions) est séquentielle. Il est aussi possible, et c'est le cas le plus courant, de décrire cet enchaînement d'une manière cyclique tel que le présente la figure 2.1-c.

En ce qui concerne la figure 2.1-b, si un objet arrive en A (cap1 est vrai) l'action impulsionnelle (notée *) associée à l'étape 1 fait démarrer le convoyeur. Celui-ci s'arrête lorsque l'objet arrive en B (cap2 vrai). La transition T3 est alors franchie et le graphe de commande retourne dans l'état initial.

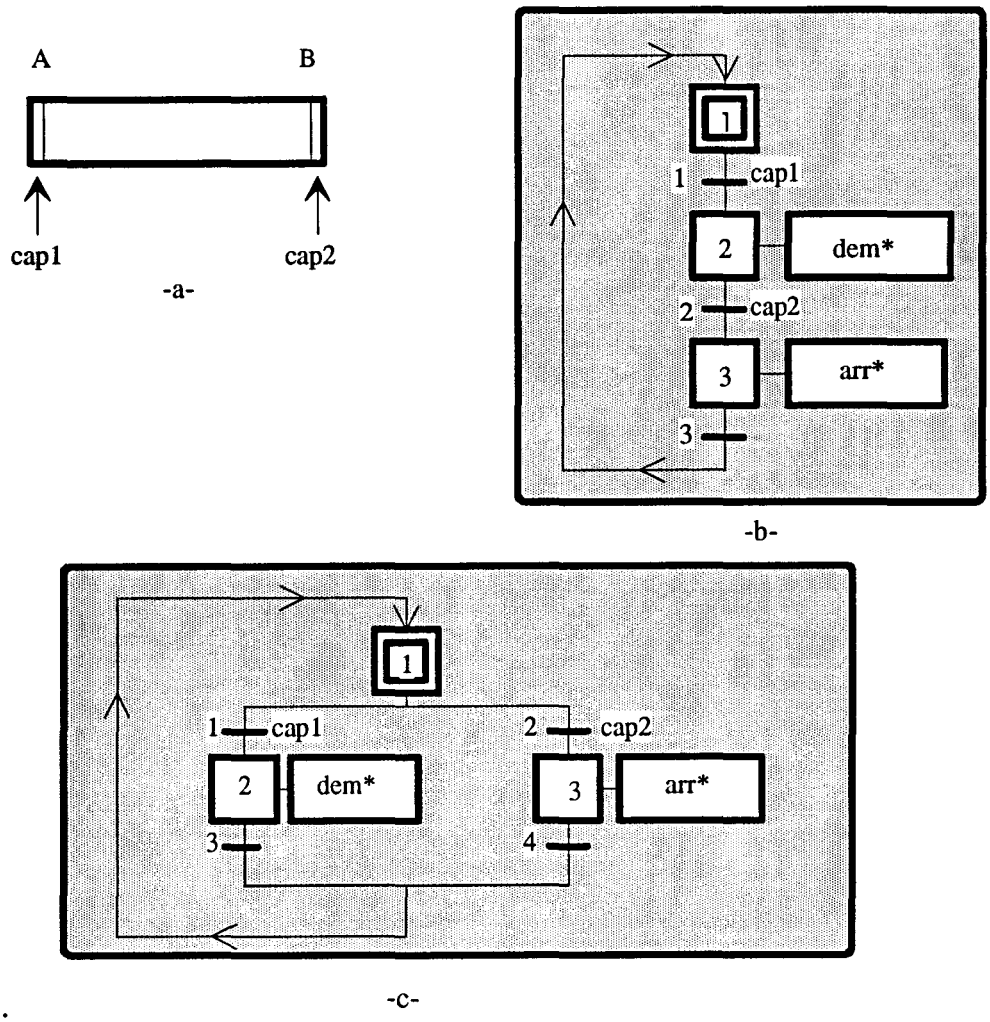


figure 2.1

Pour ce qui est de la figure 2.1-c, initialement l'étape 1 est active et dès qu'une pièce arrive en A la transition 1 est franchie, ce qui a pour conséquence de faire démarrer le convoyeur. T3 est ensuite franchie et le graphe retourne dans l'étape 1 dans l'attente du passage de la variable cap2 à vrai. En réalité la première représentation n'a été possible que parce qu'il existe un ordre dans les changements d'états des variables cap1 et cap2. Dans la terminologie adoptée un processus est une unité indivisible, c'est à dire qui ne peut pas être réparti sur plusieurs sites (cas des systèmes distribués). Par contre deux Grafcets ne sont pas forcément sur le même site.

II SPECIFICATION DE LA SYNCHRONISATION.

L'aspect événementiel prend en considération:

- les conditions sous lesquelles les actions se déclenchent, c'est à dire les événements qui les activent ou les désactivent.
- la façon dont les événements influencent le comportement du système.

Un outil de modélisation de l'aspect événementiel d'un STR doit permettre de:

- représenter les **événements** qui conditionnent l'état d'un système
- spécifier la **logique de contrôle** qui produit des actions et des événements en fonction d'événements en entrée, et fait changer d'état le système.

Une occurrence d'événement est définie comme l'apparition d'un changement d'état. **Un événement donné est alors l'ensemble des occurrences du changement d'état défini par l'opérateur et ordonné dans le temps**. Pour chaque occurrence on peut distinguer l'instant d'occurrence, l'instant de reconnaissance et l'instant de traitement. L'instant de reconnaissance, dépendant par ailleurs de l'implantation, doit être assez proche de l'instant d'occurrence afin de ne pas laisser échapper des changements d'états.

L'instant de traitement est l'instant où l'action associée à l'événement est activée. Si le traitement ne peut pas être lancé dès l'occurrence de l'événement, doit on retarder le traitement jusqu'à ce qu'il puisse être exécuté ou purement l'acquitter après un certain retard? Aussi, on associe à un événement, **une durée de vie** (time-out) qui indique le délai maximum séparant la reconnaissance du traitement (figure 2.2).

On peut rapprocher la durée de vie d'un événement de la **notion d'échéance** qui traduit la date limite à laquelle une action doit être terminée.

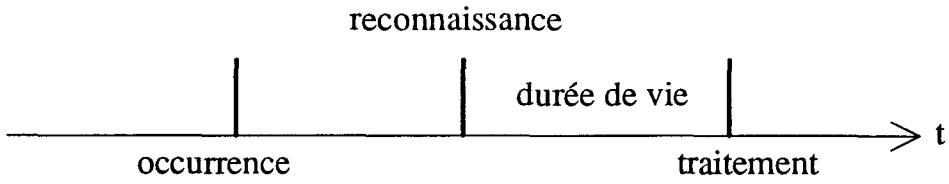


figure 2.2

L'échéance est telle que :

date de reconnaissance + durée de vie + temps maximum d'exécution ≤ date d'échéance.

Il est évident que lorsqu'on parle d'événements, il s'agit d'événements qui d'une part reflètent des changements d'états c'est à dire la dynamique de l'état du procédé de l'environnement ou du système de commande (pour ne pas dire Système de Contrôle Commande, car ceci suppose une surveillance) et qui ont d'autre part une signification vis à vis de l'évolution de la commande. Ainsi, les événements n'ayant de sens qu'au niveau informatique, ne sont pas considérés ici.

Pour pouvoir décrire les circonstances dans lesquelles une action doit être activée, on doit considérer **l'état** dans lequel se trouve le procédé, **les changements d'état** qui surviennent et **l'écoulement du temps**.

L'état du procédé, à un instant donné, est caractérisé par les valeurs de différentes grandeurs physiques et variables du procédé et de l'application. La dynamique du procédé se traduit par des changements de valeurs ou d'états. Enfin, le temps apparaît explicitement dans de nombreux cas par le biais d'activation périodique, d'activation différée dans le temps, de temps enveloppe ou de durée de vie.

II-1 Les liens de synchronisation événement-action [LAD77].

L'objet événement ne suffit pas, à lui seul, à déterminer les instants d'activation d'une action. Lors de la spécification, l'utilisateur doit, pour chaque événement utilisé, préciser à quelles occurrences il s'intéresse. C'est le lien de synchronisation événement-action.

La création d'un lien de synchronisation événement-action et l'apparition de l'événement à travers ses occurrences sont deux actes indépendants. L'instant de création du lien et l'instant d'occurrence de l'événement ne sont pas ordonnés a priori dans le temps.

La description des liens de synchronisation obtenue en déroulant toutes les occurrences pertinentes est alors complexe. Pour Ladet la définition d'un lien de

synchronisation entre un événement, ses occurrences et une action, ne fait pas intervenir chacune des occurrences indépendamment des autres, mais un groupe d'occurrences fonction du temps et de l'action choisie. Le temps se traduit et s'appréhende au travers d'instants caractéristiques. Pour la définition des liens de synchronisation ces instants sont:

- l'instant de validation de l'événement.
- l'instant d'invalidation de l'événement.
- l'instant de définition du lien de synchronisation.
- l'instant de destruction du lien de synchronisation.

Les instants de validation et d'invalidation d'une part, les instants de définition et de destruction d'autre part, vont déterminer des intervalles de temps, pendant lesquels les occurrences d'un événement seront prises en compte et respectivement exploitées.

La relation qui existe entre la considération d'un groupe d'occurrences et son exploitation à des fins d'activation dépend de la nature de l'action choisie. Selon le sens que l'on donne à la liaison événement-action et selon le contenu de l'action, on s'intéresse, soit à la présence ou l'absence d'occurrences, soit au nombre d'occurrences et à chacune d'elles.

Pour pouvoir contrôler l'ordre entre un événement et un lien de synchronisation, l'utilisateur doit distinguer deux types d'occurrences:

- les occurrences de l'événement survenant avant la définition du lien de synchronisation.
- les occurrences de l'événement apparaissant après la définition du lien de synchronisation.

En d'autres termes, l'utilisateur doit spécifier s'il s'intéresse, au moment de la définition du lien, aux occurrences passées et/ou aux occurrences futures.

II-2 Gestion des événements et des liens de synchronisation [LAD82].

Ladet a proposé la syntaxe suivante pour décrire les liens de synchronisation événement-action:

Pour (chaque) ⟨ événement⟩ (passé, futur) faire ⟨ ACTION ⟩.

Cette écriture suppose l'existence de moyens ou de directives de déclenchement d'événements, de validation et invalidation d'événements, de masquage et démasquage d'événements, d'acquittement d'événements et de destruction de liens de synchronisation.

L'éclatement de cette syntaxe aboutit aux six primitives suivantes, qui résument les différentes formes que peut prendre un lien de synchronisation:

- Pour EVT passé faire action: on s'intéresse à la première occurrence passée de EVT.
- Pour chaque EVT passé faire action: on s'intéresse à toutes les occurrences passées de EVT.
- Pour EVT futur faire action: on s'intéresse à la première occurrence futur de EVT.
- Pour chaque EVT futur faire action: on s'intéresse à toutes les occurrences futures de EVT.
- Pour chaque EVT (passé et futur) faire action: on s'intéresse à toutes les occurrences passées et futures de EVT.
- Pour EVT (passé ou futur) faire action: on s'intéresse à la première occurrence passée ou future de EVT.

Le chronogramme de la figure 2.3 résume les six cas énoncés.

L'instant t_0 représente l'instant de définition du lien de synchronisation. Il apparaît une dissymétrie entre la notion de passé et la notion de futur due au fait que la durée d'existence d'un lien au passé est toujours nulle. En effet la définition d'un lien au

passé se confond avec son exécution et sa destruction (représentée par la lettre D sur la figure), alors que les instants de définition et de destruction d'un lien au futur, sont disjoints dans le temps. Par ailleurs, cette destruction résulte soit d'un acte de l'utilisateur soit de la spécification même (cas de la première occurrence future).

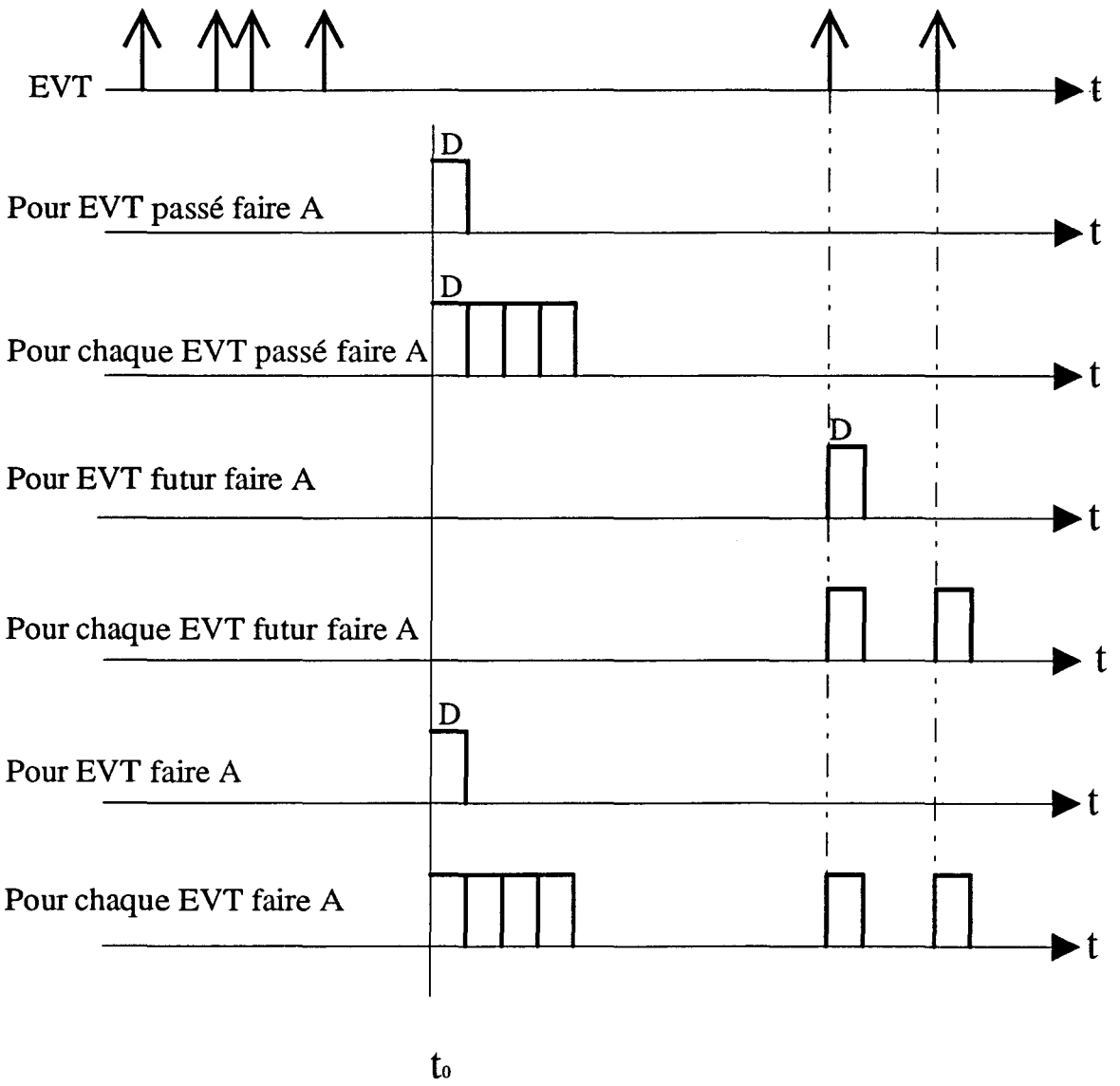


figure 2.3

Nous représentons dans ce qui suit ces diverses primitives dans le formalisme du Grafcet.

II-2-1 Pour EVT passé faire action.

Le franchissement de la transition 2 (figure 2.4) consacre l'établissement du lien de synchronisation. Si auparavant une occurrence de l'événement EVT est survenue (franchissement de la transition source numérotée 1 donc activation de l'étape 1), la transition 3 est franchie et l'action est ensuite déclenchée (l'étape 3 est pourvue d'une action impulsionnelle notée *). Dans le cas contraire on franchit la transition 4 (réceptivité $\overline{X_1}=1$, rappelons que dans le formalisme Grafcet, X_i signifie activité de l'étape i). Ce franchissement correspond à la destruction du lien de synchronisation. L'acquittement de l'occurrence mémorisée de l'événement EVT consacre le franchissement de la transition puits 5 qui a pour réceptivité ACQ.

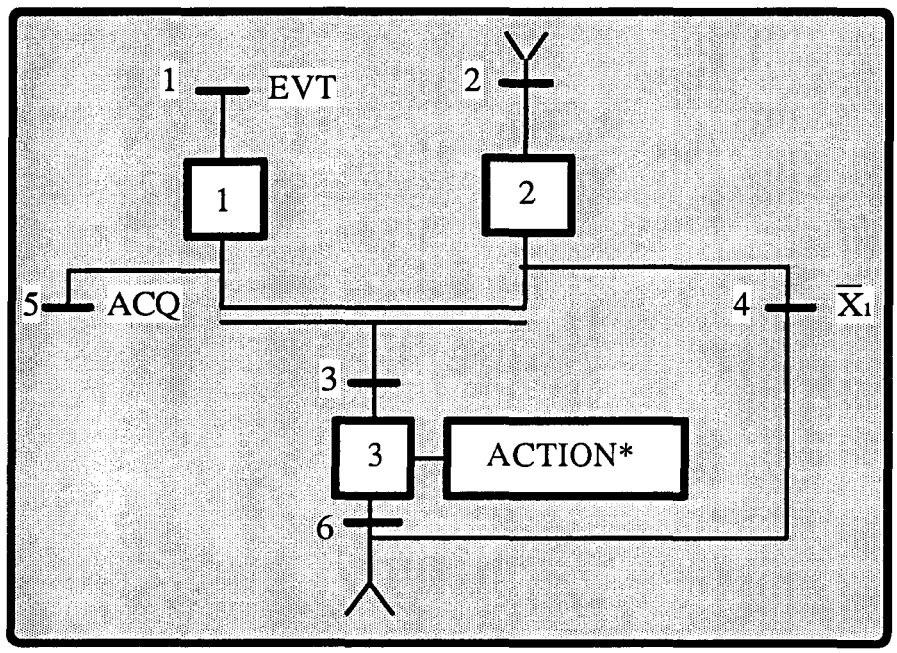


figure 2.4

II-2-2 Pour chaque EVT passé faire action.

La variable Noc représente le nombre d'occurrences comptabilisées de l'événement EVT. Elle est initialisée à Zéro (étape 1 en figure 2.5). Pour chaque occurrence de EVT, l'étape 2 est activée, l'action impulsionnelle correspondante est exécutée (Noc est incrémentée d'une unité), suivie d'une activation de l'étape 3.

Une fois le lien de synchronisation créé (étape 4 active), deux cas peuvent se présenter selon la valeur prise par la variable Noc:

- pour Noc nulle (pas d'occurrences mémorisées), le lien est immédiatement détruit.

- pour Noc strictement positive (des occurrences passées on été mémorisées), l'action est déclenchée (étape 5 active) et la variable Noc est actualisée (décrémentée à l'étape 6):

  - *si Noc est nulle, le lien est détruit.*

  - *si Noc est positive, les étapes 3 et 4 sont activées de nouveau pour examiner les occurrences restantes.*

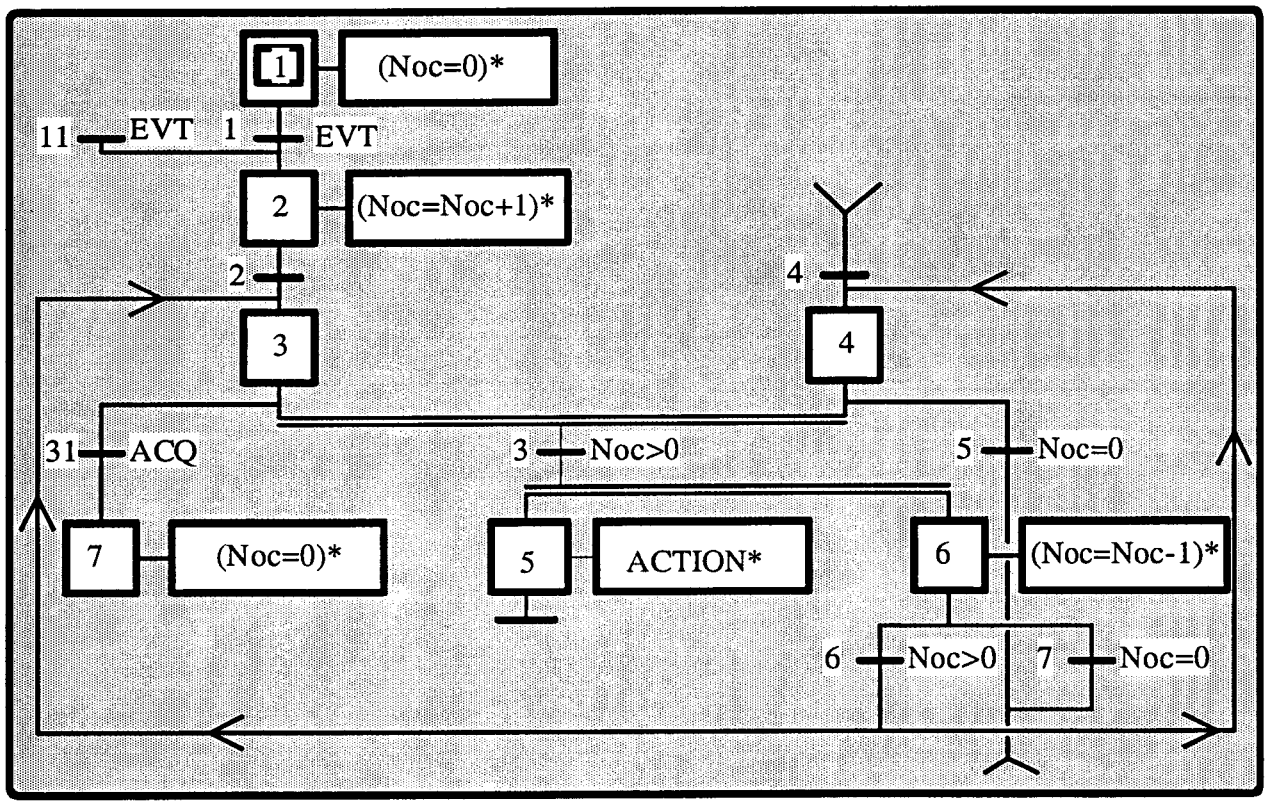


figure 2.5

En outre, lorsqu'il y a acquittement (transition 31 franchie), il s'agit d'acquitter toutes les occurrences mémorisées (Noc=0). En effet, dans l'approche Ladet, les occurrences sont appréhendées par groupe plutôt que d'une façon individuelle.

II-2-3 Pour EVT futur faire action.

Tant que le lien de synchronisation n'a pas été crée (transition 1 de la figure 2.6 non franchie), aucune occurrence de EVT n'est comptabilisée (présence de la condition X_1 dans la réceptivité associée à la transition 2). Lorsque le lien est établi et dès que surgit une occurrence de EVT, l'étape 2 est activée. L'action est alors exécutée dès le franchissement de la transition 3.

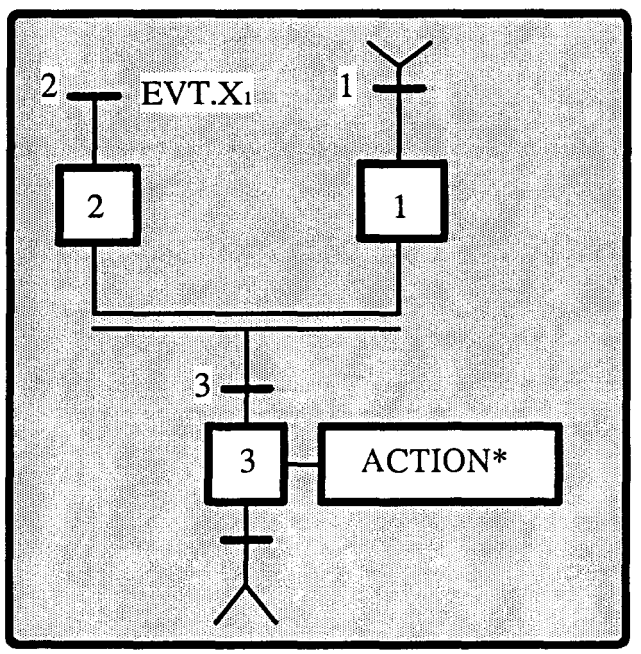


figure 2.6

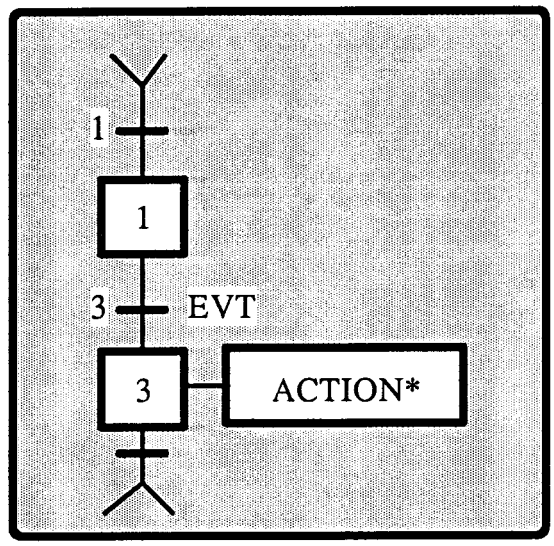


figure 2.7

La représentation précédente peut se simplifier, en tenant compte du fait que le Grafcet est réceptif à l'occurrence de EVT à partir de l'activation de l'étape 1. On obtient alors le Grafcet de la figure 2.7.

II-2-4 Pour chaque EVT futur faire action.

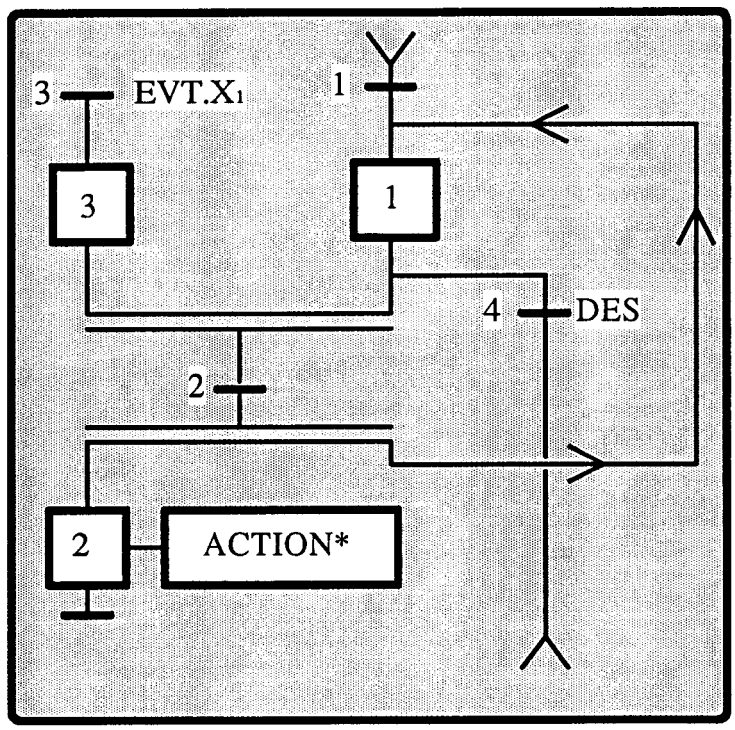


figure 2.8

Cette représentation (figure 2.8) diffère peu de la précédente sinon qu'elle introduit un arc de retour qui permet de réactiver l'étape 1 pour considérer toute nouvelle occurrence de EVT. La transition 4 de réceptivité DES correspond à la destruction du lien de synchronisation à la demande soit de l'utilisateur soit d'un autre Grafcet.

II-2-5 Pour EVT passé ou futur faire action.

Cette représentation (figure 2.9) est identique à celle de la première forme, mis à part le fait que la transition 4 de réceptivité $\overline{X_1}$ a été supprimée afin de garantir que l'étape 1 reste active, même s'il n'y a pas d'occurrences passées. L'acquittement représenté par la transition 4 de réceptivité ACQ n'a de sens que pour les occurrences passées.

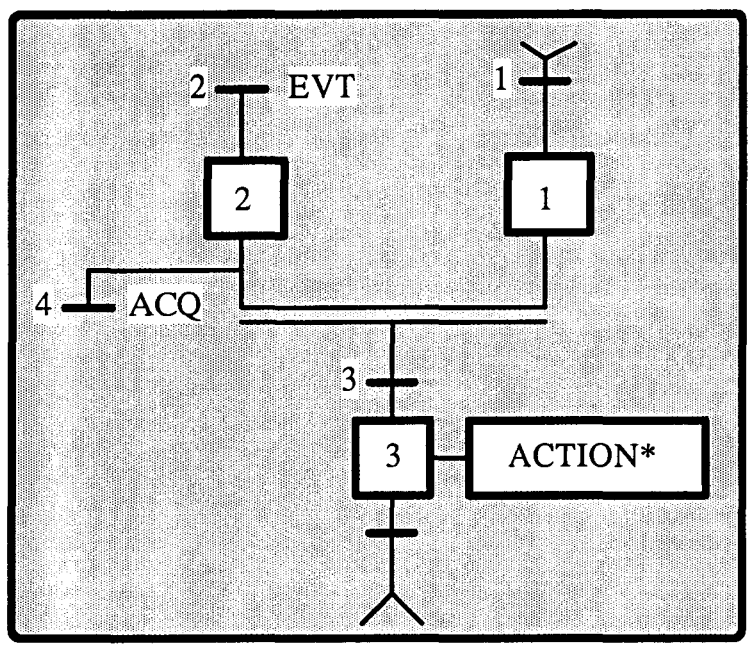


figure 2.9

II-2-6 Pour chaque EVT passé et futur faire action.

La description par Grafcet de la sixième et dernière primitive est la fusion des représentations de la seconde et de la quatrième primitive exposées précédemment. L'ajout de la condition $\overline{X_7}$ dans la réceptivité associée à la transition 11 de la figure 2.10 permet de supprimer la comptabilisation des occurrences futures de EVT dans la partie réservée aux occurrences passées, ceci pendant l'exécution du lien. En outre, la présence de la réceptivité X_7 dans la réceptivité associée à la transition 8, permet de ne comptabiliser que les occurrences futures pendant l'exécution du lien (étape 7 active).

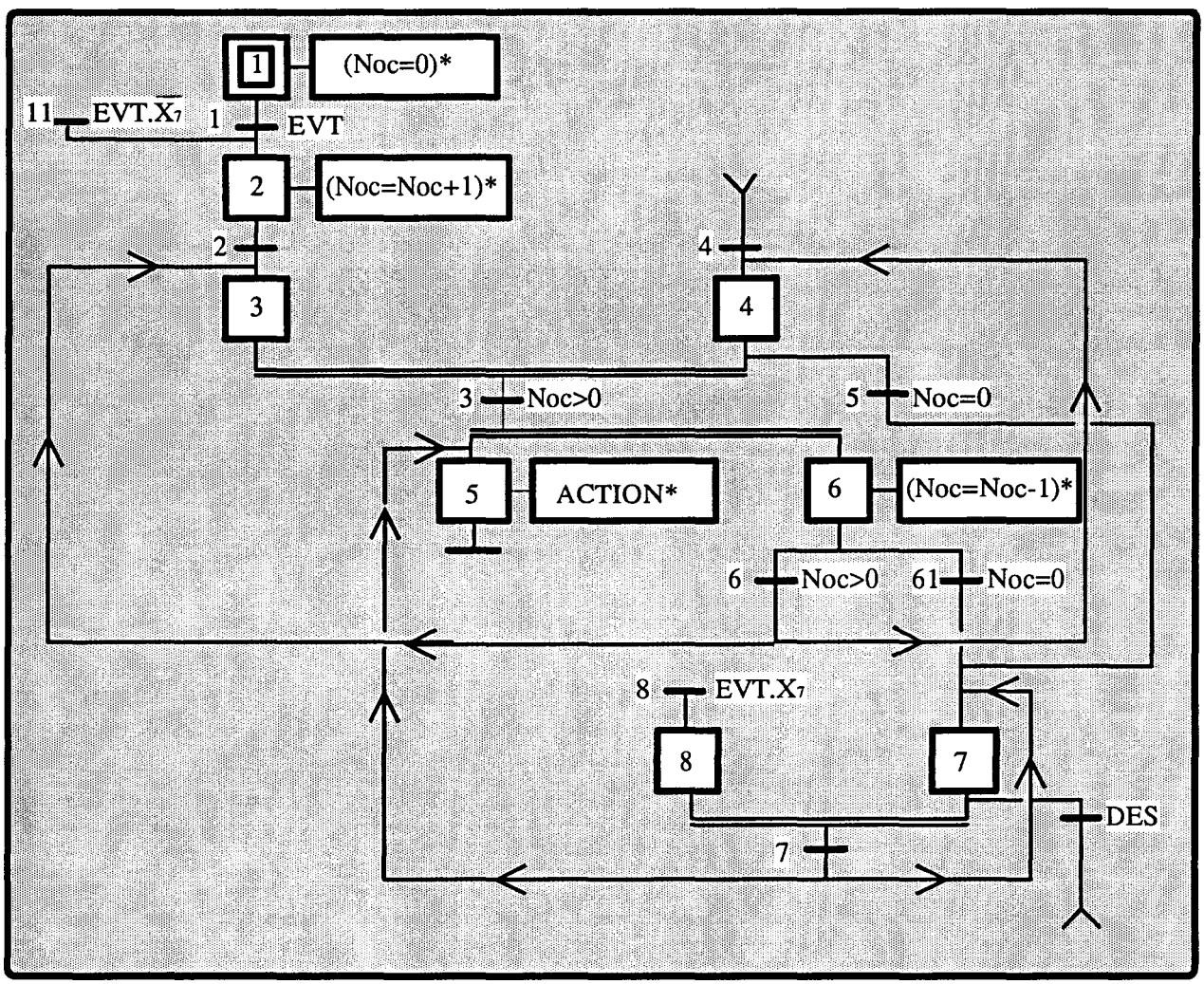


figure 2.10

II-3 Les conditions.

Dans la définition des liens de synchronisation, il est parfois nécessaire d'intégrer des conditions. En effet, on a l'habitude de traduire l'évolution dans le temps d'un système à commander non seulement avec les événements mais aussi avec les conditions.

Si les changements d'état des objets composant la commande et le procédé peuvent se traduire en termes d'événements, l'état à un instant t peut se traduire par un ensemble de valeurs traduisant des conditions.

L'utilisateur s'intéresse:

- soit à l'état dans lequel se trouvait ou se trouvera l'application lorsque se produisent les occurrences de l'événement.
- soit à l'état de la commande et du procédé au moment ou le lien de synchronisation est créé.

Ceci conduit à deux cas:

- pour un groupe d'occurrences, il s'agit de créer un lien de synchronisation qui ne considère qu'un sous-ensemble du groupe d'occurrences composé des occurrences qui se sont produites ou qui se produiront pendant que la condition est vraie.

  La syntaxe générale retenue est:

  Pour (chaque) <événement> (passé, futur) si <condition> faire <ACTION>.

- pour un groupe d'occurrences d'un événement, il s'agit de créer un lien si une condition est vraie. Le lien reste valide tant que la condition reste vraie.

  La syntaxe générale retenue est:

  Si <condition> Pour (chaque) <événement> (passé, futur) faire <ACTION>.

Le premier cas de figure a été traité par Ladet [LAD82]. La représentation par Grafcet des six formes évoquées précédemment, consiste à ajouter, pour chaque

réceptivité contenant la variable EVT, la variable condition notée C. Tout se passe comme si on créait un nouveau événement NEVT = EVT.C.

Quant au second cas, nous détaillons dans ce qui suit ce concept pour chacune des six primitives de liens de synchronisation.

II-3-1 Condition sur lien de synchronisation.

Les figures 2.11 et 2.12 ci-après résument les deux cas que l'on peut rencontrer en pratique. Ceux-ci sont obtenus en fonction:

- des instants t_0 et t_1, représentant respectivement le passage de la condition C à vrai, et son passage à faux.
- des instants pour lesquels se produisent les occurrences futures.

1. Pour la première primitive, les deux chronogrammes (figures 2.11 et 2.12) sont facilement compréhensibles. En effet la première occurrence de EVT étant mémorisée, dès que la condition C prend la valeur vrai (à t_0), le lien événement-action est créé et l'action est exécutée.

2. Pour la seconde primitive et dans les deux chronogrammes, toutes les occurrences passées de EVT sont mémorisées et dès que la condition C devient vraie, on exécute l'action pour chacune des occurrences. Le lien est détruit aussitôt après sa création.

3. Pour la troisième primitive:

   - dans le chronogramme de la figure 2.11, le lien de synchronisation est défini en t_0 mais la condition C repasse à faux à l'instant t_1. Par conséquent, le lien est détruit bien qu'aucune occurrence future ne survienne.
   - en revanche, pour le chronogramme en figure 2.12, la première occurrence survient alors que la condition C reste vraie. L'action est donc accomplie.

4. En ce qui concerne la quatrième primitive:

   - pour la figure 2.11, il n'y a pas d'activation de l'action car le lien est détruit à l'instant t_1 (C passe à faux) alors que n'est survenue aucune occurrence.

- en revanche dans le second chronogramme, il y a activation pour les deux occurrences futures survenant entre t_0 et t_1.

5. Pour la cinquième primitive, le résultat est le même pour les deux chronogrammes. L'action est demandée pour la première occurrence passée ou future dès que la condition prend la valeur vraie. S'il n'y a pas d'occurrence passée, l'action est demandée pour la première occurrence future pendant que C est vraie. Le lien est détruit dès que C passe à faux.

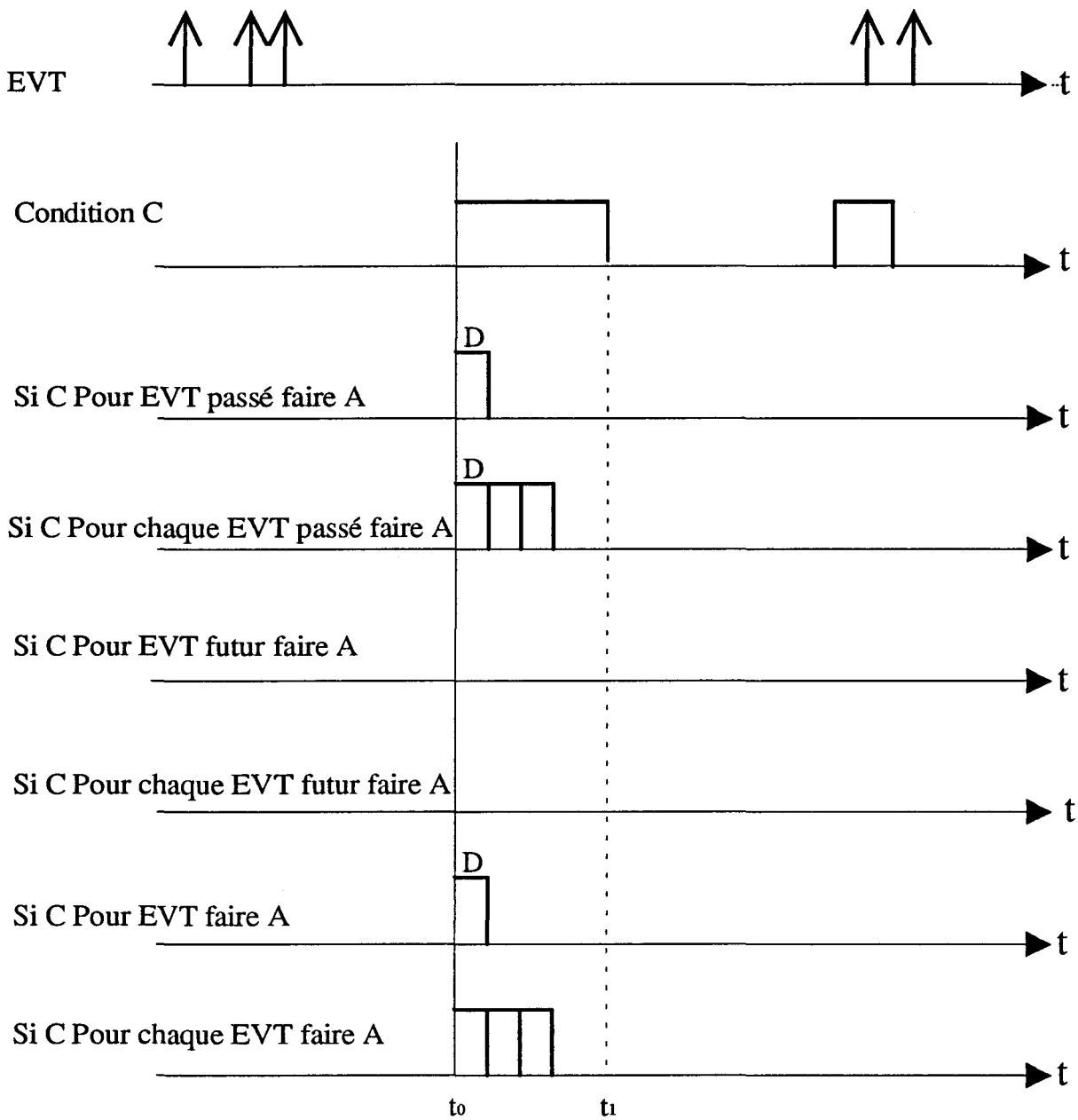


figure 2.11

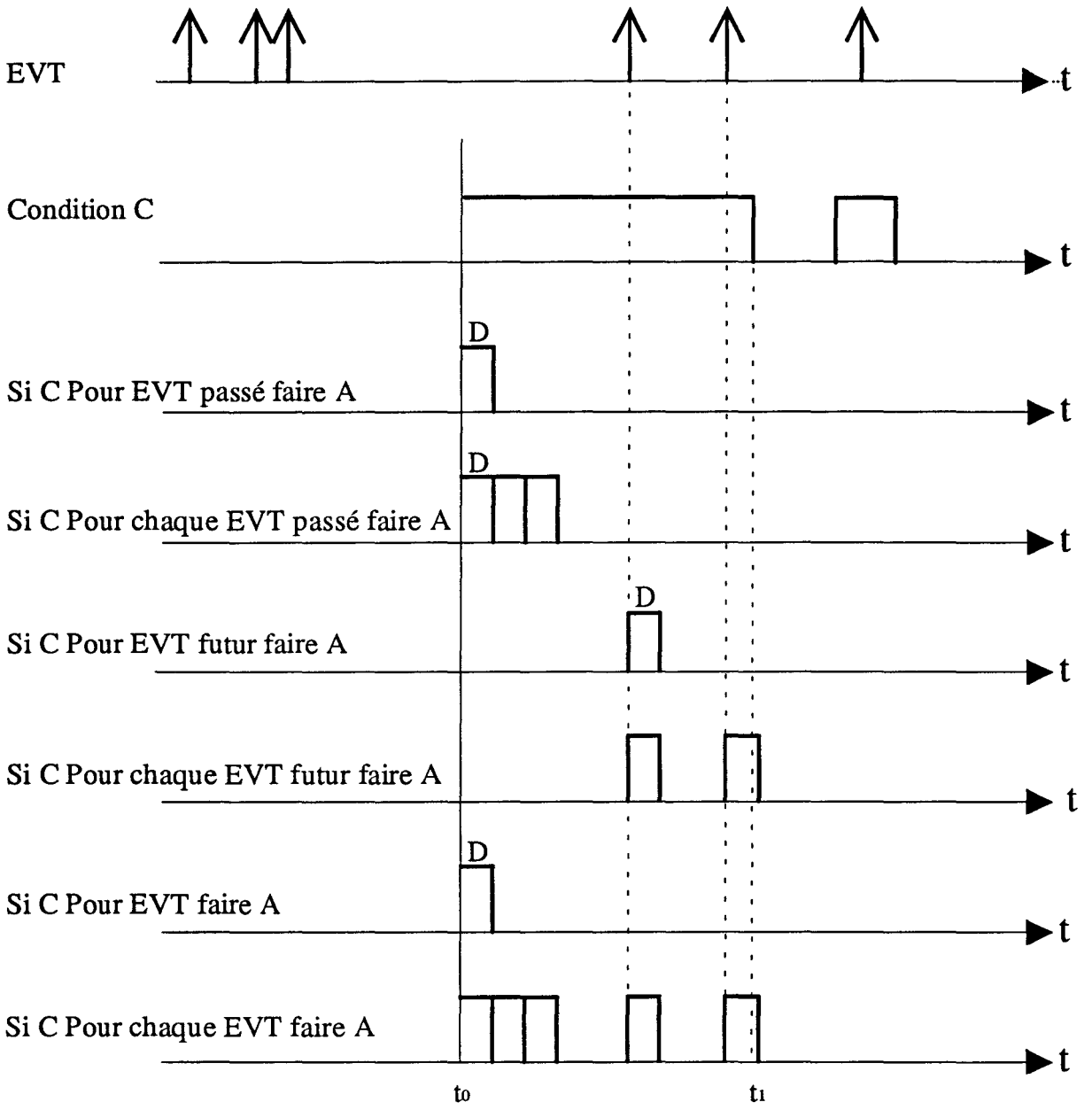


figure 2.12

6. La dernière forme révèle deux cas de figures:

   - pour le premier chronogramme, nous obtenons le même résultat que pour la seconde primitive, ceci car la condition C repasse à faux alors qu'aucune occurrence future n'apparaît.

   - l'activité, dans le second cas de figure, s'accomplit pour toutes les occurrences passées et pour les deux occurrences futures survenant entre t_0 et t_1.

La dissymétrie, entre un lien au passé et un lien au futur, constaté dans les liens de synchronisation qui n'intègrent pas de condition, persiste lorsque l'on intègre ces dernières. Le lien au futur s'établit dès que la condition devient vraie et reste vivant tant que cette même condition n'est pas passée à faux. Par contre un lien au passé est créé dès que la condition passe à vrai et est détruit aussitôt.

II-3-2 Représentation par Grafcet.

Pour les deux premières primitives, il suffit d'ajouter une transition de réceptivité C suivie d'une étape à placer en amont de la transition 2. Ce qui donne par exemple pour la première primitive la figure 2.13.

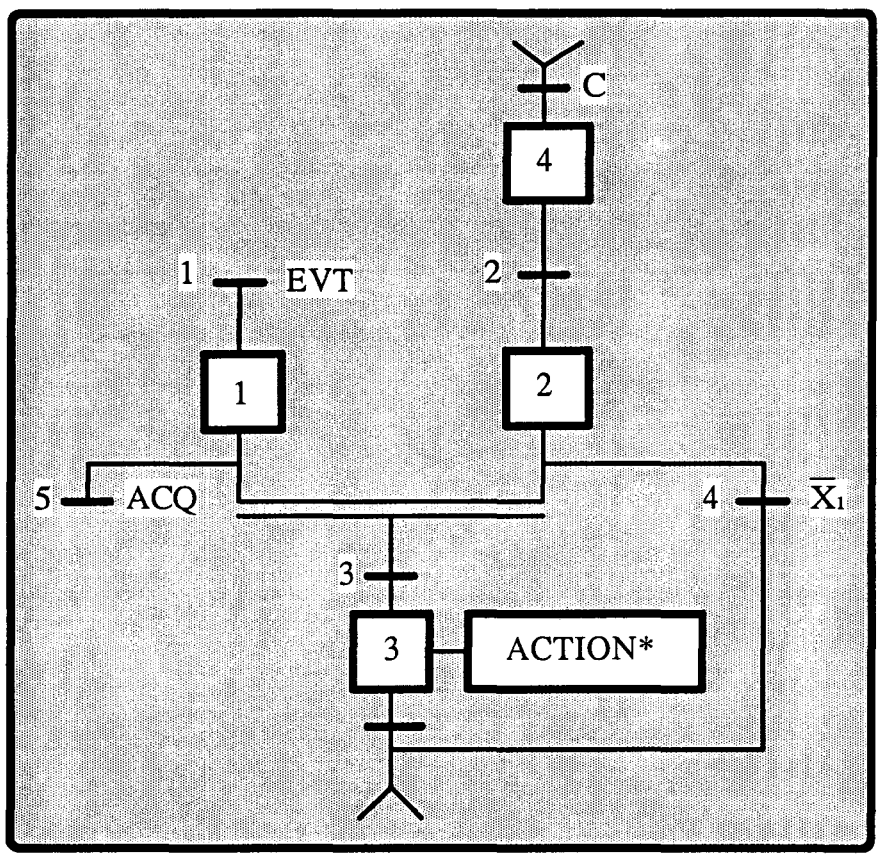


figure 2.13

Pour les trois primitives suivantes, on ajoute le même doublet que précédemment, ainsi qu'une transition de réceptivité $\overline{C}$ après l'étape de création du lien. Ceci permet de détruire le lien si la condition passe à faux. La figure 2.14 illustre cette situation dans le cas de la quatrième primitive.

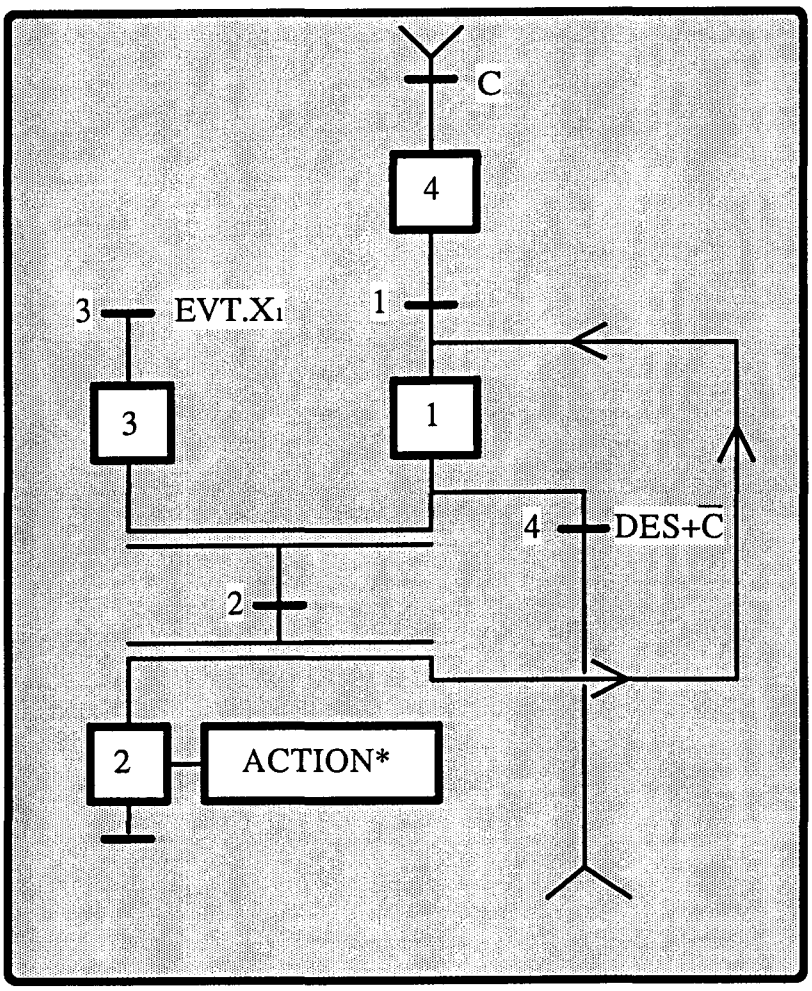


figure 2.14

Pour la dernière primitive, on ajoute le même doublet, au même endroit que précédemment, ainsi qu'une transition de réceptivité $\overline{C}$ à la sortie de la première étape traitant les occurrences futures. Ceci permet de détruire le lien dans le cas où la condition repasse à faux.

