Université de Nancy I

U.E.R. Sciences Mathématiques

Centre de Recherche en Informatique de Nancy

CONTRIBUTION A LA STRUCTURATION ET A LA PROGRAMMATION DES APPLICATIONS DE CONTROLE DE PROCEDES INDUSTRIELS

THESE

soutenue publiquement le 20 Fevrier 1985

A L'UNIVERSITE DE NANCY I

pour l'obtention du grade de

DOCTEUR DE 3éme CYCLE EN INFORMATIQUE

par

EL FAZZIKI Abdelaziz

devant la Commission d'Examen

Président:	Jean-Claude	DERNIAME
Examinateurs:	Jacques	LONCHAMP
	Joachim	TANKOANO
	Jean-Pierre	THOMESSE

CONTRIBUTION A LA STRUCTURATION ET LA PROGRAMMATION DES APPLICATIONS DE CONTROLE DE PROCEDES INDUSTRIELS

Abdelaziz EL FAZZIKI

A mes Parents

Au bout de deux ans et demi de travail, je tiens à remercier, tous ceux qui m'ont aidé, de près ou de loin à réaliser cette thèse.

Mes remerciements vont tout d'abord à Monsieur le Professeur J.C. DERNIAME, Directeur du Centre de Recherche en Informatique de Nancy, pour m'avoir accueilli dans son équipe et accepté de diriger cette recherche. Je le remercie également pour toutes les suggestions qu'il a pu me faire, sans lesquelles ce travail n'aurait pu être mené à terme, et de présider ce Jury.

Monsieur le Professeur J.P. THOMESSE, pour toutes les suggestions et les remarques constructives qu'il m'a formulées et qui m'ont permis d'améliorer ce travail.

Monsieur J. LONCHAMP, Maître-assistant à l'Université de METZ, pour avoir bien voulu s'intéresser à ce travail, et pour toutes les suggestions et remarques positives qu'il m'a faites, qui ont permis son amélioration.

Monsieur J. TANKOANO, chercheur au CRIN, pour les discussions constructives que j'ai eues avec lui sur ce travail, pour ses remarques et critiques, qui m'ont permis de progresser dans mon travail.

Monsieur OUERGHI, pour sa collaboration dans la relecture de ce document.

Ce travail a pu être réalisé de cette manière grâce à la compétence de Madame MARCHAND Danielle, lors de la frappe.

Je remercie, Monsieur UL Vireack, pour la qualité du tirage de cette thèse.

Sommaire

- SOMMAIRE -

Pages

PARTIE I

Caractérisation des Applications de Contrôle de Procédés Industriels et de leur Environnement

PARTIE II

Méthode de Décomposition

PARTIE III

Programmation

-=-=-=-

OBJECTIFS

Nos objectifs peuvent être classés en deux catégories :

I - Proposer un ensemble de guides et d'outils permettant de faciliter la construction des structures logiques des systèmes de commande de procédés industriels en terme de modules communicants [KRA 80], [DER 82] et [ZAK 84].

Partant d'un cahier des charges contenant :

a) les spécifications fonctionnelles, qui précisent l'ensemble des fonctions à réaliser, explicitant à la fois le procédé à commander, le comportement souhaité du système de commande et l'utilisation de celui-ci.

b) les spécifications opératoires, qui concernent l'ensemble des caractéristiques de fonctionnement à respecter :

c) les spécifications technologiques, qui se rapportent au choix des constituants et composants, aux contraintes liées aux conditions d'exploitation.

Notre démarche consiste à appliquer un certain nombre de règles dans leur ordre de définition, en vue d'aboutir à une première structure fonctionnelle de base, en faisant abstraction de toutes les contraintes d'aspect dynamique. Puis, en considérant ces contraintes (points b et c), essayer de définir une structure fonctionnelle pouvant être décrite facilement en FLEXI [ZAK 84].

Notre proposition ne consiste en aucun cas en une méthode de structuration stricte et absolue, néanmoins nous considérons les règles que nous proposons comme **un** ensemble **de conseils qui peuvent s'intégrer dans une démarche descendante,** pour permettre de regrouper des entités de la structure fonctionnelle dans une étape ultérieure [LON 83], ou **dans une démarche ascendante** en vue de raffiner certaines entités.

II - La réalisation d'un langage de programmation des applications de contrôle/commande de procédés industriels, permettant une implantation facile des structures fonctionnelles issues de la proposition faite pour la structure des applications.

-=-=-=-

PREMIERE PARTIE

CARACTERISATION DES APPLICATIONS DE CONTROLE DE PROCEDES INDUSTRIELS ET LEUR ENVIRONNEMENT

CHAPITRE I

PROCEDES INDUSTRIELS

I.1 INTRODUCTION

I.2 CARACTERISTIQUES D'UN PROCEDE INDUSTRIEL

I.3 ELEMENTS POUR LA SPECIFICATION D'UN PROCEDE INDUSTRIEL

3.1 Variable

3.2 Action

3.3 Macroaction

3.4 Lois de commande.

-=-=-=-=-

I.1 INTRODUCTION

Les besoins d'automatisation de procédés industriels par l'informatique conduisent de plus en plus à l'utilisation de systèmes répartis, dans lesquels chaque site participe à un objectif global. Les outils informatiques développés doivent alors tenir compte de cet environnement et des contraintes introduites par la commande et le matériel utilisé.

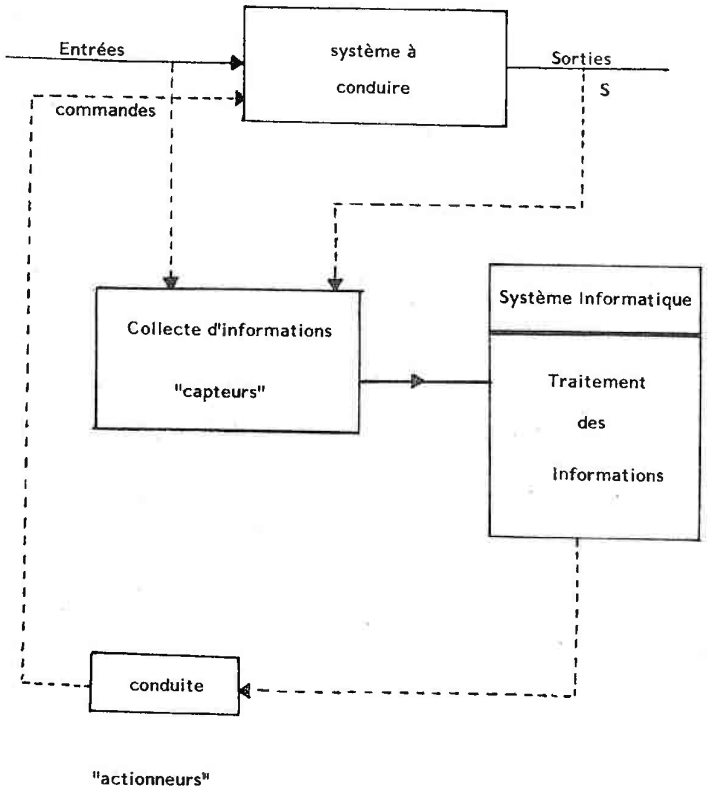
Par ailleurs, le procédé à commander est souvent composé de plusieurs activités simultanées, parfois même réparties géographiquement. Son contrôle nécessite l'exploitation d'un système qui assure :

- la mesure de certaines grandeurs de manière à observer l'évolution du procédé
- la prise en compte d'évènements survenant aléatoirement
- l'évaluation de décision à partir des informations précédentes
- la génération de grandeurs de commande, pour assurer la cohérence du comportement de l'ensemble.

Il en résulte un système de commande faisant généralement intervenir un ensemble d'activités parallèles qui doivent coopérer, afin d'assurer le bon fonctionnement de l'application. L'interaction importante avec le procédé fait apparaître pour ces activités, des contraintes de temps parfois très sévères, qu'il s'agit de respecter. Une contrainte de temps existe lorsque le temps séparant l'instant d'apparition d'une sollicitation, de l'instant d'achèvement de l'action conséquente doit rester inférieur à un temps maximum donné. Le non respect de ces contraintes peut conduire à l'application dans un état assimilable, à un état de panne matérielle.

Conduite d'une application temps réel

Schéma Global



I.2 CARACTERISTIQUES D'UN PROCEDE INDUSTRIEL

- un procédé industriel posséde sa propre dynamique que l'on peut qualifier d'évolution du procédé en fonction du temps.

- Il est souvent géographiquement réparti
- impose des contraintes de temps parfois sévères exprimées sous forme de **temps de réponse.**
- impose une grande sûreté dans le fonctionnement.

Ces caractéristiques permettent à leur tour de mettre en évidence un certain nombre de besoins, qu'il est nécessaire d'exprimer dans un langage de programmation des applications temps réel.

Ces besoins peuvent être classés en deux catégories :

- des besoins concernant l'aspect temps réel, qui doivent être exprimés au niveau du langage de programmation (Parallélisme, expression des contraintes de temps, urgence de certaines opérations, sûreté de fonctionnement,...)
- des besoins en structuration.

I.3 ELEMENTS POUR LA SPECIFICATION D'UN PROCEDE INDUSTRIEL

Un procédé industriel peut être décrit et caractérisé par les éléments suivants :

- variables
- actions
- macroactions
- lois de commande.

3.1 Variable

C'est une grandeur caractéristique du procédé, qui par une valeur donnée rend compte de l'état du procédé à un instant donné. Elle peut être mesurable, observable ou commandable [LAD 82]

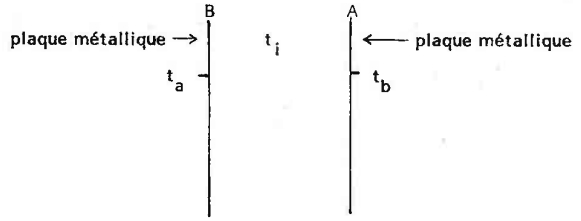
Exemples :

- variable mesurable :

"température d'un four"

- variable observable (calculable)

considérons l'exemple suivant :

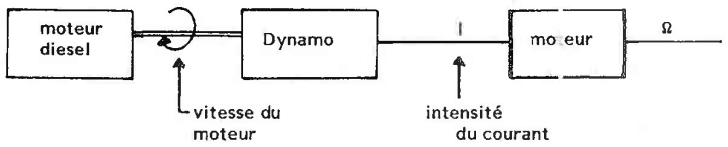


Il s'agit de déterminer la température t_i entre les deux plaques métalliques A et B. Cette température t_i varie en fonction des températures t_a et t_b.

t_a et t_b sont des variables mesurables, par contre t_i est une variable observable, on a : $t_i = f(t_a, t_b)$,

- variable commandable :

considérons l'exemple suivant d'un moteur diesel qui alimente une dynamo, qui à son tour alimente un moteur électrique :



La variable I est commandable par l'action sur le monteur, lorsqu'on augmente la vitesse du moteur, on augmente l'intensité du courant et inversement.

3.2 Action

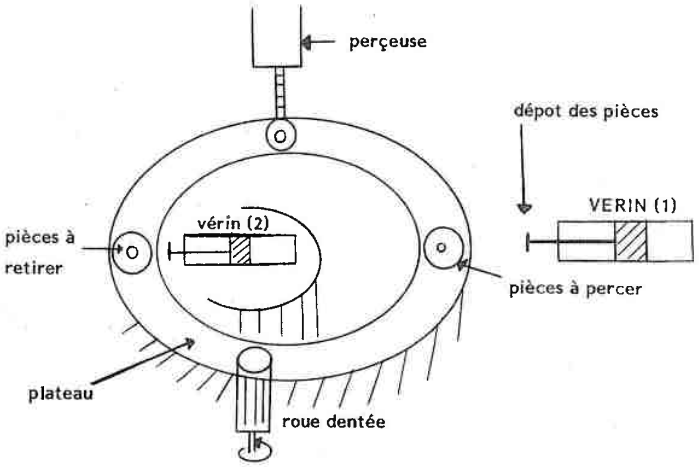
C'est une opération appliquée au procédé, soit directement, soit indirectement par l'intermédiaire d'un opérateur, et qui ne peut pas se décomposer en d'autres opérations à évolution parallèle.

3.3 Macroaction

C'est un ensemble d'actions nécessaires à la réalisation d'un objectif du procédé. L'absence de l'une des actions de cet ensemble remet en cause la réalisation de cet objectif.

exemple :

Considérons l'exemple d'un atelier de perçage de pièces :



Le procédé considéré vise un seul objectif : "percer des pièces", donc, il est composé d'une seule macroaction "percer" qui se compose des actions élémentaires suivantes :

- "déposer" une pièce
- "actionner le verin (1)" pour mettre la pièce sur le plateau
- amener la pièce sous la perceuse "actionner la roue dentée"
- percer la pièce "actionner la perçeuse"
- retirer la pièce "actionner le verin (2)".

Si l'une de ces actions est absente, le procédé tombe en panne.

3.4 Structure d'utilisation [BENMAIZA 84]

C'est un ensemble de liens entre les variables et les déclenchements des actions, exprimant l'évolution temporelle des actions et précisent :

- les conditions d'évolution des activités
- les traitements associés à chaque condition.

Son expression générale est du type :

"sur l'arrivée de chaque message, et si une condition est satisfaite, déclencher les actions correspondantes".

Par exemple, dans le cas de la pompe de KRAMER, qui sera décrit par la suite :

"sur l'arrivée du niveau bas de l'eau, arrêter la pompe".

-=-=-=-=-=-

CHAPITRE II

APPLICATIONS DE CONTROLE DE PROCEDES INDUSTRIELS

II.1. CARACTERISTIQUES D'UNE APPLICATION DE CONTROLE DE PROCEDES INDUSTRIELS.

II.2 DEFINITION DES ELEMENTS CONSTITUANT UNE APPLICATION.

- 2.1 Module
- 2.2 Unité Fonctionnelle
- 2.3 Entrée
- 2.4 Sortie

II.3 RELATION ENTRE LES ELEMENTS D'UNE APPLICATION.

- 3.1 Sorties consécutives
- 3.2 Sorties en exclusion mutuelle
- 3.3 Sorties disjointes
- 3.4 Ensemble mutuellement disjoint
- 3.5. Relation de protection entre modules.

II.4 CORRESPONDANCE ENTRE SYSTEME PHYSIQUE ET SYSTEME LOGIQUE.

II.1 CARACTERISTIQUES D'UNE APPLICATION

Une application informatique de contrôle de procédés industriels, est un ensemble de modules cablés, microprogrammés, permettant d'assurer la surveillance et la commande d'un phénomène physique. Elle se distingue des autres applications informatiques par le fait qu'elle est en interaction avec le monde réel. Elle doit permettre l'expression de tous les concepts caractérisant un procédé industriel cité au chapitre I.

II.2 DEFINITION DES ELEMENTS CONSTITUANT UNE APPLICATION

2.1 Module

C'est l'unité de structuration d'une application, il est constitué d'un ensemble d'actions élémentaires (tâches), d'entrées/sorties, et de variables. Il doit englober toutes les actions entre lesquelles existent des couplages serrés, et ne possède que des liens de messages avec l'environnement. Il correspond à une unité de répartition (un module ne peut pas être implanté sur deux sites [KRA 80], [DER 82] et [ZAK 84]).

2.2 Unité fonctionnelle

C'est un ensemble de modules assurant le contrôle de la réalisation d'un des objectifs du procédé. Elle correspond à une macroaction au niveau du système physique.

2.3 Entrée

C'est un message de l'environnement vers le système, elle correspond à la prise en compte d'une variable du procédé.

2.4 Sortie

C'est un message du système vers l'environnement, qui peut se traduire par une opération sur le procédé. Elle est caractérisée par un ensemble d'entrées en relation de causalité avec elle.

II.3 RELATIONS ENTRE LES ELEMENTS D'UNE APPLICATION

3.1 Sorties consécutives

Deux sorties sont consécutives si la fin d'exécution de l'une provoque le déclanchement de l'autre. C'est une relation de causalité entre deux sorties.

3.2 Sorties en exclusion mutuelle

Deux sorties sont en exclusion mutuelle, si à tout instant, au plus l'une d'entre elles est en cours d'exécution.

exemple :

soit une température T, dès que cette température dépasse un certain seuil TS, on exécute le traitement correspondant à la sortie S_1, sinon, celui correspondant à S_2.

Donc :

Si $T \leqslant TS$ alors exécuter S_2
sinon exécuter S_1

S_1 et S_2 sont en exclusion mutuelle, puisqu'elles ne pourront jamais être en cours d'exécution toutes les deux à la fois.

3.3 Sorties disjointes

Deux sorties sont disjointes, si leurs ensembles d'entrées sont disjoints, et si elles ne sont ni consécutives, ni en exclusion mutuelle.

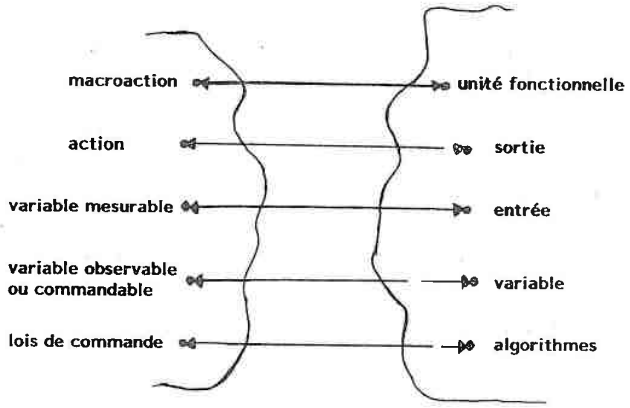
3.4 Ensemble mutuellement disjoint

Un ensemble de sorties est mutuellement disjoint, si pour tout couple de sorties de cet ensemble, ces deux sorties sont disjointes.

3.5 Relation de protection entre modules

Un module m_i est protégé d'un module m_j, si m_j ne fait appel à aucune action de m_i [RAJ 78] . Dans le cas des applications de contrôle de procédés, on considère que deux modules sont en relation de protection, s'il n'y a pas d'échange direct de messages entre eux.

II.4 CORRESPONDANCE ENTRE SYSTEME PHYSIQUE ET SYSTEME LOGIQUE



A toute entité caractérisant le système physique, on fait correspondre une entité caractérisant le système logique.

DEUXIEME PARTIE

METHODE DE DECOMPOSITION

CHAPITRE I

MODULARITE

I.1 INTRODUCTION

I.2 MODULARITE

2.1 Objectifs de la modularité

2.2 Critères d'une bonne modularisation

2.3 Cohésion des modules

2.4 Couplage des modules

2.5 Couplage et cohésion des modules dans le domaine des applications de contrôle de procédés

5.1 Cohésion des modules

5.2 Couplage des modules.

I.3 CONCLUSION.

-=-=-=-

I.1 INTRODUCTION

La maintenance du logiciel consomme la majeure partie de l'énergie et du budget de développement des projets informatiques. Des statistiques ont montré qu'entre 50 et 80 % du budget total sont absorbés par la maintenance. Ceci du fait des difficultés de transportabilité, lisibilité, maintenance... des logiciels. Ce qui a conduit à l'apparition d'une nouvelle discipline intitulée "génie logiciel", qui vise principalement la structuration des logiciels pour remédier à ces inconvénients. Et il est clair que ces mêmes problèmes sont rencontrés dans le domaine des applications de contrôle de procédés industriels. Donc, la structuration doit viser dans ce domaine les objectifs suivants :

- facilité de maintenance
- évolutivité
- transportabilité et réutilisation
- fiabilité

En effet, un procédé est rarement figé : les machines changent, certaines sont ajoutées. De plus, même lorsque le procédé ne change pas, de nouvelles fonctions peuvent être introduites, et enfin il faut tenir compte du développement du procédé qui se fait presque toujours progressivement. Lorsqu'on parle de structuration, on pense évidemment à une "méthode" de structuration. Plusieurs approches basées sur différents critères existent, mais il est difficile d'avoir une méthode précise de découpage. Mais on peut toujours se donner un guide de découpage et se fixer des objectifs.

I.2 MODULARITE

2.1 Objectifs de la modularité

Pour plus de détails sur ce concept, se référer à [DER 74] et [SOK 80]. La modularité est une technique fondamentale qui consiste à décomposer une application en un ensemble d'unités "modules". Elle permet d'atteindre plusieurs buts :

(1) réduire la complexité globale d'un problème : la complexité du travail qui consiste à découper un problème en sous problèmes, à concevoir les unités correspondantes et à décrire les relations entre ces unités est plus faible que la complexité du travail de résolution directe du problème, sans utiliser la technique de modularisation.

(2) Le découpage de l'application en unités permet aussi de réduire la complexité de vérification de la conformité d'un programme par rapport à ses spécifications.

(3) Obtenir un système plus facile à gérer, et pouvant être développé par plusieurs personnes.

2.2 Critères d'une bonne modularisation [SOK 80]

Une bonne modularisation doit vérifier les propriétés suivantes :

- chaque module a pu être développé indépendamment, avec un minimum d'interactions avec les autres modules du système.
- La compréhension et la modification d'un module demandent un minimum de connaissance des autres parties du système.
- Les erreurs dues à une "défaillance" peuvent être localisées facilement au niveau d'un module.

Afin de prendre en compte ces propriétés, il faut mettre l'accent sur l'indépendance des modules. Cette indépendance est réalisée à l'aide de deux méthodes d'optimisation :

(i) minimiser les relations entre les modules
(ii) maximiser les relations entre les éléments internes aux modules.

Ce qui revient donc à organiser les éléments de telle sorte que les éléments fortement liés soient dans un même module, on parle alors de cohésion maximale, et ceux qui n'ont pas de relations soient dans deux modules différents, on parle alors de minimisation du couplage.

2.3 Cohésion des modules (dans le cas général) [SOK 80]

La cohésion d'un module rend compte des relations qui existent entre les différents éléments du module. Le but principal d'une bonne décomposition modulaire consiste à regrouper les éléments de telle sorte que les éléments fortement liés soient dans le même module, et ceux qui n'ont pas de relation entre eux soient dans des modules différents. Du point de vue de l'évolutivité de l'application, il est essentiel de rechercher une forte cohésion. On peut distinguer plusieurs types de cohésions [SOK 80].

