LABORATOIRE D'INFORMATIQUE THéORIQUE
& APPLICATIONS DE MARSEILLE
L.I.T.A.M
Faculté des Sciences Economiques
Puvis de Chavannes
UNIVERSITé AIX-MARSEILLE II

ISSN 0291 - 5413

INFORMATIQUE FONDAMENTALE ET APPLICATIONS

SOMMAIRE

DECEMBRE 1988

Adresse postale : FACULTé DES SCIENCES ECONOMIQUES
LITAM
rue Puvis de Chavannes 13 001 MARSEILLE
91 90 13 20 P 420 et 421

EDITORIAL

E. Blanco

HIVER ET PRINTEMPS

Un vent humide et froid dans les arbres tout nus, la lumière glacée qui tarde à se lever et se reflète triste sur les flaques grisâtres. Les nuages violets rougissent à l'horizon du soleil qui se couche, vision de fin du monde.

D'un monde frissonnant qui s'endort lentement.

Et puis les jours grandissent, les soirs se font plus longs, le froid se fait plus dur, mais le soleil plus chaud. Parfois un peu de neige étouffe tous les bruits.

C'est le printemps qui couve. C'est au temps le plus sombre qu'on perçoit le bourgeon tout prêt à reverdir.

Mais combien de rafales encore traverseront les plaines, combien de matins blancs à l'onglée de mes doigts, de nuages pressés sertis d'un bleu violent, de ressacs embrumés et d'écume, avant que naisse la tiédeur.

Longtemps encore le vent soufflera froid dans les halliers de la politique, une sombre lumière qui ne porte pas d'ombre fera luire les faces blafardes de la responsabilité.. Les forêts de barreaux remplaceront les bois dans la terre glaciale.

Les pitres funambules, grimaçants agités, à l'abri sur leur fil.

Entre une affiche électorale et le pain quotidien, leur fil est surtendu. Des meutes de loups et de chiens enragés derrière les affiches, les faces de méduse de tous les chats fourrés, inquiets pour leur fourrure.

Et puis le ras-le-bol de racket incessant derrière la baguette.

Le fou du Roi s'élance et puis plastronne sur son fil qui gémit. Soucieux de son affiche avant tout autre chose. Le spectateur blasé s'écarte du spectacle. Une bise de Carnaval poursuit les faces de Carême.

Aux mauvais funambules succèdent les jongleurs et leurs tableaux de chiffres. En tête de l'affiche surgissent les comptables.

Le saltimbanque avec son bateleur jadis nous promettait la lune. Au grand banquet final nous étions conviés ... pour payer l'addition. Non contents de goinfrer ils cassaient la vaisselle. Qu'il fallait remplacer. Plus personne ne croit à tous ces beaux parleurs.

Qui les remplacera ?

Dans la sombre lueur des hivers d'après guerre surgissent lentement les Hommes-des-Dossiers. Vêtus couleur muraille ils gèrent les pantins. On ne les voit jamais, planqués derrière leurs machines ; pour eux et à jamais, le monde n'est qu'un chiffre, les gens des matricules, la Vie n'est plus qu'un compte. Vos sous sont à l'abri cachés dans un ordinateur, comme un cyprin doré dans une tire-lire.

Après l'hiver stérile des marchands de promesses, le printemps de brouillard des agités du chiffre.

Ami, on veut nous persuader qu'il n'est point de salut hors celui qu'on nous donne en pâture. A la grande foire aux rapaces on peut faire son choix, d'un côté les voilà, auréolés de vertu et ceints de pureté sur des dessous sordides, de l'autre les voici, avec leurs beaux comptables et leurs petits copains, de ci de là un clown inexistant esquisse une horrible grimace...

Ô Printemps, qu'un vent tout frais qui nous apporte le message, sous un soleil nouveau qui nous fait chaud au cœur balaie cette vermine ...

