T H E S E

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES-ARTOIS

pour obtenir le titre de

D O C T E U R

Spécialité : AUTOMATIQUE

par

BAUDEL Brigitte

GENIE LOGICIEL INDUSTRIEL : CONTRIBUTION A LA DEFINITION ET A LA MISE EN SERVICE D'UN POSTE D'AUTOMATISATION ET D'INSTRUMENTATION.

Thèse soutenue le 29 Avril 1986 devant la Commission d'Examen :

Membres du Jury :

P. VIDAL	Président et Rapporteur
S. THELLIEZ	Rapporteur
J.M. TOULOTTE	Directeur de Recherche
J. DEFRENNE	Examinateur
P. ROUAULT	Invité
T. DELABRE	Invité.

INTRODUCTION GENERALE

Le travail que nous présentons dans ce mémoire a été effectué au Centre d'Automatique de l'Université des Sciences et Techniques de Lille Flandres-Artois en collaboration étroite avec Rhône Poulenc Ingenierie et Sinorg Industrie. Il porte sur la définition et les principes de mise en oeuvre d'un poste d'Automatisation et d'Instrumentation pour la conception de systèmes de commande processus par lots.

La fabrication de logiciels d'automatisation coûte de plus en plus cher pour des résultats peu sûrs et souvent mal documentés. La mise au point et la maintenance en sont souvent très délicates. Le poste de travail envisagé prend en charge les parties les plus pénibles et sujettes à erreur comme le codage et la documentation et facilite par ailleurs la spécification.

Ce mémoire comprend quatre parties. Dans un premier chapitre, nous donnons toutes les définitions nécessaires à la compréhension de la notion de Génie Logiciel Industriel. La spécification y tient une place essentielle, car c'est elle qui conditionne la qualité de toute la chaîne de rpoduction de logiciel.

Le deuxième chapitre présente les choix effectués en vue de la définition d'un poste d'Automatisation et d'Instrumentation des processus par lots. Un langage hiérarchisé éditeur syntaxique est ainsi défini.

La troisième partie place le concepteur d'automatismes devant son poste de travail et montre sur un exemple comment il est possible d'utiliser les diverses bases de données technologiques. Une représentation graphique de la description est aussi introduite.

Enfin dans le dernier chapitre il est envisagé le passage de la spécification à l'implantation. Les méthodologies sont faites d'une part pour des machines à déroulement cyclique comme les automates programmables et d'autre part en programmation universelle entrelacée.

CHAPITRE I

POURQUOI, OU, COMMENT ET QUE FAUT-IL SPECIFIER ?

LE POSTE D'AUTOMATISATION ET D'INSTRUMENTATION (PAI)

S O M M A I R E

1.1. INTRODUCTION : POURQUOI SPECIFIER ?

L'avènement des systèmes programmés, et l'obligation d'aborder des systèmes à automatiser de plus en plus vastes, font que les concepteurs ont ressenti le besoin de méthodes d'analyse et d'outils de spécification.

En informatique classique, des principes de construction de programmes ont été définis de façon à obtenir des produits vivants.

Une nouvelle discipline est née, le Génie Logiciel (J.O.19 février 1984), ensemble des activités de conception et de mise en oeuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi.

Le cycle dans la vie d'un logiciel comprend trois étapes, itérées de nombreuses fois :

- l'élaboration,
- l'utilisation,
- la maintenance qui est (J.O. 19 février 1984) :

  . l'ensemble des actions tendant à prévenir ou à corriger les dégradations d'un matériel afin de maintenir ou de rétablir sa conformité aux spécifications.

Des études récentes ont montré que les coûts de maintenance représentent deux à quatre fois le coût du développement. La seule solution est de fiabiliser les logiciels en éliminant les erreurs de définition entraînant des modifications prévues trop tardivement dans l'élaboration du projet.

Celle-ci comprend deux phases :

- l'analyse et la conception,
- l'implémentation.

Après avoir défini le problème et l'avoir décomposé en sous-ensembles moins complexes, il devient possible de spécifier une solution en utilisant un outil formel. Cette solution doit alors être validée d'une part dans sa formalisation, d'autre part dans son adéquation en cahier des charges initial. Les grands principes de structuration et de hiérarchisation facilitent beaucoup cette première phase de l'élaboration d'un produit et lui confèreront une clarté irremplaçable lors de l'implémentation. Cette partie ne se traite pas en un seul passage et de nombreuses itérations sont parfois nécessaires pour aboutir à une solution satisfaisante.

Pour mettre en oeuvre cette solution, il convient de commencer par procéder au choix d'un matériel ou d'un langage. Les imperfections temporelles des dispositifs rendent obligatoires le passage à un modèle de réalisation et le choix d'une méthodologie d'implantation. Le codage en découle immédiatement. Puis vient la phase de déverminage et de test pour vérifier la conformité tant fonctionnelle que temporelle au cahier des charges. Cette création de logiciel s'accompagne à tous les instants d'une documentation, capitale pour permettre les modifications et la maintenance.

Cette systématisation de la mise en oeuvre est la seule démarche permettant l'abord de problèmes de grande dimension, à un coût raisonnable, en garantissant une bonne fiabilité du logiciel et une longue durée de vie. Cette démarche devrait être la même en informatique industrielle. On aurait alors le Génie Logiciel Industriel qui devrait tenir compte de la spécificité des problèmes d'automatisation, aussi bien au niveau de la description que de la conduite en temps réel.

Le produit idéal qui en résulte est conforme aux besoins réels, facile à exploiter et à maintenir ; pour cela, il doit être aisément communicable, cohérent, complet et d'une implantation bien organisée.

1.2. OU SPECIFIER ? Le Poste d'Automatisation et d'Instrumentation (PAI)

Dans l'élaboration d'un produit, les coûts se répartissent de la façon suivante : (LAM)

- 45 % en analyse et conception,
- 20 % en codage,
- 35 % en mise en oeuvre.

Il est possible de minimiser le coût de l'analyse et de la conception en introduisant un outil méthodologique utilisant un modèle formel pour faciliter la spécification.

L'aide à la documentation, les connexions à la CFAO et à la nomenclature de la partie opérative permettent également de diminuer le temps de travail, tout en améliorant la qualité.

La spécification peut être vérifiée, éventuellement, automatiquement à différents niveaux :

- syntaxique,
- formel,,
- simulation.

Le temps de codage peut être diminué par la production automatique de logiciels avec documentation, quels que soient le langage ou la machine cible.

Un système informatisé, chargé de remplir ces différentes missions, pourrait constituer le Poste d'Automatisation et d'Instrumentation (PAI), système autonome, connectable soit à un réseau serveur, soit à un réseau local d'entreprise, pour être mis en relation avec différentes bases de données.

1.2.1. Définition du PAI

Un système étant constitué d'un ensemble d'objets technologiques, tels vannes, moteurs,.., dit Partie Opérative (PO), contrôlé par un dispositif de commande (PC), via les éléments d'interface, créer un PAI permet, sur une même machine, de faire le lien entre ces objets et le cahier des charges de la commande. Ceci se fait en donnant à l'utilisateur tous les moyens informatiques actuels pour l'aide à l'édition, à la recherche documentaire, à la production automatique de programmes.

Le PAI devient ainsi une des composantes d'un système complet de conception assistée par ordinateur, intégrant la conception des installations et la gestion de la production - figure 1.1 -

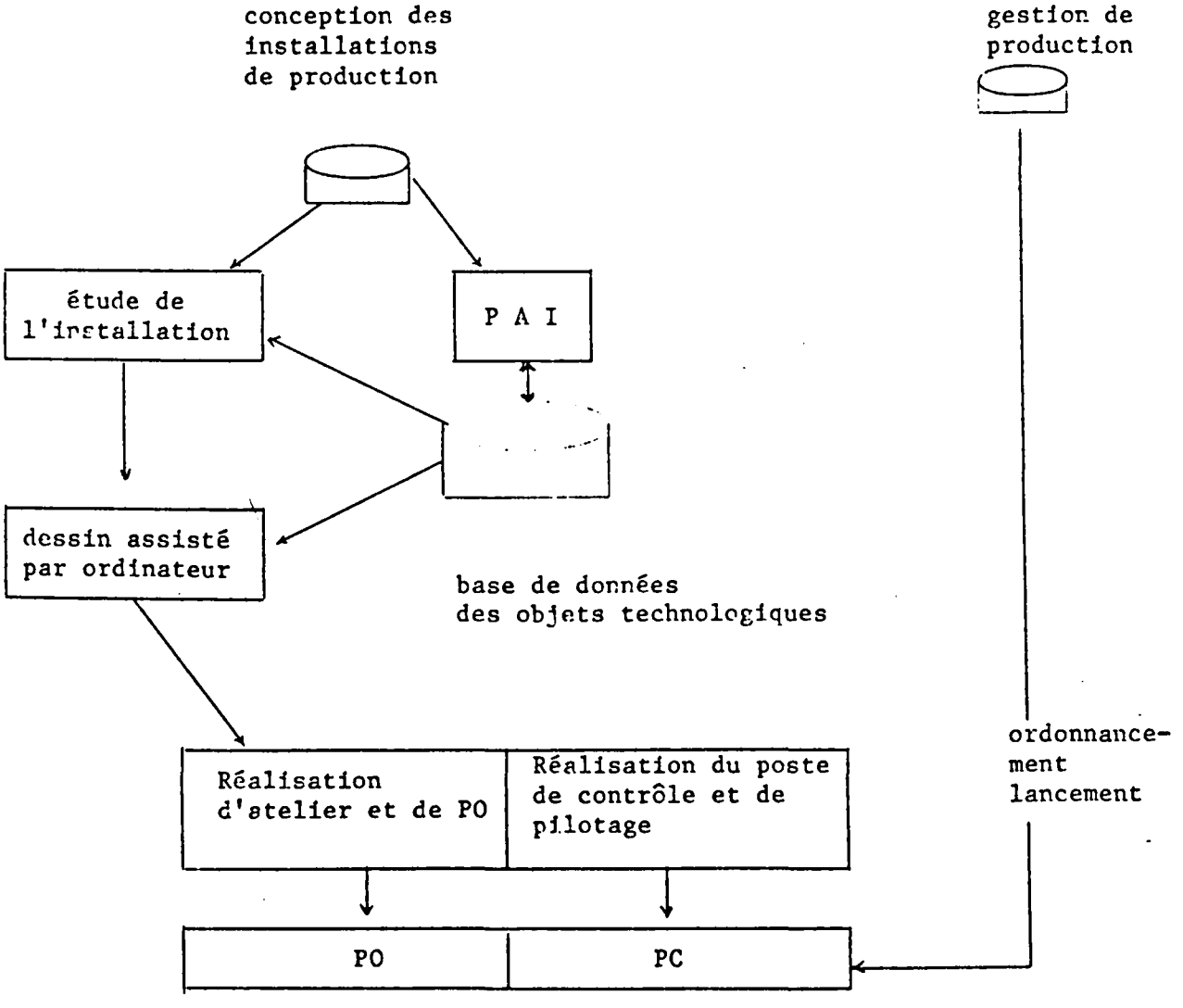


FIGURE 1.1.

La partie commande PC comprend le poste de pilotage, un réseau local de commande, des automates programmables et/ou des calculateurs et/ou des régulateurs, des éléments d'interface vers la PO.

Le PAI doit être un système modulaire et ouvert permettant l'introduction de nouveaux logiciels et l'évolution des anciens pour convenir à la fois :

à * un travail hors ligne lors de :

- la spécification,
- la validation des spécifications,
- la simulation de l'ensemble partie commande-partie opérative,
- l'aide au choix technologique,
- la programmation automatique,
- la production de documents,

à * une utilisation en ligne pour :

- l'aide à la mise en route par visualisation de l'état de PC et PO,
- le test en ligne des capteurs et des actionneurs,
- la maintenance assistée par ordinateurs.

1.2.2. Spécifier (BAB) - (BOR).

Le but de la spécification est d'obtenir un modèle de la partie commande. Ceci se fait en utilisant des outils formels de représentation des cahiers des charges, et il est souhaitable d'y adjoindre des contraintes méthodologiques. On ne doit pas introduire de discontinuité entre la nomenclature et la description de la partie commande, celle-ci pouvant être vue d'un niveau supérieur comme un objet technologique de même type que les autres.

Un soin particulier doit être donné à cette partie car elle conditionne la qualité du PAI. Le formalisme et la rigueur indispensables ne doivent pas empêcher un dialogue facile avec un utilisateur non informaticien, tant sur le plan de l'adéquation du modèle au système modélisé que sur l'ergonomie de l'introduction.

Une grande partie de ce mémoire est consacré à la spécification.

1.2.3. Validation de la spécification

La validation de la spécification commence dès l'introduction du modèle par des vérifications syntaxiques, ou mieux en utilisant un éditeur syntaxique. Toutefois, tout n'est pas vérifiable à ce niveau et il convient d'adjoindre une phase d'analyse des éléments dépendant du contexte. Ceci se fait dans un deuxième temps dit de pré-traduction.

Un modèle bien écrit n'est pas forcément valable, et il est possible, si le formalisme du modèle le permet, d'introduire des tests de validation formels. On peut se rapprocher par exemple de ce qui existe pour les réseaux de Pétri.

Malheureusement, ces tests n'apportent pas une garantie totale du bon fonctionnement du modèle, car ils sont essentiellement structurels et ne tiennent pas compte de l'interprétation associée à la structure. C'est pourquoi on fait souvent appel à la simulation.

1.2.4. Simulation

La simulation est un outil fondamental dans la conception et l'évaluation des performances d'un système, y compris dans la phase d'exploitation.

On peut y distinguer quatre niveaux :

- étude de la commande des sous-parties opératives,
- validation de l'adéquation entre le modèle et le cahier des charges,
- étude des stratégies de commande et de gestion,
- aide à l'apprentissage de la conduite.

On peut envisager une gradation dans l'usage de la simulation, celle-ci étant d'abord utilisée hors ligne à un plan macroscopique pour une évaluation rapide du fonctionnement. Le modèle est alors simplifié. Le passage au plan macroscopique peut alors être fait de façon à concevoir une conduite conforme au cahier des charges.

L'étage suivante est celle du couplage du système physique de commande avec une représentation de la partie opérative

Enfin, en ligne ou hors ligne, une simulation peut être admise pour l'aide à la conduite ou la surveillance des capteurs et des actionneurs.

Ceci implique plusieurs notions :

- le modèle de description doit permettre des spécifications exécutables et des vérifications syntaxiques et formelles ;

- il est fondamental de pouvoir associer à chacun des objets technologiques un comportement fonctionnel et de pouvoir décrire la partie opérative dans son ensemble sans discontinuité avec la description de la partie commande, la frontière entre les deux n'étant pas toujours facile à définir clairement ;

- l'ensemble doit être utilisable sur des petits systèmes informatiques par des non-informaticiens.

Cet aspect simulation sort du cadre de notre mémoire. Toutefois, le modèle adopté pour la spécification doit en tenir compte.

1.2.5. Production de logiciels (LOG)

Produire manuellement du logiciel est une opération coûteuse qui nécessite de nombreuses itérations. Les logiciels résultats sont généralement peu fiables, peu modifiables, souvent mal documentés, et avec des dépassements de délais.

La solution idéale est de pouvoir fournir à une machine une description précise et cohérente du problème en langage naturel ou semi-naturel. La machine génère alors un programme d'application parfaitement documenté. Toutes les approches pour parvenir à ce but ont en commun l'usage d'une quantité considérable d'informations, qui peuvent être organisées et manipulées suivant quatre principes différents :

1.2.5.1. Les progiciels individualisables

Cette procédure est certainement la plus facile à utiliser, mais elle ne permet d'aborder que des logiciels limités à des domaines très spécifiques.

On cherche à obtenir une description à partir d'un questionnaire à choix multiple dans un environnement interactif.

Le résultat est une base de données de l'application, approximation grossière d'une banque de connaissances.

L'expertise est en effet essentielle dans la conception d'un questionnaire cohérent et complet.

Cette description permet une génération par sélection des modules pré-programmés paramétrés, puis par substitution des valeurs de paramètres à l'aide d'un macrogénérateur, et enfin par assemblage.

On adjoint à tout ceci une production automatique de documentation.

Cette procédure est peu flexible et nécessite d'une part la définition du dictionnaire et, d'autre part, la création d'une vaste bibliothèque de modules. La maintenance du programme généré n'est pas aisée.

1.2.5.2. Les langages de très haut niveau.

On cherche à obtenir un outil de description le plus proche possible de la structure de l'application. On ne décrit pas forcément de façon ordonnée et procédurale. On peut définir des traitements sur des familles de données ou d'objets, et on fait généralement usage de fichiers à accès associatif.

L'expression est donc celle des concepts fondamentaux de l'application; le générateur est alors un compilateur. L'utilisateur dispose d'une grande liberté, mais il lui faut apprendre un langage. La génération est très efficace et les programmes faciles à maintenir.

1.2.5.3. La programmation évolutive.

L'idée est ici de partir d'un ensemble programme de base qui réalise les fonctions standards du domaine d'application. On crée alors des modifications par application d'un métaprogramme qui compose un certain nombre de règles de transformation.

On doit pouvoir disposer des effets du programme généré en suivant son exécution sur des jeux d'essais, afin d'affiner les modifications.

Le seul point fort de cette méthodes est évidemment la possibilité de maintenance du programme généré.

1.2.5.4. Les systèmes à banque de connaissance.

C'est certainement la procédure la plus prometteuse, mais qui n'est pas encore dans une phase industrielle. Elle nécessite une énorme quantité d'informations, ce qui la rend abordable sur de très puissantes machines.

Le traitement complet de la génération se fait en quatre parties.

Tout d'abord, il y a la phase d'acquisition du problème via un dialogue dans un sous-ensemble restreint de la langue naturelle ou dans un langage semi-naturel, après introduction du vocabulaire de l'application, et en usant d'un ensemble limité de structure syntaxique.

Il redevient alors possible de transformer cette solution en un modèle de haut niveau, qu'on vérifie de façon interactive dans la troisième phase. Enfin, on effectue la programmation automatique.

Ces systèmes experts offrent une très grande liberté à l'utilisateur. Ils permettent des vérifications de certains choix, des interrogations dynamiques, une structuration par niveaux.

La difficulté réside d'abord dans la puissance de traitement et la taille de la mémoire, ensuite et surtout dans la qualité de l'expertise.

1.3. COMMENT SPECIFIER : LES CRITERES D'UNE BONNE SPECIFICATION.

1.3.1. Outil de description

Un des éléments essentiels d'une bonne spécification est certainement de disposer d'un bon outil de description des cahiers des charges munis de bases théoriques saines, de façon à éviter toute ambiguïté d'interprétation. Un même modèle doit avoir la même spécification pour différents lecteurs.

Pour cela, il doit reposer sur des hypothèses temporelles très strictes, entrer dans un cadre linguistique bien délimité et, si possible, standardisé. Ce dernier point n'est plus nécessaire si on arrive à utiliser des techniques d'analyse du langage naturel, en disposant de machines suffisamment puissantes.

1.3.2. Méthodologie de description

Un bon outil ne peut être exploité de façon optimale que dans le cadre d'une méthodologie qui aide à décomposer le problème en sous-problèmes plus abordables

Des règles méthodologiques sont nécessaires afin d'obtenir une construction modulaire cohérente. La spécification est un processus d'affinement et d'élimination des erreurs. C'est pourquoi, parmi les méthodes les plus utilisées, on trouve l'analyse hiérarchique et structurée.

L'approche experte en est encore à ses débuts, mais semble très prometteuse.

1.3.3. La forme extérieure

La majorité des descriptions est basée sur une structure syntaxique rigoureuse, allant essentiellement jusqu'à obliger l'utilisateur à une introduction de la description par l'intermédiaire d'un éditeur syntaxique.

D'autre part, il est intéressant de connaître également la sémantique, c'est-à-dire le rôle et la signification de chacun des éléments décrits.

L'utilisation de représentations multiples d'un même problème peut aider la compréhension et faciliter l'élimination des incohérences et des omissions.

Ceci permet de s'adapter à une plus grande variété d'utilisateurs.

1.3.4. Généralité ou spécificité

Certains outils ou langages de spécification se veulent universels. Ils sont donc utilisables pour des problèmes touchant à des domaines divers.

Malheureusement, ils ne sont exploitables que par un personnel très spécialisé, ayant une grande connaissance de l'informatique.

A l'opposé, d'autres ne concernent qu'un secteur restreint d'application et sont abordables par les utilisateurs eux-mêmes, sans formation informatique préalable.

1.3.5. Statique et dynamique

La grande majorité des langages de programmation ou de spécification ne représente que l'aspect dynamique des traitements, c'est-à-dire comment sont manipulées les diverses données ou entités.

Mais il est par ailleurs très important de pouvoir décrire les données elles-mêmes avec leur structure, leur type, leur domaine de validité, la sémantique associée, leur valeur par défaut...

1.3.6. Apprentissage et emploi

L'obtention de la grande quantité d'informations nécessaire à la génération du logiciel ne doit pas se faire en exigeant de l'utilisateur un effort considérable ni sur le plan conceptuel, ni sur le plan du maniement.

Pour minimiser cet effort, l'outil méthodes doit être proche des habitudes de l'utilisateur. Par ailleurs, une attention toute particulière doit porter sur l'ergonomie du dialogue lors de l'introduction.

L'éditeur doit optimiser le nombre de mouvements (appui clavier par exemple), par usage :

- de menus arborescents sélectifs tenant compte, éventuellement, de contraintes syntaxiques,
- de tableaux préformatés.

L'utilisateur doit toujours savoir que faire et, en cas de problème, il doit pouvoir consulter un programme d'aide.

1.3.7. Une spécification exploitable

Les choix à faire, pour atteindre la bonne spécification, doivent tenir compte du résultat souhaité.

Une telle description n'est pas une fin en soi ; elle doit être exploitable.

L'objectif reste de produire du logiciel par des dispositifs automatiques. Il faut donc tout d'abord permettre l'accès à une simulation. On peut dire qu'une spécification doit être exécutable.

Il ne faut pas par ailleurs que la structure de la description pose des difficultés lors de la tâche de génération, en particulier en abusant de contraintes méthodologiques trop restrictives.

Un compromis est donc nécessaire pour être applicable au plus grand nombre de machines cibles .

Enfin, la procédure de description doit permettre une maintenance facile du produit logiciel obtenu.

1.4. QUE SPECIFIER ?

Il y a fréquemment des confusions lorsqu'on parle de description des cahiers des charges. En effet, il est important de bien faire la distinction entre description de problème et description d'une solution du problème. La difficulté vient du fait qu'un cahier des charges contient des éléments de problème et des éléments d'analyse préfigurant l'obtention d'un type de solution. Un degré supplémentaire est atteint lorsque la frontière partie commande-partie opérative n'est pas constante tout au long de la procédure d'obtention d'une solution. Certains choix technologiques peuvent en effet remettre en cause la position de cette frontière.

Pour montrer la différence entre décrire un problème et une solution à ce problème, envisageons le petit exemple suivant qui concerne l'optimisation dans le temps du nombre d'opérations automatiques dans une installation de dosage-malaxage :

Un malaxeur M reçoit des produits A et B pesés par la bascule C et des briquettes solubles, amenées une par une par un tapis d'amenage. L'automatisme permet de réaliser un mélange comportant les trois produits (schéma 1.4.1.).

Le cycle de fonctionnement est le suivant :

- l'action sur le bouton départ cycle "dcy" provoque le pesage et l'amenage des produits, éventuellement en simultanéité,

  . pesage du produit A jusqu'au repère a, puis pesage du produite B jusqu'au repère b, suivis de la vidange de la bascule C dans le malaxeur,

  contrainte : le mélange A + B, une fois fait, ne reste stable que pendant un temps TL 1. Le début de la vidange de la bascule C doit donc se produire avant cette limite

  . amenage de N briquettes,

  contrainte : le rempliscage des briquettes ne peut pas se faire avant l'amenage du mélange A + B. Il peut se faire en même temps.

- Le cycle se poursuit par la rotation du malaxeur et par son pivotement, la rotation du malaxeur étant maintenue pendant la vidange. Le début du pivotement ne peut se faire qu'après un intervalle de temps t à partir du début de rotation du malaxeur TL 2 $\leq \Delta t \leq$ TL 3.

- Le cycle se termine par le retour du malaxeur en Po.

1.4.1. Identification des opérations et sous-opérations.

Identifier des opérations et des sous-opérations revient à effectuer en fait une décomposition en sous-parties opératives correspondant généralement à des entités matérielles ou géographiques distinctes ; c'est déjà un pas vers une solution particulière.

Dans notre exemple, on peut dégager les opérations suivantes :

T 1 : Gestion des stocks :

Il s'agit d'une opération relativement indépendante. Il importe seulement d'avoir le stock minimum pour démarrer un cycle.

On peut aussi décomposer T 1 en :

T 11 stockage de A,
T 12 stockage de B,
T 13 stockage des briquettes.

T 2 : Préparation des liquides, qui comprend :

T 21 amenage de A,
T 22 amenage de B,
T 23 dosage de A,
T 24 dosage de B,
T 25 stockage de A + B.

T 3 : Remplissage du malaxeur, avec :

T 31 amenage de A + B,
T 32 amenage des briquettes,
T 33 comptage des briquettes.

T 4 : Malaxage décomposable en :

T 41 malaxage
(rotation)
T 42 évacuation.

Souvent, on aboutit à des décompositions matérielles trop fines vis-à-vis de certaines fonctionnalités. On est alors obligé de faire quelques regroupements. Ainsi, on aura :

T'2 : regroupement de T 21 et T 23,
T"2 : regroupement de T 22 et T 24,
T'3 : regroupement de T 32 et T 33.

On peut alors résumer l'opération T de dosage-malaxage par les ensembles suivants :

T : (T 1, T 2, T 3, T 4),

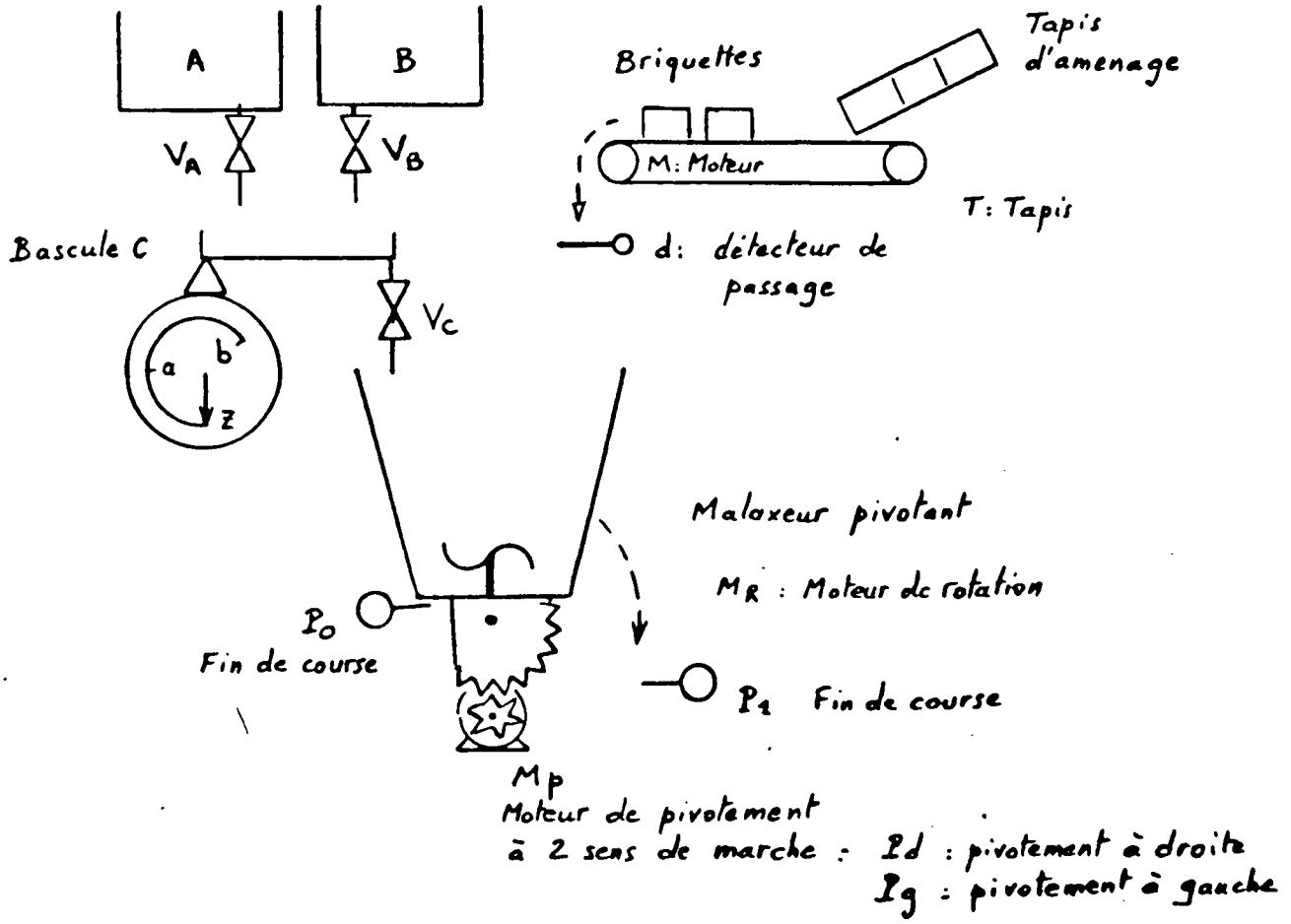
T 1 : (T 11, T 12, T 13),

T 2 : (T'2, T"2, T 25)
T'2 = (T 21, T 23)
T"2 = (T 22, T 24)

T 3 : (T 31, T'3)
T'3 = (T 32, T 33)

T 4 : (T41, T 42)

SCHEMA 1.4.1.


A
B
V_A
V_B
Briquettes
Tapis d'amenage
M: Moteur
T: Tapis
Bascule C
d: détecteur de passage
V_C
a
b
Z
Malaxeur pivotant
M_R : Moteur de rotation
P_0
Fin de course
P_1 Fin de course
M_P
Moteur de pivotement
à 2 sens de marche : Pd : pivotement à droite
Pg : pivotement à gauche

1.4.2. Etablissement des contraintes temporelles entre opérations

T 2 : débute si T 1 correcte et T 4 terminée et condition de départ,

T"2 : débute si T'2 terminée,

T 3 : débute si T 2 terminée,

T'3 : peut débuter dès que T 31 débute,

T 2 : est terminée si T 25 est terminée,

T 3 : est terminée si T'3 et T 31 terminées,

T 31 : débute au plus tard TL 1 après la fin de T"2,
ou durée de T 25 inférieure à TL 1,

T 41 : se termine en même temps que T 42,

T 42 : débute après début de T 41 + TL 2
et avant début de T 41 + TL 3,

T 4 : se termine en même temps que T 41 et T 42,

1.4.3. Recherche d'une solution optimale tenant compte des contraintes

L'obtention d'une solution optimale nécessite la connaissance de la durée de chaque opération. Dans le cas de durée paramétrée, on recherche :

. soit une solution fixe pour l'étendue des paramètres, en sachant que, si elle est plus facile à réaliser, elle peut être sous optimale dans certaines configurations de paramètres,

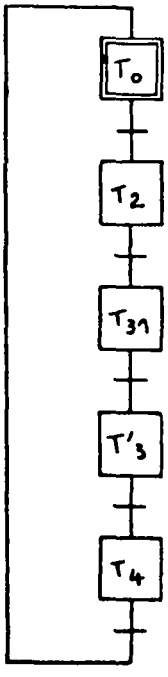
. soit une solution dépendante des valeurs des paramètres, plus délicate à mettre en oeuvre, mais toujours optimale.

Dans notre exemple, compte tenu des durées, on peut envisager plusieurs solutions pour résoudre les contraintes précédentes. Elles sont représentées, dans le formalisme du Grafcet,sur les schémas 4.2., uniquement pour l'ensemble (T 2, T 3, T 43), T 1 se déroulant indépendamment. On a adjoint une opération T 0 d'attente de cycle (de durée nulle en cycle continu normal).

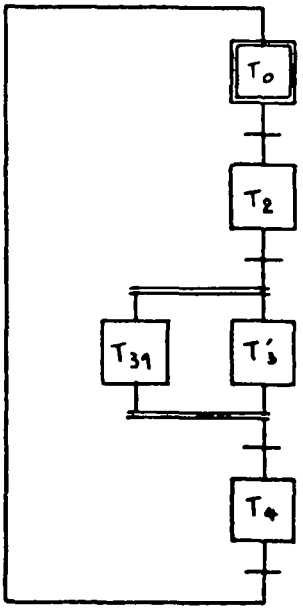
La première solution purement séquentielle entraîne une perte de temps importante en négligeant le parallélisme possible de T 31 et T'3 qui apparaît sur la deuxième.

Il est intéressant, dans la troisième solution, de regarder comment, par des éléments de synchronisation, on a pu tenir compte des contraintes pour améliorer ce problème de temps masqué.

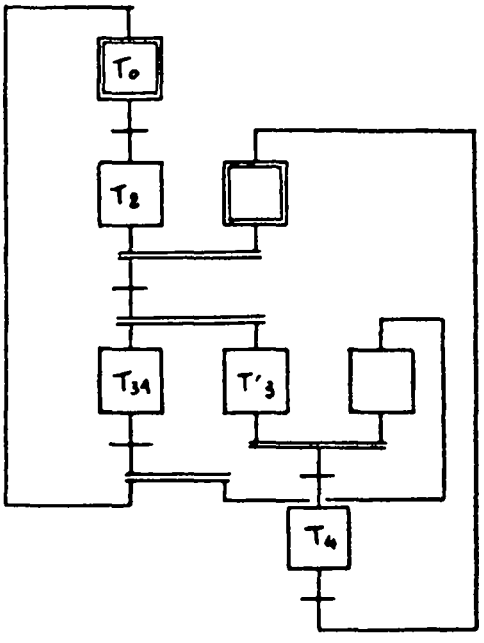
SCHEMA 1.4.2



solution 1



solution 2



solution 3

1.4.4. Vers une expression de la relation entre l'action et le temps

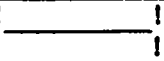
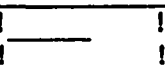
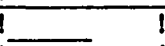
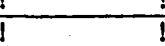
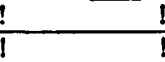
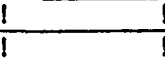
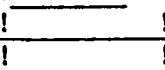
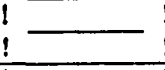
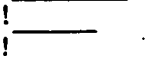
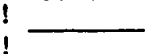
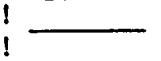
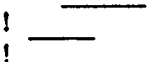
Ce que nous venons essayer d'exprimer est la relation temporelle entre les opérations.

Une opération Ti occupe un intervalle de temps (noté : également Ti) qui débute à la date DTi et se termine à la date FTi. On stipule par convention que les intervalles sont ouverts à gauche et fermés à droite.

Essayons tout d'abord de regarder les positions relatives de deux opérations T1 et T2.

Nous disons que DT1 < DT2 si l'opération T1 débute avant l'opération T2 (respectivement pour FT1 < FT2 et finit).

On obtient le tableau ci-après :

DT1/DT2	FT1/FT2	DT1/FT2	DT2/FT1	T1/T2	prédicat		noté
<	>	<	<		pendant	(T1,T2)	p (T1,T2)
<	<	<	<		chevauche	(T1,T2)	c (T1,T2)
<	<	<	=		succède	(T1,T2)	s (T1,T2)
<	<	<	>		après	(T1,T2)	a (T1,T2)
<	=	<	<		finit	(T1,T2)	f (T1,T2)
=	<	<	<		débute	(T1,T2)	d (T1,T2)
=	=	<	<		égal	(T1,T2	e (T1,T2)
=	>	<	<		débute	(T2,T1)	di(T1,T2)
>	<	<	<		pendant	(T2,T1)	pi(T1,T2)
>	=	<	<		finit	(T2,T1)	fi(T1,T2)
>	>	<	<		chevauche	(T2,T1)	ci(T1,T2)
>	>	=	<		succède	(T2,T1	si(T1,T2)
>	>	>	<		après	(T2,T1)	ai(T1,T2)

En introduisant les notations suivantes :

& conjonction, V disjonction, $\sim$ négation, => implication

<=> équivalence, $\forall$ quantificateur universel et, éventuellement,

$\exists$ quantificateur existentiel, $\exists$! existence d'un objet unique.

Il est possible d'introduire un ensemble de règles qui définissent la conduite de ces prédicats, comme, par exemple :

s (T1,T2) & p (T2,T3) => a (T1,T3)

s (T1,T2) & pi (T2,T3) => (pi (T1,T3) v d (T1,T3) v c (T1,T3)) etc.

On peut aussi induire de nouveaux prédicats, comme par exemple :

INCLUS (T1,T2) <=> (d (T1,T2) v pi (T1,T2) v fi (T1,T2))

Si maintenant on revient sur la spécification des contraintes temporelles du paragraphe 1.4.2, on s'exprime essentiellement en termes de début et de fin opération. Par exemple :

T"2 débute si T'2 est terminée (tout de suite ou après)

devient : DT"2 $\geq$ FT'2, ce qui, en termes d'intervalles,

s'écrit : s (T'2, T"2) v a (T'2, T"2)

T'3 peut débuter dès que T31 débute donne

(DT'3 = DT31) v (DT'3 > DT31) ce qui, compte tenu de l'absence

d'informations sur FT'3 et FT31, donne, à partir du tableau précédent :

(d (T'3,T31) v e (T'3,T31) v di (T'3,T31) v pi (T'3,T31) v fi (T'3,T31) v

... Ci (T'3,T31) v si (T'3,T31) v ai (T'3,T31)) ou encore :

$\sim$ (p (T'3,T31) & c (T'3, T31) & s (T'3,T31) & a (T'3, T31) & f (T'3, T31))

Ces expressions se simplifient si on connaît les durées relatives des opérations.