3.1 Cohésion coïncidentale [SOK 80]

Elle apparaît lorsqu'il n'y a pas de relations significatives entre les éléments du module. Un tel module est décrit par sa logique, et non par sa fonction. Elle est le résultat de l'une des trois situations suivantes :

- un programme déjà écrit est découpé en modules
- les modules sont créés pour éliminer les codes redondants entre d'autres modules

- on fractionne un programme à cause de la limitation de la taille de la mémoire disponible.

3.2 Cohésion logique [SOK 80]

Il y a cohésion logique, quand il existe une relation logique quelconque entre les éléments d'un module.

exemple :

un module qui sert à éditer les données, ce module réalise plusieurs fonctions (la mise à jour, l'adjonction, la suppression, etc...), et à chaque invocation, une seule fonction est exécutée.

3.3 Cohésion de type classique [SOK 80]

En plus de la cohésion logique, dans le cas de la cohésion classique, les modules sont exécutés séquentiellement dans le temps.

exemple :

le module d'initialisation, les éléments du module sont liés logiquement (l'initialisation représente la classe des fonctions logiques), de plus, ils sont liés dans le temps, parce que ces éléments sont exécutés séquentiellement au moment de l'appel.

3.4 Cohésion procédurale [SOK 80]

Les éléments du module réalisent plus d'une fonction, et sont liés entre eux pour résoudre un problème précis.

exemple :

régulariser la température d'une chaudière : le module qui a comme fonction < fermer la valve, lire la température de la chaudière, et l'enregistrer sur un journal > a une cohésion procédurale.

Les fonctions d'un module à cohésion procédurale sont liées à la solution du problème, elles tendent à avoir entre elles des liens très étroits.

3.5 Cohésion communicationnelle [SOK 80]

Le module possède une cohésion procédurale, mais en plus, toutes ses fonctions communiquent entre elles : soit elles font référence au même ensemble de données, soit elles se transmettent des données.

3.6 Cohésion fonctionnelle [SOK 80]

Cette cohésion est celle vers laquelle, il faut tendre. Un module à cohésion fonctionnelle, est un module dans lequel les éléments concourent à la réalisation d'une seule fonction.

exemple :

allouer un périphérique d'entrée/sortie.

3.7 Cohésion informationnelle [SOK 80]

Cette cohésion se trouve entre la cohésion communicationnelle et la cohésion fonctionnelle. Un module à cohésion informationnelle, réalise plusieurs fonctions qui correspondent à des points d'entrée du module et travaillent sur une structure de données unique. Ce module représente donc un rassemblement physique de plusieurs modules à cohésion fonctionnelle.

exemple :

Considérons deux modules à cohésion fonctionnelle :

- module M1 : insérer une entrée dans la table des symboles

- Module M2 : chercher une entrée dans la table des symboles.

Ces deux modules s'occupent d'une même structure de données (la table des symboles), ils peuvent être rassemblés dans un seul module à cohésion informationnelle, avec deux points d'entrée qui représentent les deux fonctions. Ainsi la structure de données est cachée à l'extérieur, et si on décide de changer la structure de la table, on ne changera qu'un seul module.

2.4 **Couplages des modules** (dans le cas général) [SOK 80]

Le couplage intermodules rend compte des différentes relations qui existent entre l'ensemble des modules d'une application. Une bonne conception modulaire est atteinte en diminuant les relations entre les modules. Donc, il est important de rechercher un faible couplage. On peut distinguer plusieurs types de couplage :

4.1 Couplage par le contenu

Des modules sont couplés par le contenu, si l'on fait référence directement à l'autre, sans passer par des noms symboliques externes ou des globaux.

4.2 Couplage par une zone commune

Les modules partagent des données globales dans un environnement commun.

4.3 Couplage par les externes

Les modules font référence au même symbole déclaré extérieurement.

4.4 Couplage par le contrôle

Un module donne des éléments de contrôle comme argument à l'autre module pour influencer son exécution.

4.5 Couplage par les paramètres et les données

Les modules communiquent entre eux par des paramètres ou échange de données.

2.5 **Cohésion et couplage des modules dans le domaine de contrôle des procédés industriels.**

On a vu dans le paragraphe précédent la cohésion et le couplage des modules dans le cadre des applications informatiques classiques. Pour les applications de contrôle, on ne s'intéresse qu'aux entrées-sorties des modules à partir desquelles on va définir la cohésion et le couplage des modules.

5.1 Cohésion des modules

La cohésion d'un module sera définie à partir des relations pouvant exister entre les entrées-sorties des modules.

*** relation entre entrées :**

la relation qui peut exister entre deux entrées d'un système est qu'elles peuvent déclencher toutes les deux une même sortie.

*** relations entre sorties :**

- Elles peuvent toutes les deux concourir à la réalisation d'un même objectif du procédé.
- Elles peuvent être en relation de succession.
- Elles peuvent être déclenchées par une même sortie.
- Elles peuvent être en relation d'exclusion mutuelle.

*** relation entre entrées et sorties :**

une entrée peut provoquer le déclenchement d'une sortie.

5.2 Couplage des modules

Dans les applications de commande de procédés, les modules ne peuvent être couplés que par des échanges de messages. Donc, pour avoir un couplage minimum entre les modules, il faut avoir un faible échange entre les modules.

I.3 CONCLUSION

Une conception modulaire est d'autant meilleure que la cohésion des modules est plus forte, et le couplage entre les modules est plus faible.

CHAPITRE II

METHODE DE DECOMPOSITION

II.1 APPROCHE

1.1 Contraintes et Concepts

- 1.1.1 Contraintes Economiques
- 1.1.2 Contraintes temporelles
- 1.1.3 Contraintes d'évolutivité
- 1.1.4 Parallèlisme de conception
- 1.1.5 Parallèlisme de l'environnement.

1.2 Démarche préconisée.

II.2 REGLES DE DECOMPOSITION

- 2.1 Règles de macrodécoupage
- 2.2 Règle isolant les sorties disjointes
- 2.3 Règles respectant le parallélisme de l'environnement et les contraintes de temps de réponse
- 2.4 Critères d'affectation des sorties restantes
- 2.5 Règles d'affectation des entrées restantes

II.3 DISCUSSION

3.1 Propriétés des règles

- 1.1 Règles de macrodécoupage
- 1.2 Règles de structuration des unités fonctionnelles
- 1.3 règles d'affectation des entrées sorties

1.4 Règles d'affectation des entrées restantes

1.5 Structuration des modules en tâches.

3.2 Toutes les entrées sorties sont exploitées par les règles

2.1 Toutes les sorties sont exploitées

2.2 Toutes les entrées sont exploitées.

II.4 RAFFINEMENT DE LA STRUCTURE FONCTIONNELLE.

II.1 APPROCHE

Une "bonne" décomposition d'une application en modules doit prendre en compte un certain nombre de contraintes, et doit être guidée par certains concepts.

1.1 Contraintes et concepts

1.1.1 Contraintes économiques

La décomposition d'une application temps réel est avant tout un problème économique, ceci dans le sens où ce qui importe le plus à un industriel est le coût de l'installation du système et la rentabilité du procédé. Donc, une méthode de conception d'une application de contrôle de procédé industriel doit prendre en compte d'éventuels changements dans l'instrumentation du procédé, éviter des duplications du matériel et permettre une grande sûreté de fonctionnement du procédé. Un procédé industriel réalisant un certain nombre d'objectifs ne doit pas être complètement pénalisé lors d'une panne au niveau d'une partie de son instrumentation intervenant dans la réalisation de l'un de ces objectifs.

1.1.2 Contraintes temporelles

Dans certaines applications de commande de procédés industriels, il y a des contraintes de temps de réponse qui sont imposées, donc, il faut profiter au maximum du parallélisme de l'environnement qui sera décrit par la suite.

1.1.3 Contraintes d'évolutivité

La structuration d'une application de commande

de procédé industriel doit prendre en compte d'éventuelles suppressions de l'une (ou plusieurs) composante (s) et extension du système (une suppression d'une composante du système doit avoir un impact minimum sur son ensemble). En plus, pour le cas d'une éventuelle extension, ceci nous paraît très important dans le sens où souvent les procédés industriels sont automatisés secteur par secteur (ou service par service).

1.1.4 Parallélisme de conception

On trouve dans RAYNAL [RAY 80], LONCHAMP [LON 83], et ZAKARI [ZAK 84], une distinction importante entre parallélisme de conception et parallélisme de l'environnement. Dans le premier, le concepteur de toute application peut décider de la décrire de façon parallèle indépendamment de son environnement. Son utilisation peut être considérée comme un outil de conception de programmes, et sert essentiellement à maîtriser leur architecture. Il ne peut pas être envisagé comme concept de base pour la structuration des applications de commande.

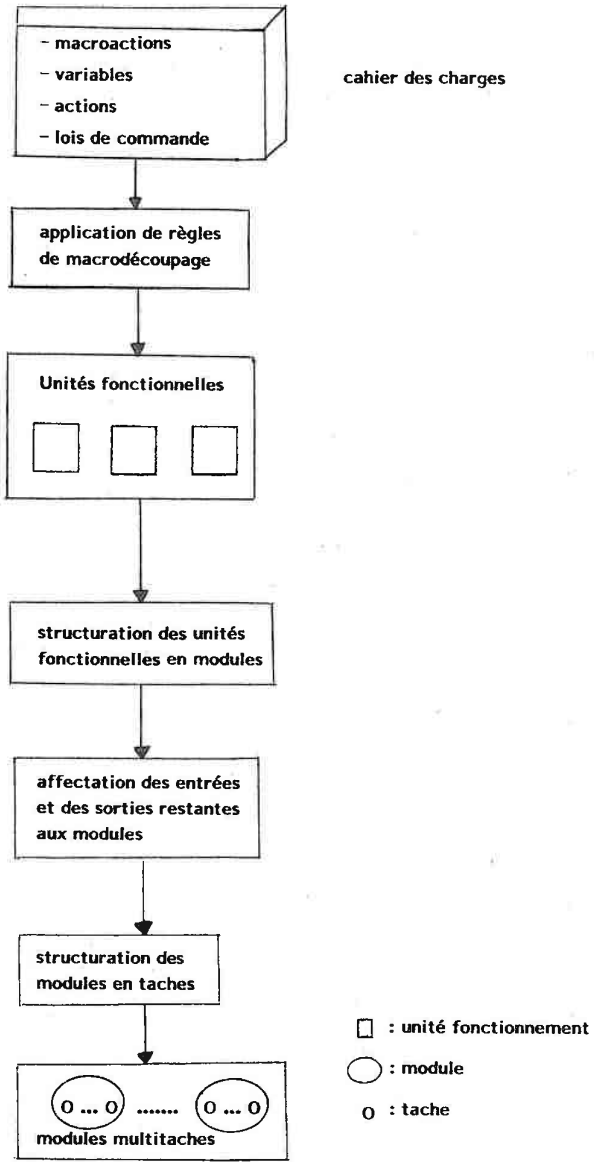
1.1.5 Parallélisme de l'environnement

Dans l'environnement d'une application de commande de procédé industriel, il y a un certain nombre de grandeurs et d'opérations pouvant évoluer en parallèle, ceci se traduit au niveau du système par une simultanеïté de ses entrées-sorties : ce sont par exemple, les cas où lors de la prise en compte d'une valeur d'une variable, il y a déclenchement de plusieurs sorties simultanément, ou lors de la prise en compte par le système de deux valeurs de variables distinctes, il y a déclenchement de plusieurs sorties au même instant.

1.2 Démarche préconisée

Notre démarche consiste à développer une méthode de décomposition essayant de respecter les contraintes citées auparavant, et basées sur les concepts cités dans les chapitres précédents. Ceci en élaborant un ensemble de règles permettant d'obtenir un ensemble de modules multitâches ayant une forte cohésion, un faible couplage et un parallélisme maximal à l'intérieur de chacun d'entre eux. Ces règles sont établies à partir d'un cahier des charges décrivant la spécification fonctionnelle du procédé et mettant en évidence l'ensemble de ses caractéristiques variables, actions, lois de commande, macroactions. La méthode proposée consiste à appliquer un ensemble de règles dans leur ordre de numérotation qui sont décrites par la suite, et vise les objectifs suivants :

- cohésion forte des modules
- couplage faible des modules
- protection des modules
- parallélisme à l'intérieur des modules respectant le parallélisme de l'environnement.



II.2 REGLES DE DECOMPOSITION

2.1 Règles de macrodécoupage

(i) Définir les différentes unités fonctionnelles de base du système.

(ii) Identifier les différentes sorties de chacune des unités fonctionnelles

(iii) Isoler les sorties intervenant dans plusieurs unités fonctionnelles, et leur faire correspondre une unité fonctionnelle supplémentaire, à laquelle seront attribuées toutes les entrées communes aux unités fonctionnelles de base.

2.2 Règles de structuration des unités fonctionnelles en modules

(iv) Pour chacune des unités fonctionnelles obtenues, définir un ensemble mutuellement disjoint, et dans le cas, où il y en a plusieurs, prendre celui regroupant le plus d'entrées. Puis pour chacune des sorties de cet ensemble faire correspondre un module auquel est associé l'ensemble des entrées de cette sortie (sauf les entrées attribuées déjà par la règle (iii)).

2.3. Règles d'affectation des sorties restantes

(v) Toute sortie dont toutes les entrées sont affectées à un module, est affectée à ce module.

(vi) Une sortie est affectée au module ayant le plus d'entrées en commun avec elle.

(vii) Une sortie est affectée à un module si elle ne créée pas d'échange entre deux modules, qui n'en possèdent pas auparavant.

(viii) Deux sorties consécutives sont associées à un même module.

(ix) Deux sorties en exclusion mutuelle sont associées à un même module.

2.4 Règles d'affectation des entrées restantes

(x) Une entrée intervenant dans le déclenchement d'un ensemble de sorties, qui sont toutes affectées à un seul module, est affectée à ce module.

(xi) Une entrée est affectée à un module de façon à ne pas créer d'échange entre deux modules qui n'en possèdent pas auparavant.

(xii) Une entrée n'intervenant dans le déclenchement d'aucune des sorties d'un module n'est pas affectée à celui-ci.

2.5 Règles de structuration d'un module en tâches

(xiii) Deux sorties pouvant être déclenchées en parallèle sont associées à deux tâches distinctes.

(xiv) Deux sorties consécutives sont associées à une même tâche.

(xv) Deux sorties en exclusion mutuelle sont associées à une même tâche.

(xvi) Une entrée intervenant dans le déclenchement d'une seule sortie du module, est attribuée à la tâche assurant le déclenchement de cette sortie.

(xvii) Deux entrées pouvant arriver simultanément, sont attribués à deux tâches distinctes.

(xviii) Deux entrées ne pouvant pas arriver simultanément, sont attribuées à une même tâche.

(xiv) Une entrée est attribuée à l'une des tâches du module.

Remarques :

- Dans le cas où une sortie a le même nombre d'entrées avec un certain nombre de modules, et ne vérifie pas les relations r_3 et r_4, elle est affectée à l'un de ces modules.

- Dans le cas où une entrée intervient dans le déclenchement de plusieurs sorties qui sont attribuées à un certain nombre de modules, et n'est pas exploitée par la règle (xi), elle est attribuée à l'un de ces modules.

II.3 DISCUSSION

3.1 Propriétés des règles

1.1 Règles de macrodécoupage

Ces règles permettent une première structuration fonctionnelle de l'application, de façon à avoir une indépendance dans la commande des différentes macroactions du procédé considéré. Elles permettent aussi une localisation facile des impacts d'éventuelles pannes du système, ce qui est très important au niveau de la maintenance des applications et de leur évolutivité.

Si on considère l'exemple du capteur solaire qui sera décrit par la suite, il est caractérisé par deux macroactions :

chauffer et **refroidir**

Chacune de ces macroactions est un ensemble d'actions élémentaires. Soit S_1 l'ensemble des sorties correspondant à la macroaction **chauffer**, et S_2 celui correspondant à la macroaction **refroidir**, et S_{12} l'ensemble intersection de ces deux ensembles

$$S_{12} = S_1 \cap S_2$$

Aux deux macroactions on associe les unités fonctionnelles U_1, U_2 et U_{12} telles que :

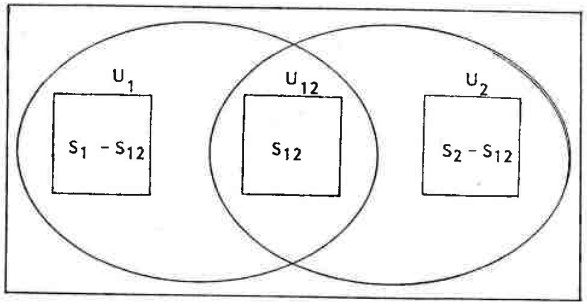
U_1 est caractérisée par l'ensemble des sorties $S_1 - S_2$

U_2 est caractérisée par $S_2 - S_1$

("-" : différence de deux ensembles).

U_3 est caractérisée par S_{12}.

Donc ces trois unités fonctionnelles assurent la commande des deux macroactions **chauffer** et **refroidir**



Pour chauffer, on n'a besoin que des unités U_1 et U_{12}.

Pour refroidir, on n'a besoin que des unités U_2 et U_{12}.

Donc, quant on a une panne au niveau d'une machine supportant un module de l'unité fonctionnelle propre au chauffage, elle n'aurait pas d'impact sur celles supportant les modules propres au refroidissement, et inversement. Ce qui permet une plus grande sûreté de fonctionnement. En plus, une modification au niveau d'un module

d'une unité fonctionnelle n'aurait pas d'impact sur les modules de l'autre unité fonctionnelle, ce qui est important au point de vue protection des modules.

1.2 Règle de structuration des unités fonctionnelles

La règle (iv) permet d'attribuer :

1) Deux sorties disjointes dans deux modules différents, ce qui va bien dans le sens d'une forte cohésion.

2) - des entrées n'ayant pas de relation entre elles sont dans deux modules différents

- des entrées ayant une relation entre elles sont dans un même module. Ce qui va bien dans le sens d'une forte cohésion des modules.

Considérons, l'ensemble S des sorties du système, et S_M un ensemble mutuellement disjoint

$$S_M = \{ S_i \} \quad i = \overline{1,k}$$

chacune des sorties S_i de S_M est caractérisée par un ensemble d'entrées

$$E_{s_i} = \{e_j\} \qquad j = \overline{1, p} \text{ tel que :}$$

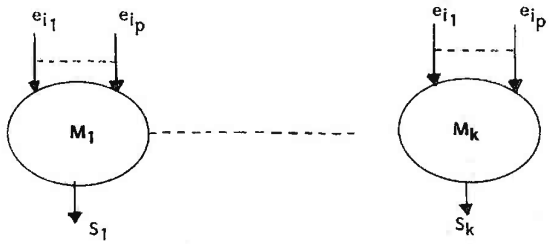
Pour tout couple $(S_i, S_j) \in S_M^2$ $i \neq j$ on a alors

$$E_{s_i} \cap E_{s_j} = \emptyset$$

et à chacune de ces sorties s_i est associé un module M_i, donc à l'ensemble mutuellement disjoint S_M correspond un ensemble de module $\{ M_i \} i = \overline{1,k}$: tel que chacun de ses modules est caractérisé par

S_i : la sortie

E_{s_i} : ensemble d'entrées



Supposons qu'une entrée e $\in E_{S_i}$ est affectée à un module M_j tel que $i \neq j$, ceci va se traduire par une création d'échange entre les modules M_i et M_j, ce qui est contradictoire avec les objectifs recherchés par la méthode.

Donc cette règle nous permet d'obtenir un ensemble de modules ayant chacun une sortie et un certain nombre d'entrées, sans aucun échange de messages jusqu'à maintenant.

1.3. Règles d'affectation des sorties restantes aux modules

3.1 La règle (v) permet d'affecter une sortie à un module de façon à avoir un minimum d'échange entre les modules (couplage minimum) :

Considérons une sortie S_i dont toutes les entrées sont affectées à un module M_i, si elle est affectée à un module M_j tel que $i \neq j$, donc il y aurait autant d'échanges en plus, entre M_i et M_j qu'il y ait d'entrées de la sortie S_i, ce qui est contradictoire avec le but visé (couplage minimum). Donc la règle (v) permet bien d'affecter une sortie de façon à avoir un échange minimum entre les modules.

3.2 La règle (vi) permet aussi d'avoir un couplage minimum. En effet, considérons une sortie s_i dont l'ensemble d'entrées est E_{s_i}, et deux modules M_i, M_j tels que $M_i \neq M_j$ dont les ensembles d'entrées sont respectivement
E_{M_i} et E_{M_j} et Card $(E_{M_i} \cap E_{S_i}) \neq$ Card $(E_{M_j} \cap E_{S_i})$. Supposons que :

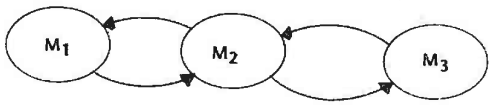
Card $(E_{M_i} \cap E_{S_i})$ > Card $(E_{M_j} \cap E_{S_i})$, et la sortie S_i est affectée au module M_j, ceci va se traduire au niveau du système par
Card $(E_{M_i} \cap E_{S_i})$ - Card $(E_{M_j} - E_{S_i})$ échanges en plus. Donc, la règle (vi) permet bien d'affecter une sortie à un module de façon à avoir un minimum d'échange.

3.3 La règle (vii) permet d'affecter une sortie de façon à préserver la protection des modules.

Considérons une sortie S_i et trois modules M_1, M_2 et M_3. E_{M_1}, E_{M_2}, E_{M_3} sont les ensembles d'entrées des trois modules considérés, et E_{S_i} celui de la sortie S_i, tels que

Card $(E_{M_1} \cap E_{S_i})$ = Card $(E_{M_2} \cap E_{S_i})$ = Card $(E_{M_3} \cap E_{S_i})$.