II-3-3 Exemples.

Les six primitives de gestion des liens de synchronisation événement-action, avec intégration éventuelle de conditions (soit sur l'événement soit sur le lien), constituent les primitives suffisantes pour spécifier toutes sortes de synchronisation. Les exemples ci-après illustrent quelques situations courantes.

1. *Autorisation de franchissement de T_1 après franchissement de T_2.*

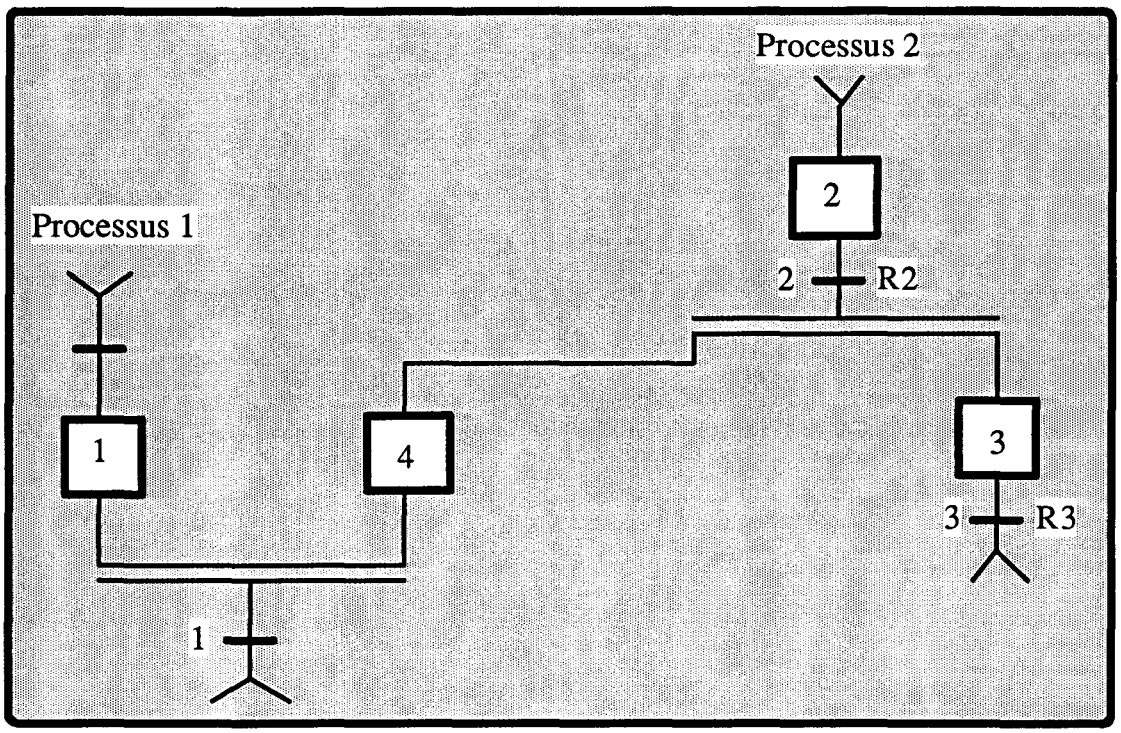


figure 2.15

En intégrant les événements et les liens de synchronisation entre les deux processus P1 et P2, on peut représenter le Grafcet de la figure 2.15 par les Grafcets de la figure 2.16.

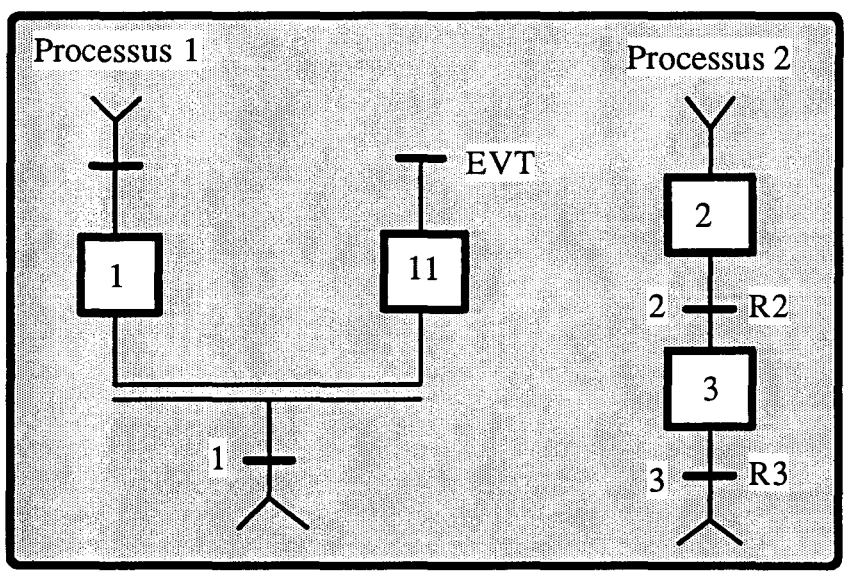


figure 2.16

La variable EVT s'identifie à la désactivation de l'étape 2 ($\downarrow X_2$). La relation entre le Processus P1 et l'événement EVT est représentée par le lien de synchronisation de la forme: Pour EVT passé ou futur faire $SUITE_1$ (celle-ci représente la suite du processus P1 après franchissement de la transition 1). On peut éventuellement intégrer une condition qui porte sur le lien si une condition apparaît dans la réceptivité associée à la transition 1.

2. *Contrôle de franchissement de T_2.*

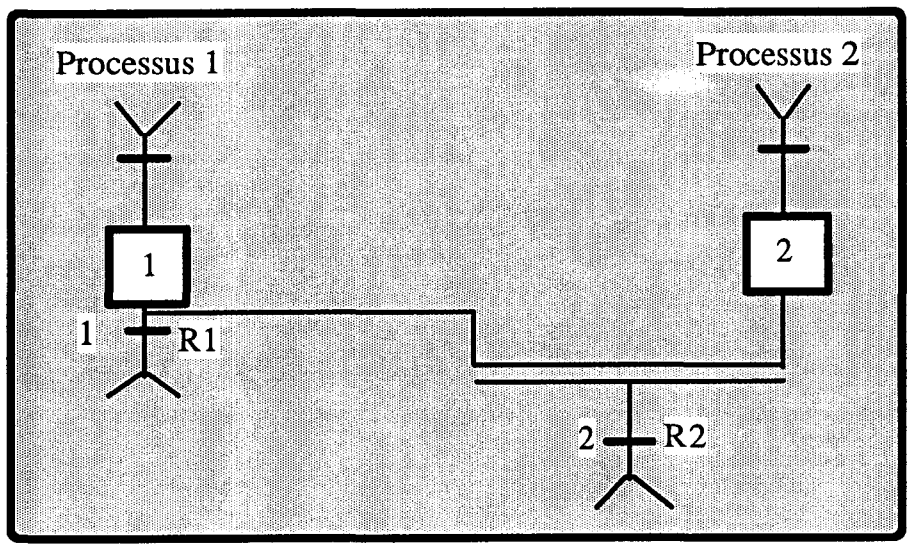


figure 2.17

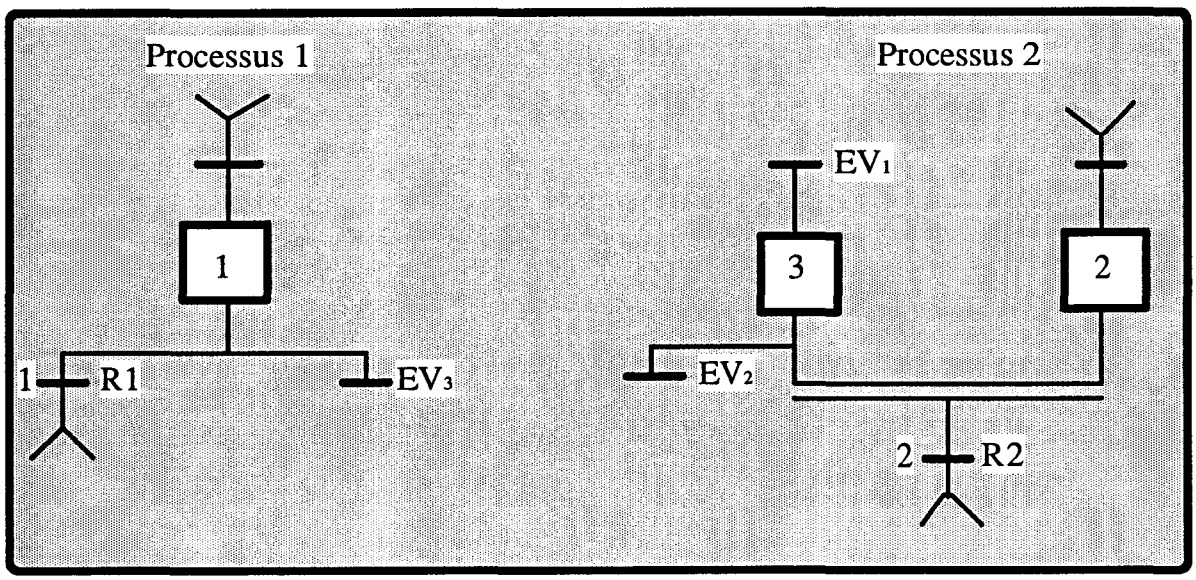


figure 2.18

La figure 2.17 présente un cas de contrôle de franchissement d'une transition 2 d'un Processus P2 par une étape numérotée 1 d'un processus P1. La synchronisation du Processus P2 par le processus P1 est représentée par l'événement EV1 (figure 2.18) symbolisant l'activation de l'étape 1 ($\uparrow X_1$). Ce lien de synchronisation est du type: Pour EV1 passé ou futur faire $SUITE_2$ (celle-ci représente la suite du Processus P2).

Le second événement soit EV2 a pour rôle d'acquitter l'événement EV1 (désactivation de l'étape 1 noté $\downarrow X_1$ après franchissement de la transition 1). Il faut aussi ajouter l'événement EV3 signalant la désactivation de l'étape 3 (franchissement de la transition 2: $\downarrow X_3$), au Processus P1. Ceci a pour effet de démarquer la place 1 du Processus P1. La liaison du Processus P1 avec EV3 est du type: Si C3 (X_1= vrai) Pour EV3 faire FIN. La relation liant le Processus P2 a EV2 est: Si C2 (X_3 = vrai) Pour EV2 faire FIN.

3. *Le rendez-vous.*

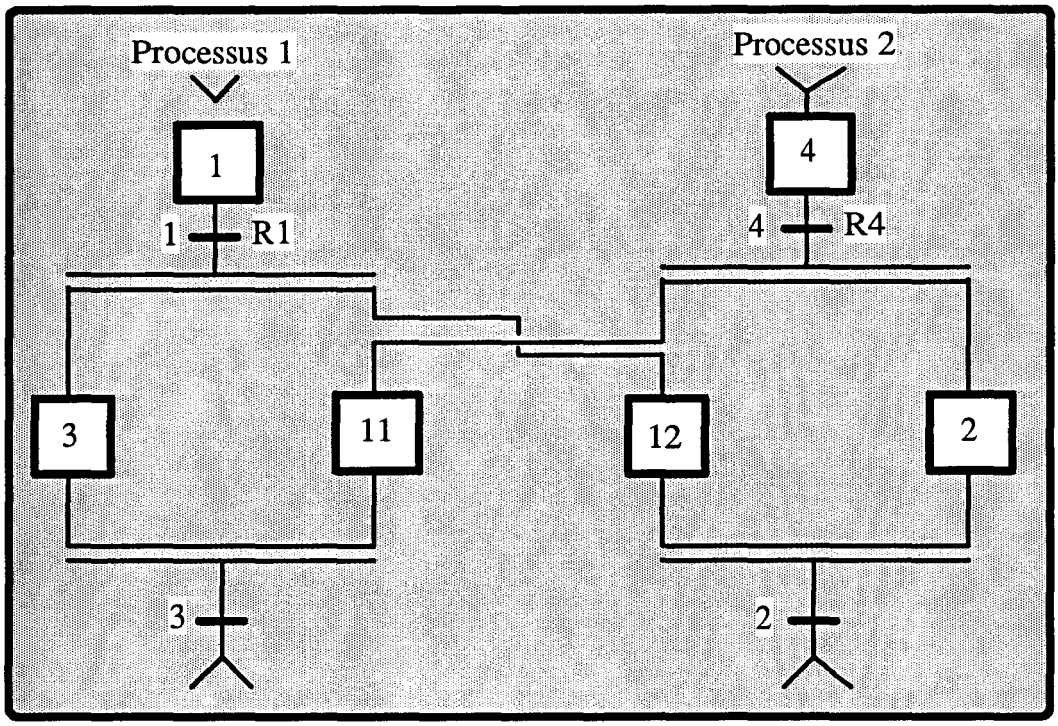


figure 2.19

La figure 2.19 illustre un rendez-vous entre deux Processus P1 et P2. Le premier processus arrivant au point de RdV (étape 3 ou 2 marquée) attend l'autre. Lorsque les deux processus sont prêts, les transitions 3 et 2 sont simultanément franchies. Chaque processus continue ensuite son chemin séparément.

On introduit donc deux événements EV1 et EV2 représentant respectivement le franchissement de la transition 4 ($\downarrow X_4$) et de la transition 1 ($\downarrow X_1$). Le lien de chaque processus PI avec l'autre est du type: Pour EV_I passé ou futur faire $SUITE_I$. La figure 2.20 illustre cette solution.

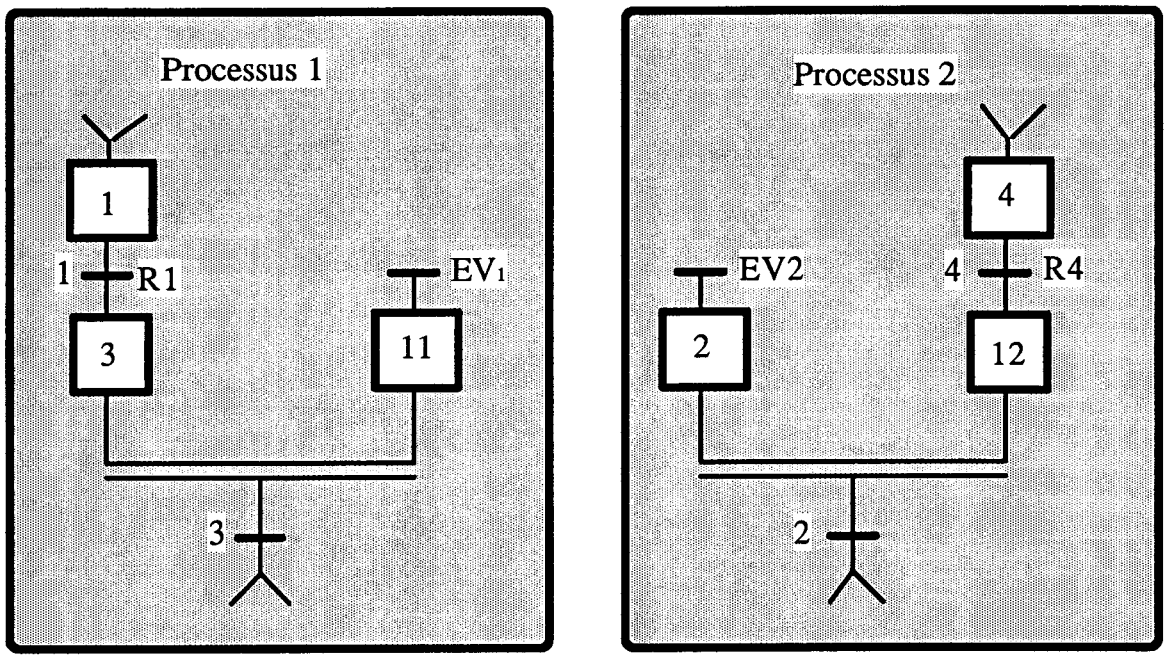


figure 2.20

III SPECIFICATION DE LA COOPERATION.

L'aspect fonctionnel d'un système temps-réel porte sur les fonctionnalités que le système doit satisfaire et sur l'information qu'il doit traiter. Il considère:

- les données sur lesquelles portent les traitements, leur provenance, leur destination, et leurs stockages intermédiaires
- les transformations qu'elles subissent, entre leur obtention en entrée et leur production en sortie.

Un outil de modélisation de l'aspect fonctionnel d'un système doit permettre de:

- représenter le travail de transformation que le système opère sur les données
- spécifier les processus qui transforment les données.

Les trois mécanismes de coopération de base, autres que la synchronisation pure, retenus dans le cadre de cette étude, sont la **compétition**, la **communication synchrone** et la **consultation**. Chaque mécanisme peut aboutir à plusieurs cas selon la politique adoptée pour la gestion des files de mémorisation. Ces mécanismes de gestion de stockages sont généralement considérés dans cette étude comme des processus **transformateurs de données**. Ils interagissent par ailleurs avec les autres processus qui constituent les ultimes transformateurs de données.

III-1 Compétition et partage de ressource.

III-1-1 Gestion par priorités.

La figure 2.21 illustre le partage d'une ressource R par trois processus P1, P2 et P3. L'activité et l'inactivité de l'étape 4 signifient respectivement ressource libre et ressource occupée (la ressource est initialement libre: l'étape 4 est une étape initiale).

La présence d'une marque dans une étape i, i= 1, 2, 3 correspond à la demande de la ressource par le Processus Pi. Quant aux étapes 11 à 1n, 21 à 2m et 31 à 3l, elles décrivent l'utilisation de la ressource respectivement par les processus P1, P2 et P3. Le franchissement de la transition T1n (T2m ou T3l) libère la ressource et dépose par conséquent une marque dans l'étape 4.

En outre, compte tenu du formalisme Grafcet, qui autorise le franchissement simultané de plusieurs transitions, il convient de régler les problèmes de conflits résultant de demandes simultanées d'accès à la ressource par les processus P1, P2 ou P3. Dans l'exemple de la figure 2.21 les conflits sont réglés en associant aux transitions T1, T2 et T3, les réceptivités internes $\overline{X}_2.\overline{X}_3$, $\overline{X}_3$ et 1. Ceci institue une hiérarchie de priorité P3 > P2 > P1.

Supposons que pendant l'utilisation de la ressource par le processus P3, parvienne une demande du processus P1 suivie de peu par une demande du processus P2. Une fois la ressource libérée par P3, elle est aussitôt réquisitionnée par P2. Entre temps, le processus P3 peut redemander la ressource. Cette dernière demande sera satisfaite dès que P2 a fini d'utiliser la ressource et ainsi de suite.

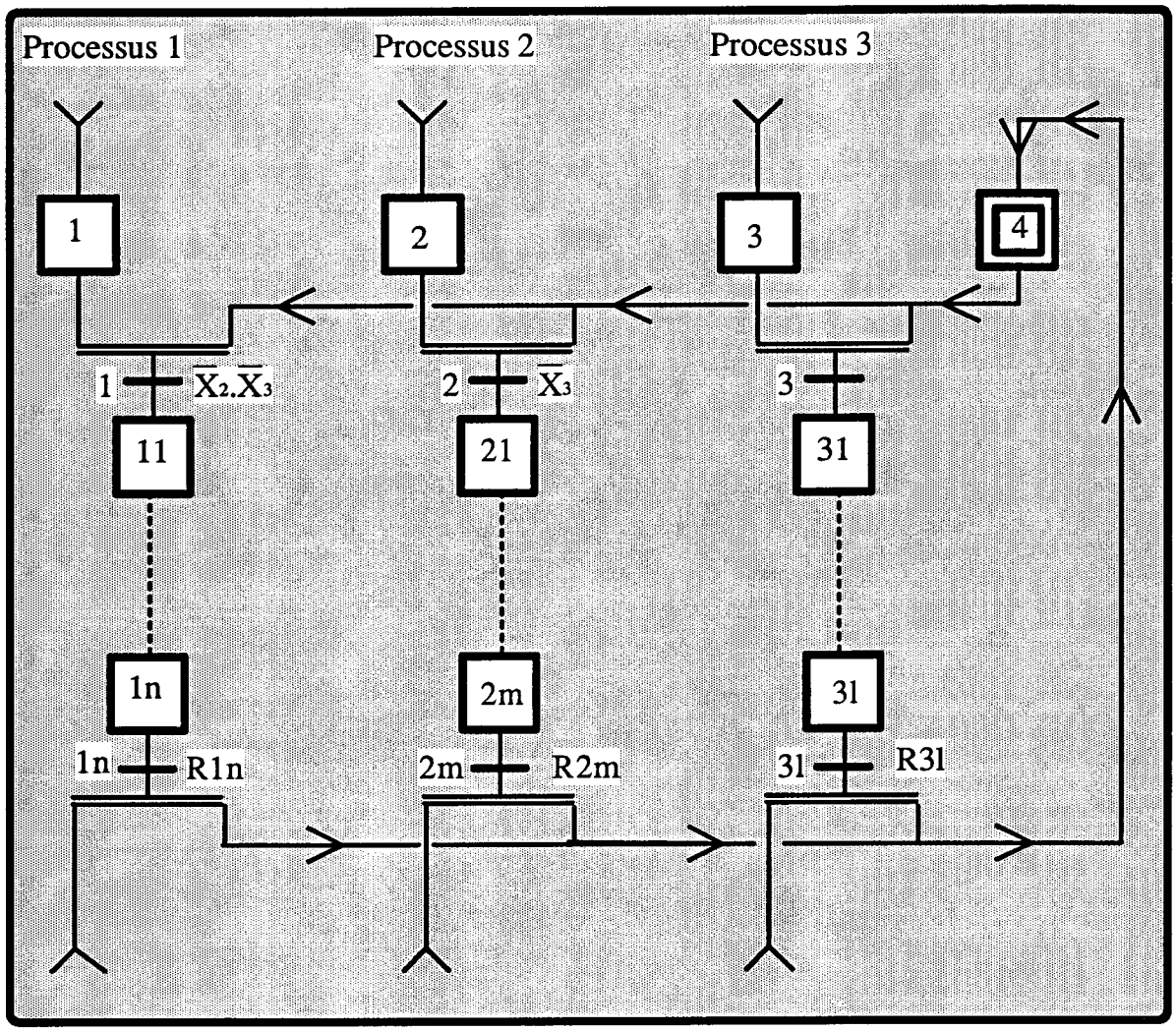


figure 2.21

III-1-2 Gestion en premier demandeur premier servi.

Ce type de situation conduit au blocage de la demande de P1 par des demandes entrelacées de P2 et P3. Pour éviter que ce genre de problème ne se produise, on peut avoir recours à une autre méthode de gestion des demandes.

La figure 2.22 propose une solution permettant d'éviter les blocages. Cette solution est du type premier demandeur premier servi. Il est, en outre, possible de mémoriser deux demandes lorsque la ressource est occupée.

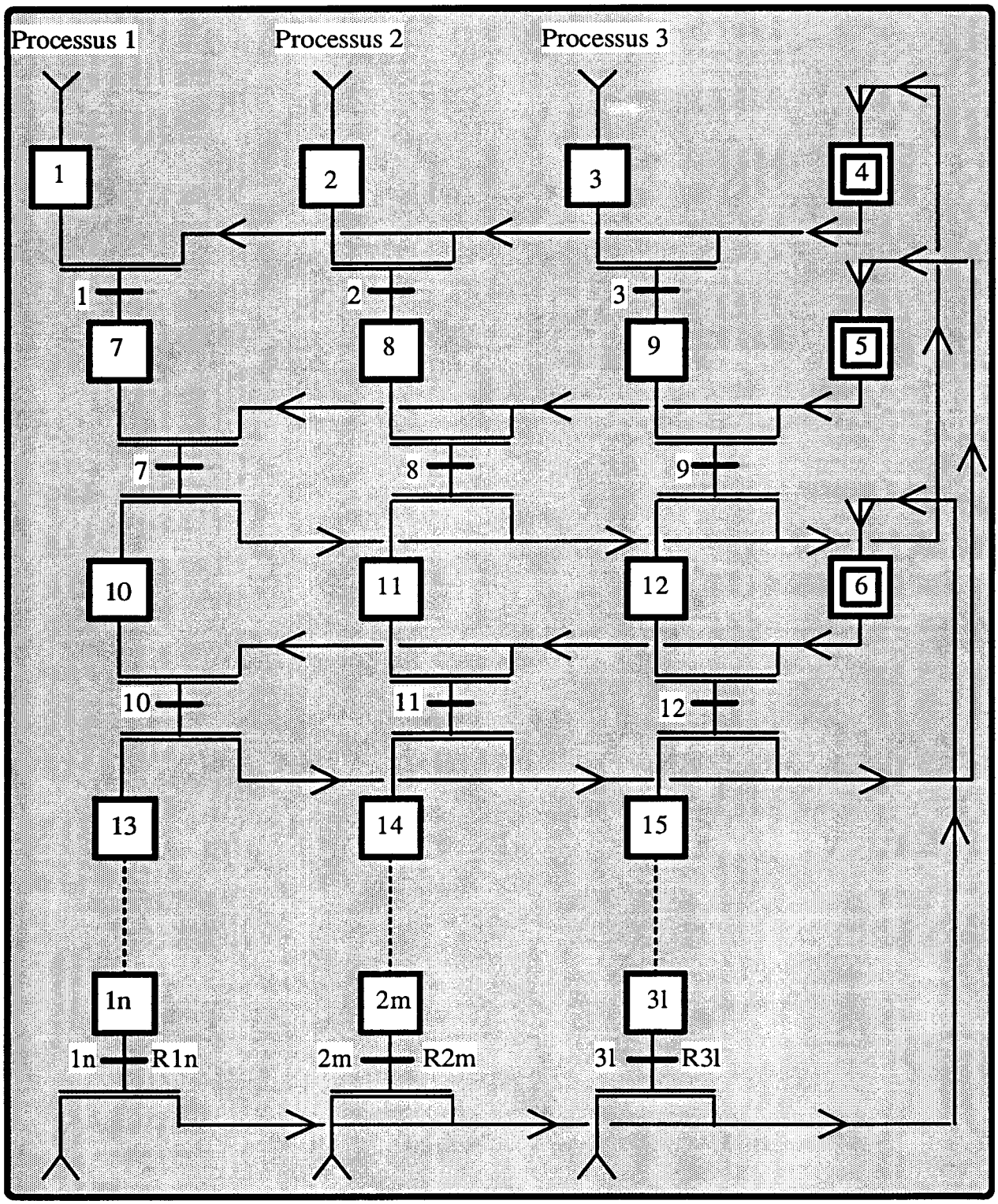

Processus 1
Processus 2
Processus 3
1
2
3
4
1
2
3
7
8
9
5
7
8
9
10
11
12
6
10
11
12
13
14
15
1n
2m
3l
1n
R1n
2m
R2m
3l
R3l

figure 2.22

Au stade de l'implantation, on fait correspondre à la gestion de la ressource r, un **processus Pr** dit Gestionnaire de la ressource et représenté par un Grafcet. La figure 2.23 illustre l'implantation d'une gestion par priorité des demandes de deux processus P1 et P2.

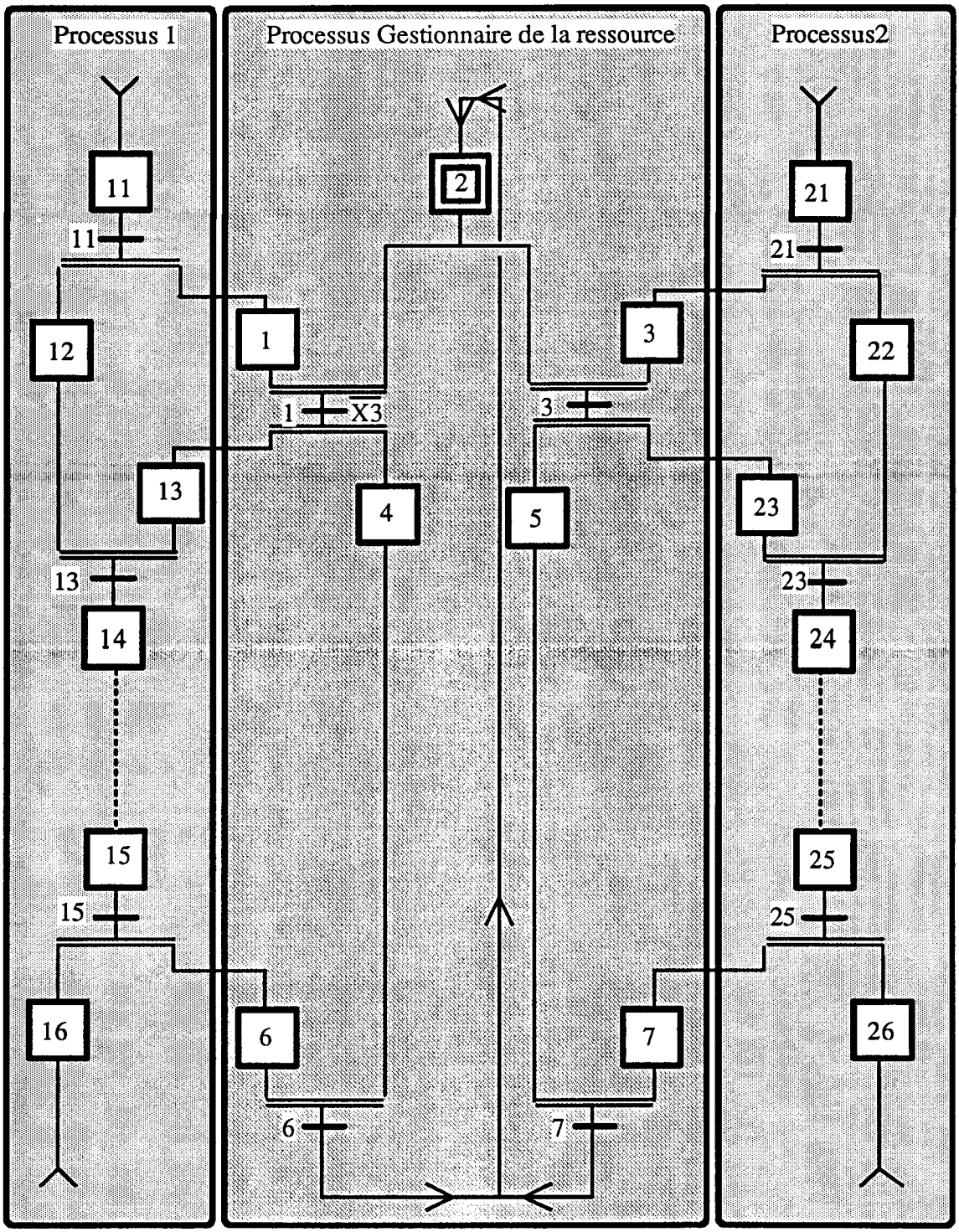


figure 2.23

III-2 Communication synchrone.

La communication synchrone est le cas où il y a communication de données entre deux processus s'accompagnant nécessairement d'une synchronisation. La seule synchronisation est assurée par la présence ou l'absence de données, ce qui est typique des systèmes pilotés par les données (data driven).

Le problème le plus classique dans ce type de communication est celui du producteur/consommateur que nous énonçons brièvement dans ce qui suit [CRO75]:

On considère deux processus, le producteur et le consommateur, qui se communiquent de l'information à travers une zone de mémoires. Cette zone à une capacité fixe de n messages de taille commune. La communication se déroule suivant les règles suivantes:

- exclusion mutuelle au niveau du message.
- le producteur ne peut placer un message dans le tampon si celui-ci est plein, le producteur doit alors attendre.
- le consommateur doit prélever tout message, une fois et une seule.
- si le producteur est en attente parce que le tampon est plein, il doit être prévenu dès que cette condition cesse d'être vraie. Il en est de même pour le consommateur avec la condition tampon vide.

La représentation par Grafcet nous amène à définir trois processus distincts: le producteur, le consommateur et un processus pour la gestion de la file des messages. Cette dernière peut être gérée de plusieurs manières. Nous avons retenu la gestion en consommation du premier arrivé (exemplaire le plus ancien) et la gestion en consommation du dernier arrivé (message le plus récent).

Un autre cas de communication synchrone, appelé **communication simple**, concerne l'interaction entre un producteur et un consommateur non séparés par une zone tampon. Le producteur place un message directement chez le consommateur sans se soucier de la disponibilité de la place.

III-2-1 Consommation premier arrivé.

La figure 2.24 présente une communication producteur/consommateur avec consommation du plus ancien message et conservation de 4 messages au maximum (étapes numérotées 6, 8, 9 et 10).

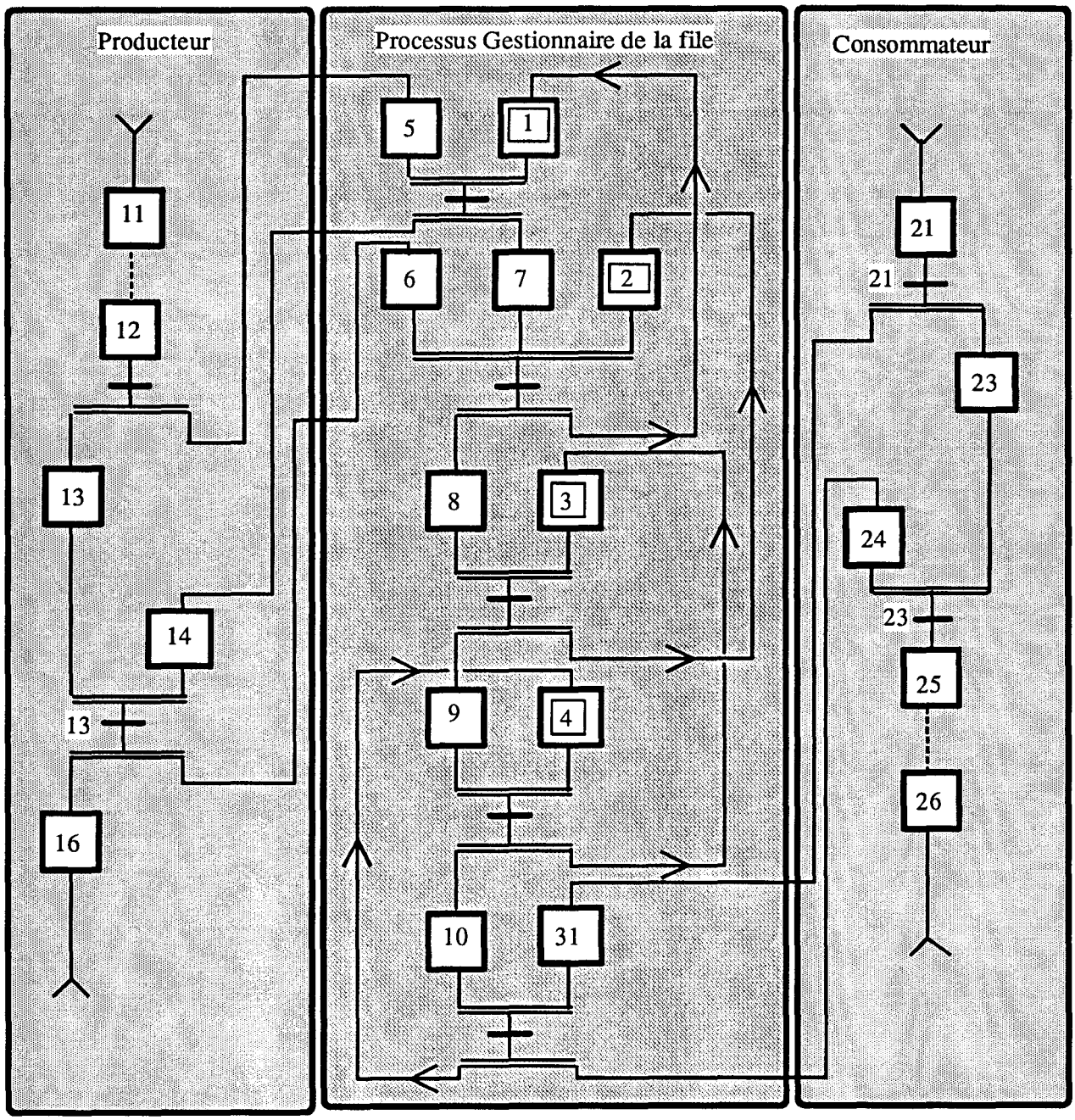


figure 2.24

Lorsque le producteur à produit un message (étapes 11 à 12), il demande s'il y a une place de libre pour déposer son message, ceci en marquant l'étape 5. Une fois qu'il reçoit le signal de disponibilité d'une place (étape 14 active), il place alors son message dans la première place de la file (étape 6).

Les étapes numérotées de 1 à 4 servent à faire avancer les messages d'un rang à un autre (FIFO). Le consommateur signale sa volonté de consommer en activant l'étape 31. Il attend ensuite que l'étape 24 soit marquée (celle-ci regroupe le message et la réponse en même temps). Une fois le message prélevé (étape 24 active), le processus de gestion des messages actualise sa file en avançant d'un rang les messages restants. Les messages sont donc consommés une seule fois

Pour le cas où il y a plusieurs producteurs et/ou plusieurs consommateurs, il faut assurer l'exclusion mutuelle entre producteurs et/ou entre consommateurs. La figure 2.25 illustre une représentation possible du cas de deux producteurs et d'un consommateur avec une file de quatre messages (étapes (7,71), 8, 9 et 10)

L'étape 1 permet de sélectionner un producteur. La demande passe ensuite dans l'étape 41 ou 42 et dès qu'il y aura une place de libre (étape 2 active), le processus reçoit l'autorisation (étape 14 ou 34 active). Il dépose alors son message dans la file (étapes 7 ou 71) qui évoluera de place en place suivant l'état de la file et de la consommation.

Un dernier, mais néanmoins important cas, concerne la relation entre un producteur (ou plusieurs) et de multiples consommateurs devant impérativement consommer les mêmes données. Contrairement au cas précèdent, pour lequel une donnée est nécessairement consommée par un seul processus, cette fois la même donnée doit être consommée par chaque consommateur.

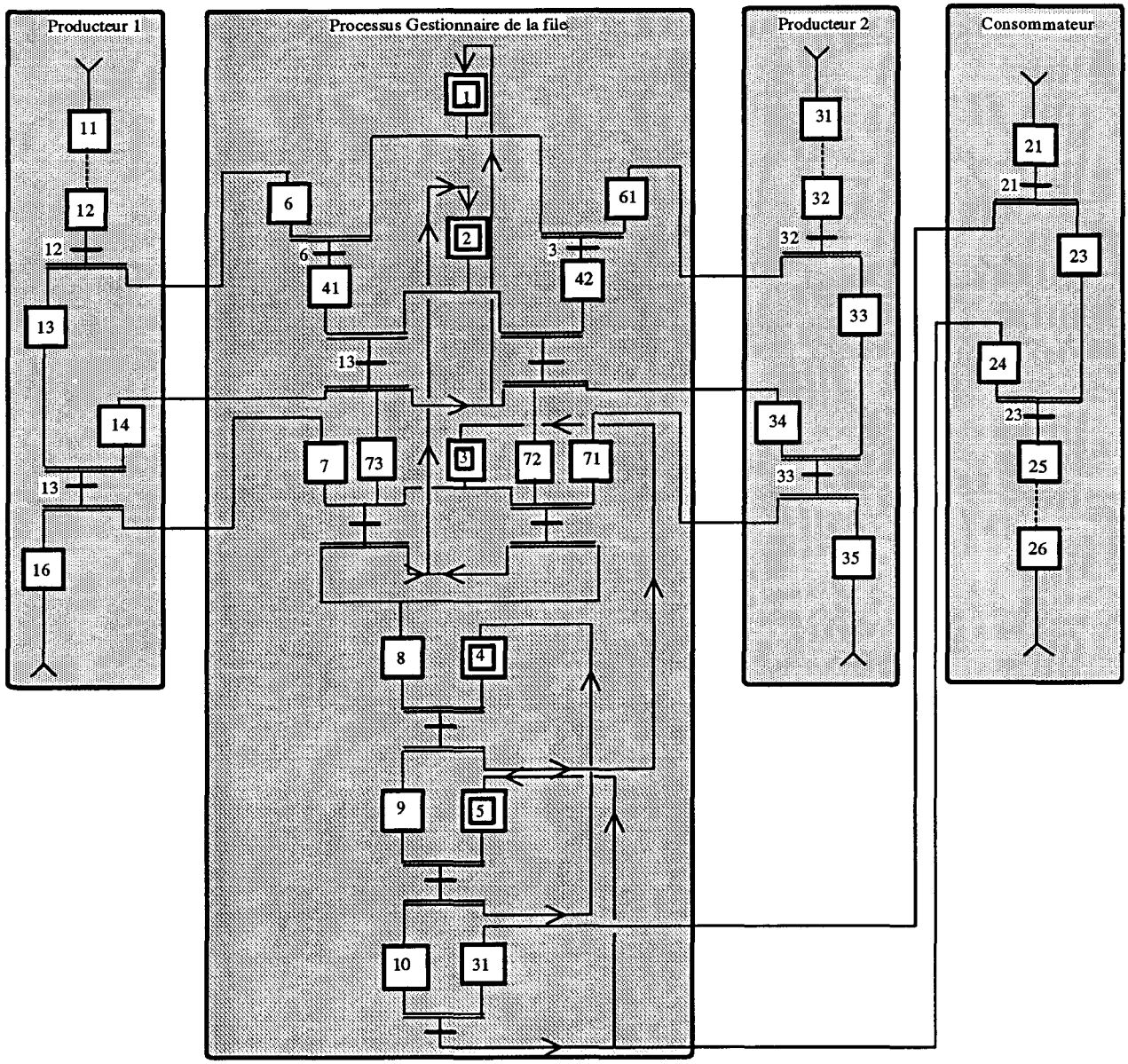


figure 2.25

La figure 2.26 présente le cas de deux consommateurs avec une file de 4 éléments gérée en FIFO. Pour exprimer ce problème, il suffit de dupliquer la file de messages en autant d'exemplaires qu'il y a de consommateurs (ici deux). Un message ne peut être mémorisé que si chacune des files possède une place de libre (étapes 21 et 22 marquées - étape 1 marquée). Le message est ensuite communiqué aux deux files (étapes 11 et 12) pour évoluer séparément dans chaque file en fonction des besoins de chaque consommateur.

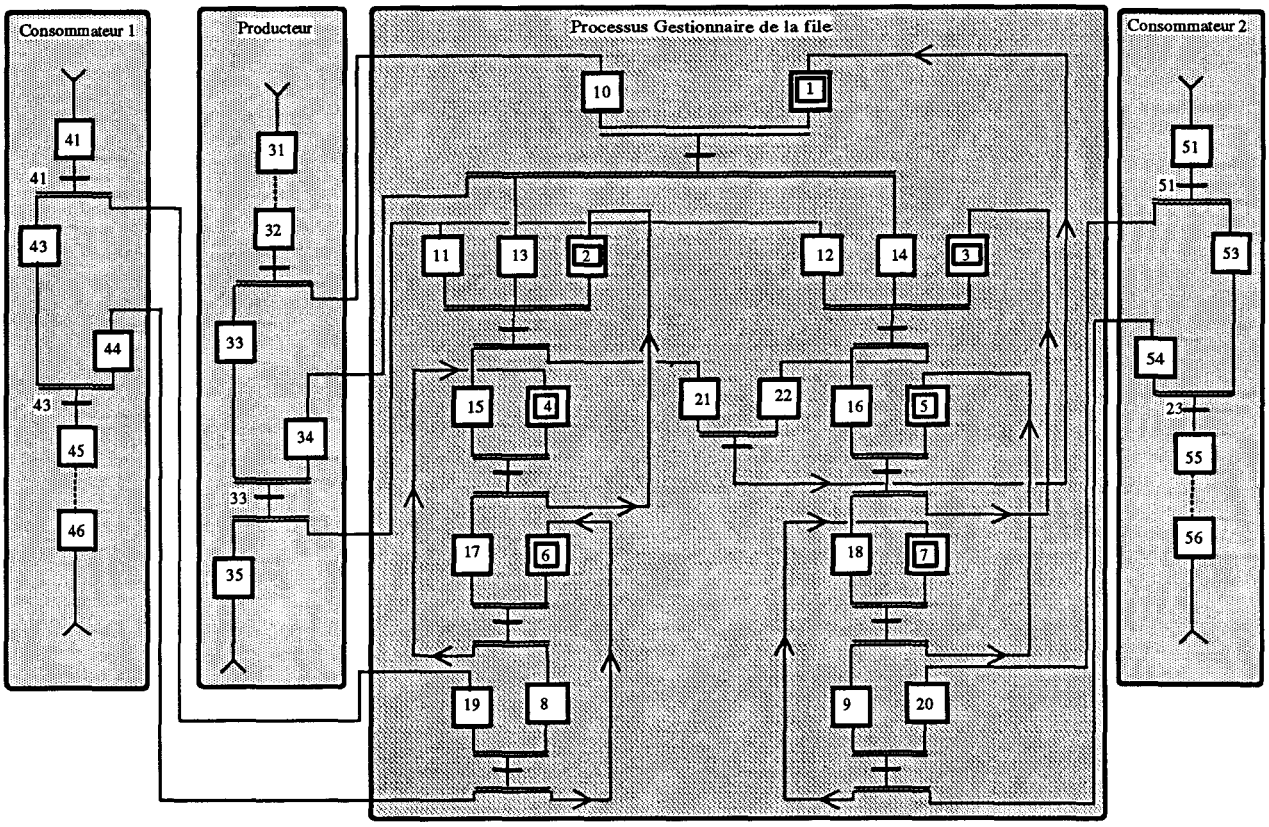


figure 2.26

III-2-2 Consommation dernier arrivé.

La figure 2.27 présente une communication producteur/consommateur avec consommation du plus récent exemplaire et mémorisation de quatre messages (étapes 6, 8, 9 et 10)

Les connexions du producteur et du consommateur avec le processus de gestion de la file sont les mêmes que dans le cas précédent (consommation premier arrivé). En effet, les transitions 6, 8, 9 et 10 munies de leurs réceptivités respectives 1, $\overline{X}_6$, $\overline{X}_6.\overline{X}_9$ et $\overline{X}_9.\overline{X}_{10}$, servent à fournir au consommateur le dernier message reçu. Ainsi, suivant la place occupée par le message fourni, il faut signaler à la place précédente, la libéralisation de la place, d'où l'ajout de quatre arcs orientés.

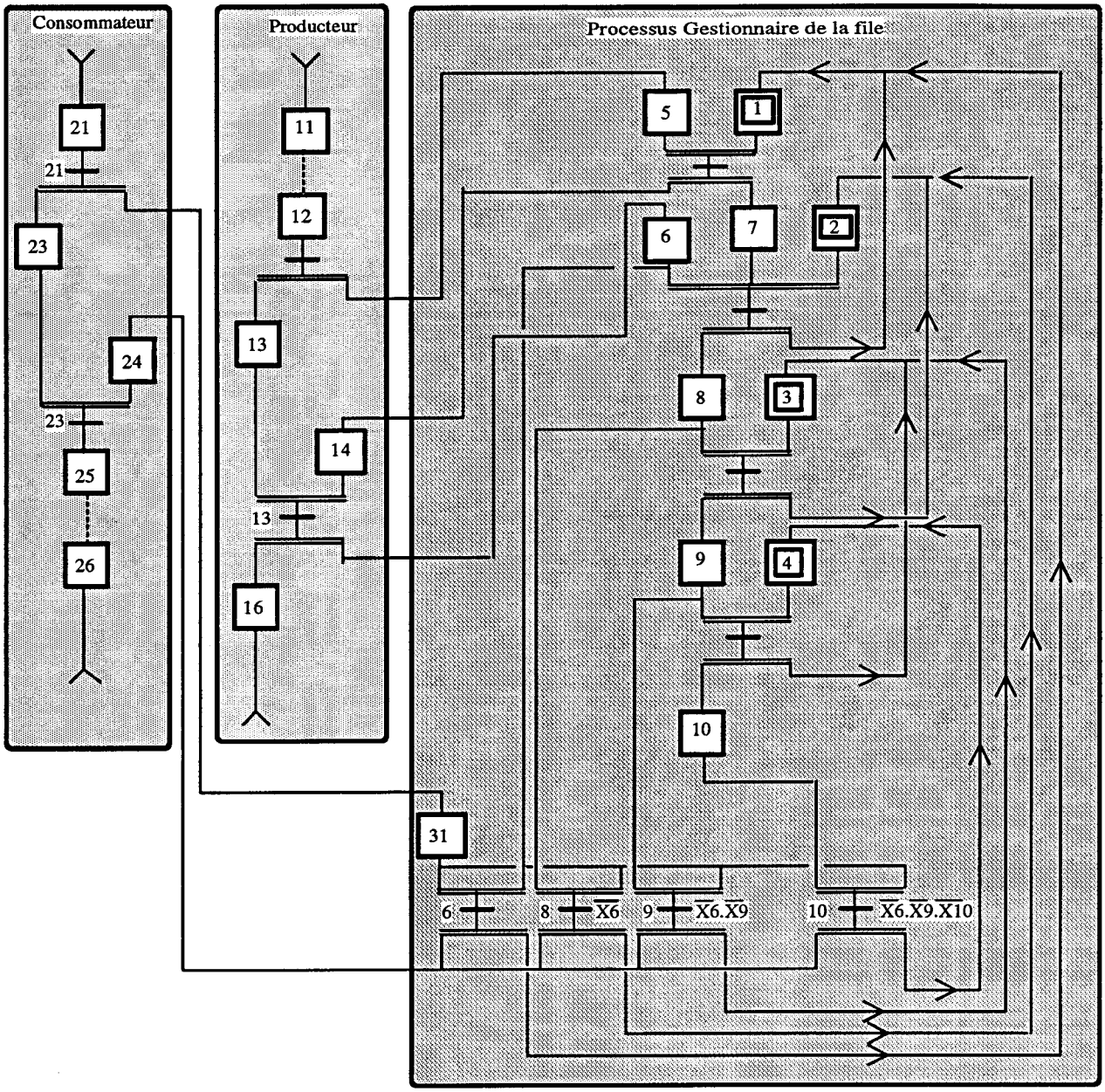


figure 2.27

III-3 Consultation.