T3 est terminé si T'3 et T31 sont terminées devient

FT3 = max (FT'3, FT31)

T31 débute au plus tard TL1 après la fin de T"2 donne

DT31 < FT"2 + TL1

Ces relations, mêmes complexes, restent insuffisantes pour exprimer tous les comportements temporels ; en effet,nous n'avons pas défini que des relations entre intervalles ou entre dates de début et de fin d'opération.

Il convient de pouvoir introduire la notion de temps absolu en référence à l'horloge universelle, par exemple :

DTi = 8 h. 25 mn. 30 s. GMT

Les deux notions temporelles ne peuvent être mises en relation que lorsqu'on a placé l'ensemble des opérations relatives dans le repère temps.

Une autre notion importante parmi les contraintes temporelles est celle de la répétition d'une opération, soit un certain nombre de fois, soit relativement à un prédicat P. On écrira :

$\{Ti\}^n$ avec $D\{ti\}^n$ et $F\{Ti\}^n$ pour l'opération itérée n fois

Ti/P répéter Ti tant que le prédicat P est vrai.

On aurait $D\{Ti/P\}$ et $F\{Fi/P\}$ les début et fin de l'opération répétée. Par ailleurs, la répétition peut aussi se faire avec des intervalles constants (par exemple toutes les 10 minutes) ou à des dates particulières (par exemple tous les mardis 8h. GMT). On écrira alors :

DTi = {10 mn} ou DTi = {mardi 8 h. GMT}

./..

Au cours d'une opération, a lieu une activité qui occupe tout ou partie de l'intervalle en fonction des conditions spécifiques. ACT (ac,T) est un prédicat vrai si l'activité ac se produit durant l'intervalle T.

Si une activité se produit sur un intervalle T, elle se produit sur tout sous-intervalle.

ACT (ac,T2) <=> ($\forall$ T1 . INCLUS (T1,T2) <=> ACT (ac,T1))

On peut introduire des fonctions comme ET, OU, XOU, PAS :

ACT (ET (ac1, ac2), T) <=> ACT (ac1,T) & ACT (ac2, T)

ACT (PAS (ac1), T) <=> ($\forall$ T2 INCLUS (T2,T) => $\sim$ ACT (ac1,T2))

ACT (PAS (PAS (ac)), T) <=> ACT (ac,T)

ACT (OU (ac1,ac2), T) <=> ACT (PAS (ET (PAS (ac1), PAS (ac2))),T)

Ces fonctions peuvent faire intervenir des prédicats tels P1, P2 :

ACT (ou (ac1/P1, ac2/P2),T) <=> P1 & ACT (ac1,T) v P2 & ACT (ac2 T)

Remarquons que, dans beaucoup de cas, P1 & P2 <=> faux et même P1 <=> $\sim$ P2. Par ailleurs, on préfère souvent utiliser la fonction XOU, ou exclusif, plutôt que OU.

ACT (XOU (ac1/P1, ac2/P2), T) <=> P1 & ACT (ac1, T) & $\sim$ ACT (ac2, T)....

.... v P2 & ACT (ac2,T) & $\sim$ ACT (ac1, T)

On pourrait ainsi développer tout un ensemble de règles de ce type.

L'activité porte sur des objets, organisés en classes. On peut définir un objet en tant que tel (cas d'une classe à un seul objet) ou relativement à son appartenance à une classe.

- Pour définir une classe, on indique son nom et la liste de ses attributs avec, pour chacun, son nom et son type.

```
CLASSE  nom de classe

  ATTRIBUTS

        atcl1  :  type 1
              .
              .
              .
        atcln  :  type n
```

Les types sont des classes particulières énumératives ou descriptives sans attribut correspondant à la définition du langage ADA ou des classes avec attributs.

La définition d'un objet peut se faire comme pour une classe :

```
OBJET  nom d'objet

  ATTRIBUTS

        atobl  :  type 1
              .
              .
              .
        atobn  :  type n
```

ou par référence à une classe

```
OBJET  nomd'objet MEMBREDE  nom de classe
```

Dans ce cas le nom de l'objet est inscrit dans la classe et l'objet prend tous les attributs de la classe

```
atcl1  nom d'objet
     .
     .
     .
atcln nom d'objet
```

deux objets ou deux classes peuvent être mis en relation

nomd'objet RELATION nomd'objet

ou nomdeclasse RELATION nomdeclasse

exemple : commandesvannes influencent capateursvannes

Par ailleurs, on peut définir la notion de sous-classes. Les attributs d'une classe peuvent être transmis aux sous-classes. Pour définir une sous-classe, on écrit :

nomsous-classe EST-DANS nomdeclasse

Une sous-classe devient alors équivalente à une classe.

Il importe maintenant de définir le contenu de l'activité ac :

ACTIVITE nomd'activité

constituants

nomobjet1 : nomclasse 1
.
.
.
nomobjetn : nomclassen

définition

a-besoin-de : nomentrée1,....nomentréeq

précondition : nomprédicat Pe

actions-non-répétées : action 1 / P1
.
.
.
action m /Pm

actions-répétées : actionrep1 /Pr1
.
.
.
actionreph /Prh

condition-finale : nomprédicat Pf

affecte : nomsortie1.....nomsortieq

Une activité porte sur les objets dont on définit les noms dans la partie "constituants". Puis, on indique dans la partie définition, d'une part les entrées à examiner et les sorties à affecter. D'autre part, le traitement proprement dit ne commence que si la précondition est vraie.

Les activités non répétées éventuellement conditionnées à un prédicat sont exécutées une seule fois, dès l'obtention de la précondition, tandis que les actions repétées s'exécutent au rythme du cycle du traitement, jusqu'à ce que la condition finale soit vraie.

Tous les prédicats Pe, Pl,...Pm, Prl, ...Prh, Pf sont des expressions logiques complexes faisant intervenir les opérateurs logiques et les opérateurs de comparaison. On pourra être amené à distinguer à ce propos la notion de variation d'un prédicat P (on dira l'événement P si on s'intéresse au fait qu'il devienne vrai) par rapport à la notion de valeur du prédicat (on dira souvent la condition P ou $\sim$ P).

Remarquons que, parfois, la variation se note $\uparrow$ P ou $\downarrow$ P suivant son sens.

1.5. CONCLUSION

Ce chapitre nous a permis d'amener toutes les notions de base concernant la définition du Poste d'Automatisation et d'Instrumentation.

La nécessité d'une bonne spécification ne fait plus aucun doute, mais encore faut-il savoir comment et que spécifier.

Pour une classe de problèmes à traiter, des choix sont à faire pour obtenir un poste de travail intéressant à utiliser et apportant une grande efficacité.

Pour les problèmes que nous avons à résoudre, le chapitre suivant tentera de justifier les choix, qui aboutiront au Poste d'Automatisation et d'Instrumentation EASYMITIS (*).

*) marque déposée CREATI/RHONE-POULENC/SINAC.

CHAPITRE II

SPECIFICATION DES CAHIERS DES CHARGES DANS LE CAS DES PROCESSUS PAR LOT

(BATCH PROCESSING)

S O M M A I R E

2.7.2. Modes de marche et d'arrêt

2.7.2.1. Définitions
2.7.2.2. Notion d'anomalie
2.7.2.3. Détection des anomalies
2.7.2.4. Point d'entrée - Point d'arrêt

2.8. OUTIL DE DESCRIPTION DE LA PARTIE COMMANDE

2.8.1. Opération élémentaire

2.8.2. Opérations structurantes

2.8.2.1. Séquence
2.8.2.2. Sélection
2.8.2.3. Itération

2.8.3. Le parallélisme et ses conséquences

2.8.3.1. Définition et objectifs
2.8.3.2. Les différentes formes de parallélisme

a/ Le parallélisme décrit structurellement
1 - Le parallélisme comportant des opérations implicites.
2 - Le parallélisme décalé

b/ Le parallélisme décrit implicitement.

2.8.3.3. Conséquences du parallélisme

a/ Définition des structures d'échange entre opérations parallèles
1 - Files d'attente FIFO
2 - Boite aux lettres

b/ Le problème des ressources communes
A - Définition
B - L'exclusion mutuelle
C - L'accès

2.8.4. Paramétrage

2.8.5. Description des niveaux supérieurs

2.9. CONCLUSION

2.1. INTRODUCTION.

Le premier chapitre nous a donné tous les éléments correspondant à une bonne spécification. Il nous faut maintenant effectuer les choix de manière à appliquer ces principes dans le cas des processus par lot (Batch processing) typiques des industries chimiques, pharmaceutiques, agro-alimentaires, etc. Le problème qui nous a été soumis est de bâtir la définition d'un Poste d'Automatisation et d'Instrumentation PAI en vue de la génération automatique de logiciels pour le contrôle de ce type de processus, soit par automate programmable, soit par calculateur industriel.

2.2. LES PROCESSUS PAR LOT (BATCH-PROCESSES).

Dans de nombreux systèmes de production, les matières premières passent de poste en poste où elles subissent, à chaque fois, une certaine transformation.

Entre ces postes existent des dispositifs de transport de toute nature : convoyeurs, chariots, tuyauterie.., chaque transformation peut résulter soit d'une série de manipulations séquentielles, soit d'un traitement de type continu. Les industries chimiques, pharmaceutiques, biochimiques, agro-alimentaires présentent des exemples typiques de ce type de production.

Un processus par lot est donc essentiellement à évolution séquentielle, le contrôle logique pouvant piloter soit des organes tout ou rien classiques, soit le lancement de taches continues (essentiellement régulation) dont le résultat se traduit par un prédicat soit simple indicateur de fin de tache soit expression logique complexe intégrant des comparaisons de grandeurs analogiques à des seuils prédéfinis.

Beaucoup de ces processus ne sont que partiellement automatisés. En effet, le coût de vannes motorisées en inox est tel qu'il devient pratiquement impossible de contrôler automatiquement toute une installation. Il convient alors de prévoir l'opérateur comme organe de commande universel et de l'inclure dans le processus de pilotage en lui donnant des ordres et en lui demandant de valider ses actions.

Une des grandes caractéristiques des traitements par lot est le paramétrage. De nombreuses fabrications se font suivant des recettes secrètes permettant, à partir d'un ensemble de matières premières, d'élaborer une variété de produits. Certaines modulations doivent être apportées aux recettes en fonction de la qualité des ingrédients.

Ces fluctuations de qualité (teneur en matières essentielles) entraînent un certain caractère aléatoire de la fabrication.

Par ailleurs, le comportement de certains traitements ou de certaines réactions chimiques ou biochimiques n'est pas complètement maîtrisé, surtout lorsqu'on utilise des quantités importantes.

C'est pourquoi l'opérateur a un rôle essentiel de pilotage et il doit pouvoir intervenir à tout moment pour modifier les paramètres d'une recette, pour conduire en manuel une phase mal engagée, pour forcer le système dans une procédure particulière pour bloquer l'effet de certains verrouillages de parties annexes.

Cette caractéristique rend très difficile l'automatisation et oblige à vérifier de nombreuses conditions de sécurité, dépendant souvent de l'état du système. Il faut donc un contrôle et une surveillance d'un niveau supérieur.

2.3. PARTIE COMMANDE - PARTIE OPERATIVE

Pour pouvoir décrire le cahier des charges de la partie commande d'un processus automatisé, il est fondamental de bien définir la frontière entre cette partie commande et la partie commandée, dite partie opérative, sur laquelle on agit par l'intermédiaire d'objets technologiques (vannes, moteurs, etc.).

Ce problème est d'autant plus complexe que la frontière n'est pas nette à cause de l'existence d'interface matérielle ou fonctionnelle. Ainsi, un ordre émis par le système de commande peut ne pas se transformer en action au niveau de la P.O. En sens inverse, un signal partant de P.O. peut ne pas donner une information à la P.C.(fig.2.1).

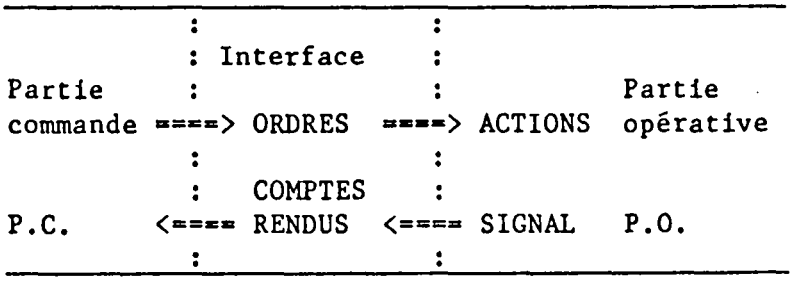


Figure 2.1

La description que nous allons étudier porte sur le système de commande, mais suppose que, par ailleurs, existe la définition des éléments d'interface.

Exemples de fonctions d'interface (fig.2.2)

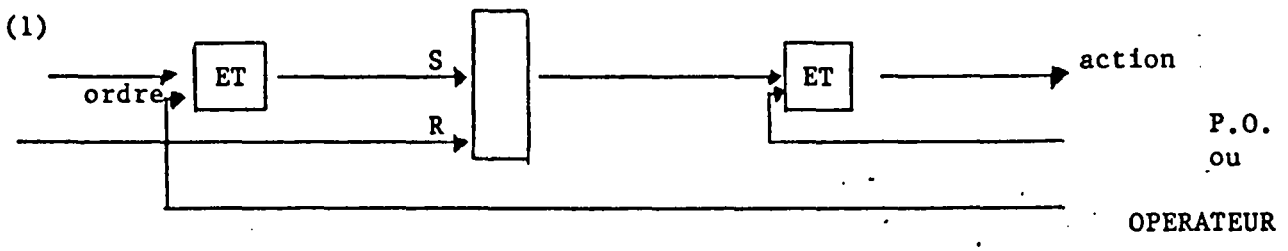


Figure 2.2.

(2) boucle : document R.P. schéma de boucle (figure 2.3).

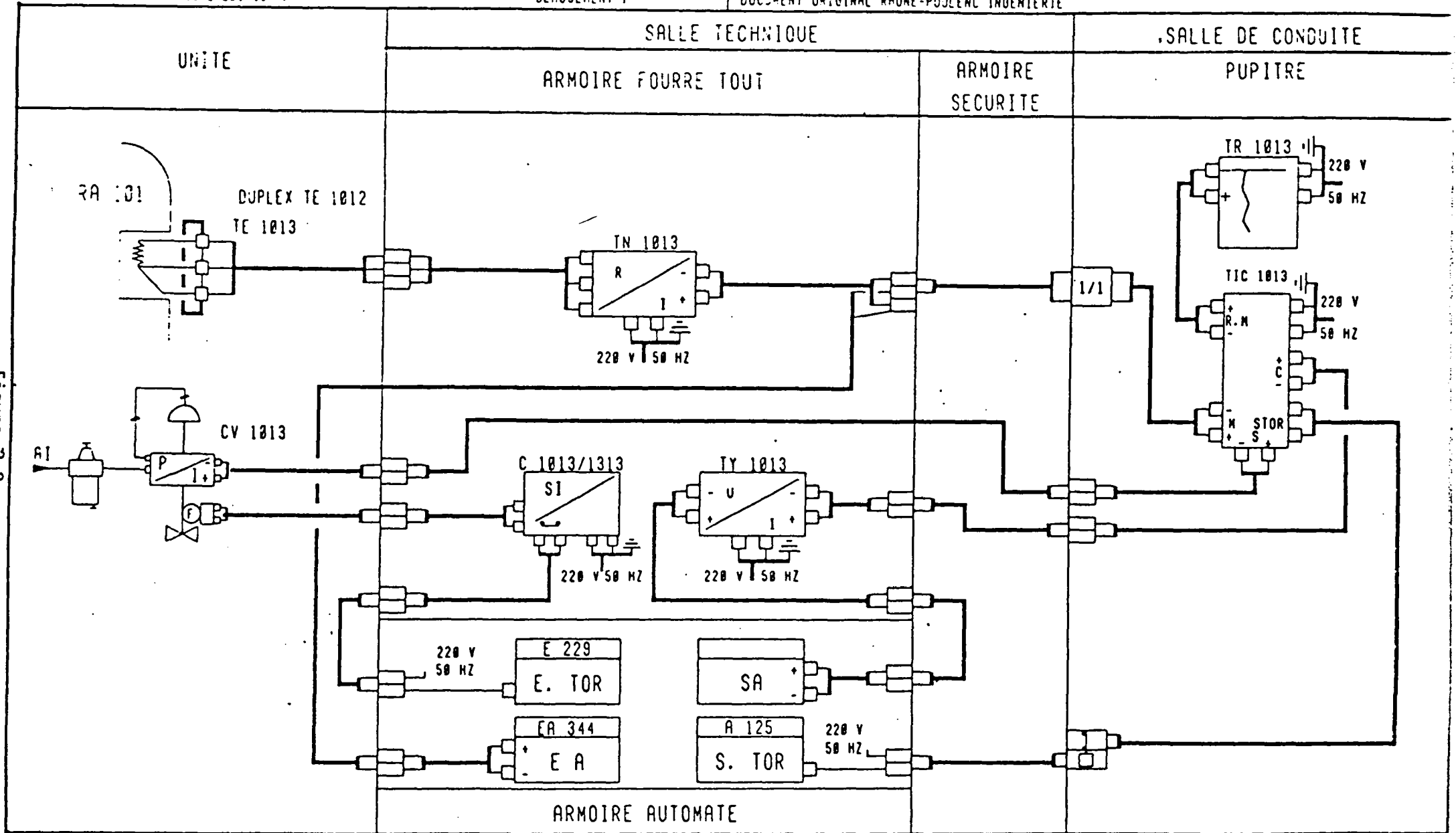


Figure 2.3

RHONE-POULENC	SCHEMA DE BOUCLE			TRC 1013	INSTRUMENTATION
	MISE A JOUR	1	4		CLIENT/AFF. : RP/SC
		2	5		ETABLISSEMENT : USINE DE MELLE
		3	6		UNITE/SECTION : RHODIASTAB
					SA 039
					DATE : 88/03/04

Nous avons à définir également une autre catégorie d'éléments commandés par P.C., et donc extérieurs à la description, même si souvent ils sont réalisés par le même dispositif matériel que P.C.

On trouvera ainsi des modules comme :

- mémoire
- compteur
- temporisation
- régulateurs
- contrôleurs de position d'axes
- élément de calcul.

Ces modules fonctionnels, dits fonctions opératives, doivent eux aussi être décrits de façon à ce que, dans tous les cas, la description de P.C. puisse être documentée.

2.4. NOMENCLATURE

Avant de pouvoir aborder la description de la partie commande, il convient de pouvoir disposer d'une documentation complète concernant tous les objets technologiques, dits postes d'instrumentation, qui nous servent à agir sur la partie opérative. Un poste d'instrumentation est un capteur ou un actionneur dont on donne la nature et la fonction, exemple : V.200, électro-vanne, vanne de purge.

La connaissance de toutes les fonctions d'interface et des fonctions opératives est également obligatoire pour fixer convenablement le rôle de la partie commande.

La nomenclature, véritable catalogue de ces objets contient, sous la forme d'une base de données technologiques, les caractéristiques de relation entre chaque objet, la partie commande et/ou la partie opérative. Elle décrit également les fonctionnalités de chaque objet en termes identiques à ceux de la description de la partie commande.

Le nomenclature est aussi destinée à servir de déclarations aux programmes rédigés selon le langage de description de la partie commande. Elle définit le nom des repères, des actions et les descripteurs d'états tels qu'ils sont utilisés par le bureau d'étude, les exploitants et les automaticiens.

Cette fonction contribue à augmenter la lisibilité des programmes par des non spécialistes et en particulier par les exploitants

2.4.1. Modèle générique.

Une des grandes caractéristiques de ces éléments est leur généricité. On décrira par exemple la classe générique "vanne" ou temporisation" avec ses lignes de commande, ses lignes d'informations, ses variables spécifiques en précisant si elles sont à lecture unique, à lecture-écriture ou à écriture unique, ses paramètres statiques (période d'horloge, capacité maximale, etc.).

Ainsi, du côté de la partie commande, on décrit les lignes de commande et d'information. Du côté de la partie opérative, quand elles existent, on précise les entrées et sorties réelles correspondantes. On décrit enfin des relations combinatoires numériques ou séquentielles de cet objet.

Par exemple, le modèle d'une vanne ouverte par manque d'air décrira :

- Une sortie réelle de type logique pour la commande de la vanne.
- Deux entrées réelles de type logique pour les liaisons fin de course.
- Une ligne de commande : FERMER.
- Trois lignes d'information : OUVERTE, FERMEE, DEFAUT.
- Les expressions d'évaluation de ces informations, à partir des entrées réelles et de la commande, expressions combinatoires; dans ce cas particulier.

Toutes les lignes sont typées :

- Les entrées réelles peuvent être soit numériques (analogique ou digitales), soit booléennes.
- Les sorties réelles sont soit numériques, soit booléennes monostables, soit booléennes bistables.

Une fonction opérative ou d'interface comporte vis-à-vis de son comportement temporel plusieurs sous-ensembles :

- Le bloc des relations à exécution sur évènement. A chaque transition montante ou descendante d'une ligne d'entrée réelle de type booléen ou sur une transition montante d'une ligne de commande de type booléen monostable, on évalue la relation associée.
- Le bloc des relations à évaluation cyclique. Dans ce cas, l'évaluation est en synchronisme avec le traitement de la partie commande.
- Le bloc des relations synchrones d'une horloge.

2.4.2. Instantiation

Le modèle générique décrit une seule fois peut être utilisé de façon multiple, à condition d'en faire l'instantiation pour chaque utilisation. Pour chacun des modules réels générés, les lignes deviennent personnalisées et prennent une existence vis-à-vis des parties commande et opératives.

Certaines variables génériques permettent de faire du paramétrage statique, en fixant à l'instantiation la liste des paramètres.

La généricité ne porte ni sur les structures, ni sur les fonctions.

Pour décrire ces objets un à un, on précise pour chacun :

- son repère : code externe associé au poste instrumentation,
- son descriptif, c'est-à-dire son libellé en clair;
- sa classe de définition, c'est-à-dire son modèle de référence.

Ainsi, dans l'exemple précédent, on aurait :

- repère V 200,
- libellé vanne de purge,
- classe OMA (ouverte par manque d'air).

Ceci entraîne que FERMER-V200 est une commande et V200-OUVERTE est une ligne d'information.

La nomenclature assure donc, pour une frontière donnée entre partie commande et partie opérative, le passage sans discontinuité entre les comportements externes et internes en définissant les primitives de départ de description de la partie commande.

2.4.3. Exploitation

La nomenclature est gérée comme une véritable base de données avec toutes les facilités d'édition et de manipulation. La gestion permet d'assurer, dans un premier temps, un contrôle de cohérence des repères de postes instrumentation utilisés dans les différents documents techniques et schémas d'installation. Dans un second temps, il permet de générer les documents de câblage vers la partie opérative.

2.5. LES GRANDES OPTIONS POUR LA DESCRIPTION DE LA PARTIE COMMANDE.

Le choix de l'outil de description est le point important pour une bonne conception de la commande. En effet, cet outil doit permettre en amont l'introduction d'une méthodologie de description et, en aval, une implantation facile sur dispositifs programmés.

Les grandes options retenues pour notre outil sont les suivantes :

2.5.1. Proche de l'utilisateur.

Le point de vue adopté est de ne faire ni un langage de programmation, car ceux-ci abordent d'emblée un trop grand niveau de détail et ont un caractère essentiellement dynamique, ni un langage universel de spécification qui serait trop général et destiné à des utilisateurs informaticiens. On veut s'adresser à des hommes de procédé ayant plus une connaissance du fonctionnement de la partie opérative que des subtilités des langages de programmation. Ce langage doit donc être d'un apprentissage aisé et facile d'emploi.

2.5.2. Une description en langage semi-naturel.

Pour permettre une production automatisée de logiciel, le choix s'est porté sur l'utilisation d'un langage de haut niveau avec certaines structures préétablies proches de l'utilisateur. Une description semi-naturelle est envisagée pour donner un texte source bâti à partir d'éléments définis par l'utilisateur agencés dans des structures de bases.

2.5.3. Avec une aide graphique.

Le graphique n'est certes pas la composante la plus fondamentale de la description. Pour les petits problèmes, un bon graphisme vaut peut-être mieux qu'un discours, mais lorsque la taille du problème augmente, la densité de graphisme rend la description difficile à utiliser.

Toutefois, le principe de description retenu permet de donner en complément à la formulation semi-naturelle une représentation graphique des structures. Il est par ailleurs possible d'obtenir un graphisme de type Grafcet ou Réseau de Pétri si on le souhaite.

2.5.4. Une approche hiérarchisée et structurée.

Cette approche est la seule permettant l'abord de problèmes importants. Elle est partout admise en programmation. Le résultat de l'analyse est une structure arborescente multiniveaux. Une des difficultés dans l'art de l'analyse est de trouver un nombre raisonnable de niveaux, en sachant qu'un nombre trop important pénalise l'implantation et qu'un nombre trop petit donne un style de description difficile à lire.

L'objectif est en tous cas d'éviter au maximum de représenter des structures séquentielles par des expressions combinatoires importantes.

2.5.5. Sans discontinuité avec les fonctions d'interfaces, les fonctions opératives et la partie opérative.

La description de la partie commande doit être directement en relation avec la description des objets technologiques. Nous avons vu, dans le paragraphe sur la nomenclature, que celle-ci définit les primitives de départ de la description.

Toutefois, certains objets peuvent se matérialiser par une structure de contrôle, comme par exemple le compteur, l'arborescence peut alors se prolonger par la description de la réalisation de fonction de comptage.

On peut ainsi se rendre compte qu'il n'y a pas réellement de discontinuité entre la description de la partie commande et la nomenclature.

2.5.6. Avec des bases théoriques saines.

Ce point est particulièrement essentiel pour garantir une qualité de la description, sa transmissibilité. Le modèle doit permettre des validations et des vérifications de cohérence. Il doit être indépendant du temps et, pour cela, répondre aux hypothèses de base suivantes :

- Le modèle évolue d'état stable en état stable par l'intermédiaire de transitoire infiniment rapide.
- Il n'y a pas de variations simultanées d'entrée.
- Il n'y a pas de variations d'entrée pendant un régime transitoire.
- Il n'y a pas de cycle transitoire permanent.
- Il n'y a pas de discontinuité de sortie liée à un passage transitoire.

./.

Le modèle choisi est un modèle booléen réceptif multiniveaux structuré, permettant d'appliquer toutes les méthodes de validation formelle, de simulation et de production automatisée de logiciels concernant les réseaux de Pétri sauf interprétés ou les grafcets.

Par ailleurs, pour éviter des redondances de description, le modèle doit être paramétrable, tant sur le plan des identificateurs que des valeurs. Enfin, il doit permettre le parallélisme et ses conséquences.

2.5.7. Introduction d'éléments sémantiques.

En plus d'une définition syntaxique, il est intéressant pour l'utilisateur de pouvoir préciser le rôle et la signification de chacun des éléments de sa description.

Dans la nomenclature, comme dans la description de la partie commande, une large place doit être réservée à cet aspect sémantique (libellés en clair, commentaires,etc.).

2.5.8. Indépendance de la technologie et génération automatique de programme.

La description est indépendante de la technologie de la partie commande et du langage du système d'implantation ultérieurement employé. Par contre, des éléments complémentaires sont introduits au niveau du traducteur pour faire intervenir la spécificité du matériel et permettre une génération automatique des programmes exécutables et une édition du dossier de câblage, des listes d'erreurs, et de la documentation.

2.6. PRINCIPE DU PAI.

Le poste d'automatisation et d'instrumentation PAI peut se décomposer en deux parties (figure 2.4).

2.6.1. Une partie interactive comportant des outils d'aide à la description de la partie commande et de son interface avec la partie opérative du processus automatisé. Ces outils sont d'une part une règle de codification qui permet la gestion des nomenclatures instrumentation sous la forme d'une base de données relationnelle, et d'autre part un langage de spécification. Ces outils sont indépendants du type de machine de commande utilisé pour l'automatisation du processus industriel. La mise en oeuvre s'effectue en mode conventionnel à l'écran.

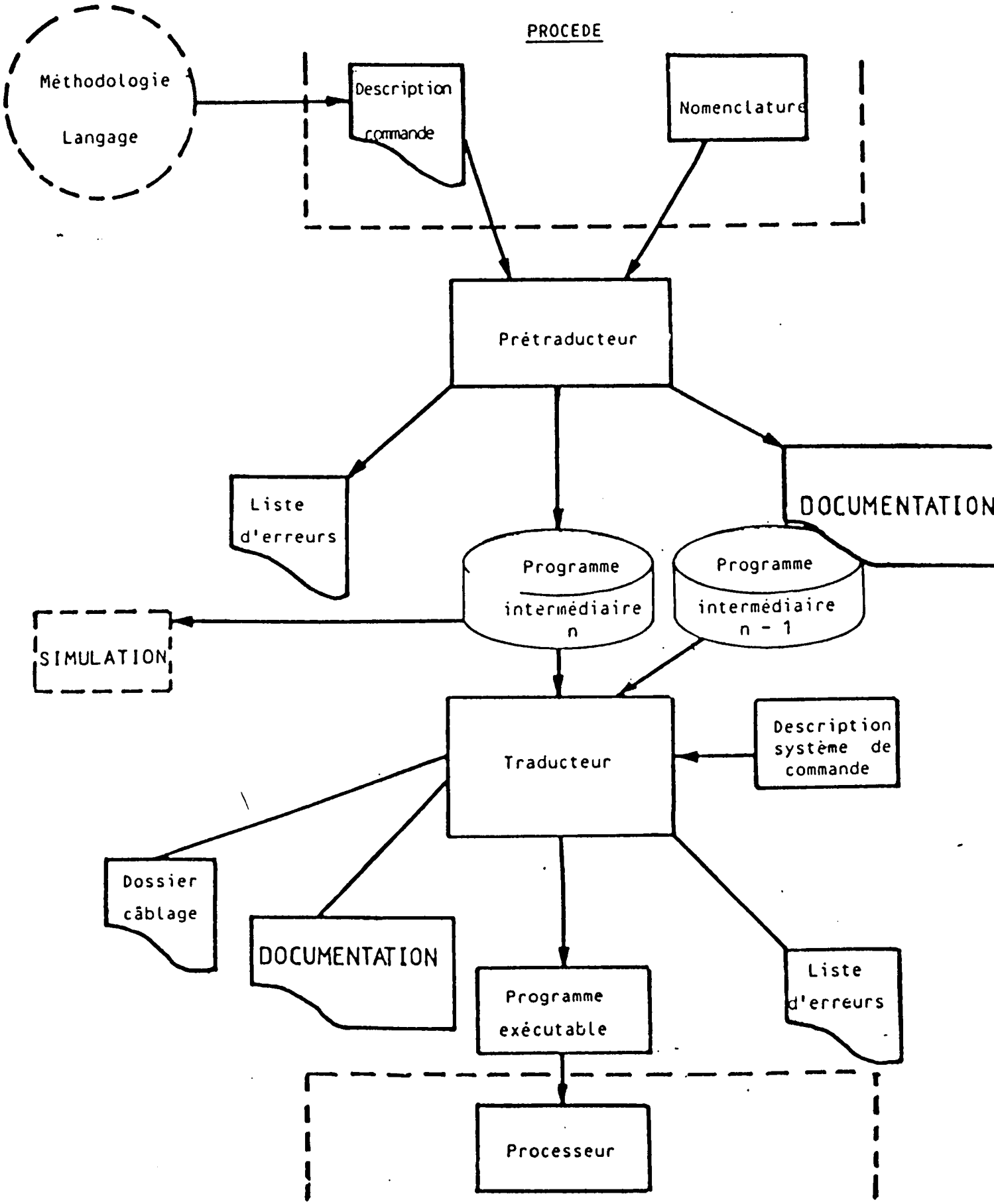

PROCEDE
Méthodologie
Langage
Description commande
Nomenclature
Prétraducteur
Liste d'erreurs
DOCUMENTATION
Programme intermédiaire n
Programme intermédiaire n - 1
SIMULATION
Traducteur
Description système de commande
Dossier câblage
DOCUMENTATION
Programme exécutable
Liste d'erreurs
Processeur
SYSTEME DE COMMANDE

Figure 2.4

2.6.2. Une partie traitement comportant des outils de réalisations sur deux niveaux :

. Un prétraducteur ayant pour but :

- La prise en compte de la description.
- La réalisation de tous les contrôles de vraisemblance et d'existence par rapport à la base de données des nomenclatures.
- La génération d'un programme suivant un langage booléen intermédiaire indépendant de la machine d'exécution.
- La fourniture d'une documentation des programmes sous forme de liste et de schémas.

. Un traducteur ayant pour but :

- La traduction des programmes en langage compatible avec les spécifications de la machine d'exécution retenue.
- La fourniture des documentations de réalisation (liste des adresses, câblage, implantation,...).
- L'assemblage de plusieurs programmes sur la même machine d'exécution.

2.7. METHODE D'ABORD DES PROCESSUS PAR LOT

2.7.1. Généralités

Pour aborder la description d'un processus par lot, le concepteur doit trouver une base de décomposition et d'analyse. Celle-ci est essentiellement la notion de phase procédé. Toutefois, pour éviter les découpages aléatoires et difficilement exploitables, il convient de tenter une définition précise de la phase procédé.Le critère de découpage est lié à la notion de contexte associé à une phase.

L'objectif d'une phase est d'effectuer un certain travail dit partie dynamique de la phase, sur une matière d'oeuvre. Pour ceci, il est nécessaire au procédé de disposer d'un ensemble d'objets technologiques dans un état particulier dit contexte. Lorsque débute la phase procédé, cet état n'est pas forcément atteint et il convient, avant d'aborder le lancement de la partie dynamique, d'achever la mise en place du contexte.

Le contexte correspond donc à l'état maintenu d'un ensemble d'objets technologiques pendant la partie active de la phase procédé.

Lorsque se termine une phase, chacun des objets technologiques du procédé se trouve dans un état dont une partie est issue du contexte de la phase et l'autre partie est le résultat de sa partie active.

L'état d'un sous-ensemble des objets technologiques utiles à l'exécution de la nouvelle phase est maintenu sans discontinuité pour constituer une partie du contexte. L'autre partie faisant appel à de nouveaux objets doit être amenée dans l'état voulu : c'est la préparation du nouveau contexte.

En utilisant cette définition de la phase, la description du fonctionnement de la partie commande s'effectue sur trois niveaux :

- La description de la phase elle-même avec la préparation du contexte, le contrôle du maintien de ce contexte et le contrôle de l'évolution de la partie dynamique.

- Le moniteur de phase procédé dont le rôle est de surveiller les anomalies lors du déroulement d'une phase, de contrôler l'état de la phase et des opérations, d'appliquer les modes de marche induits par le moniteur général, de réaliser des comptes rendus d'état vers le moniteur général, et enfin de faire la gestion des bilans.

- le moniteur général dont le rôle est d'assurer l'initialisation, le contrôle des modes de marche, le dialogue opérateur, et enfin la communication avec les autres parties d'un contrôle réparti.

L'organisation des phases entre elles doit pouvoir être décrite de la même façon que leurs fonctionnements sans discontinuité. Seule l'accessibilité à certaines informations, concernant en particulier l'état des opérations ou des phases, peut changer.

2.7.2. Modes de marche et d'arrêt

2.7.2.1. Définitions

Un des rôles essentiels du moniteur est de permettre, à la demande de l'opérateur, d'introduire divers modes de marche et d'arrêt.

On peut ainsi définir six modes de fonctionnement en dehors d'un fonctionnement manuel :

* le mode automatique.

* le mode semi-manuel où le système de commande suit l'évolution de la partie opérative sans toutefois lui appliquer les commandes calculées.

* le mode réglage ou l'opérateur peut introduire des points d'arrêt pour obtenir un fonctionnement opération par opération, séquence par séquence, ou phase par phase.

* le mode cycle à cycle : il s'agit en fait d'une marche automatique avec un point d'arrêt privilégié où il est possible d'introduire tous les paramètres nécessaires au cycle automatique suivant.

* le mode de forçage : en fonction d'une situation particulière, l'opérateur a la possibilité d'interrompre un traitement et d'en forcer une autre.

Toutefois, ceci ne devrait pas pouvoir se faire à n'importe quel moment ni dans n'importe quelles conditions, même si des verrouillages sur les commandes existent.

Le rôle de ce mode de fonctionnement est de permettre :

  . les tests de programme et le démarrage d'une installation

  . certains fonctionnements dégradés lors d'anomalie.

  . certaines modifications de paramétrage de recette en cours de fonctionnement.

* le mode d'arrêt ou de détournement se rencontre soit en cours d'arrêt d'urgence par intervention de l'opérateur, soit en cas de discordance et d'alarme où il peut y avoir nécessité de dérouter le processus vers une procédure d'urgence ou de marche dégradée.

2.7.2.2. Notion d'anomalie

Les anomalies au cours d'une phase peuvent provenir soit du contexte lui-même, lorsqu'il n'est plus possible d'en garantir le maintien, soit lors de la préparation du contexte, soit dans la partie dynamique de la phase.

Certaines anomalies de contexte n'empêchent pas pour autant le déroulement de la phase, par contre, dans d'autres cas, on est obligé de se replier en début de phase ou d'appliquer des procédure spéciales.

Lorsqu'on se trouve dans une partie dynamique, une anomalie provoque généralement un repli vers une position dépendant de l'état de la phase.

2.7.2.3. Détection des anomalies

La détection d'une anomalie peut se faire de façon statique ou dynamique. Elle est statique lorsqu'une certaine situation combinatoire est soit décelée par les sécurités actives câblées et répercutées vers la partie commande, soit calculée par l'interface logiciel.

Elle est dynamique lorsqu'elle tient compte dans le temps de la relation causale action effet. Dans ce sens, on peut ne pas obtenir d'effet, en obtenir un trop vite, en obtenir un inattendu.