M_1 possède des échanges avec M_2
M_2 possède des échanges avec M_3
M_1 et M_3 sont en relation de protection (n'échangent pas de messages)



si la sortie S_i est affectée à M_1 ou M_3, ceci va se traduire par une création d'échanges entre M_1 et M_3, donc les modules M_1 et M_3 ne seront plus en relation de protection entre M_1 et M_3, il faut affecter la sortie S_i au module M_2, puisque le nombre d'échanges sera toujours le même qu'elle soit affectée à M_1, M_2 ou M_3.

3.4 La règle (viii) permet d'avoir un couplage minimum entre les modules. Car en effet, si deux sorties S_1 et S_2 sont consécutives et sont affectées à deux modules différents M_1 et M_2, ceci va se traduire par un échange en plus entre M_1 et M_2, ce qui est contradictoire avec l'objectif recherché (couplage minimum).

3.5 La règle (ix) permet de regrouper deux sorties qui sont en exclusion mutuelle, cela veut dire que sous une condition, l'une des sorties est exécutée, et sous une autre l'autre sortie est exécutée. Les deux conditions ne sont jamais remplies à la fois. Donc, on peut considérer les deux sorties comme une seule à laquelle sont associées deux traitements qui ne sont jamais exécutés en parallèle.

1.4 Règles d'affectation des entrées restantes

Ces règles permettent de répartir les entrées restantes de façon à avoir un couplage minimum des modules.

4.1 La règle (x) permet d'affecter une entrée à un module de façon à avoir un minimum d'échange au niveau du système.

En effet, considérons un module M ayant comme ensemble de sorties l'ensemble S_M, et une entrée e, dont l'ensemble des sorties qu'elle déclenche est S_e, tels que $S_e \subset S_M$.

Si l'entrée M est affectée à un module M' tel que M' ≠ M, ceci va se traduire par un échange en plus entre M et M', ce qui est contradictoire avec l'objectif qu'on s'est fixé de couplage minimum. Donc, la règle (x) nous permet bien d'aller dans ce sens.

4.2 La règle (xi) permet d'affecter une entrée de façon à avoir un couplage minimum.

En effet, considérons un module M dont l'ensemble des sorties est S_M, et une entrée e dont l'ensemble des sorties est S_e tel que $S_M \cap S_e = \emptyset$.

Si l'entrée e est affectée au module M, ceci va se traduire par un échange en plus au niveau du système.

4.3 La règle (xii) permet d'affecter une entrée de façon à préserver la protection des modules.

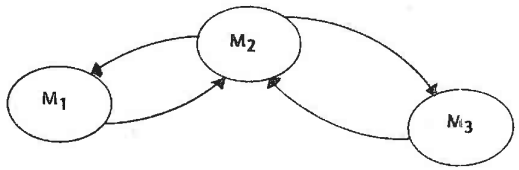
Considérons une entrée e dont l'ensemble des sorties qu'elle déclenche est S_e, et trois modules M_1, M_2, M_3 dont les ensembles de sorties sont respectivement S_{M_1}, S_{M_2}, S_{M_3} et tels que :

$S_e \cap S_{M_1} \neq \emptyset$; $S_e \cap S_{M_2} \neq \emptyset$; $S_e \cap S_{M_3} \neq \emptyset$ et

M_1 échange des messages avec M_2

M_2 échange des messages avec M_3

M_1 ne possède pas d'échanges avec M_3



Si l'entrée e est affectée au module M_1 ou M_3, ceci conduit à une création d'échange entre M_1 et M_3, ce qui n'est pas souhaitable au niveau de la protection des modules, puis le nombre d'échanges au niveau de tout le système est toujours le même, que l'entrée e soit affectée au module M_1, M_2 ou M_3. Donc autant l'affecter à M_2 et préserver la relation de protection entre M_1 et M_3.

1.5 Structuration des modules en tâches

Ces règles permettent la structuration des modules en tâches pouvant s'exécuter en parallèle tout en respectant le parallélisme de l'environnement. Ce qui nous semble très important pour la conduite des procédés où souvent les contraintes de temps de réponse sont imposées.

3.2 Toutes les sorties sont exploitées par les règles

2.1 Toutes les sorties sont exploitées par les règles

Notation

On considère le couple (S, R) où $S = \{ S_i \}\ i = \overline{1, p}$ ensemble des sorties du système, et $R = \{r_j\}\ j = 1,4$ ensemble des types de relations entre ces sorties

$S \times S \xrightarrow{r_1} \mathbb{B}$ $\mathbb{B}$ étant l'ensemble des booléens

$S \times S \xrightarrow{r_2} \mathbb{B}$

$S \times S \xrightarrow{r_3} \mathbb{B}$

$S \times S \xrightarrow{r_4} \mathbb{B}$

$\forall (S_1, S_2) \in S^2, \exists r_i \in R$ t.q $s_1\ r_i\ s_2 \quad i = 1,4$

où $s_1\ r_1\ s_2$ s_1 et s_2 sont disjointes

ou $s_1\ r_2\ s_2$ s_1 et s_2 partagent des entrées

ou $s_1\ r_3\ s_2$ s_1 et s_2 sont consécutives

ou $s_1\ r_4\ s_2$ s_1 et s_2 sont en exclusion mutuelle

La règle permet d'affecter un certain nombre de sorties, celles de l'ensemble mutuellement disjoint choisi. Soit S_M cet ensemble et soit s_i une sortie de l'ensemble S, on a alors, soit $s_i \in S_M$,

soit $s_i \in \begin{cases} S_M \\ S \end{cases}$

1) si $s_i \in S_M$ elle est exploitée par la règle (iv)

2) si $s_i \in \begin{cases} S_M \\ S \end{cases}$ alors $\exists r_i \in R\ (i \neq 1)$ et $S_j \in S_M$ telles que :

$$S_i\ r_i\ s_j$$

si $r_i = r_2$ la sortie est exploitée par la règle (vi)

si $r_i = r_3$ la sortie est exploitée par la règle (viii)

si $r_i = r_4$ la sortie est exploitée par la règle (ix).

Donc, quelque soit la sortie de l'ensemble S, elle est exploitée par l'une des règles.

2.2 Toutes les entrées sont exploitées

Considérons l'ensemble E de toutes les entrées du système, et E_M celui de l'ensemble mutuellement disjoint choisi.

Soit $e \in E$, on a alors :

soit $e \in E_M$ ou $e \in \begin{cases} E_M \\ E \end{cases}$

1) si $e \in E_M$ elle est exploitée par la règle (iv).

2) si $e \in \begin{cases} E_M \\ E \end{cases}$: soit s_e l'ensemble des sorties qu'elle déclenche $(S_e \neq \emptyset)$

et, dans ce cas deux situations peuvent se présenter :

- Il existe un module m_i dont l'ensemble des sorties est S_{m_i}, tel que $S_e \subset S_{m_i}$ et dans ce cas l'entrée e est exploitée par la règle ( ).

- Il existe un ensemble de modules $\mathcal{M}$ contenant au moins chacun une sortie qui peut être déclenchée par l'entrée e. Soit $\mathcal{M}_p$ l'ensemble des modules en relation de protection tel que $\mathcal{M}_p \subset \mathcal{M}$.

Si $\mathcal{M}_p \neq \emptyset$, alors, l'entrée est affectée à l'un des modules de l'ensemble $\begin{cases} \mathcal{M}_p \\ \mathcal{M} \end{cases}$, et dans ce cas, elle est exploitée par la règle (vii).

Si $\mathcal{M}_p = \emptyset$, l'entrée est affectée à l'un des modules de l'ensemble $\mathcal{M}$. Donc toutes les entrées-sorties sont exploitées par les règles proposées.

II.4 RAFFINEMENT DE LA STRUCTURE FONCTIONNELLE

La structure fonctionnelle obtenue, après application des règles de décomposition, doit tout d'abord être acceptée par la structure d'exécution (matériel destiné à supporter l'application) : il faut qu'il

y ait une correspondance entre la structure fonctionnelle, et la structure d'exécution. Dans le cas où il n'y a pas de correspondance, nous proposons des critères pour essayer d'adapter la structure fonctionnelle obtenue à la structure d'exécution disponible. Par contre, on ne fait aucune proposition concernant l'adaptation de la structure d'exécution à la structure fonctionnelle, pour celà, le lecteur pourra se referer à [CAL 82].

Deux cas peuvent se présenter :

1°) la structure fonctionnelle, comporte un module qu'on ne peut implanter sur une machine, et dans ce cas on propose de le découper selon les règles suivantes :

a) soit le module contient des sorties disjointes et on applique les règles IV, V,.....,

b) soit le module ne contient pas de sorties disjointes, et dans ce cas, on applique les règles suivantes :

- prendre les deux sorties du module, qui ont le moins d'entrées en commun, et leur faire associer un module chacun
- appliquer les règles V, VI, ... ,

2°) Il faut regrouper des modules pour les faire supporter par un seul site, et dans ce cas, les modules les plus couplés sont sur un même site. Les modules peuvent être couplés par les différentes entités :

- nombre de liaisons
- taille des informations échangées
- fréquence des échanges
- coût en temps des échanges.

-=-=-=-=-

CHAPITRE III

APPLICATION DE LA METHODE SUR DES EXEMPLES

-=-=-=-=-

III.1 EXEMPLE DU CAPTEUR SOLAIRE

1.1 Spécification fonctionnelle externe

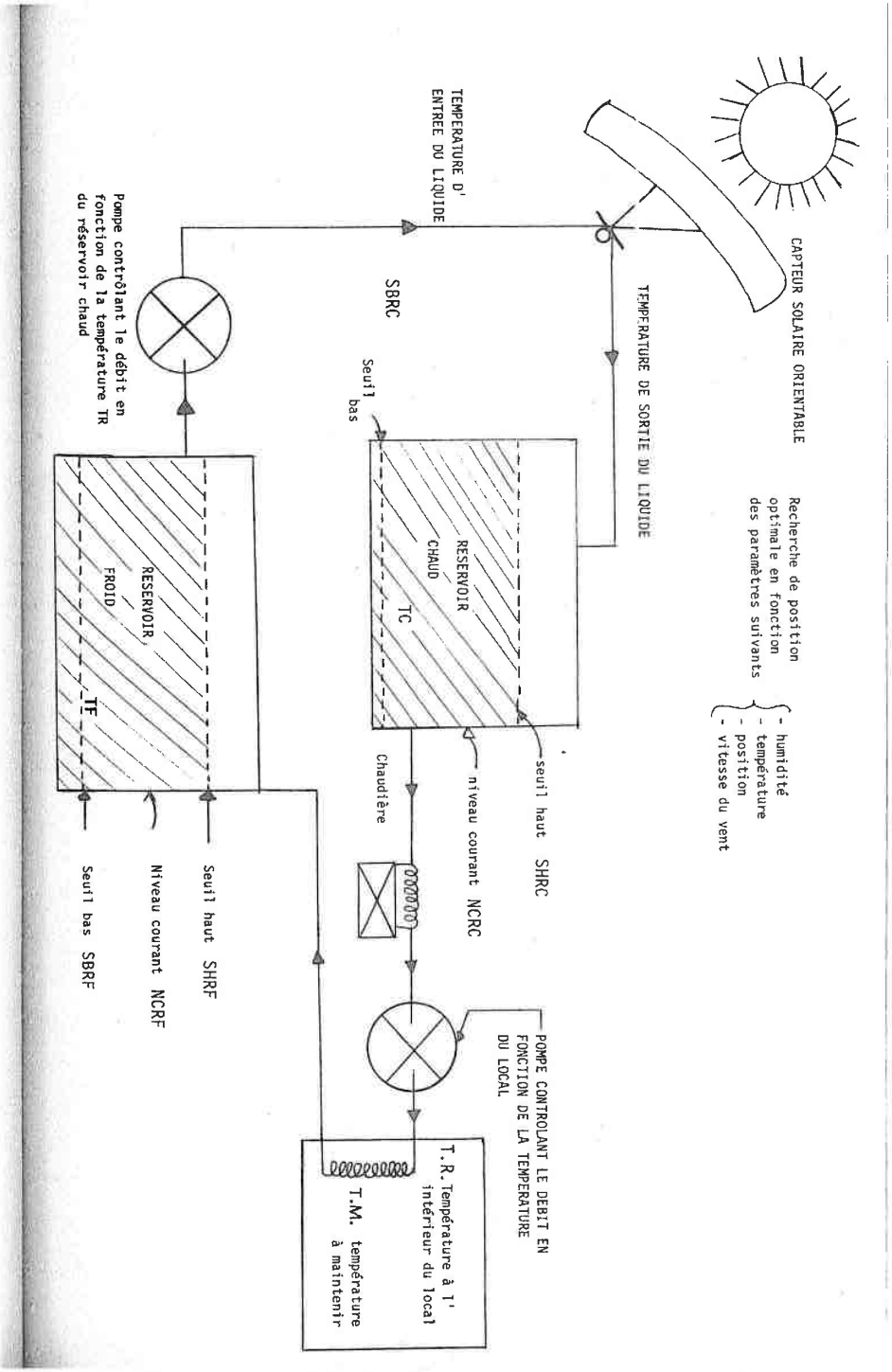
Il s'agit de décrire le fonctionnement d'un système de chauffage et de refroidissement de locaux à l'aide d'un capteur solaire orientable, présenté sous forme simplifiée dans la figure (1). Le système reçoit la commande d'initialisation, et doit réguler le fonctionnement des différentes composantes de l'environnement, en tenant compte des paramètres externes au système. Le capteur doit être orienté de façon à avoir un position optimale (permettant d'emmagasiner un maximum d'énergie). Cette orientation dépend des facteurs suivants :

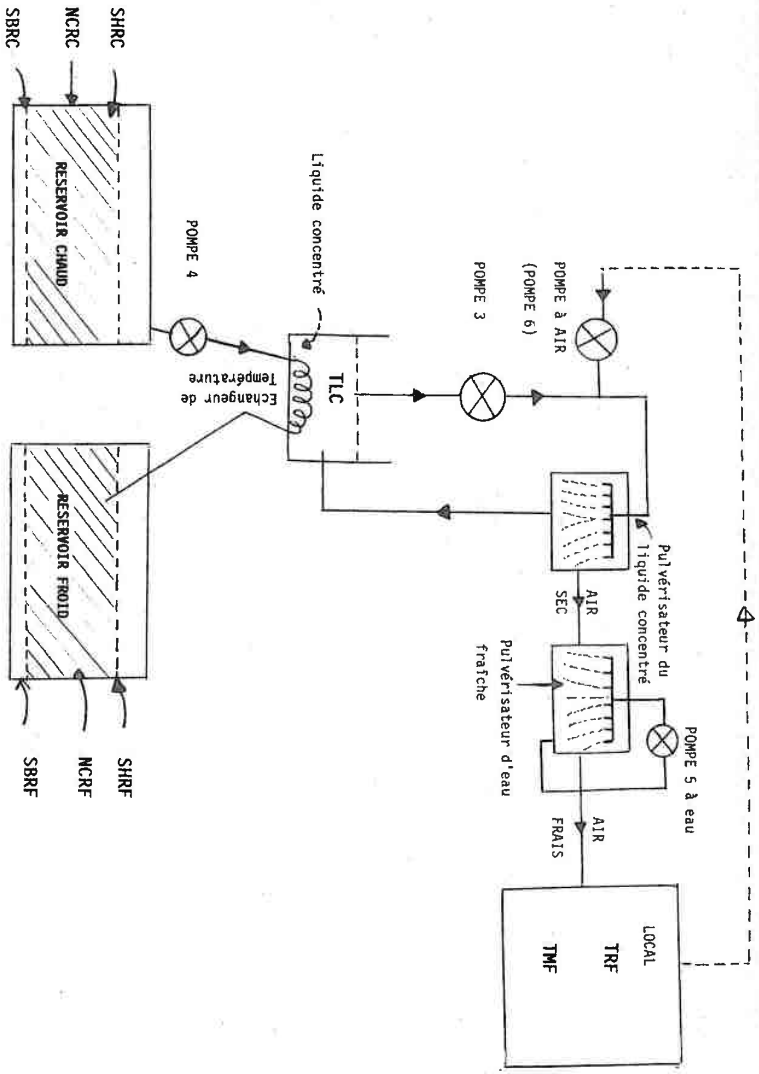
- jour de l'année
- situation géographique
- humidité
- position du soleil dans la journée
- température ambiante externe
- vitesse du vent.

Le chauffage du local est réalisé par le fait qu'un liquide provenant d'un réservoir froid (eau par exemple) traverse le capteur, et s'écoule dans un réservoir chaud avec une température de sortie TC. Suivant la température à maintenir (TM) dans le local, on agit sur les pompes (pompe 1 et pompe 2), pour réguler le fonctionnement. Si on veut augmenter la température dans le réservoir chaud, on diminue le débit du liquide à l'aide de la pompe 1, et si on veut que cette température diminue on augmente le débit. Pour réaliser la température à maintenir dans le local, on agit sur la pompe 2 de façon analogue que pour la pompe 1, en tenant compte de la température régnant dans le local (TR). Lorsque la température atteint un seuil, et qu'on ne peut plus avoir la température recherchée (TM) à l'aide du capteur, une chaudière est déclenchée tout en régulant les pompes. Le refroidissement du local, se fait à l'aide d'un liquide concentré qu'il faut chauffer régulièrement pour maintenir sa température à un

certain seuil, et de l'air qu'il faut soit déshydrater à l'aide d'un pulvérisateur du liquide concentré pour obtenir de l'air chaud, soit hydrater de nouveau à l'aide d'un autre pulvérisateur d'eau, pour obtenir de l'air frais destiné au local. Pour maintenir la température voulue dans le local, on agit sur les différentes pompes du procédé selon le même principe que pour le chauffage (voir description des actions du système).

Remarque : On suppose que le système est capable à tout instant, de calculer la température maximale qu'il peut fournir.





1.2 Variables du système

TEX : température ambiante externe
PS : position du soleil
H : humidité
VV : vitesse du vent
TF : température du liquide dans le réservoir froid
TC : température du liquide dans le réservoir chaud
TR : température régnant dans le local à chauffer
TMAX : température maximale que peut fournir le système
TS : température seuil, suivant laquelle on déclenche on non la chaudière
- si la température à maintenir dans le local dépasse la température maximale que peut fournir le système de la valeur TS, il faut arrêter le capteur et déclencher la chaudière.
- sinon il faut arrêter la chaudière, et mettre en route le capteur.

SHRF : seuil haut du réservoir froid
SBRF : seuil bas du réservoir froid
NCRF : niveau courant dans le réservoir froid
SHRC : seuil haut du réservoir chaud
SBRC : seuil bas du réservoir chaud
NCRC : niveau courant dans le réservoir chaud
TLC : température du liquide concentré
TM : température à maintenir dans le local à chauffer
TRF : température du local à refroidir
TMF : température à maintenir.

1.3 Actions du système

orientation du capteur : OC
action pompe 1 : AP1 (augmenter ou diminuer le débit)
action pompe 2 : AP2 (augmenter ou diminuer le débit)

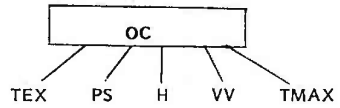
action pompe 3 : AP3 (augmenter ou diminuer le débit)
action pompe 4 : AP4 (" " " ")
action pompe 5 : AP5 (" " " ")
action pompe 6 : AP6 (" " " ")
action chaudière : ACH (déclencher ou arrêter la chaudière)

{ affichage utilisateur dans le local à chauffer : AULC

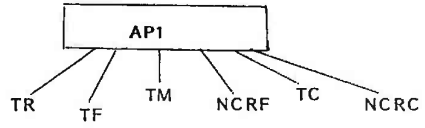
{ affichage utilisateur dans le local à refroidir : AULF

1.4 Définition des différentes entrées pour chacune des sorties correspondantes aux actions du système

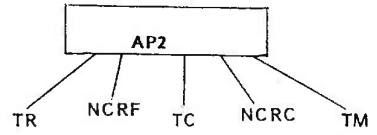
*** Orientation capteur**



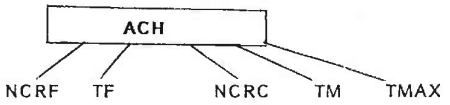
*** Action Pompe 1**



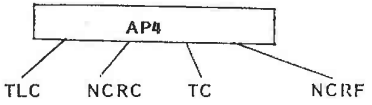
*** Action Pompe 2**



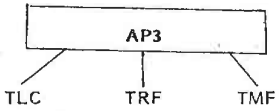
*** Action Chaudière**



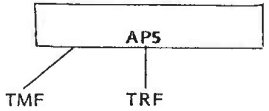
*** Action Pompe 4**



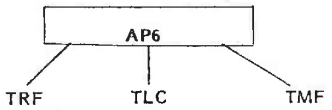
*** Action Pompe 3**



*** Action Pompe 5**

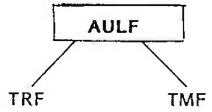


*** Action Pompe 6**



* **Affichage utilisateur dans le local à chauffer**

* **Affichage utilisateur dans le local à refroidir**



1.5 Règle d'activation des lois de commande

* **Orientation du capteur**

On suppose que le système de commande dispose des équations décrivant les relations entre les différents paramètres du capteur. Ces équations sont supposées être établies par des thermitiens.