II.2.3. Instructions de répétition :

Un bloc de répétition commence par **REPETER SI « COND »** et s'achève par **FIN_REPET** , l'exécution dépend de la condition « **COND** » évaluée juste avant l'exécution de l'instruction.

II.2.4. Appel de section :

Les sections doivent avoir été déclarées au moment de l'appel. L'appel de section Si s'écrit : **SECT Si : CONDj** ; signifie que si la condition j est vraie alors dérouler la section Si, cela permet d'utiliser la récursivité qui sera contrôlée par la condition qui suit l'appel de section.

II.2.5. déroulement en parallèle :

Deux types d'instructions permettant le déroulement en parallèle de plusieurs sections :

Le parallèle OU " **DEROULER_OU**", et le parallèle ET "**DEROULER_ET**" .

Par exemple soient 3 sections SECTION S1, SECTION S2 et SECTION S3 :

DEROULER_OU : SECT S1:COND i1,SECT S2:COND i2,SECT S3:COND i3 ; Cette instruction déroule les sections qui ont des conditions de valeur "**vrai**", le déroulement se fait en parallèle et on s'arrête lorsque l'une de ces 3 sections s'achève , l'absence de la condition dans l'appel de la section est à considérer comme "**vrai**" .

DEROULER_ET SECT S1, SECT S2, SECT S3 ; Cette instruction déroule ces 3 sections en parallèle et on s'arrête quand les 3 sections s'achèvent .

III. DESCRIPTION DU L.C.P.

III. SYNTAXE DU L.C.P. :

On utilise les règles de BACCHUS pour définir la syntaxe du L.C.P. , les symboles ::= ‹ › / appartiennent au formalisme des règles de BACCHUS et ne font pas partie du langage L.C.P., ils ont les rôles suivants :

::= signifie "se réecrit"

‹ › encadrent un non-terminal

/ sépare deux alternatives .

III.1.1. Programme :

‹programme› ::= ‹entête de programme› ‹bloc› .

‹entête de programme› ::= **PROGRAMME** ‹identificateur›;

‹bloc› ::= ‹entête de bloc› ‹corps de programme ›

‹corps de programme › ::= **DEBUT** ‹suite d'instruction › **FIN.**

‹entête de bloc› ::= ‹partie déclaration des entrées ›
‹partie déclaration des sorties ›
‹partie déclaration des variables locales ›
‹partie déclaration des durées ›
‹partie déclaration des sections ›

III.1.2. Déclarations d' E/S :

* ‹ partie déclaration des entrées › :: = ‹ Entrée simple › ‹ Entrée impulsion›

‹ Entrée simple › :: = ‹ vide › / **ENTREE :** ‹ suite de var. › ;

‹ Entrée impulsion › :: = ‹ vide › / **ENTREE-I :** ‹ suite de var. › ;

II. NOTION DU L.C.P.

La programmation en L.C.P. permet de réaliser les mêmes fonctions qu'en logique câblée mais d'une manière plus souple et surtout, adaptée à la structure de l'ordinateur, en plus il devient facile d'effectuer des modifications, et les algorithmes peuvent être améliorés ou même changés, sans qu'il soit nécessaire de procéder à un remaniement total de la partie hardware, ou même sans être obligé d'y changer quoi que ce soit.

L'intêret du L.C.P. est qu'il posséde la possibilité de construire des algorithmes images de tâches qui se déroulent en paralléle et en nombre a priori quelconque .

Dans le langage L.C.P. existent deux parties :

- **Une partie déclaration** où l'on déclare tous les objets nécessaires.

- **Une partie instruction**, où l'on spécifie les actions à éxécuter sur les objets précédemment décrits .

II.1. PARTIE DECLARATION :

Deux sortes de types de déclarations :

a. Déclarations des variables de type

ENTREE ;
ENTREE-I ;
SORTIE ;
SORTIE-I ;
LOCALE ;
DUREE ;

- VARIABLES D'E/S :

Il s'agit de commandes vers l'extérieur et de réponses aux bons accomplissements des actes commandés.