Un troisième volet de la communication concerne la consultation de messages. En effet, considérons le cas où un processus est chargé de consulter, sans les consommer, des messages contenus dans une zone mémoire. De même que pour la relation producteur/consommateur, la consultation peut porter entre autres sur le plus ancien message ou sur le plus récent.

III-3-1 Consultation premier arrivé.

Dans la figure 2.28, pour la partie connexion avec le consommateur, les étapes 10 et 24 comme précédemment, représentent respectivement la demande et l'accord de consultation du message. Il s'en suit la remise du jeton (message) dans la place d'origine, soit de l'étape 9.

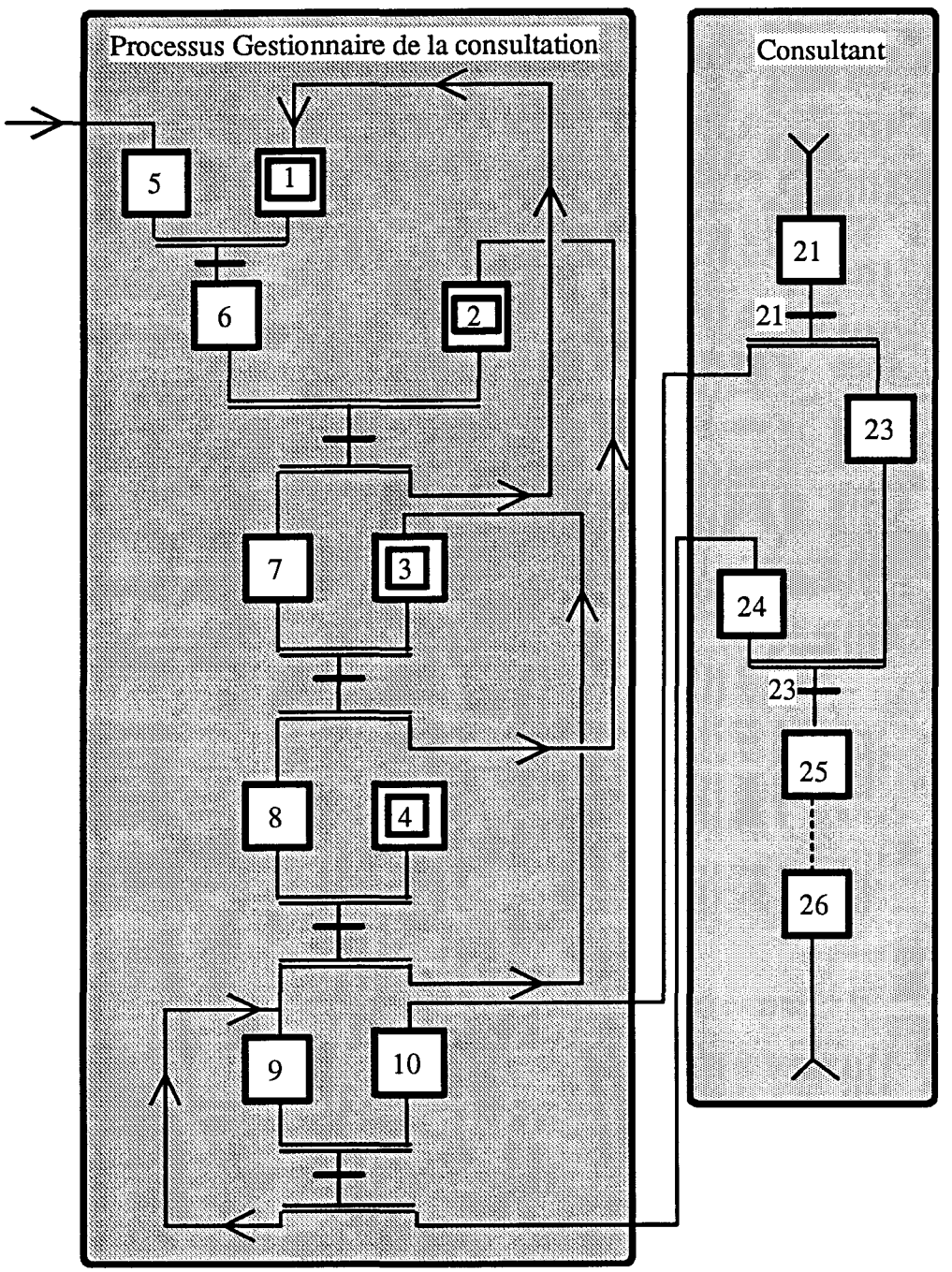


figure 2.28

III-3-2 Consultation dernier arrivé.

Pour certains processus, il peut être utile, voire nécessaire, de consulter plutôt le plus récent message que le plus ancien. Le consultant doit donc avoir la possibilité d'accéder au dernier message et le restituer après consultation. La figure 2.29 illustre ce cas.

Notons d'abord l'existence de transitions 6, 7, 8 et 9 auxquelles sont associées les réceptivités respectives 1, $\overline{X}_6$, $\overline{X}_6.\overline{X}_7$, $\overline{X}_6.\overline{X}_7.\overline{X}_8$. Celles-ci permettent de sélectionner le dernier message reçu. Dès franchissement des transitions citées, on réactive les places qui contenaient les messages (étapes 6, 7, 8 et 9), pour consultation ultérieure.

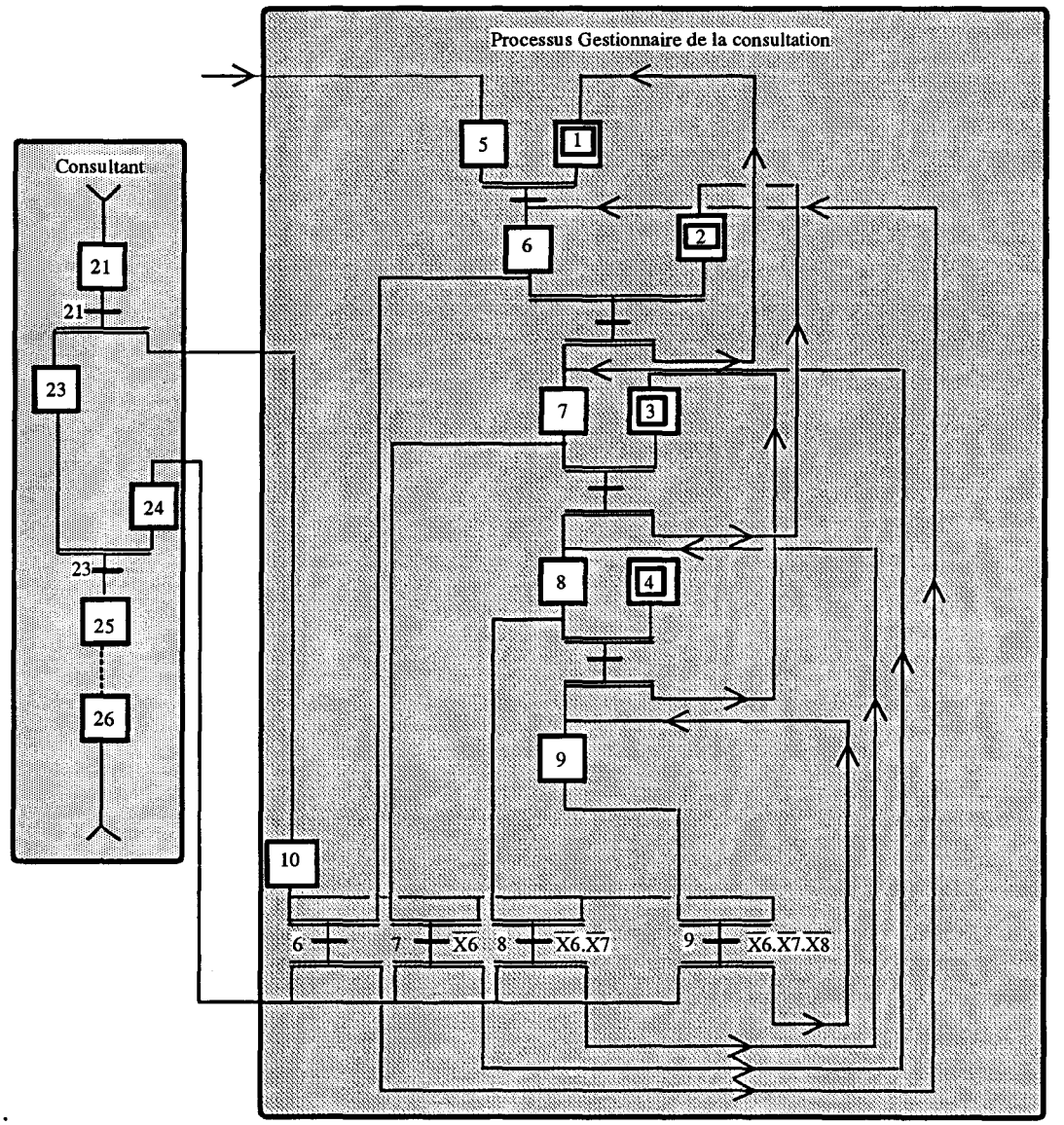


figure 2.29

CONCLUSION.

Les primitives de synchronisation énoncées constituent des outils de description pour toutes sortes de synchronisation. Toutefois l'approche proposée n'est pas utilisable dans le cas ou l'acquittement d'occurrences a lieu occurrence par occurrence.

Dans ce cas, et pour la démarche adoptée ici, le problème de l'acquittement occurrence par occurrence est pris en charge à un niveau supérieur de la commande.

Les modes de gestion des files d'attente dégagés dans les problèmes de communication et de compétition constituent des motifs ou blocs que l'on peut réutiliser dans d'autres applications.

En outre, il existe d'autres politiques de gestion notamment dans le domaine des procédés manufacturiers (gestion en priorité de la pièce ayant le temps d'usinage le plus court, ou le plus long..). Toutefois nous n'en voyons pas la nécessité pour le niveau de commande utilisé et le type d'applications envisagées .

L'utilisation du Grafcet coloré [AGA87] permet de réduire l'explosion combinatoire des Grafcets obtenus.

En toute rigueur, nous devons également spécifier l'**aspect informationnel**, c'est à dire spécifier les données et les événements, d'une part et les traitements associés a ceux-ci, d'autre part. La spécification des traitements est effectuée par un **langage textuel**. Un **dictionnaire** sert à définir les données et les événements [CHA92]. Ce dictionnaire sert, en particulier, à donner la signification des données accompagnant une place marquée comme par exemple le type ou les coordonnées d'une pièce...etc

Le chapitre suivant se propose de décrire la démarche de structuration des applications. Celui-ci doit montrer, en particulier, les processus intégrant les primitives de synchronisation et de coopération ainsi que leur obtention.

CHAPITRE III:

STRUCTURATION

DES APPLICATIONS

STRUCTURATION

DES

APPLICATIONS

INTRODUCTION.

Le chapitre précédent a présenté les moyens retenus pour spécifier, avec **l'outil synchrone Grafcet**, les caractères réactif et transformationnel du type d'applications que nous envisageons dans le cadre de cette étude. L'aspect événementiel reprend les primitives de synchronisation citées au même chapitre et qui sont à intégrer dans les processus. L'aspect fonctionnel utilise également ces primitives lorsque la coopération entre processus se réduit à la synchronisation pure. Il reprend également les mécanismes de coopération (compétition, communication, consultation) exposés au chapitre précédent. Ces mécanismes constituent généralement des processus transformateurs de données.

Pour les processus ultimes transformateurs de données il s'agit notamment de déclencher des actions en fonction d'un certain contexte. Pour compléter ce travail on doit:

- dresser un état des lieux sur les actions à exécuter dans le contexte de notre étude.
- montrer l'utilisation, à ce niveau, des outils dégagés ainsi que la spécification des processus (transformateurs de données).
- disposer d'une démarche cohérente dans la structuration des applications.

Ce chapitre expose donc dans un premier volet les diverses activités que l'on rencontre dans les cellules flexibles d'assemblage et précise les primitives ainsi que les processus utilisés.

Le second volet détaille le fonctionnement des primitives. Le troisième volet présente les processus de gestion des lieux de transfert et des lieux d'assemblage.

Le quatrième volet présente une méthode d'obtention des processus de commande, leur spécification ainsi que deux aides à la description opératoire et fonctionnelle des applications.

Enfin le cinquième volet présente la dernière étape dans la structuration des applications en intégrant les contraintes matérielles.

I ACTIVITES DES CELLULES FLEXIBLES D'ASSEMBLAGE.

Les structures industrielles font apparaître deux types d'activités associées aux cellules flexibles d'assemblage par robots:

- les activités conduisant à des actions tendant à faire évoluer l'état du processus d'assemblage. On y distingue:
  - les **actions positionnelles** assurant le transport et la distribution des composants au sein de la cellule.
  - les **actions fonctionnelles** assurant l'établissement de liaisons mécaniques donc fonctionnelles entre composants à assembler.
- les activités tendant à fournir des informations sur les composants, à gérer des zones de stockage intermédiaire ou des zones d'assemblage soit:
  - les **activités de test** effectuant des actions de mesure et d'acquisition d'informations nécessaires à l'évolution de l'état du processus d'assemblage.
  - les **activités de stockage** effectuant la gestion des lieux de transfert ou des lieux d'assemblage.

Dans ces conditions nous utilisons six fonctions principales pour décrire les activités des cellules flexibles d'assemblage.

- Fonction **transfert** exécutée par robot. Elle se compose d'une première action positionnelle (prise d'un objet puis dégagement), suivie d'une seconde action positionnelle (transport, dépôt d'objet puis dégagement).

- Fonction **convoyage** exécutée par convoyeur. Elle consiste à lancer ou arrêter un convoyeur pour acheminer un objet (ou plusieurs) d'un lieu vers un autre lieu de la cellule ou de ou vers l'extérieur.

- Fonction **assemblage** exécutée par robot. Celle-ci débute par une action positionnelle (prise d'objet puis dégagement) et se termine par une action fonctionnelle (insertion d'objet puis positionnement approprié et dégagement).

- Fonction **test** exécutable par système de vision ou machine spécialisée. Elle peut prendre l'une des deux formes suivantes:

  - **attester de la conformité** d'un produit intermédiaire ou fini.

  - **identifier** et/ou **localiser** un objet.

- Fonction **stockage intermédiaire**. C'est une activité de gestion du stockage à laquelle peut s'ajouter éventuellement une exécution sur une entité physique (exemple rotation d'un plateau). Elle a pour rôle de gérer les demandes et les accès effectifs au stockage ainsi que les demandes et les accès effectifs à la consommation des produits stockés.

- Fonction **stockage assemblage**. Elle correspond à la gestion d'un lieu d'assemblage (ou de sous assemblage) conformément à une gamme d'assemblage (série ou série-parallèle) ou à une partie d'une gamme d'assemblage (sous-assemblage). Celle-ci comporte, s'il y a lieu, une flexibilité locale (permutabilité d'actions).

Les quatre premières fonctions constituent des primitives qui sont à intégrer dans des processus dit processus de commande définis par l'application. Ces primitives sont déclenchées:

- soit par des liens de synchronisation événement-action définis au chapitre précédent.
- soit par des communications synchrones du type producteur/consommateur.

Les deux dernières fonctions constituent des processus transformateurs de données qui gèrent l'accès aux lieux concernés.

Dans ce qui suit nous détaillons dans une première phase le principe de fonctionnement des deux premières primitives. Pour les deux autres primitives il n'est pas possible de trouver une forme générale puisqu'elle dépend de l'application. Dans une seconde phase, nous décrivons les processus de gestion de lieux de stockage ou d'assemblage (deux dernières fonctions). Dans une troisième phase, nous précisons les processus de commande utilisant les quatre primitives dégagées. Une dernière phase décrit les processus de gestion des ressources partagées.

II STRUCTURATION DES PRIMITIVES.

II-1 Primitive de transfert.

Considérons un robot chargé, entre autres, de prendre une pièce d'un lieu de stockage T1 (action positionnelle), de la transporter puis de la déposer (action positionnelle) sur un lieu de stockage T2 (figure 3.1). Pour cela le processus de commande du robot doit faire appel à une primitive du type transfert.

La primitive se comporte tantôt comme un consommateur vis à vis du processus de gestion du lieu T1, tantôt comme un producteur vis à vis du processus de gestion du lieu T2 (figure 3.2).

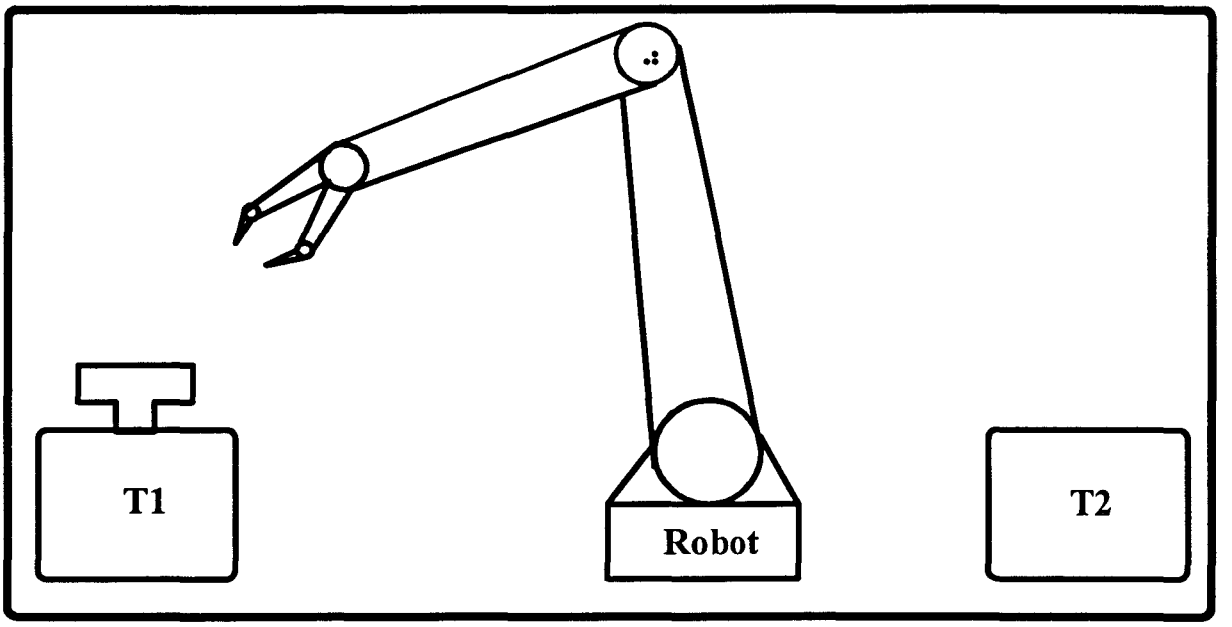


figure 3.1

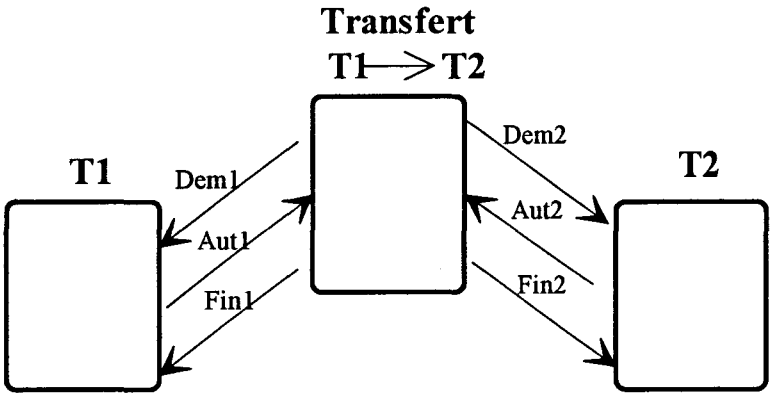


figure 3.2

L'insertion de la primitive de transfert dans le processus est illustrée par le Grafcet de la figure 3.3. En effet, dans la mesure où le robot chargé d'effectuer le transfert peut servir à d'autres primitives ou processus, il faut d'abord s'assurer qu'il y a matière à consommer un objet de T1 (arc Dem1 sur la figure 3.2). Lorsque cela est possible (Aut1), le processus doit ensuite s'assurer qu' il y a possibilité de production pour T2 (Dem2 et Aut2), avant d'engager la consommation effective (prise d'objet de T1).

Lorsque l'autorisation Aut2 est reçue, le processus utilisant la primitive de transfert doit faire une demande de réquisition des ressources physiques nécessaires (ici Robot) (arc Dem Ressource sur figure 3.3).

Lorsque le processus reçoit la confirmation (arc Aut, étape 7 active), l'étape 8 est activée. Par conséquent, il y a émission, vers le procédé (robot), d'un ordre pour l'exécution de la première action positionnelle *(Prendre objet sur T1)*.

La primitive événement-action *Pour Ef faire (signaler Fin1 et activer étape 9)* est alors exécutée. L'événement Ef représente l'événement reçu du procédé et signalant la fin de l'action demandée. Par conséquent, l'arc Fin1 signale à T1 la fin de la consommation.

Une fois l'étape 9 activée, il y a émission d'un ordre de transport puis de dépôt de l'objet sur T2 *(Déposer objet sur T2)*. La primitive *Pour Ef faire (signaler Fin2 et activer 10)* est ensuite exécutée. L'arc Fin2 sert à signaler au processus de gestion de T2 la fin de la production effective.

La connexion avec le processus de gestion de T1 peut être conçue, si nécessaire, uniquement en termes de synchronisation. Les étapes 2 et 3 ainsi que les arcs Dem1 et Aut1 sont alors remplacés par un des liens de synchronisation cités dans le chapitre précédent.

Ce cas se produit, par exemple, lorsque le transfert n'a lieu que sur occurrence d'un événement qui signale l'arrivée de la pièce à transporter.

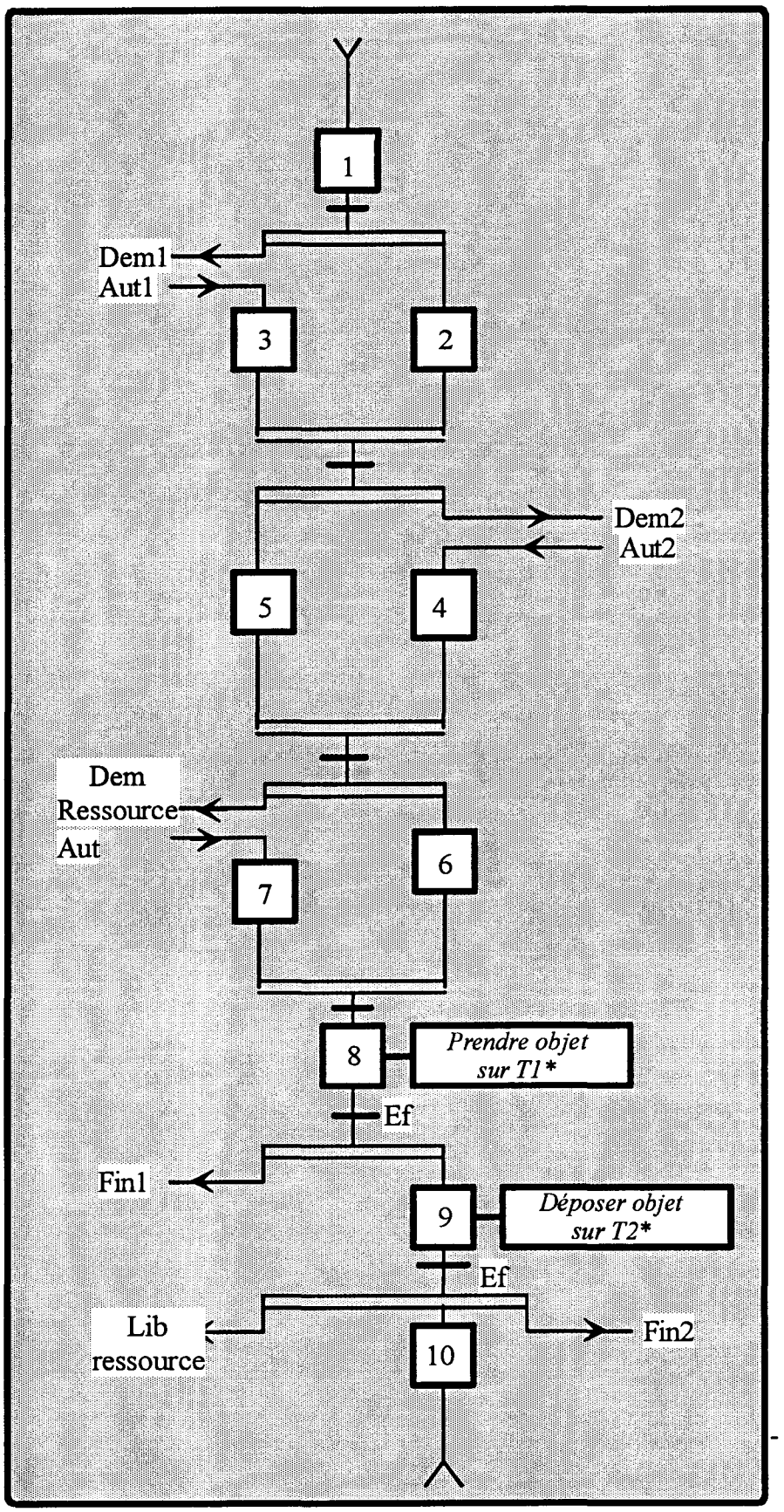

1
Dem1
Aut1
3
2
Dem2
Aut2
5
4
Dem
Ressource
Aut
7
6
8
Prendre objet
sur T1*
Ef
Fin1
9
Déposer objet
sur T2*
Ef
Lib
ressource
10
Fin2

figure 3.3

II-2 Primitive d'assemblage.

Considérons un robot chargé, entre autres, de prendre un objet M d'un lieu donné TT (action positionnelle), de le transporter puis de l'insérer sur un produit intermédiaire N (action fonctionnelle notée AF) se trouvant sur un lieu d'assemblage SA1 (figure 3.4). Pour cela, le processus de commande du robot doit faire appel à une primitive d'assemblage. Cette primitive se comporte tantôt comme consommateur du processus de gestion du lieu TT, tantôt comme producteur pour le processus de gestion du lieu d'assemblage SA1 (figure 3.5).

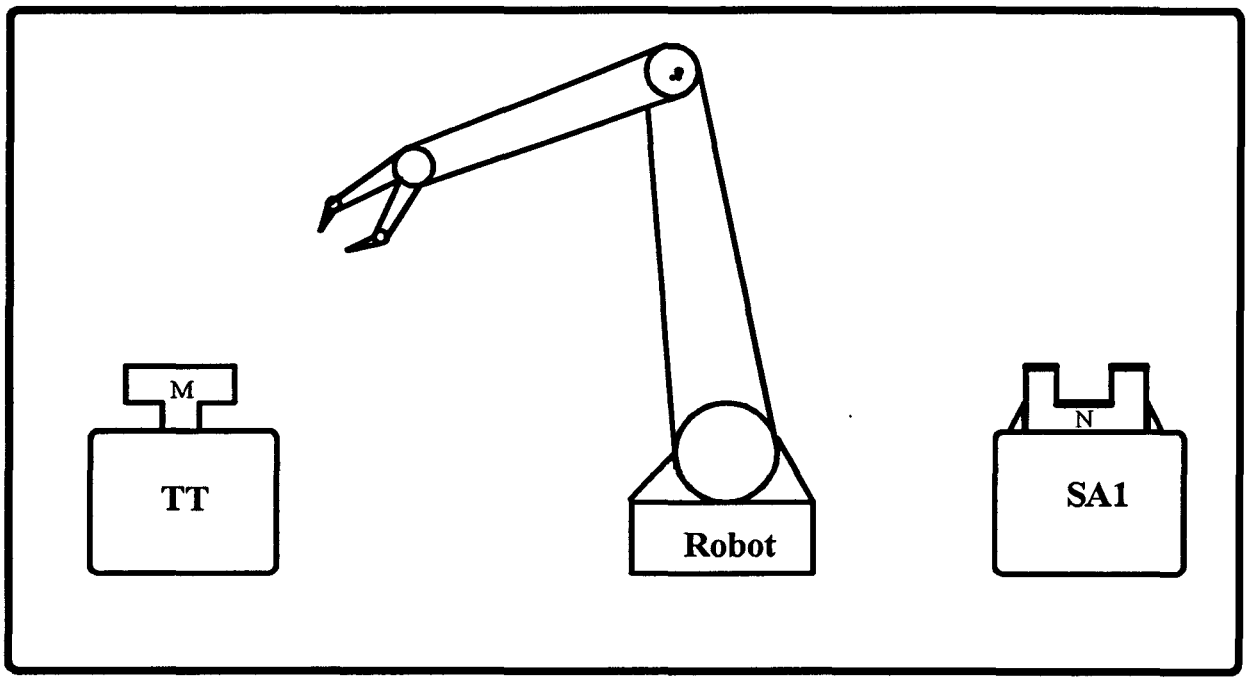


figure 3.4

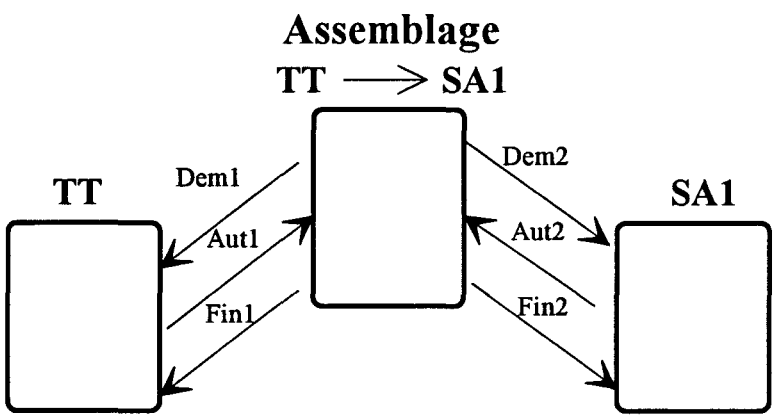


figure 3.5

La figure 3.6 représente la description par Grafcet de cette primitive.

Nous retrouvons les mêmes mécanismes que pour le transfert. La seule différence réside dans l'action associée à l'étape 9, qui est cette fois-ci une action fonctionnelle *(Assembler objet M sur PI dans SA1)*.

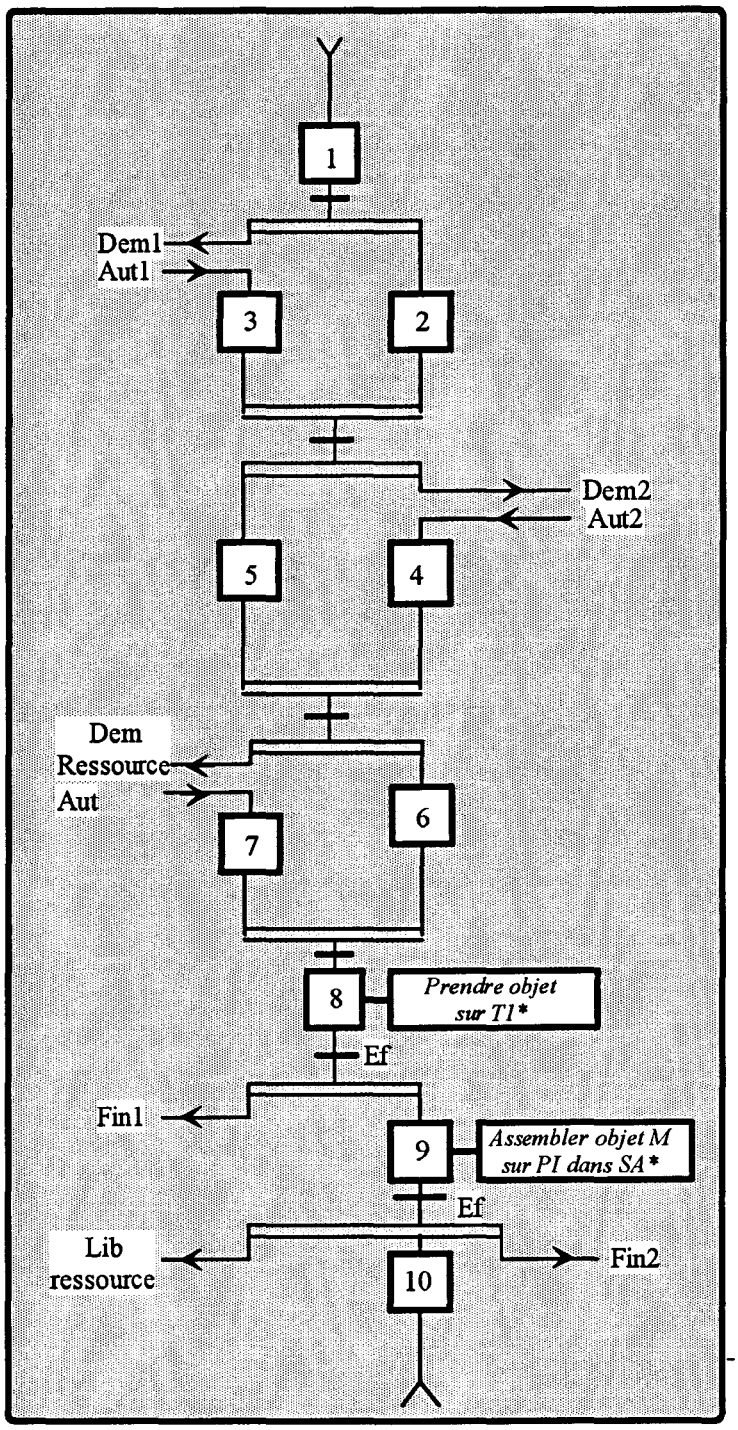


figure 3.6

III STRUCTURATION DES PROCESSUS DE GESTION DE LIEUX.

Dans le cadre de cette étude et ainsi que nous l'avons signalé plus haut, les mécanismes de compétition, communication, et consultation constituent un type particulier de **processus transformateurs de données** à coté des processus ultimes transformateurs de données que sont les processus de commande. Ceux-ci seront détaillés plus loin. Les processus les plus en vue sont les processus gestionnaires de lieux d'assemblage et de transfert.

III-1 Processus gestion lieu d'assemblage.

Un processus de gestion d'un lieu d'assemblage accompagné de ses connexions avec ses n producteurs et son unique consommateur est un cas de **communication synchrone** avec une **gestion particulière de la file d'attente** conformément à la gamme d'assemblage envisagée (figure 3.7).

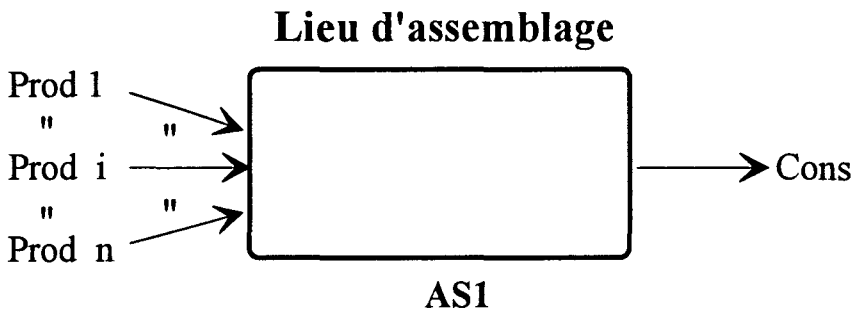


figure 3.7

Pour illustrer cela, considérons la gamme d'assemblage série-parallèle du stylo à bille présentée au chapitre I paragraphe V (figure 3.8). Chaque composant parmi les six (n=6) est appréhendé par un producteur (exécutant une primitive d'assemblage).

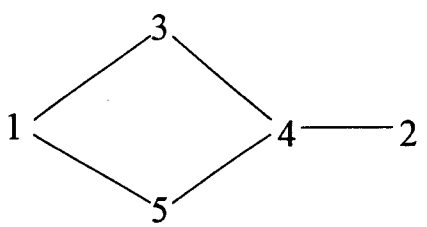


figure 3.8

La figure 3.9 présente le Grafcet du processus chargé de gérer le lieu d'assemblage AS1 ainsi que ses connexions avec ses n producteurs et son consommateur.

Pour chaque producteur, nous retrouvons les mécanismes usuels de demande, d'autorisation puis de confirmation de fin de production.

Les places numérotées 04....05 correspondent au dépôt du premier objet (corps du stylo) dans l'étau du lieu d'assemblage. Pour cette première action, il n'y a pas d'action fonctionnelle (producteur P0).

Pour les autres producteurs Pi i=1,5, les places numérotées i4....i5 correspondent à l'établissement de l'action fonctionnelle AFi correspondante. Afin de ne pas surcharger la figure, nous ne représentons pas les actions positionnelles précédant les actions fonctionnelles.

Chaque transition suivant une étape notée i5 (i=1,..,5) est conditionnée par l'événement de fin d'établissement de l'action fonctionnelle correspondante (Fin AFi).

La **gamme d'assemblage** est clairement exprimée par l'ordre de franchissement des transitions T1 à T6. Par exemple, si le processus P4 désire ramener sa pièce pour établir l'action fonctionnelle AF4, il ne sera autorisé à le faire que si la transition T4 est franchie, donc l'étape 8 marquée. Par ailleurs, cette même transition ne peut être tirée que si les actions AF3 et AF5 ont été réalisées et ainsi de suite.

Enfin, un nouvel assemblage ne peut alors s'opérer que si le consommateur a fini d'évacuer le précédent assemblage (transition Fin Cons franchie, étape 1 active).

La **flexibilité Locale** apparaît au niveau des producteurs P3 et P4 à l'issu du franchissement de la transition T3 (corps déposé). Trois cas sont possibles:

- La liaison fonctionnelle 3 est réalisée suivie de la liaison 5.
- la liaison fonctionnelle 5 est réalisée suivie de la liaison 3.
- les deux liaisons fonctionnelles 3 et 5 sont réalisées simultanément, si cela est matériellement possible.

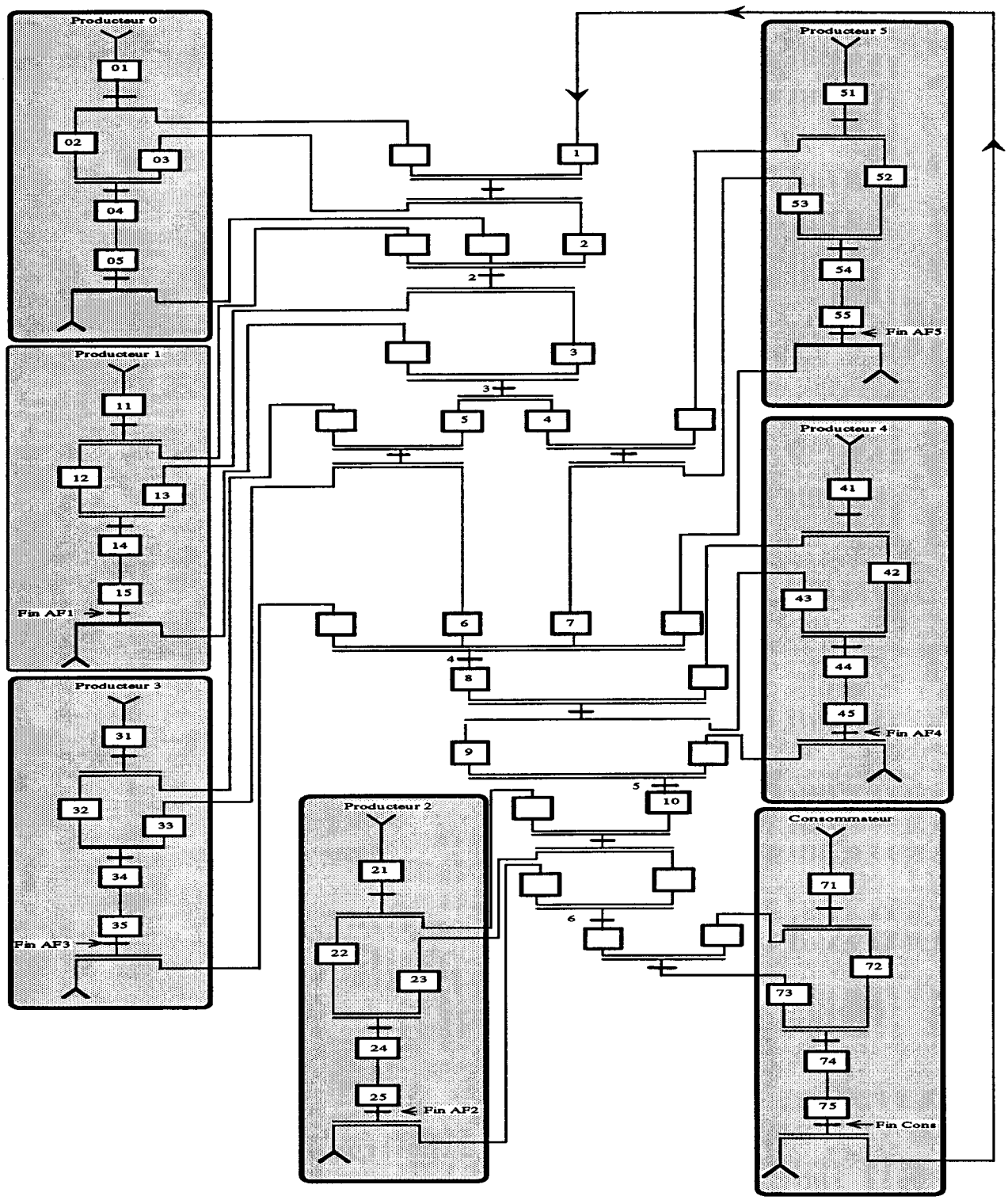

Producteur 0
01
02
03
04
05
Producteur 1
11
12
13
14
15
Fin AF1
Producteur 3
31
32
33
34
35
Fin AF3
Producteur 2
21
22
23
24
25
Fin AF2
Producteur 5
51
52
53
54
55
Fin AF5
Producteur 4
41
42
43
44
45
Fin AF4
Consommateur
71
72
73
74
75
Fin Cons
1
2
2
3
3
5
4
6
7
4
8
9
5
10
6

figure 3.9

III-2 Processus gestion lieu de transfert.

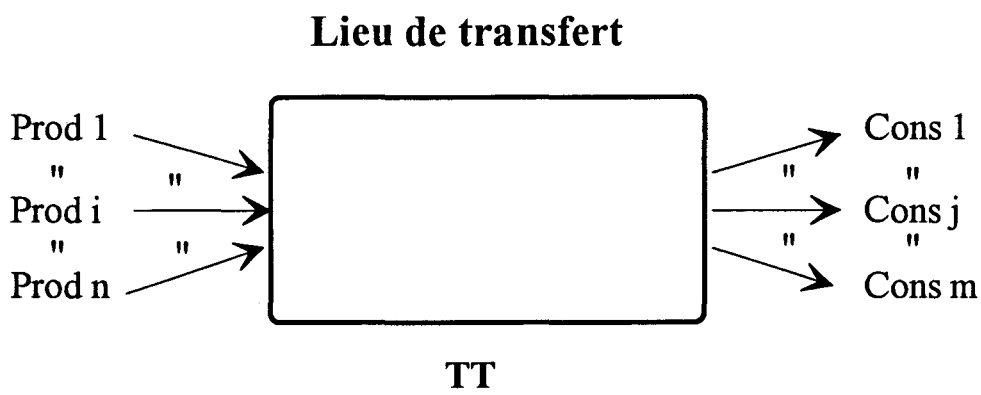


figure 3.10

Le processus de **gestion d'un lieu de transfert** et ses connexions avec ses n producteurs et ses m consommateurs est un cas typique de **communication synchrone**, telle qu'elle a été présentée au chapitre II.

Lorsque le nombre de types d'objets stockés est supérieur à 2, la gestion du processus peut se faire de deux façons selon la disposition physique de la file d'attente:

- à chaque type d'objet correspond une file d'attente de longueur li (i nombre de types d'objets) gérée en consommation de l'objet premier arrivé.

- à l'ensemble des types d'objets est associée une file d'attente unique de longueur l où sont mémorisés et stockés les objets par leur ordre d'arrivée.

Pour illustrer ceci considérons le problème de deux types d'objets (i = 2) produits par deux producteurs différents (un producteur par type) et consommés par deux consommateurs (un consommateur par type).

- La figure 3.11 illustre le premier cas pour $l_1 = 2$ et $l_2 = 2$.

  Sur cette figure, une seule file est représentée. L'autre file serait représentée de la même façon. Les deux files sont ensuite regroupées dans un processus unique, bien qu'elles n'aient pas directement de lien commun.

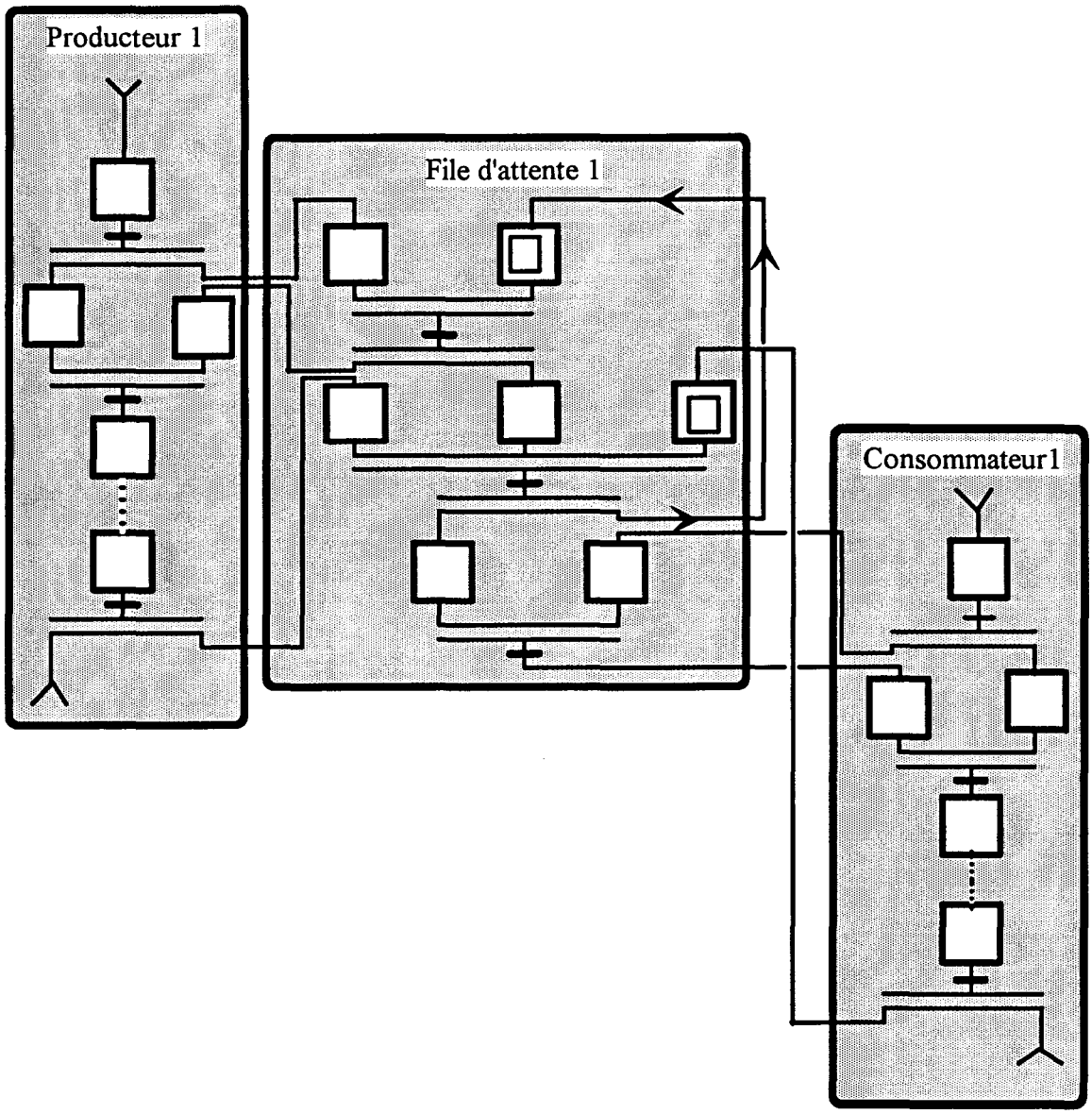


figure 3.11

Si pour un même type d'objet il y a plusieurs producteurs et/ou plusieurs consommateurs, on ajoute des structures d'exclusion mutuelle entre producteurs et/ou consommateurs concernés. En revanche, le cas de la consommation multiple du même objet ne peut être envisagé ici car il n'a pas de signification physique.

– La figure 3.12 illustre le deuxième cas avec l = 2.

Les places 8 et 32 d'une part et les places 30 et 33 d'autre part sont les places occupées respectivement par le premier type d'objet et par le second type

d'objet. Toutefois, une seule étape parmi les étapes 8 et 30 est active au maximum. Il en est de même pour les étapes 32 et 33.

L'étape 1, aidée des étapes 4,5 et 6,7, mémorise deux demandes successives (nécessairement différentes) des producteurs si la file d'attente est pleine. La réceptivité $\overline{X}_5$, associée à la transition en aval des étapes 4 et 1, donne la priorité au producteur 2 en cas de demandes simultanées. L'étape 2 permet d'autoriser un producteur à produire. Quant à l'étape 3, elle permet de faire avancer un objet de la première place vers la seconde, si celle-ci est libre.