* **Commande des pompes 1, 2 et de la chaudière**

		TR > TM		TR < TM	
		POMPE 1	POMPE 2	POMPE 1	POMPE 2
TM - TMAX > TS	MARCHE Chaudière	+	-	-	+
TM - TMAX < TS	ARRET Chaudière	+	-	-	+

+ : Augmenter le débit - : diminuer le débit

	POMPE 1	POMPE 2	CHAUDIERE
NCRF niveau courant du réservoir froid	- **Arrêt,** si NCRF atteint la valeur seuil SBRF - **Marche,** dans le cas où la pompe aurait été arrêtée	- **Arrêt,** si NCRF atteint la valeur seuil SHRF - **Marche,** dans le cas où la pompe aurait été arrêtée	- **marche** éventuellement - **Arrêt,** si NCRF atteint la valeur seuil SHRF
NCRC niveau courant du réservoir chaud	- **Arrêt,** si NCRC atteint la valeur seuil SHRC - **Marche,** dans le cas où la pompe aurait été arrêtée	- **Arrêt,** si NCRC atteint la valeur seuil SBRC - **Marche,** dans le cas où la pompe aurait été arrêtée	- **Arrêt,** si NCRC atteint la valeur seuil SBRC - **Marche** éventuellement

*** Commande des pompes 3, 4, 5, 6**

	Pompe 3	Pompe 4	Pompe 5	Pompe 6
TRF < TMF	-	-	-	+ / -
TRF > TMF	+	+	+	+ / -
NCRC niveau courant du réservoir chaud		- **Arrêt**, si NCRC atteint la valeur seuil SBRC - **Marche**, éventuellement		
NCRF niveau courant du réservoir froid		- **Arrêt**, si NCRF atteint la valeur seuil SHRF - **marche** éventuellement		

1.6 Application des règles de macrodécoupage

6.1 Application de la règle (i)

Dans cet exemple, on distingue deux macroactions :

- **chauffer**
- **refroidir**

D'après la règle (i), on leur fait correspondre deux unités fonctionnelles, soient U'_1 et U'_2 ces unités (unités fonctionnelles de base).

6.2 Application de la règle (ii)

- Sorties de l'unité fonctionnelle U'_1 (chauffer).

On sait que chaque unité fonctionnelle est caractérisée par un ensemble de sorties, soit S_1 cet ensemble, on a :

$$S_1 = \{OC, AP1, AP2, ACH, AULC\}$$

- Sorties de l'unité fonctionnelle U'_2 (refroidir).

Soit S_2 l'ensemble de ces sorties, on a alors :

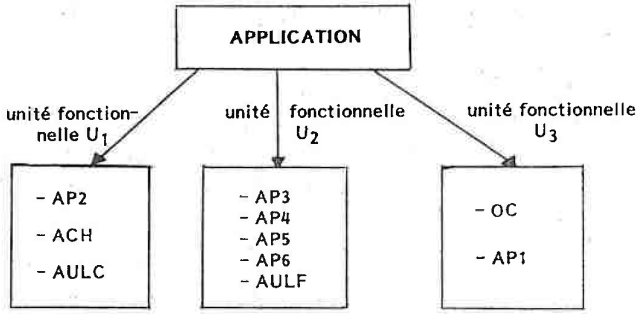
$$S_2 = \{OC, AP1, AP3, AP4, AP5, AP6, AULF\}.$$

6.3 Application de la règle (iii)

Soit S_{12} l'ensemble intersection des deux ensembles de sorties S_1 et S_2, on a :

$$S_{12} = \{OC, AP1\}$$

Donc, aux deux macroactions **chauffer** et **refroidir** seront associées trois unités fonctionnelles U_1, U_2, et U caractérisées par les ensembles de sorties $S_1 - S_{12}$; $S_2 - S_{12}$ et S_{12}



Les unités fonctionnelles U_1 et U_3, constituent le système de contrôle de la macroaction "chauffer".

Les unités fonctionnelles U_2 et U_3, constituent le système de contrôle de la macroaction "refroidir".

L'absence de U_1 n'a pas d'impact sur U_2 et réciproquement.

1.7 Structuration des unités fonctionnelles en modules

7.1 Unité fonctionnelle U_1

1.1 Application de la règle (iv)

Cette unité fonctionnelle est caractérisée par l'ensemble des sorties :

$$S_1 - S_{12} = \{AP2, ACH, AULC\}$$

*** Graphe donnant les sorties n'ayant pas d'entrées en commun**

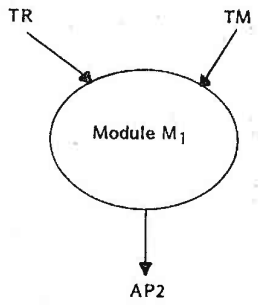
sorties / entrées	AP2	AULC	ACH
TR	▨	▨	▨
TM	▨	▨	▨
TC	▨		
TMAX			▨
N RC	▨		▨
N RF	▨		

D'après le graphe ci-dessus les trois sorties AP2, AULC, et ACH ne sont pas disjointes puisqu'elles ont deux entrées en commun TR et TM, donc l'unité fonctionnelle U_1 admet trois ensembles mutuellement disjoints qui sont réduits à des singletons : { AP2 }, {AULC }, et { ACH } d'après la règle (iv), il faut choisir celui regroupant le plus d'entrées, ce qui correspond à l'ensemble {AP2 }. Donc, à l'unité fonctionnelle U_1 on fait correspondre un seul module, car son ensemble mutuellement disjoint ne contient qu'une seule sortie.

Soit M_1 ce module, donc d'après la règle (iv), la sortie AP2 et toutes ses entrées (sauf celles attribuées à l'unité fonctionnelle U_3) seront attribuées au module M_1. A ce stade là le module M_1 aura la structure suivante :

d'après la règle (iii), l'unité fonctionnelle U_1 possède l'ensemble d'entrées suivant : {TR, TM, TMAX }, c'est l'ensemble des entrées qui est propre à l'unité fonctionnelle "chauffer" (on considère les variables qui ne sont pas réellement des entrées, mais qui impliquent des échanges entre les modules, avec la répartition des entrées

structure du module après application de la règle (iv)



On ne représente pas la variable TMAX, car ce n'est pas une entrée, mais elle sera attribuée à ce module.

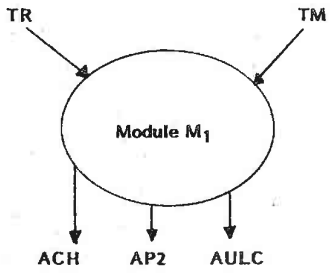
1.2 Affectation des sorties restantes aux modules

Après application de la règle (iv), on a obtenu un seul module et il reste deux sorties à affecter à ce module. Puisque l'unité fonctionnelle ne contient qu'un seul module, ces deux sorties lui sont attribuées.

1.3 Affectation des entrées restantes

Il ne reste aucune entrée à affecter, elles sont toutes attribuées par la règle (iii) et (iv).

Structure du module avec toutes ses entrées et ses sorties

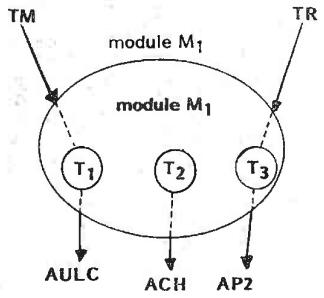


1.4 Structuration du module en tâches

Le module M_1, sera structuré d'après la règle (xiii) en trois tâches, assurant successivement le contrôle de AP2, AULC, et ACH.

D'après la règle (xvii), les entrées TR et TM sont attribuées à deux tâches distinctes, et d'après la règle (xiv), on les attribuera chacune à une tâche assurant le contrôle d'une sortie qui est en relation de causalité avec cette entrée.

Structure finale du module



les T_i sont des tâches

7.2 Unité fonctionnelle U_2

2.1 Application de la règle (iv)

L'unité fonctionnelle U_2 est caractérisée par l'ensemble de sorties

$S_2 - S_{12}$ = {AP3, AP4, AP5, AP6, AULF }, et l'ensemble d'entrées { TLC, TMF, TRF } , il n'y a que ces entrées qui seront affectées aux modules de U_2.

Soit SU_2 l'ensemble mutuellement disjoint de U_2 :

*** Graphe donnant les sorties n'ayant pas d'entrées en commun**

Sorties / Entrées	AP3	AP4	AP5	AP6	AULF
TLC	▨	▨		▨	
NCRC		▨			
NCRF		▨			
TC		▨			
TMF	▨		▨	▨	▨
TRF	▨		▨	▨	▨

D'après ce graphe, les ensembles de sorties n'ayant pas d'entrées en commun sont :{ AP4, AP5 } et { AP4, AULF }.

*** Sorties consécutives**

Les sorties AP3 et AP5 sont consécutives, elles sont en relation de causalité : chaque fois qu'on actionne la pompe 3, on actionne la pompe 5.

*** sortie en exclusion mutuelle**

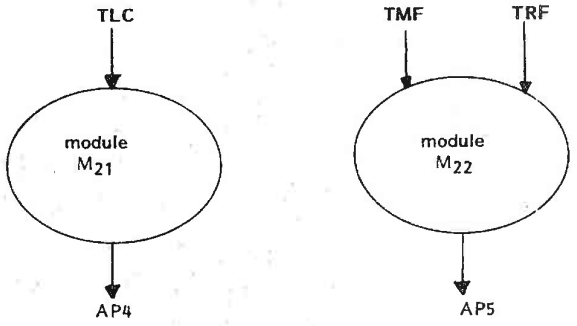
Il n'y a pas de sortie en exclusion mutuelle.

Donc, les ensembles {AP4, AP5} et { AP4, AULF } sont bien des ensembles mutuellement disjoints, car leurs sorties ne sont ni consécutive, ni en exclusion mutuelle. D'après la règle (iv), on choisit celui regroupant le plus d'entrées, soit {AP4, AP5 }.

On a donc SU_2 = { AP4, AP5 }.

Donc, à l'unité fonctionnelle U_2, on fait correspondre deux modules M_{21} et M_{22}, car SU_2 contient deux sorties.

Après application de cette règle les modules M_{21} et M_{22} auront la structure suivante :



Les autres entrées des sorties AP4 et AP5 sont attribuées aux modules de l'unité fonctionnelle U_3 par la règle (iii).

2.2 Affectation des sorties restantes

Il reste à affecter les sorties AP3, AP6 et AULF

- Affectation de AP3

D'après la règle (vi), AP3 sera affectée au module M_{22}, car elle a deux entrées en commun avec lui, et elle n'en a qu'une seule avec M_{21}. En plus, elle est consécutive avec AP5.

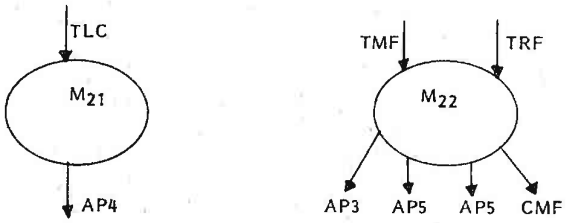
- Affectation de AP6

D'après la règle (vi), AP6 est attribuée au module M_{22}, elle a deux entrées en commun avec lui, et n'en possède qu'une seule avec M_{21}.

- Affectation de AULF

D'après la règle (vi), elle est affectée au module M_{22}, car elle a deux entrées en commun avec lui, et aucune avec M_{21}.

Donc, les modules M_{21} et M_{22} auront les sorties suivantes :



2.3 Affectation des entrées restantes

Toutes les entrées sont affectées par les règles (iv) et (iii).

2.4 Structuration des modules en tâches

Le module M_{21} ne contiendra qu'une seule tâche, car il ne contient qu'une seule sortie.

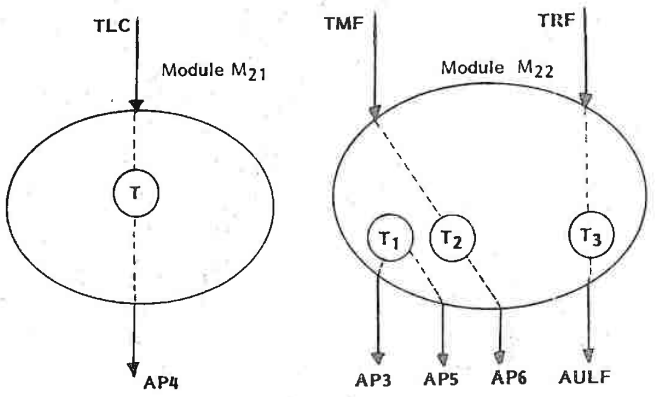
- D'après la règle (xiii), AP3, AP6 et AULF doivent être dans des tâches différentes.

- D'après la règle (xiv), AP3 et AP5 sont associées à la même tâche.

Donc, le module M_{22} sera structuré en trois tâches.

- L'entrée TLC, sera attribuée à la tâche que contiendra, le module M_{21}, d'après la règle (xvi).

- Les entrées TMF et TRF doivent être attribuées à deux tâches distinctes d'après la règle (xii). On les attribuera aux tâches assurant le contrôle des sorties AULF et AP6 d'après la règle (xix).



Structures des modules obtenues par application de ces règles

7.3 Unité fonctionnelle U_3

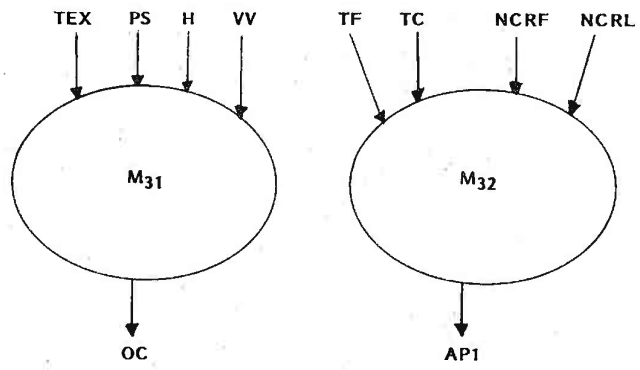
3.1 Application de la règle (iv)

L'unité fonctionnelle U_3 est caractérisée par l'ensemble de sorties S_{12} = {OC, AP1} et l'ensemble d'entrées :

{ TEX, PS, H, VV, NCRF, NCRC, TC, TF }.

Les sorties OC et AP1 n'ont pas d'entrées en commun et ne sont ni consécutive, ni en relation d'exclusion mutuelle. Donc, l'ensemble {OC, AP1} est mutuellement disjoint.

Donc à l'unité fonctionnelle U_3, on fait correspondre deux modules M_{31} et M_{32}. A chacun de ces modules est associé une sortie de l'ensemble {OC, AP1} **avec toutes ses entrées** appartenant à l'ensemble des entrées de l'unité fonctionnelle U_3.



3.2 Affectations des sorties restantes

Toutes les sorties sont affectées.

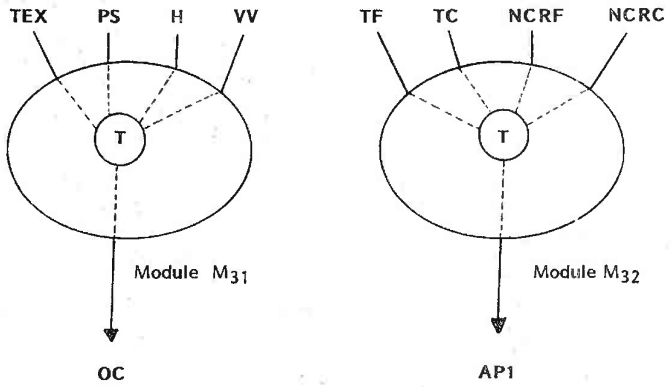
3.3 Affectation des entrées restantes

Il ne reste aucune entrée à affecter.

3.4 Structuration des modules en tâches

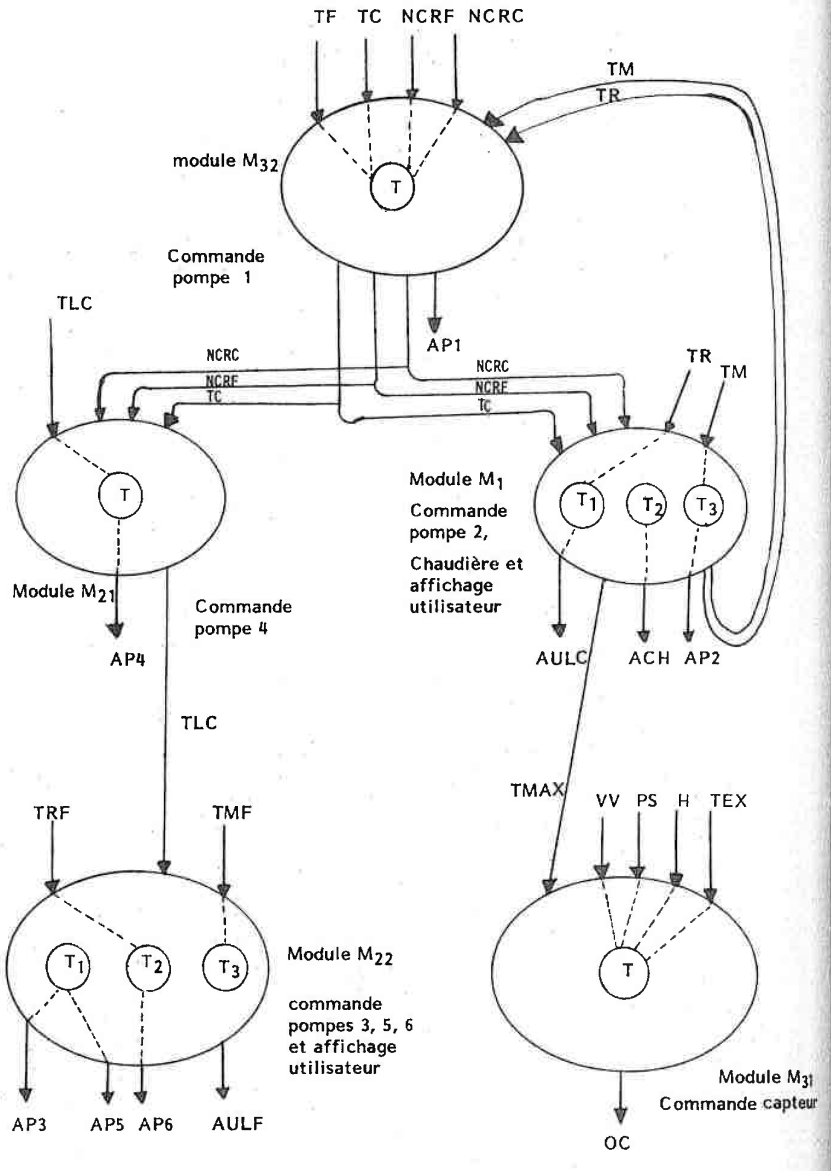
- Le module M_{31}, sera structuré en une seule tâche, car il ne contient qu'une seule sortie, et toutes ses entrées seront attribuées à cette tâche.

- De la même façon, le module M_{32} sera structuré en une seule tâche.



1.8 Echanges intermodules

Les échanges intermodules sont déduits des entrées qui correspondent aux variables mesurables, et les variables observables (ou calculables).



Les modules M_{31}, M_{32} et M_1 constituent la partie contrôle du chauffage. Les modules M_{31}, M_{32}, M_{21} et M_{22} constituent la partie contrôle du refroidissement.

Les modules qui sont propres à la commande du chauffage n'ont pas de relation directe avec ceux qui sont propres à la commande du refroidissement.

Une modification au niveau du module M_1 n'a pas d'impact sur les modules M_{21} et M_{22} et réciproquement.

Les modules ne regroupent que des entrées-sorties qui sont fortement liées entre elles.

La répartition des entrées-sorties est faite en ayant le minimum d'échange entre les cinq modules. En effet, si on change cette répartition par rapport aux cinq modules, on obtient plus d'échanges.

III.2 EXEMPLE DE KRAMER

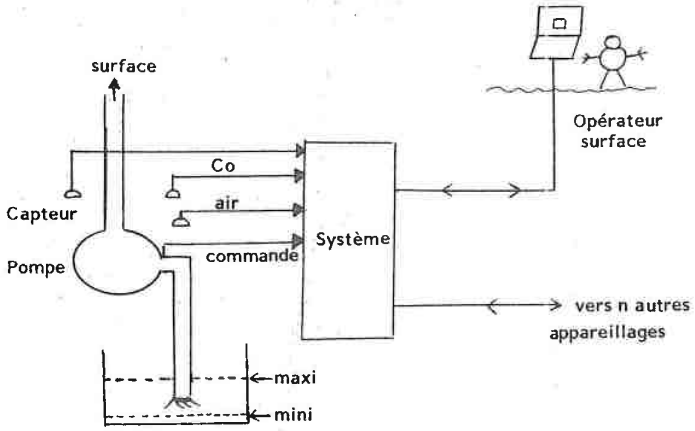
1.1 Spécification fonctionnelle externe

Cet exemple est tiré de [KRA 80]. Il s'agit d'un système de commande d'une pompe de drainage située dans une galerie souterraine d'une mine de charbon et de contrôle de son environnement atmosphérique. La pompe, après avoir été mise en marche par une commande depuis l'opérateur en surface, fonctionne automatiquement :

- démarrage lorsque l'eau est mesurée au dessus du niveau maximum
- arrêt lorsque l'eau est mesurée en dessous du niveau minimum.

Pour des raisons de sécurité, la pompe ne doit pas démarrer ou continuer à fonctionner lorsque le pourcentage de méthane dans

l'air dépasse un seuil donné. Des capteurs (de méthane, de monoxyde de carbone, d'aération) renseignent à tout instant aussi bien le système que n autres appareillages, et la surface sur l'état de l'atmosphère. Des alarmes, en cas de dépassement des seuils sont émises vers la surface (dans tous les cas), et vers la pompe pour le méthane seulement afin d'arrêter la pompe si celle- ci est en marche.



1.2 Variables du système

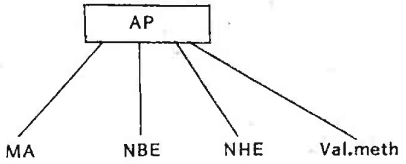
MA	:	consigne de marche ou d'arrêt de la pompe
REP	:	requête sur l'état de la pompe
NHE	:	niveau maximum de l'eau
RG	:	requête sur les états des gaz (Co, air, méthane)
Val.meth	:	teneur en méthane
Val.Co	:	teneur en monoxyde de carbone
Val.air	:	état de l'aération
NNE	:	niveau normal de l'eau
NBE	:	niveau minimum de l'eau

1.3 Actions du système

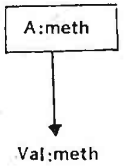
AP : action sur la pompe (démarrage et arrêt)
A.meth : alarme méthane
ACo : alarme oxyde de carbone
A.air : alarme atmosphère
EP : réponse sur l'état de la pompe
EG : réponse sur l'état des gaz (Co, méthane, air).