On appelle discordance d'ordre l'obtention d'une relation causale erronée au niveau d'un objet technologique. Par ailleurs, il peut se produire que des conditions permanentes de verrouillage empêchent une action d'être produite par la partie commande.

On dit qu'il y a discordance de tâche. La détection se fait par traitement combinatoire dans la partie commande.

Toute anomalie entraîne un état d'alarme actif vis-à-vis de l'opérateur et du moniteur de phase.

2.7.2.4. Point d'entrée - point d'arrêt

Les cas anormaux pouvant nécessiter un forçage se rencontrent :

. lors d'un anomalie générale (panne d'énergie).

. lors d'un blocage sous une action conditionnelle le forçage peut consister en une mise à 1 de la variable représentative sur l'élément bloquant.

. lors d'un blocage d'évolution de la partie commande à cause d'un capteur : le forçage est soit global sur l'expression où intervient la variable capteur, soit uniquement par mise à 1 ou à 0 de cette variable.

. lors d'une anomalie d'actionneurs ou de procédés.

Dans le premier et le dernier cas, la solution est de forcer le système en un point de sécurité prévu en fonction de la position dans le déroulement du traitement. En ce point, on peut arrêter le traitement ou dérouler des procédures particulières.

Ces replis peuvent être commandés par l'opérateur lui-même via le moniteur ou par le moniteur suivant un plan prévu et décrit.

Dans tous les cas, on peut définir l'usage d'un point d'entrée PE. Pour une phase donnée, il existe un (ou plusieurs) point d'entrée de phase tels que le moniteur général peut y accéder par forçage supérieur.

Par ailleurs, il existe des points d'entrée locaux à une phase utilisables seulement à l'intérieur de la phase par le moniteur de phase. Il est à noter que les points d'entrée généraux sont aussi locaux.

La notion de point d'arrêt permettra de résoudre tous les types de modes d'arrêt, de réglage et de fonctionnement cycle à cycle. En effet, à un point d'arrêt PA, nous associons un numéro de type (0 à n) où 0 inhibe tout arrêt : 1 à n sont utilisés par la description en sachant que 1 arrête 1, 2 arrête 1, 2...

Remarquons de plus qu'un point d'entrée peut également être un point d'arrêt, lors d'un repli avec attente de réparation. Pour utiliser les points d'arrêt, l'opérateur sélectionne un numéro de type, le système se bloque à chaque point d'arrêt sélectionné, le déblocage se fait par validation de l'opérateur.

2.8. OUTIL DE DESCRIPTION DE LA PARTIE COMMANDE

Le modèle de description choisi pour la partie commande est du type booléen réceptif multiniveaux. L'élément de base de la description est l'opération, qui peut être de plusieurs types et qui peut s'associer à d'autres opérations pour former une arborescence.

Les feuilles de l'arborescence sont vis-à-vis de l'extérieur de la partie commande les opérations exécutives c'est-à-dire celles qui envoient effectivement les ordres vers les fonctions d'interface les fonctions opératives ou la partie opérative.

Elles jouent donc un rôle particulier. On les appelle OPERATIONS ELEMENTAIRES.

Les autres opérations sont structurantes, elles ne serviront qu'à bâtir l'arborescence.

2.8.1. Opération élémentaire

Une opération élémentaire peut se représenter en termes de Grafcet par la figure 2.5. Une fois lancée, l'opération élémentaire se met en attente d'une réceptivité d'entrée RE, fonction logique complexe des entrées de la partie commande et des indications du moniteur de phase. Losque RE passe à 1 l'opération passe dans la partie d'application des ordres. Ceux-ci ont été classés en deux catégories :

- les ordres non répétés conditionnels ou non qui sont appliqués une fois seulement immédiatement après le passage à 1 de RE.

- les ordres répétés conditionnels ou non qui sont réévalués et appliqués à chaque cycle de traitements jusqu'à ce qu'une réceptivité RS dite réceptivité de sortie passe à 1.

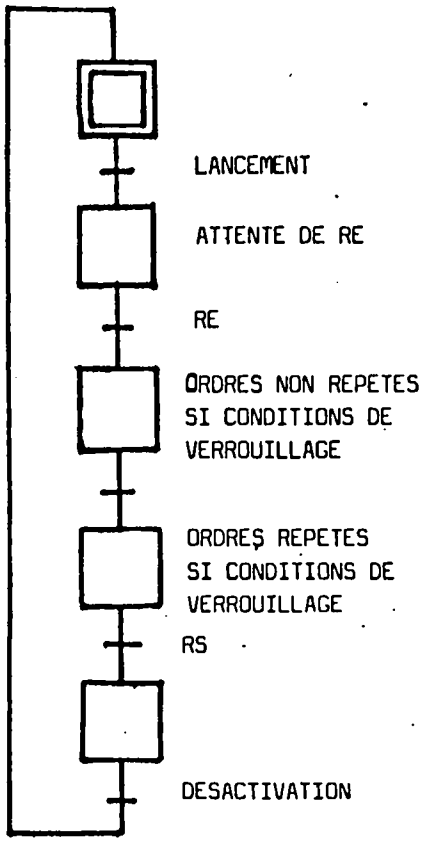
La figure 2.5. donne aussi, en correspondance avec le Grafcet de l'opération élémentaire, sa syntaxe en langage semi-naturel.

Les ordres intervenant dans les opérations élémentaires de même que les variables d'entrée figurant dans les réceptivités ou les conditions doivent être en conformité avec la définition des modules faite en nomenclature par instantiation des modèles. Compte tenu de cette définition, il est possible de définir par le moniteur de phase l'état d'une opération élémentaire ;

Celle-ci pourra être :

- inactive
- active avant RE
- active avant RS sans blocage par les conditions de verrouillage
- active avant RS avec blocage par les conditions de verrouillage
- active après RS.

Il est de plus à noter que si RE est toujours vraie la partie correspondante peut être omise. Il peut ne pas y avoir d'ordres non répétés ou répétés et les conditions de verrouillage peuvent ne pas exister.




```
* OPELEM        :NOM
  TYPE          :E
  LIBELLE       ROLE DE L'OPERATION

ATT JSQ         RE
    SI              CONDITION.1
         ALORS                          ORDRES
          ORDRE.1;                      NON
    FSI                                 REPETES
    SI              CONDITION.N
         ALORS
          ORDRE.N;
    FSI
FAIRE
         SI             CONDITION.1'
              ALORS                     ORDRES
               ORDRE.1';                REPETES
         FSI
         SI             CONDITION.N'
              ALORS
               ORDRE.N';
         FSI
JSQ           RS

) FOPEL         :NOM
```

FIGURE 2.5

./..

2.8.2. Opérations structurantes

Ces opérations ont pour objectif de définir des structures plus complexes par association d'opérations ou d'opérations élémentaires déjà définies. Elle suivent les règles de l'analyse structurée. On y trouve donc la séquence, la sélection et l'itération. On y adjoint le parallélisme.

Une fois active, une opération peut elle aussi se trouver en attente de RE. Une fois cette réceptivité d'entrée vraie, l'opération lance suivant le cas, une ou plusieurs opérations ou opérations élémentaires d'un niveau inférieur.

Les états d'une opération vus du moniteur de phase sont les suivantes :

- inactive
- active avant RE
- active après RE
- achevée lorsque toutes les opérations de niveau inférieur qu'elle controle sont achevées ou actives après RS.

La réceptivité RE peut toujours être vraie.

2.8.2.1. Séquence

L'opération séquence s'écrit suivant la syntaxe de la figure 2.6.

Les constituants de la séquence sont des opérations ou des opérations élémentaires.

Pour alléger l'écriture on pourrait admettre l'écriture directe d'ordres non répétés conditionnels ou non dans une séquence.

Ceux-ci ne seront pas alors précédés du symbole *.

```
* OPE          :NOM
  TYPE         :E
  LIBELLE      ROLE DE L'OPERATION

 ATT JSQ       RE
     SEQ
          *OPERATION.1 OU OPELEM.1;
                    :
                    :
          *OPERATION.N OU OPELEM.N;
     FSEQ

) FOPE          :NOM
```

FIGURE 2.6

2.8.2.2. Sélection

L'opération de sélection peut se faire à choix binaire ou à choix multiples exclusifs. La syntaxe est celle de la figure 2.7.

```
* OPE          :NOM
  TYPE         :E
  LIBELLE      ROLE DE L'OPERATION

     SI              CONDITION
          ALORS
          *OPERATION.1 OU OPELEM.1;
          SINON
          *OPERATION.2 OU OPELEM.2;
     FSI

) FOPE         :NOM
```

```
* OPE          :NOM
  TYPE         :E
  LIBELLE      ROLE DE 'OPERATION

     SI              CONDITION.1
          ALORS
          *OPERATION.1 OU OPELEM.1;
     OUX SI          CONDITION.2
          ALORS
          *OPERATION.2 OU OPELEM.2;
     OUX SI          CONDITION.M
          ALORS
          *OPERATION.M OU OPELEM.M;
          SINON
          *OPERATION.N OU OPELM.N;
     FSI

) FOPE         :NOM
```

FIGURE 2.7

Pour alléger la description, on pourrait admettre le remplacement de l'opération i par le contenu de cette opération ou opelem i par un ordre non répété conditionnel ou non. Ceci évite d'avoir à créer des opérations d'ordre inférieur au détriment de la lisibilité de l'opération en cours et de l'absence d'information sur l'état des opérations imbriquées non déclarées.

2.8.2.3. ITERATION

Les formes de l'opération itération sont multiples. On peut en effet trouver les structures classiques de l'analyse structurée : jusqu'à (DO UNTIL) et tant que (DO WHILE), mais aussi les structures étendues :

- attendre jusqu'à (WAIT UNTIL)

- répéter n fois (REPEAT n times)

- DO ... WHILE ... REPEAT.

Même si les formes étendues ont été admises dans le langage d'application, on ne présentera ici que la syntaxe des formes de base (figure 2.8.) où comme précédemment les opérations pourraient être remplacées par leur contenu et l'opération élémentaire par l'ordre non répété conditionnel ou non.

```
* OPE          :NOM
  TYPE         :E
  LIBELLE      ROLE DE L'OPERATION

     EXECUTER
          TANT QUE    CONDITION
          *OPERATION;
     RECOMMENCER

) FOPE         :NOM
```

FIGURE 2.8

2.8.3. LE PARALLELISME ET SES CONSEQUENCES

2.8.3.1. Définition et objectifs

La notion du parallélisme peut se rencontrer à des niveaux et des endroits divers au cours de l'élaboration d'un système de commande. Il peut exister un parallélisme de réalisation.

Les systèmes multiprocesseurs sont utilisés soit pour augmenter la vitesse du traitement, soit pour régler des problèmes d'implantation géographique, soit encore pour assurer une meilleure disponibilité. Une autre orientation du parallélisme concerne la description des processus de commande.

L'usage à cet endroit d'un parallélisme n'est pas concerné par la structure matérielle qui servira lors de l'implantation et de l'implantation en monoprocesseur. L'objectif essentiel est d'abord de permettre un découpage du système en sous-système indépendants couplés uniquement par des points de synchronisation. L'étude de chacun de ces sous-systèmes lancés en parallèle peut correspondre à deux besoins :

- lancer une action contrôlée pour obtenir un effet sur la partie opératoire,

- lancer un calcul ou un traitement numérique pour obtenir une solution.

La synchronisation indique simplement que les résultats escomptés ont été obtenus sur l'ensemble des sous-systèmes simultanés. Il est à noter qu'avec par exemple un traitement monoprocesseur, si les sous-systèmes n'ont que des calculs à effectuer, ceux-ci peuvent tous sans inconvénient, être exécutés en séquence.

Le parallélisme n'est que de description.

On ne passe simplement à la suite que lorsque tous les résultats ont été obtenus.

Par contre, s'il s'agit d'actions contrôlées, l'implantation doit permettre au cours d'un cycle de traitement, la surveillance de chacune des actions.

Le temps de cycle global est donc à partager entre les sous-systèmes placés en parallèle.

2.8.3.2. Les différentes formes de parallélisme

a) Le parallélisme décrit structurellement

La forme générale de l'opération pour le parallélisme doit conserver la notion de structuration, c'est pourquoi nous avons adopté la syntaxe de la figure 2.9.

```
* OPE            :NOM
  TYPE           :E
  LIBELLE        ROLE DE L'OPERATION

  ATT JSQ        RE
      PARALL
          *OPERATION.1 OU OPELEM.1;
           :
          *OPERATION.M OU OPELEM.M;
      FPARALL

) FOPE           :NOM
```

FIGURE 2.9

On peut ici également autoriser le remplacement d'une opération par son contenu.

Cette écriture peut couvrir deux aspects différents du parallélisme :

Le parallélisme obligatoire dans lequel il y a nécessité de mettre en oeuvre simultanément les deux opérations.

Par exemple, dans le cas d'un arrêt d'urgence, dès l'apparition du signal AU, il faut lancer à la fois un éclairage de secours et une alarme.

Le parallélisme autorisé

Dans beaucoup de cas de figure, l'énoncé apparaît de la façon suivante : une opération T2 peut être lancée dès le début d'une opération T1 (voir exemple optimisation du nombre d'opérations automatiques de dosage - malaxage du premier chapitre.

Par ailleurs à cette forme générale on peut adjoindre plusieurs cas particuliers :

1 - Le parallélisme comportant des opérations implicites

L'idée de parallélisme est ici différente,l'opération implicite sans réceptivité d'entrée cesse dès que l'opération principale parallèle cesse.

Dans le cas d'une opération implicite avec condition d'entrée, il est possible que cette réceptivité soit bloquante, conditionnant ainsi la sortie du parallélisme.

Pour introduire cette notion dans la syntaxe on peut définir une nouvelle structure (figure 2.10.) permettant dès l'introduction des vérifications syntaxiques spécifiques. On peut aussi utiliser une opération élémentaire et mettre une réceptivité de sortie prédéfinie : FPARALL

```
* OPELIMP      :NOM
  TYPE         :E
  LIBELLE

 FAIRE
              ACT1;
 JSQ FPARALL

) FOPELIMP     :NOM
```

FIGURE 2.10

Il est à noter que le remplacement d'une OPELEM dans la figure 2.9. par un ordre direct conditionnel ou non fait effet d'opération implicite. Cet usage n'est toutefois pas à conseiller.

L'opération implicite peut être introduite dans des structures plus complexes. Considérons par exemple une séquence se terminant par une opération implicite vis-à-vis du parallélisme, cette opération garde un comportement implicite et est donc valable. Il en va de même d'une sélection entre plusieurs opérations implicites. La sélection mixte entre opération implicite et normale est interdite. Pour permettre l'introduction de ces notions on peut adjoindre deux types nouveaux de structures pour permettre une validation syntaxique immédiate. On peut aussi prendre les structures normales d'opération ; le contrôle de validité se faisant alors au moment de la traduction.

./..

2 - Le parallélisme décalé

Cette forme de parallélisme est assez fréquente dans les processus par lots. On lance en parallèle plusieurs opérations par exemple OP1, OP2 et OP3 mais OPE2 ne démarre réellement que lorsqu'une conséquence directe de OP1 apparaît, OP1 continuant à rester en état actif avant RS. Il en irait de même pour OP3 relativement à OP2. Malgré la répétitivité de cette structure, il ne parait pas nécessaire de définir un nouveau type d'opération d'une part pour limiter le nombre de primitives du langage et d'autre part pour éviter le risque d'interaction mal défini entre branches parallèles. La structure de base permet de décrire une telle situation.

En terme de Grafcet, on obtient deux structures solutions en considérant par exemple OP2 comme une OPIM. Ces solutions (figure 2.11.) n'ont pas le même effet pour la fin de OP2.

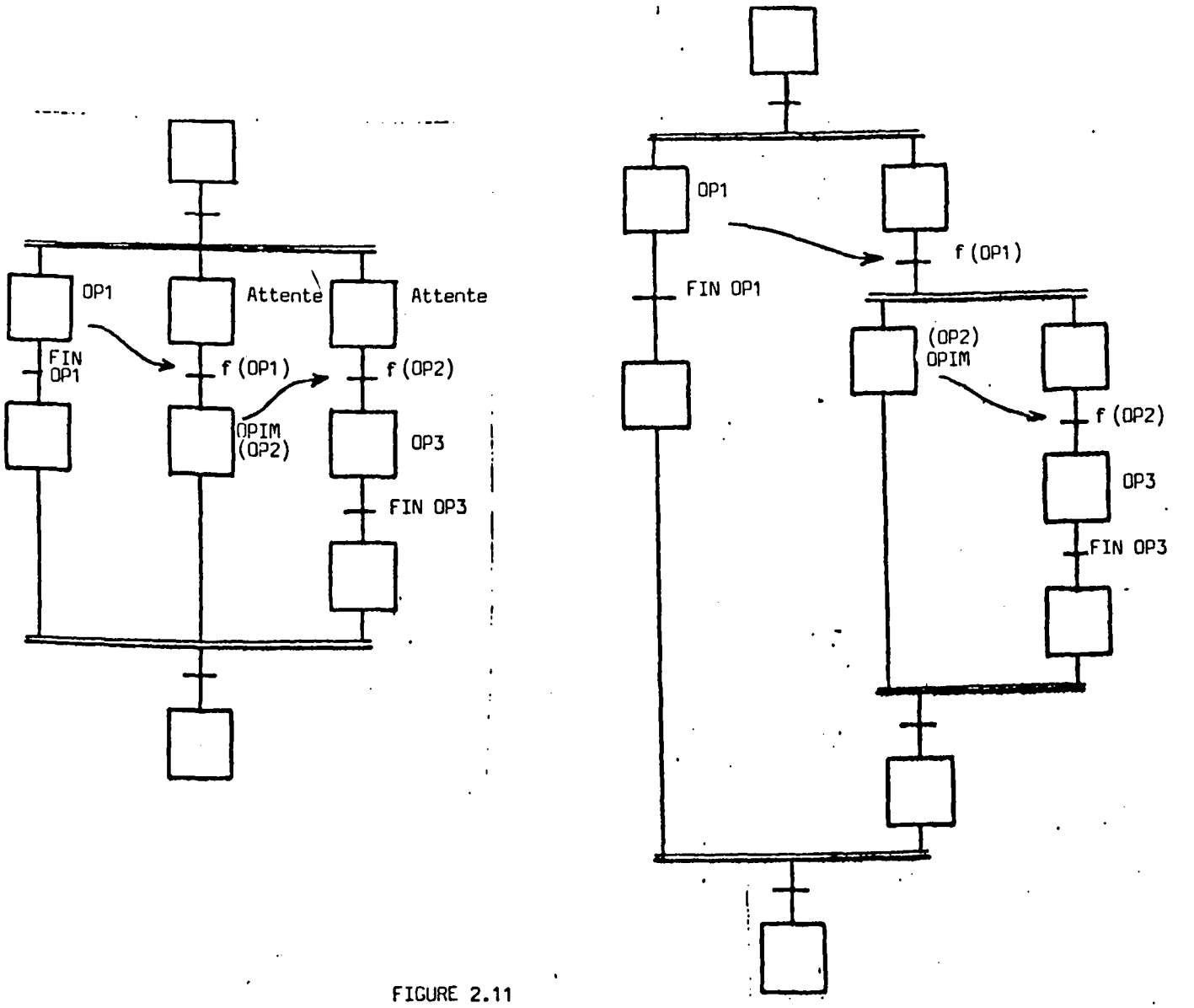


FIGURE 2.11

./..

Il est important de noter que les réceptivités f (OP1) et f (OP2) sont des fonctions logiques d'entrées de la partie commande liées au déroulement de OP1 ou de OP2. L'état des étapes correspondantes n'intervient en aucun cas dans ces fonctions.

c) Le parallélisme décrit implicitement

Dans ce cas le lien entre les différentes sous machines entrant dans le parallélisme se retrouve au niveau des réceptivités. C'est le cas typique des Grafcets couplés par l'activité des étapes. Ceci entraîne lors de l'implantation sur la machine d'exécution de régler de délicats problèmes de synchronisme. C'est pourquoi nous excluons ce type de parallélisme de notre langage.

2.8.3.3. Conséquences du parallélisme

a) Définition des structures d'échange entre opérations parallèles peut se faire suivant trois modes :

. mode sans validation

. mode avec validation

. mode interruptif.

Le premier peut s'appliquer pour transmettre par exemple un indicateur d'état entre phase et moniteur de phase. Généralement entre processus parallèles de même niveau, on préfère travailler avec validation. Le mode interruptif ne se justifie pas au niveau interne car toutes les branches en parallèles sont controlées à chaque cycle.

Dans le mode avec validation, l'échange d'information se fait par l'intermédiaire de messages pris en compte grâce à une file d'attente ou à une "boîte à lettres".

1 - File d'attente FIFO

On utilise les primitives classiques SEND (envoi) et WAIT (attente). SEND numéro message : est une opération qui a pour effet de mettre le message dans la file désignée par le numéro. En cas de saturation, on attend une place libre.

WAIT numéro : est une opération qui prend le message dans la file correspondant au numéro. En cas d'absence, on attend son arrivée. On peut éventuellement adjoindre une possibilité de forçage en tête de file.

2 - Boite à lettres
.....

Une boîte à lettres est une zone à adresse prédéfinie destinée à stocker un message. On lui associe un drapeau (flag).

L'emploi de SEND/WAIT peut ici s'envisager avec deux significations différentes.

Dans un premier cas, on inhibe SEND lorsque le drapeau est déjà à 1. La boîte à lettres est une file d'ordre 1.
La deuxième solution consiste, si le drapeau est déjà à 1, à écraser les anciennes informations.

Il est à noter que des opérations SET drapeau (mise à 1) et RESET drapeau (mise à zéro) peuvent être utilisées pour initialiser les drapeaux. Par ailleurs, un message ne doit pas contenir d'éléments d'évolution (réceptivité de sortie, état...) mais uniquement des éléments intervenant lors de sélection. Le message peut aussi être vide si on souhaite ne se servir que des drapeaux. Les éléments que nous venons de définir permettent d'effectuer la synchronisation et la coopération entre processus parallèles.

b) Le problème des ressources communes

Définition
.....

Un des problèmes induits par le parallélisme est celui des ressources communes. Il convient de régler deux difficultés :

- l'exclusion mutuelle

- l'arbitrage

2. Utilisation d'opérations partagées

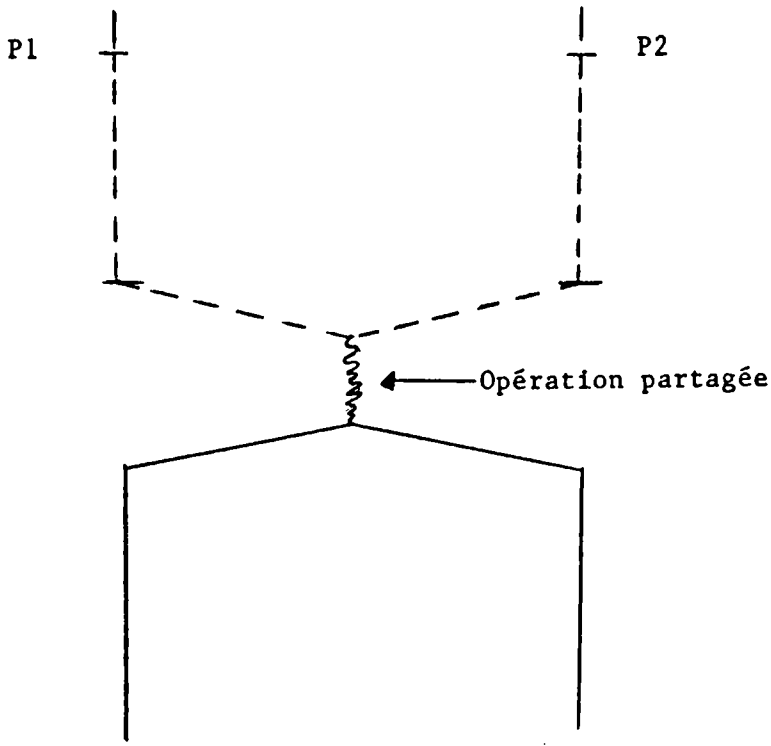


FIGURE 2.13

Dans ce cas, les processus P1 et P2 ont en commun une opération et donc toutes les variables qu'elle utilise.

L'exclusion mutuelle
.....

La notion de région critique permet de définir explicitement toutes les variables partagées.

Leur utilisation à un instant donné autorise ou non l'accès à une région critique.

L'accès
.....

Dans le cas de variables partagées, l'accès se fait simplement en fonction de l'ordre d'arrivée (premier arrivé-premier servi).

L'accès à une opération partagée peut se régler de plusieurs manières, en fonction des méthodes d'arbitrage.

1) Arbitrage à priorité fixée :

C'est une discipline simple mais imparfaite car elle peut provoquer soit la famine du processus, soit celle de la ressource.

Si par exemple : Trois processus partagent une même ressource. On donne à chaque processus (P1, P2, P3) une priorité différente.

Par contre, si P1 possède la priorité la plus haute (puis P2 et P3) et que P1 et P2 sont toujours demandeurs, P3 est en famine.

2) Arbitrage à priorité fixée avec contraintes :

Pour éviter la famine, on peut introduire des contraintes, par exemple sur le nombre de services. On associe donc à chaque processus un compteur.

c) Arbitrage à priorité dynamique :

L'arbitrage à priorité dynamique peut s'avérer plus efficace que celui à priorité fixée. Il s'agit cette fois d'organiser des structures d'attente soit du type FIFO (premier entré/premier sorti), soit à priorité relative circulaire (PRC : si un processus pose sa requête à son tour, il possède la priorité, sinon il devient le dernier de la file, personne ne possède une priorité plus haute que l'autre - la priorité est circulaire).

2.8.4. Paramétrage

Les processus par lot sont très souvent contrôlés relativement à une recette. Ceci revient à choisir en début de traitement, d'une part un certain nombre d'objets technologiques qui entrent en activité dans l'élaboration du produit, d'autre part un ensemble de valeur de paramètres. Ce choix se fait parmi un groupe de recettes prédéfinies.

Les opérations restent les mêmes tout au long de ces traitements sur le plan des structures : il suffit juste de modifier les éléments et les valeurs. Pour cela, les opérations doivent être paramétrées. Lors de la définition de l'opération, on précise les arguments formels du paramétrage. A l'usage de l'opération, on fixe les arguments réels qui doivent être cohérents en type avec ceux de la définition.

Toutefois, aux types classiques des langages de programmation, on adjoint le type fiche de paramètres permettant de sélectionner globalement un ensemble de paramètres.

Il convient de définir au préalable diverses structures de fiches et pour chacun des jeux de valeurs. Par ailleurs, il arrive fréquemment d'avoir à modifier des valeurs de variables dans une fiche de paramètres en cours d'exécution.

Si les paramètres se rapportant au matériel ne doivent pas être changés, ceux fixant certaines quantités (volume, poids, température...) peuvent être modifiés, certains à tout moment, d'autres à des instants déterminés. Il faut donc, lors de leurs définition, distinguer les paramètres accessibles.

Notons enfin la possibilité de définir des variables locales à une opération. Celles-ci doivent être déclarées explicitement. Toutes les variables non déclarées sont donc explicitement globales.

2.8.5. Description des niveaux supérieurs

Il n'y a pas de différences fondamentales entre la description de la phase et celle du moniteur de phase. Celui-ci a accès à une plus grande quantité d'informations et en particulier à l'état des opérations.

Par ailleurs, le moniteur de phase doit déterminer pour le moniteur général l'état de la phase parmi les cas suivants :

- Inactive
- Active en position d'attente avant la préparation du contexte.
- Active pendant la préparation du contexte.
- Active en attente de phase dynamique.
- Active en phase dynamique.
- Active terminée avec maintien du contexte.

De plus, on y adjoint les indications de cas anormaux :

- Anomalies de maintien du contexte.
- Anomalies de préparation du contexte.
- Anomalies de phase dynamique.
- Anomalies de blocage en position d'attente (chien de garde temporel).
- Anomalies de blocage en position de phase terminée (chien de garde temporel).

Il n'y a pas de discontinuité entre la description des opérations et celle des moniteurs, il y a changement de nature d'informations. Pour marquer cette différence, nous sommes obligés d'introduire la notion d'opération moniteur OPEM et d'opération moniteur général DPEC. Dans une OPEM, on peut introduire dans les expressions des états d'opération des commandes de marche et d'arrêt issues du moniteur général ; on peut utiliser comme actions des forçages, des déblocages de point d'arrêt, des calculs d'état de phase, etc...

Le moniteur général a accès à l'état des phases qu'il peut introduire dans des expressions, de même que des variables d'intervention de l'opérateur (clé opérateur, mode de marche et de forçage, mode d'arrêt...).

Les actions peuvent être des indications de modes dans les phases, des forçages ou des choix de phases, des comptes rendus à l'opérateur, etc...

2.9. CONCLUSION

Ce chapitre donne tous les éléments de base nécessaires à la définition du poste d'Automatisation et d'Instrumentation. De nombreux points ont pu être confrontés à des cahiers des charges industriels et donnent satisfaction.

Ce qui concerne, en particulier les parties "moniteur" et "moniteur général" reste actuellement à l'état de suggestions et devra être testé sur des exemples.

Il convient maintenant de placer un utilisateur devant son poste de travail : ceci fait l'objet du chapitre suivant.

CHAPITRE III

LE CONCEPTEUR D'AUTOMATISME DEVANT SON POSTE DE TRAVAIL

S O M M A I R E

3.1. INTRODUCTION

Dans ce chapitre nous allons envisager l'usage du poste d'automatisation et d'instrumentation sous deux aspects. Nous avons vu au chapitre précédent que la description se fait sous forme semi-naturelle mais qu'il est souhaitable pour faciliter une visualisation plus globale de disposer d'un mode graphique de représentation de l'arborescence. Une première partie lui sera consacrée. Ensuite, nous plaçons l'utilisateur devant son poste de travail et exposons sur un exemple les procédures de description.

3.2. VISUALISATION GRAPHIQUE DE L'ARBORESCENCE

3.2.1. Introduction

L'objectif est de donner à l'utilisateur un moyen de visualisation graphique de la structure et du contenu de la description par déplacement dans l'arborescence.

Dans une première étape, cet outil de visualisation constituera une aide indispensable à la description et complétera ainsi le poste de travail du concepteur de l'automatisme.

Dans une seconde étape, il permettra la visualisation dynamique temps réel du déroulement d'un programme d'automatisme et s'intégrera donc dans les fonctions d'un poste de conduite pour aider au suivi et à la compréhension du comportement du processus automatisé. L'aspect ergonomique de cet outil de visualisation graphique est primordial.

3.2.2. Cahier des charges.

3.2.2.1. Contraintes ergonomiques

L'utilitaire de visualisation doit permettre de se positionner en un point quelconque de l'arborescence et de visualiser la structure des sous-niveaux vus de ce point : parallélisme, séquence etc.

Selon la demande de l'utilisateur, une des branches pourra être détaillée à son tour et ainsi de suite. Le défilement des opérations d'un même niveau à l'intérieur d'une branche devra être autorisé.

Pour une procédure inverse, l'utilisateur pourra remonter dans la structure.

La visualisation doit permettre d'atteindre les feuilles terminales, c'est-à-dire les primitives du langage décrites dans la nomenclature. La possibilité d'une modification de la structure interne d'une opération pourra être envisagée, à partir du positionnement dans l'arborescence.

3.2.2.2. Objectif

Toutes les possibilités offertes par les outils de visualisation (couleur, animation, etc.) seront exploitées pour apporter un maximum de confort visuel.

L'interactivité homme/machine devra s'apparenter, voire se conformer au menu développé dans l'éditeur syntaxique utilisé pour l'introduction du langage de description.

Le graphisme et la couleur seront associés afin de préciser le type et la nature de chaque opération affichée.

Toutefois, le choix des couleurs devra autoriser une mise en oeuvre sur moniteur noir et blanc.

A tout moment, l'utilisateur connaîtra sa position dans l'arborescence à l'aide d'un repère de lecture (clignotements, curseur, sigle, etc.).

D'autre part, lors des demandes de création/déplacements dans l'arborescence, celui-ci éprouvera le moins possible la sensation d'attente. Une attention particulière sera portée au niveau du dialogue opérateur : guide opérateur, enchaînement des commandes, etc.

3.2.3. Graphisme.

La visualisation d'une opération doit apporter les 3 caractéristiques suivantes :

. son NOM qui comportera au maximum 17 caractères,

. son TYPE (parallélisme, séquence, itération, alternative),

. sa NATURE (opération générale, élémentaire, implicite, élémentaire-implicite).

La préférence sera donnée à la création de sigles pour la représentation des différents types alors que la couleur sera envisagée pour définir la nature.

L'opération est visualisée sur l'écran par un encadré en 2 parties (figure 3.1).

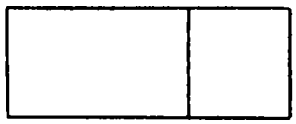


Figure 3.1

Dans la première partie apparaissent les types dont les signes sont (figure 3.2) :

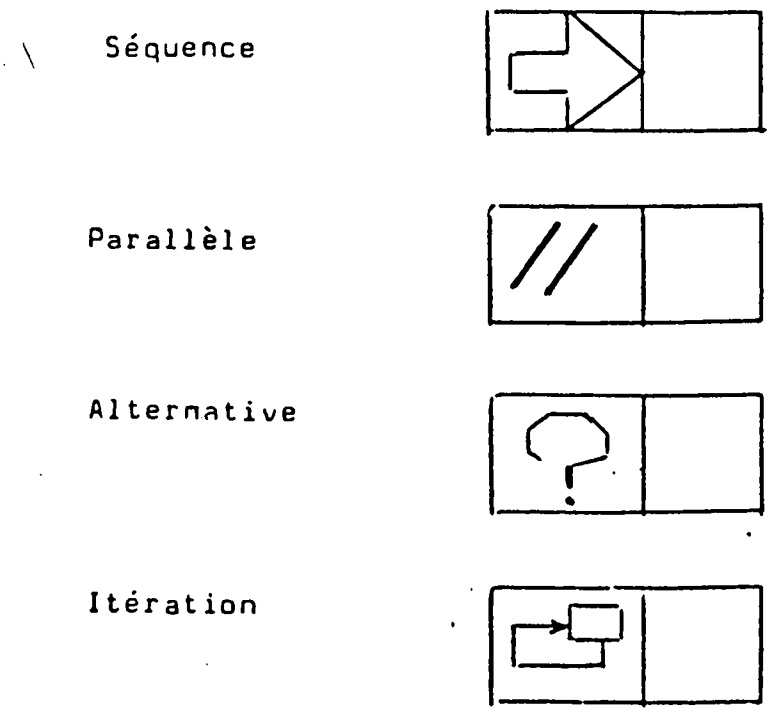
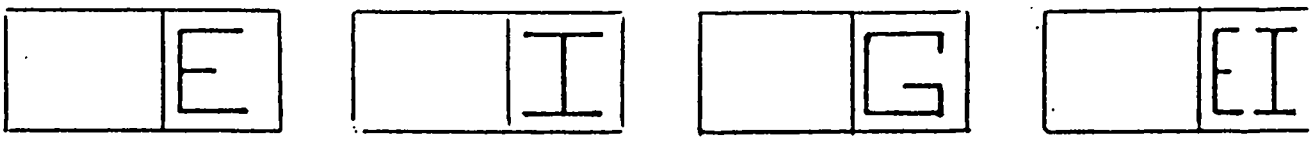


Figure 3.2

La seconde partie est réservée au caractère donnant la nature de l'opération (figure 3.3) :



Elémentaire Implicite Générale Elémentaire Implicite

Figure 3.3

La couleur apportée en supplément se définit comme suit : une couleur pour le fond, une autre pour le sigle du type.

Ce qui donne le tableau suivant (figure 3.4) :

	COULEUR FOND	COULEUR SIGLE
GENERALE	VERT	ROUGE
ELEMENTAIRE	CYAN	BLEU
IMPLICITE	BLEU	JAUNE
EL-IMPLICITE	JAUNE	BLEU

Figure 3.4

3.2.4 Représentation

Pour répondre aux contraintes de taille écran et du nombre maximum d'informations respectant la lisibilité, on a choisi une représentation en mémoire (figure 3.5) apportant les informations suivantes :

. une opération courante,

. une opération PERE de l'opération courante,

. deux opérations FRERE (de l'opération courante),

. trois opérations FILS (de l'opération courante),

. neuf opérations PETIT-FILS (de l'opération courante),

. les liens les unissant et le menu.