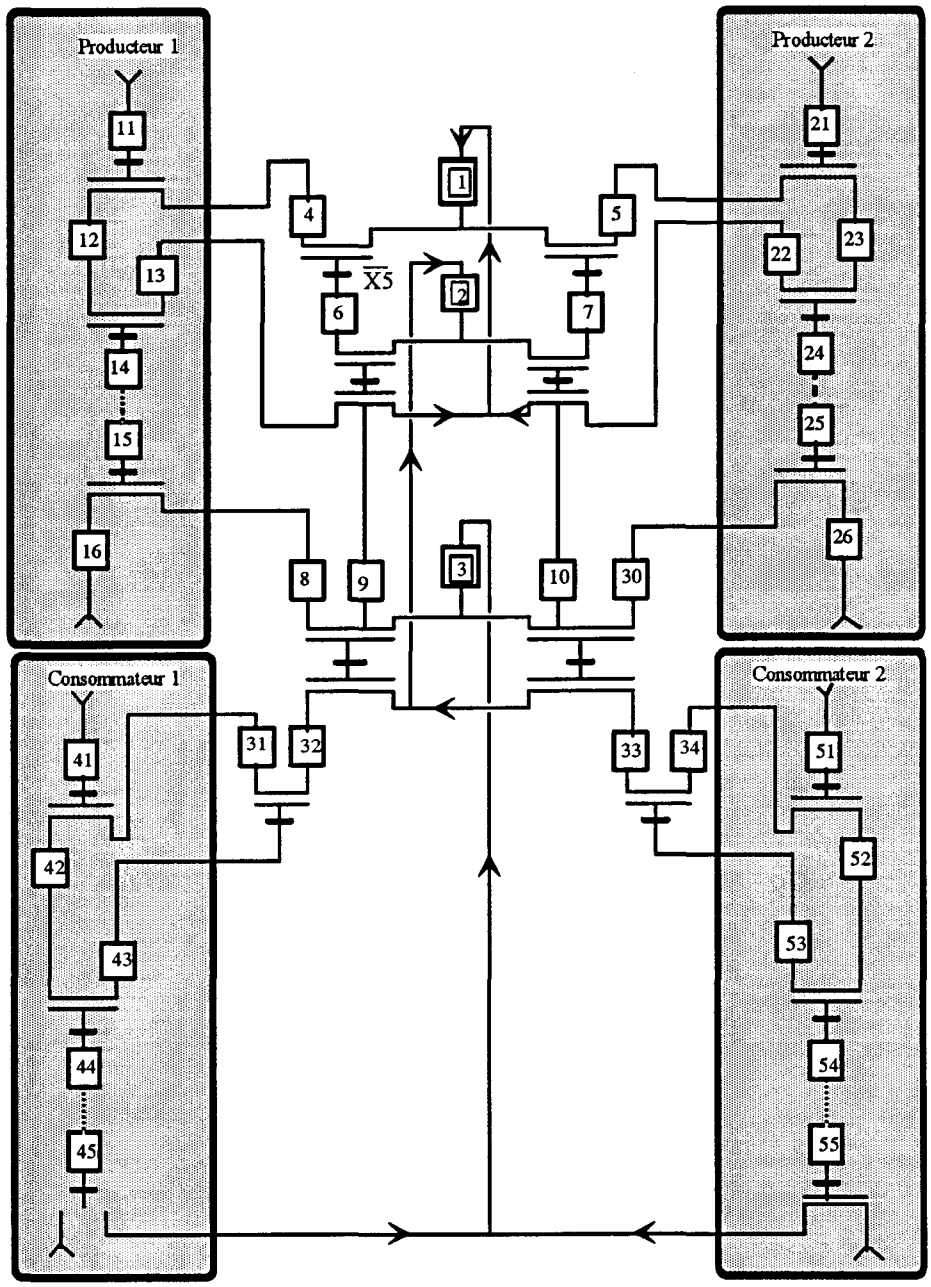


figure 3.12

On peut prévoir un mécanisme d'exclusion mutuelle si plusieurs producteurs et/ou plusieurs consommateurs se disputent la production et/ou la consommation du même type d'objets.

IV OBTENTION ET SPECIFICATION DES PROCESSUS DE COMMANDE.

Les paragraphes précédents ont mis en évidence l'éventail des primitives dont on dispose pour la description des activités des cellules flexibles d'assemblage par robots. Ce paragraphe vise à définir une méthodologie d'obtention et de spécification des processus de commande intégrant ces primitives.

IV-1 Principe d'obtention des processus de commande.

Le principe utilisé vise à obtenir dans un premier temps un maximum de parallélisme. Ceci conduit à générer un nombre maximum de processus, indépendamment des contraintes matérielles. Ce principe conduit:

- d'une part à associer un processus de commande à chaque primitive prise parmi les types "assemblage" "transfert" et "convoyage", ceci pour un lieu de départ et un lieu d'arrivée donnés. Par exemple, si une primitive de transfert est utilisée pour manipuler des objets d'un lieu T1 vers un lieu T2, on crée un processus TRANSFERT (T1, T2).

  Par ailleurs, la description du processus doit intégrer l'utilisation de la primitive citée pour tous les types d'objets manipulés entre T1 et T2, dans une application complète.

- d'autre part à associer, à chaque primitive du type "test", un processus pour chaque zone distincte de test. Par exemple le processus LOCALISE (CV) est un processus chargé de localiser des pièces sur un convoyeur CV. Celui-ci utilise nécessairement la primitive localiser (objet, CV).

IV-2 Evolution d'un processus et interactions diverses.

Un processus de commande existe dès lors que l'application existe. Pour pouvoir continuer son exécution, il doit interagir, en fonction de l'application, avec les éléments suivants (figure 3.13):

- le **procédé,** par le biais d'événements émis ou reçus de celui-ci. Ces relations sont exprimées soit par des liens de synchronisation avec acquittement collectif, cités au chapitre précédent, soit par des liens de synchronisation avec acquittement individuel (gérés à un autre niveau de la commande).

- L'**environnement**: utilisateur ou niveau supérieur de la commande (niveau Ilot). ce type d'interaction est spécifié:

  - par des liens de synchronisation (acquittement collectif ou individuel) lorsque le processus reçoit des événements de l'environnement.

  - et/ou par des relations du type producteur/consommateur lorsque le processus se comporte comme un consommateur de l'environnement.

- divers **processus de commande**. Ce type d'interaction est spécifié:

  - par des liens de synchronisation (acquittement collectif ou individuel) lorsque la relation consiste en l'émission et/ou la réception d'événements d'autres processus de commande.

  - et/ou par des relations du type producteur/consommateur, ceci lorsque le processus se comporte comme consommateur et/ou producteur.

- des **processus de gestion de ressources**: par le biais de relations du type producteur/consommateur. Ce type de relation est détaillé au paragraphe suivant, lors de l'intégration des contraintes matérielles.

- des **processus de gestion de lieux** qui matérialisent les gestions des files d'attente associées à ces lieux.

Ainsi que nous l'avons expliqué au chapitre précédent, la **spécification d'un processus de commande** doit porter sur l'**enchaînement des diverses actions**

déclenchées par ce processus, **jalonnées par des outils de synchronisation et de coopération**. Par ailleurs, cette spécification doit porter également sur la "vie" du processus. Il s'agit notamment de spécifier si le processus **boucle indéfiniment** ou s'il doit **se terminer** et dans ce cas sous **quelles conditions**.

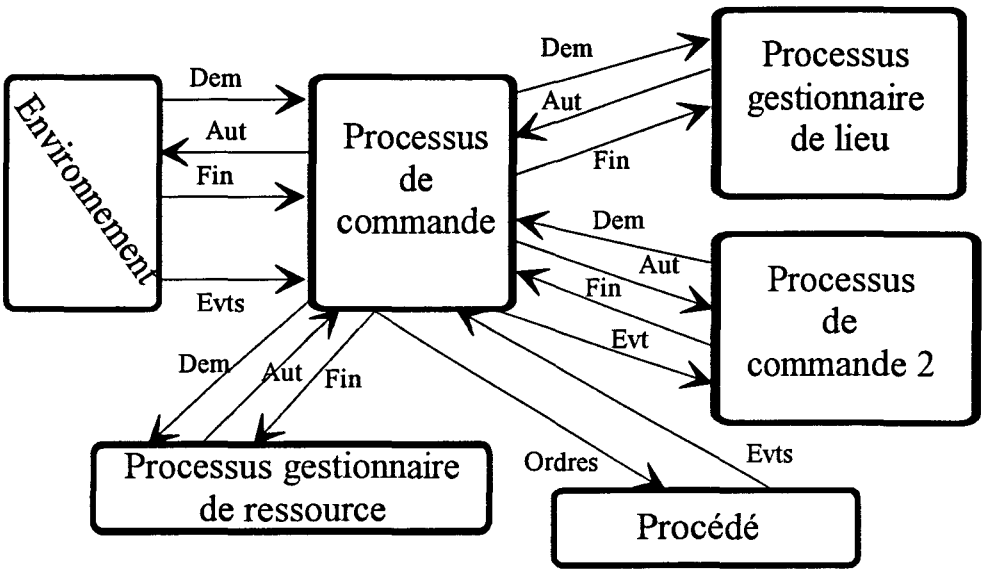


figure 3.13

IV-3 Description opératoire d'une application.

A partir de la mise en évidence des différents types de processus susceptibles d'intervenir dans la commande d'une cellule flexible d'assemblage, on opère une **description opératoire** de l'application envisagée.

Cette description se présente sous la forme d'un graphe dont les noeuds sont les différents lieux de la cellule et les arcs les divers processus de commande et de gestion de lieux intervenant sur les composants de l'assemblage.

Ce graphe permet de visualiser les différentes activités de la cellule, les lieux où se déroulent ces activités, ainsi que les itinéraires suivis par les produits.

Exemple (figure 3.14):

Soit un assemblage de 6 composants. Trois des composants sont ramenés d'un lieu CP. Les trois autres sont acheminés par convoyeur de l'extérieur vers un lieu CV2. Chaque composant est ensuite ramené vers la table de transfert TT pour être assemblé sur la table de montage TM conformément à une gamme d'assemblage. Lorsque l'assemblage est terminé, le produit est transféré vers le lieu CV1 pour test. Il est ensuite évacué vers l'extérieur par convoyeur.

Cette exemple met en évidence les lieux suivants:

- CP: casier à pièces.
- TT: table de transfert .
- TM: table d'assemblage.
- CV1: place de convoyage pour évacuation.
- CV2: place d'arrivée par convoyage.

Les divers processus agissant sur les composants sont:

- Transfert (CP, TT).
- Gestionnaire Lieu de Stockage Intermédiaire G.L.S.I (TT).
- Gestionnaire Lieu d'Assemblage G.L.A (TM).
- Assemblage (TT, TM).
- Transfert (TM, CV1).
- Transfert (CV2, TT).
- Evacuation (CV1).
- Test (CV1).

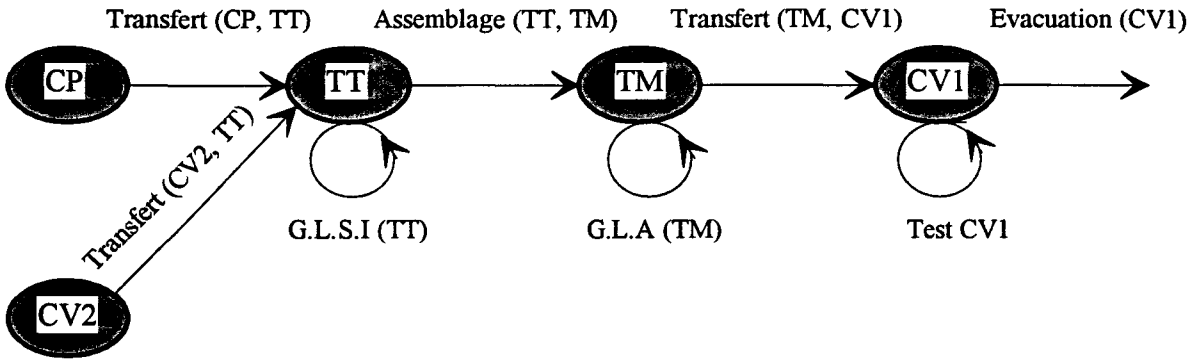


figure 3.14

IV-4 Description fonctionnelle d'une application.

Il est intéressant de s'aider d'une représentation graphique pour décrire les différentes interactions entre processus. La figure 3.15 illustre cette idée pour l'exemple précédent. Cette aide exclut toutefois les interactions avec le procédé et les processus de gestion de ressources afin de ne pas alourdir la représentation.

Chaque processus est représenté par un rectangle. Une relation producteur/consommateur du type **communication simple** (sans tampon) est représentée par **une flèche** partant du producteur et se dirigeant vers le consommateur. Ainsi le processus TRANSFERT (TM, CV2) se comporte comme producteur; en communication simple, du processus TEST (CV2). Une relation **producteur/consommateur normal** (séparés par un tampon) est représentée par **deux flèches à double sens**. La première lie le producteur au processus tampon. La seconde lie ce dernier au consommateur. Pour illustrer cela, l'interaction entre le processus TRANSFERT (CP, TT) et le processus ASSEMBLAGE (TT, TM) via le processus tampon G.L.S.I (TT), est du type producteur/consommateur.

Un événement est représenté par une flèche brisée. Un exemplaire de cette flèche sert à spécifier le lieu de départ de l'événement (processus émetteur s'il existe). D'autres duplications servent à montrer les processus réceptifs à ce même événement. Le processus TRANSFERT (CV, TT), par exemple, reçoit trois types d'événements E1, E2, E3 d'un système de reconnaissance (non représenté ici). Nous ne représentons pas

non plus les interactions en amont du processus de gestion du lieu CP afin de n'a pas alourdir le schéma.

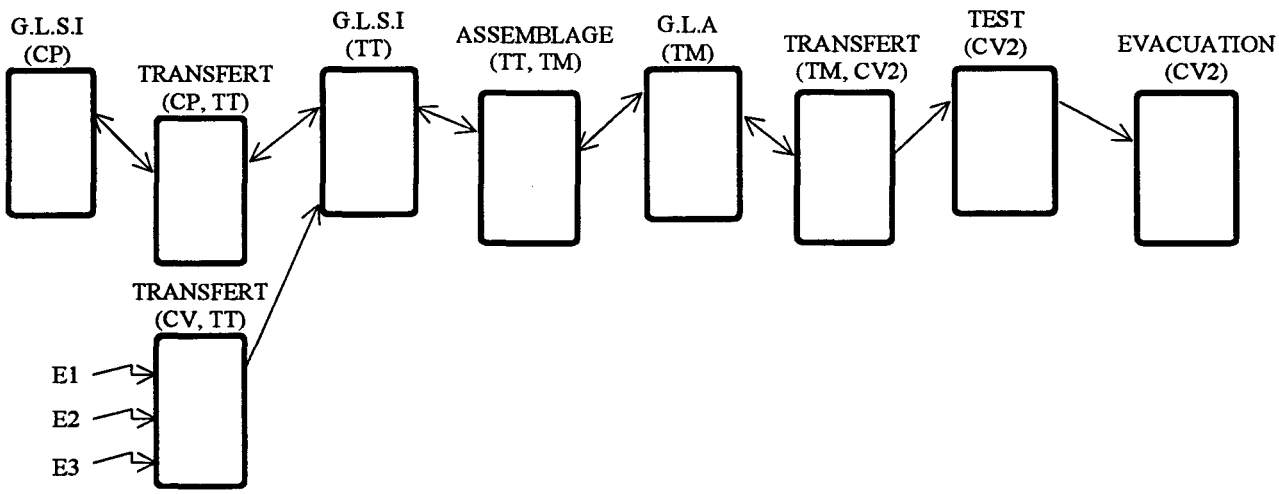


figure 3.15

V STRUCTURATION DES APPLICATIONS AVEC INTEGRATION DES CONTRAINTES MATERIELLES.

Le but de cette étape est de faire apparaître sur la description de l'application les contraintes relevant de l'architecture matérielle de la cellule et plus particulièrement des robots, systèmes de test et convoyeurs.

En effet, la phase précédente de structuration aboutit à un maximum de processus. Par ailleurs, l'architecture matérielle impose fréquemment, pour des questions de performances, le partage d'un même matériel par plusieurs processus de commande. Il s'agit par exemple de partager l'utilisation d'un robot par deux processus de transfert.

Ce partage, du fait de l'**indivisibilité d'une primitive**, ne peut avoir lieu pendant l'exécution de celle-ci. Une exception est faite lorsque l'application, pour respecter des contraintes temporelles, autorise la préemption. Ce cas de figure est traité au chapitre suivant.

La structuration du partage d'une ressource consiste à associer à chaque ressource un processus chargé de gérer les accès à cette ressource par les processus de commande concernés. La politique de gestion des demandes, comme on l'a exposé au chapitre II

paragraphe 2, peut être soit, par priorité, soit en premier demandé premier servi ou encore une combinaison des deux modes. L'interaction d'un processus de commande avec un processus de gestion de ressource (figure 3.16) consiste alors en la formulation d'une demande par le premier processus (arc Dem Ressource). Lorsque celui-ci reçoit l'autorisation (arc Aut), il accède à la ressource. A l'issue de l'utilisation de la ressource, le processus de commande libère cette dernière et le signale au processus de gestion de la ressource (Lib).

La figure 3.17 illustre le partage d'un robot par deux processus de commande: TRANSFERT (T1, T2) et TRANSFERT (CV, T2). Le premier transfert des pièces de T1 vers T2. Le Grafcet a, est une partie de ce processus qui représente la primitive transférer. Une fois que toutes les conditions sont réunies (objet présent, place d'arrivée libre) la réquisition du robot est demandée (arc a). Le processus de gestion de la ressource robot (Grafcet b) reçoit la demande et l'examine en donnant la priorité, en cas de demandes simultanées, au processus TRANSFERT (CV, T2). Ce dernier (Grafcet c) est chargé de transférer trois types d'objets signalés par les événements E1, E2 et E3 (arrivée sur convoyeur). La primitive transférer est cette fois-ci regroupée dans une macro étape car il n'y a pas de nécessité de signaler la fin de prise d'un objet sur le convoyeur au processus de gestion amont puisque ce dernier n'existe pas.

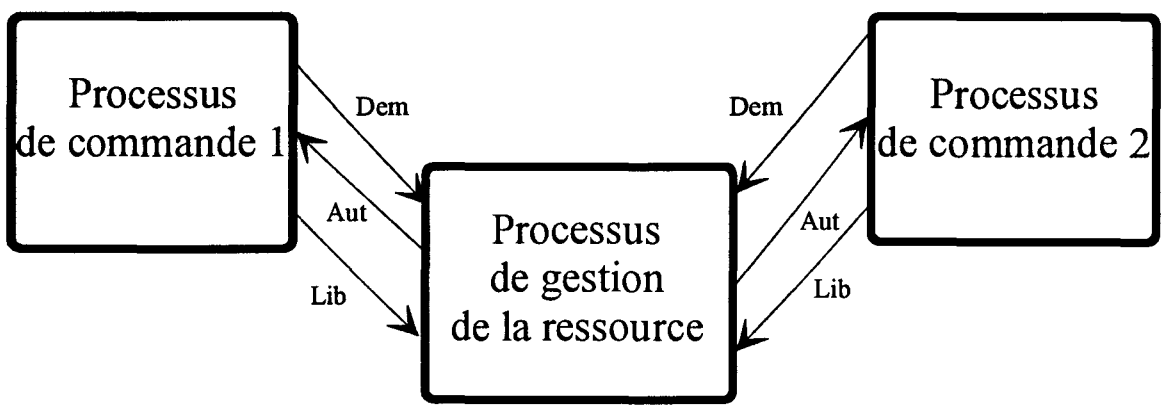


figure 3.16

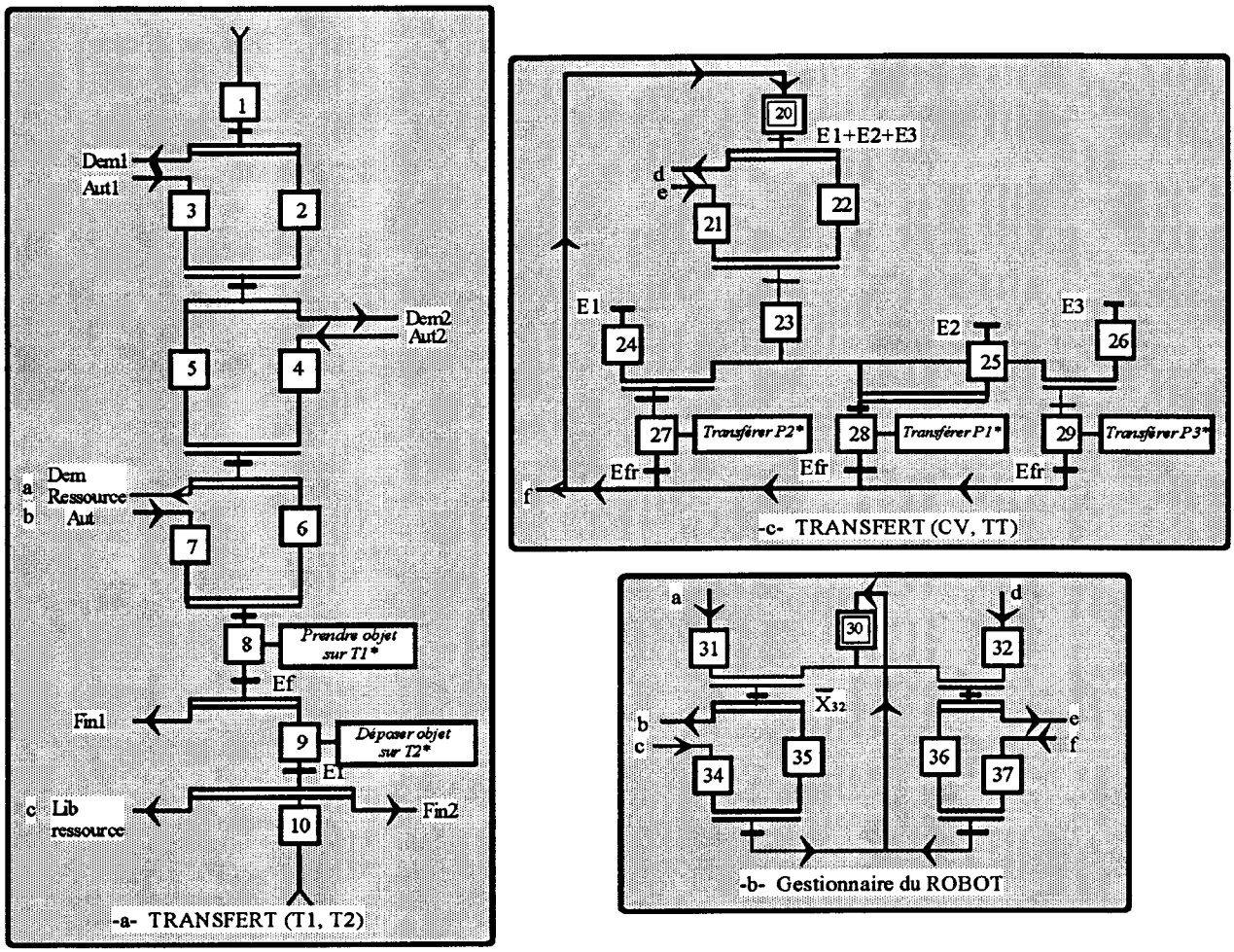


figure 3.17

CONCLUSION.

Toutes les primitives que nous avons examinées constituent, selon la connexion avec le processus amont:

- soit un cas typique de communication synchrone.
- soit un cas de synchronisation.

En outre, ces primitives peuvent être vues:

- soit comme des actions à déclencher avec un lien de synchronisation approprié.

- soit comme des actions à intégrer dans un processus qui évolue lui même en fonction de liens de synchronisation ou de communications synchrone.

- soit comme des processus transformateurs de données. C'est la cas des zones tampon en communication synchrone.

Par ailleurs, nous avons montré que les processus de gestion de lieux de transfert ou de lieu d'assemblage, constituant par ailleurs des processus transformateurs de données, représentent un cas typique de communication synchrone.

En outre, la structuration de l'application en deux étapes (sans intégration puis avec intégration des contraintes matérielles) permet de composer l'application en processus de commande, en processus de gestion de lieu et en processus de gestion de partage de ressources.

Les différentes interactions entre les processus d'une part et entre les processus et le procédé et/ou l'environnement d'autre part sont clairement spécifiées et structurées par les divers outils du chapitre II.

Enfin, du fait de la variété des possibilités d'utilisation des primitives de test et de convoyage, nous n'avons pas jugé nécessaire de les représenter. Nous verrons toutefois quelques exemples dans le chapitre suivant qui décrit en particulier la méthodologie adoptée pour la spécification des contraintes temporelles et pour la spécification des liens de synchronisation avec acquittements individuels.

CHAPITRE IV:

METHODOLOGIE DE

LA COMMANDE

METHODOLOGIE DE LA COMMANDE

INTRODUCTION.

Le chapitre précédent a exposé la méthode de structuration des applications sans spécification des contraintes temporelles. Ce chapitre expose dans un premier temps l'esprit de la méthodologie adoptée pour la spécification de la commande.

Dans un second temps, il développe un modèle cohérent pour la hiérarchie en Grafcet, ainsi qu'un éventail de macroactions existantes et de macroactions nouvellement définies. En outre, il propose des règles associées à chacune de ces macroactions.

Dans un troisième temps, le chapitre expose, dans le détail, la méthodologie de commande adoptée dans le cadre de cette étude. Celle-ci se présente sous forme hiérarchique. Les contraintes temporelles sont alors naturellement gérées à un niveau bien défini de cette hiérarchie.

Finalement, plusieurs exemples illustrent le niveau d'expression des contraintes temporelles et de la gestion des événements à occurrences acquittables d'une façon individuelle.

I METHODOLOGIE DE LA COMMANDE.

Dans les applications temps-réel, les contraintes temporelles se manifestent, soit sous forme de durée de vie sur les événements, soit sous forme de temps de réaction ou d'échéances sur les actions.

Le retard enregistré sur le traitement de l'occurrence d'un événement se répercute directement sur l'échéance de l'action que cette occurrence doit lancer. Traiter une application en termes de durée de vie ou bien en termes d'échéances sont deux moyens équivalents. Dans l'approche que nous avons développée, nous nous intéressons au premier aspect, c'est à dire aux durées de vie des événements.

Ainsi que nous l'avons vu au chapitre I, on peut classer les événements en deux types, selon le caractère urgent ou non de la **réaction attendue:**

- les **événements non contraignants**: c'est le type d'événements pour lesquels si l'occurrence n'est pas traitée en temps opportun, le fonctionnement du système n'est pas modifié fondamentalement (contrainte faible).

- les **événements contraignants**: c'est le type d'événements, dont chaque occurrence nécessite un traitement dans les normes (réaction **immédiate** ou **différée**), faute de quoi le fonctionnement du système peut dériver vers des états incontrôlables (contrainte forte).

Les primitives de synchronisation événement-action citées au chapitre II permettent de spécifier le comportement attendu de l'application pour chaque événement. En outre, lorsque une occurrence ou un groupe d'occurrences d'un événement non contraignant sont invalidées (durées de vie dépassées), il suffit d'acquitter l'ensemble de ces occurrences.

En revanche, **l'aspect décisionnel** dans les actions à générer pour respecter les contraintes temporelles (contraintes fortes) n'est pas représenté par ces primitives. Ces problèmes de contraintes temporelles sont effectifs lorsque les ressources partagées par plusieurs processus et nécessaires à l'accomplissement des actions voulues sont occupées et ne peuvent être libérées dans les temps voulus. Un niveau décision doit alors intervenir en connaissance des priorités accordées aux événements, afin d'opérer des **suspensions** et **préemptions** de ressources au détriment des événements de faibles

priorités et au profit des événements de fortes priorités. Ce même niveau doit ensuite veiller à la **reprise des actions suspendues**, lorsque les ressources deviennent libres et que les priorités des événements s'y prêtent.

Tout se passe comme si le niveau décisionnel devait être à l'écoute de tout événement en provenance du procédé (ressources) et de la situation des actions en cours, afin de surveiller le déroulement de celles-ci et d'intervenir chaque fois que les contraintes risquent d'être enfreintes.

Cette idée maîtresse, qui guide la démarche que nous avons élaborée, considère essentiellement deux niveaux de commande en fonctionnement normal: un **niveau Application** et un **niveau Surveillance Temporelle et Décision** (figure 4.1).

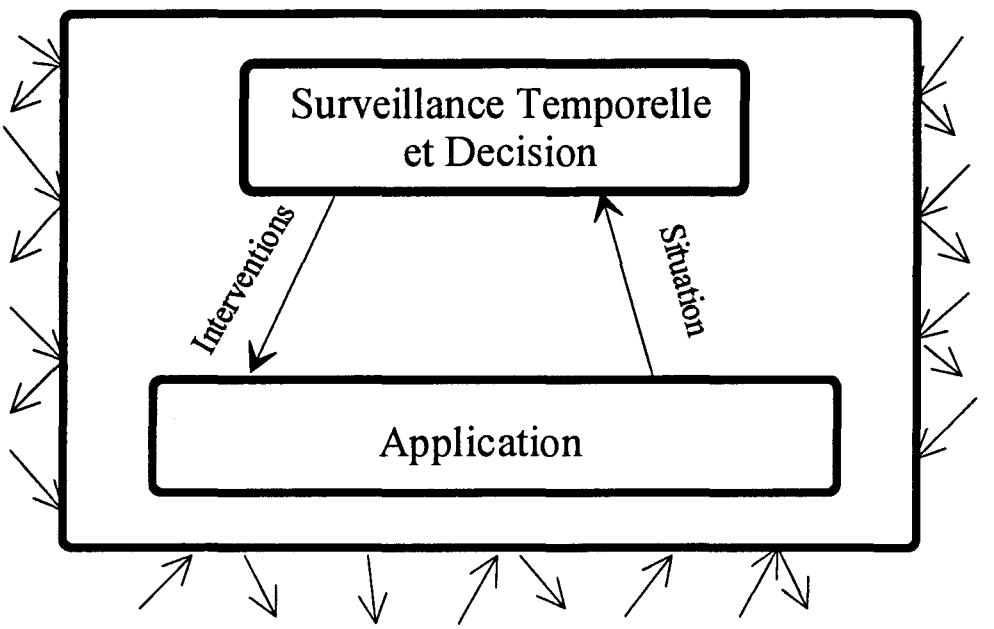


figure 4.1

Le niveau **Application** commande le procédé comme si aucun problème de contraintes temporelles n'existe. Les ressources sont allouées, en cas de demandes simultanées, en priorité au processus qui génère l'action activée par l'événement le plus contraignant. Le niveau **Surveillance**, constamment informé de la situation du niveau **Application** ainsi que de celle de l'environnement (événements), suit le déroulement des actions et intervient lorsqu'il juge (décision) que les contraintes temporelles risquent d'êtres enfreintes.

Pour pouvoir spécifier ces divers niveaux, décisions et interventions, nous utilisons largement les notions de hiérarchies et de macroactions du Grafcet. Toutefois, dans la mesure où ces notions ne sont pas encore tout à fait figées, nous définissons ci dessous les cadres formels pour ces extensions.

II HIERARCHIE ET MACROACTIONS.

La description par Grafcet de systèmes complexes peut aboutir à des Grafcets de taille si importante qu'ils deviennent très vite touffus, difficiles à élaborer et à comprendre. La notion de **macroaction** en termes de *initialiser, masquer, désactiver* a été introduite [DEN83] afin de décrire l'influence globale d'un Grafcet sur un autre Grafcet. L'idée sous-jacente est l'existence de **hiérarchies** entre Grafcets. Cette notion de macroaction a été plus tard étendue par le GREPA [GRE85] pour les macroactions *forçage* et *figeage*.

La notion de hiérarchie a conduit à définir une typologie de Grafcet: le **Grafcet connexe**, le **Grafcet partiel** constitué d'un ou de plusieurs Grafcets connexes et le **Grafcet Global** qui est l'ensemble de tous les Grafcets partiels décrivant le système (Annexe 2). Les macroactions sont alors les actions générées à partir d'un Grafcet partiel d'un niveau hiérarchique, vers un autre Grafcet partiel d'un niveau inférieur.

Pour résoudre le non déterminisme dû a l'application simultanée du forçage avec la règle d'évolution 4, le groupe GREPA [ADE92] a déposé un projet de norme concernant cette macroaction à la lumière du modèle temporel du Grafcet proposé par les mêmes auteurs (Annexe 2) [CEI88]. Cette proposition, par ailleurs confortée par l'Analyse Non Standard (ANS) [FRA92], stipule en particulier **la priorité du forçage sur les règles d'évolutions** (Règle de forçage 2-a).

Ces derniers développements (typologie et règles de forçage) constituent un véritable progrès pour le Grafcet dans le sens où ils permettent de décrire des systèmes qu'il était impossible de décrire auparavant. La puissance du Grafcet est ainsi accrue.

Toutefois ces développements soulèvent un certain nombre de questions:

1. Un Grafcet partiel d'un niveau hiérarchique n (noté NH_n) peut-il gouverner uniquement des Grafcets partiels du niveau inférieur NH_{n+1} ou bien peut il gouverner aussi des Grafcets partiels des niveaux NH_i $i > n+1$?

2. quel domaine de visibilité peut avoir un Grafcet partiel d'un niveau NH_i sur les situations de tous les autres Grafcets partiels des autres niveaux hiérarchiques?

3. quel champ d'application est concerné par la règle 2-b du forçage (Annexe B)? En d'autres termes **l'effet produit** par un ordre de forçage en provenance d'un niveau NH_i est-il disponible aux niveaux autres que le niveau NH_{i+1}?

4. est il nécessaire d'avoir des règles supplémentaires, en plus des règles d'évolution, pour régir les autres ordres (existant ou restant à définir)?

Les deuxième et troisième questions font référence à l'hypothèse formulant la notion de **perception globale** (diffusion instantanée) d'une information dans les approches synchrones [AND92-b]. La réponse à ces deux questions est tributaire, pour une large part, de la réponse donnée à la première question.

Nous formulons dans ce qui suit une proposition de modèle qui élabore et exploite des réponses aux questions soulevées. Par la suite, et pour les besoins de notre propre méthodologie de commande, nous développons de nouvelles macroactions dans le prolongement du contexte original du forçage.

II-1 Modèle proposé.

II-1-1 Hiérarchie: définition et limitation.

Le modèle proposé repose sur l'**hypothèse fondamentale** de la limitation du champ d'action des Grafcets partiels d'un niveau NH_i, aux Grafcets partiels du niveau immédiatement inférieur soit NH_{i+1}.

En effet, hiérarchiser un problème consiste à reléguer une partie ou la totalité de ce problème à un autre niveau. Pour résoudre cette partie de problème, ce dernier niveau peut à son tour se décharger sur le niveau suivant et ainsi de suite.

Dans le contexte de la commande, un niveau qui reçoit un ordre d'un niveau supérieur doit prendre toutes les dispositions pour mener à terme les actions nécessaires (en particulier donner des ordres au niveau inférieur), puis pour rendre compte de la fin d'exécution au niveau demandeur. Ceci est le schéma classique du "maître" et de "l'esclave" que l'on oppose souvent au schéma de coopération.

Conformément à cette notion de hiérarchie, il est tout à fait logique qu'un niveau NH_i d'un Grafcet global ne puisse gouverner que le niveau immédiatement inférieur soit NH_{i+1}. Il reste à charge de ce dernier de répercuter les ordres reçus sur le niveau NH_{i+2} et ainsi de suite.

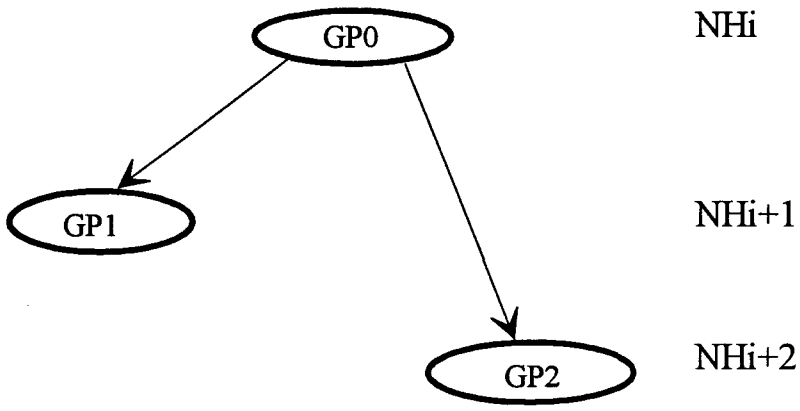


figure 4.2

En vertu de cette formulation, l'arc représentant un ordre de forçage d'un Grafcet partiel de numéro 0 (noté GP_0) d'un niveau NH_i sur un Grafcet GP_2 d'un niveau NH_{i+2}, tel que illustré par la figure 4.2, ne peut exister.

II-1-2 Ordre et situation.

Dans le même ordre d'idées on imagine facilement qu'un Grafcet partiel qui gouverne un autre Grafcet partiel puisse connaître instantanément la situation de celui-ci. En revanche, un Grafcet partiel gouverné ne peut connaître du Grafcet partiel gouvernant, que les ordres en provenance de celui-ci (figure 4.3).

Par ailleurs, le modèle proposé limite la connaissance de l'état du niveau inférieur (NH_{i+1}) que doit posséder un Grafcet partiel d'un niveau NH_i, aux situations des Grafcets partiels qu'il gouverne à ce même niveau. Ainsi, dans la figure 4.3, le Grafcet GP_0 ne connaît pas la situation du Grafcet GP_{12} du fait qu'il ne le gouverne pas.

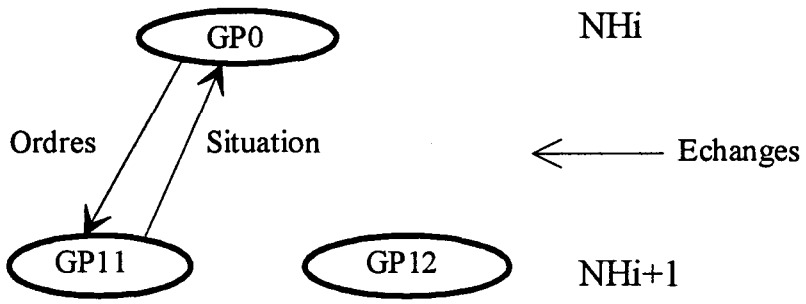


figure 4.3

En outre, dans le sens horizontal (même niveau), la communication entre Grafcets partiels se fait par la connaissance instantanée de situations. En effet, un Grafcet partiel donné doit avoir connaissance à chaque instant de la situation de chaque Grafcet partiel du même niveau hiérarchique (figure 4.4).

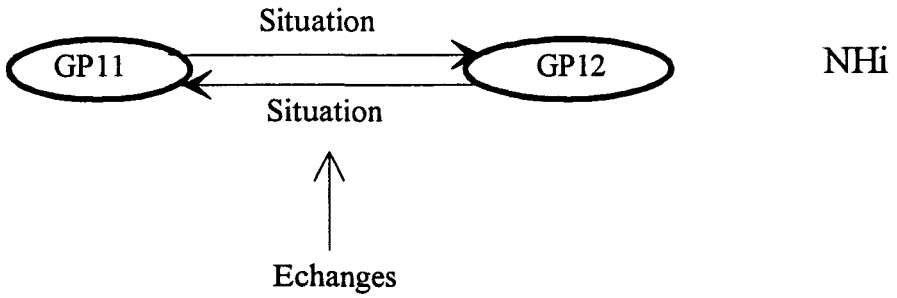


figure 4.4

II-1-3 Domaine de visibilité de la règle de forçage 2-b.

Enfin puisqu'un Grafcet partiel d'un niveau NH_i ne peut forcer que des Grafcets partiels du niveau NH_{i+1} et que par ailleurs la situation d'un Grafcet partiel forcé au niveau NH_{i+1} n'est pas visible au niveau immédiatement inférieur à celui-ci soit NH_{i+2}, il est inutile de répercuter l'effet produit par le forçage sur les niveaux inférieurs au niveau NH_{i+1}. En outre, cette information doit être disponible instantanément sur tout le niveau NH_{i+1}.

Le domaine d'application de la règle 2-b de forçage **concerne** donc **uniquement** le niveau NH_{i+1}, c'est à dire **le niveau où se situe le Grafcet partiel forcé.**

La figure 4.5 illustre cette formulation. Dès que la variable a passe à un, l'ordre de forçage est émis en direction du Grafcet GP_5. En vertu des règles 1 et 2-a du forçage, le Grafcet GP_5 prend immédiatement la situation {8}. De plus, la règle 2-b stipule que d'une part l'information GP_5 forcé à {8} est disponible partout au niveau NH_{i+1} et d'autre part que toute situation à ce niveau n'existe qu'en tenant compte de cette information. Les transitions 10.1 et 10.2 sont donc franchies simultanément (X8 = 1 et a = 1) (figure 4.6).

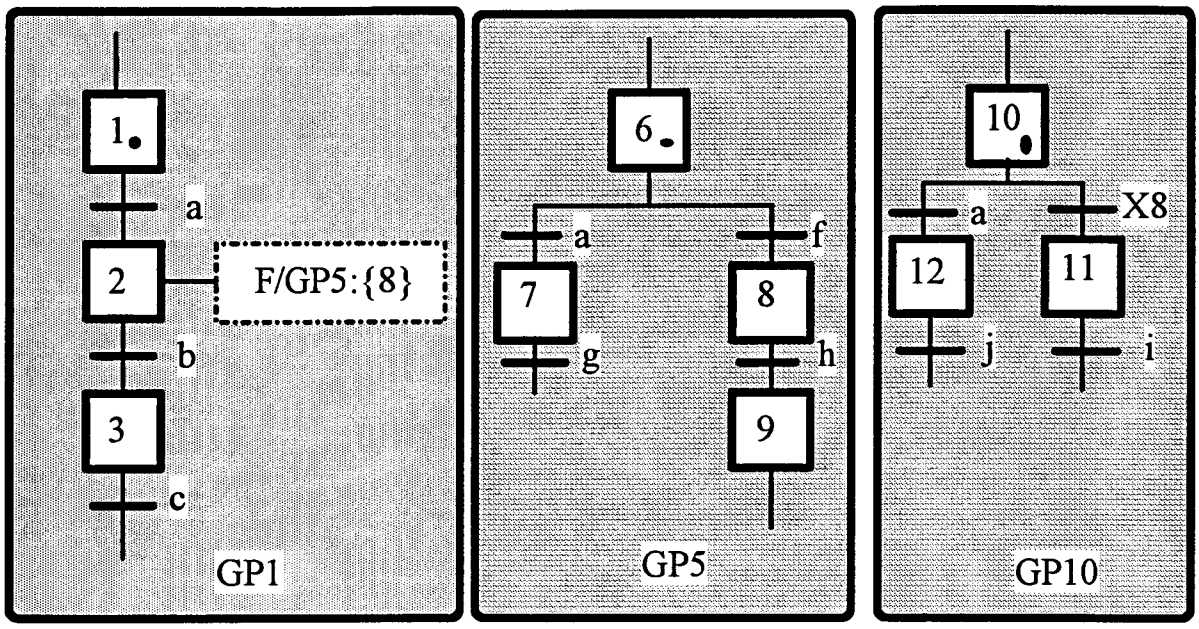


figure 4.5

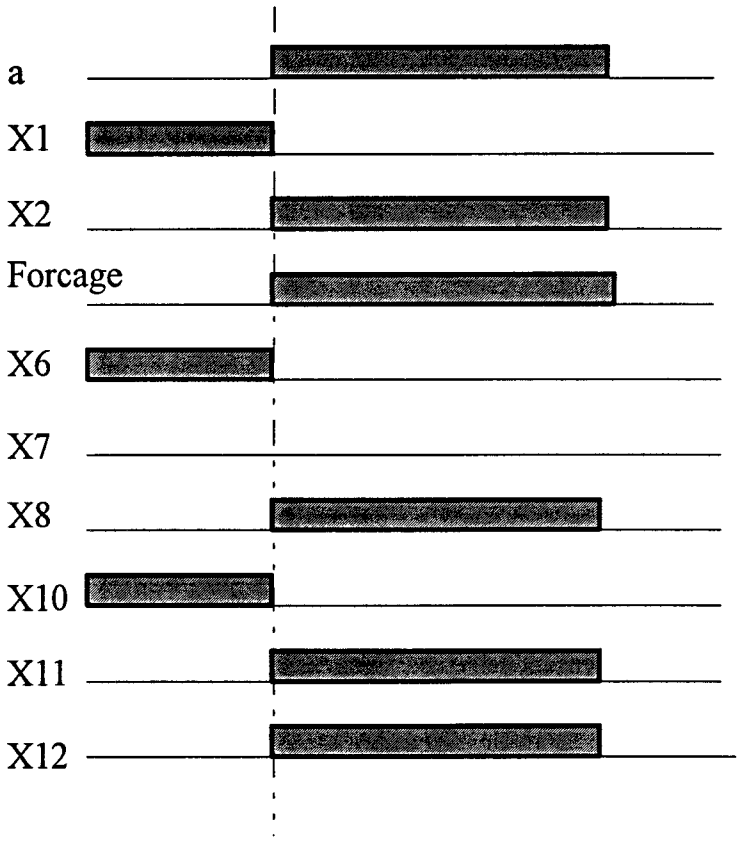


figure 4.6

II-2 Extensions de macroactions(figure 4.7).

Les macroactions qui existent actuellement dans la littérature pour le modèle Grafcet telles que: forçage, figeage, figer et libérer, sont des macroactions qui, lorsqu'elles sont émises par un Grafcet partiel, agissent sur la totalité des étapes ou transitions du Grafcet partiel gouverné.

Dans certains cas de figure, on a besoin de modifier uniquement l'état de certaines étapes ou de ne figer qu'un nombre restreint de transitions. Nous avons introduit un second type de macroactions que nous appelons **ordre partiel** par opposition au premier type que nous appelons **ordre total**.

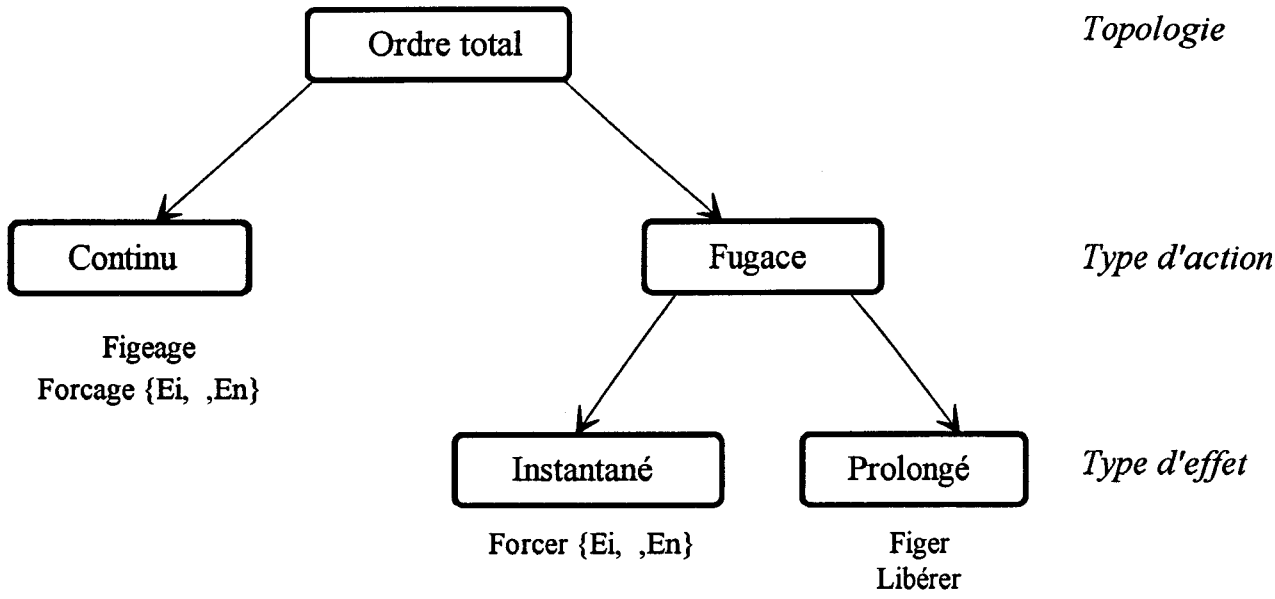
Chacun de ces deux types peut lui-même se décomposer en deux sous-types selon la nature de l'action qui génère l'ordre: **continu** ou **fugace** (impulsionnel). Le sous-type fugace peut, à son tour, se décomposer, selon la durée de l'effet de l'ordre, en deux classes: **instantané** ou **prolongé** (figure 4.7)

Chaque ordre cité est, tout comme le forçage, un **ordre interne consécutif à une évolution**. Certains ordres rentrent en conflit avec les règles d'évolution ou sont sans effet devant celles-ci, alors que d'autres ne présentent aucun problème vis à vis de ces mêmes règles. Aussi, afin d'assurer la cohérence entre les divers ordres du formalisme Grafcet conformément aux règles qui régissent le forçage et à la hiérarchie du Grafcet (**hypothèses d'homogénéité**):

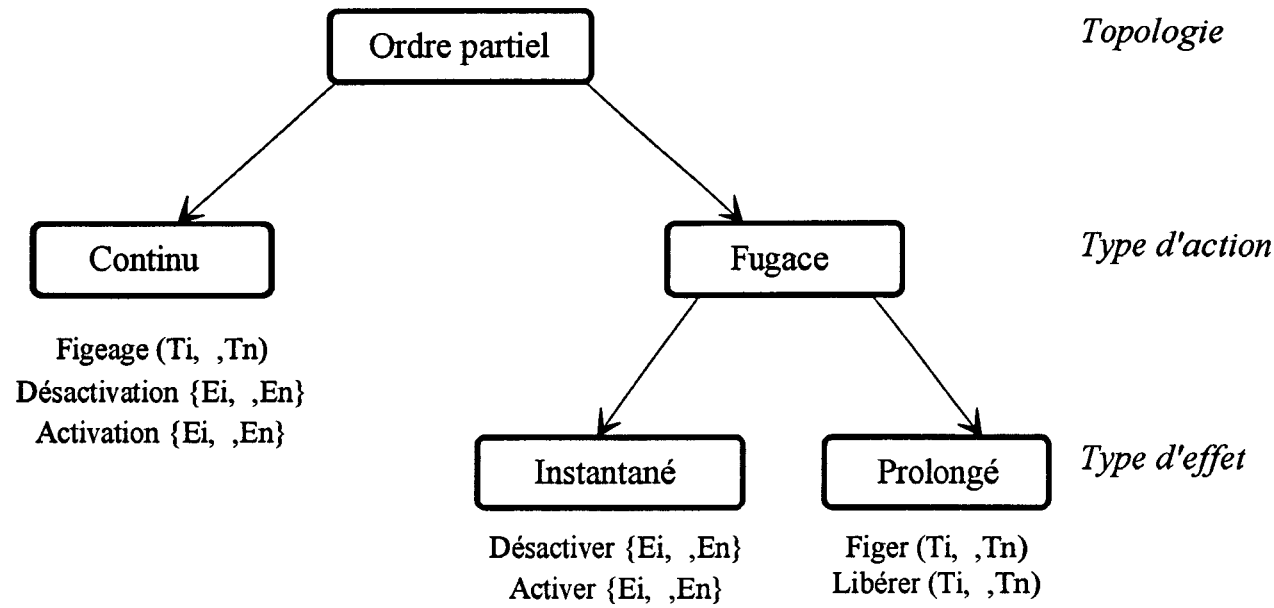
- nous préconisons la conservation des règles 1 et 2-b pour tous les ordres, d'autant plus que ces règles se réfèrent à la notion de perception globale dans le Grafcet.

- nous suggérons la conservation de la priorité des ordres sur les évolutions du Grafcet .

Cependant, les natures diverses des ordres existant et le fait que certains parmi ceux-ci peuvent entrer en conflit avec les règles d'évolution suggèrent la nuance de la règle 2-a suivant la macroaction considérée.