1.4 Définition des différentes entrées pour chacune des sorties correspondantes aux actions

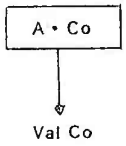
*** Action pompe**



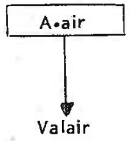
*** Alarme methane**



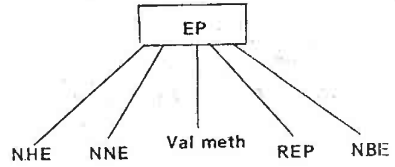
*** Alarme oxyde de carbone**



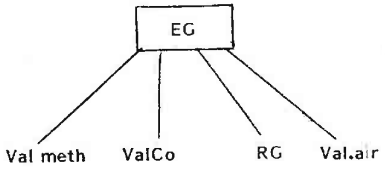
*** Alarme air**



*** Etat pompe**



*** Etat gaz**



1.5 Application des règles de macrodécoupage

Il y a une seule macroaction dans le procédé considéré,

on lui associe donc une unité fonctionnelle, soit U cette unité.

1.6 Structuration de l'unité fonctionnelle en modules

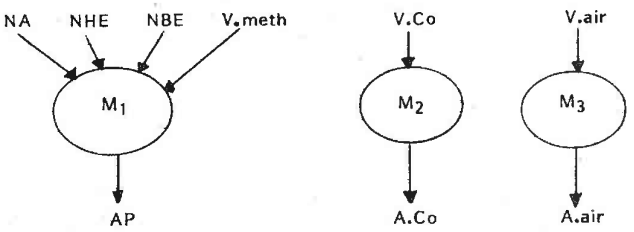
6.1 Application de la règle (iv)

* Il n'y a pas de sorties consécutives, ni en relation d'exclusion mutuelle (on ne fait aucune supposition sur la lecture des gaz).
* Graphe donnant les sorties n'ayant pas d'entrées en commun.

Sorties / Entrées	AP	A.meth	A.Co	Aair	EP	EG
MA	▨					
REP					▨	
NHE	▨				▨	
NBE	▨				▨	
RG						▨
V.meth	▨	▨				▨
V.air				▨		▨
V.Co			▨			▨
NNE					▨	

D'après le graphe ci-dessus, on constate qu'il y a deux ensembles mutuellement disjoints { Ameth, Aco, Aair } et { AP, ACo, Aair } , d'après la règle (iv), on choisit celui regroupant le plus d'entrées, qui est le second. Donc à l'unité fonctionnelle U, on va faire correspondre

trois modules M_1, M_2, et M_3 dont chacun contiendra une sortie de l'ensemble mutuellement disjoint choisi avec toutes ses entrées :



6.2 Affectation des sorties restantes

Il reste Ameth, EP et EG à affecter.

- Affectation de Ameth

D'après la règle (v), cette sortie est attribuée au module M_1, car toutes ses entrées lui sont attribuées.

- Affectation de EP

D'après la règle (vi), elle est affectée au module M_1, car elle a trois entrées en commun avec lui, et une avec les autres.

- Affectation de EG

Cette sortie a une entrée en commun avec chacun des modules. Donc, elle est affectée à M_1 d'après la règle (vii), car sinon on va créer un échange entre M_2 et M_3 qui sont en relation de protection :

6.3 Affectation des entrées restantes

Il reste les entrées REP, RG, et NNE à attribuer.

- Attribution de REP

D'après la règle (x), elle est attribuée au module M_1, car elle n'intervient que pour le déclenchement de la sortie EP qui est attribuée à M_1.

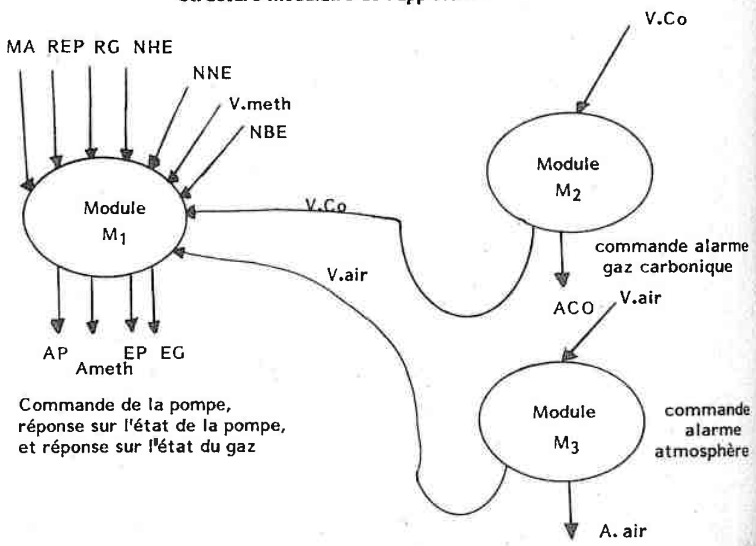
- Attribution de RG

D'après la règle (x), elle est attribuée au module M_1, car elle n'intervient que pour le déclenchement de EG qui est attribué à M_1.

- Attribution de NNE

D'après la règle (x), elle est attribuée au module M_1, car elle n'intervient que pour le déclenchement de EP qui est attribué à M_1.

Structure modulaire de l'application



- Tout ce qui est nécessaire au contrôle de la pompe est regroupé dans le module M_1.

- Les sorties A.Co et A.air sont attribuées à deux modules distincts, parce qu'elles n'ont pas de relation entre elles. Donc, les modules ne regroupent que des éléments qui sont fortement liés.

- La répartition des entrées-sorties entre les trois modules est faite en ayant un échange minimum entre les modules M_1, M_2 et M_3.

1.7 Structuration des modules en tâches

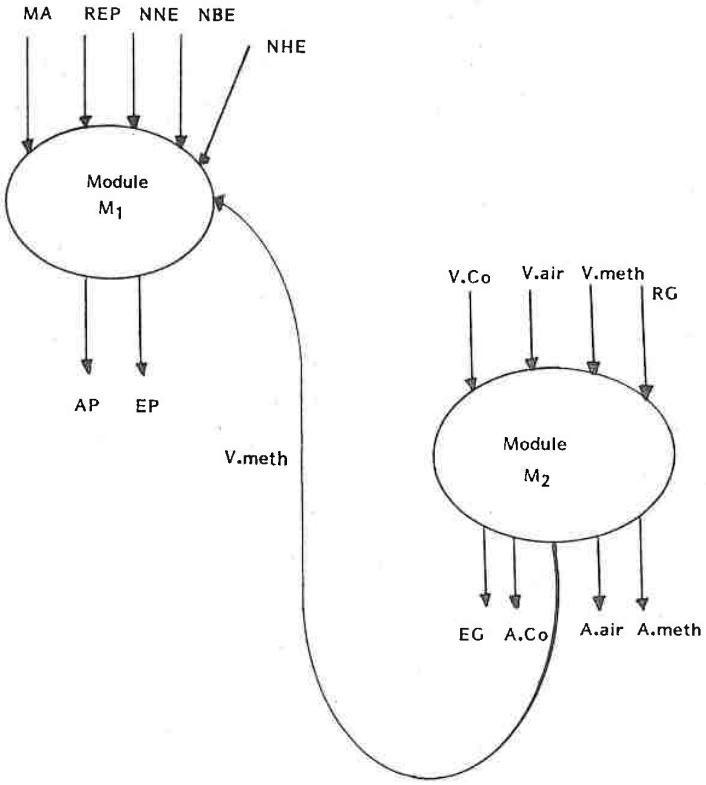
Les modules M_2 et M_3 seront mono-tâches, car ils ne contiennent qu'une seule sortie chacun.

D'après la règle (xiii), le module M_1 sera structuré en quatre tâches, chacune assurant le contrôle de l'une de ses sorties.

Remarque :

En supposant que les lectures des gaz sont séquentielles comme dans [KRA 80], on obtient la structure modulaire suivante : (fig. 1)

D'après le graphe précédent, on aura un ensemble mutuellement disjoint { AP, A.Co }, car les sorties A.meth, A.Co, et A.air sont cette fois en exclusion mutuelle.



- Le module M_1 sera structuré en deux tâches d'après la règle (xiii).

- Le module M_2 sera structuré en deux tâches d'après les règles (xiii) et (xv).

-=-=-=-=-

CHAPITRE IV

ETUDE DE QUELQUES PROPOSITIONS

-=-=-=-=-

I PROPOSITION DES MACHINES ABSTRAITES [GALINIER 78]

I.1 Introduction

Dans le cadre du projet SURF, [GALINIER, CHERBONNEAU, DELACROIX] ont proposé une méthodologie de conception de programmation de logiciels fiables à partir des travaux de B. LISKOV et de E.W. DIJKSTRA basés sur les machines abstraites.

I.2 Proposition des machines abstraites

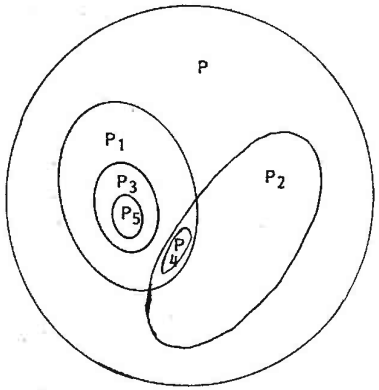
L'idée principale de la méthode est de laisser au développeur le choix de déterminer les aspects caractéristiques de son problème pour élaborer une solution qu'il juge la mieux adaptée, et de proposer un modèle mettant en évidence le degré de complexité et d'indépendance de chaque sous problème.

L'approche préconisée par la méthode est descendante par niveaux d'abstraction successives ; [à chaque étape, il s'agit d'obtenir la description logique satisfaisante du nouveau problème à envisager, puis de développer la solution de celui-ci en termes des abstractions introduites]. Il convient de rechercher d'abord des abstractions intéressantes pour l'expression du problème et formuler ensuite la solution en faisant usage des abstractions utiles. La résolution d'un problème consiste alors à définir plusieurs machines abstraites ayant chacune ses ressources abstraites propres et son jeu d'instructions abstraites, puis programmer le traitement de chaque sous problème sur la machine présentant la structure et le fonctionnement adaptés.

Les points essentiels de la méthode sont :

- un problème P doit être examiné comme une composition de problèmes typiques élémentaires P_1, P_2, ... P_k.

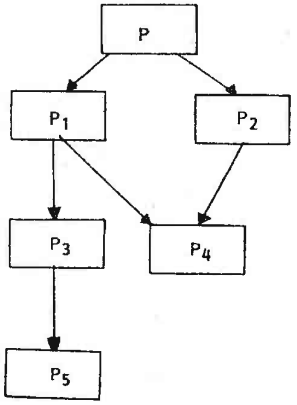
Exemple :



- P_3 et P_4 sont deux sous problèmes distincts
- La résolution de P_3 dépend de la résolution de P_5
- La résolution de P_5 doit être indépendante de P_3
- Toute solution de P_4 doit convenir à P_1 ou P_2.

- Une machine abstraite (MA) de ressource R et d'opérations I_1, I_2, I_3 supporte une abstraction qui s'exprime par le type R et les fonctions I, et peut exécuter un ou plusieurs programmes correspondant à un problème déterminé.

La solution au problème P définie comme résultante des solutions partielles des sous problèmes P_1, P_2,..., P_5 peut être développée par exécution d'un programme sur cinq machines réparties sur les trois niveaux d'abstraction :



niveau d'abstraction 1

niveau d'abstraction 2

niveau d'abstraction 3

La structure hiérarchique du système est déduite de celle du problème, et sa preuve est réduite aux preuves de chacune des unités indépendantes.

I.3 Conception par machines abstraites

- Une machine abstraite est un ensemble de ressources et d'instructions, concevoir une application par MA consiste à développer une série de machines, chacune d'elles étant introduite pour résoudre un élément particulier du problème. La conception est exprimée dès qu'il est possible d'exprimer toutes les ressources et toutes les instructions des différentes MP dans un langage choisi pour la programmation.

- Une ressource est une entité matérielle/ou logicielle dont dispose, ou peut disposer la MA pour exécuter ses instructions. On distingue deux types de ressources :

. ressources propres

. ressources partageables — ressources importées / ressources exportées.

- Une MA possède un jeu d'instruction comme une machine réelle.

I.4 Structuration d'une application par machines abstraites

La conception structurée d'une application est l'assimilation du problème posé suivant une certaine définition, à un ensemble de sous problèmes déduits par séparation du problème initial et rattachement à un ou plusieurs niveau d'abstraction, auxquels sont associées des machines abstraites spécialisées. Le système ainsi obtenu présente une structure réseau des relations entre les différentes MA qui le composent, dont on peut donner deux modèles :

- . une représentation globale de l'architecture conçue
- . et un modèle développé ultérieurement exprimant la circulation des données et les transferts du contrôle entre les différentes MA.

L'intérêt de tels modèles est de permettre au concepteur de définir intégralement son système par une représentation graphique.

I.5 Conclusion

Si on essaie d'appliquer la méthode à des applications de contrôle de procédés industriels, une machine abstraite sera introduite pour la commande d'une opération appliquée au procédé. Chacune des ces machines sera caractérisée par un ensemble de

ressources qui sont des entrées-sorties du système, et des opérations de communication avec l'environnement de l'application. Mais les éléments de décision permettant de guider le concepteur ne sont pas très explicites, et il n'y a pas de critères ou de règles permettant de satisfaire certaines contraintes que l'on peut se poser avant la conception d'une application.

II PROPOSITION DE M. JACKSON [JAK 78]

II.1 Introduction

La conception d'un système d'information peut être approchée à partir d'un point de vue fonctionnel. Partant de la question fondamentale "pourquoi un système ?" on progresse naturellement à se poser les questions suivantes :

"que doit faire le système ?"

"quelles sont les fonctions du système ?"

Puisque la fonction d'un système est généralement compliquée, on cherche une méthode pour maîtriser sa complexité, et on trouve cette méthode dans les raffinements successifs ou dans une conception descendante.

Le point de vue fonctionnel est largement accepté comme base nécessaire pour toutes les réflexions concernant une méthodologie de développement. Mais, malgré tout, il présente un certain nombre d'inconvénients, et une approche alternative mériterait des considérations sérieuses.

II.2 Inconvénients du point de vue fonctionnel

. Le premier inconvénient d'une approche fonctionnelle est tout simplement la difficulté, car la fonction totale du système

est très difficile à saisir par une seule personne à un instant donné. Comment par exemple, un concepteur, dans une première étape de raffinement, pour un détail ultérieur, ne peut rendre ses décisions invalides.

. Le second inconvénient est que la fonction du système est assujettie à des modifications arbitraires aussitôt que le système est construit.

. Le troisième inconvénient apparaît en premier en guise d'un avantage : il y a unification conceptuelle dans la conception des processus à partir du niveau le plus haut au niveau le plus bas du système. On considère chaque niveau du système conceptuellement comme une machine, au niveau le plus bas on a des machines réelles fournissant des opérations élémentaires, et au niveau le plus haut on a des machines virtuelles. Mais l'inconvénient crucial de cette unification, est qu'il n'y a pas de point clair auquel on étend les outils du concepteur, pour enlever ceux du constructeur.

II.3 Modélisation

M.A. JACKSON, suggère que ces difficultés soient raccourcies, en passant l'idée de "fonction" au second rôle. Au lieu de considérer le système comme un projet réalisant quelque chose pour calculer ses sorties à partir de ses entrées, on doit le considérer dans un premier temps comme un modèle de la réalité avec laquelle, il est concerné. La fonction du système est secondaire, et est considérée comme superposable au modèle : un modèle donné doit supporter plusieurs fonctions.

II.4 Forme de base d'un modèle

- Entités

Chaque entité active dans le monde réel, est représentée

par un processus dans le modèle. Si on considère que dans le monde réel, il y a des clients, des commandes, des fournisseurs et des employés, alors il doit y avoir dans le modèle autant de processus que d'entités actives dans la réalité ; "un processus pour chaque client, pour chaque commande,...".

- Communication de processus

La communication des processus est entièrement effectuée par des flux de données.

- Entrées/sorties du système

En plus des listes de données qui connectent les processus, il y a des listes de données qui connectent le système au monde réel. Ces listes de données représentent la synchronisation et la coordination avec l'extérieur.

II.5 Conclusion

La proposition de M.A. JACKSON consiste, en une structuration par modélisation du monde réel ; chaque processus du système est l'image d'une entité active dans le monde réel. C'est une approche par les objets qui est substituée à l'approche par les fonctions. Elle peut être intéressante dans le cas où l'on manipule des objets, dont on peut étendre ou modifier les opérations qu'on peut leur appliquer, par contre, il nous semble qu'elle n'est pas d'un apport dans le cas des applications de commande de procédés industriels : car chaque composante du procédé est caractérisée par une fonction bien déterminée. Et la proposition revient à associer à chaque entité du réel, une entité dans le système logique.

III PROPOSITION DE LONCHAMP [LON 83]

III.1 Introduction

La proposition d'une méthode de structuration en agents parallèles [LON 83] , se base sur la définition de trois types de parallélisme, qui sont les suivants :

(1) parallélisme de l'environnement
(2) parallélisme de conception
(3) parallélisme de la mise en œuvre

C'est une démarche de conception en trois étapes correspondant à la prise en compte de ces trois types de parallélisme :

- conception de l'architecture logique globale de l'application
- conception logique détaillée de l'application
- conception de la structure d'exécution.

III.2 Structuration logique globale de l'application

Cette étape de structuration est guidée pour l'essentiel par les exigences du parallélisme de l'environnement. Partant de la spécification initiale, elle consiste à décrire l'application comme un agent unique d'où entrent et sortent les messages externes (consignes, ordres, comptes rendus, sorties externes). L'analyse pour raffiner cet agent unique en plusieurs agents séquentiels consiste à établir une classification des couples de types de messages externes en deux catégories vis à vis des opérations d'entrée/sortie :

- couples sans possibilité d'entrée/sortie simultanées
- couples avec possibilité d'entrée/sortie simultanées.

La règle de base est que l'on s'interdit de "séquentialiser" artificiellement des entrées ou des sorties externes avec possibilités de simultanéité en les rattachant au même agent séquentiel.

Le reste de la structure s'appuie sur la spécification des relations entre entrées, sorties et données statiques, et sur la minimisation des flux de messages.

III.3 Conclusion

Les éléments de décision permettant de la guider sont :

- le parallélisme de l'environnement, ce qui nous semble très important pour les applications de commande de procédés industriels, car ceci doit aider au respect des contraintes de temps de réponse qui sont souvent très sévères.

- La minimisation des communications, qui permet d'avoir un couplage faible entre les agents constituants l'application.

IV SADT [IGL 82]

IV.1 Introduction

SADT est une méthodologie, créée en 1974 par Douglas T. ROSS pour fournir au concepteur d'une application, une approche disciplinée permettant d'étudier les problèmes posés par les besoins à satisfaire avant de chercher à leur apporter une solution. Sept concepts essentiels sont à la base de SADT.

IV.2 Concepts de SADT

. SADT étudie un problème en vue de construire un modèle dans le but d'exprimer une compréhension en profondeur de ce qu'est le problème. Plusieurs modèles SADT, exprimant

différents points de vue sur le problème pouvant être nécessaires pour la compréhension complète.

. L'analyse de tout problème est menée de manière descendante, modulaire, hiérarchique et structurée.

. SADT différencie autant qu'il est possible le modèle fonctionnel, et le modèle de conception : l'étude du problème et la description de sa solution.

. SADT modèlise à la fois les objets (documents, informations, données) et les activités (réalisées par des hommes, des machines, des programmes). Le modèle SADT complet d'un problème montre ces deux aspects et les liaisons entre eux.

. Le langage de SADT est graphique, il permet de montrer les modules, leurs relations et leur intégration dans une structure hiérarchique.

. SADT favorise un travail d'équipe discipliné et coordonné, et permet de mettre en évidence les résultats, qui reflètent le mieux la qualité de ce travail.

. SADT oblige à consigner sous forme écrite tous les choix importants faits pendant l'analyse et la conception. Les documents ainsi créés sont accessibles à tous pour être revus et éventuellement corrigés.

IV.3 Modèle SADT

Un modèle SADT est constitué par un ensemble de diagrammes. Au niveau le plus général, un seul diagramme résume l'ensemble du sujet abordé. Chaque diagramme des niveaux inférieurs

apporte un nombre limité de détails sur le sujet bien délimité. Chacun de ces diagrammes s'imbrique très exactement dans le modèle entier, précisant ainsi les relations entre ses composants et le reste du système.

SADT permet de s'intéresser aux deux aspects de tout système : les données et les activités. le modèle créé comporte deux parties : une décomposition des activités et une décomposition des données. La décomposition des activités détaille, les fonctions sous forme de boîtes, tout en montrant les objets qu'elles manipulent, sous forme de flèches. La décomposition des données détaille les données sous forme de boîtes, et les activités qui les créent ou les utilisent sous forme de fléches. Dans un diagramme d'activité, chaque boîte reçoit une étiquette qui est un verbe, sur la partie gauche de la boîte arrivent les données d'entrées étiquetées avec un nom. Sur le sommet arrivent les données de contrôle en fonction desquelles se déroulera l'activité. Cette information a deux buts : premièrement la distinction entre données d'entrées et données de contrôle permet au concepteur de différencier, parmi les données utilisées, la première fonction, celles qui ne seront pas transformées par l'activité de celles qui modifient le comportement de la fonction elle-même : les données de contrôle ; deuxièmement, cette distinction permet de mieux évaluer la cohésion de la représentation fonctionnelle que constitue les boîtes d'un diagramme. Si toutes les relations étaient du genre entrées/sorties, le couplage de type procédural serait le seul aspect à évaluer. Les relations établies par les données de contrôle permettent donc de raffiner l'évaluation du couplage et de donner une meilleure idée de la qualité du diagramme considéré.