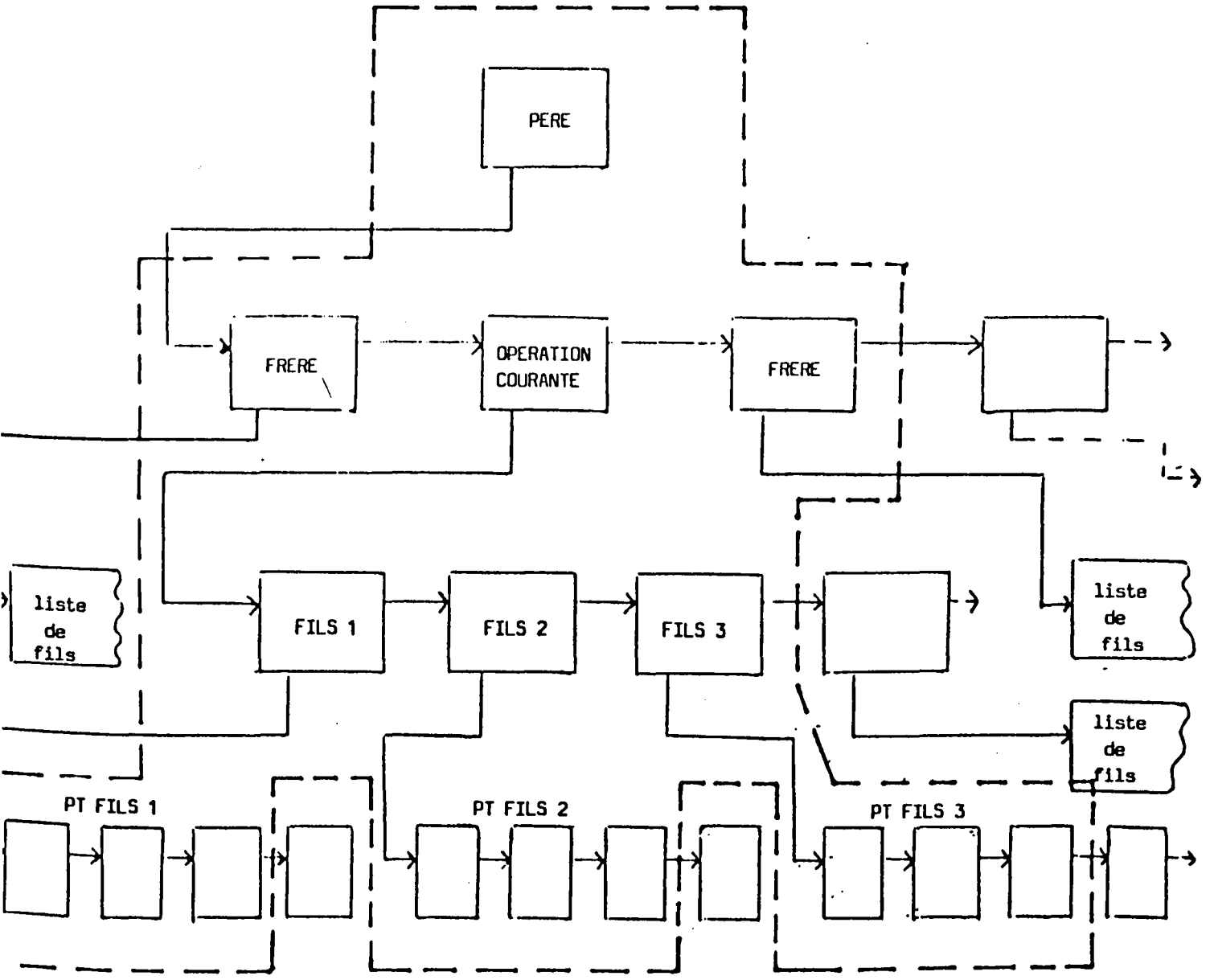
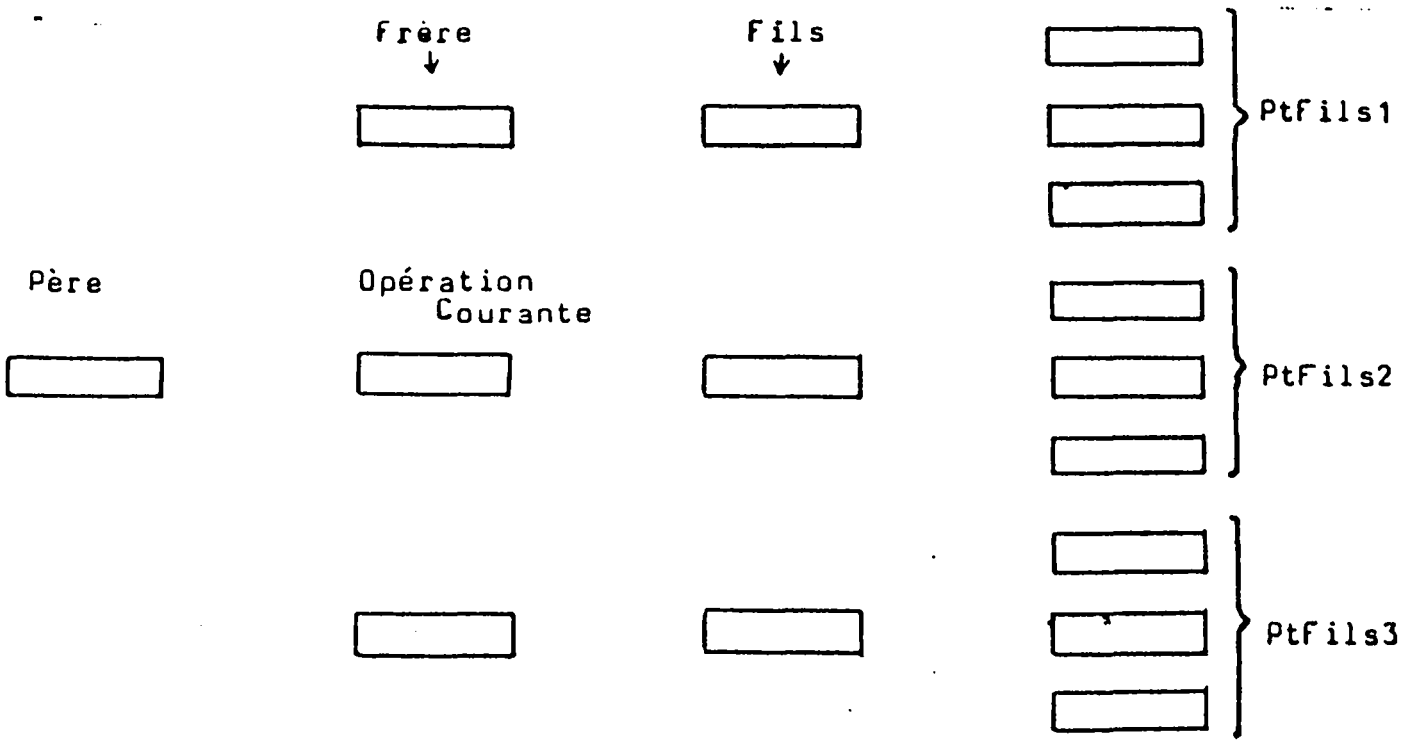


Figure 3.5

./..

Ce qui donne à l'écran (figure 3.6) :



1 : REMONTE 2 : PT FILS (1) 3 : PT FILS (2) 4 : PT FILS (3)

5 : FRERE 6 : FILS 7 : ARBO/EDIT

Figure 3.6

3.2.5 Déplacements

Après avoir choisi "ARBORESCENCE" dans le menu édition, le programme propose un père, c'est-à-dire la première opération sans ascendant qu'il aura trouvé en lisant le catalogue. Il est possible alors de faire défiler la liste des opérations PERE en appuyant sur N.

Après une réponse positive, le programme va demander de confirmer si c'est bien cette opération que l'on souhaite examiner. Dans le cas contraire, il attend un nom rentré au clavier. Si le nom n'existe pas, il repart au début et propose à nouveau la liste des PERES.

On peut décider d'examiner une opération quelconque dans l'arborescence. Pour ce faire, il suffit de taper "O" à la demande de PERE, puis "N" à la confirmation et de rentrer ensuite le nom désiré. Cette démarche peut poser un problème lors du déplacement dans l'arbre. En effet, si l'on souhaite examiner le PERE d'une opération utile en plusieurs endroits, le programme va demander de lui en désigner un, pour qu'il puisse rechercher les FRERES de l'opération.

3.2.5.1. Déplacement local.

Etant positionné en lecture dans une liste donnée, l'utilisateur peut faire défiler cette liste, soit en avant, soit en arrière. Suivant le positionnement dans la représentation, ce défilement entraîne des modifications plus ou moins importantes au niveau du contenu de l'écran : une lecture dans les feuilles (PETIT-FILS) de la représentation ne fera évoluer que les feuilles elles-mêmes, alors qu'une lecture au niveau des FRERES transformera, à l'exception du père, la totalité de la représentation. Chaque lecture affecte d'une manière particulière la représentation et donc la disposition des pointeurs de lecture. Cela entraîne la nécessité de réaliser une procédure de lecture différente pour chaque liste représentée.

3.2.5.2. Déplacement global

Il existe deux concepts de déplacements entraînant une transformation totale de la représentation :

. Le premier est la remontée dans l'arborescence : le PERE de l'opération courante remplace alors l'opération courante.

. Le second est la descente : étant en lecture dans une liste quelconque, l'opérateur décide d'explorer la partie inférieure de l'arborescence. Pour ce faire, il a à sa disposition la commande COURANT.En validant cet ordre, l'utilisateur demande que l'opération sur laquelle il était situé en lecture devienne l'opération courante.

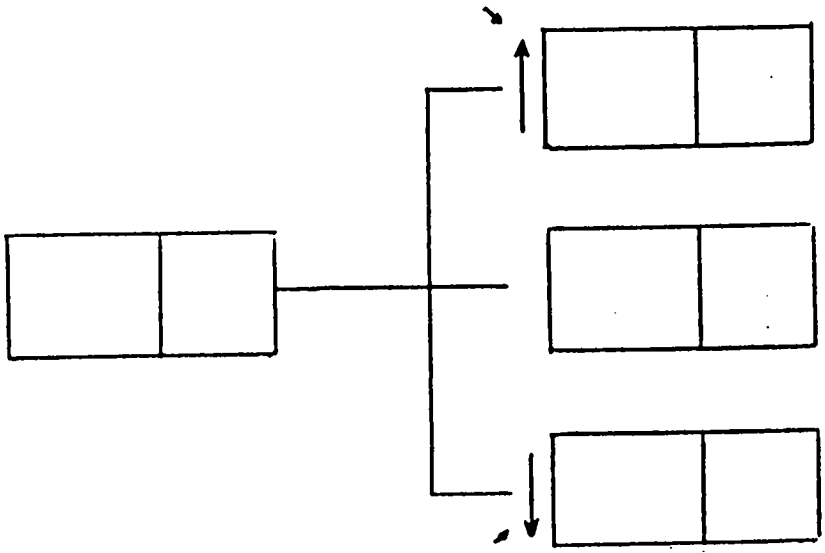
On remarque qu'une lecture dans la liste des FRERES constitue une demande implicite de variation d'opération courante.

Chaque déplacement global, que ce soit la remontée ou le changement d'opération courante, nécessite une translation de la totalité des pointeurs de lecture à l'intérieur de l'arborescence dynamique, alors qu'un déplacement local n'affecte que partiellement la valeur des pointeurs.

La représentation est évidemment incomplète. Rien dans le cahier des charges ne limitant le nombre de fils d'une opération, il faut penser à signaler si ce nombre est supérieur à trois.

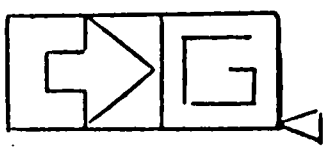
C'est pour cette raison qu'apparaîtra parfois une flèche devant une opération (Figure 3.7).

Figure 3.7



Il reste pour une meilleure compréhension, à signaler quelle est l'opération en cours d'examen. Pour ce fait, un sigle va se placer au bas de l'opération (Figure 3.8).

Figure 3.8



Remarque : REPRESENTATION SPECIALE

2 solutions se présentent : lui donner un nom et le déclarer à part
: utiliser les sous-structures, c'est-à-dire :

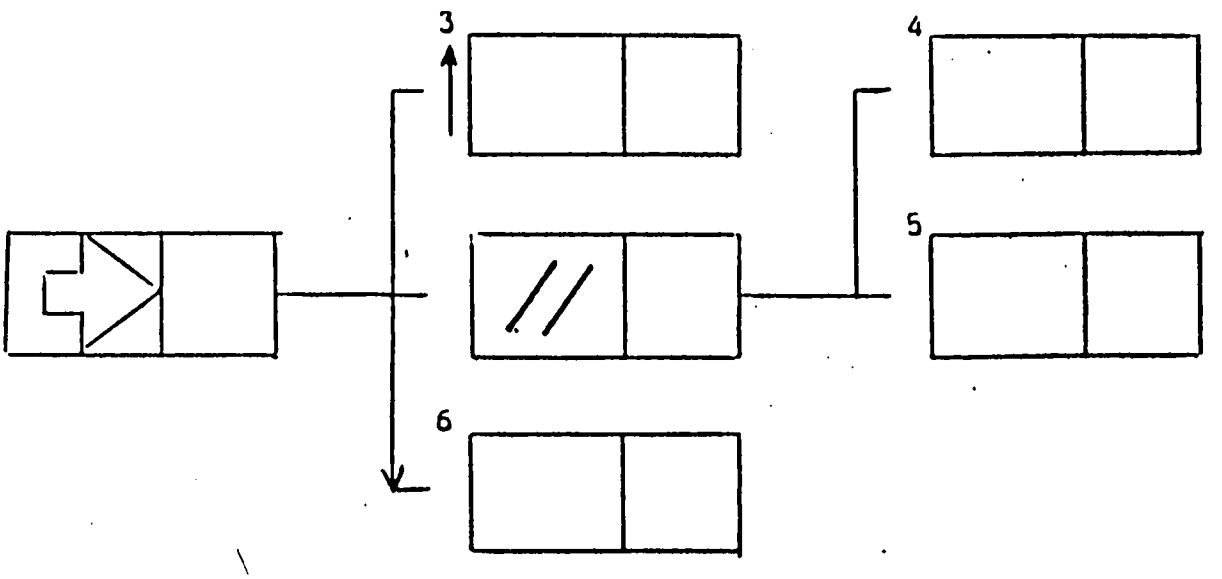


Figure 3.9

On a ainsi défini une nouvelle opération qui a un type et une nature mais pas de nom. Elle s'écrira donc comme tel à l'écran.

3.3. L'OPERATEUR FACE AU POSTE D'AUTOMATISATION ET D'INSTRUMENTATION (P.A.I)

3.3.1. Introduction

Automatiser consiste à contrôler et surveiller un certain nombre d'ojets technologiques agissant sur la matière d'oeuvre. Ces objets sont décrits dans une nomenclature et, comme ils se retrouvent très souvent, on utilise la notion de modèle, comme il a été défini au chapitre précédent. On dispose ainsi d'une base générale des modèles (B.G.M.) qui s'enrichit avec l'expérience.

Pour passer à une application particulière, on extrait de cette base les modèles utiles et on constitue la base des modèles de l'application i (B.M.A.i.). Celle-ci peut d'ailleurs être complétée par quelques modèles spécifiques. Le passage à la description du système de commande : base de description de la commande B.D.C.i. nécessite l'instantiation des modèles de l'application i, ce qui nous donne la base des modèles utiles (B.M.U.i.). Le concepteur d'automatisme, face à son poste d'automatisation et d'instrumentation, dispose donc d'un certain nombre de ces bases et son objectif est de les modifier, d'accéder à leur contenu, d'effectuer des suppressions, des transferts entre bases (figure 3.10.).

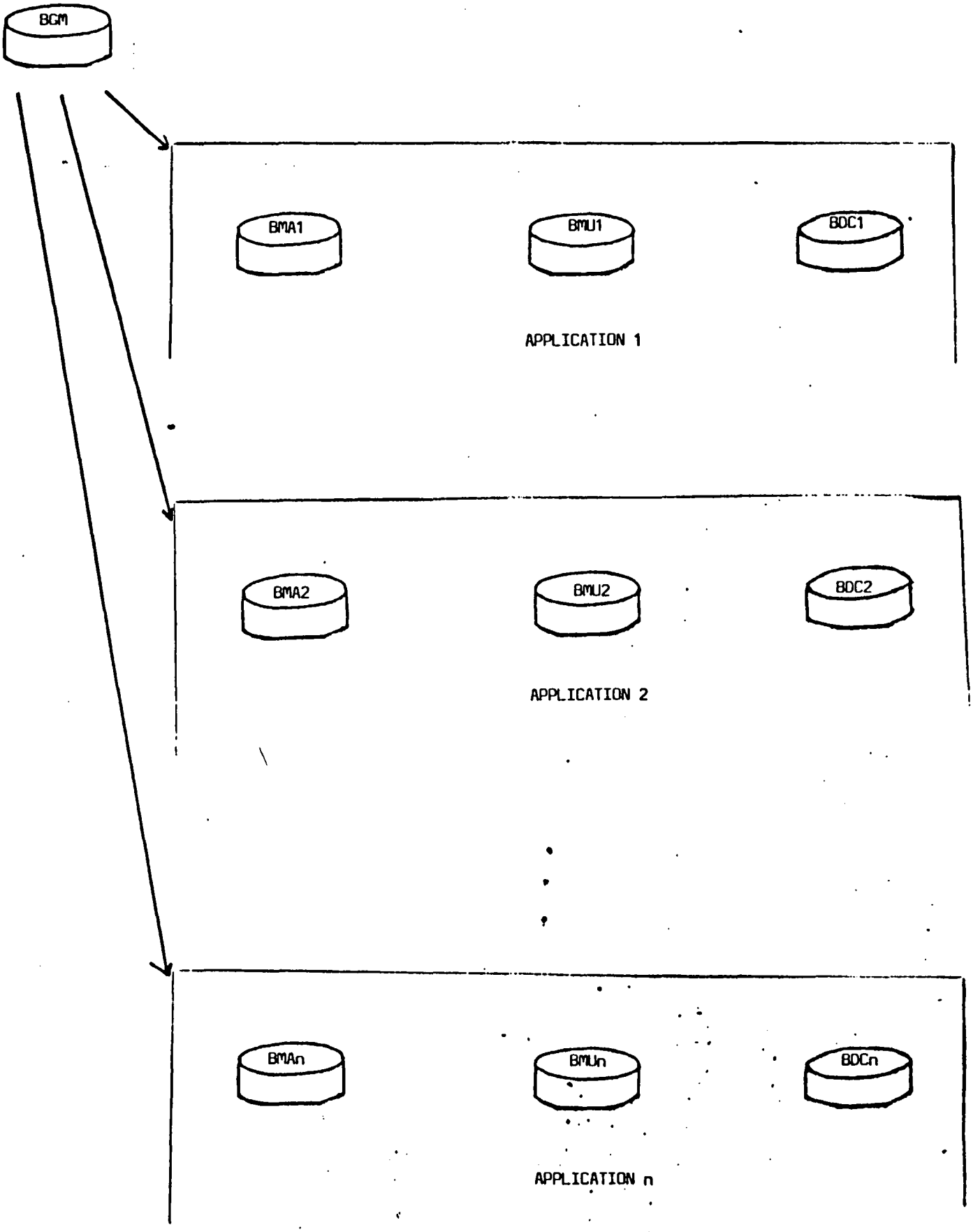

BGM
BMA1
BMU1
BDC1
APPLICATION 1
BMA2
BMU2
BDC2
APPLICATION 2
BMAn
BMUn
BDCn
APPLICATION n

Figure 3.10

./..

Nous allons maintenant montrer sur une application comment son utilisateur peut décrire ses modèles, les modules correspondants et l'ensemble des opérations.

Nous supposons pour cela B.M.A. vide et nous n'utilisons pas de modèles prédéfinis dans B.G.M.

On peut résumer les différentes opérations applicables à chaque base :

sur B.G.M. on peut : ajouter ; supprimer ; transférer depuis B.M.A. : obtenir la liste des modèles (noms et libellés) ; lister ou imprimer un ou plusieurs modèles ;

sur B.M.A.i. on peut : transférer depuis B.G.M. ; ajouter, supprimer ou modifier un modèle ; obtenir la liste des modèles, lister ou imprimer un ou plusieurs modèles ;

sur B.M.V.i. on peut : instantier à partir de B.M.A.i. ; modifier l'instantiation : ajouter, modifier ou supprimer des modules ; obtenir le listing d'instantiation ;

sur B.D.C.i. on peut : obtenir la liste des opérations ; ajouter, supprimer ou modifier une opération ; lister ou imprimer : une opération, une liste d'opérations, un ensemble d'opérations (arborescence correspondante) ; transférer une opération et ses conséquents depuis une autre B.D.C.i. ; accéder à la représentation graphique.

3.3.2. Définition de l'installation

Considérons figure 3.11. un processus de critallisation comportant un réacteur RA 101 avec ses circuits d'alimentation et ses annexes (agitation, vapeur...).

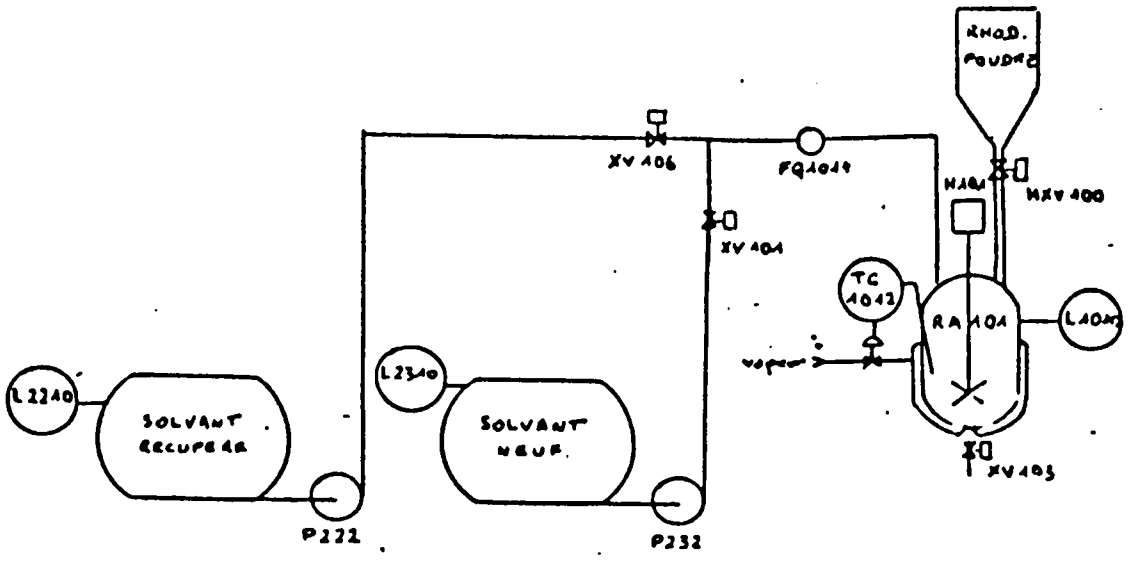


FIGURE 3.11.

./..

Le mode opératoire peut être schématiser de la façon suivante :

La première étape consiste en la mise en route de l'agitation du réacteur RA101 à partir d'un seuil de niveau de ce réacteur. Puis, on procède au chargement d'une quantité Q1 de solvant récupéré. Ensuite, on complète éventuellement avec du solvant neuf. Dès qu'on reçoit l'autorisation on charge le produit solide. Le chauffage programmé du réacteur RA101 est lancé. On procède au maintien en température pendant un temps donné t1, pour effectuer la dissolution.

On refroidit ensuite jusqu'à un seuil de température. Puis, on maintient l'agitation pendant un temps donné t2, pour effectuer la cristallisation.

3.3.3. Définition de la base des modèles de l'application

Nous commençons par rechercher les objets technologiques à contrôler. On en effectue la liste en les regroupant par famille modélisable. On obtient ainsi la classification de la figure 3.12.

NOM D'OBJET	MODELE
XV 101	Vanne FMA
XV 103	Vanne FMA
XV 106	Vanne FMA
HXV 100	Vanne MANU
P 222	Moteur
P 232	Moteur
N 101	Moteur
FQ 1014	Compteur FL
TI 1012	Mesure
TI 1013	Mesure
LI 1010	Mesure
LI 2210	Mesure
LI 2310	Mesure
TC 1012	REG EXT
TC 1013	REG EXT

Figure 3.12

Ceci maintient nous permet d'aborder la description des modèles de fonction d'interface. Les deux exemples proposés ci-dessous (figures 3.13 et 3.14) concernent d'une part une vanne fermée par manque d'air et d'autre part un poste de mesure avec seuils. Les opérations qui y sont référencés sont décrites dans la base B.C.D.

MODELE	LIBELLE
VANNE [illegible]	vanne [illegible] fermeture par manque d'air

INTERFACE PARTIE OPERATIVE - LIGNES E/S

NOM_E/S	NAT	LIBELLE	TYPE	ACC	OPERATION	PARAMETRES
SAO	[illegible]	sortie ordre de commande	mono			
FCO	entree	fin de course d'ouverture	mono			
FCF	entree	fin de course de fermeture	mono			

INTERFACE PARTIE COMMANDE - LIGNES CMDE OU ETAT

NOM_C/E	NAT	TYPE	LIBELLE	OPERATION	PARAMETRES
OUVERTE	commde	mono			
OUVERTE	etat	mono			
FERMEE	etat	mono			
DEFAUT	etat	mono			

OPERATIONS A EXECUTION CYCLIQUE

NOM_OPE	PARAMETRES
[illegible]	

Figure 3.13

MODELE	LIBELLE
MESURE	poste de mesure avec seuils

. CONSTANTES CARACTERISTIQUES

NOM_CTE	LIBELLE	TYPE
DIFF	differentiel de seuil	reme

INTERFACE PARTIE OPERATIVE - LIGNES E/S

NOM_E/S	NAT	LIBELLE	TYPE	ACC	OPERATION	PARAMETRES
MESU	entre	entree de mesure analogique	reme	lec		

INTERFACE PARTIE COMMANDE - LIGNES CMDE OU ETAT

NOM_C/E	NAT	TYPE	LIBELLE	OPERATION	PARAMETRES
SH	commde	reme			
SB	commde	reme			
ALH	etat	mono			
ALB	etat	mono			

OPERATIONS A EXECUTION CYCLIQUE

NOM_OPE	PARAMETRES
ALARMB	
ALARMH	

Figure 3.14

./..

Nous introduisons aussi dans la base des modèles la description de certaines fonctions opératives, importantes pour décrire la partie commande. Nous présentons (fig. 3.15 et 3.16) un modèle de compteur et un modèle de temporisation.

MODELE	LIBELLE
COMPTEUR1	compteur sur mesure de debit lineaire

INTERFACE PARTIE OPERATIVE - LIGNES E/S

NOM_E/S	NAT	LIBELLE	TYPE	ACC	OPERATION	PARAMETRES
EMML	entre	entree de mesure de debit lineaire	nume			
BT1	entre	base de temps a la seconde	mono			

INTERFACE PARTIE COMMANDE - LIGNES CMDE OU ETAT

NOM_C/E	NAT	TYPE	LIBELLE	OPERATION	PARAMETRES
VALCOUR	etat	nume			
PRED	cmde	nume	Valeur de predetermination		
FCOMPT	etat	bool			

OPERATIONS A EXECUTION CYCLIQUE

NOM_OPE	PARAMETRES
FINCOMPT FL	

Figure 3.15

MODELE	LIBELLE
TEMPO	temporisation

INTERFACE PARTIE OPERATIVE - LIGNES E/S

NOM_E/S	NAT	LIBELLE	TYPE	ACC	OPERATION	PARAMETRES
BT1	entre	base de temps a la seconde	mono			
BT10	entre	base de temps au 1/10 de seconde	mono			
BT100	entre	base de temps au 1/100 de seconde	mono			

INTERFACE PARTIE COMMANDE - LIGNES CMDE OU ETAT

NOM_C/E	NAT	TYPE	LIBELLE	OPERATION	PARAMETRES
VALCOUR	etat	nume			
PREDET	cmmde	nume			
FTEMPO	etat	bool			
BT2	cmmde	mono		INIT TEMPO	
COMPTER1	cmmde	mono			
COMPTER10	cmmde	mono			
COMPTER100	cmmde	mono			

OPERATIONS A EXECUTION CYCLIQUE

NOM_OPE	PARAMETRES
FIN TEMPO	
COMPTAGE TEMPO1	
COMPTAGE TEMPO10	
COMPTAGE TEMPO100	

Figure 3.16

./..

3.3.4. Instantiation des modèles (base des modules de l'application B.M.U.)

L'installation comporte plusieurs vannes F.M.A., plusieurs moteurs... Il convient donc maintenant de définir chacune de ces entités en faisant l'instantiation des modèles correspondants.

Pour un code modèle donné, par exemple vanne F.M.A., on définit un module par son repère, un libellé exprimant en clair son rôle et par son usage soit normal soit comme ressource partagée.

Pour l'ensemble des objets technologiques et des fonctions opératives, on obtient respectivement les listes des figures 3.17 et 3.18.

LISTE DES OBJETS TECHNOLOGIQUES

CODE MODELE	REFERE	LIBELLE	USAGE
VANNE FMA	XV101	CHARGE DE SOLVANT NEUF DANS RA101	N
VANNE FMA	XV103	VIDANGE DE RA101	N
VANNE FMA	XV10b	CHARGE DE SOLVANT RECUPERE DANS RA101	N
MESURE	L11010	NIVEAU DANS RA101	N
MESURE	L12210	NIVEAU CITERNE SOLVANT RECUPERE	F
MESURE	L12310	NIVEAU CITERNE SOLVANT NEUF	F
MESURE	T11012	TEMPERATURE DANS RA101	N
MOTEUR	N101	AGITATION DE RA101	N
MOTEUR	P222	TRANSFERT DE SOLVANT RECUPERE	N
MOTEUR	P232	TRANSFERT DE SOLVANT NEUF	N
SIGNAL	POUDRE	CHARGEMENT DE POUDRE DANS RA101	N

FIGURE 3.17

COMPT FL	FQ1014/A	COMPTAGE DE SOLVANT RECUPERE DANS RA101	N
COMPT FL	FQ1014/B	COMPTAGE DE SOLVANT NEUF DANS RA101	N
VANNE H	HXV100	CHARGE DE POUDRE DANS RA101	N
TEMPORISE	TEMPO.100	FIN DE DISSOLUTION DANS RA101	N
RAMPE	RAMPE1	CHAUFFAGE PROGRAMME DERA101	N
REGEAT	TC1012	REGULATION DU CHAUFFAGE DE RA101	N

FIGURE 3.18

3.3.5. Définition de la base de description (B.D.C.)

3.3.5.1. Principe

La définition de la base de description suit les principes énoncés au Chapitre II.

On les applique aux différentes Fonctions d'Interface, Fonctions Opératives et à la description de la Partie Commande.

Les opérations élémentaires, feuilles de l'arborescence, se réfèrent à la Base des Modèles Utiles (B.M.U.).

Celle-ci nous donne les noms des actions utilisables et les lignes d'informations correspondant à des compte-rendus d'actions.

3.3.5.2. Saisie

L'introduction de ces modèles se fait par l'intermédiaire d'un éditeur syntaxique. Il propose d'une part des enchaînements de menus (il suffit alors de sélectionner une des fonctions proposées dans chacun d'eux), d'autre part, il sert de guide lors de l'introduction des éléments d'une opération. Il propose des mots clés et un contrôle sur les éléments introduits par l'utilisateur.

Prenons comme exemple la saisie de l'opération élémentaire AGIT.RA101. La démarche de l'opérateur pour créer cette opération peut se décrire ainsi : après avoir sélectionner OPERATION dans le premier menu, il précise son choix grâce au deuxième menu (Touche opération élémentaire).

Il doit alors apparaître le premier mot clé * OPELEM : (figure 3.19). Il suffit alors de poursuivre l'introduction des autres éléments de cette opération. On obtient la description complète de AGIT.RA101 (figure 3.19) où les éléments prédéfinis sont soulignés.

```
* OPELEM        :AGIT.RA101
  TYPE          :S
  LIBELLE       AGITATION DE RA101

  ATT JSQ       MESU LI1010>NIV AGIT RA101
  FAIRE
                LANCER N101;
  JSQ             FPARALL

) FOPEL         :AGIT.RA101
```

FIGURE 3.19

3.3.5.3. Application

On introduit quelques fonctions de l'application RHODIA, de la manière indiquée au paragraphe 3.3.5.2.

L'analyse d'une application se fait suivant une procédure descendante.
Par contre, l'introduction est montante.

On va donc décrire en premier lieu les opérations associées aux fonctions d'Interface (Figures 3.20 - 3.21), aux Fonctions opératives (Figures 3.22 - 3.23) ; puis on introduit les opérations élémentaires (Figures 3.24 - 3.25) ; et enfin, les opérations (Figures 3.26 - 3.27).

MODELE : VANNE FMA

```
* OPELEM      :ETAT VANNE
  TYPE        :S
  LIBELLE     OPERATION SYNCHRONE DU MODELE VANNE FMA

      SI              OUVRIR
           ALORS
            SRC;
      FSI
      SI              FCO & /FCF
           ALORS
            OUVERTE;
      FSI
      SI              FCF &/FCO
           ALORS
            FERMEE;
      FSI
      SI              (OUVRIR & /FCO) ! (/OUVRIR & /FCF)
           ALORS
            DISCORD;
      FSI

  FOPEL       :ETAT VANNE
```

FIGURE 3.20

MODELE : MESURE

```
* OPELEM        : ALARMB
  TYPE          : S
  LIBELLE       OPERATION SYNCHRONE DU MODELE MESURE

     SI              MESU <SB ! (ALB & (MESU< (SB+DIFF))
          ALORS
           ALB;
     FSI

> FOPEL          : ALARMB
```

```
* OPELEM        : ALARMH
  TYPE          : S
  LIBELLE       OPERATION SYNCHRONE DU MODELE MESURE

     SI              MESU >SH ' (ALH & (MESU >(SH-DIFF)))
          ALORS
           ALH;
     FSI

> FOPEL          : ALARMH
```

Figure 3.21

MODELE : TEMPO

```
* OPELEM      : INIT TEMPO
  TYPE        : S
  LIBELLE     OPERATION ASYNCHRONE DU MODELE TEMPO

      FTEMPO: =0;
      VALCOUR: =0;

> FOPEL       : INIT TEMPO
```

```
* OPELEM      : FIN TEMPO
  TYPE        : S
  LIBELLE     OPERATION SYNCHRONE DU MODELE TEMPO

    SI              VALCOUR >= PREDET
       ALORS
        FTEMPO: =1;
    FSI

> FOPEL       : FIN TEMPO
```

```
* OPELEM      : COMPTAGE TEMPO1
  TYPE        : S
  LIBELLE     OPERATION SYNCHRONE DU MODELE TEMPO

    SI              COMPTER1 & BT1 & (VALCOUR < PREDET)
       ALORS
        VALCOUR: =VALCOUR + 1;
    FSI

> FOPEL       : COMPTAGE TEMPO1
```

```
* OPELEM      : COMPTAGE TEMPO10
  TYPE        : S
  LIBELLE     OPERATION SYNCHRONE DU MODELE TEMPO

    SI              COMPTER10 & BT10 & (VALCOUR < PREDET)
       ALORS
        VALCOUR: =VALCOUR + 1;
    FSI

> FOPEL       : COMPTAGE TEMPO10
```

```
* OPELEM      : COMPTAGE TEMPO100
  TYPE        : S
  LIBELLE     OPERATION SYNCHRONE DU MODELE TEMPO

    SI              COMPTER100 & BT100 & (VALCOUR < PREDET
       ALORS
        VALCOUR: =VALCOUR + 1;
    FSI

> FOPEL       : COMPTAGE TEMPO100
```

Figure 3.22

./..

MODELE : COMPTEUR FL

```
* OPELEM        :INITCOMPT FL
  TYPE          :S
  LIBELLE       OPERATION ASYNCHRONE DU MODELE COMPTEUR FL

      VALCOUR:=0;
      FCOMPT:=0;

> FOPEL         :INITCOMPT FL
```

```
* OPELEM        :FINCOMPT FL
  TYPE          :S
  LIBELLE       OPERATION SYNCHRONE DU MODELE COMPTEUR FL

    SI              VALCOUR >= PREDET
       ALORS
        FCOMPT:=1;
    FSI

> FOPEL         :FINCOMPT FL
```

```
* OPELEM        :COMPTAGE FL
  TYPE          :S
  LIBELLE       OPERATION ASYNCHRONE DU MODELE COMPTEUR FL

    SI              BT1 & (VALCOUR < PREDET)
       ALORS
        VALCOUR:=VALCOUR + (EAML/3600);
    FSI

> FOPEL         :COMPTAGE FL
```

Figure 3.23

DESCRIPTION DES OPERATIONS ELEMENTAIRES DU PROCEDE

```
$ OPELEM        :AGIT.RA101
  TYPE          :S
  LIBELLE       AGITATION DE RA101

 ATT JSQ       MESU LI1010>NIV AGIT RA101
 FAIRE
          LANCER N101;
 JSQ           FFARALL

> FOPEL         :AGIT RA101
```

```
$ OPELEM        :SOLVREC.RA101
  TYPE          :S
  LIBELLE       CHARGE DE SOLVANT RECUPERE DANS RA101

 FAIRE
          SI              XV101 FERMEE & XV103 FERMEE & /ALH LI1010
              ALORS
               OUVRIR XV106;
          FSI
          SI              XV106 OUVERTE & /ALB LI2210
              ALORS
               LANCER F222;
          FSI
          SI              F222 MARCHE
              ALORS
               COMPTER FQ1014(Q1);
          FSI
  JSQ          FCOMPT FQ1014 ! ALB LI2210

> FOPEL         :SOLVREC RA101
```

```
$ OPELEM        :SOLVNEUF.RA101
  TYPE          :S
  LIBELLE       CHARGE DE SOLVANT NEUF DANS RA101

 FAIRE
          SI              XV106 FERMEE & XV103 FERMEE & /ALH LI1010
              ALORS
               OUVRIR XV101;
          FSI
          SI              XV101 OUVERTE & /ALB LI2310
              ALORS
               LANCER F232;
          FSI
          SI              F232 MARCHE
              ALORS
               COMPTER FQ1014(Q1);
          FSI
  JSQ          FCOMPT FQ1014

> FOPEL         :SOLVNEUF RA101
```

Figure 3.24

./..

```
* OPELEM      :POUDRE.RA101
  TYPE        :S
  LIBELLE     AUTORISATION DE CHARGEMENT DE POUDRE DANS RA101

 FAIRE
        SI          (MESU LI1010>NIV SOLV RA101) & N101 MARCHE
            ALORS
             AUTORISER HXV100;
        FSI
 JSO            SIGNAL POUDRE=1

 * FOPEL       :POUDRE RA101
```

```
* OPELEM      :CHAUFFAGE.RA101
  TYPE        :S
  LIBELLE     CHAUFFAGE PROGRAMME DE RA101

 FAIRE
        SI          N101 MARCHE
            ALORS
             ACTIVE TC1012;
             CALCRAMP RAMPE1;
             CONS TC1012:=VALCOUR RAMPE1;
        FSI
 JSO            FTEMPO T100

 * FOPEL       :CHAUFFAGE RA101
```

```
* OPELEM      :FIN.DISSOLUTION
  TYPE        :S
  LIBELLE     MAINTIEN EN TEMPERATURE POUR DISSOLUTION DANS RA101

 ATT JSO      MESU TI101>DISSOLUTION
 FAIRE
          COMPTER T100;
 JSO          FTEMPO T100

  FOPEL       :FIN DISSOLUTION
```

```
* OPELEM      :REFROID.RA101(VAL:N)
  TYPE        :S
  LIBELLE     REFROIDISSEMENT PROGRAMME DE RA101

  FAIRE
        SI          N101 MARCHE
            ALORS
             ACTIVE TC1013;
             CALCRAMP RAMPE1;
             CONS TC1013:=VALCOUR RAMPE2;
        FSI
  JSO           MESU TI101≤VAL

   FOPEL      :REFROID RA101


* OPELEM      :FIN CRIST
  TYPE        :S
  LIBELLE     MAINTIEN DE L'AGITATION POUR CRISTALLISATION DANS RA101

  FAIRE
          COMPTER T101;
  JSO          FTEMPO T101

   FOPEL       :FIN CRIST
```

Figure 3.25

./..