Ordre total
Topologie
Continu
Fugace
Type d'action
Figeage
Forcage {Ei, ,En}
Instantané
Prolongé
Type d'effet
Forcer {Ei, ,En}
Figer
Libérer

-a-


Ordre partiel
Topologie
Continu
Fugace
Type d'action
Figeage (Ti, ,Tn)
Désactivation {Ei, ,En}
Activation {Ei, ,En}
Instantané
Prolongé
Type d'effet
Désactiver {Ei, ,En}
Activer {Ei, ,En}
Figer (Ti, ,Tn)
Libérer (Ti, ,Tn)

-b-

figure 4.7

II-2-1 Ordre total continu: cas du Figeage.

La macroaction *figeage* consiste à figer toute évolution d'un Grafcet partiel gouverné. Ainsi, dans la figure 4.8, tant que l'étape 10 est active, l'évolution du Grafcet GP_5 est figée.

L'application de l'ordre de figeage ne rentre pas en conflit avec les règles d'évolution. Cependant, sans l'hypothèse de priorité de l'ordre sur les évolutions du modèle, la situation à laquelle serait figé le Grafcet GP_5 serait {7,..}. Alors que si l'on tient compte de cette même hypothèse, la situation atteinte est {6,..}. Il ne s'agit pas d'un conflit mais de comportements différents. L'hypothèse d'homogénéisation des règles confère donc, dans certains cas de figure, un comportement différent de celui du figeage classique.

Dans la nouvelle formulation, le figeage reste prioritaire sur les évolutions du modèle tant que l'étape contenant l'action qui l'a généré reste active (étape 10 figure 4.8). **La règle 2-a du forçage reste la même pour le figeage.**

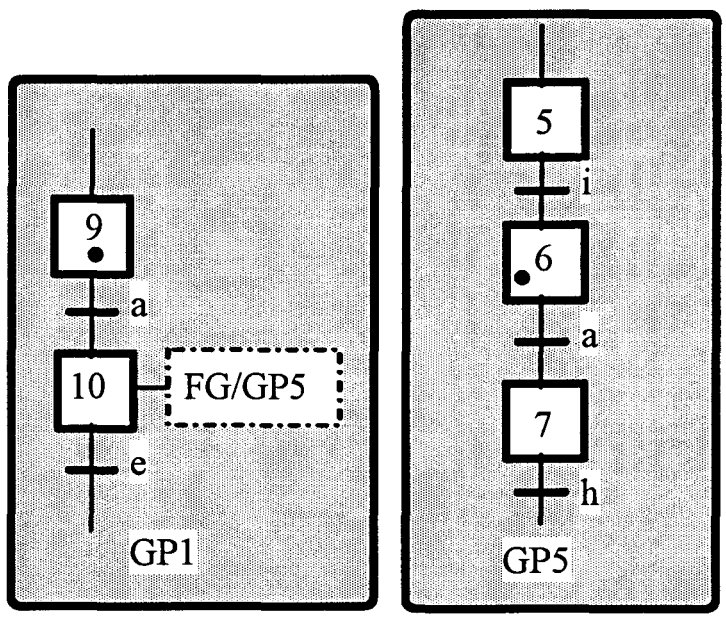


figure 4.8

II-2-2 Ordre total fugace instantané: cas de Forcer.

La macroaction *forcer* est la macroaction impulsionnelle équivalente à l'ordre continu forçage. Elle s'exerce instantanément sur un Grafcet partiel et libère ensuite l'évolution de ce Grafcet. Tout comme la macroaction *forçage,* l'ordre forcer peut entrer en conflit avec la règle 4 d'évolution. La priorité conférée à l'ordre forcer sur les règles d'évolution permet de supprimer ce conflit. **La règle 2-a demeure valable pour ce cas, tout en sachant que l'ordre est de nature fugace.**

II-2-3 Ordre total fugace prolongé: cas de Figer suivi de Libérer.

La macroaction *figer* suivie de *libérer* est l'équivalent en actions impulsionnelles de l'ordre figeage. Comme c'est le cas de l'ordre figeage, l'ordre figer n'entre pas en conflit avec les règles d'évolution. L'utilisation des hypothèses d'homogénéisation peut conférer dans certains cas, comme pour le figeage, un comportement différent du comportement classique.

L'ordre reste valide et prioritaire sur l'évolution du modèle, même lorsque l'étape qui l'a généré n'est plus valide (étape 10 figure 4.9). Il le reste jusqu'à ce que l'ordre libérer soit émis. Ce dernier ordre redonne aux règles d'évolution leur priorité.

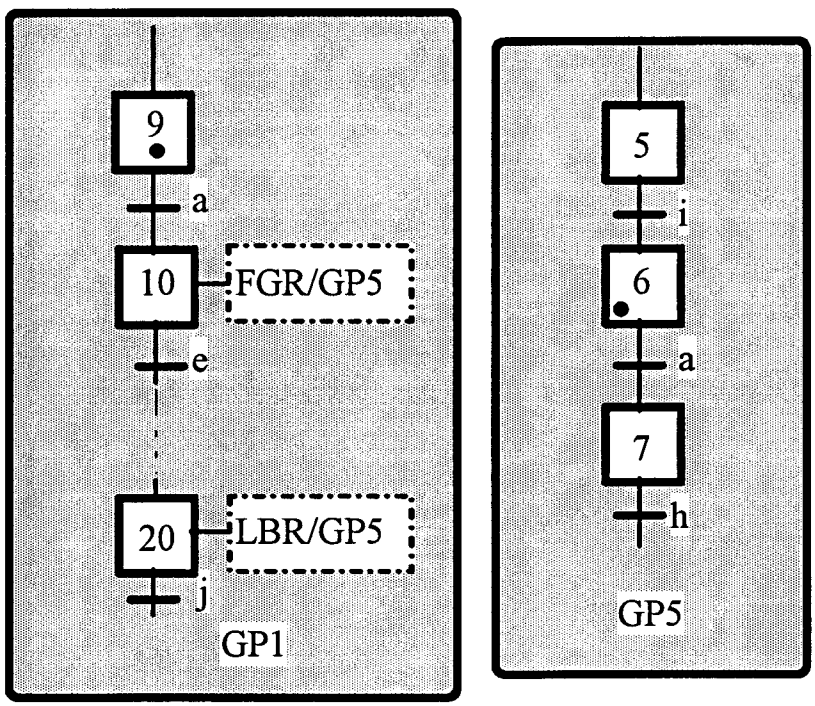


figure 4.9

La règle 2-a devient:

A toute apparition d'une nouvelle situation, l'application d'un ordre figer est prioritaire à toute activité du modèle tant que l'ordre libérer n'a pas été appliqué.

II-2-4 Ordre partiel continu.

Les ordres partiels agissent sur une partie d'un Grafcet mais autorisent l'évolution du modèle pour la partie restante du Grafcet. Le type ordre partiel continu est un ordre partiel qui est émis en tant qu'action continue. Nous dénombrons dans ce type trois exemples: *figeage {Ti, ,Tj}*, *désactivation {Ei, ,Ej}* et *activation {Ei, ,Ej}*.

L'ordre figeage {Ti, ,Tj} consiste à figer les évolutions des transitions Ti, ,Tj . L'ordre désactivation {Ei, ,Ej} consiste à désactiver les étapes Ei, , Ej tant que l'ordre est valide. L'ordre activation {Ei, ,Ej} consiste, contrairement au précédent ordre, à activer les étapes désignées tant que l'étape ayant émis l'ordre reste active.

L'ordre désactivation {Ei, ,Ej} que nous avons étudié de plus près, peut être sans effet lorsqu'il est appliqué simultanément avec la règle d'évolution 5. La figure 4.10 illustre cela.

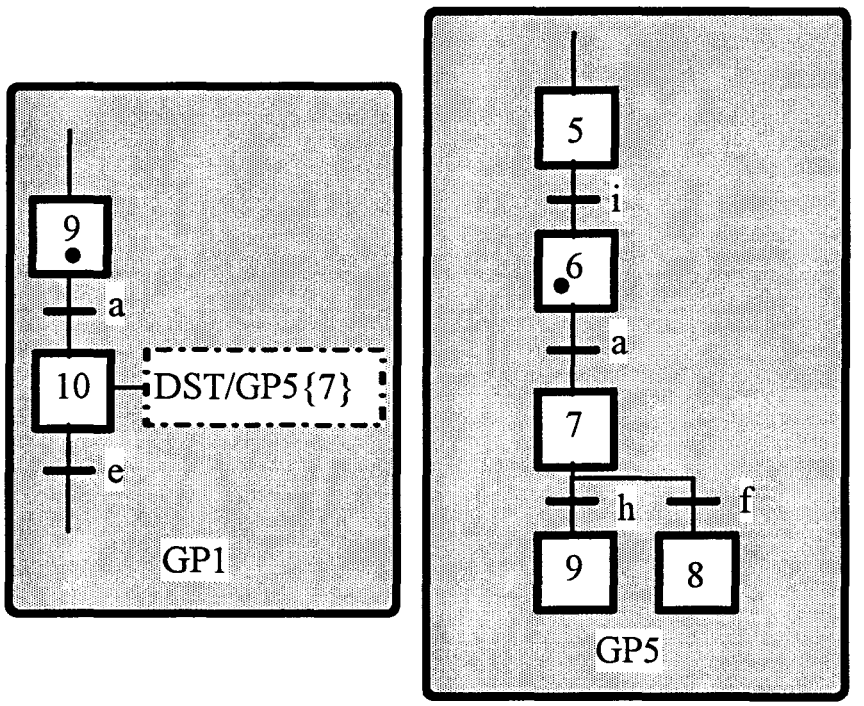


figure 4.10

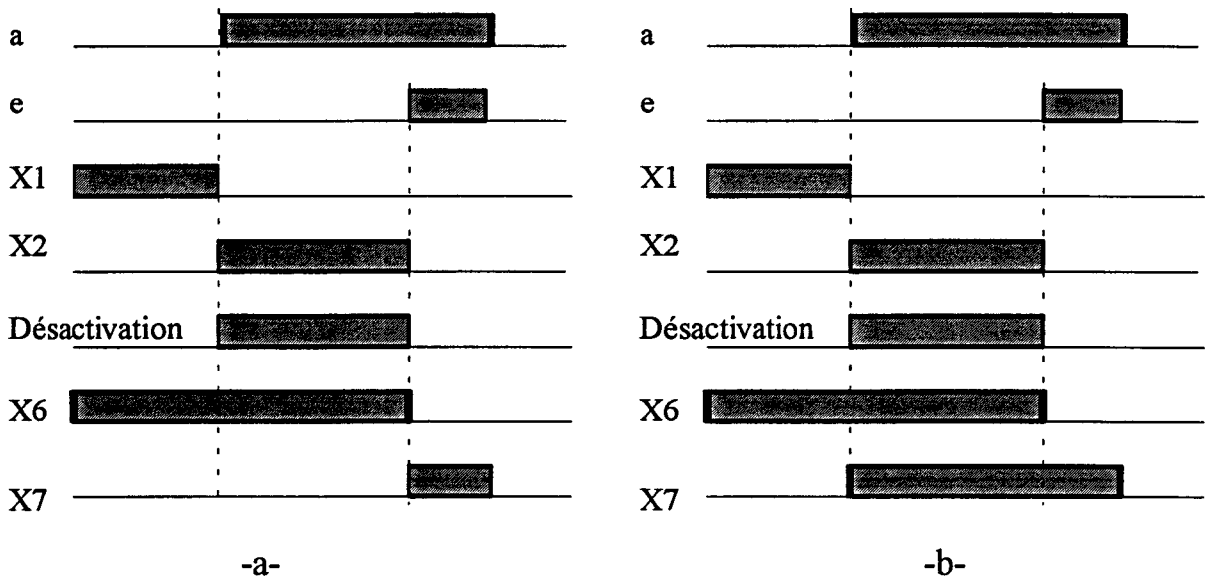


figure 4.11

En effet, sans application des hypothèses d'homogénéisation (l'hypothèse de priorité en particulier), et du fait de l'existence de la règle d'évolution 5 (qui stipule qu'une étape simultanément activée et désactivée reste active), les sorties obtenues sont représentées en figure 4.11-a. Ceci revient à considérer que la règle 5 est prioritaire par rapport à l'ordre. La macroaction n'a donc pas d'effet. Dans le cas contraire, c'est à dire priorité de l'ordre sur l'évolution du modèle, on obtient les sorties illustrées par la figure 4.11-b.

La règle 2-a doit nécessairement exprimer la priorité de l'ordre sur les règles d'évolution. En outre, une fois l'ordre appliqué en priorité, on doit laisser évoluer le modèle pour les autres parties (étapes et transitions non impliquées dans l'ordre) sans remettre en cause la priorité de l'ordre en cas de conflit.

Cette règle s'énonce donc comme suit:

A toute apparition d'une nouvelle situation l'application d'un ordre partiel continu est prioritaire, puis simultanée avec priorité à cet ordre en cas de conflit, à toute évolution du modèle.

II-2-5 Ordre partiel fugace instantané.

Le type ordre partiel fugace instantané est un type à rapprocher du type ordre total fugace instantané. La différence entre les deux réside dans le fait que le premier agit uniquement sur une partie du Grafcet partiel gouverné, alors que le second agit sur sa totalité. C'est le cas des macroactions *désactiver {Ei, ,En}* et *activer {Ei, ,En}*. Lorsque l'ordre est émis, il est immédiatement exécuté par le Grafcet partiel gouverné. Celui-ci reprend son fonctionnement normal tout de suite après.

L'ordre désactiver, comme c'est le cas pour son homologue en continu, peut présenter un conflit avec la règle 5. L'hypothèse de priorité résout ce conflit. Du fait que cet ordre est de nature fugace et compte tenu de l'hypothèse de priorité, il n'est donc jamais utilisé simultanément avec les règles d'évolution contrairement à l'ordre continu équivalent. La règle 2-a telle qu'énoncée pour le forçage exprime bien son fonctionnement et ne nécessite pas de changement.

II-2-6 Ordre partiel fugace prolongé: cas de Figer.

La macroaction impulsionnelle *figer {Ti, ,Tn}* est l'équivalent dans le type partiel de l'ordre total figer. Lorsque l'étape dont l'action exécutant cet ordre, est activée, celui-ci est émis vers le Grafcet partiel concerné. Ce dernier fige les évolutions des transitions Ti, ,Tj jusqu'à ce qu'il reçoive l'ordre inverse: libérer. Entre temps le Grafcet partiel gouverné évolue pour le reste des transitions tout en restant conforme à ce figeage. La règle 2-a doit exprimer l'hypothèse de priorité ainsi que la simultanéité du figeage et de l'évolution. Elle peut s'énoncer ainsi:

A toute apparition d'une nouvelle situation, l'application d'un ordre partiel fugace prolongé figer {Ti, ,Tn} est prioritaire, puis simultanée avec priorité à cet ordre en cas de conflit, à toute évolution du modèle tant que l'ordre inverse libérer {Ti, ,Tn} n'a pas été émis.

III DESCRIPTION DE LA COMMANDE.

Le premier paragraphe de ce chapitre a mis en évidence l'idée principale de la démarche originale développée pour la spécification de la commande. Ce paragraphe développe cette méthodologie:

- en reprenant la méthode utilisée pour la structuration des applications exposées au chapitre précédent.
- en intégrant en plus des contraintes temporelles, d'une part les problèmes de gestion des événements acquittés **occurrence par occurrence** et d'autre part les arrêts d'urgence.
- en reprenant le modèle proposé pour la hiérarchie et les macroactions dégagées pour le Grafcet.

La méthodologie de la commande en fonctionnement normal (sans tenir compte de l'aspect surveillance à dissocier de la surveillance temporelle) se résume en une hiérarchisation de la commande en trois niveaux formant un Grafcet global unique. Les trois niveaux hiérarchiques (notés chacun NH_i i = 1, 2, 3)) sont (figure 4.12):

- Arrêt d'Urgence et Reprise éventuelle.
- Surveillance Temporelle, Décision et Gestion des événements à acquittement occurrence par occurrence.
- Application (telle que définie au chapitre précédent).

Le Grafcet global envoie, via le niveau Application, des ordres au procédé et reçoit de celui-ci des comptes rendus. L'environnement (utilisateur ou niveau supérieur de la commande) envoi des ordres (commandes, demandes de renseignements..) au Grafcet global. Celui-ci doit éventuellement, en retour, renvoyer des comptes rendus ou des informations (figure 4.12).

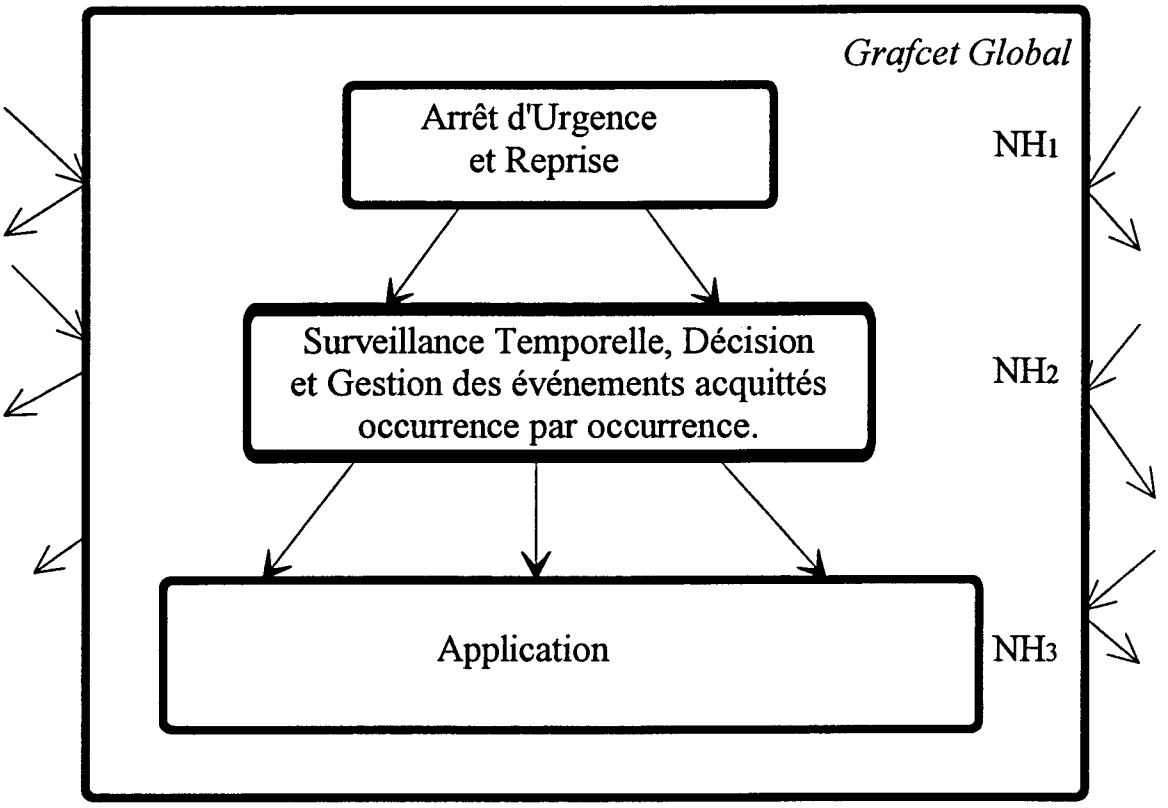


figure 4.12

III-1 Niveau hiérarchique 3: Application.

La structuration de l'application a été traitée au chapitre précédent. Ce paragraphe situe le passage de l'application structurée à l'application structurée dans la typologie Grafcet. Ceci se fait essentiellement en trois étapes:

1. avant d'aborder la suite il est nécessaire d'opérer la déconnexion de tous les arcs partant d'un processus et aboutissant à un autre dans l'application structurée. Tout arc est remplacé par les activités des étapes correspondantes. La figure 4.13 présente un exemple qui concerne le cas le plus usuel. Les Grafcets mis en jeu deviennent ainsi des Grafcets connexes.

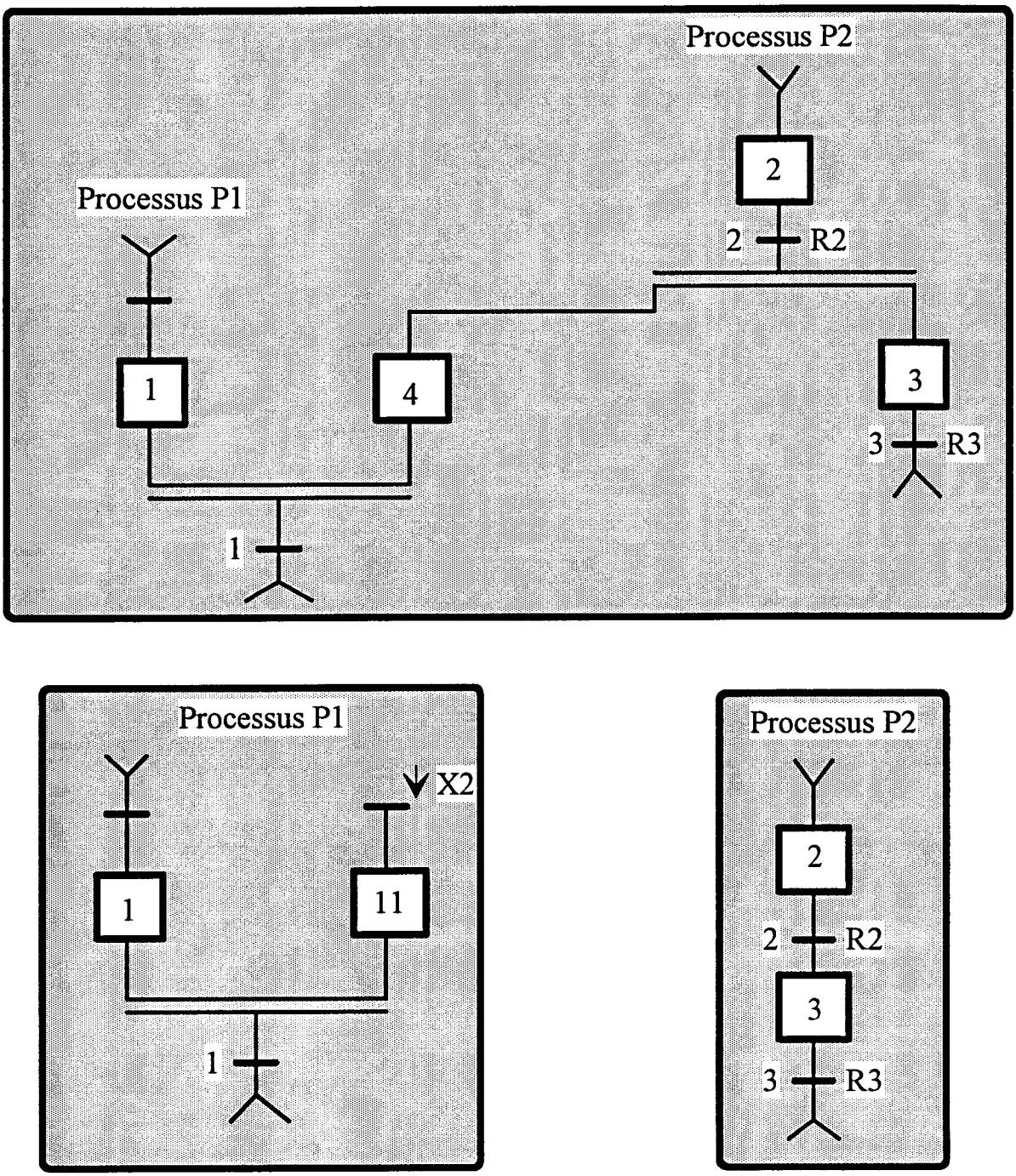


figure 4.13

2. Ensuite, conformément aux définitions données au chapitre précédent, on considère comme Grafcet partiel (exemple en figure 4.14):

   - chaque processus gestionnaire de ressource (Grafcet GP_{22}).
   - chaque processus de commande (Grafcets GP_{21}, GP_{23} et GP_{12}).
   - chaque processus de gestion de lieu (Grafcet GP_{11}).

3. Enfin, on associe un Grafcet partiel arrêt d'urgence et de suspension à chaque ressource (Grafcets G_{ar1} et G_{ar2} en figure 4.14).

III-2 Niveau hiérarchique 2: Surveillance Temporelle Décision et Gestion des événements à acquittements occurrence par occurrence.

On trouve ici des Grafcets partiels de trois types:

- **gestionnaire des contraintes temporelles:** le Grafcet est chargé de surveiller le déroulement temporel de l'application et de redéployer les ressources en cas de besoin. Pour illustrer cette fonction, considérons le cas où un système de vision, représenté par le Grafcet GP_{22} à la figure 4.14, est partagé par deux processus: un processus de localisation (GP_{21}) et un processus de test (GP_{23}).

  Par ailleurs, les opérations de localisation présentent des contraintes de temps relatives à la durée limitée pendant laquelle un objet reste sous la caméra. En outre, afin de respecter les contraintes temporelles, il est autorisé de faire une préemption du système de vision lors de son utilisation par le processus de test, ceci au profit du processus de localisation. Une partie du problème est exprimée au niveau Application. En effet, celui-ci donne priorité au processus de localisation en cas de demandes simultanées.

  Un Grafcet partiel de niveau NH_2 est chargé lorsqu'il juge que les contraintes temporelles risquent d'êtres enfreintes, d'effectuer la séquence d'opérations suivante:

  - ***suspendre*** *le processus de test en cours d'exécution par initialisation du Grafcet:* ***Arrêt d'urgence et Suspension du Système de vision*** *(Grafcet GP_{ar2}).*
  - *attribuer le système de vision, à l'issue de la suspension, au processus de localisation (Grafcet GP_{21})* ***(préemption).***
  - ***reprendre ou achever,*** *à l'issue de l'activité lancée à l'étape précédente, l'exécution de l'opération de test qui était en cours.*

- **gestionnaire d'événement** à acquittement occurrence par occurrence. En effet, pour les primitives de définition des liens de synchronisation événement-action du chapitre II, les acquittements ont lieu par groupes d'occurrences. Dans le cas où les occurrences sont nécessairement acquittées une par une, il faut créer un processus périphérique pour gérer les acquittements de ces événements. Ce type de processus est disposé à ce niveau de la hiérarchie plus par commodité que par nécessité (uniquement pour soulager le niveau Application). A chaque gestionnaire d'événement est associé un Grafcet partiel (GP_1 en figure 4.14). Il est possible d'avoir, pour certaines applications, les deux types de gestionnaires qui agissent sur le(s) même(s) Grafcet(s) partiel(s) du niveau Application. Il faut néanmoins prendre des précautions pour éviter l'activation simultanée de macroactions destinées au même Grafcet.

- **Grafcet auxiliaire**. Ce Grafcet est chargé de figer les Grafcets partiels de l'application puis de lancer les Grafcets partiels ***Arrêt d'urgence et Suspension***, du niveau hiérarchique 3 en cas d'arrêt d'urgence (GP_{aux1} et GP_{aux2} en figure 4.14).

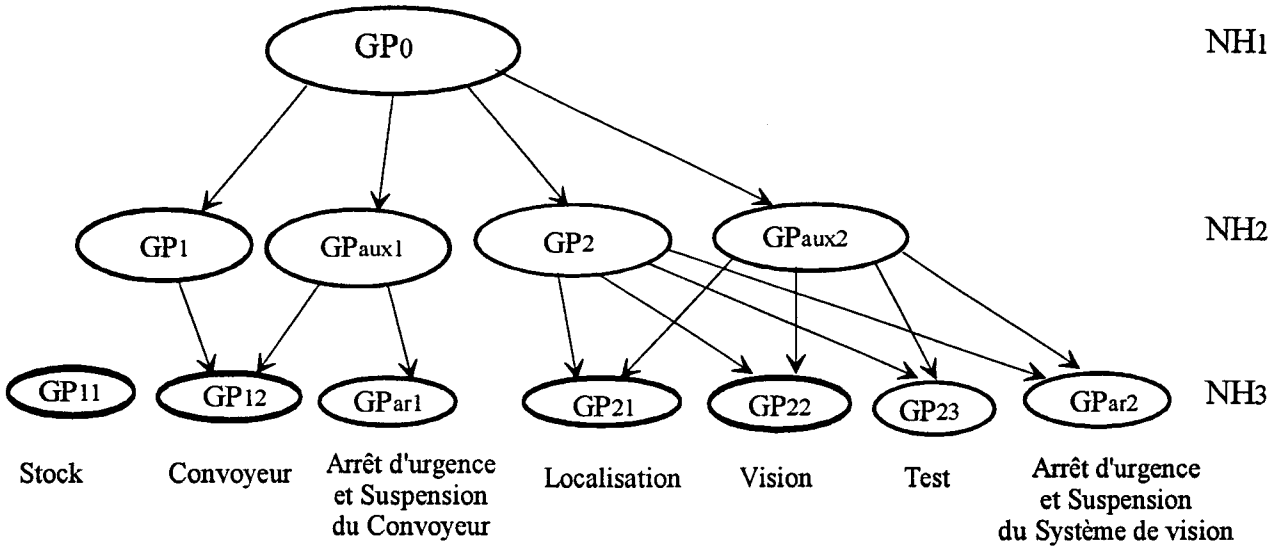


figure 4.14

III-3 Niveau hiérarchique 1: arrêt d'urgence et reprise.

A ce niveau, on trouve le Grafcet partiel (GP_0 en figure 4.14) chargé, en cas d'événement d'arrêt d'urgence, provenant soit de l'opérateur (environnement) soit du procédé (défaillance), de figer l'évolution des Grafcets partiels de gestion du niveau hiérarchique 2 et de lancer les Grafcets auxiliaires du même niveau. On peut aussi confier à ce niveau la fonction de reprise après arrêt ou défaillance (non représentée ici).

IV DESCRIPTION DETAILLEE DU NIVEAU HIERARCHIQUE 2.

IV-1 Gestion d'événements occurrence par occurrence (figure 4.15).

Soit A un type d'objet arrivant sur un convoyeur CV au point M. Chaque objet de ce type arrivant en M doit être transféré sur la table TT. Ce transfert s'effectue dans la mesure de la disponibilité du robot qui reste dédié à cette fonction. Le robot dispose pour chaque objet arrivant en M, de T secondes (le temps qu'il faut à cet objet pour parcourir le trajet MN) pour le prendre, avant qu'il ne disparaisse au delà du point N.

Au moment où arrive un objet en M, le robot peut être occupé à transférer un autre objet parvenu avant celui-ci. Si la libération du robot intervient avant que T secondes ne se soient écoulées, l'objet est prélevé sinon il est perdu. Du fait de la possibilité d'acquittement sur les données, on ne peut plus exprimer le transfert du convoyeur CV vers la table TT en termes classiques de producteur/consommateur.

On associe donc à l'arrivée de l'objet en M un événement Eva dont les occurrences ont chacune une durée de vie de T secondes. Lorsque T secondes s'écoulent pour une occurrence donnée, il y a émission d'un événement ACQ (Acquittement).

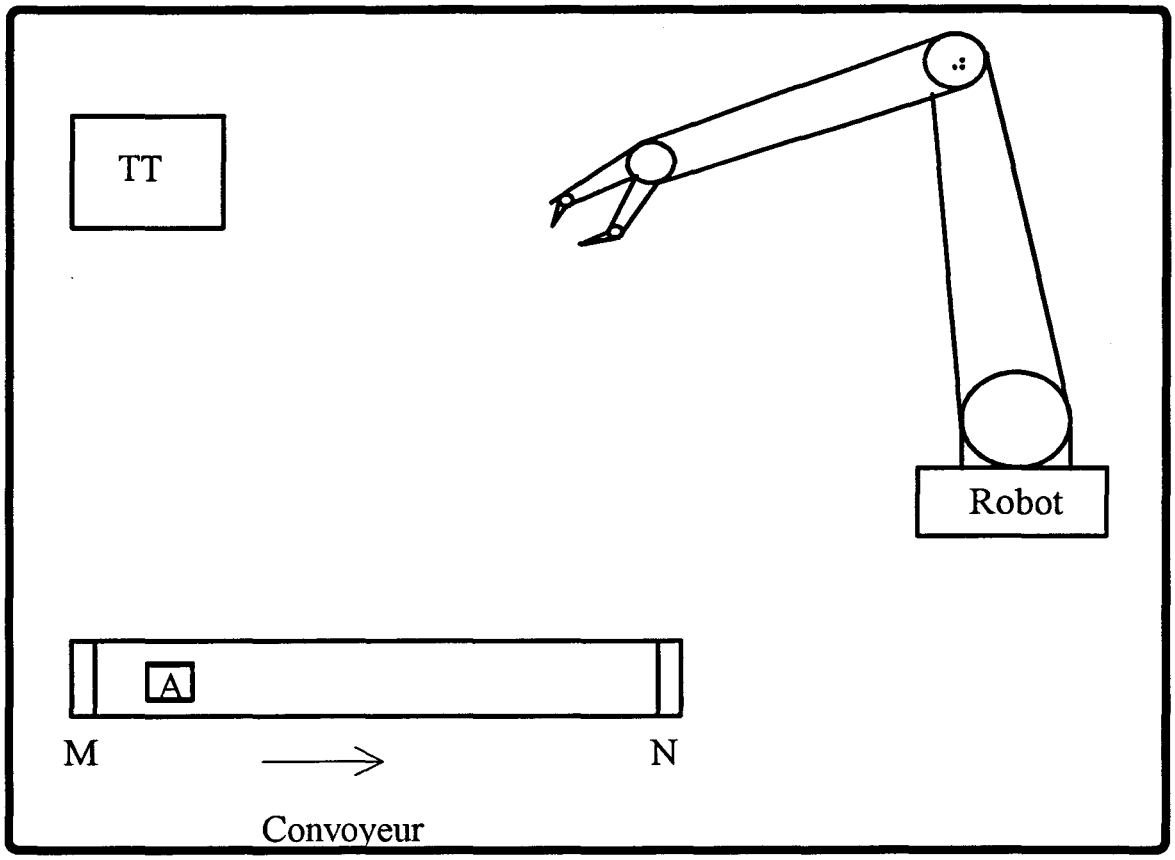


figure 4.15

Les six formes de liens de synchronisation événement-action avec leurs extensions (introduction de condition) ne suffisent pas à elles seules pour la description du processus TRANSFERT (CV, TT). En effet, dans l'esprit primitive lien de synchronisation, l'acquittement se fait pour le groupe d'occurrences mémorisées alors que dans notre exemple, l'acquittement doit intervenir séparément pour chaque occurrence. La figure 4.16 présente l'esprit de la solution envisagée.

Deux Grafcets GP_1 et GP_2 hiérarchisés respectivement aux niveaux NH_2 (Surveillance Temporelle, Décision et Gestion) et NH_3 (Application), servent à spécifier le cas exposé précédemment. En effet le Grafcet GP_1 mémorise toutes les occurrences de l'événement Eva et supprime les occurrences dont les acquittements sont parvenus.

Il a aussi pour rôle de faire parvenir au processus TRANSFERT (CV, TT), dès que celui-ci est prêt à accomplir une tâche (événement Ef parvenu), l'occurrence de l'événement Eva la plus anciennement mémorisée et non acquittée.

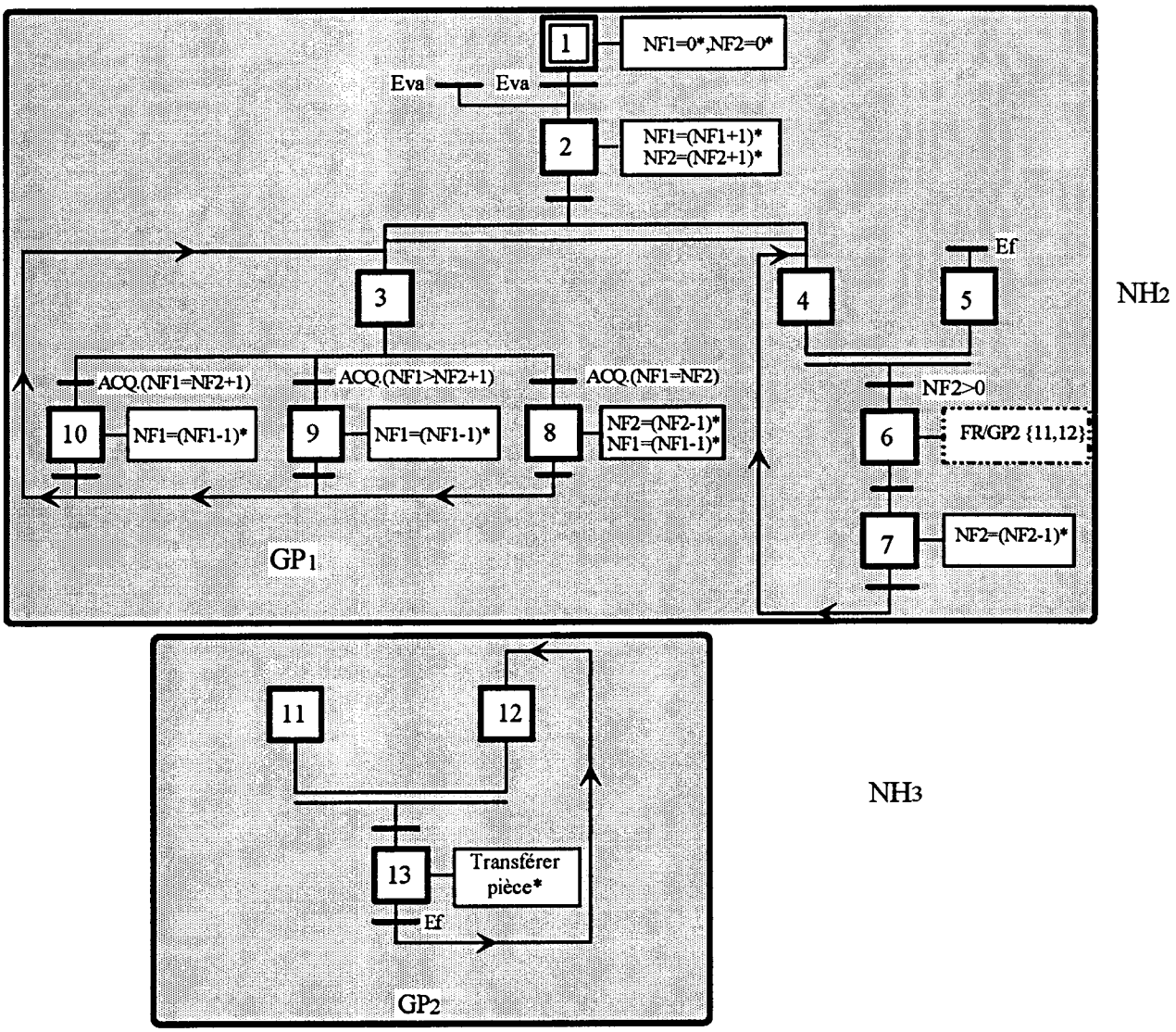


figure 4.16

Pour toutes ces raisons, le gestionnaire des événements Eva dispose de 2 compteurs NF1 et NF2 incrémentés sur chaque occurrence de l'événement Eva. Le compteur NF2, qui comptabilise les occurrences non encore servies, est décrémenté (si NF2 > 0) à chaque fois qu'une occurrence a été transmise au processus TRANSFERT (CV, TT) (forçage exécuté en étape 6).

Le compteur NF1 sert à compter le nombre d'occurrences encore vivantes (non acquittées). Lorsqu'un événement ACQ (Acquittement) survient trois cas peuvent se présenter:

- Les deux compteurs sont égaux. Il faut donc les décrémenter tous les deux car l'acquittement concerne l'occurrence comptabilisée en tête de NF2 et de NF1.

- NF1 = NF2 + 1. L'acquittement concerne l'occurrence en cours de consommation. Son acquittement ne devrait pas intervenir sauf dans le cas où le robot chargé d'effectuer le TRANSFERT (CV, TT) est une ressource partagée avec d'autres processus et que cette même ressource n'a pas encore été allouée au processus TRANSFERT (CV, TT).

- NF1 > NF2 + 1. L'acquittement concerne une occurrence déjà consommée.

Le processus TRANSFERT (CV, TT) est représenté par un lien de synchronisation du type: pour EVT futur faire <transférer pièce>. Le processus boucle ensuite sur lui même. L'événement EVT correspond à l'exécution de la macroaction *forcer* GP_2 *{11, 12}* en provenance du gestionnaire d'événement.

IV-2 Surveillance Temporelle et Décision.

L'exemple précédent a montré que pour certains cas il est nécessaire d'acquitter les événements occurrence par occurrence. Ceci s'effectue par la mise en oeuvre d'un gestionnaire d'événements. Par ailleurs, certains problèmes nécessitent la suspension puis la préemption d'actions au profit d'événements contraignants. Pour illustrer, considérons l'exemple suivant (figure 4.17).

Soit un robot chargé d'effectuer sur demande de l'opérateur les transferts successifs de deux types de pièces Pc1 et Pc2 d'un lieu CP vers un lieu TT. Pour la prise des pièces du lieu CP, le robot a besoin d'avoir des informations sur la position et l'orientation de l'objet désiré (Pc1 et Pc2). Un système de vision est chargé, à la demande, de fournir au robot de tels renseignements. Jusqu'ici le problème peut se résoudre par des relations du type producteur/consommateur.

Par ailleurs, trois types d'objets P1, P2 ou P3 arrivent d'une façon asynchrone sur le convoyeur dans un ordre aléatoire. L'arrivée d'un objet parmi ceux-ci est signalée par un événement Ev qui, comme dans l'exemple précédent admet pour chacune de ses occurrences une durée de vie de T secondes. L'arrivée d'un événement doit être

signalée au système de vision chargé de reconnaître le type d'objet arrivé (P1, P2 ou P3) et de communiquer le résultat au processus TRANSFERT (CV, TT).

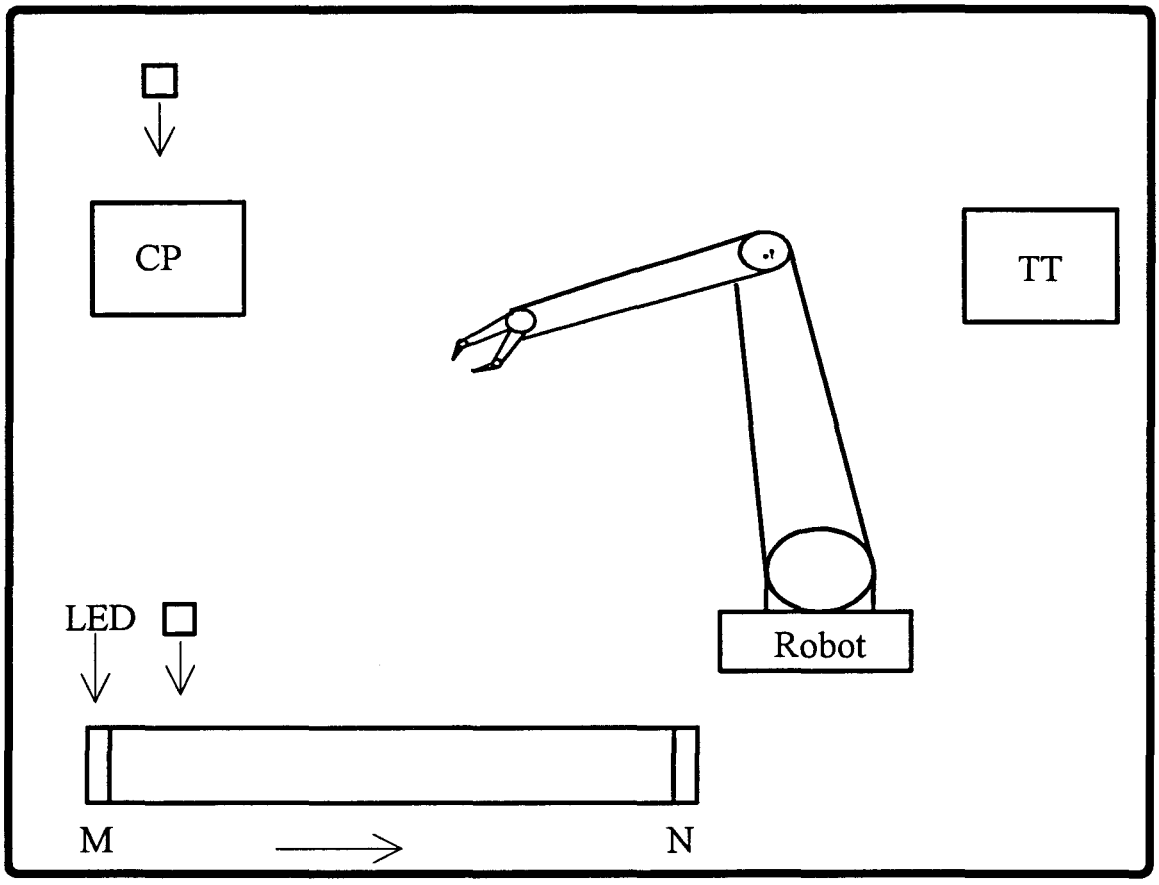


figure 4.17

Contrairement au cas précédent, au bout d'un délai d'attente raisonnable, on autorise la suspension d'une action en cours sur le robot et/ou sur le système de vision puis la préemption de ces entités au profit de l'occurrence de l'événement Ev se produisant. On suppose qu'entre deux occurrences successives de Ev le système a le temps d'exécuter les actions associées à la première occurrence.

La description fonctionnelle de cette application représentée en figure 4.18 distingue 6 processus (pour ne pas alourdir la représentation, on exclut le processus gérant les demandes de l'utilisateur).

Le processus IDENTIF (CV) est activé par l'événement Ev et élabore une information, parmi trois possibles (correspondant aux types de pièces), à destination du processus TRANSFERT (CV, TT).

Le processus LOCALISE (CP) est activé par une demande du processus TRANSFERT (CP, TT) auquel il rendra la réponse (information). Le processus TRANSFERT (CP, TT) se comporte à la fois comme producteur et comme consommateur du processus LOCALISE (CP) (deux communications simples).

La gestion du lieu CP est représentée par un processus G.L.S.I ayant deux consommateurs (un par type de pièces).

En outre, les processus IDENTIF (CV) et LOCALISE (CP) se partagent l'accès au système de vision avec priorité pour le processus IDENTIF (CV) et possibilité de suspension et de préemption au bout d'un délai Tsv (< T).

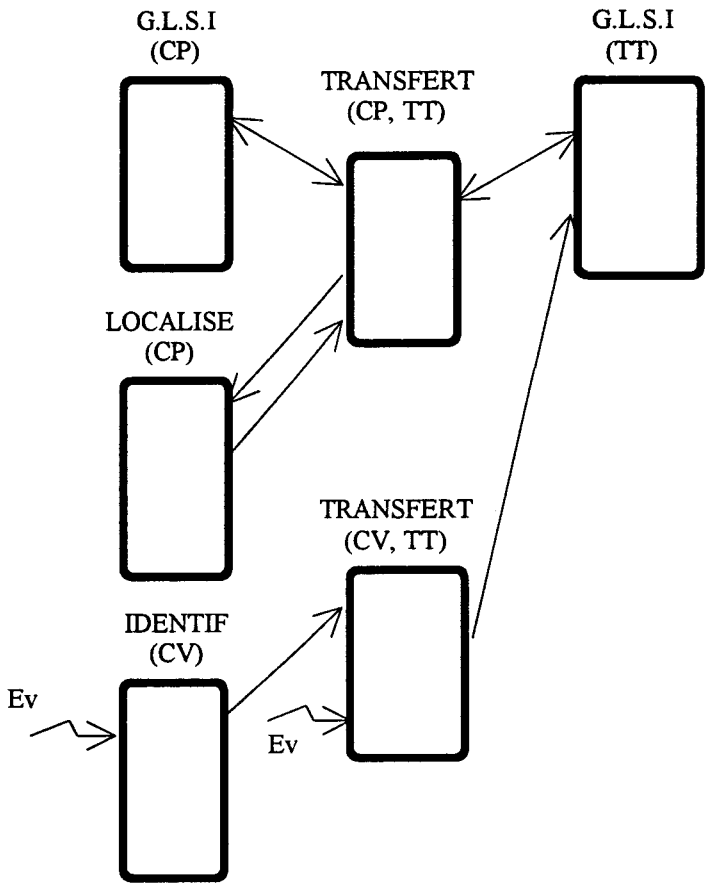


figure 4.18

Le processus TRANSFERT (CP, TT) est activé par demande de l'opérateur. Il se comporte, en outre, comme consommateur du processus G.L.S.I (CP) et producteur du processus G.L.S.I (TT).

Le processus TRANSFERT (CV, TT) activé par occurrence de l'événement Ev, se comporte en tant que consommateur du processus IDENTIF (CV) et en tant que producteur du processus G.L.S.I (TT), sans toutefois d'autorisation préalable (on suppose qu'il y a toujours au moins une place disponible sur le lieu TT pour un objet en provenance du convoyeur: communication simple).

Les processus TRANSFERT (CV, TT) et TRANSFERT (CP, TT) se partagent l'accès au robot avec: priorité pour TRANSFERT (CV, TT), et possibilité de suspension puis de préemption, au bout d'un délai d'attente Tr.

Au niveau Application sont situés deux sites. Un site de niveau NH_3 représente une entité physique (robot ou système de vision) partagé par plusieurs processus ((TRANSFERT (CV, TT) et TRANSFERT (CP, TT)), ou (IDENTIF (CV) et LOCALISE (CP))).

Au niveau NH_2 on retrouve deux Grafcets partiels du type gestionnaire de contraintes temporelles (un par site). Les tâches de chaque Grafcet partiel sont: surveiller l'affectation d'une ressource aux processus, le cas échéant suspendre puis réaffecter la ressource, faire reprendre aux processus suspendus les états qui prévalaient avant la suspension.

Ainsi sur le site système de vision sont localisés:

1. Au niveau NH_2 (figure 4.19 et 4.20):

   - Le processus LOCALISE (CP) qui reçoit des demandes de localisation de pièces du type: Pc1 (étape 3 marquée) ou bien Pc2 (étape 4 marquée). Les étapes 1 et 2 mémorisent, dans l'ordre, deux demandes (c'est le cas le plus général où les deux producteur/consommateur sont différents).

     Pour la première demande arrivée, on fait une demande de réquisition de la ressource partagée système de vision (arc A). Une fois que la ressource est attribuée (arc B, étape 9 active), il y a exécution de la fonction désirée. Lorsque

cette exécution est terminée, la ressource partagée est libérée et le résultat est communiqué au producteur/consommateur correspondant (arc C).

- Le processus IDENTIF (CV) reçoit l'occurrence Ev (lien passé ou futur sur première occurrence de l'événement Ev), puis demande la ressource système de vision (arc D). Lorsque celle-ci lui est accordée (arc E, étape 23 active) il y a exécution de l'action correspondante. Suivant le résultat de l'action, il y a communication du résultat à l'étape correspondante du processus TRANSFERT (CV, TT) (arcs K, L et M) puis libération du système de vision (arc F).