Le dessous de la boîte est utilisé pour montrer le mécanisme supportant l'activité. L'étape finale du processus consiste à relier, à vérifier mutuellement, la décomposition en données et la décomposition en activités. Chacune sera systèmatiquement explorée, pour déterminer si les emplois des éléments duaux sont cohérents. Ceci permet de réduire les erreurs et les oublis qui auraient pu échapper aux vérifications précédentes.

IV.4 Conclusion

SADT est une méthode graphique permettant d'**analyser** et de spécifier une application, de façon à faciliter sa lisibilité et la communication des spécifications entre les différents partenaires impliqués dans le projet. Elle permet aussi de mettre en évidence les différentes relations entre les éléments pouvant constituer des modules, et d'évaluer leur couplage et leurs cohésions. Mais, elle ne propose aucun critère de structuration.

TROISIEME PARTIE

PROGRAMMATION

CHAPITRE I

INTRODUCTION

INTRODUCTION

La conception des langages de programmation, est influencée actuellement par deux facteurs :

- l'utilisation des langages de programmation modulaires et parallèles.
- Le développement d'architectures réparties.

Ces deux éléments conduisent à exprimer une application, comme un ensemble de processus relativement indépendant les uns des autres. Selon l'application considérée, les processus qui concourent à sa réalisation ont nécessairement des relations de **coopération** entre eux. De plus la mise en œuvre de plusieurs processus à l'aide d'un nombre limité de ressources, pose des problèmes de compétition. La coopération de plusieurs processus à l'exécution d'une tâche commune ; nécessite en général une communication d'information entre ces processus. L'ensemble des relations de compétition qui existent entre les processus d'une application, fixe leur déroulement dans le temps. Ceci définit la synchronisation entre ces processus.

Ces langages doivent répondre aux exigences suivantes :

- contenir des mécanismes de synchronisation et de communication
- pouvoir assurer le parallélisme des programmes
- permettre l'utilisation d'entrées/sorties, parmi lesquelles : entrées/sorties directes ; entrées/sorties sous contrôle du superviseur avec possibilité de mise en attente
- répondre aux contraintes de réception des signaux du type hiérarchisé. Ils doivent répondre en toute sécurité à l'acquisition des données qui arrivent aléatoirement et en volume variable
- répondre à des contraintes de temps de réponse pour les entrées/sorties et le traitement

- activités multiples d'un même traitement (disposer d'horloges), tenir compte du mécanisme de priorité
- protection et sécurité pour éviter la perte des données
- facilités de dialogue homme-machine
- surveillance de processus.

-=-=-=-=-=-

CHAPITRE II

DISCUSSION DU LANGAGE LTR-V3

II.1 INTRODUCTION

II.2 DISCUSSION

II.3 CONCLUSION.

-=-=-=-=-

II.1 INTRODUCTION

LTR-V3 est un langage de programmation destiné plus particulièrement à la programmation des applications temps réel. Il est constitué d'un langage du type PASCAL auquel ont été rajoutés certains concepts qui sont le prolongement de ceux de PASCAL, dont parfois ils lévent certaines limitations, et d'autre part de nouveaux concepts comme les exceptions, les modules, la communication qui seront décrits en annexe 1. Pour plus de détails, se reférer au manuel d'utilisation [PAR 84].

II.2 DISCUSSION

De la présentation précédente du mécanisme de communication, et suivant les caractéristiques du langage ; il ressort qu'un certain nombre de critères ne sont pas satisfaits en LTR-V3, et qu'ils le sont dans d'autres langages que l'on peut qualifier de la même famille. En plus, le langage n'offre pas certains bénéfices des méthodes nouvelles du génie logiciel (concernant la modularité et l'abstraction). Une étude non exhaustive de certains inconvénients est présentée ci-dessous :

- Le mécanisme de communication en LTR-V3 ne permet pas de définir une protection et un haut degré de localité, puisqu'il se base sur des ressources communes, contrairement à d'autres langages : ADA [ICH 79], DP [BRINCH HANSEN 78], CSP[HOA 78] et PLITS.

- En LTR-V3 (cas aussi de CHILL "CITT, HIGH LEVEL LANGAGE"), les outils de communication peuvent être utilisés de manière non fiable, et s'apparentent plus à des primitives de bas niveau. En effet, un processus en LTR-V3 qui a accès à une boîte aux lettres peut y prélever un message dont il n'est pas le destinataire, c'est

le cas ou un utilisateur écrit :

receive (T2, message) au lieu de
receive (T1, message) et le

processus exécutant cette instruction peut voir les deux tampons T_1 et T_2.

- En LTR-V3, il n'existe pas de structure de contrôle non déterministe, combinée avec des primitives RECEIVE qui permet des attentes multiples de messages, comme c'est le cas pour les autres langages cités ci-dessus (sauf PLITS et DP).

- Dans tous les langages cités (sauf CHILL), on ne peut pas répondre à une sollicitation parmi plusieurs avec une certaine priorité. Pour réaliser cela, il faut prendre la gestion des priorités attribuée aux messages dans les textes de programmes (on ne peut pas envoyer un message avec une certaine priorité).

- En LTR-V3, il est impossible de personnaliser la communication entre deux processus, en effet, dès qu'un processus peut accèder à une boîte aux lettres, tous les processus qui ont la même visibilité que lui pourront y accèder. Il serait possible de permettre tous les degrés de personnalisation, si on pouvait établir des boîtes aux lettres privées pour un nombre bien déterminé de processus.

Sur le sens de la communication, en LTR (cas aussi de CHILL), la symétrie fait perdre un avantage important du fait d'une mauvaise protection des tampons (les processus appelant peuvent voir les processus appelés), qu'est la possibilité d'exprimer des bibliothèques de processus réalisant des fonctions particulières : un processus récepteur ne peut voir le processus émetteur et n'a pas besoin de connaître son identité. Par exemple, si on considère un processus serveur et des processus clients, en ADA les processus clients peuvent être programmés indépendamment, ne faisant référence qu'à l'entrée de la communication. Tandis qu'en LTR, les processus

clients peuvent aussi déposer des messages et doivent connaître le nom du module auquel appartient le processus pour pouvoir accèder au tampon, il n'y a pas d'usage strict des boîtes aux lettres pour chaque catégorie de processus (émetteur ou récepteur).

- Tous les langages cités peuvent être supportés dans une architecture distribuée, sauf LTR-V3.

- Le degré de couplage entre processus communiquants en LTR, est plus fort que dans les autres langages, puisque les interlocuteurs communiquent à l'aide de variables qu'ils partagent dans un environnement commun. Ce qui présente les inconvénients suivants :

i) la modification d'un processus peut entraîner une modification dans tous les modules comportant ses interlocuteurs, c'est le cas de modification d'une caractéristique d'un tampon par exemple.

ii) Ce type de communication rend difficile la réutilisation des modules dans lesquels sont définis des processus communicants.

- Un inconvénient commun à tous les langages cités : la communication ne fait intervenir que deux interlocuteurs à un instant donné, il n'y a pas de primitive de diffusion.

II.3 CONCLUSION

De l'étude précédente, il ressort qu'il serait souhaitable de disposer d'un langage de haut niveau permettant :

- de rendre plus effective la sécurité, la fiabilité et la lisibilité des applications constituées de processus communiquants.
- une construction facile des solutions des problèmes de communication

- d'exprimer toute forme de communication : synchrone, asynchrone, symétrique,...
- la séparation entre la description des processus et l'expression de leur communication.
- de disposer de types de communication prédéfinis.

-=-=-=-=-

CHAPITRE III

-=-=-=-=-

III.1 INTRODUCTION

Pour intégrer les concepts de FLEXI décrits dans le chapitre précédent à LTR-V3, on est conduit à résoudre deux problèmes :

- Premièrement, il n'existe pas de compilateur qui permet de compiler un programme écrit en Flexi. Pour rendre une telle compilation possible, il faut, soit modifier le compilateur LTR-V3 pour prendre en compte les nouveaux concepts, soit utiliser un préprocesseur qui permet de passer d'un texte écrit en FLEXI en un texte LTR-V3. (C'est cette deuxième solution que nous préconisons).

- Deuxiémement, il faut définir et implanter un noyau de communication. Ce noyau de communication sert d'interface entre les tâches de l'application et la station de transport.

III.2 STATION DE TRANSPORT

C'est un logiciel, un matériel ou un assemblage plus ou moins complexe de composants logiciels et/ou matériels grâce auquel deux entités coopérent selon un protocole constituant l'ensemble des conventions permettant la protection contre des pannes de communication et assurant la synchronisation. Cette station de transport se compose de plusieurs stations réparties. Selon le réseau dont on dispose, elle peut faire partie du réseau de transport, ou faire partie du niveau de traitement, ou encore divisée en deux parties l'une dans le niveau traitement, l'autre dans le réseau de transport [LOR 79].

Remarque :

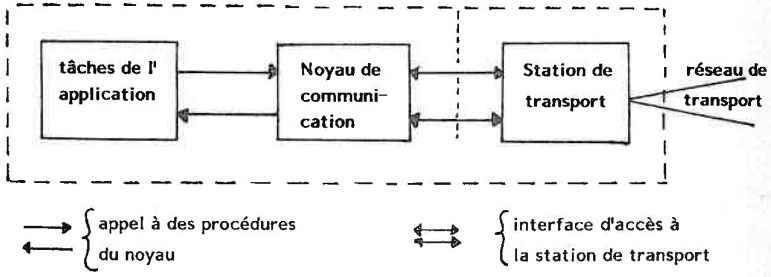
Nous prenons comme exemple de stations de transport, la station ST version 2 de cyclades [MAM 82], dont les principales fonctions seront décrites dans le paragraphe IV.5.

III.3 NOYAU DE COMMUNICATION

3.1 Définition

Son principal rôle est d'offrir un ensemble de primitives que les tâches de l'application utilisent pour communiquer les unes avec les autres. Il constitue une couche intermédiaire du système, et sa spécification externe est définie par les couches traitement (applications) et interface de transport. On considère alors, un noyau de communication comme :

- un ensemble de tâches
- un ensemble de structure de données
- et une interface d'accès à la station de transport



Le noyau de communication est perçu par les tâches d'une application comme une boîte noire, qu'elles peuvent invoquer par des appels de procédures. Il fait partie du niveau traitement, et doit s'exécuter concurrement avec les composantes de l'application. Une démarche simple de réaliser cette concurrence, est celle qui consiste à considérer le noyau de communication comme une composante de l'application, et par conséquent de les faire supporter ensemble par le même système.

Le noyau de communication assure à la demande les fonctions suivantes :

- demander à la station de transport l'émission et la réception des messages, entre modules de l'application (sur le réseau)
- demander au système supportant l'application, l'émission et la réception de messages externes (entre les modules de l'application et son environnement).
- demander la connexion des ports par la station de transport
- demander la déconnexion des ports d'un module, à la fin d'exécution de celui-ci.

3.2 Structure externe du noyau de communication

Des fonctions citées ci-dessus découle la structure suivante du noyau de communication :

. Tâches du noyau de communication

- tâche "demande connexion"

Cette tâche permet d'attendre l'établissement des connexions des ports par la station de transport, et de lancer l'exécution d'un module dont tous les ports d'entrée, de sortie et de service sont connectés à l'extérieur.

- Tâche "demande - déconnexion"

Elle permet de gérer les déconnexions des ports de l'application, en signalant à la station de transport, la déconnexion d'une liaison à la suite de la fin d'exécution d'un module.

. Procédures du noyau de communication

- Procédure "envoyer-notification"

Cette procédure permet la réalisation de l'envoi du type de communication T_1 de FLEXI, elle correspond à la primitive NSEND de FLEXI.

- Procédure "Recevoir-notification"

Cette procédure permet la réalisation de la réception du type de communication T_1 de Flexi. Elle correspond à la primitive NRECEIVE de Flexi.

- Procédure "envoyer-commande"

Cette procédure permet la réalisation de l'envoi du Type de communication T_2 de Flexi. Elle correspond à la primitive BSEND de Flexi.

- Procédure "recevoir-commande"

Cette procédure permet la réalisation de la réception du type T_2 de Flexi. Elle correspond à la primitive BRECEIVE de Flexi.

- Procédure "envoyer-service"

Cette procédure permet la réalisation de l'envoi du Type de communication T_3 de Flexi. Elle correspond à la primitive SSEND de Flexi.

- Procédure "recevoir-service"

Cette procédure permet la réalisation de la réception du type T_3 de Flexi. Elle correspond à la primitive BRECEIVE de Flexi.

Tâche émission

Elle est chargée de la gestion des émissions de messages.

Tâche réception

Elle est chargée de la gestion de la réception des messages.

Procédure envoi-externe

Chargée de la gestion des émissions de messages externes.

Procédure réception-externe

Chargée de la gestion de la réception de messages externes.

Les appels à ces procédures avec les paramètres effectifs adéquats sont engendrés par le préprocesseur, et traduisent les actions de communication dans le texte de l'application sur le réseau.

Le programme principal de chaque site activera les tâches "demande-connexion" et "demande déconnexion" au début du lancement de l'application. Les phrases correspondant à ces activations sont engendrées par le préprocesseur.

INTERFACE DU NOYAU DE COMMUNICATION

```
INTERFACE OF NOYAU
    Procédure envoyer-notification (P : integer ; m : Refinout word (*) ;
                                    type : word (*) ; priorité : integer) ;
    Procédure recevoir-notification (P:integer ; m:Refinout word (*); type:word (*)) ;
    Procédure envoyer-commande  (P:integer ; m:Refinout word (*) ; type: word (*);
                                    priorité : integer) ;
    Procédure recevoir-commande  (P:integer ; m:Refinout word (*) ; type:word (*) ;

    Process     demande-connexion ;
    Process     demande-déconnexion ;
    Process     émission ;
    Process     réception;

    Procédure   envoi-externe (message : word (*)) ;
    Procédure   réception-externe (message : word (*)) ;
END INTERFACE
```

3.3 DESCRIPTION DES FONCTIONS DU NOYAU DE COMMUNICATION

3.3.1 Points de synchronisation entre tâches du noyau et tâches de l'application

A chaque tâche de l'application, sont associés quatre évènements :

- l'événement expiration de délai d'attente : "expiration délai"
- l'événement qui permet de réveiller la tâche pour continuer son exécution, une fois que toutes les connexions du module sont établies : "Démarrage des tâches"
- l'événement qui permet à une tâche d'attendre un message : "attente de message"
- l'événement qui permet de réveiller une tâches en attente sur saturation d'anticipations : "fin saturation"

Ces événements sont utilisés à l'intérieur des procédures { BRECEIVE, BSEND, SRECEIVE }.

3.3.2 Structures de données

Pour réaliser ses fonctions le noyau de communication dispose d'une collection d'informations réunie par le préprocesseur. Cette collection d'informations est constituée de deux tables :

a) Tables d'implantion des modules : "TABLE-MODULES"

Cette table contient la liste de répartition des modules sur les sites du réseau, avec pour chaque module la liste de ses ports :

nom du module	nom du site	liste des ports

TABLE DES MODULES

b) Table des ports : "TABLE DES PORTS"

Cette table contient les informations relatives aux ports implémentés sur le site. Chacune des entrées de la table contient les informations suivantes :

- nom du module auquel appartient le port
- liste des types de messages qui peuvent transiter par le port
- nombre d'anticipations
- chaîne des connexions : pour un port de sortie, elle fournit la liste des ports d'entrée qui lui sont connectés. Cette liste contient :
  - * le nom du port d'entrée
  - * le nom du module auquel il appartient
  - * un pointeur sur le descripteur du flot qui existe entre ce port et le port de sortie considéré.
  - * un pointeur sur l'élément suivant de la liste.

Pour un port d'entrée, elle fournit :

- la liste des messages attendus sur ce port, chaque élément de cette liste contient :
  - * le type du message attendu, priorité
  - * l'adresse où sera rangé le message attendu
  - * une variable contenant l'adresse où sera rangé le nom du port de sortie sur lequel la réponse sera envoyée, dans le cas d'une demande de service
  - * un pointeur sur l'élément suivant

- la liste des messages reçus et non encore consommés, chaque élement de cette liste contient :

  * le type du message reçu et sa priorité

  * le texte du message

- pour un port d'entrée, elle indique le descripteur de la porte locale associé à ce port
- zone de communication : zone intermédiaire qui permet de stocker les lettres en provenance de la station de transport avant d'être décomposées en messages, types de messages et priorité
- Etat port : indique si le port est connecté avec l'extérieur ou non
- un pointeur sur le port suivant appartenant au même module.

  * Les lettres échangées entre stations de transport ont la structure suivante :

Type message	Texte message	Priorité

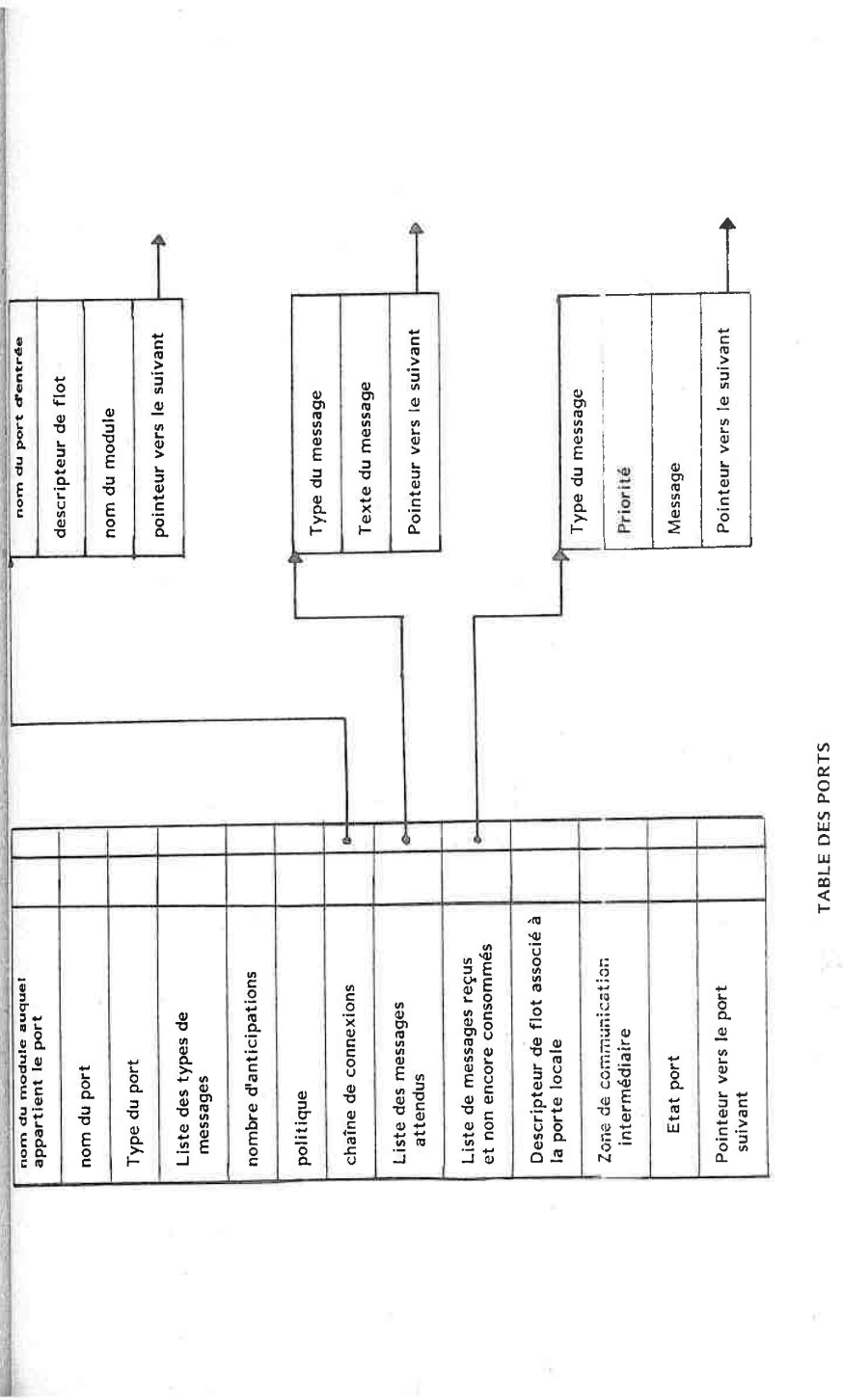


TABLE DES PORTS

3.3.3 Description des tâches du noyau

3.3.3.1 Tâche "Demande de connexion"

- Cette tâche utilise les tables "TABLE MODULES" et "TABLES DES PORTS", et les opérations de la station de transport "FCB" et "DOPEN"

- Fonctions

. Demander l'occupation des tables "TABLE MODULES" et "TABLE des PORTS".

. Pour chacun des ports de "TABLE des PORTS" demander une ouverture de flot à la station de transport

. Attendre l'évènement "EVT CONNECT"

. Si ce port est un port d'entrée, mettre son état à l'état valide sinon (il s'agit d'un port de sortie), si tous les flots, dont ce port est la source sont ouverts, mettre l'état de ce port à l'état valide.

. Si ce port, est le dernier port dont la connexion est attendue par les tâches du module auquel il appartient, signaler l'événement "démarrage des tâches" pour lancer l'exécution des tâches d'un module dont tous les ports sont connectés.

. Libérer les tables.

3.3.3.2 Tâche "demande de déconnexion"

- Cette tâche utilise les tables "TABLE MODULES" et "TABLE DES PORTS" l'opération "DCLOSE" de la station de transport, et l'événement EVTDISCN signalé par l'interface de transport.