DESCRIPTION DES OPERATIONS DU PROCEDE

```
* OPE          :RHODIA
  TYPE         :S
  LIBELLE       OPERATION GENERALE DE PREPARATION DE RHODIA

 ATT JSQ      INIT
      PARALL
           *AGIT RA101;
           SEQ
                *CHARGES RA101;
                *DISSOLUTION RA101;
                *CRIST RA101;
           FSEQ
      FPARALL

 FOPE         :RHODIA
```

```
* OPE          :CHARGES,RA101
  TYPE         :S
  LIBELLE      CHARGEMENT DES PRODUITS DANS RA101

     SEQ
          RAZ SIGNAL FOUDRE;
          INIT FQ1014;
          SH LI1010:=SH1 LI1010;
          SB LI2210:=SB1 LI2210;
          *SOLVREC RA101;
          SI            VALCOUR FQ1014<Q1
              ALORS
              *SOLVNEUF RA101;
          FSI
          *FOUDRE RA101;
     FSEQ

 FOPE         :CHARGES RA101
```

```
* OPE          :DISSOLUTION,RA101
  TYPE         :S
  LIBELLE      OPERATION DE DISSOLUTION DANS RA101

      PARALL
          SEQ
                LIM RAMPE1:=LIM1 RAMPE1;
                INC RAMPE1:=INC1 RAMPE1;
                INIT RAMPE1;
               *CHAUFFAGE RA101;
          FSEQ
          SEQ
                RAZ T100;
                PREDET T100:=TEMPS1;
               *FIN DISSOLUTION;
          FSEQ
      FPARALL

 FOPE         :DISSOLUTION RA101
```

Figure 3.26

```
* OPE          :CRIST,RA101
  TYPE         :S
  LIBELLE      OPERATION DE CRISTALLISATION DANS RA101

     SEQ
          LIM RAMPE2:=LIM1 RAMPE2;
          DEC RAMPE2:=DEC1 RAMPE2;
          INIT RAMPE2;
         *REFROID RA101(25);
          RAZ K101;
          PREDET K101:= TEMPS2;
         *FIN CRIST;
     FSEQ

* FOPE         :CRIST RA101
```

Figure 3.27

3.3.6. Représentation graphique de l'application

La figure 3.28 donne une représentation graphique globale de l'application traitée RHODIA (Figures 3.26 - 3.27).

La représentation suit les principes énoncés au paragraphe 3.2. RHODIA contient deux opérations en parallèle : une opération élémentaire AGIT.RA101 et une séquence RHODIA !

Ce dernier nom est généré automatiquement puisque cette séquence n'en possède pas. Il en est de même pour toute structure non nommée. Cette information apparaît sur le graphique (dans ce cas, la deuxième partie de l'élément est vide).

L'opération courante RHODIA ! (encadré spécial) se décompose en trois éléments : une séquence (CHARGES.RA101), un parallélisme (DISSOLUTION.RA101) et une dernière séquence (CRIST.RA101).

A l'écran, tous les éléments petits-fils de l'opération courante ne peuvent être présentés pour la séquence CHARGES.RA101. Trois éléments sont positionnés, la flèche figurant au niveau de l'opération élémentaire SOLVREC.RA101 indique l'existence d'autres éléments.

L'opération DISSOLUTION.RA101 comporte 2 séquences en parallèle (DISSOLUTION.RA10! et DISSOLUTION.RA10").

L'opération CRIST.RA101 possède des éléments en séquence. Cette figure ne permet pas de tous les découvrir.

Le schéma 3.28 reprend l'opération CRIST.RA101 et fait apparaître 7 éléments de la séquence.

On y distingue 2 opérations élémentaires, les autres éléments sont des actions immédiates.

Les pointillés représentent la limite de l'écran.

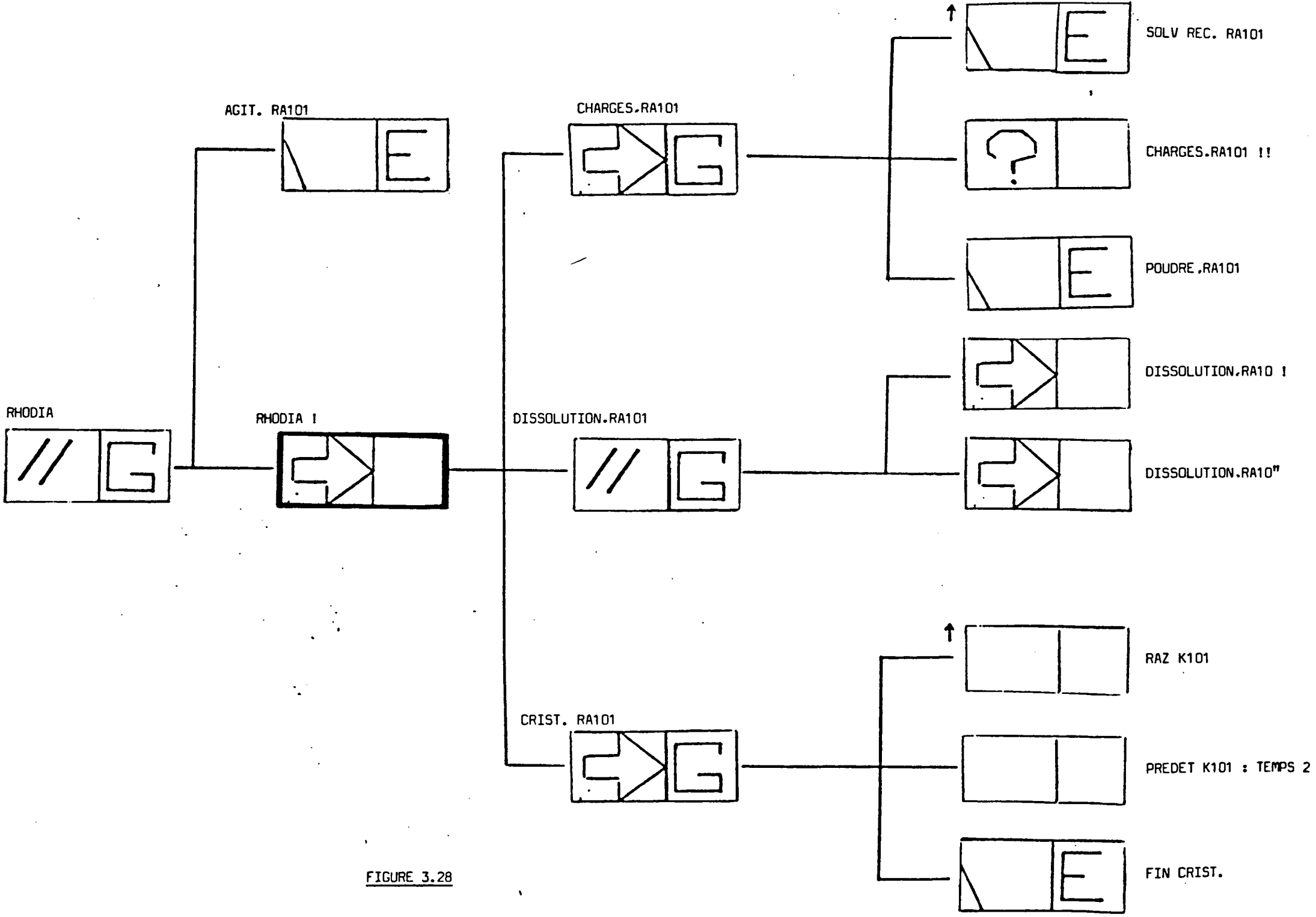

AGIT. RA101
CHARGES.RA101
SOLV REC. RA101
CHARGES.RA101 !!
POUDRE.RA101
RHODIA
RHODIA !
DISSOLUTION.RA101
DISSOLUTION.RA10 !
DISSOLUTION.RA10"
RAZ K101
CRIST. RA101
PREDET K101 : TEMPS 2
FIN CRIST.

FIGURE 3.28

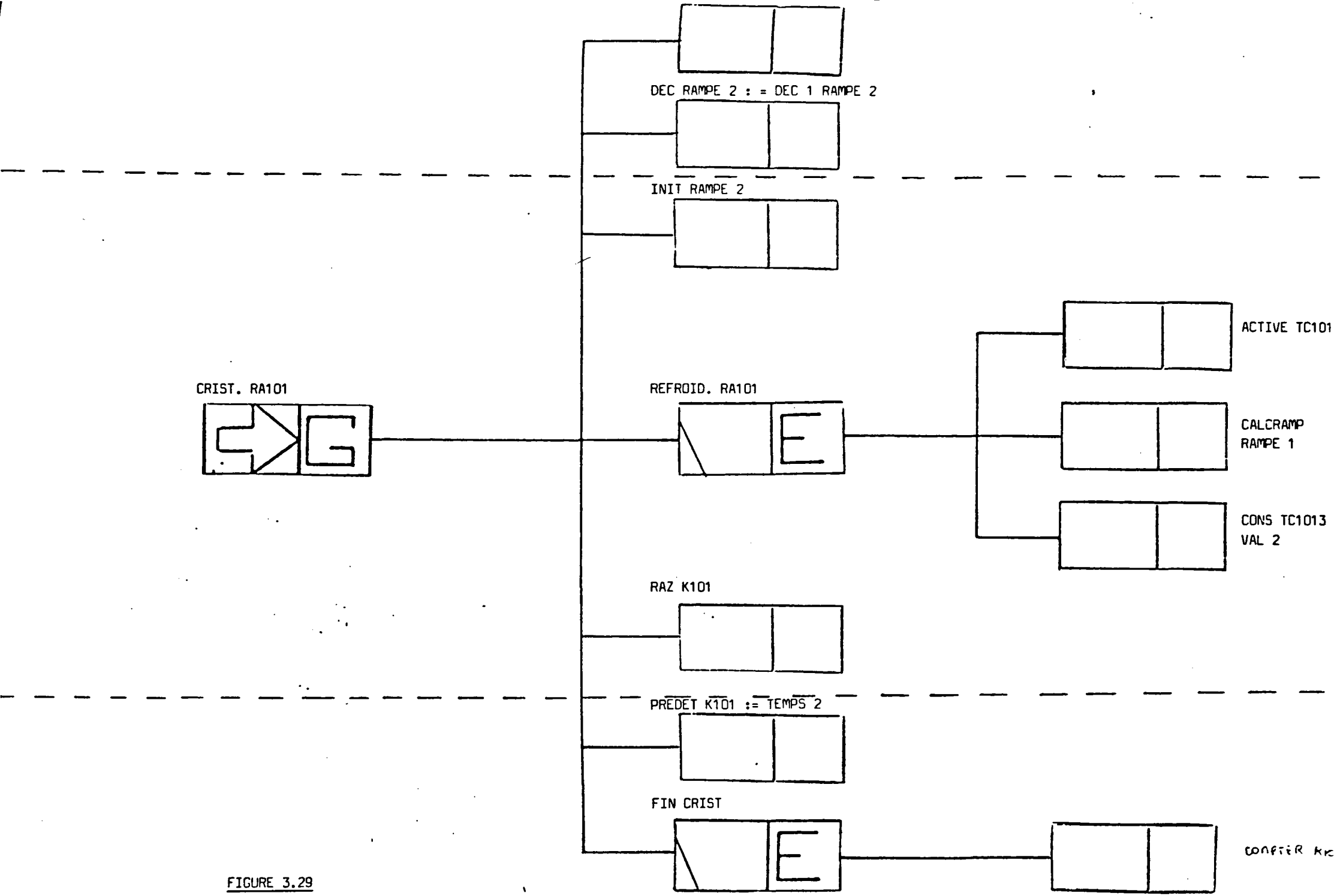

DEC RAMPE 2 : = DEC 1 RAMPE 2
INIT RAMPE 2
ACTIVE TC101
CRIST. RA101
REFROID. RA101
CALCRAMP
RAMPE 1
CONS TC1013
VAL 2
RAZ K101
PREDET K101 := TEMPS 2
FIN CRIST

FIGURE 3.29

3.4. Conclusion

Le concepteur placé devant son poste de travail a la possibilité de spécifier son cahier des charges.

Ce chapitre nous a montré comment il pouvait opérer et également obtenir une représentation graphique de sa description et la documentation associée. Ceci est déjà très intéressant mais il est possible d'aller plus loin, en générant automatiquement les programmes pour la machine d'exécution. Le chapitre suivant aborde les méthodologies permettant cette génération.

CHAPITRE IV

IMPLANTATION DES MODELES RECEPTIFS MULTINIVEAUX

S O M M A I R E

4.1. INTRODUCTION - GENERALITES (TOU)

Les outils formels de spécification sont tous basés sur des hypothèses temporelles très strictes, de façon à donner une interprétation unique indépendante du temps pour divers utilisateurs.

Lorsqu'on cherchera à passer à une matérialisation, il faudra s'arranger, malgré les imperfections temporelles du matériel, pour respecter les conséquences des hypothèses théoriques de l'outil, qui sont :

- . le modèle évolue d'état stable en état stable de façon infiniment rapide, mais discrète,
- . il n'y a pas de variations simultanées d'entrée,
- . il n'y a pas de variations d'entrée pendant un régime transitoire,
- . un passage transitoire n'entraîne pas de discontinuité sur les sorties.

Partant d'une représentation théorique, il convient maintenant de tenir compte de la réalité matérielle, c'est-à-dire du fait que le temps du cycle de traitement n'est plus infiniment court. Il peut donc arriver qu'à l'échelle du temps de cycle deux entrées varient simultanément ou qu'une variation d'entrée se produise durant un régime transitoire.

On est donc amené, pour éliminer les conséquences de ces imperfections du matériel, soit à apporter au modèle de spécification quelques adjonctions ou modifications pour tenir compte des variations simultanées, soit à utiliser une méthodologie d'implantation bien adaptée pour maîtriser les régimes transitoires.

Il est important de savoir que, pour une machine d'implantation donnée et pour un modèle dont on connaît toutes les caractéristiques de synchronisme d'évolution et de régime transitoire, il existe une méthode systématique de passage à la matérialisation.

Pour garantir la disjonction des choix ou de certaines évolutions, il suffit de traiter en séquence, de façon asynchrone, chacune de ces évolutions. La relation d'ordre qui en découle hiérarchise les conflits. On peut aussi adjoindre une disjonction combinatoire.

Il ne faut, en aucun cas, effectuer tous les calculs en figeant les résultats dans des variables internes et en les utilisant par le traitement de l'évolution, car on risque de fabriquer des franchissements simultanés.

Le problème le plus délicat est celui du régime transitoire ; on peut admettre deux philosophies de gestion des traitements de régimes instables.

La première, formelle et universelle, ne regarde les entrées et n'applique les sorties que dans un état stable. Il y a donc (figure 4.1) recherche de stabilité.

Ceci se fait au détriment de la vitesse de traitement car le temps de traitement est dépendant de la longueur du transitoire. Par ailleurs, la programmation de cette procédure est plus lourde.

La seconde (figure 4.2) suppose que le modèle permet, sans inconvénient, un traitement sur plusieurs cycles machine (pas de course critique ou de discontinuité de sortie), ou ne présente pas de transitoire, ou qu'il est possible d'établir une relation d'ordre sur les évolutions de façon à régler le transitoire sur un seul cycle.

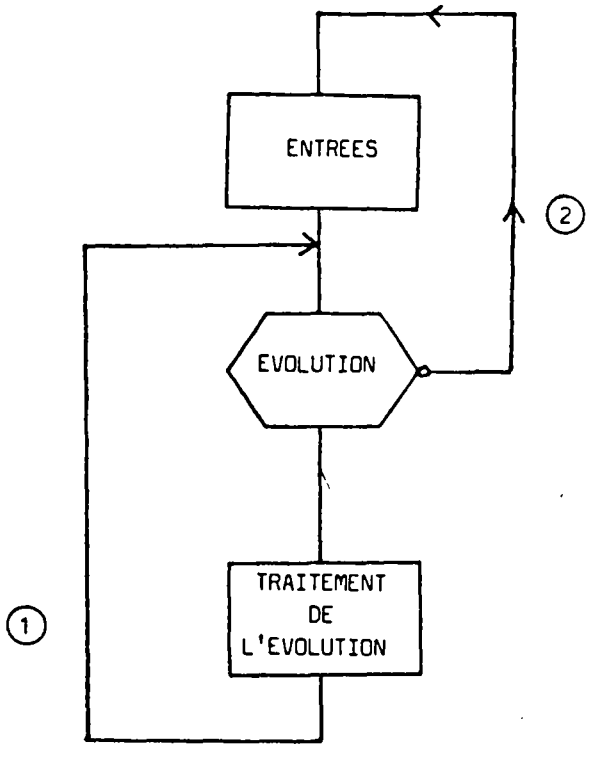


Figure 4.1

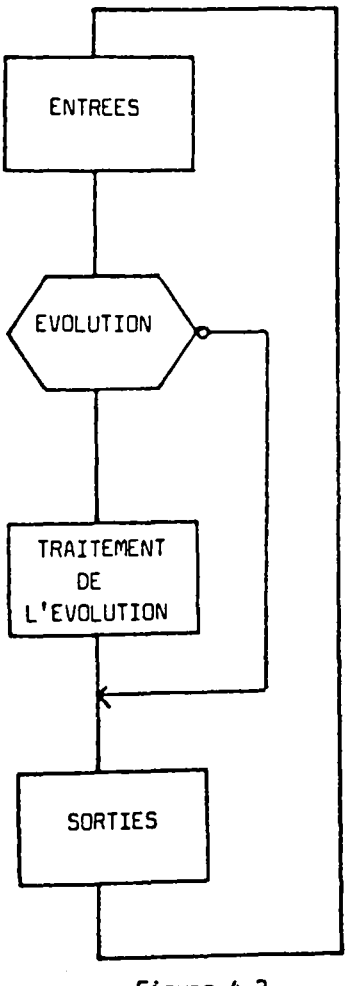


Figure 4.2

Sur la figure 4.1, nous n'avons pas fait figurer les sorties car il convient d'envisager deux cas. On trouve au niveau des opérations élémentaires des actions non répétées lancées transitoirement dont la notification doit avoir lieu en (1), même si l'affectation aux lignes de sortie se fait en (2), comme pour les actions répétées.

L'algorithme de la figure 4.2 ne garantit pas la discontinuité des sorties maintenues sur deux états stables consécutifs. On peut s'en sortir en utilisant des sorties mémorisées mises à (1) avant le passage en action répétée de la première occurence des sorties et mise à zéro lors du franchissement de la réceptivité de sortie de la dernière occurence des sorties.

Pour les implantations orientées "données", l'usage de méthodes systématiques de passage induit l'obtention d'une table représentative du modèle possédant une construction constante.

Dans le cas des implantations orientées "programmes", on trouve, au niveau de la mémoire de programme, une structure constante, sur laquelle on peut retrouver facilement tous les éléments correspondant aux opérations, aux transitions entre opérations, aux réceptivités d'entrée ou de sortie, aux actions. Ce point est particulièrement important lorsqu'on veut apporter des modifications ultérieures ou en cas d'intervention pour anomalie sur un site industriel. Il est en effet possible, moyennant une formation minimum sur ces méthodologies, de donner à un personnel d'intervention le moyen de déchiffrer rapidement des programmes ou des tables de données issues de concepteurs différents.

Le choix du matériel est très important ; car, même s'il est possible de trouver une méthode de passage à la réalisation, quelle que soit la machine programmable, il sera nécessaire de tenir compte de contraintes :

* de réalisation de fonctions opératives ou d'interface (existence ou non de traitements numériques, fonctions de comptage, de gestion de temps...),

* de taille de problème (mémoire de programme ou de données),

* de moyens de dialogue avec l'opérateur,

* de la puissance de certains logiciels de base existants (RTOS, gestion de mémoire de masse, existence de logiciel de communication, éditeurs, contrôleurs de visualisation).

Pour bien montrer les différents types de problèmes relatifs à la puissance de la machine d'implantation, nous allons nous placer aux extrêmes, d'une part sur un automate en version minimale, d'autre part sur un calculateur universel muni d'un compilateur entrelacé FORTH.

4.2. IMPLANTATION SUR AUTOMATE PROGRAMMABLE

L'automate programmable industriel est une machine spécifique des traitements de systèmes à évolution séquentielle.

Toutefois, sous ce vocable, on trouve des matériels très divers en taille et en puissance. Certains n'ont que quelques entrées-sorties et un langage minimum (booléen et temporisation), d'autres sont de véritables ordinateurs de processus avec, par exemple, un compilateur de Grafcet. Mais la majorité des automates programmables procède du déroulement cyclique de la mémoire. Seules quelques machines du haut de gamme sont proches des ordinateurs.

Pour aborder le choix d'une méthodologie, nous considérons un classement des machines vis-à-vis de l'instruction de saut. Tout d'abord, les plus rudimentaires, celles ne disposant pas d'instructions de saut ne peuvent effectuer que des calculs booléens. Il convient donc de ramener le problème séquentiel à un problème combinatoire.

Plusieurs méthodes ont été proposées, mais certaines directes transpositions de la logique câblée sont à éviter. L'utilisateur doit bien comprendre que, dans les circuits câblés, les informations évoluent en parallèle et en asynchonisme.

Dans un programme, les évaluations des diverses variables binaires représentatives des opérations s'effectuent en séquence pouvant amener de nouveaux types d'aléas, si l'on n'y prend pas garde.

Les automates avec saut avant, qui couvrent avec les précédents l'ensemble des machines à déroulement cyclique, autorisent des mises en oeuvre plus proches de la spécification.

L'utilisateur peut alors retrouver dans la mémoire les éléments du modèle.

Toutefois, à ce niveau, plusieurs difficultés surviennent car il s'agit de trouver un compromis entre l'utilisation des variables internes, l'occupation mémoire, et la représentation interne de toutes les opérations. Pour obtenir l'état de toutes les opérations, il faut disposer de très nombreuses variables internes et adjoindre les éléments de programme de gestion de l'activité de ces opérations. De façon à moduler l'implantation, il semble nécessaire de prévoir une procédure de dépliage permettant de supprimer la représentation interne de certaines opérations et de leurs états. Pour faire ceci, l'opérateur pourrait, par exemple, indiquer avant traduction la liste des opérations à conserver.

Une autre difficulté existe également avec l'instruction de saut lorsque celle-ci se fait par modification du compteur ordinal : la durée du cycle de l'automate n'est pas constant. Compte tenu de la pauvreté de ces machines pour la gestion du temps, cette variation de temps de cycle empêche de l'utiliser en base de temps.

4.2.1. Structure générale d'une implantation en déroulement cyclique

Sur une machine à déroulement cyclique, l'ordre logique d'implantation en mémoire de programme pourrait être le suivant :

. Prise en compte des entrées réelles et définitives des macrostructures d'entrée. En effet, on réalise très souvent une grande partie des fonctions opératives ou d'interface sur l'automate programmable. Ceci se traduit, pour chaque fonction, par une ou plusieurs structures logiciables constantes ou paramétrables.

. Définition des éléments de base de temps dans les cas où on souhaite utiliser le temps de cycle ou ses multiples comme périodes d'horloge.

. Structure de contrôle de niveau moniteur.

. Structure de contrôle de la description de base.

. Calcul des sorties fictives puis des sorties réelles, c'est-à-dire tenant compte des conditions de verrouillage.

. Définition des macrostructures de sortie et affectation des sorties, car on implante également des fonctions opératives ou d'interface côté commande.

Le point délicat sur lequel nous focaliserons notre attention est celui de la définition d'une structure de contrôle de la description, c'est-à-dire celui du traitement d'un modèle séquentiel réceptif multiniveaux.

4.2.2. Contraintes d'implantation.

L'automate programmable sur lequel doit s'implanter le modèle possède les particularités techniques suivantes :

. Machine à déroulement cyclique avec prise automatique des entrées en début de cycle et affectation automatique des sorties en fin de cycle.

. Pas d'utilisation de l'instruction de saut de façon à permettre l'usage de la base de temps cycle automate, ou mise en oeuvre de techniques de saut ineffectif.

. Affectation unique des variables au cours d'un cycle.

4.2.3. Transformation d'un modèle hiérarchique en multigraphe d'état.

Les contraintes liées à l'automate sont telles que la seule procédure pour représenter les diverses opérations et leur assemblage est de se ramener à une écriture purement combinatoire. Si, par ailleurs, on souhaite conserver la notion d'opération avec les divers états associés, on doit passer par une décomposition du modèle hiérarchique en un ensemble de graphes d'état, chacun représentant une opération. Le couplage entre ces graphes, c'est-à-dire entre les divers niveaux de la hiérarchie doit se faire de façon complètement synchrone et donc par des techniques de type appel-réponse.

Les transitions liées à l'évolution dans la hiérarchie doivent être traitées suivant l'un des deux algorithmes précédemment indiqués.

Les structures élémentaires qui découlent de la décomposition peuvent alors être codées de façon canonique au combinatoire suivant la minimisation souhaitée du nombre de variables internes utilisées. Remarquons que les structures présentées ne tiennent pas compte d'un éventuel dépliage, envisagé ultérieurement.

La représentation est celle des graphes d'état booléens réceptifs, le formalisme est celui des grafcets ou des réseaux de Pétri ordinaires saufs, l'écriture est celle des grafcets.

4.2.3.1. Opération élémentaire.

L'opération élémentaire peut posséder plusieurs états que l'on retrouve sur la figure 4.3.

L'étape initiale représente l'inactivité de l'opération. On y attend l'ordre de lancement venant d'un niveau supérieur. Ceci permet de passer en attente de la réceptivité d'entrée RE qui amène l'exécution des actions non répétées, puis des actions répétées. Quand la réceptivité de sortie passe à 1, l'opération est finie ; l'étape correspondante sert d'indicateur au niveau supérieur qui répond pour désactiver l'opération.

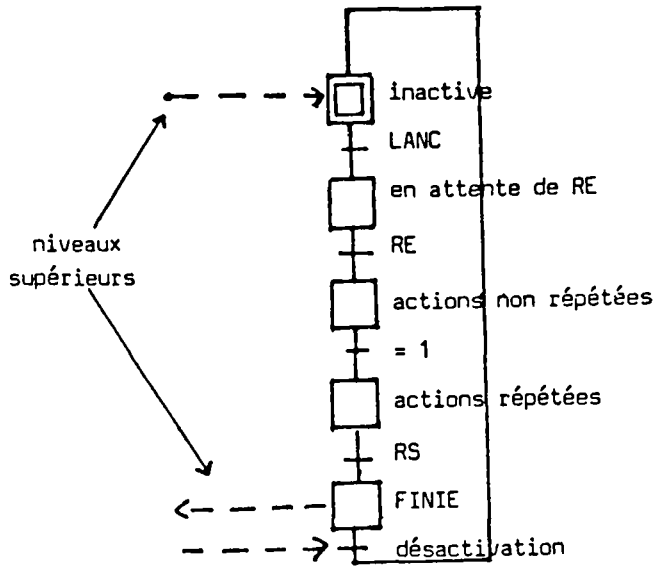
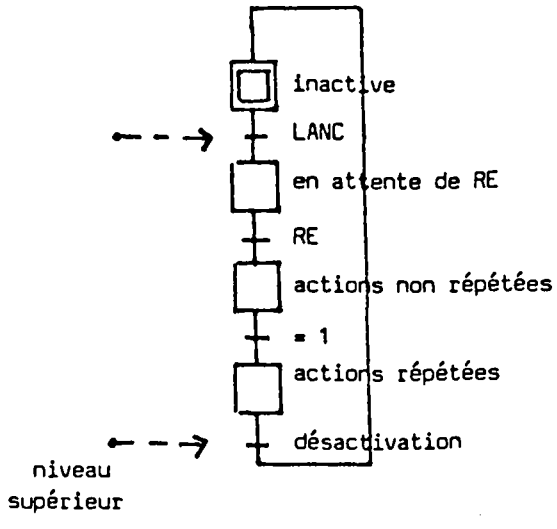


Figure 4.3

4.2.3.2. Opération élémentaire implicite.

Cette opération ne prend son sens que dans le parallélisme. Elle ne se termine pas par elle-même, mais avec la fin du parallélisme.

Un indicateur venant du niveau supérieur désactive donc directement l'opération comme on le voit sur la figure 4.4.



./..

Figure 4.4

4.2.3.3. Structures supérieures.

Après ces deux structures de base intervenant en tant que feuilles de l'arborescence, c'est-à-dire comme éléments exécutables, il convient d'examiner les structures de contrôle des niveaux supérieurs. Le nombre de niveaux pouvant être quelconque, ces structures doivent pouvoir être mises en relations avec les couches au-dessus et en dessous.

Dans tous les cas, on retrouvera les étapes d'inactivité et d'attente de RE. Comme chacune des structures, y compris les opérations élémentaires et élémentaires implicites, peuvent être utilisées plusieurs fois, l'ordre de lancement est la somme logique des divers lancements et l'ordre des désactivations est la somme logique des diverses désactivations.

Dans le cas où une opération est une ressource, il faut lui adjoindre les éléments de contrôle d'accès.

Enfin, les structures du type moniteur ne diffèrent pas essentiellement des structures que nous présentons, elles peuvent utiliser l'état d'activité ou d'inactivité des étapes de chacune des opérations, et par ailleurs forcer les opérations dans un état déterminé.

4.2.3.4. Séquence.

Cette première structure, une fois lancée, attend la réceptivité d'entrée pour lancer la première opération OPE1 qui peut être une opération élémentaire ou non. La fin de cette opération fait passer à OPE2 qui sert aussi de désactivation pour OP1. Et on poursuit ainsi jusqu'à la fin de la dernière opération OPn qui fait passer dans l'état FIN qui désactive OPn et sert d'indicateur pour le niveau supérieur qui renvoie la désactivation de la séquence.

La figure 4.5. donne le graphe d'état de cette structure.

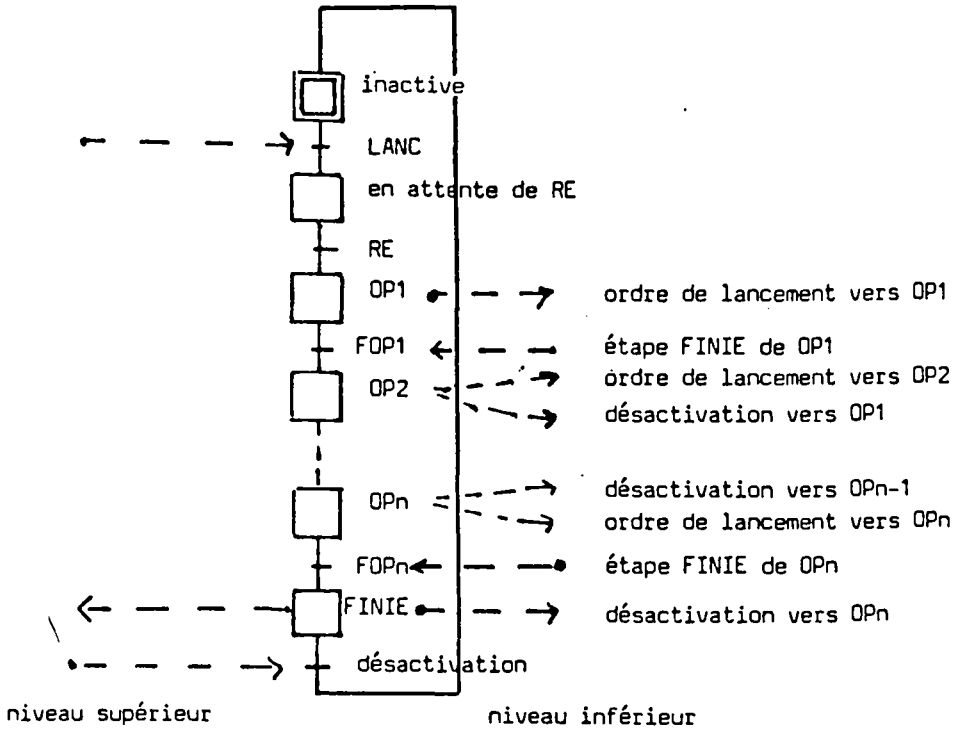


Figure 4.5

Remarques : Une séquence peut se terminer par une opération élémentaire implicite. Elle doit être utilisée dans un parallélisme. on doit alors apporter à la figure 4.5 la modification représentée figure 4.6.

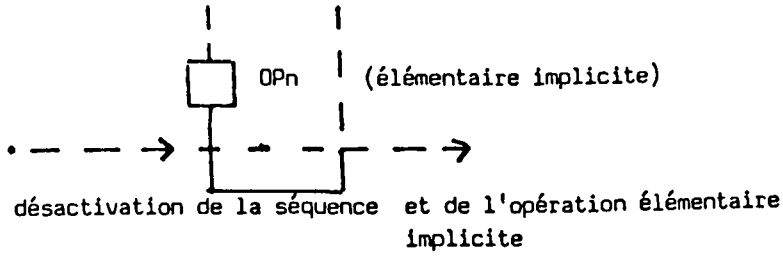


Figure 4.6

La désactivation est issue du parallélisme qui pilote l'implicite.

./..

4.2.3.5. Choix.

La figure 4.7 donne le graphe d'une opération de choix binaire venant du niveau supérieur ; l'ordre de lancement rend l'opération active et en attente de RE. Un aiguillage lié à la condition permet d'orienter vers OP1 ou OP2 qui peuvent être élémentaires ou non. Ceci lance le niveau inférieur qui retourne une indication de fin (étape FINIE correspondante) qui permet la procédure de désactivation comme dans la séquence. Toutefois, l'étape FINIE pour le niveau inférieur sert aussi bien pour la désactivation de OP1 que de OP2.

Dans le cas d'un choix multiple, le principe de fonctionnement est semblable. Le schéma est celui de la figure 4.8.