- Le processus de gestion du système de vision reçoit 2 types de demandes (arc A et D, étapes 32 ou 33) et attribue la ressource avec priorité au processus IDENTIF (CV) (arc E) dès que le système de vision devient libre. Une fois que l'action est terminée, il reçoit la confirmation de libération de la ressource (arcs C et F, étapes 34 et 36).

2. Au niveau NH_1 (figure 4.21):

Le Grafcet GP_1 attend l'occurrence Ev. Dès que celle-ci se produit, une temporisation de Tsv secondes est lancée (représentant le délai d'attente au bout duquel intervient GP_1).

Durant le temps Tsv il est possible:

- que le processus IDENTIF (CV) soit le premier à demander la ressource. Si c'est le cas, sa demande sera satisfaite (X24=1) et le Grafcet GP_1 boucle sur l'étape initiale.

- soit que le processus LOCALISE (CP) libère la ressource (événement Efsv = 1), la demande du processus IDENTIF (CV) est alors satisfaite sans intervention. Le Grafcet GP_1 boucle sur l'étape initiale.

Si au bout de Tsv secondes l'événement Efsv ne s'est pas produit et que la réceptivité X24 est nulle alors le Grafcet partiel GP_1 intervient, en figeant les transitions 13 et 14 de GP_{11}, puis en suspendant l'action ou le programme en cours (cela nécessite, ainsi que pour la reprise de cette même action à l'étape 45, le lancement d'un Grafcet auxiliaire).

Le Grafcet GP_{12} (IDENTIF (CV)) est, à la fin de la suspension, forcé à l'étape 24 pour commander l'activité désirée. Lorsque celle-ci est terminée (Ep1 + Ep2 + Ep3), on déclenche la reprise de l'action suspendue dont la fin devrait se solder par l'événement Efsv. Les transitions 13 et 14 du Grafcet GP_{11} sont alors libérées. Il faut ensuite annuler la demande de la ressource par le processus IDENTIF (CV) (celle-ci ayant été satisfaite par forçage), ceci en désactivant les étapes 32 et 36 du Grafcet GPsv.

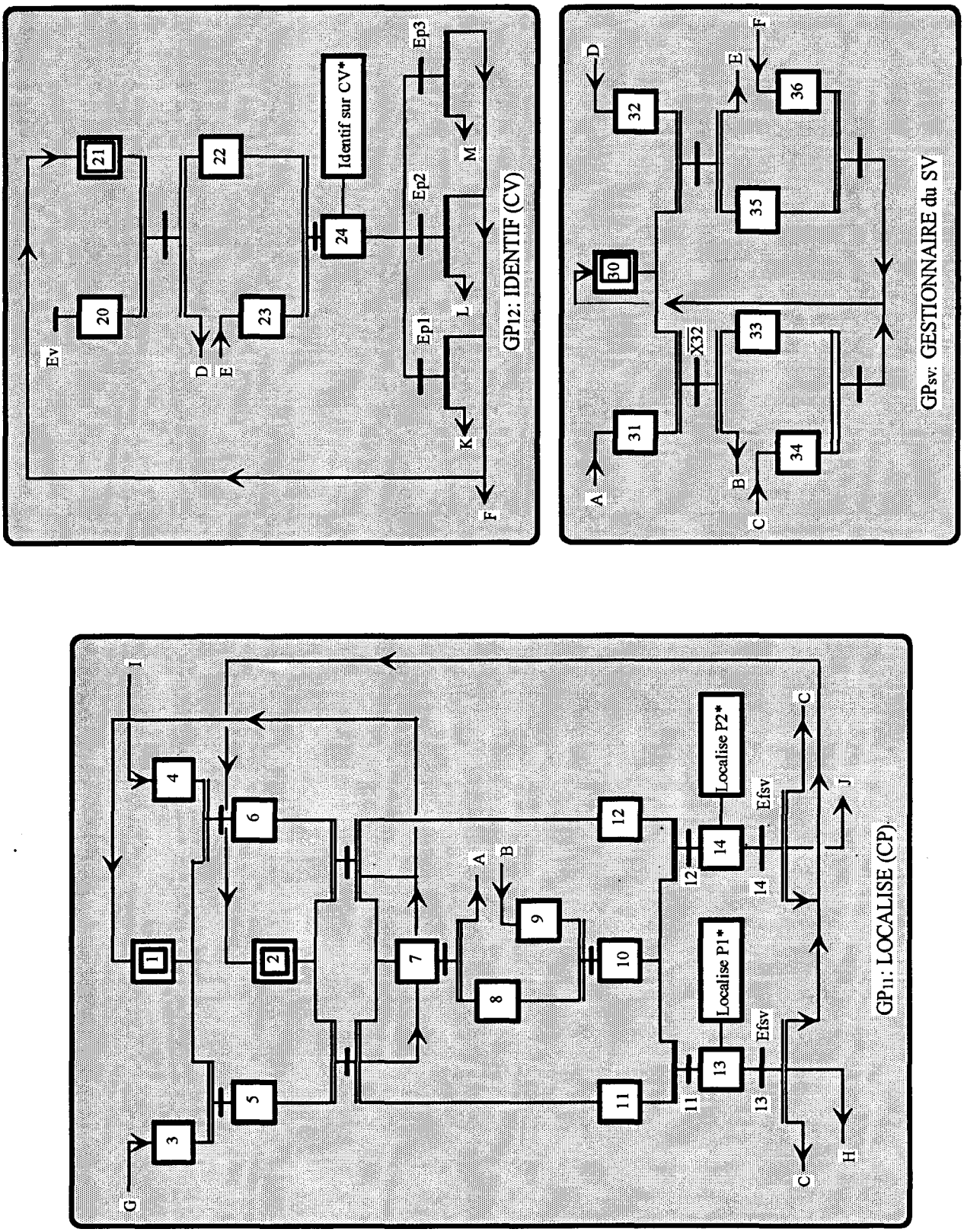

GP12: IDENTIF (CV)
Ev
20
21
22
23
24
D
E
Identif sur CV*
Ep1
Ep2
Ep3
K
L
M
F
GPsv: GESTIONNAIRE du SV
A
31
30
32
33
34
35
36
X32
B
C
D
E
F
GP11: LOCALISE (CP)
G
I
1
2
3
4
5
6
7
8
9
10
11
12
13
14
A
B
Localise P1*
Localise P2*
Efsv
Efsv
11
12
13
14
C
J
H
C

figure 4.19

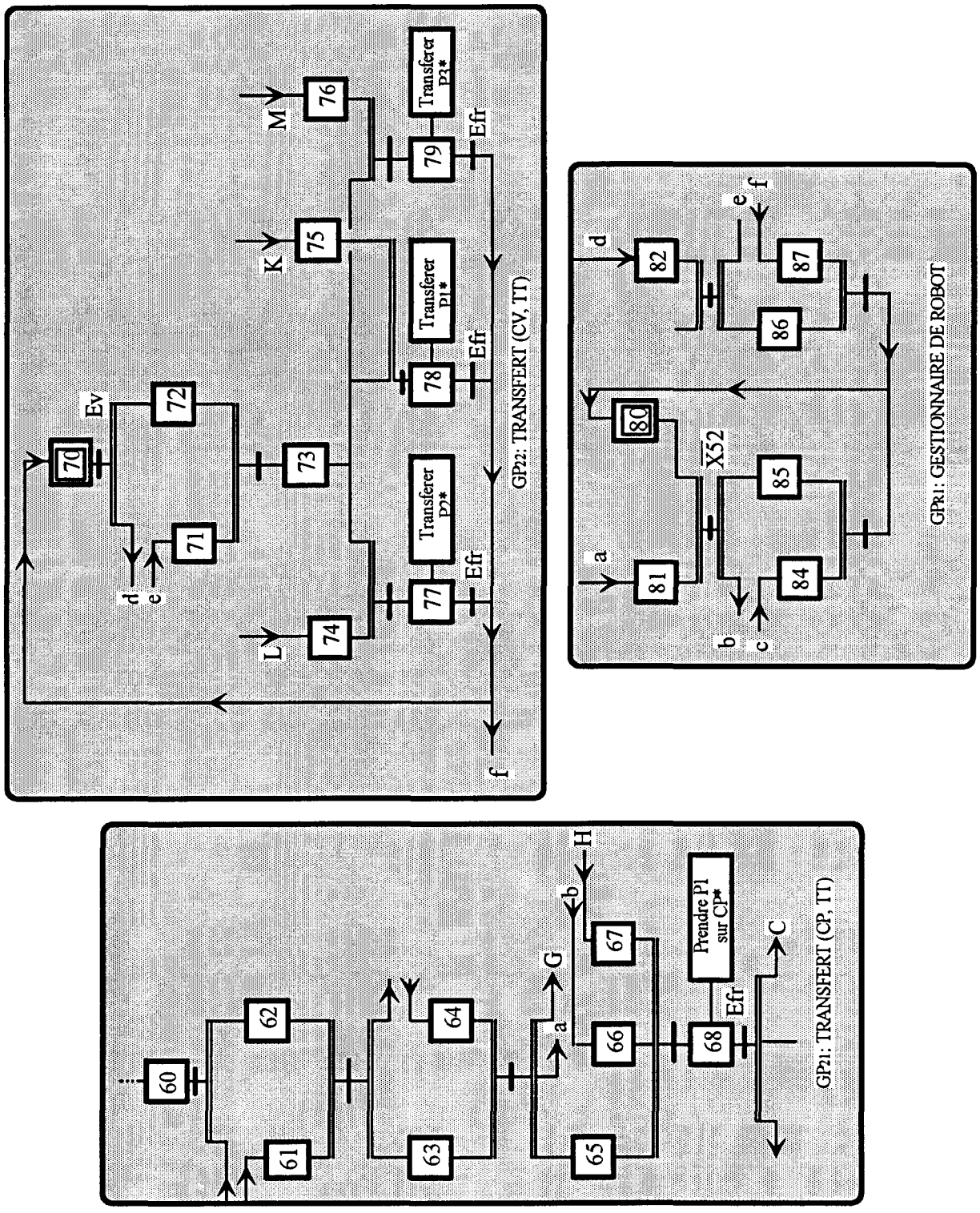

GP2: TRANSFERT (CV, TT)
GP1: GESTIONNAIRE DE ROBOT
GP2: TRANSFERT (CP, TT)
Transferer P2*
Transferer P1*
Transferer P3*
Prendre P1 sur CP*
Ev
Efr
X52

figure 4.20

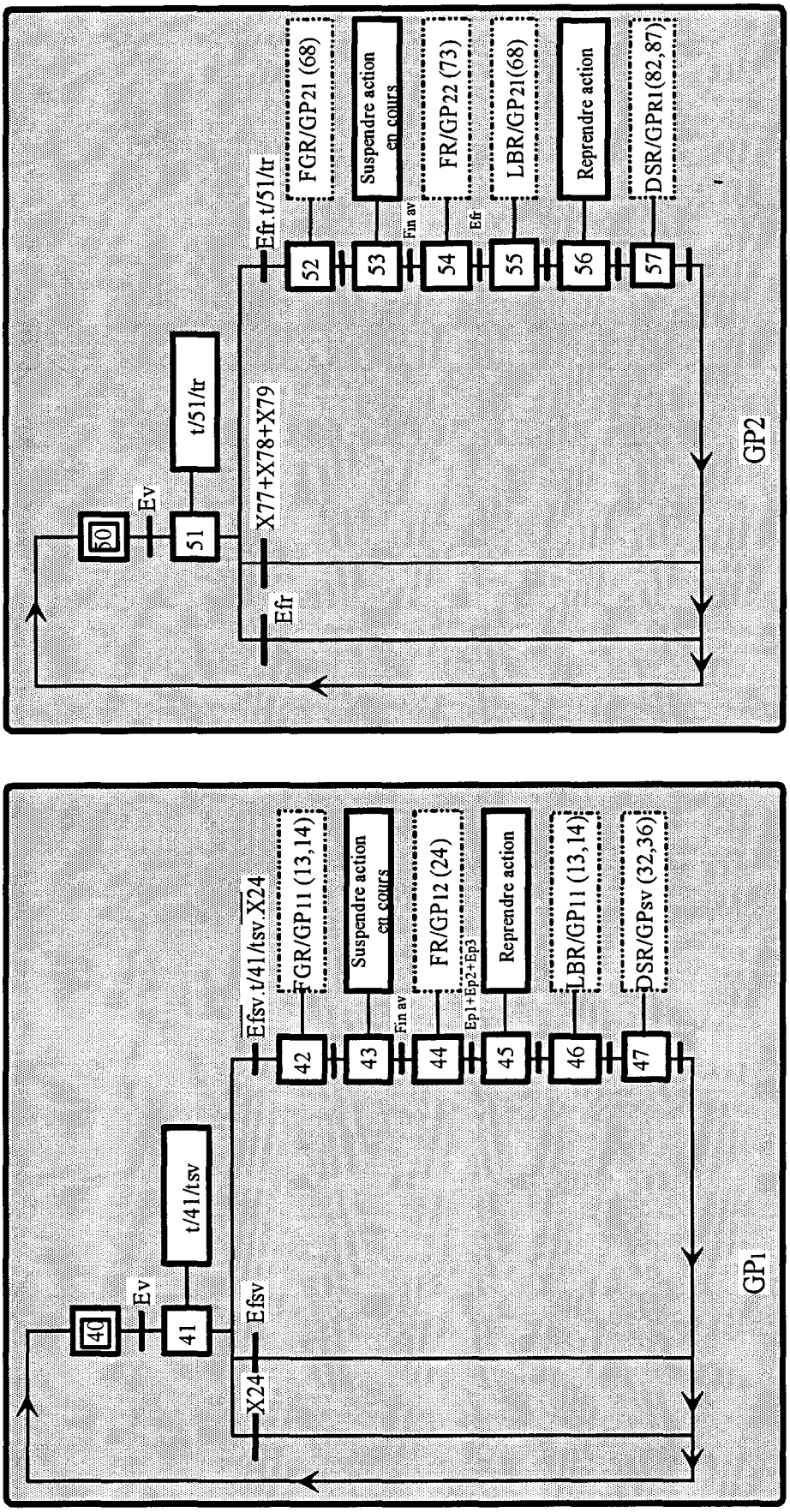

GP2
GP1
50
51
52
53
54
55
56
57
40
41
42
43
44
45
46
47
Ev
t/51/tr
X77+X78+X79
Efr
Efr.t/51/tr
FGR/GP21 (68)
Suspendre action en cours
Fin av
FR/GP22 (73)
Efr
LBR/GP21(68)
Reprendre action
DSR/GPR1(82,87)
t/41/tsv
X24
Efsv
Efsv .t/41/tsv.X24
FGR/GP11 (13,14)
Suspendre action en cours
Fin av
FR/GP12 (24)
Ep1+Ep2+Ep3
Reprendre action
LBR/GP11 (13,14)
DSR/GPsv (32,36)

figure 4.21

CONCLUSION.

Dans ce chapitre, nous avons exposé l'esprit de la méthodologie originale que nous avons adoptée pour la spécification des applications de commande en temps-réel et notamment en ce qui concerne les contraintes temporelles.

La méthodologie de spécification intègre:

- les outils de spécification de la synchronisation/communication exposés au chapitre I.
- la méthode conçue pour la structuration des applications exposée au chapitre II.
- une extension des outils précédents pour les événements acquittés occurrence par occurrence.
- et enfin la spécification des contraintes temporelles.

Nous avons proposé puis utilisé une formalisation de la notion de hiérarchie dans le Grafcet qui nous semble très prometteuse. Néanmoins, d'autres questions restent posées. En particulier y a t il une ou plusieurs hiérarchies dans un Grafcet global? Dans le cas où on tolère plusieurs hiérarchies, comment communiquent ces hiérarchies?

Nous avons aussi conçu un nouveau type de macroactions: le type ordre partiel et dressé un classement exhaustif des macroactions. Dans le but de préserver une cohérence globale sur les ordres en liaison avec la notion de hiérarchie et dans le contexte original du forçage, nous avons opté pour la conservation des règles de forçage pour la totalité des ordres avec quelquefois des nuances.

Dans le prolongement logique de ce chapitre, le chapitre suivant doit en particulier montrer les manières avec lesquelles seront interprétés les Grafcets accompagnés de ces développements. Les algorithmes d'interprétation doivent en tout point être conformes au modèle Grafcet. Par ailleurs, ces algorithmes vont servir à l'implantation de la commande.

CHAPITRE V:

INTERPRETATIONS

ET MISE EN OEUVRE

INTERPRETATIONS

ET

MISE EN OEUVRE

INTRODUCTION.

Le chapitre précédent a exposé la méthodologie de commande adoptée. Celle-ci utilise particulièrement les récents développements du Grafcet en matière de macroactions. Nous avons, en outre, développé de nouvelles macroactions et règles associées. Le présent chapitre se situe dans le prolongement du précédent dans le sens où:

- il expose, dans une première partie, le modèle d'interprétation que nous avons développé. Celui-ci interprète séparément chaque Grafcet partiel et introduit pour cela deux notions fondamentales: fin de cycle hiérarchique et cycle de traitement. Dans le contexte du modèle proposé pour la hiérarchie il met en exergue les informations que doivent s'échanger les divers Grafcets impliqués ainsi que les synchronisations nécessaires pour cela. En outre, il fait la différence entre l'interprétation du niveau hiérarchique NH_1 et celle des autres niveaux hiérarchiques.

- il met en évidence, dans un contexte d'implantation centralisée, la transcription de la commande en tâches du langage temps-réel ADA, les besoins auxiliaires ainsi que la gestion des diverses priorités. Il fixe ensuite la composition et la gestion de la partie interface entre la commande et le procédé.

- il enchaîne en exposant des carences du langage ADA, auxquelles il propose l'intégration d'un exécutif temps-réel.

I INTERPRETATIONS.

I-1 Etat de l'art.

Le modèle Grafcet original [ADE77] admet principalement deux interprétations possibles (c'est à dire deux sémantiques): l'interprétation sans recherche de stabilité et l'interprétation avec recherche de stabilité [AFC83] [ALL89]. L'introduction des règles de forçage a suscité des propositions d'interprétations intégrant celles-ci:

- l'interprétation séparée [LHO92], basée sur la séparation de l'interprétation en deux types:
  - interprétation des forçages: confiée au Grafcet global.
  - interprétation des étapes et transitions: confiée aux Grafcets partiels.
- l'interprétation synchrone [CHA92].

Nous proposons ci-dessous un algorithme d'interprétation qui intègre tous les ordres cités au chapitre précédent et qui interprète séparément chaque Grafcet partiel.

I-2 Interprétation par Grafcets partiels séparés.

Pour un Grafcet global, un seul événement peut se produire à la fois (une seule entrée change à la fois). Dans ce qui suit, nous entendons par événement un événement externe. Lorsqu'il s'agit d'un événement interne (changement de situation) ou d'un ordre interne, nous y ferons directement allusion.

Tous les Grafcets partiels d'un Grafcet global reçoivent les mêmes entrées (événements) et dans le même ordre. Le respect d'une part de la priorité conférée aux ordres (règles associées aux ordres) sur les règles d'évolution et d'autre part de la notion de hiérarchie, nécessite d'opérer un **décalage d'interprétation entre un niveau et son niveau inférieur**.

Pour illustrer cela, reprenons l'exemple exposé au chapitre précédent (figures 5.1 et 5.2) pour les situations atteintes représentées sur la première figure. La variable a passe à un. Si le traitement du Grafcet GP_5 débute au même instant que celui du Grafcet GP_1,

le Grafcet GP_5 atteint la situation {8} en passant d'abord par la situation {7,..}. Ce résultat est contraire à la règle 1 qui stipule qu'un Grafcet gouverné prend instantanément et directement la situation imposée.

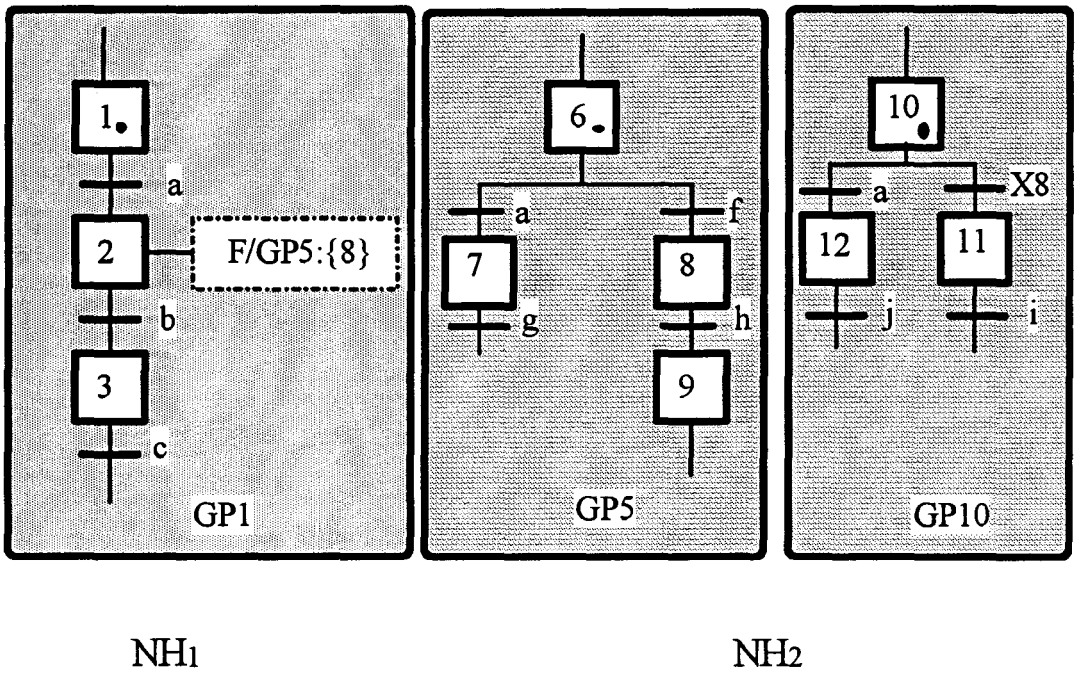


figure 5.1

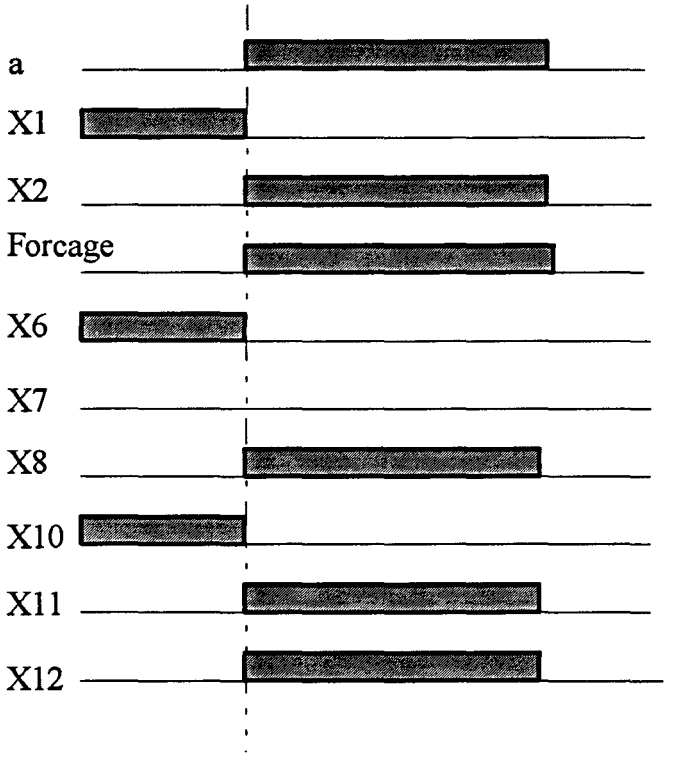


figure 5.2

Cette éventualité suggère que le traitement de GP_5 pour un événement, ne commence que lorsque le traitement (en particulier exécution des actions) du Grafcet GP_1 pour ce même événement est terminé .

En outre, lorsque un Grafcet partiel entame un traitement, il doit être au courant non seulement des situations des Grafcets partiels du même niveau qui prévalaient avant le début de ce traitement mais aussi des ordres émanant des Grafcets partiels du niveau supérieur (même ceux qui ne le gouvernent pas).

C'est le cas du Grafcet GP_{10} de l'exemple précédent. En effet, celui-ci doit être au courant de l'ordre de forçage à destination de GP_5. Faute de quoi, la situation atteinte par GP_{10}, lorsque la variable a passe à un, est {12, }, alors qu'elle devrait être {11,12, }.

Ces deux remarques imposent que **le traitement d'un Grafcet partiel d'un niveau donné pour un événement, ne peut débuter que si tous les Grafcets partiels du niveau supérieur ont achevés leur traitement pour ce même événement.**

Par ailleurs, un Grafcet partiel d'un niveau donné doit être au courant instantanément des situations des Grafcets partiels qu'il gouverne.

Outre le **traitement** que subit un Grafcet partiel lors de l'occurrence d'un **événement**, il doit également **évoluer en fonction des événements internes et des ordres internes**. Conformément à la priorité conférée aux ordres vis à vis des règles d'évolution, le modèle d'interprétation proposé est composé de deux phases: le traitement des ordres (étape 1 figure 5.3) suivi du traitement des événements externes et des événements internes (étape 2).

Ainsi, le modèle d'interprétation proposé interprète séparément chaque Grafcet partiel et repose sur deux notions fondamentales: **cycle de traitement** (correspond soit à l'une soit aux deux étapes) et **fin de cycle hiérarchique** que nous expliquons dans les paragraphes suivants.

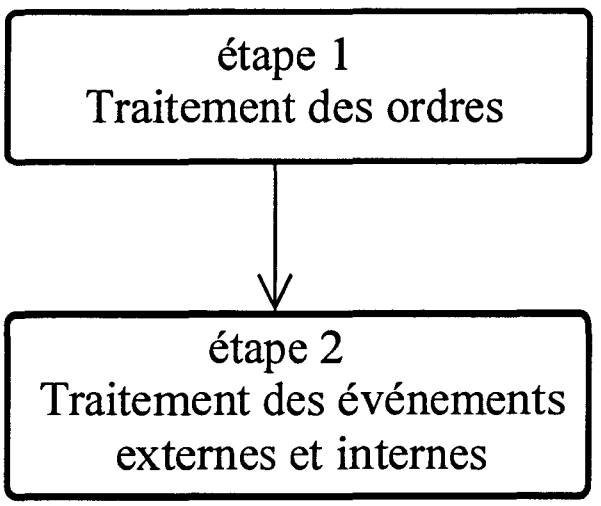


figure 5.3

L'interprétation est le siège d'une forte synchronisation dans les échanges d'informations (ordres, situations), entre Grafcets partiels et implique également la période de ces échanges.

I-3 Informations, synchronisation des échanges: fin de cycle hiérarchique.

La figure 5.4 illustre deux niveaux hiérarchiques représentant respectivement les Grafcets partiels GP_1, GP_2 et GP_3, GP_4. Le Grafcet GP_1 gouverne GP_3 et GP_2 gouverne GP_4.

La figure 5.5 résume les **diverses informations** que doivent se passer les Grafcets partiels de deux niveaux successifs. Ceci se produit en deux étapes:

1. chaque Grafcet partiel du niveau inférieur communique sa situation (**situation GP_3, situation GP_4**) au Grafcet partiel qui le gouverne ainsi qu'à tous les Grafcets partiels de son niveau.

2. chaque Grafcet partiel gouvernant communique à **tous les Grafcets partiels du niveau inférieur** ses éventuels ordres (**ordre GP_3, ordre GP_4**). Par ailleurs, il recueille les situations des Grafcets partiels qu'il gouverne (**situation GP_3, situation GP_4**).

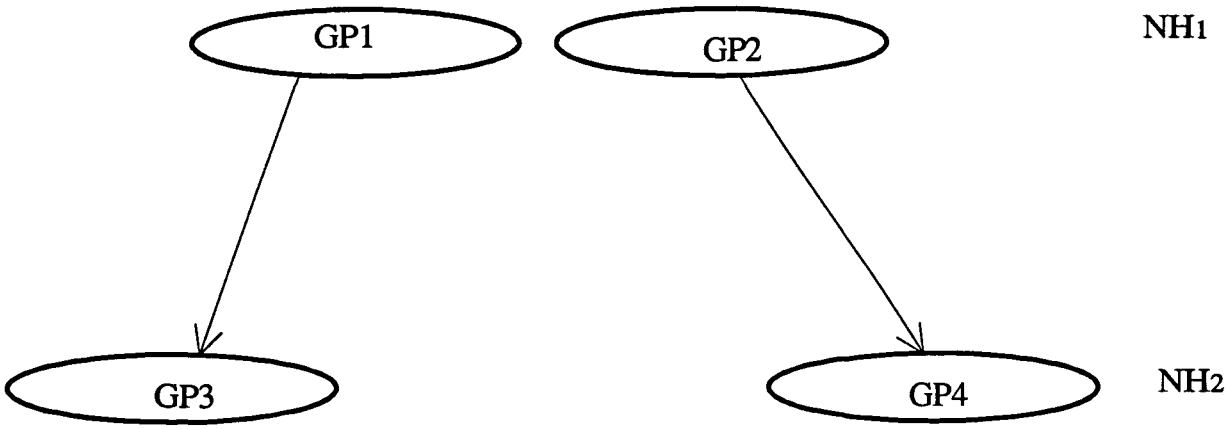


figure 5.4

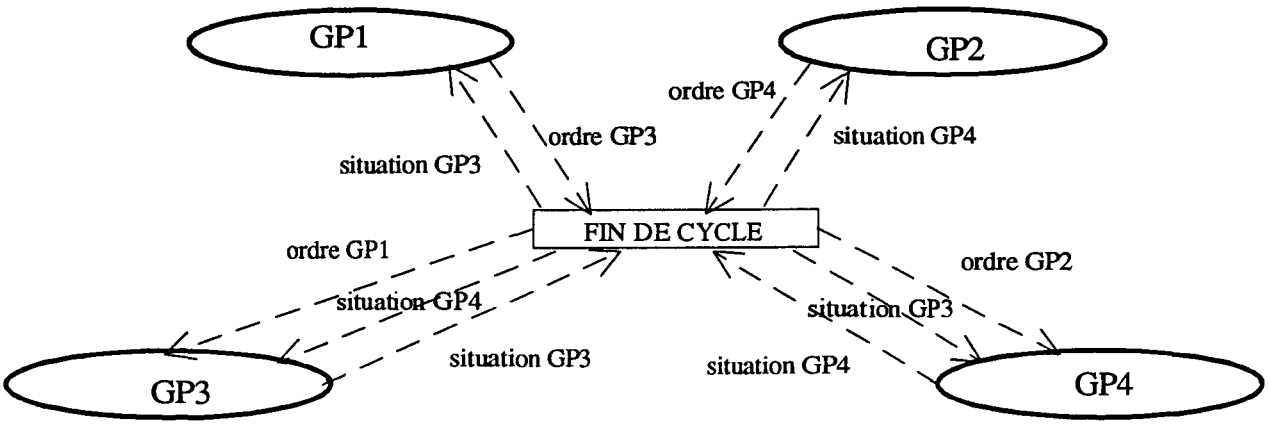


figure 5.5

Nous appelons **fin de cycle d'un niveau NH_i,** pour une image donnée (événement, situation), l'instant pour lequel:

- chaque Grafcet partiel a fini son cycle de traitement et peut entamer un cycle de traitement pour une nouvelle image.
- chaque Grafcet partiel du niveau NH_{i+1} peut entamer son cycle de traitement pour la première image.

Par rapport aux échanges d'informations, la fin du cycle d'un niveau NH_i suppose que:

- chaque Grafcet partiel du niveau NH_{i+1} a effectué la première étape.
- chaque Grafcet partiel du niveau NH_i a effectué la seconde étape.

Une hypothèse sous-jacente est faite. Celle-ci considère que **chaque Grafcet partiel de tout niveau, qui n'est pas sous l'effet d'un ordre, dispose du temps nécessaire pour le traitement d'un événement avant l'apparition d'un nouvel événement.**

I-4 Cycle de traitement (figure 5.3).

Il est nécessaire de fixer le **cycle de traitement** de chaque Grafcet partiel. Celui-ci correspond, pour chaque Grafcet partiel, à la période minimale au bout de laquelle il est possible d'avoir des échanges avec le niveau supérieur et/ou avec le niveau inférieur (fin de cycle hiérarchique).

Le cycle de traitement consiste alors à appliquer l'ordre reçu (étape 1) puis à interpréter le Grafcet pour l'événement externe, et/ou l'évolution interne qui se sont produits (étape 2).

Pour un Grafcet partiel gouverné, il est nécessaire que le traitement en étape 2 (figure 5.3) soit sans recherche de stabilité. En effet, dans le cas d'une interprétation avec recherche de stabilité, il est possible qu'un ordre soit émis vers un Grafcet partiel gouverné alors que celui-ci n'a pas encore terminé son cycle de traitement (état stable non atteint). Par conséquent, il ne peut considérer cet ordre dès son apparition, ce qui constitue une enfreinte à la règle 1.

En revanche, du fait qu'un Grafcet partiel du niveau NH_1 ne peut être gouverné, on peut utiliser indifféremment une interprétation avec ou sans recherche de stabilité.

En outre, pour un Grafcet partiel gouverné le cycle de traitement est différent selon qu'il y a eu ou non un ordre de reçu et pour le premier cas selon la nature de cet ordre.

I-4-1 Interprétation niveau NH_1.

Les Grafcets partiels de ce niveau ont la particularité de ne pas être gouvernés. Le cycle de traitement correspond uniquement à l'étape 2 de la figure 5.3. La figure 5.6 présente l'algorithme d'interprétation. C'est une interprétation classique sans recherche de stabilité. La fin de cycle de traitement correspond à l'instant où les actions sont exécutées. Comme l'interprétation est sans recherche de stabilité, l'algorithme lit l'éventuel événement en entrée et évolue une seule fois en tenant compte en même temps de celui-ci, des situations des Grafcets partiels de même niveau ainsi que des situations des Grafcets partiels gouvernés.

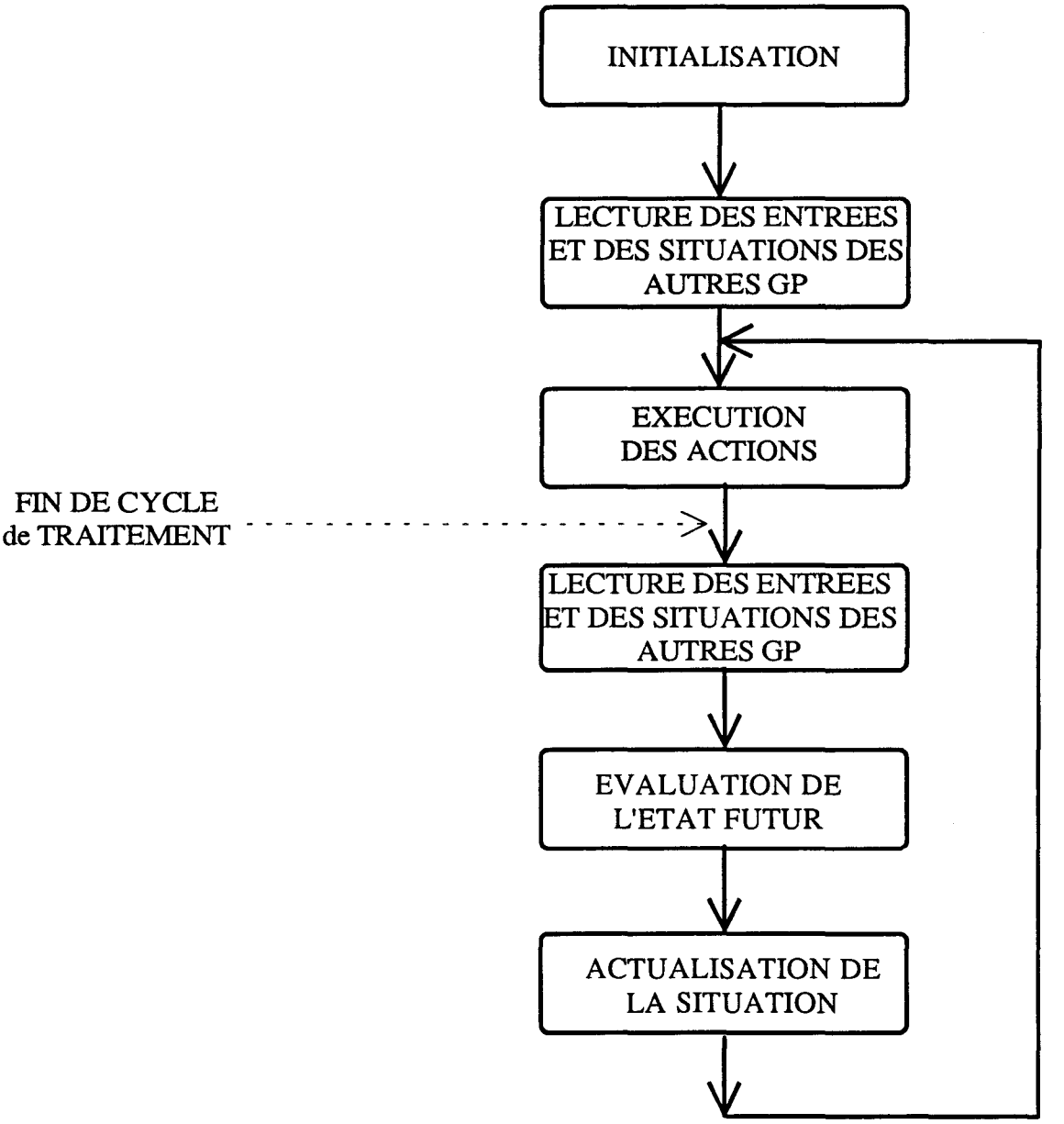


figure 5.6

I-4-2 Interprétations des autres niveaux hiérarchiques.

Pour ce type de niveaux, l'interprétation va dépendre de la nature de l'ordre reçu (c'est à dire selon les règles associées). En effet, sept cas se présentent conformément aux étapes de la figure 5.3:

1. si **aucun** ordre n'est reçu, on exécute directement l'étape 2.

2. dans le cas d'un ordre **total continu**, le cycle consiste à exécuter uniquement l'étape 1. En effet, ce type d'ordres n'autorise pas l'interprétation du Grafcet, ceci tant que son invalidation n'est pas reçue.

3. pour un ordre **total fugace instantané**, le cycle consiste à exécuter les deux étapes de la figure 5.3. L'étape 2 est un traitement usuel sans recherche de stabilité (**traitement normal**).

4. pour un ordre **total fugace prolongé**, le cycle se restreint à l'étape 1, ceci dans l'attente de l'ordre de libération.

5. pour un ordre **partiel continu**, les deux étapes sont intégrées dans le cycle. L'interprétation de l'étape 2 présente une particularité par rapport au cas précédent. En effet, il s'agit de conserver l'ordre reçu prioritairement aux règles d'évolution en cas de conflit (**traitement avec priorité**).

6. pour un ordre **partiel fugace instantané**, les étapes 1 et 2 sont exécutées successivement. L'étape 2 est un traitement classique sans recherche de stabilité (**traitement normal**).

7. enfin, pour un ordre **partiel fugace prolongé**, l'étape 1 est accomplie, suivie de l'étape 2 avec la particularité de la priorité de l'ordre sur les règles d'évolution (**traitement avec priorité**).

Par ailleurs, l'**exécution des actions** a lieu uniquement à la fin du cycle de traitement. Ainsi, pour un traitement qui n'intègre que l'étape 1, l'exécution des actions (c'est à dire des ordres sauf au niveau NH_3) s'effectue à la **fin de cette étape**. Par contre, pour un cycle qui intègre les deux étapes, les actions sont exécutées uniquement à la **fin de l'étape 2**.

Le début du cycle de traitement d'un Grafcet partiel d'un niveau NH_i (i différent de 1) est conditionné par la fin de cycle du niveau NH_{i-1}. En revanche, la fin de ce même cycle conditionne la fin de cycle du niveau NH_i.

I-4-2-1 ALGORITHME GLOBAL.

La figure 5.7 résume la totalité de l'algorithme intégrant les divers ordres. Pour entamer un cycle de traitement, le Grafcet doit être informé des éventuels ordres en provenance du niveau supérieur, de la situation de chaque Grafcet partiel du même niveau et des situations des Grafcets partiels qu'il gouverne.

Si **aucun ordre** en direction de ce Grafcet n'est parvenu, on interprète le Grafcet pour l'événement externe éventuel qui aurait déclenché le cycle ainsi que pour les situations des Grafcets de même niveau et des Grafcets que celui-ci gouverne (**traitement normal**). Dans le cas contraire, il faut **chercher la nature de l'ordre** reçu d'abord total ou partiel ensuite continu ou fugace et pour ce dernier cas instantané ou prolongé.

Considérons un premier exemple: si un ordre **total continu** est reçu, l'ordre est appliqué suivi de l'exécution des actions des étapes nouvellement actives. Par la suite le Grafcet ne peut évoluer. Il faut attendre l'invalidation de l'ordre (nécessairement par **la fin d'un cycle du niveau supérieur**). Lorsque **l'ordre est invalidé**, il faut interpréter le Grafcet (**traitement normal**). A la fin de ceci, on boucle sur l'**attente de fin de cycle du niveau supérieur**.

Un second exemple concerne un **ordre partiel instantané**. Lorsque celui-ci est reçu, il faut **appliquer** cet ordre (cas de *activer*) puis interpréter le Grafcet (**traitement normal**). Lorsque ceci est terminé, l'algorithme boucle sur la phase d'**attente de la fin de cycle du niveau supérieur**.

A chaque fin de cycle de niveau hiérarchique, les Grafcets partiels d'un niveau NH_i transmettent les informations (ordres) aux Grafcets partiels du niveau NH_{i+1} et reçoivent de ceux-ci leurs situations.

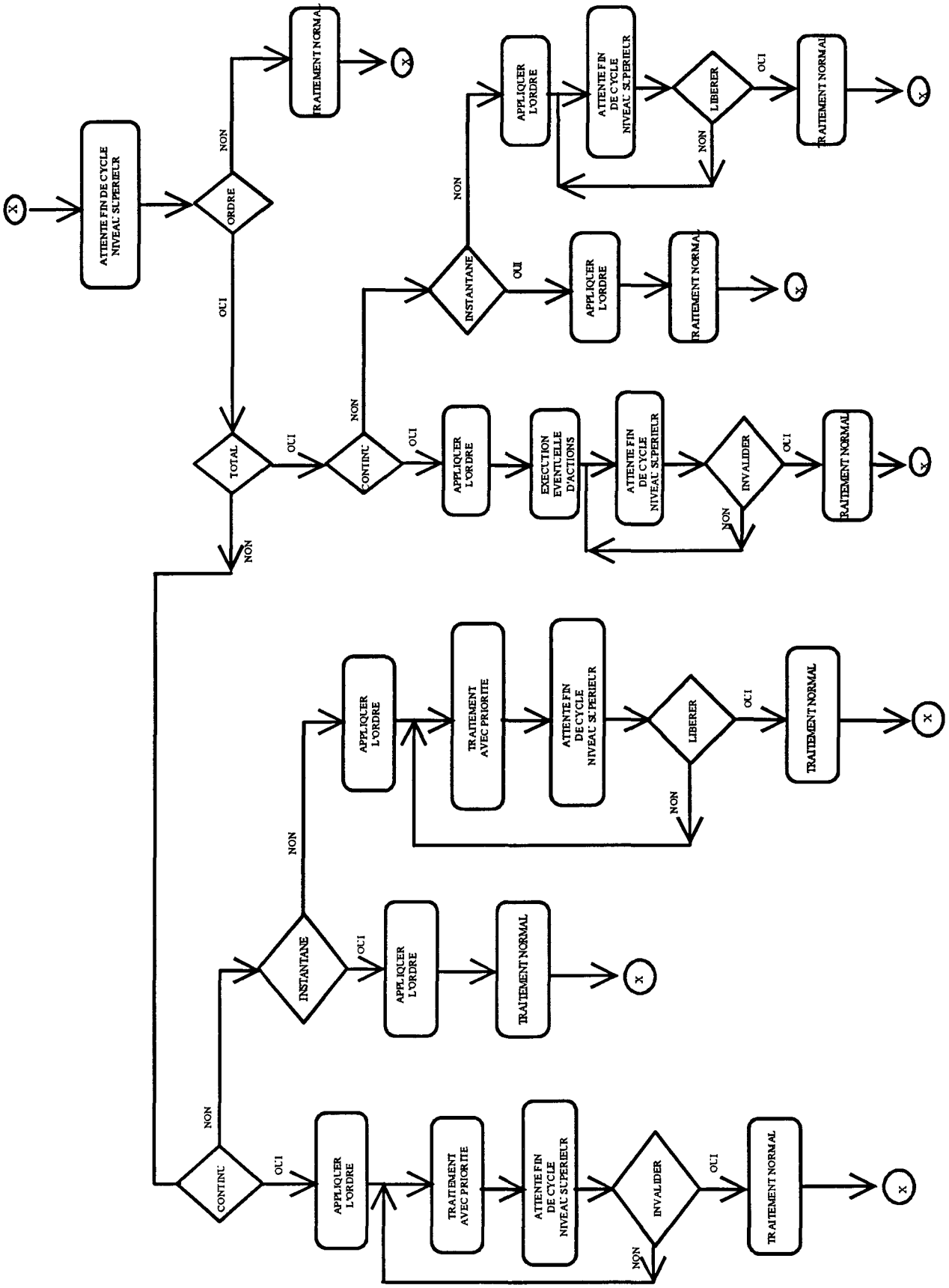

ATTENTE FIN DE CYCLE NIVEAU SUPERIEUR
ORDRE
NON
OUI
TRAITEMENT NORMAL
INSTANTANE
APPLIQUER L'ORDRE
ATTENTE FIN DE CYCLE NIVEAU SUPERIEUR
LIBERER
TRAITEMENT NORMAL
TOTAL
CONTINU
EXECTION EVENTUELLE D'ACTIONS
INVALIDER
TRAITEMENT AVEC PRIORITE

figure 5.7

I-4-2-2 ALGORITHMES DE TRAITEMENT.

Deux algorithmes de cycle normal existent selon l'existence et la nature de l'ordre:

- **traitement avec priorité**, c'est le cas lorsqu'un ordre partiel continu ou fugace prolongé est reçu. Il s'agit alors d'interpréter le Grafcet et de garder l'ordre présent et prioritaire sur les règles d'évolution, jusqu'à la réception, soit de l'invalidation (ordre partiel continu), soit de l'ordre *libérer* (cas de *figer { }*). L'algorithme (sans recherche de stabilité) est représenté en figure 5.8, sa fin correspond à la fin du cycle de traitement.

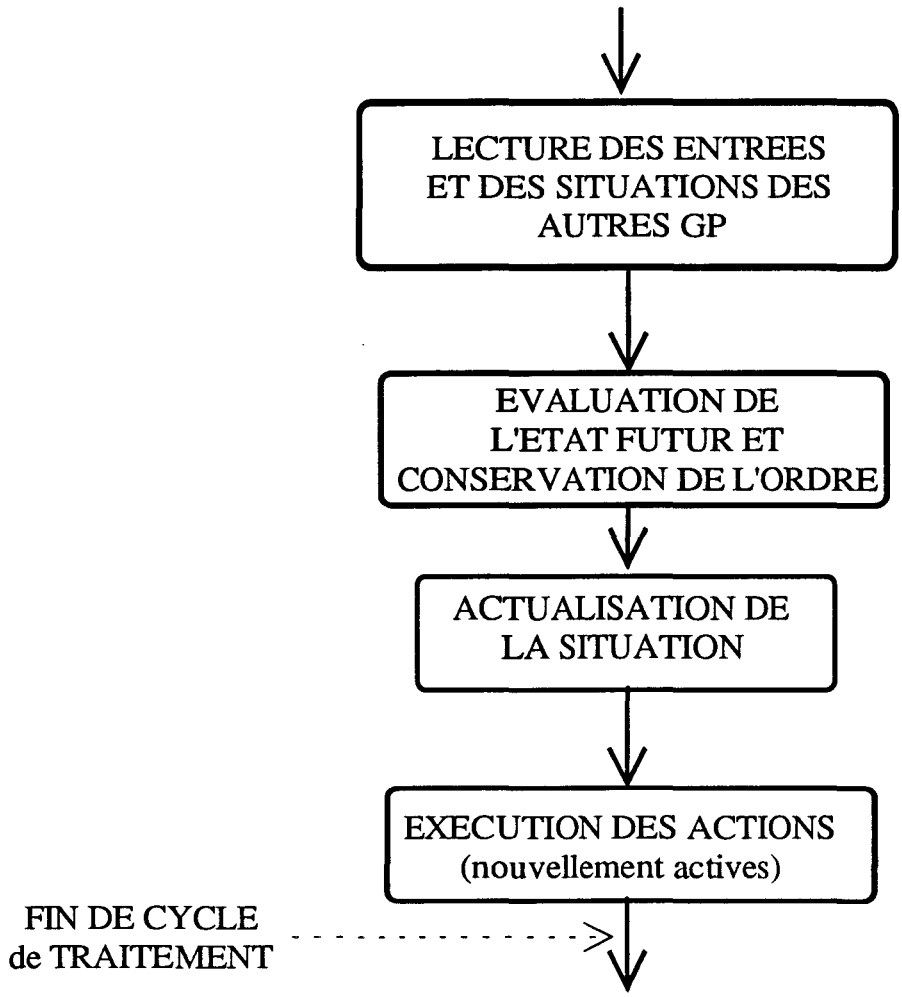


figure 5.8

- **traitement normal**. Il s'agit d'interpréter le Grafcet pour l'événement externe et/ou l'évolution interne et/ou l'ordre (cas de *libérer* et *libérer { }*) qui sont reçus. L'algorithme est représenté en figure 5.9. La fin de cet algorithme correspond également à la fin du cycle de traitement.