- Fonctions

. Demander l'occupation des tables

. Pour tout module qui termine son exécution, demander une déconnexion de tous ses ports.

. attendre l'évènement EVTDISCN

. Mettre l'état de ces ports à l'état invalide

. Libérer les tables.

3.3.3.3 Tâche "émission"

Elle utilise :

- une file "Demande émission" qui contient les demandes d'émission de messages (descripteur, longueur, adresse)
- un évènement "EVTEMISSION" qui est signalé par une tâche de l'application
- table des ports
- l'opération "DSENDLT" de la station de transport.

Fonctions :

. attendre l'événement "EVTEMISSION"

. remettre cet évènement à l'état non arrivé

. demander l'occupation de la table des ports

. tantque la file "demande émission" est non vide faire

- prélever un élément de la file
- appeler l'opération "DSENDLT"
- signaler l'évènement "fin saturation"

fait

. libérer la table des ports.

3.3.3.4 Tâche "réception"

Cette tâche utilise :

- une file "Demande de réception", qui contient les demandes de réception faites par les tâches de l'application
- un évènement signalé par une tâche de l'application : "requête réception", lorsqu'elle veut recevoir un message
- un évènement "message reçu", signalé par la station de transport
- table des ports
- l'opération DRECLT de la station de transport.

Fonctions :

. attendre l'un des deux événements "requête réception" ou "message reçu"

si "requête réception" est signalé alors

- demander l'occupation de la table des ports
- prélever tous les éléments qui sont dans la file
- appeler l'opération "DRECLT"
- remettre l'événement "requête réception" à l'état non arrivé.

. sinon pour chaque port d'entrée :

. si une lettre a été reçue sur la porte associée à ce port alors décomposer la lettre en un message, un type et une priorité (éventuellement)

. ajouter le message à la liste des messages reçus

. remettre l'événement "message reçu" à l'état non arrivé

. signaler l'événement "attente de message".

3.3.4 Procédures du noyau

1 * Procédure "envoyer-notification"

L'appel à cette procédure se fait comme suit :

envoyer-notification (num : integer ; m : words (*) ; p : integer ; TYP : word (*))

où num : est un port de sortie
n : message à envoyer
p : prioriété éventuelle d'envoi du message
TYP : type du message

Fonctions

- construction d'une lettre ayant la forme :

Type message	texte message	priorité

- demander l'occupation de la table des ports
- pour chaque port d'entrée connecté au port dont le numéro est num, émettre vers la station de transport une lettre sur le flot existant entre ces deux ports
- libérer la table des ports
- signaler l'événement "EVTEXPRESSION" à la tâche émission

2 * Procédure "recevoir-notification"

L'appel à cette procédure se fait comme suit :

recevoir-notification (num : integer ; n : word (*) ; TYP : word (*)) ;

où num : numéro du port d'entrée
m : message attendu
TYP : type du message

Fonctions :

. Demander l'occupation de la table des ports

. Si dans la liste des messages reçus sur le port considéré, il existe un message de type TYP alors, affecter ce message au message attendu (les messages sont prélevés selon la politique indiquée dans le port, on prend toujours le message de priorité la plus élevée).

3 * Procédure "envoyer-commande"

L'appel à cette procédure se fait comme suit :

envoyer-commande (num : integer ; m : word (*) ; TYP : word (*) ; P : integer) ;

où num : numéro d'un port de sortie
n : message à envoyer
TYP : type du message
P : priorité
T : délai d'attente (dans le cas où le délai d'attente n'est pas spécifié T = 0)

Fonctions

. Demander l'occupation de la table des ports
. Construire une lettre de la forme

Type message	Texte message	Priorité

. Pour chaque port d'entrée connecté au port considéré, émettre vers la station de transport une lettre sur le flot existant entre ces deux ports.
. si nombre anticipations = nombre de requêtes non satisfaites alors attendre "fin saturation" ou "expiration délai"
. libérer la table des ports
. signaler l'événement "EVTEMISSION" à la tâche émission

4 * Procédure "recevoir-commande"

Elle est appelée comme suit :

recevoir-commande (n : integer : m : word (*) ; TYP : word (*) ;
T : integer)

où

n : numéro du port
m : message attendu
TYP : type du message
T : délai d'attente
quand ce délai d'attente n'est pas spécifié, T = 0

Fonctions :

. Demander l'occupation de la table des ports
. Si dans la liste des messages reçus sur le port considéré, il existe des messages de type TYP, alors prélever un message de cette liste selon la politique indiquée dans le port (on prend toujours le message de plus haute priorité). Supprimer le message prélevé de la liste des messages reçus.

. sinon, ajouter cette demande à la liste des messages attendus
- signaler "requête-réception"
- attendre l'un des évènements "atteste-de-message" ou "expiration de délais".

5 * Procédure "envoyer-service"

L'appel à cette procédure est réalisé par les deux appels aux procédures suivantes :

envoyer-commande (-, -, -, -)
recevoir-commande (-, -, -, -)

6 * Procédure "recevoir-service"

L'appel à cette procédure est réalisé par les deux appels aux procédures suivantes :

recevoir-commande (-, -, -, -)
envoyer-commande (-, -, -, -)

III.4 LE PREPROCESSEUR

4.1 Définition

Le préprocesseur a deux fonctions principales :

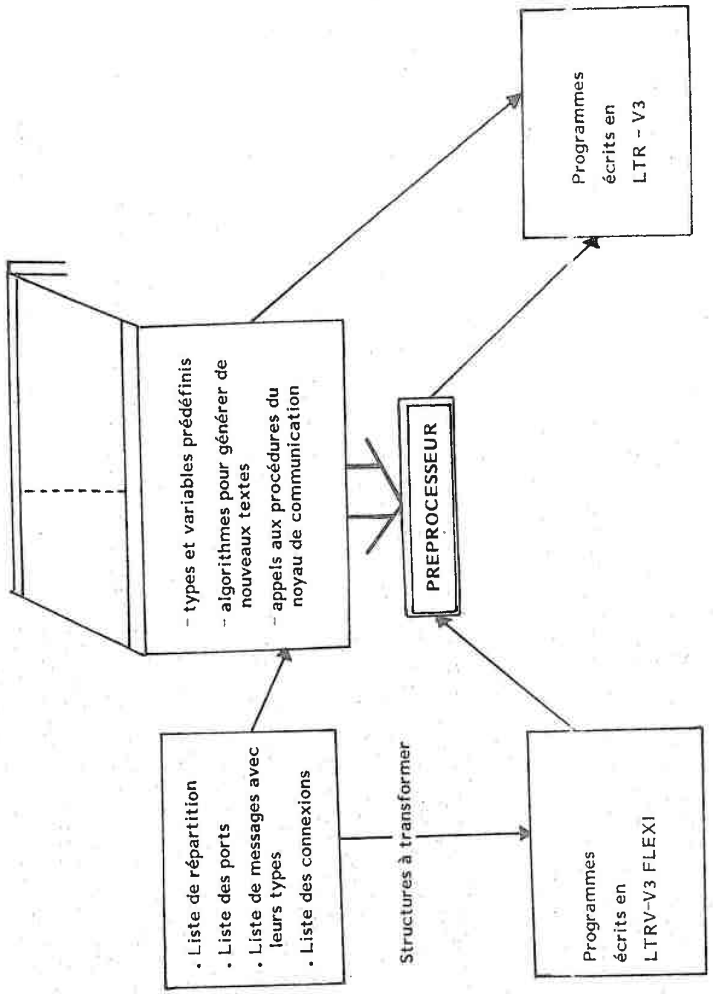
- une fonction de contrôle
- et une fonction de transformation.

La transformation des structures suppose, pour chaque structure, la définition de :

- Types et variables prédéfinis insérés directement par le préprocesseur dans le texte à transformer
- un texte à générer en fonction de la structure à transformer

- et des procédures du noyau de communication réalisant les opérations correspondant à la structure à transformer.

Les contrôles syntaxiques et sémantiques sont faits par les préprocesseurs, ceci permet d'optimiser le temps d'exécution des services du noyau de communication.



4.2 Transformations et générations par le préprocesseur

Les concepts de FLEXI qu'on veut intégrer à LTR-V3 sont décrits en annexe 2. Nous nous proposons dans ce paragraphe de spécifier les différents concepts et structures à transformer par le préprocesseur, les différentes phrases à générer par le préprocesseur, et les différents contrôles à effectuer par celui-ci.

4.2.1 Liste de répartition

Syntaxe

{ MACHINE < nom machine > : < liste de modules >
END MACHINE }*

< liste de module > ::= < nom du module > ; [< liste de module >]

A la rencontre d'une phrase de répartition, le préprocesseur génère l'appel à une procédure "répartition des modules" (nom machine, liste des modules), qui va insérer dans la table des modules :

- nom machine
- et liste des modules.

4.2.2 Liste de connexions

Syntaxe

LINK {< PCONNEXION >, }* ENDLINK
< PCONNEXION > ::= < Port de sortie > TO { < Port d'entrée > } *
< Port de sortie > ::= < nom de port >
< Port d'entrée > ::= < nom module >, < nom de port >

A la rencontre d'une phrase de connexion, le préprocesseur génère un appel à une procédure LINK (Port de sortie, nom du module, liste des ports d'entrée)

qui va insérer dans la table des ports :

- Port de sortie
- et faire le chaînage de celui-ci avec la liste des ports d'entrée auxquels il est connecté.

4.2.3 Les ports

EXTERNAL (ou INTERNAL)
ENTRY (ou EXIT, EXIT-ENTRY, ENTRYEXIT) PORTS
nom-du-port : Type 1, Type 2,..., anticipations
.
.
.
END ENTRY (ou EXIT,...)

A la rencontre d'une telle phrase le préprocesseur génère un appel à une procédure qui va insérer dans la table des ports, les types du port et le nombre d'anticipations.

4.2.4. Primitives de communication

4.2.4.1. Primitive

NSEND < idf de message > FROM < idf de port de sortie > [priorité]

Lors de la rencontre d'une telle primitive, le préprocesseur génère l'appel à la procédure du noyau suivant :

envoyer-notification (idf de port sortie, idf message, priorités)
Type du message

dans le cas où l'option priorité n'est pas spécifiée, elle est remplacée par zéro dans l'appel de la procédure.

4.2.4.2 Primitive

N-RECEIVE < idf de message > ON < idf de port d'entrée >

Lors de la rencontre d'une telle primitive, le préprocesseur génère l'appel à la procédure suivante d'un noyau :

recevoir-notification (idf de port d'entrée, idf de message, type du message) ;

4.2.4.3

Primitive BSEND < idf de message > FROM <idf de port de sortie >
[priorité], [Time out de (P)]

A la rencontre de cette primitive le préprocesseur génère l'appel à la procédure suivante du noyau :

envoyer-commande (idf de port de sortie, idf message, priorité,
type du message, P)

Dans le cas où la priorité et le timeout ne sont pas spécifié, ils sont remplacés par des zéros.

4.2.4.4

Primitive BRECEIVE < idf message > ON < idf port d'entrée >
[timeout (P)]

A la rencontre d'une telle phrase, le préprocesseur génère l'appel à la procédure suivante du noyau :

recevoir-commande (idf port d'entrée, idf message, type de message, P) dans le cas où le timeout n'est pas spécifié, le P est remplacé par zéro dans l'appel.

4.2.4.5

Primitive SSEND < message1 > FROM < Port service >
Réponse < message2 > [priorité], [Time out (P)]

A la rencontre d'une telle phrase, le préprocesseur génère les appels aux procédures suivantes du noyau :

- envoyer-commande (idf port service, message 1, type message 1, priorité,B)

Dans le cas où la priorité et le timeout ne sont pas spécifiés, ils sont remplacés par des zéros.

- recevoir-commande (idf port service, message 2, type message2,P)

Dans le cas où le timeout n'est pas spécifié, le P est remplacé par zéro.

4.2.4.6

Primitive SRECEIVE < message1 > ON < port de service >
REPLY < message 2 > [Time out (P)]

A la rencontre d'une telle phrase, le préprocesseur génère les appels aux procédures suivantes du noyau :

recevoir-commande (port de service, message1, type message1,0,P)
envoyer-commande (port de service, message2, type message2,0,0).

4.2.4.7 Texte à insérer dans le programme principal de l'application

Dans le début du programme principal le préprocesseur doit insérer les phrases suivantes :

```
SCHEDULE "Demande connexion"
SCHEDULE "Demande déconnexion"
SCHEDULE "Emission"
SCHEDULE "Réception"
```

Il doit insérer aussi dans les interfaces des modules de l'application la directive "USE NOYAU".

4.3. Contrôles à effectuer par le préprocesseur

On ne dispose que d'un compilateur LRTV3, il est donc nécessaire de transformer une application LTR-V3-FLEXI en une application LTRV3. Le préprocesseur assure le contrôle syntaxique de tout ce qui n'est pas du LTR-V3. Les contrôles effectués sont les suivants :

- Il détecte les noms des identificateurs qui sont dans le noyau de communication, et utilisés dans une application LTR-V3-FLEXI
- la déclaration d'un message qui fait référence à un type non déclaré
- si un message n'est pas interne et externe à la fois
- la déclaration d'un port qui fait référence à un type non déclaré
- si un port n'est pas interne et externe à la fois
- si un port de service n'est pas utilisé avec un nombre d'anticipations supérieur à 1
- si une connexion fait réfèrence à un module ou à un port inexistants
- si une connexion utilise comme source un port d'entrée
- si une connexion utilise comme destination un port de sortie
- si un port de service appartient à une structure convergente ou divergente
- si les types autorisés par les ports d'entrée et les ports de sortie d'une connexion ne sont pas comptatibles
- une demande d'émission sur un port d'entrée
- une demande de réception sur un port de sortie
- si le port utilisé dans une action de communication n'est pas déclaré
- le type du message n'est pas autorisé par le port auquel la demande est faite
- une action de communication fait référence à un port et un message dont les types ne sont pas compatibles
- un module interne déclare un port ou un message externe
- un port de sortie n'est pas utilisé dans plusieurs connexions
- les actions de communication des types notification ou commande ne font par référence à un port de service
- l'action d'envoi du type de communication service, n'utilise qu'un port de service du type EXIT.ENTRY
- l'action de réception du type de communication service, n'utilise qu'un port de service du type ENTRY-EXIT.

CONCLUSION

Dans ce travail, nous nous sommes intéressé à décrire une méthode de structuration logique des applications de contrôle de procédés industriels. Ces procédés industriels sont, par hypothèse, décrits complétement dans un cahier des charges qui précise l'ensemble des éléments les caractérisant, à savoir : les macroactions, les actions, les variables et les structures d'utilisation.

Partant de ces données, nous appliquons un ensemble de règles (dans leur ordre de définition), aboutissant à une structure fonctionnelle spécifiée par un ensemble de modules à faible couplage et forte cohésion entre leurs éléments. La discussion des régles énoncées montre bien qu'elles vont dans le sens de ces objectifs.

Notre proposition ne prend pas en compte les problèmes relatifs aux structures d'exécution, qui peuvent faire l'objet d'un travail ultérieur ; néanmoins nous avons proposé des idées qui permettent d'adapter une structure fonctionnelle à une structure d'exécution. D'autre part, il est à remarquer que la méthode proposée ne conduit pas à une structure fonctionnelle figée, ce qui permet donc d'avoir une certaine souplesse envers la détermination d'une structure d'exécution adaptée.

Nous estimons que cette correspondance (structure fonctionnelle —> structure d'exécution) nécessite une étude approfondie des différents types de couplage en vue de proposer des règles de détermination d'une structure d'exécution.

-=-=-=-=-

ANNEXE 1

LE LANGAGE LTR-V3

Le Langage LTR-V3

1. STRUCTURE D'UNE APPLICATION

Une application LTR-V3 est constituée d'un ensemble de modules ayant des liens de visibilité entre eux et d'un programme principal.

2. STRUCTURE D'UN MODULE

L'outil de structuration principal du langage est le module, il permet de grouper différentes déclarations. Il comporte deux parties :

a) une partie interface (facultative), qui permet d'encapsuler un ou plusieurs modules, ne laissant visible que certains objets : ces objets peuvent être des constantes, des types, des variables, et des entêtes de sous-programmes, ou de processus.

b) Une partie du module (facultative). Cette partie peut comporter : des déclarations modulaires internes, des directives d'implémentation (facultatives), des articles de traitement et des directives USE.

3. LES PROCESSUS.

Les processus définissent des traitements qui peuvent être exécutés en parallèle, et pouvant communiquer entre eux, ou se synchroniser. La synchronisation et la communication peuvent être affinées par des primitives temps réel qui sont les suivantes :

1) Phrase de création de tâche : SCHEDULE

Cette phrase permet de créer et d'activer sur le processus indiqué, une nouvelle tâche avec les paramètres et la priorité spécifiés :

- si aucune condition n'est spécifiée, la tâche est créée dans l'état éligible, et se déroule de façon indépendante de la tache créatrice

- Si une condition CLOSED est spécifiée, la tâche créatrice est mise en attente de la fin d'exécution de la tâche créée

- si une condition WAITING est spécifiée, la tâche est créée en attente de l'expression d'événements indiqués

- si une condition EVERY est spécifiée, une nouvelle tâche sera activée sur le même processus, avec la période indiquée, créant ainsi un cycle de tâches.

2) Phase d'attente : WAIT (expression d'événement)

Cette phrase permet à une tâche de se mettre en attente d'une ou de plusieurs expressions d'événements. Les expressions spécifiées sont successivement évaluées, si l'une d'entre elles est réalisée (valeur "LIP"), le groupe de phrases correspondant est exécuté, sinon la tâche est mise en attente. Et si l'une des expressions se réalise, la tâche est mise à l'état éligible. Lorsqu'elle passe à l'état "en cours d'exécution", le groupe de phrases correspondant à l'expression d'événements réalisée est exécutée.

3) Phrase d'accés : ACCESS

Cette phrase mentionne une liste de déclarations. Deux phrases d'accès faisant référence à un même article de déclaration, s'exécutent en exclusion mutuelle.

4. SYNCHRONISATION EN LTR-V3

Les divers processus d'un système n'évoluent pas indépendamment, il existe entre eux des relations qui dépendent de la logique de la tâche à accomplir, et qui fixent leur déroulement dans le temps. Cet ensemble de relations définit la synchronisation entre les processus du système. Cette synchronisation consiste donc à construire un mécanisme permettant :

- a un processus d'en bloquer un autre, ou se bloquer lui-même en attendant un signal d'un autre processus.
- D'activer un processus ou un ensemble de processus éventuellement en attente.

La synchronisation en LTR-V3, est une synchronisation par événement, c'est le mécanisme d'actions indirectes. Elle met en jeu des objets intermédiaires (appelés événements) connus des processus coopérants et manipulables par eux uniquement à travers des opérations indivisibles. Ces objets intermédiaires appartiennent à une classe de données partageables par tous les processus concernés par la synchronisation.

Ces événements sont des événements mémorisés et représentés par des variables qui à un instant donné prennent la valeur arrivée ("up"), ou non arrivée ("down"). Un processus se bloque si et seulement si l'événement qui l'attend n'est pas arrivé. Le déclenchement d'un événement débloque un processus ou tous les processus qui l'attendent.

Le type prédéfini "EVENT" permet de représenter ces objets. C'est un type à usage restreint sur lequel sont définies trois opérations qui sont les suivantes :

- "SETEV" permet de mettre un événement à l'état "up"
- "RESETEV" permet de mettre un événement à l'état "down"
- "PULSEV" permet de réaliser en une seule opération l'équivalent des opérations "SETEV" et "RESETEV".

5. COMMUNICATION INTER-TACHES

5.1 Caractéristiques de cette communication

La communication en LTR-V3 est établie par un mécanisme de boîtes aux lettres, avec des primitives SEND permettant d'adresser un

message dans une boîte aux lettres, et des primitives RECEIVE permettant de prélever un message d'une boîte aux lettres. Elle a les caractéristiques suivantes :

- Asynchrone

Le processus émetteur peut adresser un message sans se préoccuper de l'état du destinataire, et inversement le récepteur peut prélever le message quand il en a besoin sans se préoccuper de son producteur.

- Indirecte

Lors de l'envoi d'un message, le destinataire n'est pas spécifié explicitement, on passe par l'intermédiaire d'un buffer qui sert de boîte aux lettres.

- Non déterministe

Il y a attente d'un message de n'importe quel processus. La primitive de réception est de la forme suivante :

Receive message ; sans aucune identification du processus émetteur.

- Symétrique

Elle est exprimée en deux opérations symétriques : l'envoi d'un message et la réception d'un message. Elle est vue de façon analogue du côté de l'émetteur et du côté du récepteur.

- Bi-point

A un instant donné, elle ne fait intervenir que deux interlocuteurs.

5.2 Envoi et réception de messages

2.1 Envoi de messages

En LTR-V3, on distingue trois types d'envoi de messages :

1) Envoi avec attente sur saturation.

La procédure prédéfinie "SEND" possède deux paramètres :

- le premier désigne une boîte aux lettres
- le second désigne le type ou le sous-type figurant dans la déclaration de boîte aux lettres.

Elle permet de déposer un message dans la boîte aux lettres. Si des tâches sont en attente de message sur cette boîte aux lettres, celui-ci est attribué à la tâche la plus ancienne en attente dans la priorité la plus élevée. Si la boîte aux lettres est saturée, la tâche est mise en attente de dépot de message.

2) Envoi de message avec délai d'attente

La procédure prédéfinie "SEND" comporte trois paramètres :

- le premier désigne une boîte aux lettres
- le second désigne un objet de type ou sous type figurant dans la déclaration de la boîte aux lettres
- le troisième désigne le délai d'attente.

C'est un envoi avec un délai d'attente sur la fin de saturation de la boîte aux lettres.

3) Envoi de message sans mise en attente sur la saturation :

La procédure "Test and SEND" comporte deux paramètres :

- le premier désigne une boîte aux lettres
- le second désigne un objet de type ou sous type figurant dans la déclaration de la boîte aux lettres.