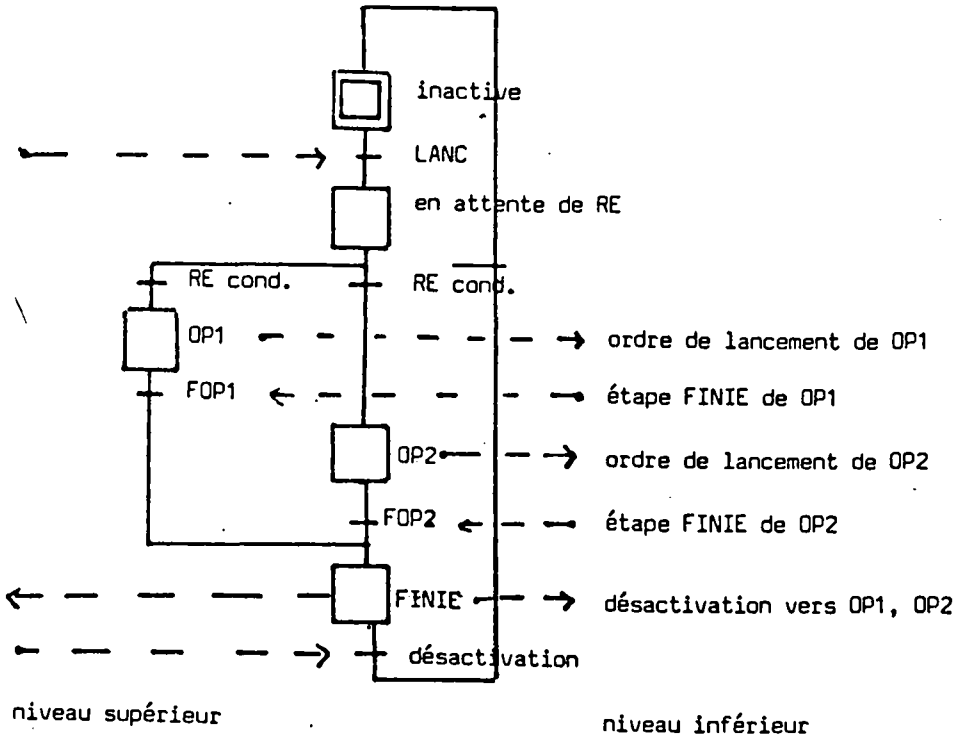


Figure 4.7

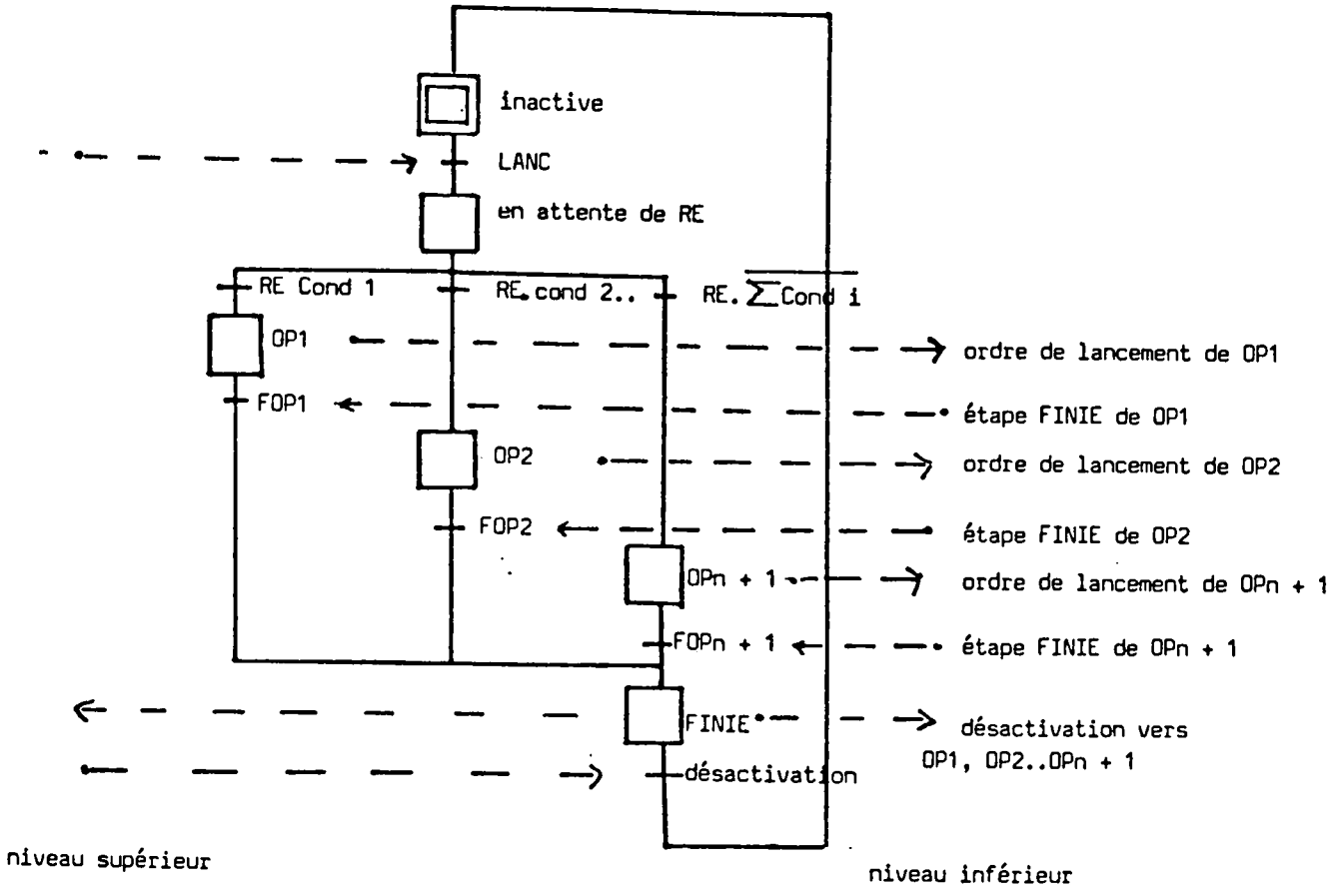


Figure 4.8

L'opération OPn+1 peut être omise (absence de SINON).

Remarques : Dans le cas où OP1 et OP2 sont toutes les deux des opérations élémentaires implicites ou des séquences à fin implicite (seul cas admissible avec des opérations implicites).

La figure 4.7 doit être modifiée comme sur la figure 4.9 ; il en serait de même pour la figure 4.8

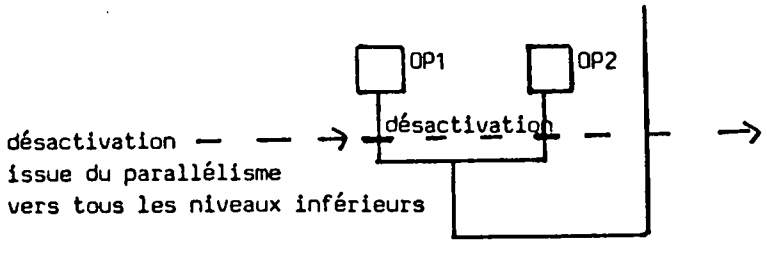


Figure 4.9

4.2.3.6. Parallélisme.

Une fois l'opération lancée et la réceptivité d'entrée vraie, on lance, au niveau inférieur, toutes les opérations intervenant dans le parallélisme. Celles-ci peuvent être élémentaires, élémentaires implicites ou autres, mais au moins une n'est pas élémentaire implicite.

Cette opération, décrite figure 4.10, n'est finie que si toutes les opérations inférieures non implicites sont finies, ce qui permet le dialogue avec le niveau supérieur et la désactivation du niveau inférieur pour toutes les opérations.

On a vu précédemment que cette désactivation issue du parallélisme est transmise à toutes les couches intégrant l'opération implicite et à cette opération elle-même.

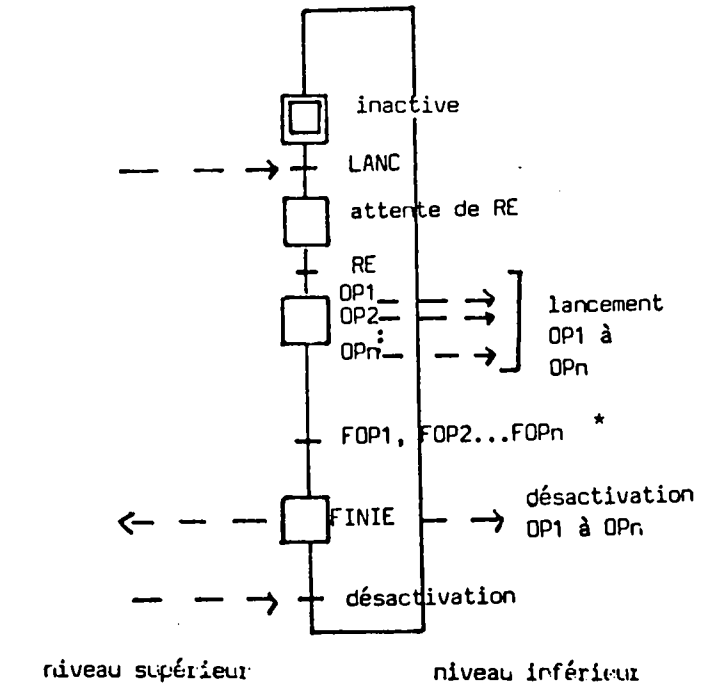


Figure 4.10

* Pour les opérations implicites, on remplace FOPi par l'activité de l'étape action répétée de l'opération implicite i.

./..

4.2.3.7. Itération.

Cette structure est un peu plus difficile à mettre en oeuvre car il convient de prendre quelques précautions relativement à la réactivation de l'opération. Pour un des cas d'itération, on obtient la figure 4.11.

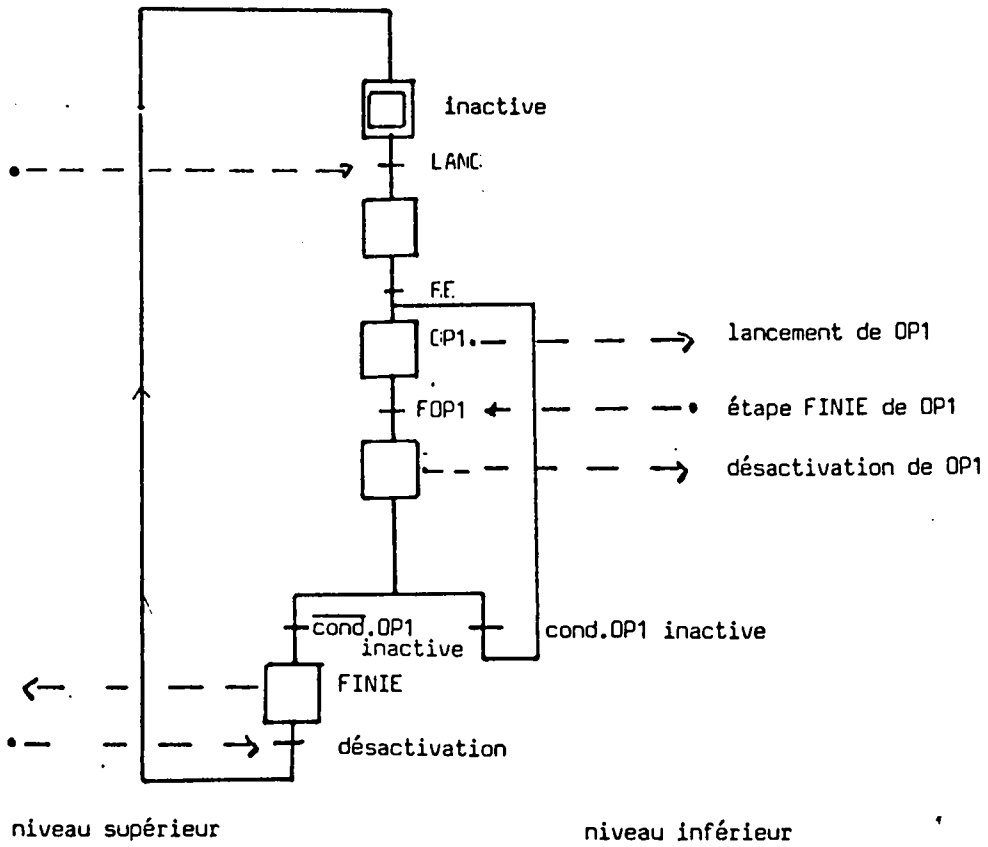


Figure 4.11

De façon à pouvoir réactiver l'opération OP1, il faut vérifier que OP1 est réellement inactive. Cette situation se retrouve également dans le cas d'une séquence dans laquelle deux opération identiques se suivent. On aurait alors la configuration de la figure 4.12.

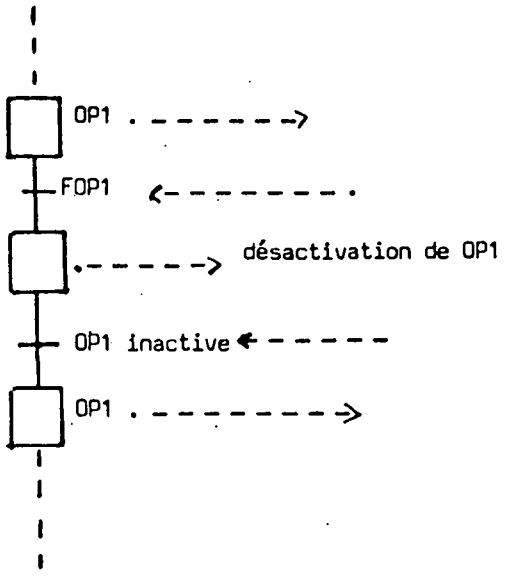


Figure 4.12

4.2.3.8. Racine de l'arborescence.

L'absence de niveau supérieur fait qu'à la racine de l'arborescence on peut faire disparaître ce qui concerne le lancement et la désactivation finale, soit par mise à une des réceptivités correspondantes, soit par suppression des étapes et transitions inutiles.

4.2.3.9. Remarques.

a. Les graphes que nous avons présentés dans les paragraphes précédents explicitent par des étapes tous les états possibles d'une opération. Il serait possible d'effectuer des condensations, d'une part en reportant les actions non répétées au niveau du franchissement de la transition de réceptivité RE des opérations élémentaires ou élémentaires implicites, d'autre part en effectuant le regroupement des deux étapes : inactive et en attente de RE et en remplaçant RE par RE.LANC.

Pour obtenir l'état de l'opération, il suffit alors de faire un traitement combinatoire.

b. Dans tous les schémas précédents, lors de la désactivation d'une opération, on lance l'ordre de désactivation vers les niveaux inférieurs sans vérifier que celle-ci est réellement effectuée.

Ceci, dans la majorité des cas, ne pose aucun problème ; on a pourtant vu que des précautions doivent être prises dans le cas de l'itération. En toute rigueur, on doit vérifier par une transition que les étapes à désactiver sont passées dans l'état inactif. Ceci entraîne, pour la séquence de la figure 4.5, la représentation de la figure 4.12 bis.

Cette vérification alourdit considérablement la programmation et peut être évitée en général. Son usage dans le cas d'opérations implicites est souvent nécessaire comme l'illustre le petit exemple suivant où OPEL4 est une opération implicite, OPEL1, OPEL2 et OPEL3 des opérations élémentaires.

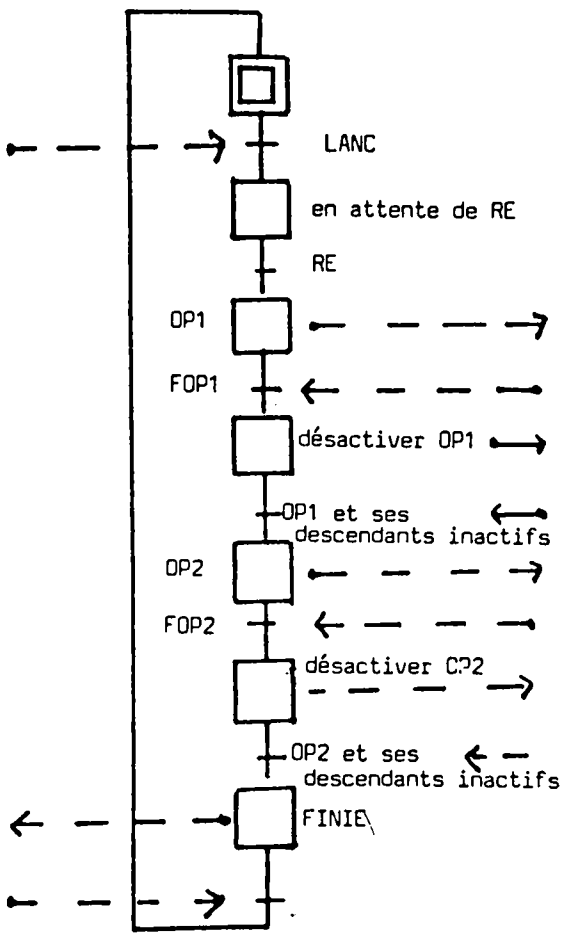


Figure 4.12 bis

```
* OPE            :OP0
  TYPE           :E
  LIBELLE

     SEQ
          *OP2;
          *OP3;
     FSEQ

) FOPE           :OP0
```

```
* OPE            :OP1
  TYPE           :E
  LIBELLE

     SEQ
          *OPEL3;
          *OPEL4;
     FSEQ

) FOPE           :OP1
```

```
* OPE            :OP2
  TYPE           :E
  LIBELLE

     PARALL
          *OP1;
          *OPEL1;
     FPARALL

) FOPE           :OP2
```

```
* OPE            :OP3
  TYPE           :E
  LIBELLE

     PARALL
          *OPEL2;
          *OPEL4;
     FPARALL

) FOPE           :OP3
```

(Il y correspond les graphes de la figure 4.12 ter).

./..

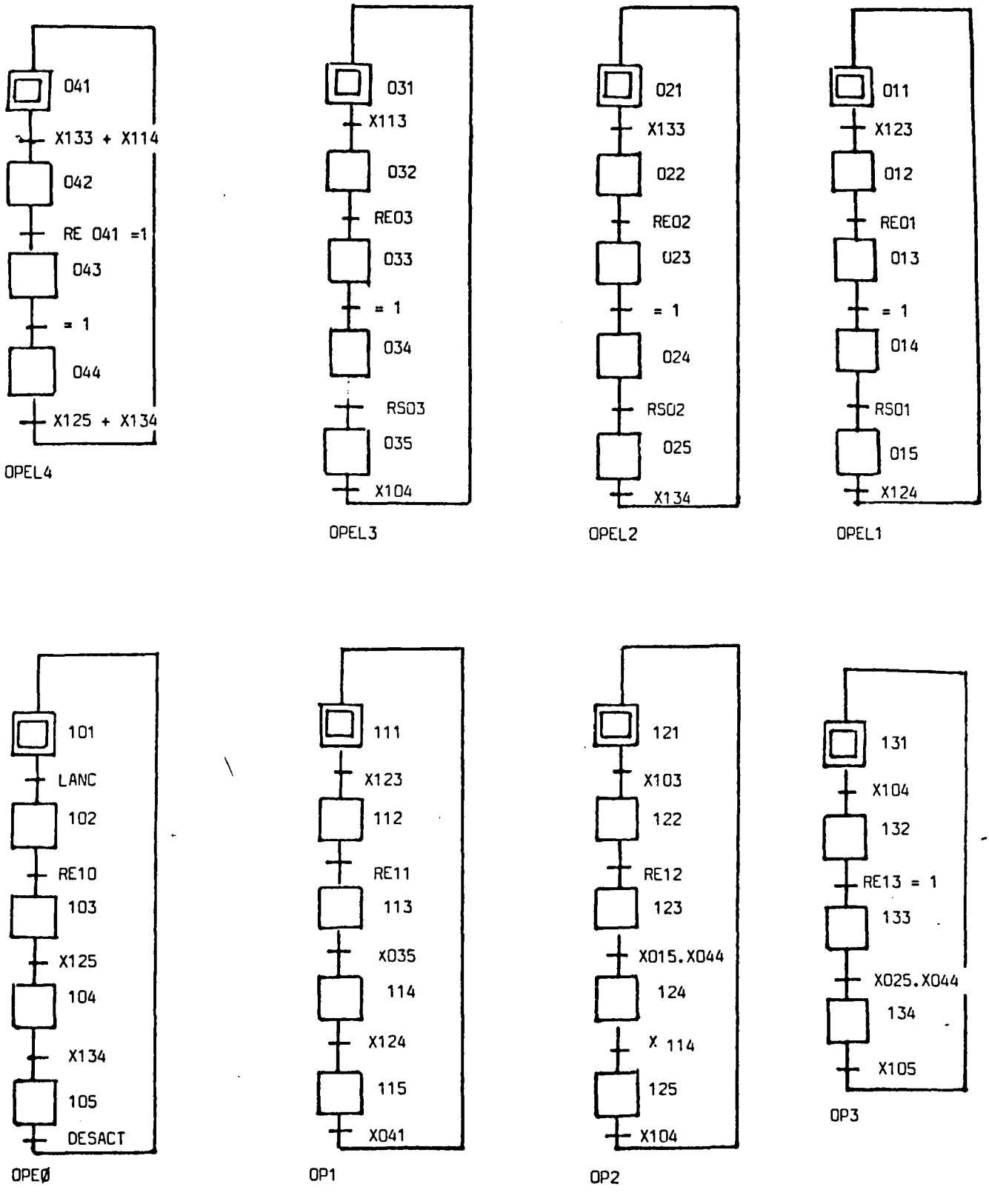


Figure 4.12 ter

./..

On suppose que les réceptivités d'entrée RE04 et RE13 sont toujours à 1. Pour éviter les aléas de fonctionnement, on doit vérifier en cascade les désactivations. OP2 désactive OP1 qui désactive OPEL4. OP1 reste dans l'étape 115 tant que OPEL4 n'est pas inactive, OP2 reste dans l'étape 124 tant que OP1 n'est pas inactive. Lorsque OP1 devient inactive, OP2 peut signaler au niveau supérieur sa fin d'activité, ce qui crée une évolution de l'étape 103 à l'étape 104 et, par retour, la désactivation d'OP2.

4.2.4. Exemple.

Pour illustrer l'usage des structures que nous venons de présenter, nous proposons l'exemple suivant (figure 4.13) :

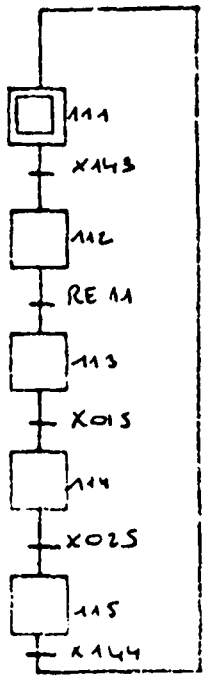
- 10PEL, 20PEL, 30PEL, 40PEL, 50PEL, 70PEL, 80PEL, 90PEL, 100PEL, et 120PEL sont des opérations élémentaires.
- 60PEL ET 110PEL sont des opérations élémentaires implicites.

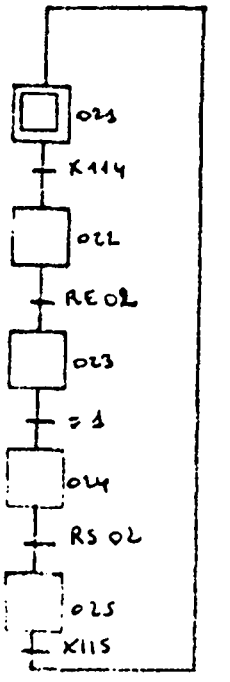
De façon à effectuer le couplage entre les divers graphes d'état, on utilise les variables représentatives de l'activité des étapes :

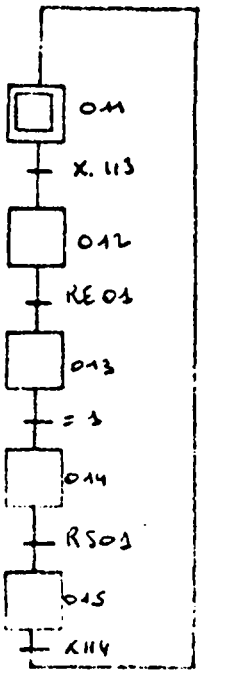
- soit Xi l'activité de l'étape i,
- et i et $\overline{Xi}$ son inactivité.

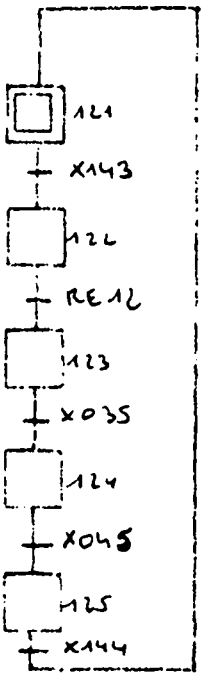
Il est à noter qu'avec le principe d'appel-réponse dont nous faisons usage, il n'y a aucun problème de synchronisme de franchissement.

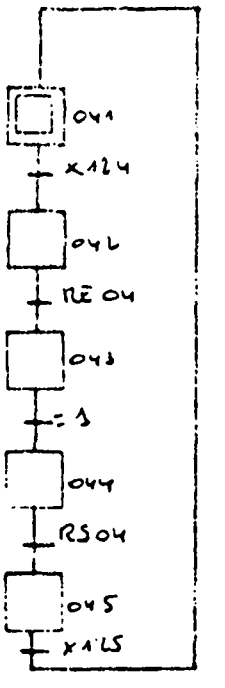
Enfin, l'opération 110PE est une itération qui se trouve être racine de l'arborescence, c'est pourquoi son graphe a pu être réduit par rapport à celui de la figure 4.11.

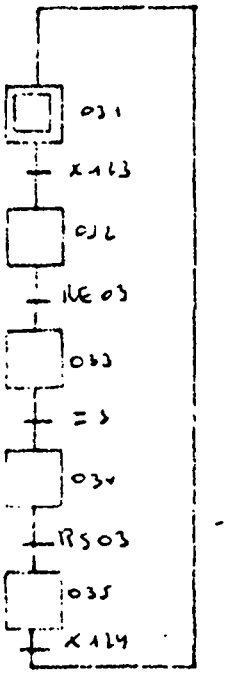

111
X143
112
RE 11
113
X015
114
X025
115
X144

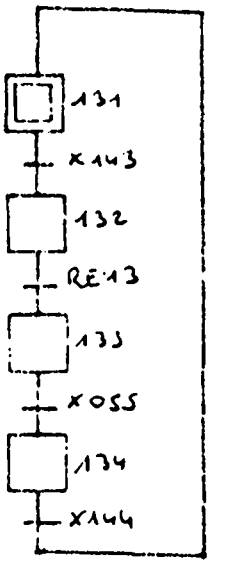

021
X114
022
RE02
023
=1
024
RS02
025
X115

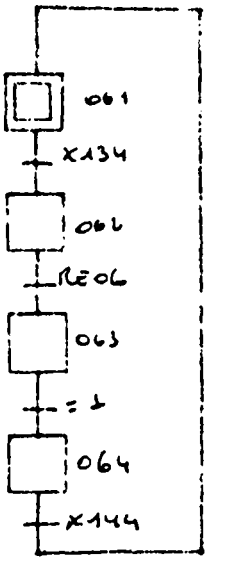

011
X.113
012
RE01
013
=1
014
RS01
015
X114


121
X143
122
RE12
123
X035
124
X045
125
X144


041
X124
042
RE04
043
=1
044
RS04
045
X125


031
X143
032
RE03
033
=1
034
RS03
035
X124


131
X143
132
RE13
133
X055
134
X144


061
X134
062
RE06
063
=1
064
X144

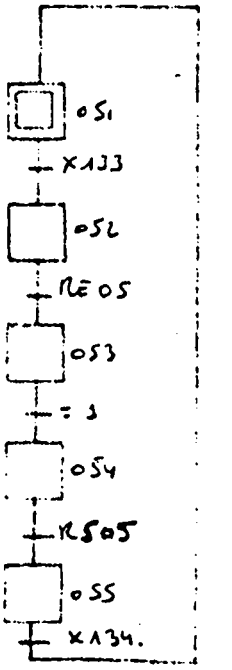
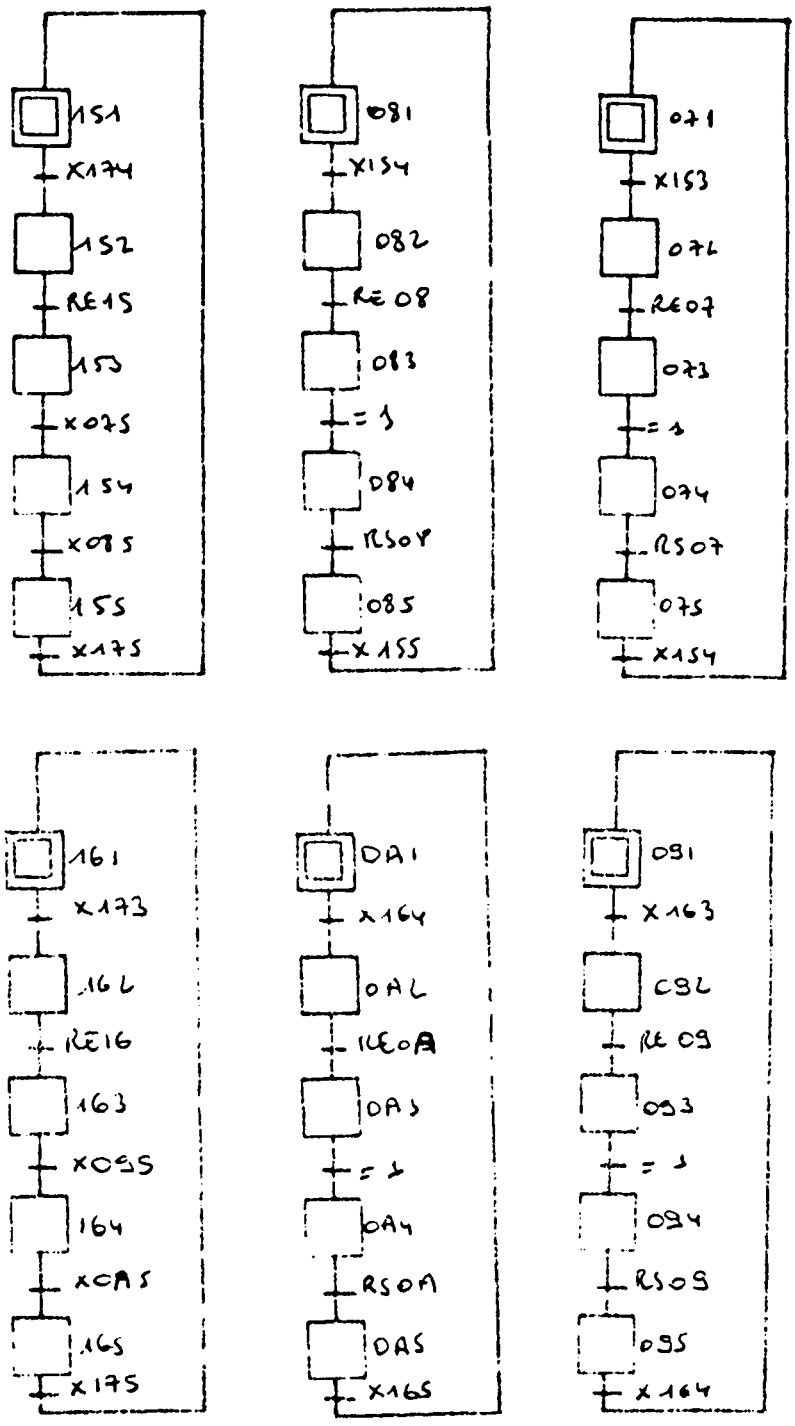

051
X133
052
RE05
053
=1
054
RS05
055
X134.

Figure 4.13



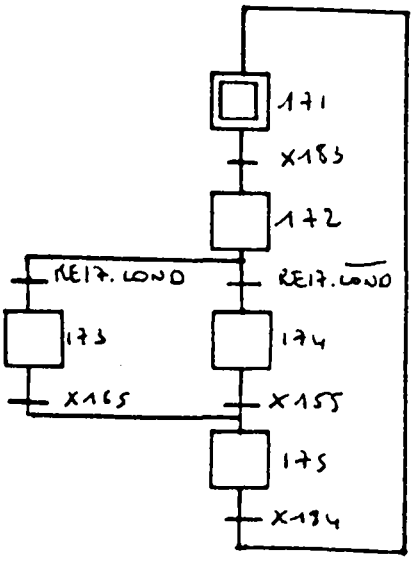
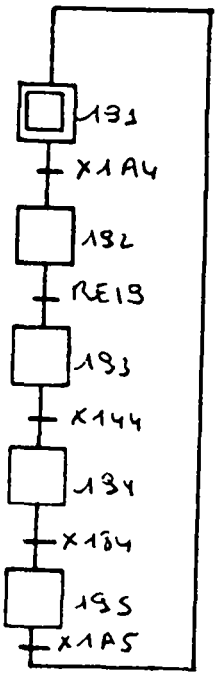
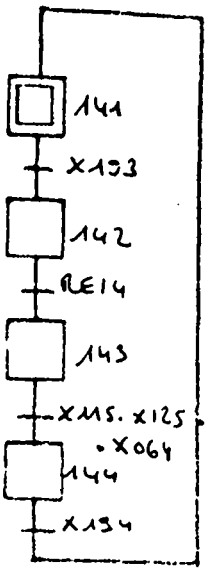
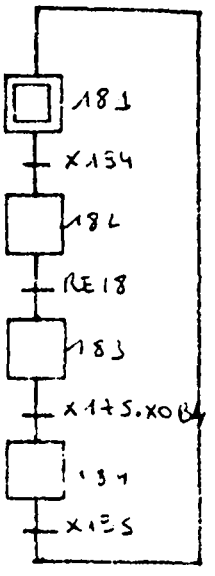
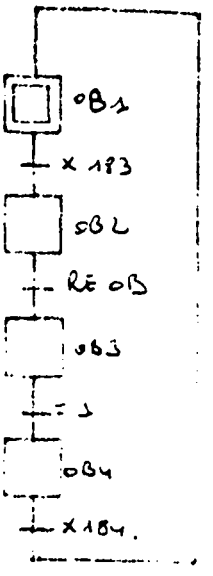


Figure 4.13 (suite)









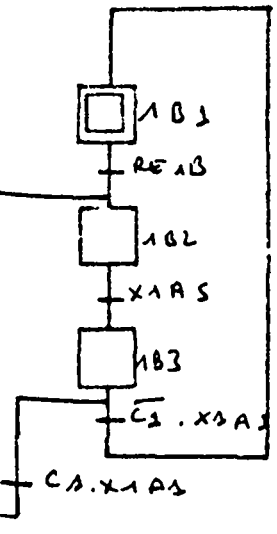
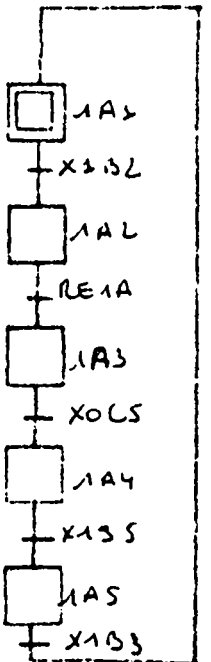
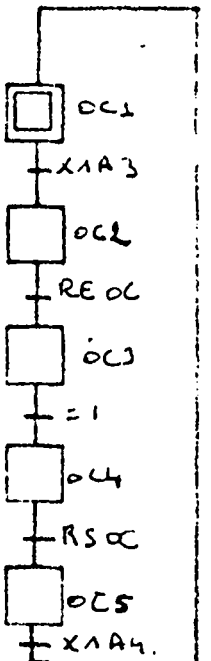


Figure 4.13 (fin)





4.2.5. Codage des graphes et passages aux équations combinatoires

Pour coder chacun des graphes d'état, on peut, soit utiliser une variable interne par étape (on a alors un codage canonique), soit prendre, pour l'ensemble du graphe, n variables, de façon à avoir $2^n \geq m$, où m est le nombre d'étapes. Dans le premier cas, la machine d'implantation doit posséder un nombre important de variables internes, mais fournit, très facilement, l'état des opérations. Cet état doit être reconstitué dans le second cas.

4.2.5.1. Codage canonique.

Soit par exemple le graphe d'une opération de choix (figure 4.14) où une variable Xi est associée à chacune des étapes. plusieurs procédures de traitement sont envisageables.

(α) Utilisation de la définition d'un graphe booléen réceptif.

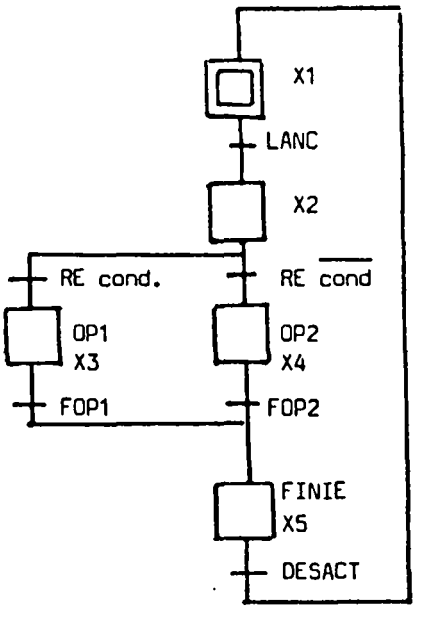


Figure 4.14

Un graphe booléen réceptif est défini par la relation matricielle booléenne :

$$X(h + 1) = (Rij).\ X(h)$$

où (Rij) est la matrice des réceptivités telles que

$Rii + \sum \overline{Rij} = Rii$ ou souvent plus simplement

$Rii = \sum \overline{Rij}$ et où X(h) représente le vecteur d'état à l'instant h.

En appliquant ces définitions aux graphes de la figure 4.14, on obtient :

$$X1 = \overline{LANC}.\ X1 + X5.\ DESACT$$

$$X2 = LANC.\ X1 + \overline{RE}.\ X2$$

$$X3 = X2.\ RE.\ COND + X3.\ \overline{FOP1}$$

$$X4 = X2.\ RE.\ COND + X4.\ \overline{FOP2}$$

$$X5 = X3.\ FOP1 + X4.\ FOP2 + X5.\ \overline{DESACT}$$

Le traitement des Xi doit se faire de façon synchrone, c'est-à-dire qu'il convient de figer les conditions d'activation et de désactivation de chacune des étapes. On retrouve là une méthode connues sous le nom "Activation-Désactivation", dans laquelle, pour chacune des étapes de chaque graphe d'état, on détermine la condition d'activation Ai qui est la somme logique des conditions de franchissement de chacune des transitions de sortie de l'étape i et la condition de désactivation Di qui est la somme logique des conditions de franchissement de chacune des transitions de sortie de l'étape i.

On représente alors l'étape i par une variable interne Xi régie par l'équation d'une mémoire priorité à l'activation :

$$Xi = Ai + Xi \cdot \overline{Di}$$

Pour le graphe de la figure 4.14, on obtient ainsi :

$A1 = X5 \cdot DESACT$ $\qquad$ $D1 = A2$

$A2 = X1 \cdot LANC$ $\qquad$ $D2 = A3 + A4$

$A3 = RE \cdot Cond.X2$ $\qquad$ $D3 = X3 \cdot FOP1$

$A4 = RE \cdot \overline{Cond}.X2$ $\qquad$ $D4 = X4 \cdot FOP2$

$A5 = D3 + D4$ $\qquad$ $D5 = A1$

et les Xi correspondants

(i = 1 à 5)

(β) Traitement global du graphe d'état.

Comme un graphe d'état n'a au plus qu'une étape active, lors d'une évolution seule une transition est franchissable, celle qui active l'étape pour qui elle est transition d'entrée.

On en déduit la méthode suivante représentée figure 4.15 pour l'exemple de la figure 4.14.