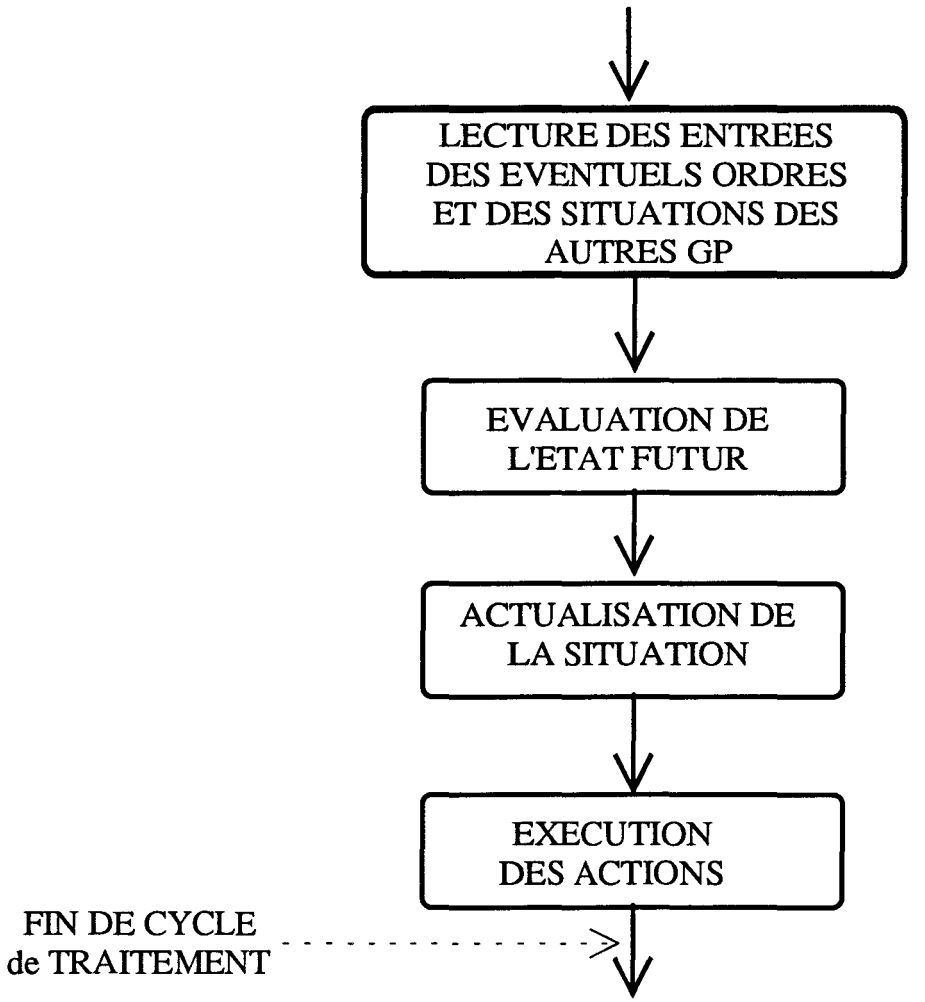


figure 5.9

II MISE EN OEUVRE.

Le Grafcet est un outil de spécification souvent utilisé pour la description des commandes locales. Plus généralement, c'est un outil de spécification du comportement attendu des systèmes de production automatisés. En outre, le Grafcet reste valable pour toutes les phases du cycle de vie du produit.

L'implantation du Grafcet peut s'opérer de deux façons:

- soit par **algorithme d'implantation**: on peut toujours, à partir de certains choix de codage des entités du Grafcet, mettre en place des algorithmes de programmation de l'outil de modélisation Grafcet pour un système informatique défini.

- soit par **implantation sur machine dédiée:** c'est la cas le plus fréquent. Le support matériel est l'automate programmable industriel (API). Celui-ci est programmable avec des langages inspirés du Grafcet (LIG).

Dans le cadre de cette étude et ainsi que nous l'avons expliqué au chapitre I, le Grafcet est utilisé pour la description du niveau commande globale dans les cellules flexibles d'assemblage. C'est pourquoi, dans un contexte d'implantation centralisée, la commande est nécessairement programmée sur un ordinateur (type IBM pc).

Cette programmation est alors effectuée dans un langage du type évolué ou du type assembleur. Nous proposons l'utilisation d'un **langage évolué temps-réel ADA** [DOD79]. L'utilisation du langage ADA pour la commande d'ateliers de production n'est pas nouvelle en soi [VOL84][VOL87][SAL88]. L'utilisation du langage ADA dans le cadre de cette étude, ainsi que nous le verrons plus loin, va au delà de ces préoccupations. Elle permet entre autres de procurer à l'utilisateur, des moyens naturels d'écriture de tâches ainsi que des mécanismes de synchronisation de haut niveau.

Par ailleurs, il est souvent nécessaire d'intégrer un **exécutif temps-réel** du fait:

- des **contraintes temporelles supplémentaires** apparaissant lors de la mise en oeuvre de l'application sur un processeur unique.

- de certaines **limites du langage ADA** dans le gestion des tâches et du temps.

II-1 Implantation centralisée.

La structure du système est décomposable de façon traditionnelle en une partie commande et une partie opérative (procédé) (figure 5.10). La partie commande peut, elle-même, se décomposer en trois parties:

- le séquencement de la commande, décrit en Grafcet.
- l'interface entre le séquencement et le procédé. Celle-ci décrit la correspondance entre les événements d'entrée et les actions du Grafcet d'une part, et les procédures informatiques d'entrée-sortie, par interfaces programmables par exemple, d'autre part.
- enfin l'interface environnement-système (utilisateur ou niveau supérieur de commande) constituée par exemple par une animation graphique renseignant l'utilisateur sur l'état de son système et recueillant des demandes en provenance de cet utilisateur.

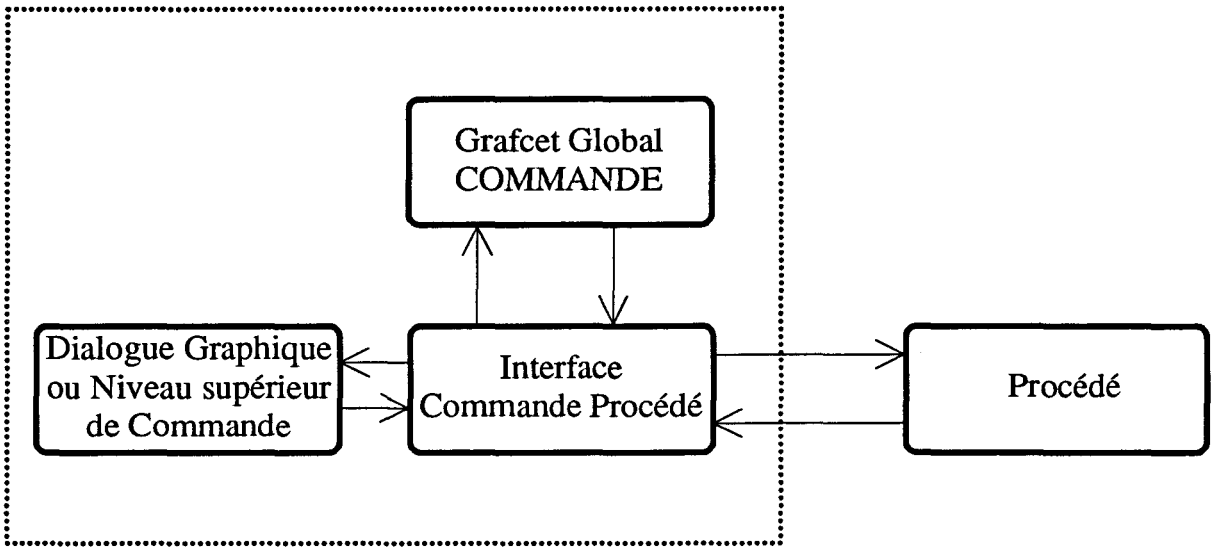


figure 5.10

Par ailleurs, le niveau interface peut induire des **contraintes supplémentaires** dues:

- aux **différences des vitesses de transmission** des diverses parties du procédé.
- aux **fréquences d'occurrences des événements** qui proviennent du procédé.
- éventuellement au fait que c'est le **même processeur** qui exécute en même temps la commande, l'interface et peut être même le dialogue graphique aussi.

II-2 Implantation du niveau commande sous langage ADA.

Dans le cadre de cette étude, les trois parties de l'application sont implantées avec le langage ADA. Le présent paragraphe montre la transcription des Grafcets partiels du niveau commande en tâches ADA ainsi que les tâches auxiliaires nécessaires.

II-2-1 Transcription des Grafcets partiels en tâches ADA.

Chaque Grafcet partiel, d'abord interprété algébriquement en équations de récurrence [DEF86] [ZAH80], est transcrit sous forme d'une tâche ADA (T0, T1 en figure 5.11). Cette implantation, dite **algorithme d'implantation,** par ailleurs conforme aux interprétations énoncées au premier paragraphe, diffère suivant le niveau hiérarchique considéré:

- aux niveaux NH_2 et NH_3 (niveaux bas), une tâche procède dans l'ordre suivant:

  1. communique sa situation au(x) Grafcet(s) partiel(s) gouverneurs(s) ainsi qu'aux Grafcets partiels de même niveau hiérarchique.
  2. recueille la situation des Grafcets partiels du même niveau ainsi que l'ordre émanant du niveau supérieur.
  3. effectue un cycle de traitement.
  4. communique les ordres aux Grafcets partiels gouvernés et recueille leurs situations.
  5. retour à l'étape 1.

– au niveau NH_1, une tâche effectue les étapes suivantes, ceci après une étape d'initialisation:

1. communique sa situation aux Grafcets partiels de son niveau hiérarchique et recueille leurs situations.
2. effectue un cycle de traitement.
3. communique au niveau inférieur les ordres émis et reçoit les situations des Grafcets partiels gouvernés.
4. retour à l'étape 1.

II-2-2 Centralisation des informations entre niveaux consécutifs.

Le respect de l'interprétation algorithmique énoncée à la première partie de ce chapitre et des algorithmes d'implantation cités précédemment, impose de centraliser les informations (situations et ordres), entre un niveau et le suivant .

Une tâche dite **centralisatrice** est chargée de réaliser cette fonction. Dans le cas de notre commande (3 niveaux hiérarchiques), deux tâches centralisatrices s'imposent: la tâche $\mathbf{Tcen_{12}}$ centralisant les informations entre les niveaux $\mathbf{NH_1}$ **et** $\mathbf{NH_2}$, et la tâche $\mathbf{Tcen_{23}}$ pour les niveaux $\mathbf{NH_2}$ **et** $\mathbf{NH_3}$ (figure 5.11).

L'activité d'une tâche $\mathbf{Tcen_{i\,i+1}}$ se déroule suivant le schéma suivant:

1. recevoir la situation de chaque Grafcet partiel du niveau NH_{i+1}.
2. recevoir les ordres en provenance du niveau NH_i.
3. transmettre les ordres vers le niveau NH_{i+1}.
4. transmettre la situation de chaque Grafcet partiel gouverné du niveau NH_{i+1} au(x) Grafcet(s) partiel(s) gouverneur(s) du niveau NH_i.
5. diffuser la situation de chaque Grafcet partiel du niveau NH_{i+1} aux Grafcets partiels du même niveau.
6. retour à l'étape 1.

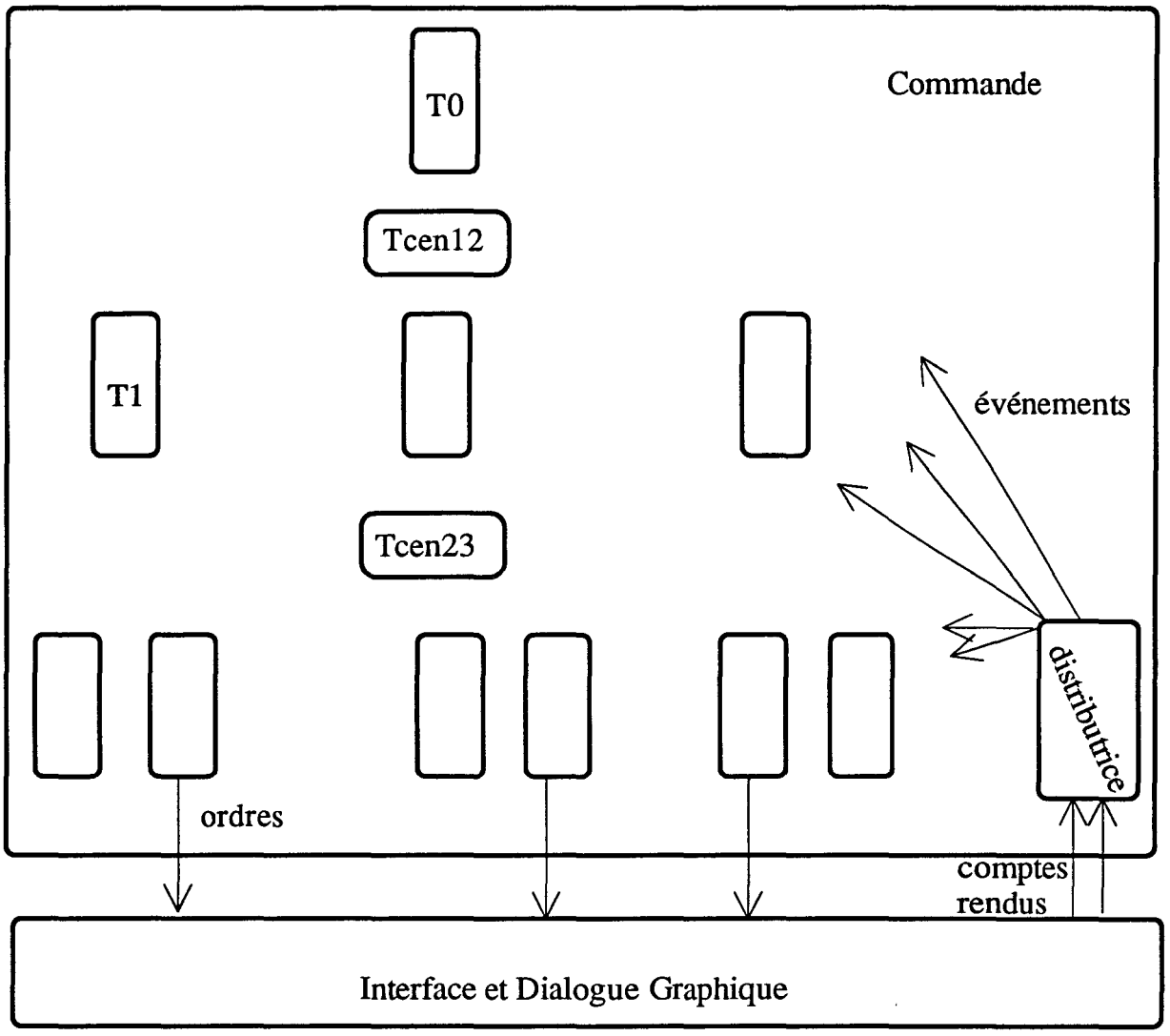


figure 5.11

I-2-3 Tâche distributrice d'événements.

Les événements en provenance de l'environnement ou du procédé doivent être, selon le modèle Grafcet, transmis dans l'ordre et dès leur apparition à tous les Grafcets partiels de tous les niveaux (figure 5.11).

L'algorithme d'interprétation conçu, opère un décalage d'interprétation entre chaque niveau et son suivant. Ainsi un Grafcet partiel du niveau hiérarchique NH_2 ou NH_3 ne consulte un événement que lorsqu'il est effectivement capable de s'interpréter pour cet événement (ce Grafcet n'est pas sous l'effet d'un ordre total continu ou total fugace prolongé et l'événement est encore vivant).

Une **tâche** dite **distributrice** recueille les événements en provenance des parties interface et dialogue graphique et en informe chaque Grafcet partiel à la demande de celui-ci. Elle doit également acquitter tout événement qui n'a pas encore été transmis à la totalité des Grafcets partiels, si un acquittement le concernant est reçu.

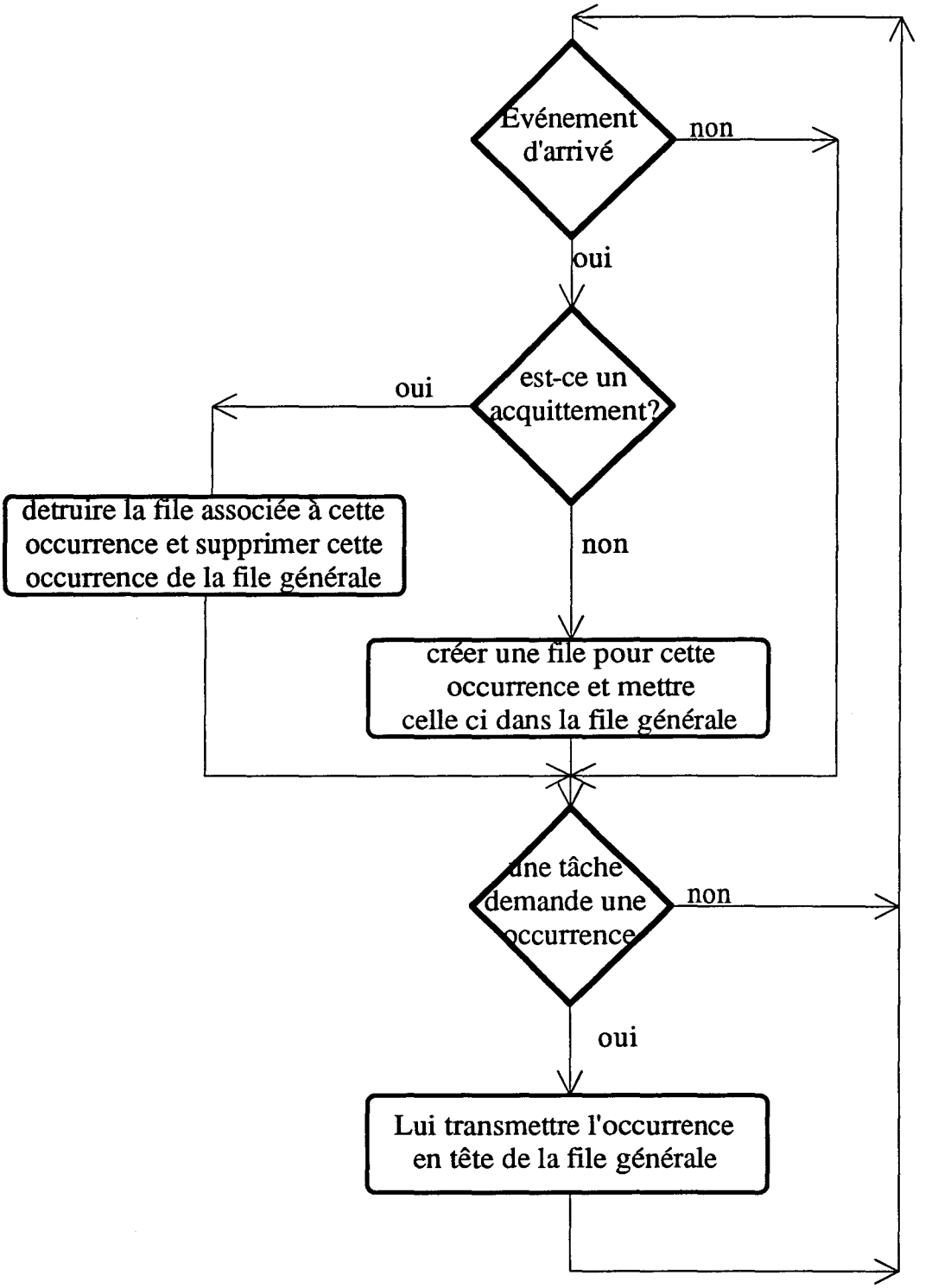


figure 5.12

L'algorithme de la figure 5.12 résume le fonctionnement de cette tâche. Deux types de files sont créées: une file générale unique contient toutes les occurrences d'événements reçues (gérée en FI FO), et une file associée à chaque occurrence qui se produit. Cette tâche connaît toutes les tâches (Grafcets partiels) du niveau commande et garde en mémoire les tâches qui n'ont pas encore consommé cette occurrence. Par ailleurs, les diverses interactions entre les tâches citées (tâches de commande, centralisatrices et distributrice, ont lieu par le mécanisme de haut niveau qu'est le **Rendez-vous**.

II-3 Gestion des tâches du niveau commande.

Le paragraphe précédent a mis en évidence un éventail de tâches pour l'implantation de la commande. Une question importante se pose: quelles priorités accorder aux diverses tâches d'un même niveau hiérarchique et quelles priorités relatives accorder à ces divers niveaux et aux tâches auxiliaires?

La même priorité est attribuée à toutes les tâches représentant des Grafcets partiels d'un même niveau hiérarchique, ceci conformément à l'algorithme d'interprétation élaborée et du fait que :

- l'on exécute un seul cycle de traitement pour chaque Grafcet partiel (tâche).
- et que tous les Grafcets partiels d'un même niveau sont interprétés simultanément.

Ces priorités égales se manifestent donc par un traitement en **temps partagé** des diverses tâches représentant les **Grafcets partiels d'un même niveau**. La valeur du temps de traitement doit être inférieure au plus petit temps de cycle afin d'assurer la simultanéité dans les actions et les évolutions.

Par ailleurs, afin de respecter les règles associées aux ordres internes, il est nécessaire d'opérer un **changement de priorité des niveaux** en fonction de la fin de cycle hiérarchique. En effet, lorsque se produit un événement externe, le niveau NH_1 effectue un cycle de traitement pour chacun de ses Grafcets partiels.

A la fin de cycle du niveau NH_1, les Grafcets partiels de ce niveau repartent chacun pour un nouveau cycle de traitement, mais cette fois-ci sans événement externe (de par l'hypothèse sous-jacente faite dans le premier paragraphe)

Ainsi, pour assurer la règle 1 (simultanéité), il est alors nécessaire de donner la plus haute priorité au niveau qui vient d'entamer son cycle pour cet événement c'est à dire au niveau NH_2 et ainsi de suite.

En ADA les priorités sont statiques et rattachées à une tâche et non pas à un message d'entrée. Il est toutefois possible de disposer d'une tâche possédant la plus haute priorité pour ordonnancer dynamiquement les autres.

Ce rôle est joué par les tâches centralisatrices d'informations. En effet, celles-ci sont au courant des fins de cycle hiérarchique. A l'initialisation, les tâches centralisatrices accordent la plus haute priorité des divers niveaux, au niveau NH_1. Lorsque la fin de cycle de ce même niveau est atteinte, la tâche $Tcen_{12}$ attribue au niveau NH_2 la plus haute priorité et ainsi de suite.

Lorsque toutes les tâches du niveau NH_3 ont fini leurs cycles de traitement, la priorité doit revenir au niveau NH_1. Comme il n'existe pas de tâche centralisatrice entre ces deux niveaux, c'est la tâche $Tcen_{23}$ qui doit, à la fin de sa première étape et une fois sur deux, opérer ce changement de priorité (entrelacé avec le changement de priorité des niveaux NH_2 et NH_3). Les tâches centralisatrices ont par conséquent des priorités plus hautes que celles des divers niveaux hiérarchiques. La tâche distributrice, en raison des événements qu'elle manipule, est dotée de la plus haute priorité dans le niveau commande.

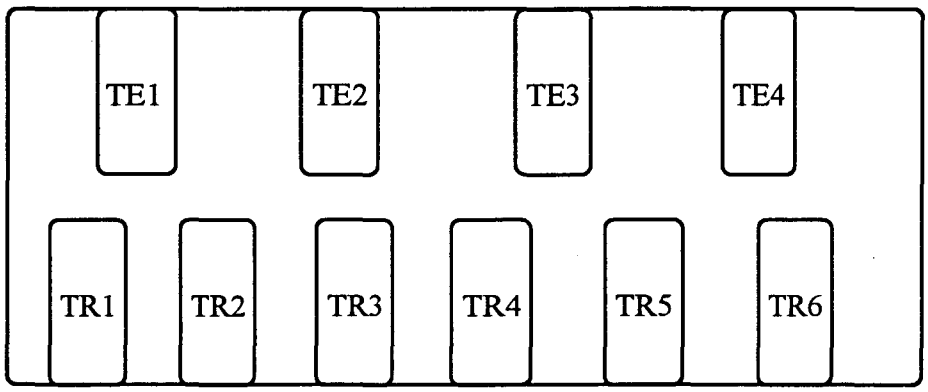
II-4 Implantation et gestion des tâches au niveau interface.

Les tâches de commande du niveau hiérarchique NH_3 émettent des ordres (demandes d'exécution) vers l'interface. Par ailleurs, le procédé et la partie dialogue interagissent avec l'interface par des événements et/ou comptes rendus.

La partie interface est composée de deux types de tâches: les tâches émettrices et les tâches réceptrices (figure 5.13).

Les **tâches émettrices** doivent, en fonction des ordres reçus de la partie commande, générer les commandes nécessaires en direction du procédé et envoyer les comptes rendus en direction de la partie dialogue graphique.

Les **tâches réceptrices** doivent être toujours à l'écoute de tout changement en provenance du procédé et du dialogue graphique. Lorsqu'une tâche réceptrice reçoit un événement, elle en informe la tâche distributrice du niveau commande.



NIVEAU INTERFACE

figure 5.13

Les **tâches d'émission** sont des tâches différées qui sont déclenchées sur demande du niveau commande.

En revanche, une **tâche de réception** est:

- soit une tâche immédiate (associée à une interruption si cela est possible).
- soit une tâche périodique (à activer toutes les T secondes correspondant à la fréquence maximale d'occurrence de l'événement correspondant).

Les tâches de réception sont les tâches les plus prioritaires de tout le niveau interface. En effet, c'est à leur niveau qu'intervient d'une part le délai entre le **temps**

d'occurrence et le **temps de traitement** d'un événement et d'autre part la possibilité de **perte d'informations (débit).**

Les tâches d'émission ont la possibilité de recevoir un ordre d'urgence alors qu'elles sont déjà en train de générer les commandes nécessaires pour un ordre précédent. Elles ne peuvent donc **opérer un avortement des commandes générées pour l'ordre précédent et générer les commandes pour l'ordre d'urgence.**

Ces tâches ne peuvent être vues comme des tâches ADA qui exécutent à elles seules les opérations nécessaires pour l'émission des ordres (impossibilité de préemption).

Un solution consiste à **associer à chaque tâche d'émission, des tâches auxiliaires dont le rôle est de générer les commandes pour l'émission des ordres.** La tâche d'émission a alors pour fonction de contrôler l'exécution de ces tâches auxiliaires.

Par ailleurs, le langage ADA présente un inconvénient majeur dans la manipulation du temps [CAR90]. En effet, la précision de cette manipulation dépend de la réalisation du paquetage logiciel "calendar" ainsi que des intervalles d'appel à l'ordonnanceur. Par conséquent, le **délai** (instruction delay (t)) est un **délai minimum** et ne peut servir que de délai de garde. Ce n'est pas un délai maximum associable à une échéance stricte. Cet inconvénient se répercute aussi bien sur le niveau interface que sur le niveau commande.

Pour parer aux deux carences précédentes, nous sommes amenés à intégrer un **exécutif temps-réel** ainsi que nous l'expliquons dans le paragraphe suivant.

II-5 Intégration d'un exécutif temps-réel.

Un **exécutif temps-réel** est un ensemble de programmes sur lesquels s'appuie une application, et dont le but est de gérer la machine physique (c'est à dire gérer l'ensemble des ressources dont peut avoir besoin une application). Il doit en particulier cacher les propriétés physiques des machines utilisées pour ne laisser voir que les propriétés logiques.

Une partie des activités de l'exécutif, notamment celles concernant la programmation séquentielle, est prise en charge par le langage évolué avec lequel est écrite

l'application. Les parties de l'exécutif plus proches du matériel constituent le **noyau**. Les autres parties de l'exécutif, ou **exécutif de requêtes**, traitent les commandes de l'usager et fournissent l'interface pour les requêtes d'accès aux périphériques, la gestion mémoire et la gestion des processus (synchronisation, ordonnancement).

Le processeur idéal doit comporter un mécanisme d'interruption autorisant plusieurs niveaux de priorités, un changement de contexte, associé à une unité de gestion mémoire (UGM).

Si l'UGM possède plusieurs niveaux de privilèges, l'exécutif temps-réel est exécuté au niveau le plus privilégié. Les entrées/sorties et les programmes assurant la sécurité du système au niveau suivant, et les programmes d'application au niveau le moins privilégié [TSC90]. Le choix du processeur dépend des contraintes de temps, du nombre de tâches à piloter et de la sécurité qu'il faut garantir.

Pour un exécutif temps-réel, le logiciel peut être fourni sous la forme d'une mémoire morte à placer sur la carte processeur, ou sous la forme d'une bibliothèque de fonctions au format utilisé par la machine de développement. L'utilisation de ces noyaux se fait à l'aide de primitives appelées à travers des interruptions logicielles.

Pour des raisons de stratégie ou de configuration matérielle, les exécutifs varient d'une application à l'autre. Afin de faciliter la réalisation des exécutifs, plusieurs standards ont été proposés, citons entre autres Mascot (standard britannique), Tc8 (Ewics) et Sceptre [BNI82].

Ce dernier standard distingue trois niveaux de construction:

- agence: unité fonctionnelle de construction d'un exécutif temps-réel. On peut trouver des agences de gestion mémoire, d'horlogerie etc..
- le noyau intégrant les opérations élémentaires.
- l'ordonnanceur qui attribue le processeur à une tâche.

Les langages temps-réel sont équipés de leurs propres exécutifs temps-réel. Mais ainsi que nous l'avons vu précédemment pour ADA, celui-ci présente des insuffisances. L'intégration d'un exécutif temps-réel qui s'interface avec ADA peut se faire selon le schéma bloc décrit en figure 5.14.

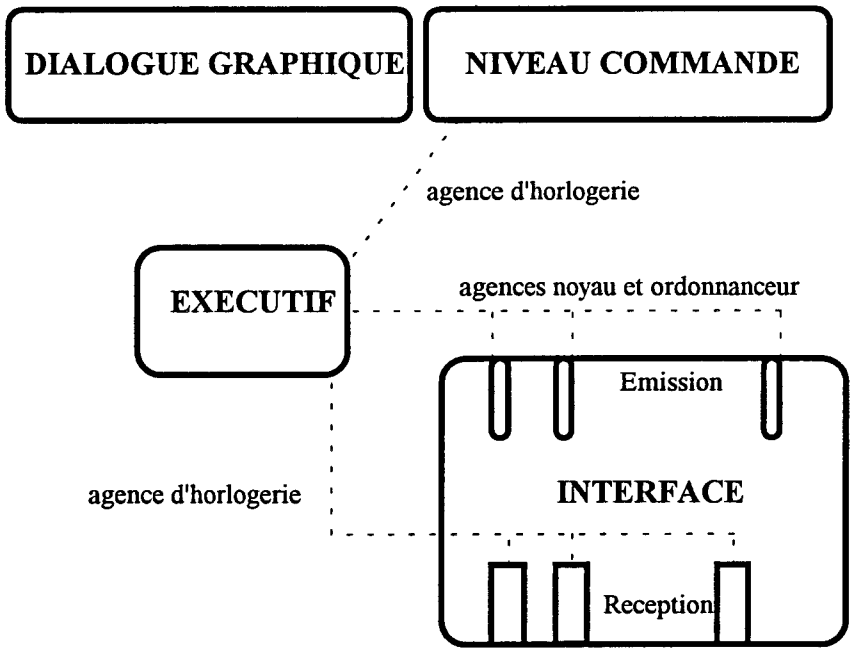


figure 5.14

Les tâches du niveau commande ainsi que celles du niveau interface sont des tâches ADA gérées par l'exécutif de ce dernier.

- Les **tâches de niveau commande** font appel à l'**agence d'horlogerie temps-réel** de l'exécutif pour manipuler le temps (particulièrement celles du niveau NH_2: Surveillance Temporelle, Décision et Gestion d'événements).

- Les **tâches d'émission** du niveau interface font appel à l'exécutif pour la **création, destruction, gestion et ordonnancement des tâches auxiliaires** de génération des commandes nécessaires.

  Ainsi une tâche d'émission ayant lancé une demande pour l'émission d'un ordre au procédé, est capable de traiter un arrêt d'urgence. En effet, dans ce cas, il suffit de suspendre la tâche auxiliaire de génération des commandes qui est en cours, si celle-ci n'est pas terminée, puis de lancer une tâche auxiliaire appropriée à l'arrêt d'urgence.

- Les **tâches de réception** du niveau interface font appel au service de l'**agence d'horlogerie temps-réel** de l'exécutif pour la manipulation du temps.

La partie dialogue graphique est constituée essentiellement d'une tâche ADA. Celle-ci a pour rôle de communiquer avec l'utilisateur par des événements et comptes rendus. Elle doit également communiquer avec le Grafcet Global, via l'interface, par événements et comptes rendus. Cette tâche, gérée par l'exécutif d'ADA, utilise par ailleurs des empaquetages logiciels pour l'affichage graphique.

L'exécutif est la partie qui possède le niveau de priorité le plus haut parmi l'ensemble: exécutif, niveau interface, niveau commande, dialogue graphique. Le niveau interface admet le niveau de priorité immédiatement inférieur.(la partie réception est prioritaire par rapport à la partie émission). Le niveau commande vient ensuite au niveau de priorité inférieur. En raison de la faible occurrence des arrêts d'urgence et du faible débit d'échanges avec la partie graphique, celle-ci admet le niveau de priorité le plus bas.

CONCLUSION.

La première partie de ce chapitre a développé les étapes du modèle d'interprétation intégrant l'éventail des ordres et règles associées exposés au chapitre précédent. Il s'agit d'interpréter, en prévision d'une implantation répartie, chaque Grafcet partiel séparément. Tout en gardant les mêmes notions fondamentales, à savoir cycle de traitement et fin de cycle hiérarchique, on peut interpréter ensemble tous les Grafcets partiels d'un même niveau hiérarchique: c'est l'**interprétation par niveaux**.

Pour une implantation répartie, le modèle d'interprétation développé nécessite des réseaux très rapide. Faute de quoi le temps de cycle de traitement serait inférieur au temps de transfert des informations. Le réseau retarderait alors l'interprétation.

La deuxième partie de ce chapitre a montré, pour un contexte centralisé et pour la méthodologie de commande adoptée, l'implantation d'une application.

Celle-ci se décompose en trois bloc: la commande, l'interface et l'interface graphique:

- la transcription des Grafcets partiels de la commande en tâches ADA, institue une traduction naturelle et automatique de la commande. Le respect, dans la mise en oeuvre, de la fin de cycle hiérarchique et des hypothèses du Grafcet, introduit des tâches supplémentaires dans la commande.

- l'interface PC-Procédé est décomposé en deux types de tâches ADA: le tâches d'émission et les tâches de réception.

- enfin les limites du langage ADA, dans la manipulation du temps et des tâches, impose d'intégrer un exécutif temps-réel.

CHAPITRE VI:

ETUDE

D'UN CAS

ETUDE

D'UN CAS.

INTRODUCTION.

Ce chapitre constitue un exemple d'application de l'ensemble de la méthodologie de spécification/mise en oeuvre de la commande en temps-réel dans les cellules flexibles d'assemblage.

Un premier volet de ce chapitre présente les composants impliqués dans l'assemblage ainsi que les gammes d'assemblage possibles telles que présentées au chapitre I.

Un second volet, en reprenant les moyens des chapitres II et III, décrit l'application par les étapes successives de description opératoire, description fonctionnelle, structuration des processus et intégration des contraintes matérielles. Ce niveau intègre, en outre, la spécification de la gamme d'assemblage retenue.

Le troisième volet situe, par rapport à la méthodologie de commande adoptée au chapitre IV, la composition de la Partie Commande de cette application.

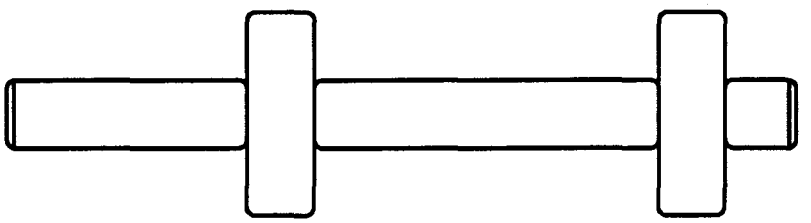
Le quatrième volet présente les différentes entités physiques composant la cellule d'assemblage utilisée.

Un cinquième volet expose la connexion physique de la Partie Commande (PC) avec le Procédé ainsi que le logiciel d'interfacage écrit.

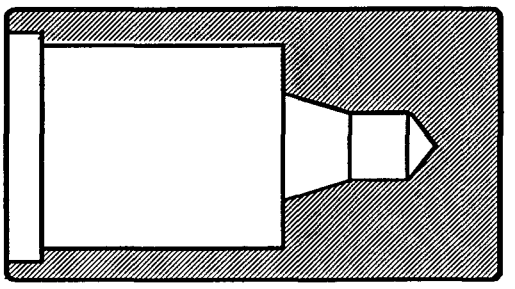
Enfin, le sixième et dernier volet présente l'aspect logiciel complet de l'interface PC-Procédé ainsi que le contexte particulier d'exécution en temps-réel.

I PROCESSUS D'ASSEMBLAGE.

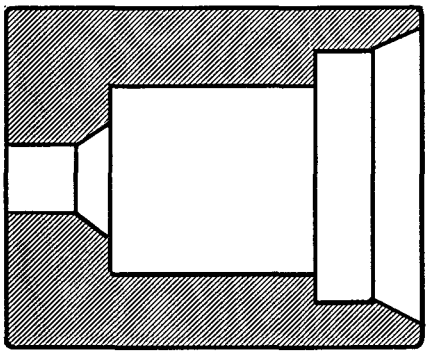
Il s'agit d'assembler un moteur éléctrique à partir de trois composants: rotor, stator et capot. Il existe une variante de deux rotors, trois capots et trois stators. La figure 6.1 représente un exemplaire de chaque composant parmi les trois. Ces diverses variantes permettent d'assembler, avec toutes les combinaisons possibles, dix-huit moteurs différents.



ROTOR



CAPOT



STATOR

figure 6.1

Deux liaisons fonctionnelles L1 et L2 sont nécessaires pour l'obtention du moteur (graphe des liaisons fonctionnelles en figure 6.2). L'existence d'une unique relation n-aire soit R(L1, L2) (il est impossible de réaliser L1 si L2 est établie), autorise une seule gamme d'assemblage possible soit l'établissement de la liaison L1 suivie de la liaison L2.

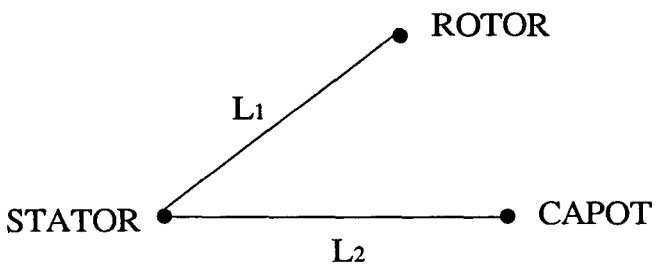


figure 6.2

II DESCRIPTION DE L'APPLICATION.

II-1 Description opératoire.

Les divers composants sont initialement disposés dans un casier à pièces CP. Pour que ces objets soient prélevés, il est nécessaire de disposer d'une part d'un robot et d'autre part d'un système de vision. Ceux-ci sont ensuite transportés par ce même robot vers la table TT puis assemblés par robot(s) sur la table de montage TM selon la gamme d'assemblage retenue. Le moteur obtenu est transféré par un robot vers le convoyeur CV pour être évacué vers l'extérieur.

On distingue ainsi selon la terminologie adoptée au chapitre IV:

- cinq processus de commande: LOCALISE (CP), TRANSFERT (CP, TT), ASSEMBLAGE (TT, TM), TRANSFERT (TM, CV1) et EVACUATION (CV1).

- un processus de gestion du lieu de stockage intermédiaire TT: G.L.S.I (TT).
- un processus de gestion du lieu d'assemblage TM: G.L.A (TM).

La figure 6.3 présente la description opératoire de cette application.

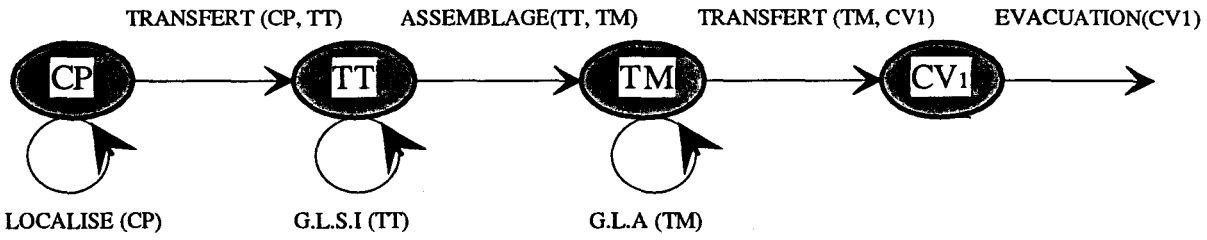


figure 6.3

II-2 Description fonctionnelle.

La figure 6.4 montre les interactions entre divers processus intervenant dans l'assemblage.

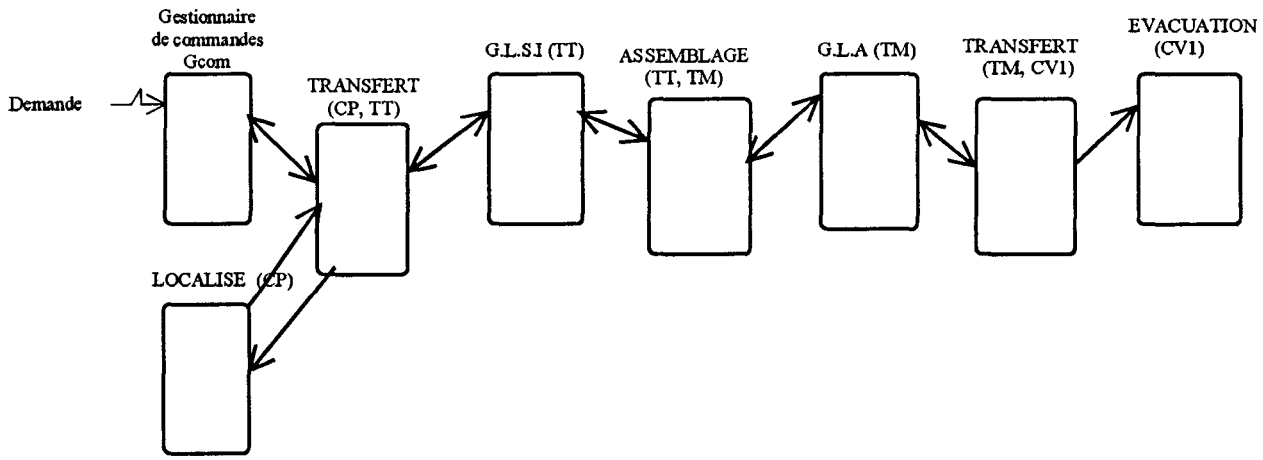


figure 6.4

Nous rappelons qu'une flèche brisée symbolise un événement. Une flèche normale, partant d'un processus vers un autre, représente une communication synchrone du type

simple (producteur/consommateur sans tampon). En revanche, une flèche à double sens, représente une communication synchrone type producteur/consommateur.

L'assemblage s'effectue sur demande de l'opérateur (commande) via un interface de dialogue graphique. L'interfaçage avec l'utilisateur est représenté par un processus gestionnaire de commandes qui constitue un tampon de mémorisation de celles-ci (file d'attente de n éléments d'un type d'objets, gérée en consommation du plus ancien élément).

Le processus TRANSFERT (CP, TT) est le consommateur de ces commandes une par une (communication synchrone). L'existence d'une commande sert à activer ce même processus. Celui-ci se comporte tantôt comme consommateur des processus LOCALISE (CP) et Gcom (Gestionnaire des commandes) et tantôt comme producteur du processus gestionnaire du lieu TT (G.L.S.I (TT)) et du processus LOCALISE (CP).

Le processus gestionnaire du lieu de stockage intermédiaire TT est un ensemble de trois files d'attente (une file par objet) chacune gérée en consommation du premier élément et ayant une capacité d'un élément. Il interagit avec trois producteurs (il s'agit ici du même producteur soit TRANSFERT (CP, TT)) et trois consommateurs (il s'agit encore du même consommateur soit ASSEMBLAGE (TT, TM)) (figure 6.5).

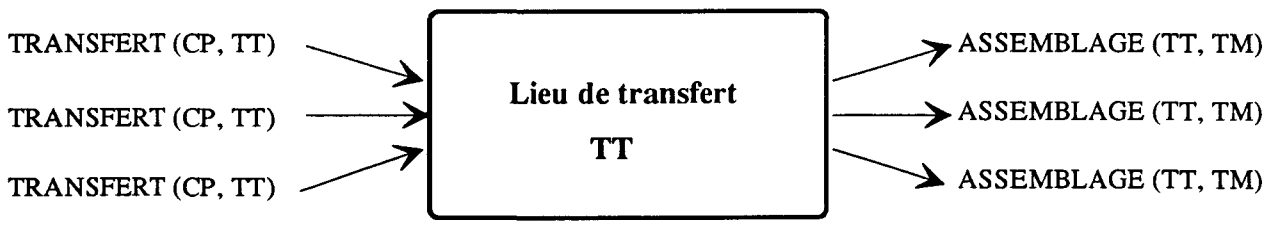


figure 6.5

Le processus gestionnaire du lieu d'assemblage, soit G.L.A (TM), a pour support la gamme d'assemblage retenue. Celle-ci, ainsi que nous l'avons signalé au chapitre III, est clairement exprimée par l'ordre de franchissements des transitions représentant l'établissement des liaisons fonctionnelles. Le processus G.L.A (TM) interagit avec trois producteurs (il s'agit ici du même producteur ASSEMBLAGE (TT, TM)) et un processus consommateur soit TRANSFERT (TM, CV1) (figure 6.6)

figure 6.6

II-3 Structuration des processus.

Le processus TRANSFERT (CP, TT) a pour mission de transférer séquentiellement les objets rotor, stator et capot du lieu CP vers le lieu TT et sur demande de l'utilisateur. Le meilleur temps de cycle, représentant l'ordre dans lequel les objets seront transférés en fonction de la gamme d'assemblage retenue, peut être évalué par l'Algèbre des dioïdes [COH85] [COH90] ou par réseaux de Pétri temporisés [HIL89] [CHA88].

Dans notre cas, l'existence d'une unique gamme d'assemblage impose que le meilleur temps de cycle est celui dont l'ordre de transfert correspond à l'ordre d'assemblage soit stator puis rotor puis capot.

Le processus TRANSFERT (CP, TT), après avoir reçu une commande de la part du processus gestionnaire de commandes, entame un cycle de transfert dans l'ordre cité plus haut. Il utilise ainsi pour le transfert de chaque objet la primitive transfert.

Le processus ASSEMBLAGE (TT, TM) doit, sur autorisation du processus G.L.S.I (TT), acheminer les composants de l'assemblage conformément à la gamme d'assemblage sans quoi il y a risque de blocage. Son cycle d'assemblage correspond à la prise, le déplacement puis l'insertion des composants dans l'ordre stator, rotor, capot. Il utilise pour cela la primitive assemblage pour les trois cas.

Le processus EVACUATION (CV1) a pour mission d'évacuer chaque moteur déposé par le processus TRANSFERT (TM, CV1). Dès qu'une autorisation d'évacuation est

reçue, le processus EVACUATION (CV1) entame un cycle correspondant au lancement du convoyeur. Celui-ci s'arrête automatiquement quelques instants après si un nouveau lancement n'est pas intervenu.

II-4 Intégration des contraintes matérielles.

Un premier robot R1 est alloué au processus TRANSFERT (CP, TT). Un système de vision est réservé au processus LOCALISE (CP). Un convoyeur est dédié au processus EVACUATION (CV1). Un second robot R2 est partagé par les processus ASSEMBLAGE (TT, TM) et TRANSFERT (TM, CV1). Il faut donc un processus gestionnaire de la ressource robot R2 soit GR_{R2}.

Rigoureusement, ce processus doit intégrer une attribution du robot R2 en priorité à un des deux processus en cas de demandes simultanées. Néanmoins du fait que:

- le processus ASSEMBLAGE (TT, TM) ne peut recevoir l'autorisation d'entamer un cycle que si le lieu TM est libre
- et que, dans le pire des cas, le processus TRANSFERT (TM, CV1) n'a pas encore fini son transfert lorsque TM est libre,

il est impossible d'avoir deux demandes simultanées de la ressource R2. Par conséquent, il n'est pas nécessaire d'utiliser les priorités au sein du gestionnaire de cette ressource. En outre, ainsi que nous l'avons signalé au chapitre III, le processus gestionnaire de la ressource robot R2 utilise les outils de partage de ressources (compétition) exposés au chapitre II.

III LA COMMANDE.

Il apparaît, d'après le paragraphe précédent, que toutes les connexions s'expriment en termes de producteur/consommateur. En effet, cette application ne présente pas de contraintes temporelles. Par conséquent, il n'existe pas de Grafcet gestionnaire de contraintes temporelles au niveau hiérarchique NH_2. En revanche, les Grafcets auxiliaires existent pour le cas d'arrêt d'urgence. La figure 6.7 présente les différents niveaux hiérarchiques de la commande représentant cette application.

Au niveau NH_1, on retrouve le Grafcet partiel GP_0 pour l'arrêt d'urgence et la reprise.

Au niveau NH_2 se trouvent les Grafcets partiels auxiliaires: GP_{aux1} pour le robot R1, GP_{aux2} pour le système de vision, GP_{aux3} pour le robot R2 et GP_{aux4} pour le convoyeur. Il n'est pas nécessaire de créer de tels Grafcets pour les processus de gestion de lieu (G.L.S.I (TT) et G.L.A (TM)) et pour le processus de gestion des commandes (GCom) (NH_3), du fait que ceux-ci n'exercent pas d'action sur le procédé.

Au niveau NH_3, on trouve respectivement:

- pour le site robot R1, les Grafcets partiels: GP_{11} pour le processus TRANSFERT (CP, TT) et GP_{ar1} pour l'arrêt d'urgence.
- pour le site système de vision les Grafcets partiels: GP_{21} pour le processus LOCALISE (CP) et GP_{ar2} pour l'arrêt d'urgence.
- pour le site robot R2, les Grafcets partiels: GP_{31} pour le processus ASSEMBLAGE (TT, TM), GP_{32} pour le processus gestionnaire de la ressource robot R2, GP_{33} pour le processus TRANSFERT (TM, CV) et GP_{ar3} pour l'arrêt d'urgence.
- pour le site convoyeur, les Grafcets partiels: GP_{41} pour le processus EVACUATION (CV1) et GP_{ar4} pour l'arrêt d'urgence.
- pour le lieu de stockage intermédiaire TT, le Grafcet partiel GP_{51} gestionnaire de ce lieu.
- pour le lieu d'assemblage TM, le Grafcet partiel GP_{61} gestionnaire de ce lieu.
- et enfin, pour l'interfaçage avec l'utilisateur, le Grafcet partiel GP_{71} qui mémorise les commandes en attendant que le processus TRANSFERT (CP, TT) soit disponible.

La Partie Commande reçoit du procédé trois types d'événements: EF_{R1} (fin de tâche sur robot R1), EF_{R2} (fin de tâche sur robot R2) et EF_{sv} (fin de tâche sur système de vision). Elle reçoit également de l'utilisateur un événement demande via le processus tampon Gcom.