Chaque variable représente une suite de, au plus 4 bits. Chaque bit est soit une commande pour une variable de sortie, soit une information indiquant un état de machine commandée, renvoyée à l'ordinateur.

Deux types d'E/S : E/S stable ou E/S à impulsion .

= VARIABLES LOCALES :

Les variables locales utilisées pour mémoriser des données binaires servent aussi à connaître l'état de la commande envoyée sur une ou plusieurs machines.

= VARIABLES DE DUREES :

Les variables de durées servent à mémoriser des données exprimées en heures, minutes et secondes.

b. Déclarations des sections :

Les sections sont des éléments de programmes destinés à être utilisés plusieurs fois ou inclus dans des boucles .

II.2. PARTIE INSTRUCTION :

II.2.1. Instructions de condition :

Si on a déclaré **ENTREE B(3)** ; cela signifie que B représente 3 bits, une condition s'écrit par exemple :

COND B(2)=0 ; la condition porte sur le deuxième bit de B; s'il est égal à 1 alors on se bloque sur la condition jusqu'à ce qu'elle change, sinon on passe à l'instruction suivante.

II.2.2. Instructions d'affectation :

On peut imposer un jeu de valeurs à une variable : A := 101 ; ce qui signifie que trois organes sont connectés pour être commandés par ces 3 bits : le premier et le dernier sont mis au activité, et le second est au repos .

Si A,B,C ont la même dimension, alors on peut écrire par exemple :

A := A+B, A := B.C,

Si A = 101, B = 011 et C = 110 alors : A + B = 111, B.C = 010 .

On peut écrire également : **A := □B** ce qui donnerait A = 100 .

LANGAGE DE CONDUITE DES PROCESSUS

PRESENTATION & APPLICATION

S. H I L A L A

C . R . Subject Classification informatics :

Résumé :

Notre but est de réaliser un système permettant une implantation générale et aisée d'un système de conduite de processus, qui sera adaptée à la structure de l'ordinateur.

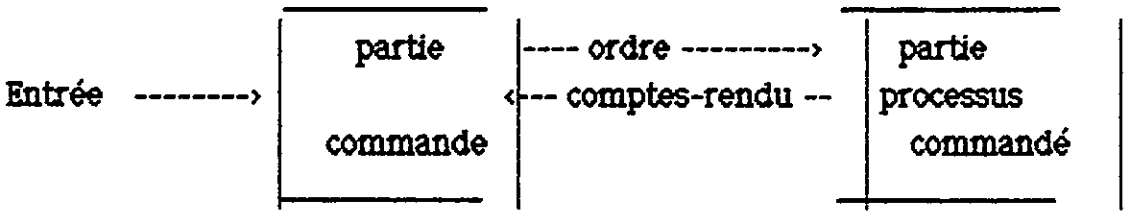
L'objet de cet article sera la définition d'un **Langage de Conduite de Processus "L.C.P"**, répondant au besoin de notre système.

Une illustration de ce langage à travers une application à l'automatisation d'une station de mélange.

I. INTRODUCTION

L'ensemble constitué par le matériel (hardware) et le logiciel (software) d'un système de conduite de processus est relié au monde extérieur par des organes d'accès qui lui permettent de communiquer ou d'interroger des dispositifs physiques qu'il a à commander ou à contrôler.

Pour un problème d'automatisation, le système se décompose en deux parties : la partie **"commande"** regroupant les fonctions de commande et la partie **"processus commandé"** représentant le procédé physique à automatiser.



Dans un premier temps, il s'agit de représenter clairement les relations entre ces deux parties, c'est-à-dire de définir les fonctions désirées, l'enchaînement de ces fonctions, les modes de marche, ...

Le cycle de travail est une suite d'actions qui ne pourront être obtenues que si la partie commande émet des ordres convenables au moment voulu, ces moments voulus seront déterminés d'après les informations provenant de la partie commandée.