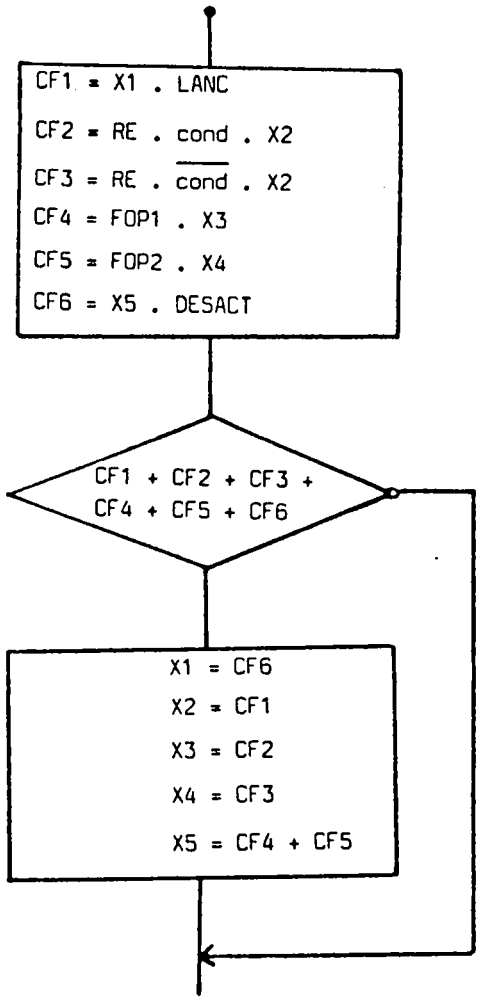

CF1 = X1 . LANC
CF2 = RE . cond . X2
CF3 = RE . $\overline{cond}$. X2
CF4 = FOP1 . X3
CF5 = FOP2 . X4
CF6 = X5 . DESACT
CF1 + CF2 + CF3 +
CF4 + CF5 + CF6
X1 = CF6
X2 = CF1
X3 = CF2
X4 = CF3
X5 = CF4 + CF5

Figure 4.15

L'inconvénient de cette écriture est d'utiliser un saut effectif, ce qui entraîne une modification du temps de cycle machine.

Ceci ne respecte donc pas une des contraintes précédemment définies. En modifiant la présentation (figure 4.16), on peut rendre tous les traitements combinatoires et obtenir un cycle constant avec des sauts ineffectifs.

CF1 = X1 . LANC CF2 = RE . Cond . X2 CF3 = RE . Cond . X2 CF4 = FOP1 . X3 CF5 = FOP2 . X4 CF6 = X5 . DESACT
CS = CF1 + CF2 + CF3 + CF4 + CF5 + CF6
X1 = CF6 . C5 + X1 . C5 X2 = CF1 . C5 + X2 . C5 X3 = CF2 . C5 + X3 . C5 X4 = CF3 . C5 + X4 . C5 X5 = (CF4 + CF5) . C5 + X5 . $\overline{C5}$

Figure 4.16

4.2.5.2. Codage combinatoire

A cause du déroulement cyclique et de la structure de la programmation envisagée, il n'y a aucun risque d'aléas de fonctionnement de type course critique et il est donc possible d'adopter un codage combinatoire quelconque.

Ainsi, dans l'exemple de la figure 4.14, pour coder les cinq étapes, il est nécessaire d'utiliser trois variables Z1, Z2, Z3, et on peut prendre par exemple le codage du numéro de l'étape en binaire naturel avec Z1 en poids forts, d'où :

$$X1 = \overline{Z1}\ \overline{Z2}\ Z3$$

$$X2 = \overline{Z1}\ Z2\ \overline{Z3}$$

$$X3 = \overline{Z1}\ Z2\ Z3$$

$$X4 = Z1\ \overline{Z2}\ \overline{Z3}$$

$$X5 = Z1\ \overline{Z2}\ Z3$$

Ce qui donne :

$$Z1 = X4 + X5$$

$$Z2 = X2 + X3$$

$$Z3 = X1 + X3 + X5$$

4.2.6. Dépliage de l'arborescence.

De façon à diminuer la quantité de variables internes nécessaires à la représentation en machine, on peut encore procéder au dépliage de l'arborescence au niveau de certaines opérations dont on accepte de perdre l'état détaillé immédiat ou la représentation directe en machine. Considérons par exemple une partie de la figure 4.13 portant sur l'opération 70PE dans laquelle on accepte de ne pas faire figurer directement 60PE. Cela revient donc à écrire :

```
* OPE          :70PE
  TYPE         :E
  LIBELLE

     SI              COND
          ALORS
          SEQ
               *90PEL;
               *100PEL;
          FSEQ
          SINON
          *50PE;
     FSI

) FOPE         :70PE
```

./..

Ce qui correspond à la figure 4.17.

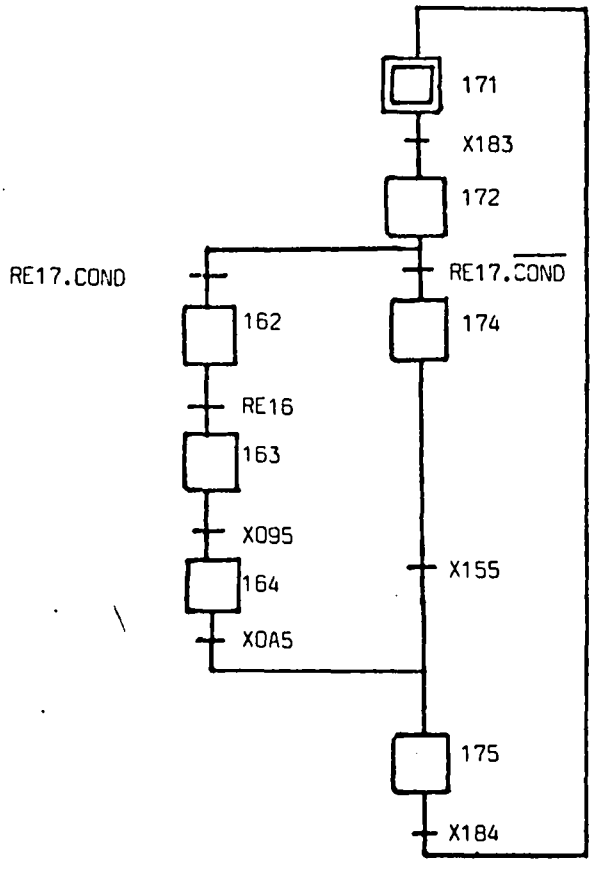


Figure 4.17

En codage combinatoire, ceci permet de gagner le codage de 6OPE.

En codage canonique, on gagne trois variables internes.

Il est possible de retrouver l'état de 6OPE en utilisant les combinaisons de X162, X163, X164.

4.3. IMPLANTATION EN CODE INDIRECT ENTRELACE.

L'objectif des techniques entrelacées est de conserver en mémoire l'implantation de la structure telle qu'elle apparaît dans la description. Classiquement, ces techniques sont basées sur des modèles structurés, arborescents, et n'intégrant pas la notion de parallélisme. Par ailleurs, les feuilles de l'arborescence sont du code machine exécutable.

Nous allons donc reprendre ces notions et y apporter les modifications nécessaires pour tenir compte d'une part d'un traitement temps réel avec parallélisme et d'autre part de la notion d'opération élémentaire, feuille de notre description.

Le langage correspondant à un code indirect entrelacé, et que nous utiliserons comme point de départ de notre implantation, est le FORTH. Nous présentons, en annexe 1, un résumé des éléments de ce langage nécessaires à la compréhension de l'exposé.

4.3.1. Notion d'interpréteur interne.

Supposons un programme écrit suivant une hiérarchie structurée. Les feuilles de l'arborescence sont des programmes en langage machine. Envisageons, pour simplifier l'exposé, uniquement des séquences de procédure au niveau supérieur.

L'arbre de la figure 4.18 se traduit en un programme correspondant à une suite d'appels de sous-programmes.

```
PRG0    : CALL PRG1
          CALL PRG2

PRG1    : CALL PRGM1
          CALL PRGM2

PRG2    : CALL PRGM3
          CALL PRGM4

PRGM1   : .
          .
          RET

PRGM2   : .
          .
          RET

PRGM3   : .
          .
          RET

PRGM4   : .
          .
          RET
```

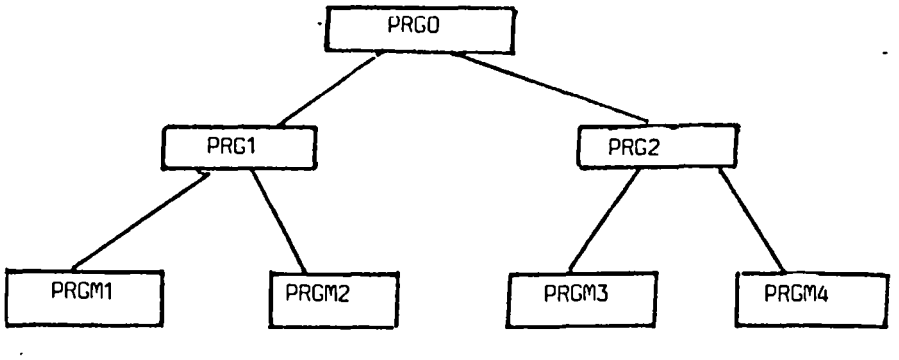


Figure 4.18

L'idée de base du codage indirect entrelacé est de supprimer les CALL, de faire la distinction entre les niveaux supérieurs et les feuilles en utilisant le principe suivant :

a) Définition du niveau supérieur.

PRG : En tête (adresse d'un programme constant spécifique de l'en-tête)

.

.

. suite d'adresses des PRGi ou de PRGMi

.

Fin (adresse d'un programme constant spécifique de FIN).

b) Définition de niveau inférieur.

PRGM : Adresse du programme en langage machine.

Ce programme doit se terminer par un saut à un programme constant qui permet de passer à la suite.

En notant, comme en FORTH, DOCOL l'adresse du programme en-tête, SEMIS l'adresse du programme de FIN, et NEXT la routine de fin de programme machine, le programme en entrelacé pour la figure 4.18 devient :

```
PRG0  :   DOCOL
          PRG1
          PRG2
          SEMIS

PRG1  :   DOCOL
          PRGM1
          PRGM2
          SEMIS

PRG2  :   DOCOL
          PRGM3
          PRGM4
          SEMIS

PRGM1 :   PRGM1 + 1
            .
            .
          JMP NEXT

PRGM2 :   PRGM2 + 1
            .
            .
          JMP NEXT

PRGM3 :   PRGM3 + 1
            .
            .
          JMP NEXT

PRGM4 :   PRGM4 + 1
            .
            .
          JMP NEXT
```

./..

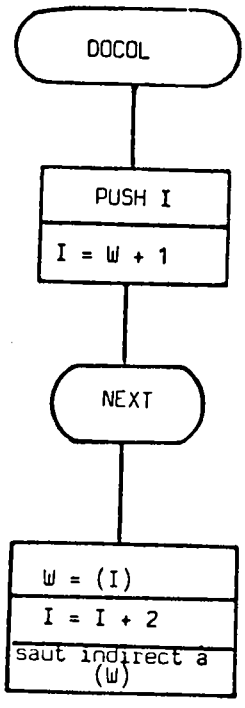
L'ensemble des programmes DOCOL, SEMIS et NEXT constitue l'interpréteur interne qui a pour rôle de suivre l'évolution dans l'arborescence (figure 4.19). Pour cela, il utilise une pile dite de retour, un pointeur d'interprétation I et un pointeur tampon W. Sur la pile de retour, on utilise deux opérations PUSH et POP qui empile et dépile.

Pour faire le passage des valeurs entre les divers morceaux de programme exécutables, on peut passer par des variables ou par une pile dite pile de paramètres.

Les structures de contrôle classiques, itération et alternative se réalisent à partir de deux routines de branchement relativement au pointeur d'interprétation.

BRANCH offset réalise un saut inconditionnel avec la valeur de l'offset et ØBRANCH un saut conditionnel à la valeur nulle du sommet de pile paramètre.

L'organisation en mémoire de ces structures est donnée figures 4.20 et 4.21.


DOCOL
PUSH I
I = W + 1
NEXT
W = (I)
I = I + 2
saut indirect a (W)

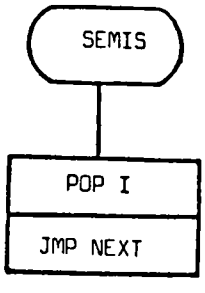
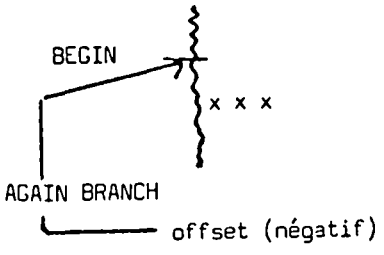

SEMIS
POP I
JMP NEXT

Figure 4.19

BOUCLE INFINIE

BEGIN x x x AGAIN

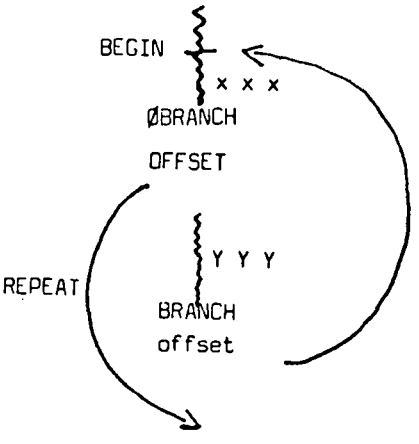


ITERATION CONTROLEE

BEGIN x x x UNTIL

identique à BEGIN AGAIN
mais avec ØBRANCH au
lieu de BRANCH

BEGIN x x x WHILE y y y REPEAT



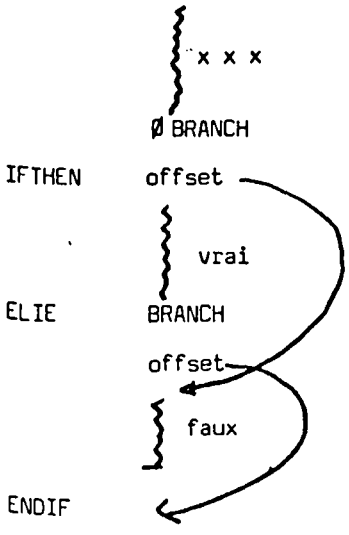
x x x place dans la
pile une valeur à
tester par ØBRANCH

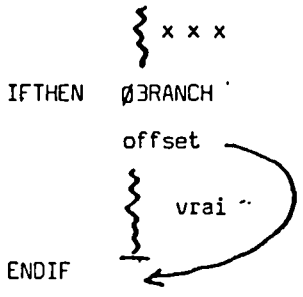
FIGURE 4.20

ALTERNATIVE

x x x IFTHEN ~~ vrai ~~ ELSE ~~ faux ~~ ENDIF

x x x place dans la pile paramètre une valeur à tester.
La partie ELSE peut ne pas exister.





On peut imbriquer les structures

x x x IFTHEN vrai 1 ELSE y y y IFTHEN vrai 2 ELSE faux 2 ENDIF ENDIF

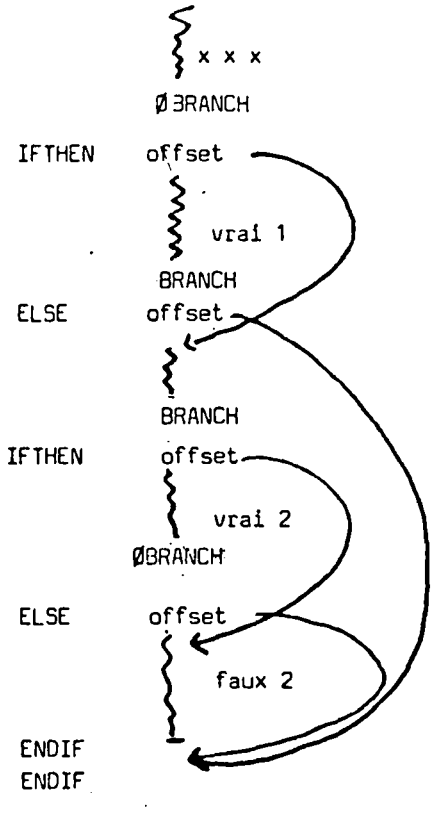


FIGURE 4.21

./..

4.3.2. Interpréteur avec parallélisme.

- Le traitement avec parallélisme que nous souhaitons introduire doit se faire en temps réel, c'est-à-dire qu'à chaque cycle de traitement nous avons à considérer l'évolution d'un certain nombre d'opérations élémentaires. Le cycle complet doit suivre un des schémas proposés en figure 4.1. ou 4.2.

  Le parallélisme du traitement des opérations élémentaires est assuré par une opération PARALL qui peut induire la gestion d'une arborescence. Il est donc nécessaire, pour chaque branche entrant dans le parallélisme, de disposer d'une pile de retour avec ses variables de gestion, d'un pointeur d'interprétation et d'un registre de travail. On y adjoindra un registre indicateur IND, donnant l'état de la branche.

  Pour mesurer la profondeur du parallélisme, on utilise une variable PAR.

  Une des grandes particularités des traitements effectués dans chaque opération élémentaire est de laisser la pile "paramètre" dans le même état après traitement qu'avant traitement. Il n'est donc pas nécessaire de créer plusieurs piles "paramètres".

  L'évolution à l'intérieur d'une opération élémentaire se fait en quatre temps, correspondant à des sous-opérations :

  TESTRE: a pour rôle de tester, au cours d'un cycle de traitement, la réceptivité entrée. Si elle n'est pas vraie, on reste sur TESTRE pour le cycle suivant, sinon on prépare le passage à ANR pour le cycle suivant.

  ANR : au cours d'un cycle, envoie les commandes non répétées et, au cours du même cycle, fait passer à AR.

  AR : à chaque cycle, envoie les commandes dites répétées, teste la réceptivité de sortie, prépare pour le cycle suivant. Si la réceptivité de sortie n'est pas vraie, on reste sur AR. Sinon, on termine l'opération élémentaire.

  Le quatrième temps est une attente de la fin des autres branches venant en parallèle. Elle est gérée par la fin du parallélisme FPARALL qui a lancé ses branches. Une commutation des branches est assurée par TRET automatiquement mise en place.

  Le traitement correspond donc à la structure générale de la figure 4.22 où la branche générale s'occupe des entrées et sorties, des fonctions d'interfaces et des traitements généraux séquentiels.

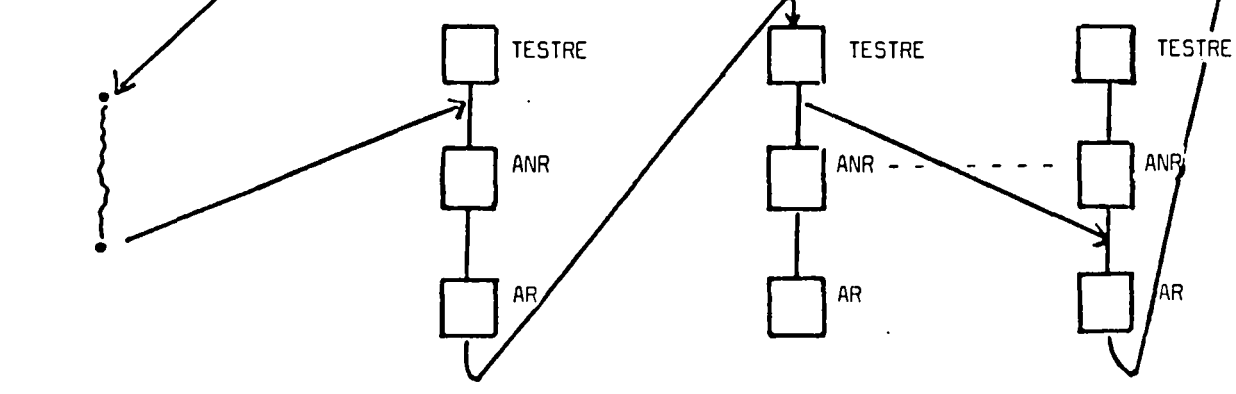

BRANCHE
GENERALE
BRANCHE 1
BRANCHE 2 - - - - -
BRANCHE n
TESTRE
ANR
AR
TESTRE
ANR
AR
TESTRE
ANR
AR

FIGURE 4.22

Le principe de l'interpréteur FORTHMITIS est donné figure 4.23. On notera que W, I, et IND fait référence à la branche en cours.

Pour illustrer l'usage de l'interpréteur, on considère l'exemple donné figure 4.24.

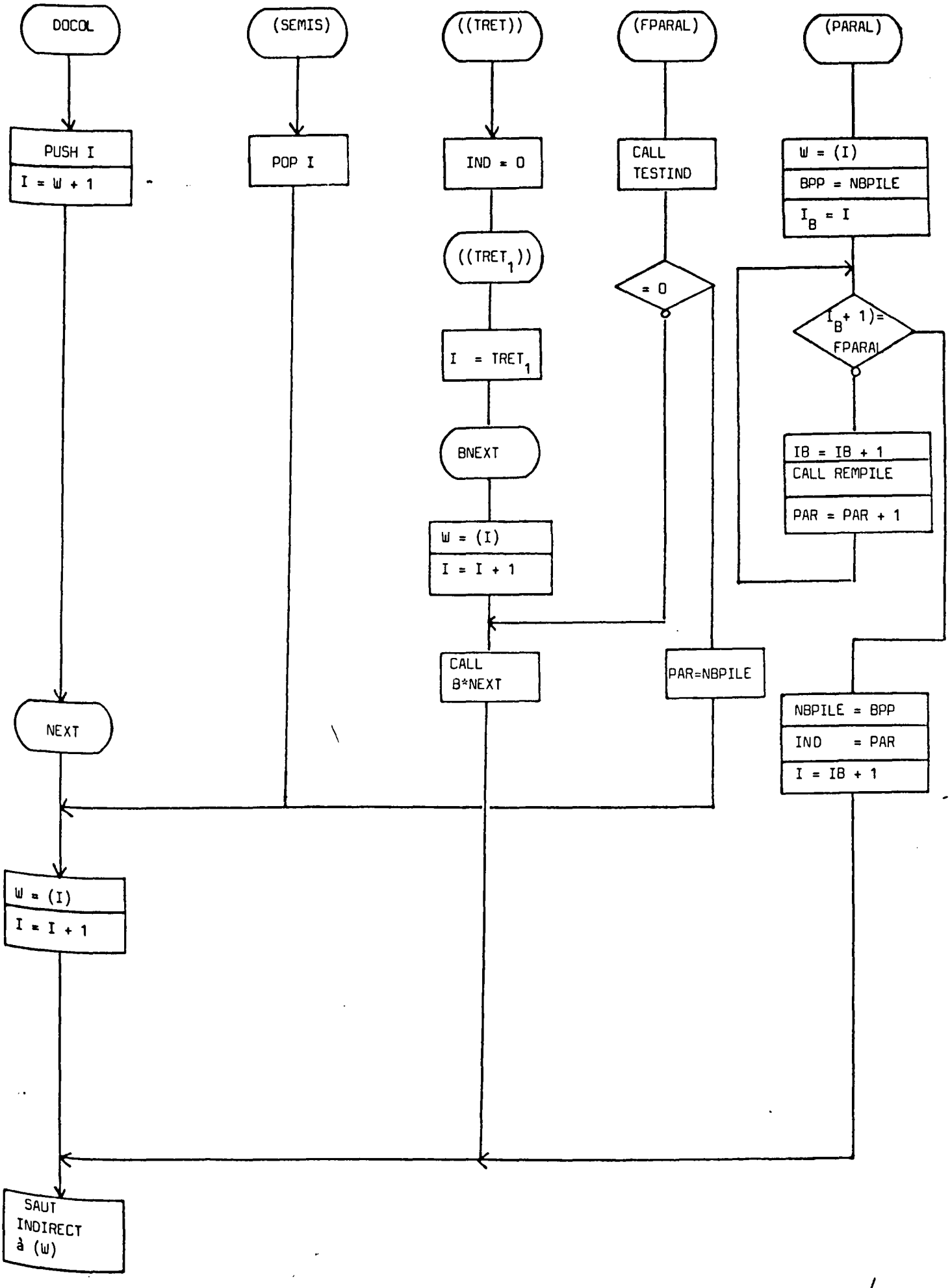

DOCOL
PUSH I
I = W + 1
(SEMIS)
POP I
((TRET))
IND = 0
((TRET1))
I = TRET1
BNEXT
W = (I)
I = I + 1
CALL B*NEXT
(FPARAL)
CALL TESTIND
= 0
PAR=NBPILE
(PARAL)
W = (I)
BPP = NBPILE
IB = I
IB + 1)= FPARAL
IB = IB + 1
CALL REMPILE
PAR = PAR + 1
NBPILE = BPP
IND = PAR
I = IB + 1
NEXT
W = (I)
I = I + 1
SAUT INDIRECT à (W)

Figure 4.23

./..

```
* OPE            :OPE1
  TYPE           :E
  LIBELLE

      PARALL
          *OPE2;
          *OPE3;
      FPARALL

) FOPE           :OPE1
```

```
* OPE            :OPE2
  TYPE           :E
  LIBELLE

     SEQ
         *OPEL1;
         *OPEL2;
     FSEQ

) FOPE           :OPE2
```

```
* OPE            :OPE3
  TYPE           :E
  LIBELLE

     SEQ
         *OPEL3;
         *OPE4;
     FSEQ

) FOPE           :OPE3
```

```
* OPE            :OPE4
  TYPE           :E
  LIBELLE

      PARALL
          *OPEL4;
          *OPEL5;
      FPARALL

) FOPE           :OPE4
```

Figure 4.24

Il y correspond l'implantation en mémoire de la figure 4.24 et l'évolution de la figure 4.25 à partir d'un programme dit moniteur qui est le noyau constant de l'interpréteur.

```
 0   DEBUT :   OPE1
 1             MONITEUR
 2   OPE1 :    DOCOL
 3             PARAL
 4             OPE2
 5             OPE3
 6             FPARAL
 7             SEMIS
 8   OPE2 :    DOCOL
 9             OPEL1
10             OPEL2
11             SEMIS
12   OPE3 :    DOCOL
13             OPEL3
14             OPEL4
15             SEMIS
16   OPE4 :    DOCOL
17             PARAL
18             OPEL4
19             OPEL5
20             FPARAL
21             SEMIS
22   OPEL1 :   DOCOL
23             TESTRE1
24             ANR1
25             AR1
26             SEMIS
27   OPEL2 :   DOCOL
28             TESTRE2
29    \        ANR2
30             AR2
31             SEMIS
32   OPEL3 :   DOCOL
33             TESTRE3
34             ANR3
35             AR3
36             SEMIS
37   OPEL4 :   DOCOL
38             TESTRE4
39             ANR4
40             AR4
41             SEMIS
42   OPEL5 :   DOCOL
43             TESTRE5
44             ANR5
45             AR5
46             SEMIS
```

Figure 4.25

	NBMLE	PAR	W1	I1	RS1	IND1	W2	I2	RS2	IND2	W3	I3	RS3	IND3
(SEMIS)	1	1		0	0									
NEXT	1	1	2	1										
OPER1 DOUBL	1	1		3	1									
NEXT	1	1	PARAL	4	1									
(PARAL)	1	2	8	6	1	2	12	TRET		1				
OPER2 DOUBL	1	2		9	1,6	2								
NEXT	1	2	22	10	1,6	2			//					
OPER4 DOUBL	1	2		23	1,6,10	2								
NEXT	1	2	TESTRE4	24	1,6,10	2			//					
(TESTRE4) BNEXT	2	2	ANR4	25	1,6,10	2			//					
OPER3 DOUBL	4	4			//			13	TRET	1				
NEXT	2	2					3C	14	TRET	1				
OPER5 DOUBL	2	2			//			33	TRET,14	1				
NEXT	2	2					TESTRE3	34	TRET,14	1				
(TESTRE3) BNEXT	1	2					ANR3	35	TRET,14	1				
(ANR2) NEXT	1	2	AR2	26	1,6,10	2			//					
(AR2) BNEXT	2	2	SEMIS	27	1,6,10	2			//					
(ANR3) NEXT	2	2		//			AR3	36	TRET,14	1				
(AR3) BNEXT	1	2	\	//			SEMIS	37	TRET,14	1				
OPER6 SEMIS	1	2		10	1,6	2			//					
NEXT	1	2	27	11	1,6	2								
OPER7 DOUBL	1	2		28	1,6,11	2			//					
NEXT	1	2	TESTRE7	29	1,6,11	2								
(TESTRE7) BNEXT	2	2	ANR2	30	1,6,4	2			//					
OPER3 SEMIS	2	2			//			14	TRET	1				
NEXT	2	2					16	15						
OPER4 DOUBL	2	2			//			17	TRET,15	1				
NEXT	2	2					PARAL	18	TRET,15	1				
(PARAL)	2	3			//			19	TRET,15	3	48	TRET		1
OPER4 DOUBL	2	3						38	TRET,15	3				
NEXT	2	3			//		TESTRE4	39	20	3	//			
(TESTRE4) BNEXT	3	3			//		ANR4	40	TRET,15 20	3		//		
OPER5 DOUBL	3	3			//				//			43	TRET	1
NEXT	3	3									TESTRE5	44	TRET	1
S(TESTRE5) BNEXT	1	3			//				//		ANR5	45	TRET	1

Figure 4.26

./..

	NSAUT	PAR	W1	I1	RS1	IND1	W2	I2	RS2	IND2	W3	I3	RS3	IND3
(ANIL) NEXT	1	3	AR2	31	1,6,11	2	ANR4	4C	TRET,15 ,16	3	ANR5	4F	TRET	1
(AR4) BNEXT	2	3	SENIS	32	1,6,11	2								
(ANR4) NEXT	2	3		//			AR4	41	TRET,15 ,20	3		//		
(AR4) BNEXT	3	3					SENIS	42						
(ANR5) NEXT	3	3		//				//			AR5	46	TRET	1
(AR5) BNEXT	1	3									SENIS	47	TRET	1
OPEC1 (SENIS)	1	3		11	1,6	2			//			//		
OPEC1 NEXT			SENIS	12	1,6	2								
OPEC1 (SENIS)				6	1	2			//			//		
OPEC1 NEXT	1	3	FPARAL	7	1	2								
OPEC2 (FPARAL)	2	3	FPARAL	7	1	2			//			//		
OPEC4 (SENIS)	2	3			//			20	TRET,15	3		//		
OPEC4 NEXT	2	3					FPARAL	21	TRET,15	3				
OPEC4 (FPARAL)	3	3			//				//			//		
OPEC5 (SENIS)	3	3			//				//			TRET		1
OPEC5 NEXT	3	3									(TRET)	TRET+1		1
(TRET))	3	3										TRET2		0
BNEXT	1	3									(TRET2)	TRET2 +1		
OPEC3 (FPARAL)	2	3	FPARAL	7	1	2			//			//		
OPEC6 (FPARAL)	2	2			//		SENIS	22	TRET, ,15	3		//		
(SENIS) NEXT	2	2			//		SENIS	15 16	TRET TRET	3 3		//		
OPEC3 (SENIS) NEXT	2	2			//		(TRET)	TRET TRET+1		3		//		
(CTRET)) ((TRET2))	1	2			//		(TRET1)	TRET2 TRET2 +1		0 0			//	
(FPARAL, CSENIS) NEXT MONITEUR.	1 1 1	1 1 1	SENIS MONITEUR	8 1 2	1	2								

Figure 4.26 (fin)

4.3.3. Implantation de l'interpréteur avec parallélisme.

- La mise en place de l'interpréteur avec parallélisme se fait en langage machine 8085, à partir d'une compilation entrelacée FORTH dont les écrans sont donnés en annexe 2.

  Elle utilise la structure des données suivantes pour laquelle on donne le nom, le nombre d'octets, la valeur initiale, le rôle.

  Constantes.

  NPILE - 2 octets - 10 - nombre maximum de branches parallèles

  TPILE - 2 octets - 20 - taille de la pile de retour pour chaque branche.

  Variables.

  PAR - 1 octet - 1 - indicateur de profondeur de parallélisme

  NBPILE - 1 octet - 1 - numéro de la branche en cours de traitement

  BPP - 1 octet - Ø - tampon de NBPILE

  IB - 2 octets - Ø - tampon de I

  I1 - 2 octets - Ø)
  W1 - 2 octets - Ø) sauvegarde de I, W, du pointeur de
  PPILE 1 - 2 octets - Ø) pile de retour et de IND pour
  IND1 - 1 octet - Ø) la première branche

  PIND - 2 octets - IND1 - variable contenant l'adresse de l'indicateur en cours de traitement.

PILERET : NPILE fois la structure suivante :

IN - 2 octets	- Ø	)
		)
Wn - 2 octets	- Ø	)
		) sauvegarde pour la branche n
INDn - 1 octet	- Ø	)
		)
PPILEn - 2 octets	- Ø	)
PILEn - 2*TPILE	- Ø	- pile de retour

Les sous-éléments de PILERET ne sont pas définis par leur dénomination, mais comme un tableau unique. Il convient d'initialiser tous les pointeurs de pile PPILEn à la valeur de la dernière case de la pile.

4.4. CAS DES IMPLANTATIONS MULTINIVEAUX

La description, telle que nous l'avons envisagée, ne tient pas compte de la structure du système d'implantation. Les technologies actuelles permettent la mise en oeuvre de dispositifs répartis, ce qui entraîne une décomposition de la description.

Dans l'absolu, on pourrait considérer une coupe de l'arborescence en sous-ensembles sous plusieurs niveaux ; toutefois, ceci ne nous paraît pas très réaliste, car, à notre avis, il convient de tenir compte de la notion de phase procédé introduite précédemment.

Associé à chaque phase, il existe un moniteur de phase qui utilise des informations d'état des opérations de la phase. L'entité minimum la plus pratique à manipuler est donc la phase avec son moniteur, car ceci évite la transmission des informations d'état.

On peut alors envisager plusieurs structures réparties :

- à découpage monophase et coordination verticale,
- à découpage multiphase et coordination verticale,
- à découpage monophase et coordination horizontale,
- à découpage multiphase et coordination horizontale.

Le premier cas suit de très près la description, car la coordination verticale coïncide avec la notion de moniteur général. Dans le deuxième cas, il peut être nécessaire d'éclater le moniteur général en plaçant des moniteurs généraux locaux assurant les synchronisations locales et un synchroniseur supérieur.

Les systèmes à coordination horizontale sont plus difficiles à mettre en oeuvre car fortement dépendants des protocoles d'échange. En effet, le problème est de réussir à répartir le moniteur général ou la synchronisation supérieure. Cette répartition ne peut se faire que par un protocole sûr d'échange, comme par exemple la technique de la page tournante dans les réseaux d'automates

4.5. CONCLUSION

Dans ce chapitre, nous avons montré comment il est possible de définir une méthode d'implantation systématique de notre modèle multiniveaux sur des systèmes aussi différents que calculateurs et automates programmables.

Dans un cas comme dans l'autre, le passage du modèle théorique avec ses hypothèses temporelles très strictes au modèle de réalisation nécessite de prendre des précautions, en particulier sur le traitement des régimes transitoires. Un modèle multiniveaux introduit effectivement de nombreux transitoires de relation entre les niveaux.

D'autres méthodes pourraient être introduites généralisant les méthodes mononiveau. Mais l'essentiel est surtout de voir qu'une démarche complète peut être faite depuis la spécification jusqu'à la matérialisation.

L'implantation proposée sur calculateur est par ailleurs fort intéressante, car on peut constater que les notions de base du langage EASYMITIS peuvent être mises en oeuvre moyennant des adjonctions très simples à un interpréteur FORTH existant.

. CONCLUSION GENERALE

Dans ce travail nous avons tenté la définition du poste d'Automatisation et d'Instrumentation pour les processus par lots et nous en avons montré la faisabilité. Pour qu'une telle définition soit significative industriellement, il a fallu qu'elle soit validée sur des exemples réels et de nombreuses itérations avec nos partenaires industriels ont été nécessaires.

Un tel poste ne peut en effet être l'oeuvre d'un chercheur solitaire mais le résultat d'une confrontation entre expériences diverses : conceptions d'automatisme, Informaticiens producteurs de logiciels et universitaires. Dans ce sens, la collaboration autour de ce travail a été exemplaire.

Le projet n'est pas achevé pour autant. Ce mémoire marque uniquement la fin de la phase prototype. De gros efforts restent encore à faire pour en réaliser la diffusion sur le marché.

Notre mémoire ouvre par ailleurs quelques perspectives prometteuses tant sur le plan de la spécification que de l'implantation. Nous souhaitons en effet continuer l'exploration de la notion de spécification experte soit relativement au calcul des précédents temporels soit en langage orienté objet. La poursuite de travaux dans ce sens ne remet pas en cause ce qui a été fait mais constitue un niveau supérieur de définition. Nous pensons également, sur le plan de l'implantation, que les technqiues entrelacées avec parallélisme que nous avons développées constituent le moyen direct de passer du poste de bureau d'étude à un véritable poste d'exécution.

Le Poste d'Automatisation et d'Instrumentation est un système ouvert qui ne peut que s'étoffer et s'enrichir au cours du temps.

CHAPITRE IV

A N N E X E 1

FORTH

S O M M A I R E

1. GENERALITES

Le langage de programmation de la "FOURTH GENERATION" (en abrégé FORTH) est un langage puissant, compromis entre un langage de très haut niveau et le langage machine.

Il fut inventé par Charles H. MOORE vers les années 60. Il utilise la notation post-fixée ou polonaise inverse. Les paramètres doivent précéder les opérateurs. Il présuppose l'existence de deux piles de mots de 16 bits (la pile des paramètres et la pile de retour).

Dans la version de base, on admet :

- les entiers en simple précision sur 16 bits,
- les entiers en double précision sur 32 bits,
- les caractères ASCII.

Le type d'entités manipulées se trouve dans l'usage de définitions spécifiques.

Le dictionnaire est un élément fondamental du système FORTH. Il apparaît comme une liste chaînée dont l'élément de base est le mot.

Le vocabulaire de base comprend environ 200 mots.

1.1. Utilisation en interpréteur.

On exécute directement les mots présents dans le dictionnaire simplement en les nommant.

Le séparateur de mots est l'espace. Par exemple, l'addition de 2 nombres s'effectuera de la manière suivante :

- introduction des nombres puis de l'opérateur,
- impression du résultat.

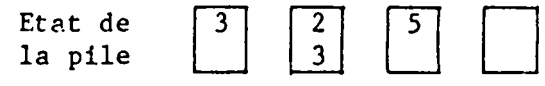
Exemple : 3 2 + .

Les nombres vont être empilés.