Elle permet de tester si la boîte aux lettres est saturée, et dans ce cas là, le processus exécutant cette primitive n'est pas bloqué, et dans le cas contraire, il y a dépot du message.

2.2 Réception des messages

Comme pour l'envoi, il y a trois types de réception de messages.

1) Réception avec attente

La procédure prédéfinie "RECEIVE" comporte deux paramètres :

- le premier désigne une boîte aux lettres
- le second désigne un objet de type ou sous type figurant dans la déclaration de la boîte aux lettres :

Elle permet de prélever le plus ancien message de la boîte aux lettres, si cette dernière est vide, le processus exécutant cette primitive est mis en attente.

2) Réception de messages avec délai de garde

La procédure prédéfinie "RECEIVE" comporte trois paramètres :

- le premier désigne une boîte aux lettres
- le second désigne un objet du type ou sous type figurant dans la déclaration de boîte aux lettres
- le troisième désigne le délai d'attente.

Elle offre les mêmes fonctions que la procédure SEND avec délai de garde.

3) Réception de messages sans mise en attente :

La procédure prédéfinie "TEST and RECEIVE" comporte deux paramètres :

- le premier désigne une boîte aux lettres
- le second désigne un objet de type ou sous type figurant dans la déclaration de la boîte aux lettres.

Elle offre les mêmes fonctions que la procédure "TEST and SEND".

ANNEXE 2

LE LANGAGE FLEXI

1. INTRODUCTION

Pour plus de détails se référer à [ZAK 84]. Une application Flexi est un ensemble de modules communicants, sa programmation se fait en deux niveaux :

- le niveau application : il consiste à caractériser les modules, et décrire leurs échanges : types des messages, les connexions, types des communications, etc...
- Le niveau module : il consiste à écrire les modules dans un langage de programmation existant (LTRV3, ADA, ...), pour compléter les interfaces de modules décrites en Flexi.

Les paragraphes suivants, exposent les principaux concepts que propose Flexi pour décrire une application temps réel, qu'on veut intégrer à LTRV3.

2. LE CONCEPT DE PORT

Un port est un point d'entrée ou de sortie d'un module, c'est une entité de communication intermédiaire sur laquelle sont définies des opérations d'envoi et de réception de message. Dans Flexi, on distingue deux types de ports :

- des ports utilisés uniquement en entrée ou uniquement en sortie
- des ports de "service" utilisés en entrée/sortie.

Remarque :

En Flexi, un port d'entrée est considéré comme propre à une tâche, et un port de sortie est partageable pour toutes les tâches d'un module. On propose la rectification suivante : tous les ports d'entrée ou de sortie sont partageables par toutes les tâches du module.

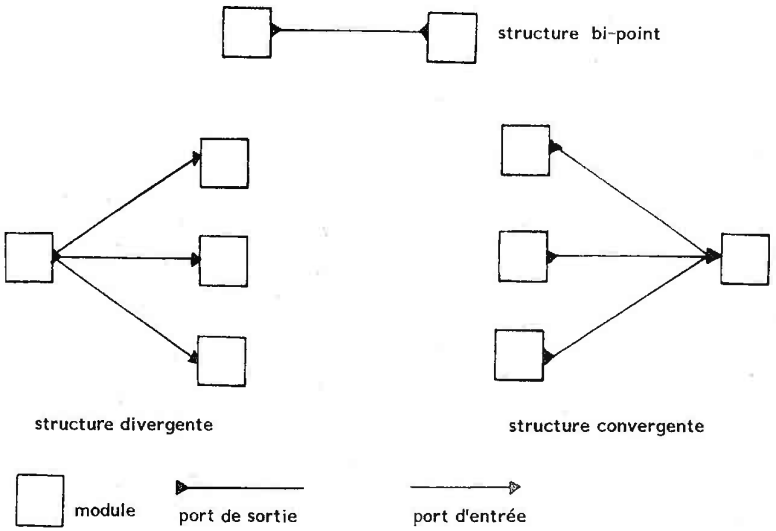
3. LES CONNEXIONS

Une connexion est définie du point de vue logique, comme un chemin ou un ensemble de chemins empruntés par les messages qu'échangent les modules. Il y a deux types de connexion en Flexi

- les connexions de base
- les connexions construites.

On propose de ne considérer que les connexions de base. Ces connexions représentent les trois structures suivantes :

- structure bipoint : elle définit un seul chemin, reliant un port de sortie à un port d'entrée
- structure divergente : elle relie un port de sortie à n ports d'entrée (avec n > 1).
- structure convergente : elle relie n ports de sorties à un port d'entrée (n étant > 1).



4. LA COMMUNICATION

Une communication est définie par un protocole d'échange (réglementation et spécification du type de l'échange), et une sémantique de l'échange (comportement des entités communicantes vis à vis des données échangées). Le couplage d'un protocole et d'une sémantique définit un type de communication. Flexi propose trois types de communication.

4.1 Le type T_1 ou notification

La source envoie le message sans se préoccuper ni de sa réception, ni de son traitement par le destinataire, de la même façon, la réception d'un message se fait sans se préoccuper par l'émission. Les primitives réalisant ce type de communication sont les suivantes :

Envoi :

NSEND < idf message > FROM < idf port de sortie >

Réception :

NRECEIVE < idf message > FROM < idf port d'entrée >.

4.2 Le type T_2 (ou commande)

La source envoie un message tout en s'assurant de sa réception du côté du destinataire, si le message est arrivé la communication est complète, sinon, elle est incomplète. La réception pour ce type de communication est bloquante. Les primitives réalisant ce type de communication sont les suivantes :

Envoi :

BSEND < idf message) FROM < idf port de sortie >

Réception :

BRECEIVE < idf message > ON < idf port d'entrée >

4.3 Le type T_3 (ou service)

Le destinataire, à la réception d'un message exécute le traitement associé et renvoie systématiquement une réponse consécutive à ce traitement. Une communication de ce type commence à l'envoi du message et se termine à la réception de la réponse par la source. Les primitives réalisant ce type de communication sont les suivantes :

Envoi :

SSEND < message 1 > **FROM** < Port de service > **RESPONSE** < message 2 >

Réception :

SRECEIVE < message 1 > **ON** < Port service >
DO
traitement
END
REPLY < message 2 >

Les types T_2 et T_3 peuvent être utilisés avec des délais de garde :

BSEND < idf message > **FROM** < idf port sortie > [**TIMEOUT** (P)]

BRECEIVE < idf message > **ON** < idf port d'entrée > [**TIMEOUT** (P)]

SSEND < idf message > **FROM** < port service > **RESPONSE** < idf message > [**TIMEOUT** (P)]

SRECEIVE < idf message > **ON** < Port service > [**TIMEOUT** (P)].

La clause **TIMEOUT** est optionnelle, P désigne la durée d'attente maximale d'un acquittement. L'absence de cette clause peut provoquer une attente infinie.

Remarque :

On propose de rajouter à Flexi la possibilité d'envoyer des messages avec une certaine priorité. Les primitives d'envoi auront la forme suivante :

NSEND < idf message, priorité > **FROM** < idf port sortie >

BSEND < idf message, priorité > **FROM** < idf port sortie >

SSEND < idf message, priorité > **FROM** < port service >

Le paramètre priorité est optionnel.

5. REPARTITION DE L'APPLICATION

Tout au long de la description d'une application, FLEXI incite à faire abstraction du réseau de processeurs, qui va la supporter. Ainsi, une fois l'application entièrement spécifiée, l'analyse des contraintes économiques, physiques et géographiques, du procédé qu'elle commande, détermine la liste d'implantation de ses différents modules sur les machines du réseau.

Il est nécessaire d'indiquer au système cette répartition. Flexi en fait un outil à part entière, dans l'étape de programmation de l'application.

-=-=-=-=-

ANNEXE 3

STATION DE TRANSPORT ST VERSION 2 DE CYCLADE

Nous présentons dans cette annexe, les services offerts par la station de transport ST2 de cyclades au moyen de l'ensemble des primitives constituant sa description externe. Pour plus de détail, se référer à [MAM 82].

1. Déclaration d'un contexte de porte ou de flot

Toute entité désirant accéder à l'interface de transport par l'intermédiaire d'une porte ou d'un flot doit obligatoirement disposer d'un contexte de porte ou de flot.

Syntaxe :

FCB (nomPL, nomSTDIS, nomPDIS, NBREANT, CONTX)

où :

nomPL est le nom de la porte locale
nom STDIS est le nom de la station distante
NBREANT est le nombre maximum d'anticipations
CONTX est la variable qui contiendra le contexte alloué par la station.

2. Ouverture d'une porte ou d'un flot

Une entité qui dispose d'un contexte de porte ou de flot, peut demander à l'interface de transport l'ouverture de la porte ou du flot.

Syntaxe :

DOPEN (CNTX, INLMAX, OUTLMAX, EVTCONNECT, CONNECTFILE)

où

CNTX est le contexte de la porte du flot à ouvrir
INLMAX est la longueur maximale des lettres en réception
OUTLMAX est la longueur maximale des lettres en émission
EVTCONNECT est un événement à signaler après ouverture de la porte du flot

CONNECTFILE est la file contenant le contexte de la porte ou du flot ouvert par la station de transport.

3. Fermeture d'une porte ou d'un flot

Lorsqu'une entité termine son travail sur une porte ou un flot, elle le signale à l'interface de transport.

Syntaxe :

DCLOSE (CNTX, EVTDISCN, LISTDCONNECT)

où

CNTX est le contexte de la porte ou du flot à fermer
LISTDCONNECT est la file où sera communiqué le contexte de la porte ou du flot fermé par l'interface de transport.
EVTDISCN est un élément à signaler après la fermeture d'une porte.

4. Envoi de lettres

Après l'ouverture d'un flot, une entité peut envoyer des lettres sur ce flot.

Syntaxe :

DSEND (CNTX, LGLT, TEXT)

où : CNTX est le contexte du flot sur lequel on veut émettre
LGLT est la longueur de la lettre à émettre
TEXT est le texte de la lettre à émettre.

5. Retrait des lettres

Après l'ouverture d'un flot, une entité peut recevoir des lettres, en les demandant à l'interface de transport.

Syntaxe :

où : DRECLT (CNTX, LGLT, VAR-RES, EVT)

CNTX est le contexte de la porte ou du flot sur lequel on veut recevoir
LGLT est la longueur de la lettre à recevoir
VAR-RES est la zone qui contiendra cette lettre
EVT est l'événement à signaler, si la liste des lettres à recevoir sur CNTX n'est pas vide.

-=-=-=-=-=-

Bibliographie

[BLO 83] BERTOURNE Cl., FILALI M, LAGARDE J.M., OLIVIER G., PADIOU G., SAYAN A.

Méthode de programmation structurée des sytèmes répartis.
ENSICA - ENSEEIHT - TOULOUSE.

[BOU 83] BOUDEBOUS Z.

Evaluation de la modularité.
Rapport de DEA - Université de NANCY I - Septembre 84.

[BMV 81] BOUSSINOT F., MEYIA F., VAPNE J.

Un langage de description de système temps réel.
THOMSON CSF, Laboratoire central de recherche PARIS.

[CAL 82] CALVEZ J.P., GUILLEMIN T.

Un système d'aide à la conception pour les applications de commande en temps réel.

[CAL 82] CALVEZ J.P.

Une méthodologie de conception des systèmes multimicro-ordinateur pour les applications de commande en temps réel.
Thèse de Docteur ès-Sciences - Université de Nantes Novembre 1982.

[BRI 78] BRINCH H.

Distributed processes, on concurrent programming concepts.
CACM 21.11.1978 p. 934 - 941.

[CCI 80] Définition du Langage CHILL.
Commission d'étude XI. CITT - Août 1982.

[COR 81] **CORNAFION** - Nom collectif pour ANDRE F., BANINO J.S., BETOURNE C., FERRIE J., HERMAN D., KRAKOVIAK S., MAZARE G., MOSSIERE J., ROUSSET DE PINAT, SEGUIN J., VERJUS J.P.

Systèmes informatiques répartis. Concepts et techniques.
Editions DUNOD 1981.

[CRO 76] **CROCUS** - nom collectif pour BALLINO J., BETOURNE C., BRIAT J.J., CANET B., CLEEMANN E., DERNIAME J.C., FERRI Y., KAISER C., KRAKOVIAK S., MOSSIERE J., VERJUS J.P.
Systèmes d'exploitation des ordinateurs.
Editions DUNOD (BORDAS PARIS 1975).

[DER 79] DERNIAME J.C., FINANCE J.P.
Spécification, utilisation et réalisation.
Ecole d'été de l'AFCET - Monastir 1979.

[DER 81] DERNIAME J.C.
"Cours de DEA - option Génie Logiciel"
Université de NANCY I - Année 1980 - 1981.

[DEZ 83a] DERNIAME J.C., ZAKARI A.
Spécification d'application de conduite d'un atelier flexible.
Convention Informatique Latine (CIL) BARCELONE 6 au 9 Juin 1983.

[DEZ 83 b] DERNIAME J.C., ZAKARI A.
Langage de conception pour des applications réparties de conduite de procédés.
Journées BIGRE 83, LE CAP D'AGDE.

[EFA 82] EL FAZZIKI A.
Description du parallèlisme en LTR-V3.
Rapport de DEA - Université de NANCY I, Septembre 1982.

[GAL 78] GALINIER M., CHERBONNEAU B., DELACROIX M.
Une méthode de conception et de développement de programmes.
Rapport final de la convention 77006 - Toulouse Mai 1978.

[GRA 82] GREGOR V., BOCHMANN and M. RAYNAL
Structured Specification of Communicating Systems.
IRISA RENNES - Université de Montréal - Janvier 1982.

[ICH 79] ICHBIAH J. ETAL
Rational for Design of the ADA programming Language
SIGPLAN notices Vol. 14, n° 16 (June 79).

[IGL 82] Introduction à SADT.
IGL France - 1982.

[LIS 79] LISKOV B.
Primitives for distributed computing.
Proc. 7th ACM SIGOPS SYMP. on operating syst. principles. 12/1979.

[LIS 82] LISKOV B., SCHEIFFER R.
Guardians and Actions : Linguistic support for robust distributed Programs.
POPL Janvier 1982.

[LON 83] LONCHAMP J.
Structuration Logique en terme d'agents communicants des applications réparties en conduite de commande/contrôle de processus.
Journée BIGRE 1983, LE CAP D'AGDE.

[LON 82] LONCHAMP J.
Spécification et validation de l'architecture globale des applications temps réel réparties.
Rapport interne CRIN - Novembre 1982.

[LOR 79] **LORRAIN** - Nom collectif pour DERNIAME J.C., MEYER D., SCHAFF R., THOMESSE J.P., VIAULT D.
Réseaux téléinformatiques.
Editions HACHETTE (1979).

[MAF 77] FELICIA M. et ANDRé D.
Etude comparative des principaux langages temps réel.
Institut de Programmation de l'Université Pierre et Marie Curie (PARIS VI) Septembre 1977.

[MAM 82] MAMMERI Z.

Etude de la spécification interne et l'implantation d'applications réparties en conduite de procédé industriel et étude de la réalisation d'un noyau de communication.

Rapport DEA - Université de Nancy I - 1982.

[LAD 82] LADET P.

Contribution à l'étude des systèmes informatiques répartis pour la commande des procédés industriels.

Thèse de Docteur ès-science 1982 - INPG.

[NGU 82] NGUYEN VAN LU

Un langage modulaire de modélisation de systèmes concurrent.

[PAR 72] PARNAS D.L.

A Technique for software modules specification with examples.

Com. ACM - May 1972.

[PAR 77] PARNAS D.L.

Use of Abstract interfaces in the development of Software Embeded Computer Systems.

Dept of Computer Sciences - Univ. of north Cordins at diapol HIL.NAVAL Research Laboratory WASHINGTON - June 1977.

[PAR 84] PARAYRE M.

LTR-V3 - Manuel officiel de référence.

DGA Janvier 1984.

[PER 83] PERRIN G.R., DERNIAME J.C., THOMESSE J.P.

Caractéristiques de la Communication pour la conception de systèmes répartis.

CIL BARCELONE - Juin 1983.

[JAC 78] JACKSON M.A.

Information systems : modelling, sequencing and transformation.

3rd Int. Conf. on Soft engenering Atlanta - Mai, 1978.

[KRA 80] LISTER, MAGEE, SLOMAN, KRAMER.

Distributed process control system programming and configuration.

[KRA 78] KRAMER, CUNINGHAM

Towards a notation for the fonctionnal design of distributed processing systems.

Proc. of IEEE work conf. on parall proc.

[OUZ 82a) OUERGHI M.S., ZAKARI A.

Un exemple de système communicant.

Rapport interne CRIN n° 82 - R - 032.

[OUZ 82b) OUERGHI M.S., ZAKARI A.

Vers une bonne expression de la communication dans un milieu parallèle.

Rapport interne CRIN n° 82-R-61.

[OUE 84] OUERGHI M.S.

Sémantique algébrique d'un langage de programmation supportant le concept de processus communicants.

Doctorat de 3ème cycle en informatique - NANCY I. MARS 1984.

[PTH 82] PERRIN G.R., THOMESSE J.P.

Cours de DEA - Option Parallélisme.

Université de NANCY I - 1982.

[RAY 81] RAYNALD

Contribution à l'étude de la coopération entre processus dans les langages et les systèmes informatiques.

Doctorat ès-Sciences - Université de RENNES Avril 1981.

[SOK 80] SOK S.

Evolutivité du Logiciel.

Thèse de 3ème cycle - Université de NANCY I - Septembre 1980.

[ZAK 84] ZAKARI A.

FLEXI : Langage de conception d'application de conduite de procédés industriels.

Thèse de 3ème cycle en Informatique - NANCY I. Mars 1984.

[THO 80] THOMESSE J.P.

SYGARE : Une structuration pour la conception d'application en temps réel et réparties.

Thèse de docteur ès-Sciences INPL 1980.

[TAN 85] DERNIAME J.C., TANKOANO J.

Démarche de structuration logique des systèmes informatiques répartis pour la commande des procédés industriels dans les ateliers flexible : une approche par les objets.

CRIN - Université de NANCY I.

[RAJ 78] VACLAV RAJLICH

Problems on module interconnection language.

Research Institute for mathematical machines Lorétanské (Zechoslovakia). 1978.

[SAS 83] LISSANDRE M., LAGIER P., SKALLI A.

SAS un système d'assistance aux spécifications.

IGL PARIS France - Septembre 1983.

[ZEL 78] MARVIN V. ZELKOWITZ

Perspectives on software engineering.

Institute for computer sciences and technology National Bureau of Standards. WASHINGTHON D.C. 20234.

[BEN 84] BEN MAIZA M.

Le concept d'évènements dans la spécification et la programmation d'applications temps réel.

Thèse de Docteur Ingénieur - Université de NANCY I.

[SKA 84] SKALI J.

Integration des concepts de FLEXI à LTR-V3.

Rapport de DESS - Université de NANCY I - CRIN.

NOM DE L'ETUDIANT : EL FAZZIKI Abdelaziz

NATURE DE LA THESE : Doctorat 3ème cycle en Informatique

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 13 FEV. 1985 n° 304

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. MAINARD

RESUME

LE THÈME PRINCIPAL ABORDÉ DANS CE TRAVAIL EST LA MISE AU POINT D'UNE MÉTHODE DE STRUCTURATION LOGIQUE DES APPLICATIONS DE CONTRÔLE/COMMANDE DE PROCÉDÉS INDUSTRIELS.

L'IDÉE DE BASE EST : PARTANT D'UNE SPÉCIFICATION FONCTIONNELLE DÉTAILLÉE DU PROCÉDÉ INDUSTRIEL, CONTENANT UNE DESCRIPTION DES DIFFÉRENTS ÉLÉMENTS LE CARACTÉRISANT (VARIABLES, MACROACTIONS, ACTIONS, STRUCTURE D'UTILISATION), ESSAYER D'ABOUTIR À UNE STRUCTURATION DE L'APPLICATION EN UN ENSEMBLE DE MODULES COMMUNICANTS, AYANT UNE FORTE COHÉSION ET UN FAIBLE COUPLAGE ENTRE EUX. CETTE STRUCTURATION EST RÉALISÉE À PARTIR D'UN CERTAIN NOMBRE DE RÈGLES QUI SERONT APPLIQUÉES DANS LEUR ORDRE DE DÉFINITION, ET PERMETTANT DE REGROUPER DANS UN MODULE DES ÉLÉMENTS FORTEMENT LIÉS, ET DANS DEUX MODULES DISTINCTS DES ÉLÉMENTS QUI N'ONT PAS (OU PEU) DE RELATIONS ENTRE EUX.

LE SECOND THÈME ABORDÉ ICI, EST LA RÉALISATION D'UN LANGAGE DE PROGRAMMATION DES APPLICATIONS DE CONTRÔLE/COMMANDE DE PROCÉDÉS INDUSTRIELS. CE LANGAGE DE PROGRAMMATION EST LA COMPOSITION DE DEUX LANGAGES : FLEXI ET LTR-V3. IL S'AGIT D'ÉTENDRE LTR-V3 POUR L'ADAPTER À DES APPLICATIONS RÉPARTIES EN LUI RAJOUTANT LES NOUVEAUX CONCEPTS DE FLEXI. CETTE EXTENSION EST POSSIBLE GRÂCE À LA RÉALISATION D'UN PRÉPROCESSEUR ET D'UN NOYAU DE COMMUNICATION. ELLE REPREND LES TRAVAUX SUR FLEXI ET LES INTÈGRE DANS UNE MÉTHODE DE CONCEPTION.

MOTS CLÉS : APPLICATION DE CONTRÔLE/COMMANDE DE PROCÉDÉS INDUSTRIELS, VARIABLES, MACROACTION, ACTIONS, STRUCTURES D'UTILISATION, COHÉSION, COUPLAGE, NOYAU DE COMMUNICATION, PRÉPROCESSEUR.