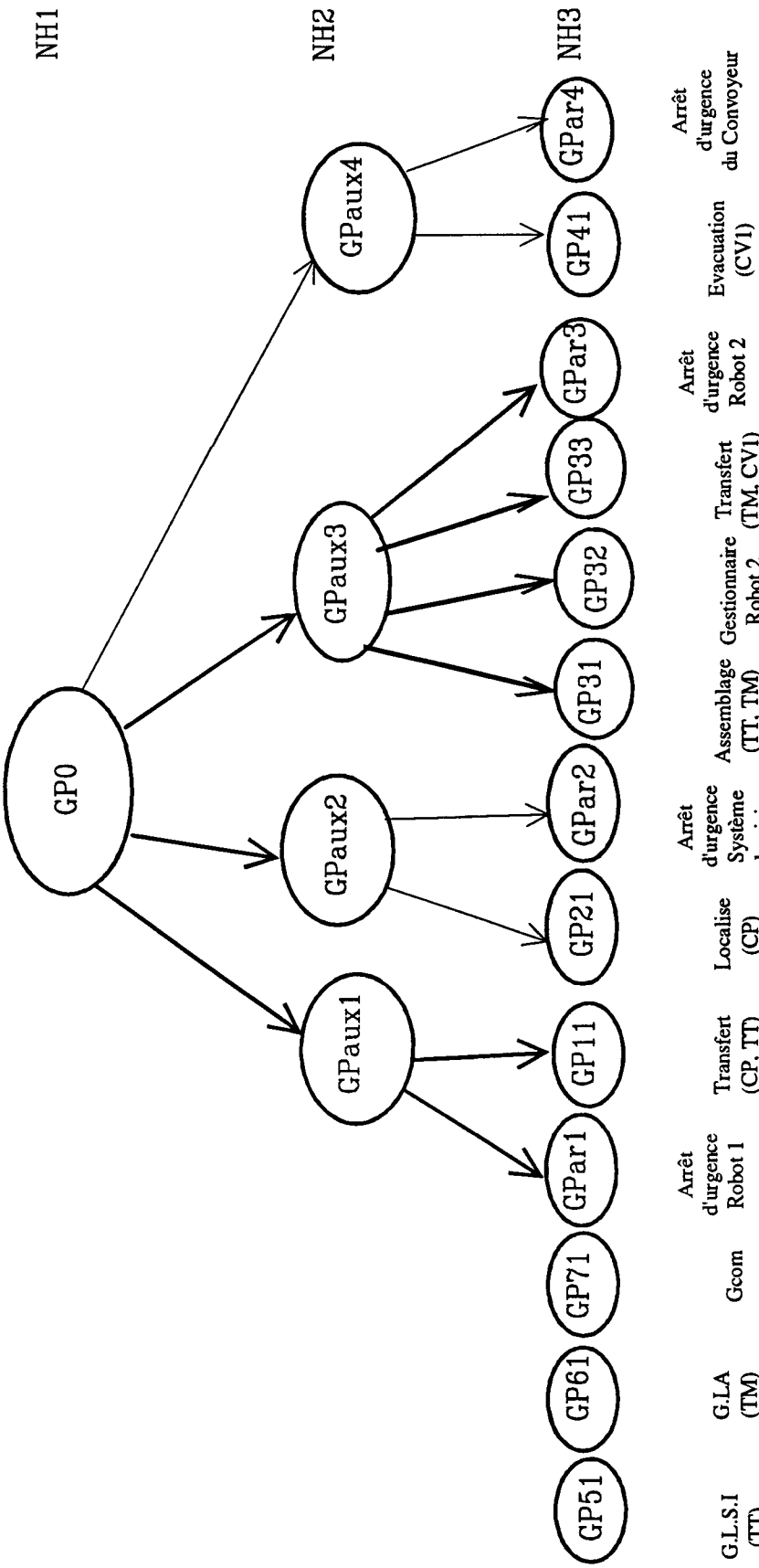

NH1
NH2
NH3
GP0
GPaux1
GPaux2
GPaux3
GPaux4
GP51
GP61
GP71
GPar1
GP11
GP21
GPar2
GP31
GP32
GP33
GPar3
GP41
GPar4
G.L.S.I (TT)
G.LA (TM)
Gcom
Arrêt d'urgence Robot 1
Transfert (CP, TT)
Localise (CP)
Arrêt d'urgence Système de vision
Assemblage (TT, TM)
Gestionnaire Robot 2
Transfert (TM, CV1)
Arrêt d'urgence Robot 2
Evacuation (CV1)
Arrêt d'urgence du Convoyeur

figure 6.7

IV CONSTITUANTS PHYSIQUES.

La cellule d'assemblage sur laquelle s'est effectué notre travail est composée:

- de deux robots six axes **Puma 520** de Staübli [PUM86] programmés en langage robot **VAL** [VAL87]. Des programmes sont écrits sur chacun des robots en fonction des opérations qu'il doit accomplir. Il s'agit alors de lancer ces programmes à partir de la Partie Commande.

- d'un **système de vision** l'**Expert** de chez Allen Bradley Servotronics [BRA86] programmé en langage de vision **PLV** [PLV87]. Le système de vision est équipé dans le cadre de cette manipulation de deux caméras CCD. Une première caméra identifie les objets du type stator et une seconde caméra localise les objets des types rotor et capot. Un programme de reconnaissance et de localisation d'objets est écrit sur le système de vision. La Partie Commande émet un ordre (objet, lieu) au système de vision qui en réponse envoie les informations (coordonnées, orientation) de l'objet demandé.

- d'un convoyeur circulaire vu en tant qu'entrée-sortie d'un robot.

- d'un ordinateur IBM PC_{AT} équipé d'une carte **val multiport** [MUL87] qui remplace la liaison série RS232 par huit (8) liaisons du même type. Cette dernière fournit aussi un interface, dit **superviseur**, pour la communication avec les robots, dans le protocole **DDCMP** spécifique à ceux-ci. Un tel interface peut également être utilisé en tant que liaison **non-DDCMP**. Il appartient alors au programmeur de gérer la liaison concernée. La partie commande, la partie dialogue graphique et la partie interface avec le procédé résident au sein de cet ordinateur.

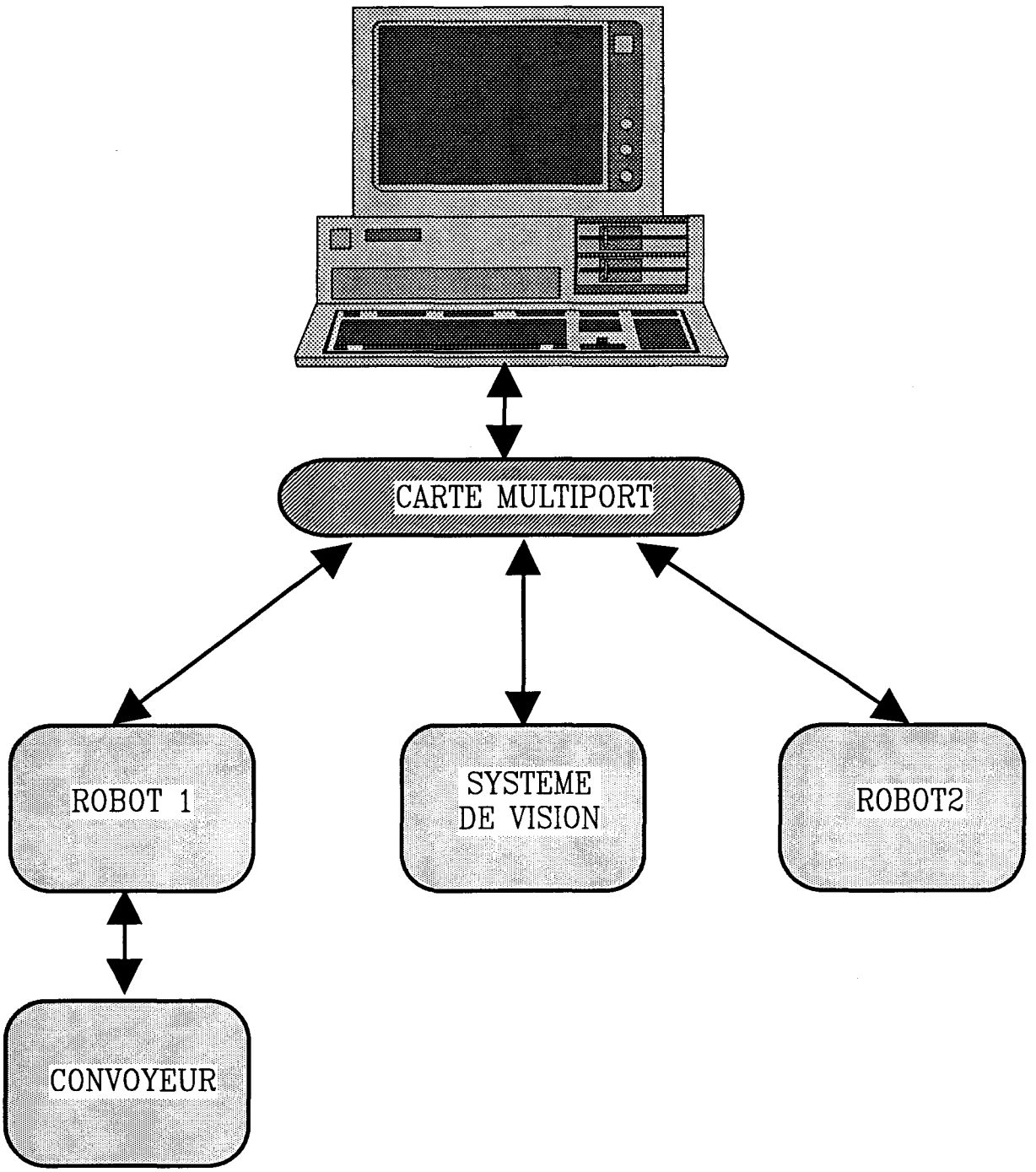

CARTE MULTIPORT
ROBOT 1
SYSTEME
DE VISION
ROBOT2
CONVOYEUR

figure 6.8

V CONNEXION PC-PROCEDE ET LOGICIEL D'INTERFACE.

La communication d'un système externe avec un robot Puma 520 se fait au travers de six unités logiques ayant chacune des fonctionnalités spécifiques et sous des formats bien déterminés [PUM85].

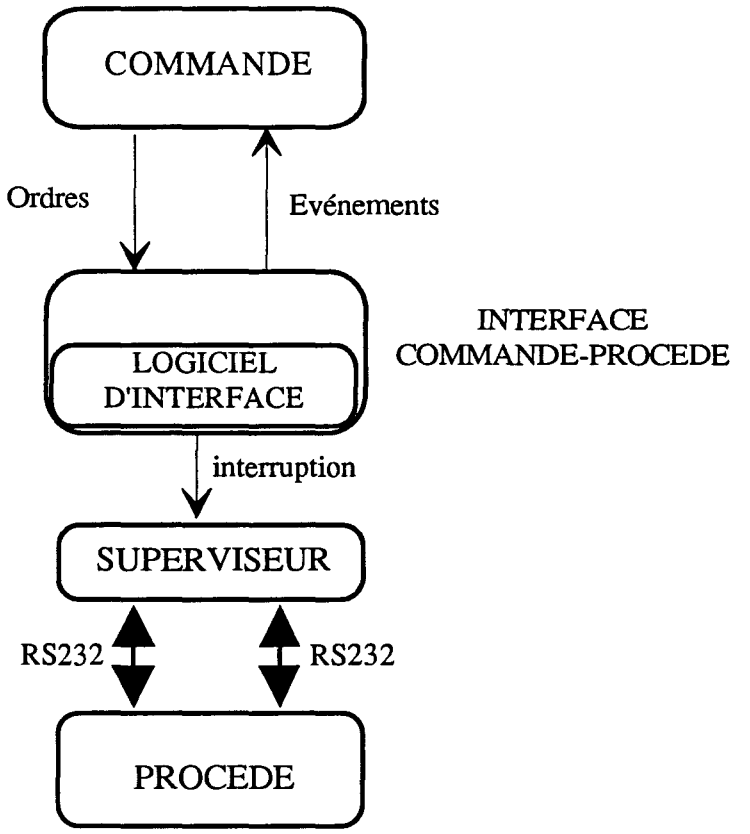


figure 6.9

La communication entre une application et le superviseur se fait par interruption logicielle et passation de l'adresse de début d'un buffer contenant les spécifications de l'opération désirée. Nous avons donc écrit un **logiciel interface** entre la Partie Commande et le Superviseur pour des communications type DDCMP ou non-DDCMP. Le logiciel écrit en langage ADA, est structuré essentiellement en deux niveaux (figure 6.10):

- au niveau bas un module dit *Chareq* fournit, au niveau haut, des procédures usuelles d'ouverture d'un port série, de lecture d'un port etc.. C'est ce niveau qui communique avec le Superviseur.

- au niveau haut un module dit *Exec* contient des procédures évoluées usuelles pour les robots et le système de vision. Elles sont du type *exécuter tâche X, calibrer robot i, Localiser objet x sur yy* etc. Ce module utilise des procédures du module *Chareq*.

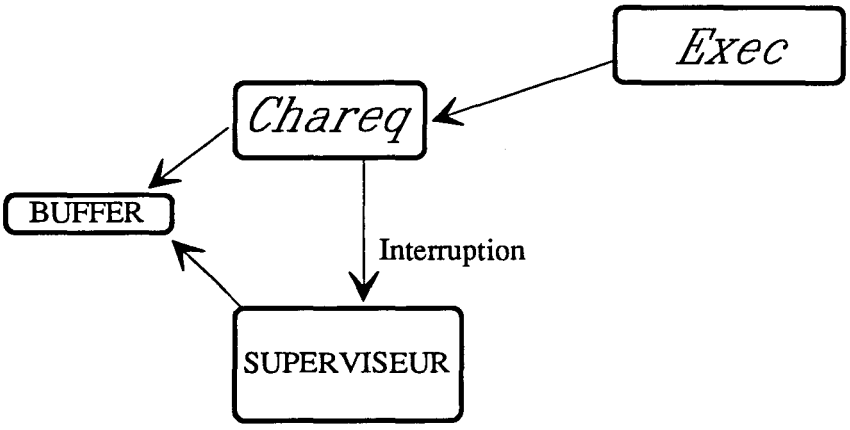


figure 6.10

VI L'INTERFACE PC-PROCEDE.

L'interface Partie Commande-Procédé est composée de deux types de tâches (figure 6.11):

- les tâches d'émission vers le procédé. Il en existe quatre:

  - tâche TE_{R1} pour l'émission des ordres générés par la PC vers le robot R1 et le convoyeur.

  - tâche TE_{R2} pour l'émission des ordres générés par la PC vers le robot R2.

  - tâche TE_{SV} pour l'émission des ordres en provenance de la PC et à destination du système de vision.

  - tâche TE_{DI} pour l'émission de comptes rendus en direction du dialogue graphique.

- les tâches de réception du procédé. Il en existe également quatre:

  - TR_{R1} tâche de réception des comptes rendus en provenance du robot R1 et du convoyeur.

  - TR_{R2} tâche de réception des comptes rendus en provenance du robot R2.

  - TR_{SV} tâche de réception des comptes rendus en provenance du système de vision.

  - TR_{DI} tâche de réception des événements et commande en provenance du dialogue graphique.

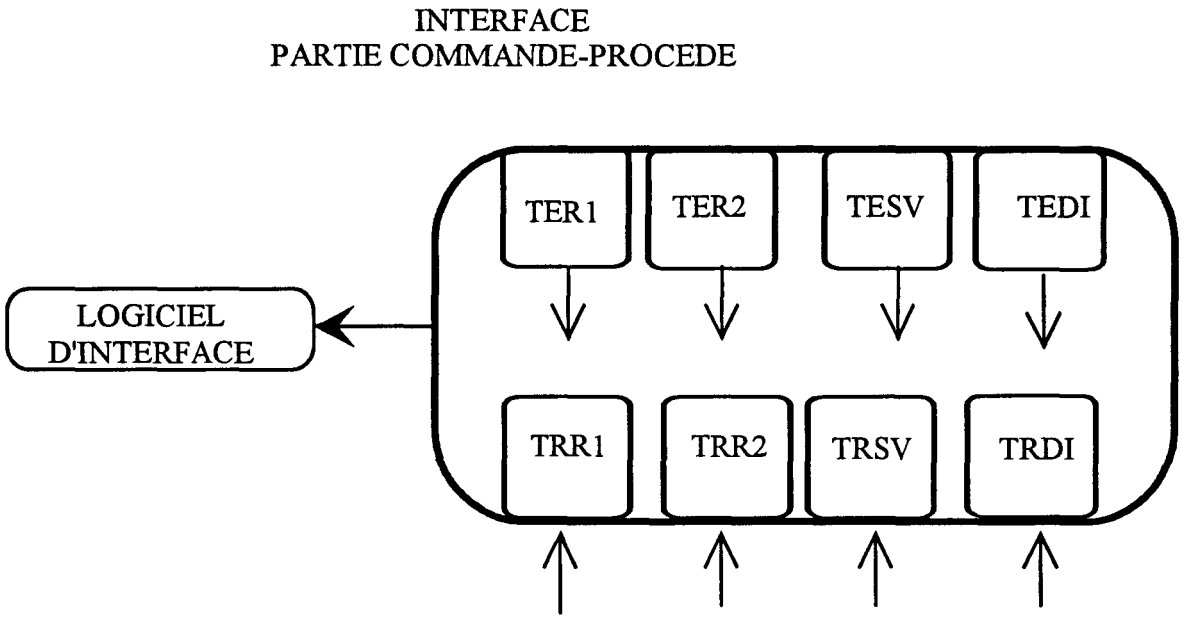


figure 6.11

Les différentes tâches de l'interface sont des tâches ADA:

- les diverses tâches d'émission, constamment en attente, sont déclenchées par demande de la PC qui transmet en même temps les paramètres de l'opération. Ces différentes tâches utilisent des procédures du logiciel d'interface et plus précisément du module *Exec*.

- les tâches de réception sont constamment actives et scrutent, chacune selon son propre rythme, les événements en provenance du procédé (ceux-ci arrivent d'une façon aléatoire). En outre, pour accomplir leurs fonctions, ces tâches utilisent des procédures du module *Exec* du logiciel d'interface.

  Ainsi que nous l'avons expliqué au chapitre précédent, ces tâches sont groupées en deux niveaux de priorités: la plus haute priorité pour les tâches de réception et la priorité suivante pour les tâches d'émission.

  En outre, du fait de l'absence de contraintes relatives fortes au niveau des tâches de réception (les diverses liaisons fonctionnent avec des vitesses sensiblement égales), celles-ci se déroulent en temps partagé les unes par rapport aux autres (même temps de scrutation).

Le compilateur ADA utilisé pour la programmation de l'application est la version 2.1 de Meridian [MER88] maintenant dépassée. Celle-ci n'était pas équipée du temps partagé (time slicing, déclarer un temps d'exécution pour chaque tâche). Nous avons dû forcer ce partage en insérant des instructions de suspension: **delay (0.0)**.

En toute rigueur et bien qu'il n'y a pas de contraintes temporelles au niveau de l'application même ni de contraintes relatives fortes au niveau réception de l'interface, il faut utiliser un exécutif temps-réel. Ceci est nécessaire pour l'accomplissement des arrêts d'urgence lorsque la tâche d'émission concernée est déjà en cours d'émission.

Nous avons néanmoins, par un artifice de programmation, évité l'utilisation d'un exécutif temps-réel. Cela est obtenu en dédoublant les tâches d'émission TE_{R1}, TE_{R2} et TE_{SV} pour donner naissance aux tâches TE'_{R1}, TE'_{R2} et TE'_{SV}. Celles-ci sont en attente et sont réceptives exclusivement à un ordre d'arrêt d'urgence. Dès qu'une de ces tâches réquisitionne le processeur, elle le garde jusqu'à ce qu'elle termine son exécution: il n'y a pas d'insertion de l'instruction delay (0.0).

Ainsi, lorsque l'arrêt d'urgence s'adresse aux trois tâches dédoublées, elles sont exécutées d'une façon séquentielle. Ceci ne constitue pas un handicap, car les temps d'exécution des ordres sur les robots sont relativement longs devant les temps d'exécution de ces mêmes tâches. L'utilisateur a l'impression de voir toutes les machines s'arrêter en même temps.

CONCLUSION.

L'application spécifiée et mise en oeuvre de la manière exposée dans ce chapitre, a constituée un exemple d'application de l'ensemble de la méthodologie de spécification/mise en oeuvre de la commande en temps-réel dans les cellules flexibles d'assemblage.

En effet, cette application a repris:

- la méthode de structuration des applications exposée au chapitre IV, qui, elle même, intègre les outils de spécification des caractères réactif et transformationnel des applications temps-réel.
- la méthodologie de commande adoptée du chapitre IV.
- les modèles d'interprétation et de mise en oeuvre exposés au chapitre V.

Toutefois, cette application n'a pas intégré des contraintes temporelles (absence de gestionnaires de contraintes temporelles au niveau hiérarchique NH_2). Les algorithmes d'interprétations liés notamment au modèle de hiérarchie et aux ordres proposés sont néanmoins validés du fait de l'intégration des arrêts d'urgence.

La mise en oeuvre de l'application, qui par ailleurs a nécessité une programmation périphérique importante (langage de vision..) s'est avérée, comme on s'y attendait, très satisfaisante.

Il serait toutefois intéressant de réécrire l'application avec une version temps partagé du compilateur. On doit alors évaluer précisément les temps à octroyer aux différentes tâches pour l'exécution.

Un interface graphique fournit à l'utilisateur un menu graphique, souple, interactif et utilisable en parallèle avec l'application. Celui-ci dispose de moyens pour la commande de moteurs, le suivi de production, l'arrêt d'urgence par touche et l'arrêt du déroulement de l'application.

CONCLUSION GENERALE.

La recherche d'un accroissement de productivité et de qualité ne constitue pas une préoccupation nouvelle du secteur industriel. C'est une des constantes de l'histoire industrielle dans une économie de marché, même si la concurrence est aujourd'hui plus que jamais le moteur et le censeur de la rationalisation de l'outil de production.

L'irruption de l'informatique dans les industries manufacturières s'est d'abord traduite par des techniques d'automatisation de postes de travail. Au fur et à mesure des progrès sectoriels de ces techniques est apparu le problème de leur cohérence globale.

L'enjeu de la Productique est donc non seulement le progrès des technologies d'automatisation, mais encore et peut être surtout l'intégration des techniques et des méthodes d'automatisation dans la problématique de production.

L'un des freins à la réalisation de cet objectif est la diversité des méthodes, souvent inadaptées, de spécification des applications de commande temps-réel particulièrement dans le cas des cellules flexibles d'assemblage.

Sur le plan de la mise en oeuvre, l'absence de l'automatisation de celle-ci, constitue également un frein à la réalisation de cette intégration.

Notre contribution à ce problème de spécification est avant tout un travail d'intégration d'outils et d'une stratégie.

En effet, Nous avons appréhendé ce problème par l'utilisation d'un outil synchrone: le Grafcet et par des moyens puissants d'expression de l'aspect fonctionnel et événementiel des applications de commande temps-réel dans les cellules flexibles d'assemblage.

Par ailleurs, il fallait structurer les applications non seulement pour faire face à la complexité du système analysé, mais également pour destiner la représentation à des utilisateurs.

Sur le plan de la transitique, nous avons suggéré une structuration logique de la commande, basée sur une hiérarchisation des objectifs. Ces niveaux hiérarchiques se justifient par la pertinence temporelle de ces divers objectifs.

Sur le plan outil, cette démarche a nécessité de nouveaux développements autour du Grafcet dans la continuité de ceux apportés par l'ADEPA (hiérarchie et macroactions).

L'aspect informationnel, concernant la définition des données, événements et traitements associés, n'a pas été formalisé dans le cadre de ce travail, bien que nous avons dû en tenir compte lors de la réalisation.

Sur le plan de la mise en oeuvre, les développements autour du Grafcet que nous avons apporté à coté de ceux de l'ADEPA ont nécessité de nouvelles interprétations algorithmiques originales. L'utilisation du langage temps-réel ADA aux divers niveaux de l'application puis ses carences dans la manipulation du temps, dans un contexte centralisé, ont suggéré l'intégration d'un exécutif temps-réel approprié. Par ailleurs, le langage de mise en oeuvre étant le langage ADA, nous avons récupéré un nombre considérable de bibliothèques écrites avec ce langage ou d'autres langages.

Enfin, l'ensemble de la démarche méthodologique dans la spécification et la mise en oeuvre de la commande en temps-réel a été expérimenté sur un cas de cellule flexible d'assemblage de moteurs électriques. Les résultats se sont avérés, comme attendu, très concluants puisqu'ils ont permis, en particulier, de valider l'ensemble de la méthodologie et des algorithmes d'interprétation proposés.

Le travail présenté dans le cadre de cette étude, s'intègre dans une démarche globale de conduite d'atelier flexible d'assemblage. Cette démarche suppose, en particulier, l'intégration de l'outil Grafcet, modélisant la commande globale, avec les outils de modélisation de la partie conduite.

En outre, il faut également tenir compte de l'aspect surveillance au niveau commande globale: surveillance de la commande et surveillance du procédé [SAH86].

Dans le cadre de l'étude que nous nous étions fixée, la flexibilité du système de production en termes de changement de gamme d'assemblage n'est pas prise en compte. Cette souplesse est nécessaire pour mieux répondre aux divers aléas qui se produisent dans les cellules d'assemblage (meilleure flexibilité). Cette éventualité

suggère la restructuration des applications notamment par la mise en oeuvre de processus associés aux objets de l'assemblage rendant la transitique du système plus complexe. La prise en compte de ce type de flexibilité constitue un prolongement logique de ce travail.

Par ailleurs, des travaux sont actuellement en cours pour la réalisation d'un interface graphique et convivial:

- pour la spécification des processus d'assemblage conformément à la méthodologie exposée.

- pour l'obtention de l'implémentation finale à partir de la simple exécution de la spécification. Une combinaison du Grafcet, de la démarche de structuration des applications en motifs réutilisables et du langage ADA, offrant par ailleurs des constructions génériques, est une opportunité intéressante [BOO82][BOO83].

Au demeurant, nous espérons contribuer par cette thèse à l'élaboration de principes méthodologiques pertinents pour une meilleure intégration de la commande temps-réel dans les systèmes d'assemblage.

ANNEXES

ANNEXE 1: LE GRAFCET, ELEMENTS DE BASE.

Le **Grafcet** exprime les comportements attendus de systèmes logiques. Sa **représentation**, faite à partir d'**éléments graphiques de base**, comprend des **étapes**, des **transitions** et des **liaisons orientées**.

Ses **évolutions** sont définies par cinq **règles d'évolution**. Son **interprétation** se traduit par des **actions** associées aux étapes et des **réceptivités** associées aux transitions.

1 Les étapes.

Une étape caractérise le comportement invariant d'une partie ou de la totalité du système isolé représenté. A un instant donné et suivant l'évolution du système, une étape est soit active soit inactive.

Une étape se représente par un carré identifié par un repère alphanumérique. Pour indiquer les étapes qui sont actives à cet instant, un point peut être placé dans la partie inférieure des symboles des étapes concernées.

Les étapes initiales se représentent par un double carré. Elles indiquent les étapes qui sont actives au début du fonctionnement (situation initiale).

1-2 Les actions associées à une étape.

Une ou plusieurs actions peuvent être associées à une étape. Elles traduisent ce qui doit être fait chaque fois que l'étape à laquelle elles sont associées est active.

Ces actions peuvent correspondre à des émissions d'ordres vers la partie opérative ou vers des éléments extérieurs au système décrit, ou consister en des commandes de fonctions opérations associées telles que des mémoires, compteurs, opérateurs à retard etc.

2 Les transitions.

Une transition indique la possibilité d'évolution entre plusieurs étapes. Cette évolution s'accomplit par le franchissement de la transition qui provoque un changement d'activité des étapes.

Une transition est représentée par un trait perpendiculaire aux liaisons joignant deux étapes. Il n'y a toujours qu'une seule transition entre deux étapes, quels que soient les chemins parcourus. Une transition est soit validée soit non validée. Elle est dite validée lorsque toutes les étapes immédiatement précédentes reliées à cette transition sont actives.

2-1 Les réceptivités associées aux transitions.

A chaque transition est associée une condition logique appelée condition de transition ou réceptivité qui peut être soit vraie soit fausse. La réceptivité est une fonction booléenne écrite à droite du symbole de transition. Elle est constituée d'informations externes (informations d'entrées,..) et/ou internes (états d'étapes..).

3 Les liaisons orientées.

Les liaisons orientées relient les étapes aux transitions et les transitions aux étapes. Elles indiquent les voies d'évolution. Les liaisons orientées se représentent par des lignes horizontales ou verticales. Par convention, le sens des évolutions s'effectue toujours du haut vers le bas. Des flèches doivent être utilisées lorsque cette convention n'est pas respectée.

4 Les règles.

4-1 Règles de syntaxe.

L'alternance étape-transition et transition-étape doit toujours être respectée quelle que soit la séquence parcourue. Deux étapes ou deux transitions ne doivent jamais être reliées par une liaison orientée. La liaison orientée relie obligatoirement une étape à une transition ou une transition à une étape.

4-2 Règles d'évolution.

Règle 1: Situation initiale.

La situation initiale d'un Grafcet caractérise le comportement initial de la partie commande vis-à-vis de la partie opérative, de l'opérateur et/ou des éléments extérieurs. Elle correspond aux étapes actives au début du fonctionnement. Elle traduit généralement un comportement de repos.

Règle 2: Franchissement d'une transition.

Une transition est dite validée lorsque toutes les étapes immédiatement précédentes reliées à cette transition sont actives. Le franchissement d'une transition se produit lorsque la transition est validée et que la réceptivité associée à cette transition est vraie. Lorsque ces deux conditions sont réunies, la transition devient franchissable et est alors **obligatoirement franchie.**

Règle 3: Evolution des étapes actives.

le franchissement d'une transition entraîne simultanément l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.

Règle 4: Evolutions simultanées.

Plusieurs transitions simultanément franchissables sont simultanément franchies.

Règle 5: Activation et désactivation simultanée d'une étape.

Si, au cours du fonctionnement, la même étape est simultanément activée et désactivée elle reste active.

ANNEXE 2: EXTENSIONS DU GRAFCET.

1 Système isolé.

On a l'habitude de diviser les systèmes automatisés de production en deux sous-ensembles communicants: la partie commande et la partie opérative (procédé). Pour définir la partie commande, qui réalise la fonction décision dans le système, il faut au préalable définir ses contours vis à vis (figure A2.1):

- de la partie opérative, d'une part, par le biais:
  - d'ordres à donner à celle ci.
  - d'informations nécessaires à l'élaboration de ces décisions.
- de l'environnement, d'autre part, par le biais:
  - d'ordres reçus de celui ci.
  - d'informations à communiquer à celui ci.

Le Grafcet est un modèle formel de représentation des comportements attendus des systèmes logiques. L'utilisation du Grafcet à tout système nécessite la **détermination préalable de la frontière de ce système par rapport à son environnement.**

La modélisation repose donc sur la partition de l'univers en deux sous univers (figure A2.2):

- celui à l'intérieur de la frontière appelé système isolé.
- celui de l'extérieur de la frontière avec lequel le système entre en action.

Par ailleurs pour que la description soit fidèle au modèle Grafcet, il faut que la pertinence du choix des variables d'entrée conduit à ce que dans le vecteur d'entrée du système isolé, une seule composants peut varier à un seul instant donné.

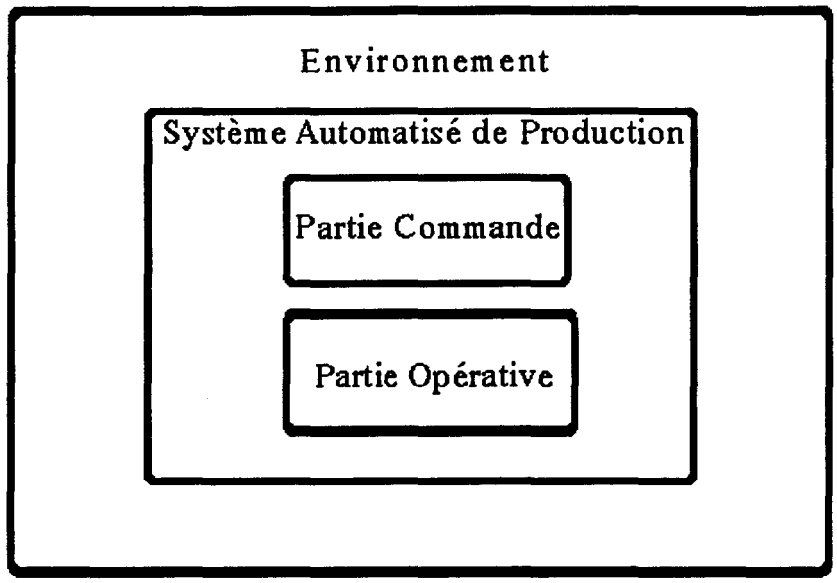


figure A2.1

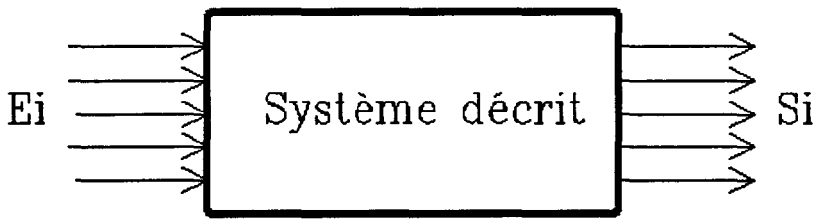


figure A2.2

2 Modèle temporel du Grafcet.

Un système de décision doit être:

- **attentif**, c'est à dire réceptif à tout événement externe et à tout instant.
- **efficace**, c'est à dire capable de traiter les conséquences de tout événement externe.
- **déterministe**, c'est à dire avoir un même comportement (sorties) pour des circonstances identiques (entrées et état interne).

Le modèle Grafcet postule que tous les événements externes sont pris en compte dès leurs occurrences et pour toutes les incidences. Ceci suppose que toutes les évolutions et affectations de sorties consécutives à un premier événement sont accomplies avant

l'apparition d'un nouvel événement externe. Il vérifie par conséquent les deux premières qualités.

La description d'un système est déterministe si, à toute séquence de variation du vecteur d'entrée correspond une et une seule séquence de variation du vecteur de sortie. Le modèle Grafcet élabore les sorties à partir des entrées et de l'état interne du système. Le déterminisme du modèle Grafcet nécessite la formalisation d'un modèle temporel qui le régit. La norme européenne CEI/IEC 848 établit une partition temporelle identique à la frontière de description établi pour l'isolement du système. En outre, cette norme formule les **postulats temporels** suivants:

- La frontière du système définit une partition de l'univers en un **interne** et un **externe** caractérisés chacun par une échelle de temps. Ces deux échelles interne et externe sont non commensurables.

  Les intervalles de temps interne sont infiniment petits donc considérés nuls à échelle de temps externe, d'où le caractère **synchrone** du Grafcet. C'est à dire que la **causalité est considéré à temps nul**. Ce qui confirme que la partie commande doit être très rapide par rapport à la partie opérative.

- **Postulats relatifs au temps externe.**

  - dans le temps externe, seuls sont connues les dates d'occurrences des événements externes.

  - Les occurrences d'événements externes non corrélés sont temporellement distincts.

- **Postulats relatifs au temps interne.**

  - dans le temps interne, seules sont connues les dates d'occurrences des événements internes.

  - les changements de situation sont des événements.

  - la durée séparant l'instant où une transition est franchissable de l'instant où elle est effectivement franchie (durée d'évolution) est non nulle à échelle de temps interne.

3 Partition d'un Grafcet.

Un **Grafcet connexe** est tel qu'il existe toujours une liaison orientée explicite entre deux éléments quelconques (étape ou transition).

Un **Grafcet partiel** est un sous-ensemble de plusieurs Grafcets connexes. Un Grafcet connexe pris séparément peut aussi constituer un Grafcet partiel. Un Grafcet partiel est actif si au moins une de ces étapes est active.

Un **Grafcet global** décrivant un système est l'ensemble de tous les Grafcets partiels décrivant le comportement de ce système.

On définit pour un Grafcet partiel l'ensemble des situations suivantes:

- situation initiale: correspond à l'ensemble de ses étapes actives à l'instant initial.
- situation courante: correspond à l'ensemble de ses étapes actives à l'instant considéré. Elle est notée {*}.
- situation vide: correspond à la situation dans laquelle aucune étape n'est active. Elle est notée { }.
- situation donnée: correspond à la situation dans laquelle seules les étapes i, j,.. sont actives. Elle est notée {i, j,..}.

Dans la norme CEI 848 une situation est nommée par NS: GRn, {LE} où

NS est le nom de la situation, GRn est le nom du Grafcet auquel NS appartient, LE est la liste des étapes actives de GRn pour cette situation.

4 Hiérarchie et Forçage.

Les ordres de forçage permettent de modifier de manière interne la situation d'un Grafcet partiel à partir d'un autre Grafcet partiel. Cette relation de dépendance entre Grafcets partiels implique une hiérarchie entre ces deux graphes.

En outre, une cohérence d'ensemble pour l'application de cette hiérarchie est nécessaire pour assurer le déterminisme du système décrit:

- si un Grafcet force un autre Grafcet la réciproque est impossible.

- à tout instant de fonctionnement, un Grafcet ne peut être forcé que par un et un seul autre Grafcet.

Le forçage permet d'imposer à un Grafcet une situation qu'il aurait été impossible ou difficile d'atteindre directement. A l'intérieur de la frontière de description, cette dernière est régit par:

- les règles générales du Grafcet permettant les évolutions des Grafcets en fonction des événements externes et des événements internes (interactions entre Grafcets).

- les ordres internes de forçage.

Pour lever toute ambiguïté, notamment en cas de conflit, la prise en compte des règles d'évolution et de l'application d'un ordre de forçage doit être bien définie. Les règles suivantes résolvent tout conflit:

Règle 1 (RF1):

Le forçage est un ordre interne consécutif à une évolution. Pour une situation comportant un ou plusieurs ordres de forçage, les Grafcets forcés prendront **immédiatement** et **directement** la ou les situations imposées.

Règle 2 (RF2):

- **-a-** à toute apparition d'une nouvelle situation l'application du forçage est prioritaire par rapport à toute activité du modèle (évolution, affectation des sorties,..).

- **-b-** les règles d'évolution ne s'appliquent qu'à une situation pour laquelle le Grafcet partiel forcé est dans la situation imposée par le Grafcet forçant.

Il en résulte que seules les situations où les Grafcets forcés sont dans les situations imposés, sont des situations internes. En d'autres termes ne peuvent exister que des situations libres de tout forçage.

REFERENCES BIBLIOGRAPHIQUES

REFERENCES BIBLIOGRAPHIQUES.

[**ADE77**] Rapport final de la commission AFCET: Normalisation de la représentation d'un cahier des charges d'un automatisme logique, Revue Automatique et Informatique Industrielles, Novembre - décembre 1977.

[**ADE92**] ADEPA, AFCET, Le Grafcet, Ed Cepadues 1992.

[**AFC83**] Groupe Systèmes Logiques de l'AFCET, "Grafcet, interprétations algébriques et algorithmiques et temporisations", le Nouvel Automatisme, Septembre 1983.

[**AGA87**] S.Agaoua, "Spécification et commande des systèmes à événements discrets, le Grafcet coloré", Thèse de Doctorat de l'université de l'INPG, Juillet1987.

[**ALB82**] J.Albus, C.Maclean, A.Barbera, M.Fitzgerald, "An architecture for real-time control in a manufacturing facility", IFAC information control problems in manufacturing technology, Maryland, USA, 1992.

[**ALL89**] H.Alla, R.David, "Du Grafcet aux Réseaux de Petri", Ed Hermès, Paris 1989.

[**AND92-a**] C.André, M.A.Péraldi, "Grafcet et langages synchrones", Congrès Grafcet'92, Paris 1992.

[**AND92-b**] C.André, "Les approches synchrones et les systèmes automatisés de production", Réunion annuelle Groupement de recherche en automatique, Paris, septembre1992.

[**BEN90**] A.Benveniste, "Les langages synchrones: des noyaux logiciels pour la spécification et la conception des systèmes de contrôle-commande", Colloque AFCET le temps-réel, Nantes, Octobre 1990.

[**BNI82**] Bureau d'orientation de la normalisation en informatique, "Sceptre: proposition de noyau normalisé pour les exécutifs temps-réel", T.S.I, vol 3, 1984.

[**BOO82**] G.Booch, "Software design with ADA" ACM Sigplan notices, Septembre 1992.

[**BOO83**] G.Booch, Software engineering development with ADA", Ed Benjamin Cummings, 1983.

[**BOU84**] A.Bourjault, "Contribution à une approche méthodologique de l'assemblage automatisé", Thèse de doctorat Es-sciences physiques, Besançon 1984.

[**BOU86**] A.Bourjault, F.Lhote, "Modélisation d'un processus d'assemblage", APII, 1986, 20.

[**BOU87-a**] A.Bourjault, D.Chappe, J.M.Henrioud, "Elaboration automatique des gammes d'assemblage à l'aide de réseaux de Petri", APII, 1978, 21.

[**BOU87-b**] A.Bourjault, J.M.Henrioud, "Détermination des sous-assemblages d'un produit à partir des séquences temporelles d'assemblage", APII, 1987, 21.

[**BOU88-a**] J.P.Bourey, "Structuration de la partie procédurale du système de commande de cellules de production flexibles dans l'industrie manufacturière", Thèse de doctorat de l'université de Lille I, Mars 1988.

[**BOU88-b**] J.P.Bourey, .J.C.Gentina, "Structuration de la partie procédurale du système de commande de cellules flexibles de production", AFCET Congrès Automatique, Grenoble, Octobre 1988.

[**BRA86**] Allen Bradley Servotronics, "Documentation sur le système de vision Expert", 1986.

[**CAL82**] J.P.Calvez, " Une méthodologie de conception des systèmes multi-microordinateurs pour les applications de commande en temps-réel", Thèse de doctorat Es-sciences physiques, Nantes, Novembre 1982.

[**CAR90**] C.Carrez, C.Kaiser, "Le langage ADA et le temps-réel", Colloque AFCET le temps-réel, Nantes, Octobre 1990.

[**CAS91**] P.Capsi, N.Halbwachs, P.Raymond, D.Pilaud, "The synchronous data flow programming language LUSTRE", Proceedings of the IEEE, Vol 74, N 9, Septembre 1991.

[**CEI88**] Etablissement des diagrammes fonctionnels pour systèmes de commande/Preparation of functions charts of control systems, Norme International/International Standard, CEI/IEC 848, Décembre 1988.

[**CHA88**] D.Chappe, A.Bourjault, "Une méthodologie de determination de l'organisation d'un atelier d'assemblage", Congrès AFCET Automatique, Grenoble, Octobre 1988.

[**CHA92**] V.Chapurlat et al, "Modèle structuré de spécification de systèmes à événements discrets utilisant le Grafcet: le modèle ACSY", Congrès Grafcet'92, Paris, 1992.

[**COU83**] M.Courvoisier et al, "Le projet SECOIA, document de définition", rapport LAAS, No.83091, Décembre 1983.

[**COU84**] M.Courvoisier, M.Bigou, R.Valette, C.Desclaux, K.Benzakour, " The SECOIA project", 6th European Conf on elec, Eurocon, Brighton, U.K, Septembre 1984.

[**COH85**] G.Cohen et al, "Une théorie linéaire des systèmes à événements discerts", Rapport INRIA, Janvier 1985.

[**CRO75**] Groupe Crocus, "Systèmes d'exploitation des ordinateurs", Ed Dunod, Paris, 1977.

[**DAL84**] Y.Dallery, H.Deneux, R.David, "Recherche d'une même base de description en vue de la simulation et de la commande d'un atelier flexible: utilisation du Grafcet", Congrès AFCET Automatique, Besançon, Novembre 1983.

[**DEF86**] J.Defrenne, "Modélisation de la partie opérative - Impact sur la sécurité et la maintenance des automatismes à évolution séquentielle", Thèse de docteur Es-sciences physiques, Lille I, 1986.

[**DEN83**] H.Deneux, "Contribution à l'étude des réseaux d'automates interconnectés" Thèse de docteur-ingenieur, Grenoble, Février 1983.

[**DOD79**] Reference manual for the ADA programming language, U.S.Dept of Defense, 1983.

[**ELL90**] J.P.Elloy, "Qu'est ce que le temps-réel", Colloque AFCET le temps-réel, Nantes, Octobre 1990.

[**FRA92**] J.P.Frachet, G.Colombari, "Eléments pour une sémantique temporelle du Grafcet et des systèmes dynamiques fondée sur l'analyse non standard" Congrès Grafcet'92, Paris, 1992.

[**GOM84**] H.Gomaa, "A software design method for real-time systems", Communications of the ACM, Vol 27, N 9, September 1984.

[GOM86] H.Gomaa, "Software development of real-time systems", Communications of the ACM, Vol 29, N 7, July 1986.

[GRE85] GREPA, le Grafcet, de nouveaux concepts, Ed Cepadues, Toulouse, 1985.

[HAR85] D.Harel, A.Pnueli, "On the development of reactive systems in logic and models of concurrent systems", NATO ASI series, Vol 13, Springer verlag, 1985.

[HAT87] D.J.Hatley, I.A.Pirbhai, "Startegies for real-time system specification", Dorset house publishing Co inc, 1987.

[HAT91] D.J.Hatley, I.A.Pirbhai, "stratégies de spécification des systèmes temps-réel (SA-RT)" Ed Masson, 1991.

[HIL89] H.Hillion, J.M.Proth, "Performance evaluation of job-shop systems using timed event-graphs", IEEE trans on automatic control, Vol 34, N 1, Jan 89.

[ILI90] J.M.Ilié, "CLEOPATRE, conception, validation, evaluation à l'aide d'outils réseau de Petri d'applications temps-réel", Colloque AFCET le temps-réel, Nantes, Octobre 1990.

[KOM84] N.Komoda et al, "An autonomous decentralized control system for factory automation", Computer, Decembre 1984.

[LAD77] P.Ladet, "Outils de structuration en temps-réel pour la commande des procédés industriels", Thèse de 3ème cycle, INPG Grenoble, 1977.

[LAD82] P.Ladet, "Contribution à l'étude des systèmes informatiques répartis pour la commande des procédés industriels", Thèse de docteur Es-sciences physiques, Grenoble, Juin 1982.

[LEG91] P. Le Guernic, T.Gautier, M.Le Borgne, C.Le Maire, "Programming real-time applications with SIGNAL", Proceedings of IEEE, Vol 79, N 9, Septembre 1991.

[LHO88] F.Lhote, M.Meunier, R.Chossat, "familles de produits et assemblage flexible", Congrès AFCET Automatique, Grenoble, Octobre 1988.

[LHO92] P.Lhoste, H.Panetto, M.Roesch, "Grafcet: de la syntaxe à la sémantique", Congrès Grafcet'92, Paris, 1992.

[MAR92] L.Marcé, P.Le Parc, "Modélisation de la sémantique du Grafcet à l'aide de processus synchrones", Congrès Grafcet'92, Paris, 1992.

[**MER88**] Meridian adavantage, "Compilateur ADA pour pc, version 2.2", 1988.

[**MOA85**] M.Moalla, "Réseaux de Petri interprétés et Grafcet", Revue TSI de l'AFCET, Vol 4, N 1, Jan-Fev 1985.

[**MOS89**] J.M.Moser, R.Stepourjine, J.Vial, "Méthodologie de conception d'une cellule flexible d'usinage", APII, 23, 1989.

[**MUL87**] Unimation, "Val II- IBMpc Supervisor interface", June 1987.

[**OLI86**] P.Olivier, "Analyse et synthèse des systèmes d'assemblage automatisé", Thèse de doctorat de l'université de Franche-Comté, Mars 1986.

[**PER90**] J.P.Perrez, "Systèmes temps-réel, Méthodes de spécification et de conception", Ed Dunod, 1990.

[**PET74**] J.L.Peterson, T.H.Bredt, "A comparison of models of parallel computation", Proc IFIP, 1974.

[**PIC87**] B.Picard d'Estelan, B.Descotes-Genon, "ESCAL: un poste opérateur Grafcet", APII, 1987, 21.

[**PLV87**] Allen Bradley Servotronics, "Documentation sur le langage PLV", 1987.

[**PRU87**] F.Prunet et al, "Méthodologie et implantation automatique de commande d'automatisme à l'aide de la chaîne Piastre", APII, 1987, 21.

[**PUM85**] Unimation, "Puma Mark III- Val II and Val plus robot, Equipment manual", 1985.

[**PUM86**] Unimation, "Programming manual, User's Guide to VAL II, Version 2.0", december 1988.

[**SAH86**] A.Sahraoui, M.Courvoisier, R.Valette, "Some considérations on minitoring systems", IEEE int conf IECON-86, Milwaukee, Ws, Septembre 86.

[**SAH87**] A.Sahraoui, "Contribution à la commande et à la surveillance d'ateliers flexibles", Thèse de doctorat de l'université Paul Sabatier de Toulouse, Octobre, 1987.

[**SAL88**] Y.Sallez, "Conception d'un système de programmation hors ligne de cellules robotisées intégrant le langage ADA", Thèse de doctorat de l'U.V.H.C, Avril 1988.

[SIM80] H.Simon, "Le nouveau management. La décision par les ordinateurs", Ed Economica, Paris, 1980.

[SZY88] J.Szymanski, "Un exemple de système de conception des applications temps-réel", Rapport Réflexion sur le temps-réel, CNRS, 1988.

[TEM88] Rapport du groupe de réflexion sur le temps-réel, "Le temps-réel", CNRS, Juin 1988.

[THO80] J.P.Thomesse, "SYGARE: une structuration pour la conception d'applications en temps-réel et réparties", Thèse de doctorat Es-sciences physiques, INPL Nancy, 1980.

[TSC90] D.Tschirhart, "Commande en temps-réel, conception et mise en oeuvre d'un exécutif temps-réel", Ed Dunod, 1990.

[VAL87] Unimation, "Cours de robotique: Val II et Val +", 1987.

[VOL84] R.A.Volz, T.N.Mudge, D.A.Gal, "Using ADA as a programming language for robot-based manufacturing cells", IEEE trans on sys end cyb, Vol 14, 1984.

[VOL87] R.A.Volz et al, "Translation and execution of distributed ADA programs", IEEE trans soft eng, 1987.

[WAR85] P.T.Ward, S.J.Mellor, "Structured developement for real-time systems", Yourdon press, 1985.

[WAR87] P.T.Ward, S.J.Mellor, "Structured developement for real-time systems", Yourdon press, 1987.

[WAL84] C.Walter, "Control software specification and design: an overview", IEEE computer, Vol 17, N 2, february 1984.

[YOU75] E.Yourdon, L.L.Constantine, "Structured design: fundamentals of a discipline of computer program an systems design", Prentice-hall, 1975.

[ZAH80] J.Zahd, "machines séquentielles, traité d'électricité", Vol 11, Ed Georgi, 1980.

TABLE

DES MATIERES

TABLE DES MATIERES