+ est un mot Forth qui remplace les 2 nombres en sommet de pile par leur somme.

. mot du dictionnaire qui prend le sommet de pile et l'affiche.

3 2 + .



1.2. Le dictionnaire

La possibilité d'accroître le vocabulaire de base est une des particularités de FORTH.

L'utilisateur peut adjoindre de nouveaux mots en définissant une nouvelle entité à partir des anciennes.

Celle-ci va se retrouver sous forme compilée dans le dictionnaire, et l'utilisateur pourra alors l'exécuter simplement en la nommant, où l'employer dans d'autres définitions.

L'évolution du dictionnaire est limitée à la taille de la mémoire.

Cette possibilité d'adjonction de mots au dictionnaire constitue un premier niveau d'extensibilité.

1.3. Les familles de mots.

La création de familles de mots constitue un second niveau d'extensibilité.

On définit un comportement à la définition et un à l'exécution.

Exemple : La famille VARIABLE pourra s'utiliser de la façon suivante :

- à la définition : VARIABLE TOTO entraîne l'introduction dans le dictionnaire du nom TOTO et réserve un emplacement pour un mot de 16 bits;
- à l'exécution : TOTO entraîne la mise en "pile des paramètres" de l'adresse du mot de 16 bits précédemment réservé.

On peut définir autant de nom de variables que l'on veut avec, à chaque fois, la même structure dans le dictionnaire.

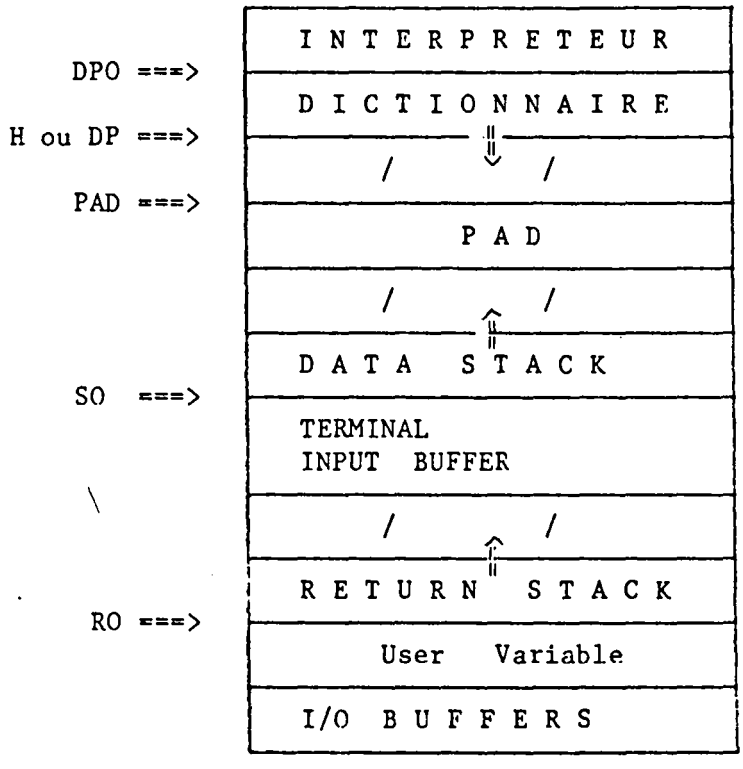
2. STRUCTURE d'un système FORTH.

2.1. Les différents composants.

La base du système est le dictionnaire que l'on vient compléter par compilation.

Initialement, il comprend les mots clés principaux en langage machine et les mots secondaires écrits en FORTH.

Structuration mémoire d'un système FORTH :



> IN ou input stream buffer
Repère la position courante
dans le flot d'entrée

2.2. Le dictionnaire.

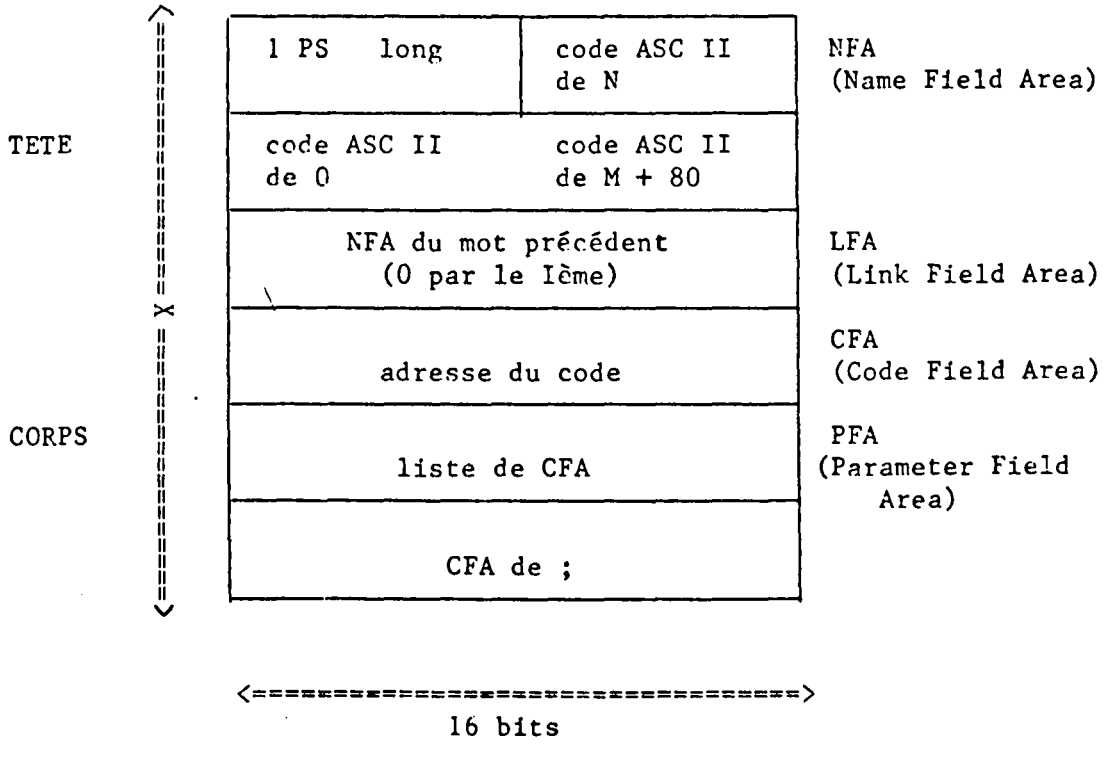
2.2.1. Structure.

Le dictionnaire est une chaîne de définitions.

Toutes les définitions comportent :

- un nom,
- un chaînage,
- un code pointeur,
- un champ de paramètres.

L'entrée dans le dictionnaire de la définition NOM a donc la structure suivante :



La tête comprend :

* P : "précédence bit" permet de rendre le mot immédiat (voir + loin)

* S : "Smudge bit" indique que la définition est valide ou non (bit = 1)

* Longueur (5 bits) : elle est généralement inférieure à 31.

* On note ensuite le code ASCII des 3 premières lettres ou du mot complet.

* Chaînage au mot précédemment introduit dans le dictionnaire.

Le corps est constitué par :

* CFA : contient un pointeur vers un programme définissant ce qu'il convient d'effectuer lors de l'exécution du mot.

* PFA : début d'une liste des CFA des constituants du mot que l'on définit.

* CFA de ; termine la compilation du mot dans le dictionnaire.

2.2.2. Opérations de base sur le dictionnaire.

2.2.2.1. Adjonction de mots au dictionnaire

L'adjonction d'un mot au dictionnaire ou "compilation" consiste en la création d'une nouvelle entrée, dans le dictionnaire. Celle-ci sera chaînée à la définition précédente.

La compilation utilise principalement la COLON DEFINITION

: nom Définition ;

où Définition est une suite de mots déjà définis dans le dictionnaire et de nombres.

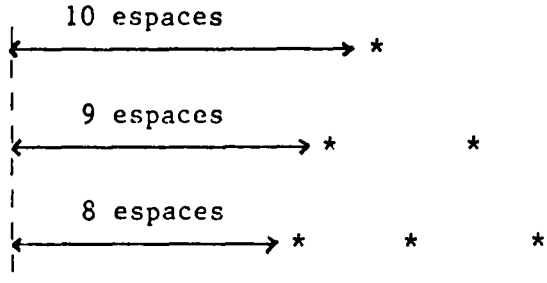
: fait passer en mode compilation. Pour les mots connus rencontrés dans la définition, le système place, dans le dictionnaire, les CFA de ces mots.

Dans le cas d'un nombre, il place un code spécial suivi du nombre écrit en binaire, après transformation à partir de l'entrée, relativement à la base en cours.

Dans le cas d'un nombre, il place un code spécial suivi du nombre écrit en binaire, après transformation à partir de l'entrée, relativement à la base en cours.

Exemple : Affichage d'un triangle sur l'écran.

Pour obtenir la figure suivante :



Il faut définir trois mots en FORTH : ETOILE - MARGE - TRIANGLE.

```
- : ETOILE   42   EMIT   ;
```

où : définit le début d'une procédure.

ETOILE est le nom de cette procédure.

\ 42 est le code ASCII décimal de *.

EMIT est un mot qui imprime sur l'écran le symbole dont la valeur ASCII est en sommet de pile.

; définit la fin de la procédure.

```
- : MARGE  CR  SPACES  ;
```

où CR entraîne un retour à la ligne du curseur.

SPACES imprime le nombre n d'espaces placé en sommet de pile (n doit donc précéder SPACES).

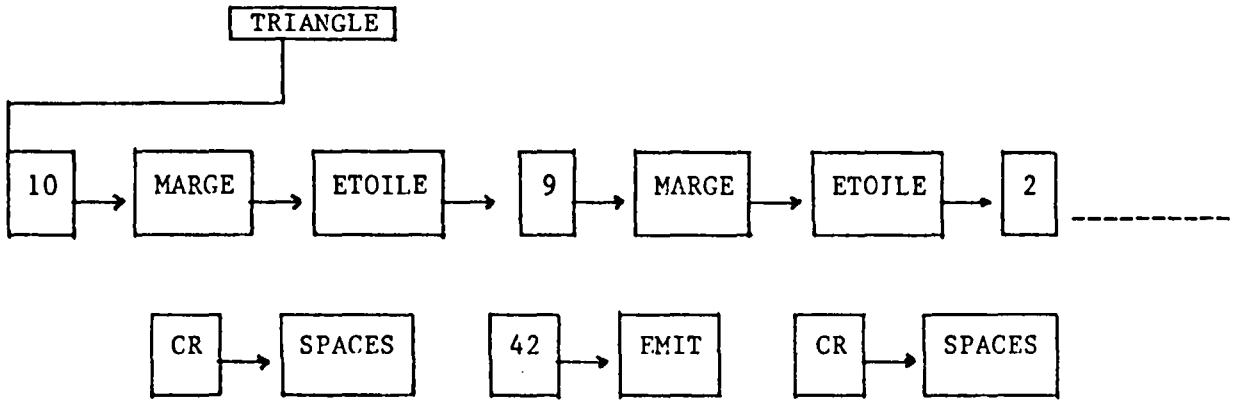
MARGE positionne le curseur à la ligne suivante, à n espaces du bord de l'écran.

```
- : TRIANGLE  10 MARGE ETOILE
              9 MARGE ETOILE 2 SPACES ETOILE
              8 MARGE ETOILE 2 SPACES ETOILE
                             2 SPACES ETOILE
```

Remarque : L'emploi d'une structure de boucle permet d'optimiser le programme TRIANGLE.

./..

Structure arborescente :



Les feuilles terminales de l'arbre sont soit des nombres, soit des définitions de base.

1) Analyse descendante :

On décompose cette action jusqu'à obtention de mots du dictionnaire.

2) Ecriture ascendante :

A partir des éléments de base, on constitue des mots de plus en plus précis, afin d'obtenir le mot réalisant l'action souhaitée.

2.2.2.2. Notion de vocabulaire.

L'introduction d'un mot dans le dictionnaire se fait à l'intérieur d'un vocabulaire. Généralement, il existe 3 vocabulaires de base entrelacés dans le dictionnaire :

FORTH - EDITOR - ASSEMBLEUR

L'utilisateur peut adjoindre autant de vocabulaire qu'il le souhaite. Le même mot peut être utilisé dans plusieurs vocabulaires avec des significations différentes. On dispose de 2 pointeurs : CONTEXT et CURRENT.

. CONTEXT pointe vers le vocabulaire lors d'une recherche dans le dictionnaire.

. CURRENT contient l'indice du vocabulaire où une nouvelle définition est compilée.

Chaque vocabulaire est chaîné à un parent (le vocabulaire CURRENT au moment de sa création).

VLIST permet de connaître pour un vocabulaire donné :

. les différents éléments de ce vocabulaire et ceux de ses ascendants.

L'exemple suivant illustre la notion de vocabulaire :

* Création d'un vocabulaire VOCABULARY ANIMAUX IMMEDIATE (la signification de IMMEDIATE sera donné ultérieurement).

qui crée le vocabulaire ANIMAUX sous le vocabulaire CURRENT. Cette création a pour effet l'adjonction de la structure suivante, au dictionnaire :

NFA	en-tête ANIMAUX
	Chaînage vers le mot précédemment introduit dans le dictionnaire
	CFA de VOCABULARY
PFA	Adresse de la tête de chaîne fictive d'un mot dont le nom est ' '
PFA+2	NFA du dernier mot créé dans le vocabulaire
PFA+4	Chaîne vers le vocabulaire précédent

L'adresse du PFA+4 du dernier vocabulaire introduit est noté dans une variable VOC-LINK.

* L'adjonction de définitions sous ce vocabulaire se fait par :

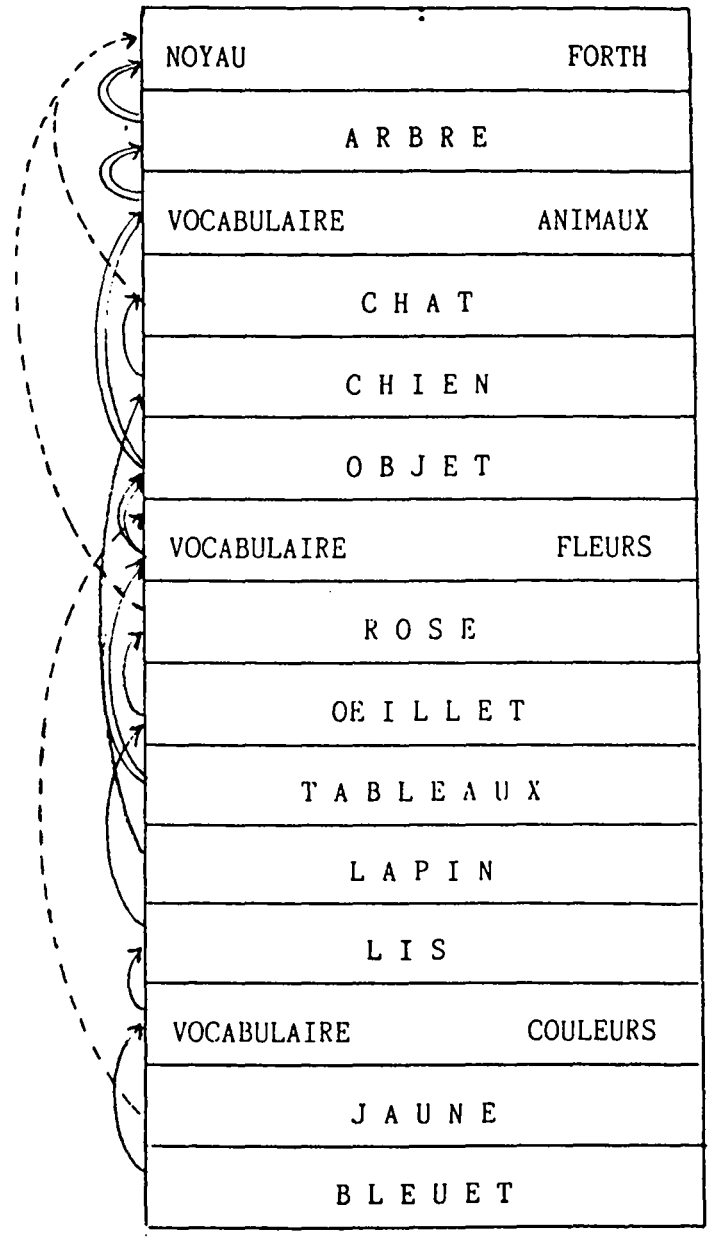
ANIMAUX DEFINITIONS

Il suffit alors d'introduire les définitions.

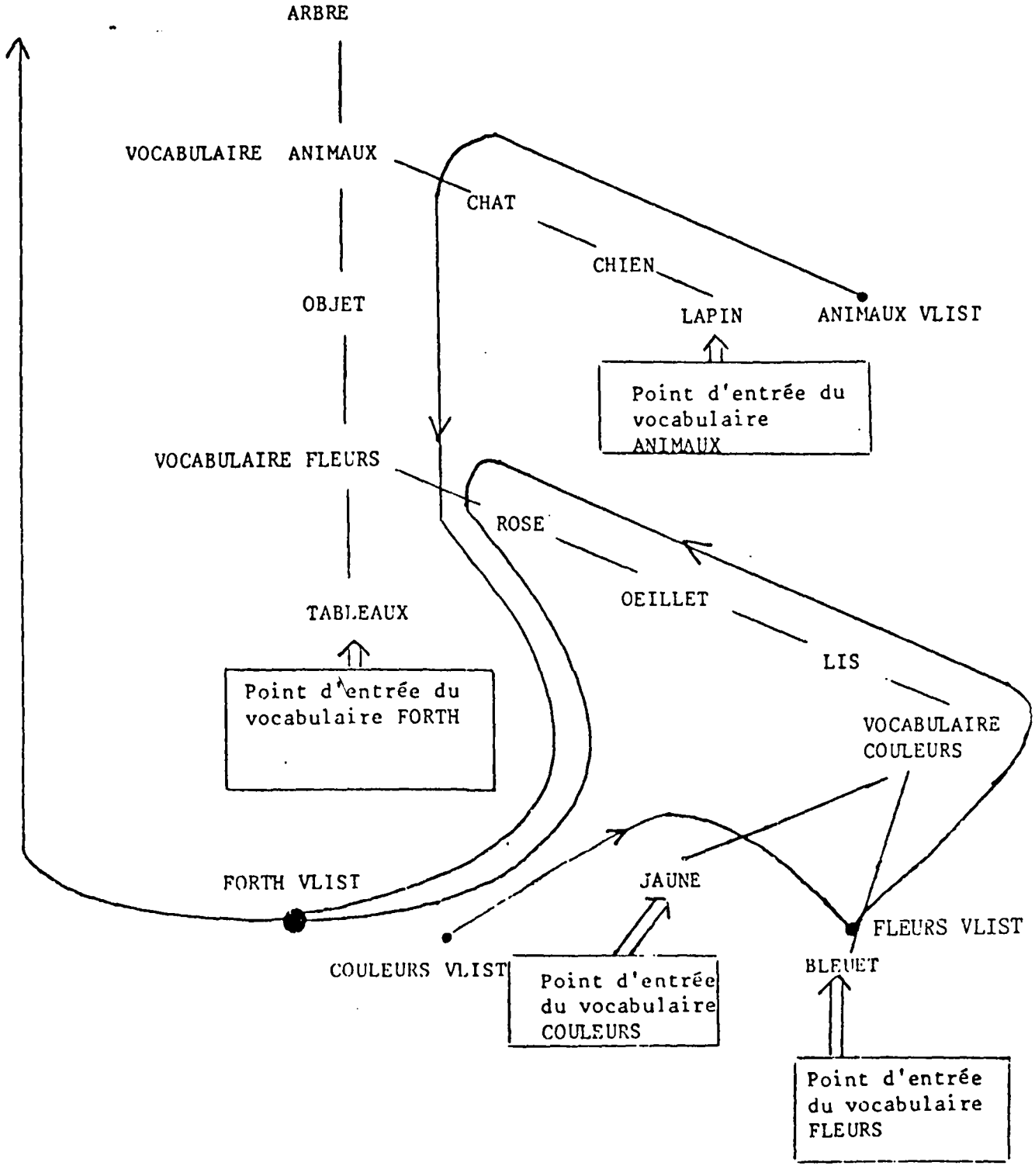
Le dictionnaire est ramifié en vocabulaire, structuré en chaîne linéaire, partant du nom du vocabulaire jusqu'à l'élément terminal accessible.

Cette structure est mise en évidence par l'exemple suivant :

```
: ARBRE ;
VOCABULARY ANIMAUX IMMEDIATE
ANIMAUX DEFINITIONS
: CHAT ;
: CHIEN ;
FORTH DEFINITIONS
: OBJET ;
VOCABULARY FLEURS IMMEDIATE
FLEURS DEFINITIONS
: ROSE ;
: OEILLET ;
FORTH DEFINITIONS
: TABLEAUX ;
ANIMAUX DEFINITIONS
: LAPIN ;
FLEURS DEFINITIONS
: LIS ;
VOCABULARY COULEURS IMMEDIATE
COULEURS DEFINITIONS
: JAUNE .
FLEURS DEFINITIONS
: BLEUET ;
```



STRUCTURE DANS LE DICTIONNAIRE



SCHEMA DE L'ARBORESCENCE FORMEE PAR LES VOCABULAIRES

./.

2.2.2.3. Recherche d'éléments dans le dictionnaire.

Le repérage d'un élément du dictionnaire (dans le vocabulaire CONTEXT) est facilité par les définitions de base FORTH suivantes :

- @ et ! permettent respectivement de charger dans la pile et de ranger à partir de la pile la valeur d'un élément (sur 16 bits) à une adresse donnée.
  C@ et C ! sont utilisés pour les bytes (8 bits).

- HERE donne l'adresse du premier emplacement libre dans le dictionnaire. La définition de HERE est la suivante :

  ```
  : HERE  H @  ;      (H ou DP)
  ```

- ' NOM - recherche l'adresse d'un nom dans le dictionnaire.

  Lorsque le mot existe, son PFA est empilé (en cas d'échec ABORT fait une initialisation du système).

- A partir du PFA d'un nom, le CFA est obtenu par

  ```
  : CFA  2 -  ;
  ```

  Le CFA permet d'exécuter, de faire un vidage de mémoire, de lire ou ranger une valeur.

- Le LFA se déduit du CFA par

  ```
  : LFA  2  -  ;
  ```

- A partir du CFA, on obtient le NFA par la définition NFA.

- LATEST empile le NFA du dernier mot créé dans le vocabulaire CURRENT.

2.2.2.4. Suppression d'éléments dans le dictionnaire.

EMPTY permet de vider complètement le dictionnaire utilisateur. Toutefois, il est possible d'en supprimer seulement une partie, en utilisant FORGET mot.

FORGET entraîne la suppression de toutes les définitions de l'entrée du dictionnaire jusqu'au mot inclus.

Lors de l'adjonction de définitions au dictionnaire, on insère généralement un mot vide.

Celui-ci sert de repère, en cas de suppression d'une partie des définitions du dictionnaire.

2.2.2.5. Forçage d'éléments dans le dictionnaire ou modification du comportement des définitions.

Ces différentes opérations sont réalisées par les mots FORTH suivants :

- CREATE nom : créé simplement l'entrée nom dans le dictionnaire (tête et code pointeur).

  A l'exécution, nom retourne l'adresse de son PFA.

- n ALLOT réserve n bytes dans le dictionnaire.

  Définition de ALLOT

  ```
  : ALLOT  HERE + DP ! ;
  ```

- SMUDGE permet de valider une définition, c'est-à-dire de forcer le "SMUDGE bit" de l'en-tête.

- LITERAL ne s'utilise que dans une définition. A la compilation place dans le dictionnaire, le CFA de la partie exécutoire de LITERAL suivi de la valeur en sommet de pile. A l'exécution, la valeur précédemment compilée est placée en sommet de pile.

- , vient placer la valeur en sommet de pile à l'adresse définie par HERE.

  Définition de ,

  ```
  : HERE ! 2 ALLOT ;
  ```

  C, est utilisé pour les bytes (8 bits)

- WORD lit un mot à partir du flot d'entrée (Input Stream Buffer) en tenant compte d'un caractère ASCII délimiteur placé en sommet de pile.

  Il range ce mot à l'adresse HERE avec le compte de caractères dans le premier byte et laisse l'adresse en pile.

- IMMEDIATE rend immédiat le dernier mot entré dans le dictionnaire (forçage de "Precedence bit" de l'en-tête).

  Exemple : JOUR . "JOUR" ; IMMEDIATE

  : MOIS JOUR . "HEURE" ;

  La compilation de MOIS provoque l'impression de JOUR.

  Son exécution entraîne l'affichage de HEURE.

  L'emploi de IMMEDIATE permet de définir des mots ayant un comportement différent à la compilation et à l'exécution.

  Les structures de contrôle utilisent cette propriété.

- [fait sortir du mode compilation

-] force en mode compilation.

- SMUDGE permet de valider une définition, c'est-à-dire de forcer le "Smudge bit" de l'en-tête.

- COMPILE met dans le dictionnaire le CFA du mot s'il existe. S'il s'agit d'un nombre, il place le CFA de LITERAL, puis le nombre.

  Un autre cas constitue une erreur.

- [COMPILE] permet d'utiliser, dans une définition, un mot immédiat comme un mot ordinaire

2.3. Les familles de mots.

Une famille de mots se définit par :

: NOM FAMILLE Traito < BUILDS Trait 1 DOES > trait 2 ;

où :

- Traito est utilisé pour l'impression de commentaires à imprimer lors de l'utilisation de NOM FAMILLE.
- Trait 1 indique les actions à accomplir lors de la définition d'un nouveau membre de la famille.
- Trait 2 précise le travail à effectuer lors de l'exécution d'un membre de la famille.

La définition de la famille est stockée dans le dictionnaire. Elle servira de calque, pour définir d'autres membres de cette famille.

Exemple : Création de la famille VARIABLE

: VARIABLE < BUILDS 2 ALLOT DOES > ;

VARIABLE VAL crée dans le dictionnaire la structure suivante :

VAL	NFA	< BUILDS crée l'entête, fait le chaînage et réserve 2 cases puis compile 2 ALLOT en réservant 2 bytes
chaînage	LFA	
adresse de la partie exécutoire de DOES >	CFA	
valeur	PFA	DOES> place un pointeur dans les 2 cases réservées

A l'exécution de VAL, on va au CFA. Celui-ci place l'adresse de PFA en sommet de pile.

3. JEU DE DEFINITIONS DE BASE

Les paragraphes précédents ont donné la structure de FORTH et les définitions jouant sur le dictionnaire. L'ensemble de base comprend par ailleurs des définitions permettant les traitements classiques de l'informatique.

3.1. Arithmétique et pile.

Les opérations arithmétiques se font par l'intermédiaire de la pile et et portent sur les entrées simple ou double précision. On trouve par exemple addition, soustraction, multiplication, division entière (en quotient et reste), mise à l'échelle, incrémentation, décalage, valeur absolue, minimum, maximum, diverses comparaisons.

La pile de paramètres étant le point central du système, de nombreuses définitions ont pour objet de modifier l'ordre dans la pile ou de pratiquer des échanges entre pile des paramètres et pile de retour.

3.2. Structure de contrôle

Nous avons pu voir que l'organisation de FORTH entraîne une structuration arborescente des programmes. C'est pourquoi les définitions de contrôle conserve ce caractère structuré. On y trouve :

IF - THEN - ELSE

BEGIN - UNTIL

DO - LOOP

3.3. Mémoire

Une des grandes puissances de FORTH est de pouvoir rester très proche de la machine et des informations en mémoire. En effet, des définitions permettent de manipuler tout emplacement de la mémoire, que ce soit dans le dictionnaire, les piles ou les divers buffers. Les définitions ! C C@ C ! déjà rencontrées en sont un exemple typique.

3.4. Entrées - sorties et disque.

Il est possible de dialoguer avec la machine, soit pour l'introduction de valeurs dans n'importe quelle base, soit pour l'affichage de résultat, soit pour les manipulations de chaînes de caractères.

De plus, une structure de mémoire virtuelle par bloc de 1 K (soit sur écran) existe pour la sauvegarde de la source des définitions compilées ou pour contenir des données. On peut très facilement charger, ranger, chaîner des blocs par l'intermédiaire des I/O BUFFERS.

3. JEU DE DEFINITIONS DE BASE

Les paragraphes précédents ont donné la structure de FORTH et les définitions jouant sur le dictionnaire. L'ensemble de base comprend par ailleurs des définitions permettant les traitements classiques de l'informatique.

3.1. Arithmétique et pile.

Les opérations arithmétiques se font par l'intermédiaire de la pile et et portent sur les entrées simple ou double précision. On trouve par exemple addition, soustraction, multiplication, division entière (en quotient et reste), mise à l'échelle, incrémentation, décalage, valeur absolue, minimum, maximum, diverses comparaisons.

La pile de paramètres étant le point central du système, de nombreuses définitions ont pour objet de modifier l'ordre dans la pile ou de pratiquer des échanges entre pile des paramètres et pile de retour.

3.2. Structure de contrôle

Nous avons pu voir que l'organisation de FORTH entraîne une structuration arborescente des programmes. C'est pourquoi les définitions de contrôle conserve ce caractère structuré. On y trouve :

IF - THEN - ELSE

BEGIN - UNTIL

DO - LOOP

3.3. Mémoire

Une des grandes puissances de FORTH est de pouvoir rester très proche de la machine et des informations en mémoire. En effet, des définitions permettent de manipuler tout emplacement de la mémoire, que ce soit dans le dictionnaire, les piles ou les divers buffers. Les définitions ! C C C ! déjà rencontrées en sont un exemple typique.

3.4. Entrées - sorties et disque.

Il est possible de dialoguer avec la machine, soit pour l'introduction de valeurs dans n'importe quelle base, soit pour l'affichage de résultat, soit pour les manipulations de chaînes de caractères.

De plus, une structure de mémoire virtuelle par bloc de 1 K (soit sur écran) existe pour la sauvegarde de la source des définitions compilées ou pour contenir des données. On peut très facilement charger, ranger, chaîner des blocs par l'intermédiaire des I/O BUFFERS.

BIBLIOGRAPHIE

(ALB) J. ALBUKERQUE — Spécification et validation d'automatismes logiques interconnectés
- Th.doctorat de 3° cycle : EEA Automatique Toulouse III 1982 (2746) -

(BAB) R.G. BABB, R. KIEBURTZ — Workshop on Models and Languages for Software Specification and Design
- IEE Computer, March 1985 -

(BOR) A. BORGIDA, J. MYLOPOULOS — Knowledge Representation as the Basis for Requirements Specifications
- Computer IEE, 1985 -

(BRA) C.W. BRAMS — Réseaux de Pétri - Théorie et pratique.
. Tome 1 : théorie et analyse
. Tome 2 : Modélisation et application
- Masson 1983 -

(BRO) F. BROWAEYS, H. DERRIENNIC, P. DESCLAUD, and co — Sceptre : proposition du noyau normalisé pour les exécutifs,temps réel.
- TSI volume 3, n° 1, 1984 -

(BRO) J.L. BROWN, R.L. BEARD — CoSyn TM
A new Tool for Process Control

(BUL) G. BULL, R. MITCHELL — Exception Handling Considered Harmful
IFAC Real Time Programming 1983
Session 4, Languages and Language implementation

(CAT) R.G.G. CATTEL — Using Machine Descriptions for Automatic Derivations of code Generators
- Information Technology, North Holland 1978 -

(CAZ) C. CAZALOT, J.M. DUMAS, P.M. GROJEAN, B. CARRIERE, F. PRUNET
Conception assistée des programmes de contrôle des automatismes
- Computer Aided of Sequential Processes,
Laboratoire d'Automatique et de Microélectronique,
UST Languedoc Montpellier -

Congrès automatique 1983 sur productique et robotique intelligente (237-47)
15-17 novembre 1983, Besançon

(DAN) P. DANGELO
Structured analysis simplifies modern Software design
- Electronic Design September 3, 1981 -

(DEM) M. DEMUYNCK, B. MEYER
Les langages de spécifications EDF
- Bulletin de la Direction des études et recherches, série C,
Mathématiques informatique n° 1, 1979 (39-60) -

(DEN) H. DENEUX
Contribution à l'étude des réseaux d'automates programmables interconnectés
- Thèse Docteur Ingénieur,
Institut National Polytechnique de Grenoble,
9 février 1983 -

(FIH) S.L. FIHN, J.A. NYQUIST
Specifying Batch Process Control Strategies :
A structured Approach Intech ,Octobre 1982

(GIR) B. GIRARD, G. MICHEL
Les langages évalués pour le temps réel.
- Nouvel automatisme, janvier/février 1978 -

(GIR) C. GIROD
Sur la conception et la réalisation d'un logiciel de simulation des réseaux de Pétri
- Thèse Docteur Ingénieur informatique,
25 septembre 1984 -

(GRO) C. GROSS
Progiciels et langages de contrôle commande
- Electronique industrielle n° 39, octobre 1982 -

(HAA) V.H. HAASE
Real time behaviour of programs IEEE Transactions on Software Engineering,vol.SE 7, N° 5, septembre 1981

(HOM) G. HOMMEL
Software Production for Computer Controlled Automation Systems
- IFAC/IFIP Conference S23-7, 1980, 14-17 octobre,
Dusseldorf, Germany -

(KRA) J. KRAMER, J. MAGEE, M. SLOMAN, A. LISTER
Conic and integrated approach to distributed computer control systems
- IEE Proc. Vol 130,PT E n° 1, January 1983 -

(LAH) N. LAHMAR
Sur un système de gestion de processus multitâches : application à la commande et à la simulation
- Thèse Docteur troisième cycle électronique, 9 février 1982 -

(LAM) A.V. LAMSWEERDE
Automatisation de la production de logiciels d'application : quelques approches :
. 1ère partie, vol. 1, n° 6
. 2ème partie, vol. 2, n° 1
. 3ème partie, vol. 2, N° 2
TSI

(LEV) N.G. LEVESON
Software Safety in Computer Controlled Systems
- Computer,vol.17, n° 2, 48-55, February 1984 -

(LUD) J. LUDEWIG
Espresso - A system for Process Control Software Specification
IEEE Transactions on Software Engineering, vol SE.9 , n° 4, july 1983 -

(MAC) I.M. MACLEOD
The application of Modern Software Engineering Techniques in the development of a process control Package
IFAC/IFIP Symposium 393-7, 1983, 5 - 8 octobre 1982, Madrid, Spain.

(MAN) M. MANOFF
Control Software comes to Personal
- Computers Control Engineering, march 1984 -

(MIC) G. MICHEL, G. LAURGEAU, B. ESPIAU
Les automates programmables
- Dunod technique 1979 -

(MOA) M. MOALLA, R. DAVID
Extension du Grafcet pour la représentation de systèmes temps réel complexes
- R.A.I.R.O. Automatique/systems analysis and Control 1981, vol 15, n° 2, Bordas Dunod -

(PAR) J.P. PARSY
Dispositif interactif d'aide à la conception des programmes de commande de processus logiques industriels
- Docteur 3ème cycle automatique, thèse 1er juin 1981 -

(PET) S.L. PETERSON
Pétri net theory and the modeling of systems
- Prentice Hall, 1981 -

(RUB) M. RUBINOFF
M.C. YOVITS
Advances in Computers, vol 15
- Academie Press, 1976 -

(SAB) J.P. SABATHE
Description d'une méthode utilisée pour implémenter des fonctions d'automatismes dans un minicalculateur.
- Journées AFCET Informatique.
Programmation globale des synchronisations dans les applications en temps réel, 3/4 novembre 1977 -

(SEV) G. SEVERNS
J. HEDRICK
Planning control methods for batch processes
- Chimical Engineering, april 18, 1983 -

(SHA) M.J. SHAH
F. SCHNEIDER
Control Hierarchy in a distributed Process Control System.
Software for Computer Control 1982
Proceedings of the third IFAC/IFIP Symposium (133-8)
5-8 octobre 1983, Madrid -

(SIL) M. SILVA SUAREZ
Contributions à la synthèse programmée des automatismes logiques.
- Thèse Docteur Ingénieur, Grenoble, juin 1978 -

(SNE) H.M. SNEED
Softing Software Engineering System.
- J.Syst.and Software (USA), vol.3, n° 1, 63-76, March 1983 -

(STE) H.V. STEVSLOFF
Advanced Real Time Languages for Distributed Industrial Process Control
- IEEE February 1984 - Computer -

(SYL) J. SYLVAN
Control Software for Factory automation Computer Design
April 21, 1983.

(TAK) H. TAKAHASHI
An automatic Controller Description Language
- IEEE, Transactions on Software Engineering,vol SE.6, n° 1, January 1980 -

(TAU) R.C. TAUSWORTHE — Structured Programming and Software Engineering and Hard Real Time minicomputer systems, 1978, IEEE

(TAY) D. TAYLOR — A portable pseudo-code for Pascal like languages -Sigplan Notices V.19, January 1984 -

(TOU) J.M. TOULOTTE, J. THELLIEZ — Grafcet et logique industrielle programmée - Eyrolles 1980 -

J.M. TOULOTTE, J. THELLIEZ — Applications industrielles du Grafcet - Eyrolles 1982 -

(VER) C. VERCAUTER — Sur un ensemble d'outils d'aide à la spécification et à la conception des systèmes industriels - Thèse Docteur 3ème cycle Informatique Lille I, 5 juillet 1982 -

(WAL) C. WALTER — Control Software Specification and Design : an overview. - Computer IEEE 1984 - February -

(ZAV) P. ZAVE — An operational Approach to Requirements Specification for Embedded Systems - IEEE Transactions on Software Engineering, vol.SE.8, n° 3, May 1982 -

P. ZAVE — Executable Requirements for Embedded Systems. Dept. of Computer SCI - Univ of Maryland - College Park USA.

5TH International Conference on Software Engineering 295-304, 9-12 March 1981, San Diego, USA.