Plusieurs méthodes permettent la matérialisation d'une structure de commande adaptée à une application précise, telles que la logique cablée programmée, les automates programmables, etc . . . utilisant le langage graphique de description **(GRAFCET)** ou **les réseaux de pétri** .

L'idée retenue est de réaliser un système "matériel-logiciel" permettant une implantation générale et aisée du système de conduite de processus.

Notre système se compose d'un langage L.C.P. "Langage de Conduite des Processus", et une carte de développement articulé autour de l'unité centrale **"UC/LITAM"**, équipée d'un interpréteur de ce langage

‹ suite de var. › :: = ‹ type de var. › / ‹ Suite de var. ›, ‹ type de var. ›

‹ type de var. › :: = ‹ identificateur › (‹ nb bits ›)

‹ identificateur › :: = ‹ lettre › ‹ suite l. ou ch. ›

‹ suite l. ou ch. › :: = ‹ lettre › ‹ suite l. ou ch. › / ‹ chiffre ›

‹ suite l. ou ch. › / ‹sslg›‹ suite l. ou ch. ›/‹ vide ›

‹ lettre › :: = A/B/.../Z/a/b/.../z

‹ chiffre › :: = 0/1/2 . . . /9

‹ nb. bits › :: = 1/2/3/4

** ‹ partie déclaration des sorties › :: = ‹ Sortie simple › ‹ Sortie impulsion ›

‹ Sortie simple › :: = ‹ vide › / **SORTIE** ‹ suite de var. › ;

‹ Sortie impulsion › :: = ‹ vide › / **SORTIE-I** ‹ suite de var. › ;

III. 1.3 Déclaration des variables locales :

‹ Partie déclaration des variables locales › :: = ‹ vide › / **LOCALE :** ‹ suite de var. › ;

III. 1.4 Déclaration des durées :

* ‹ partie déclaration des durées › :: = ‹ vide › / **DUREE :** ‹ suite de D › ;

‹ suite de D › :: = ‹identificateur› / ‹ suite de D ›, ‹identificateur ›

III 1.5. Déclaration des sections :

‹ partie déclaration des sections › :: = ‹ vide › / ‹ suite de section ›

‹ suite de section › :: = ‹ bloc de section › / ‹ suite de section › ‹ bloc de section ›

‹ bloc de section › :: = ‹ entête de section › ‹ instruction section ›

‹ entête de section › :: = **SECTION** ‹ identificateur › ;

‹ instruction section › :: = **DEB-SECT** ‹ suite d'instruction › **FIN-SECT ;**

III 1.6. Les instructions :

‹suite d'instruction › ::=‹instruction›/‹ instruction ›‹ suite d'instruction›

‹ instruction › :: = ‹ instruction d'affectations › ; /

‹ inst. conditionnelle › ; /‹ instruction composée › ;

III. 1.6.1. Les instructions d'affectation :

‹ instruction d'affectation › :: = ‹ affectation simple › / ‹ affectation de temps ›

‹ affectation simple › :: = ‹ var. composée › : = ‹ expression ›

‹ var. composée › :: = ‹ variable › / ‹ variable ›, ‹ variable ›

‹ variable › :: = ‹ identificateur › / ‹ identificateur › (‹ N° bit ›)

‹ N° bit › ::= 1/2/3/4/

‹ expression › :: = ‹ exp. simple › / ‹ exp. simple › ‹ op. › ‹ exp. simple ›

/ □‹ exp. simple ›

‹ exp. simple › :: = ‹ valeur binaire › / ‹ variable ›

‹ valeur binaire › :: = ‹ bit › / ‹ bit › ‹ bit › / ‹ bit ›‹ bit › ‹ bit ›/

‹ bit › ‹ bit › ‹ bit ›‹ bit ›/

‹ bit › :: = 0/1

‹ op › :: = + /.

‹ affectation de temps › :: = ‹ var. de D › : = ‹ exp. de D › / ‹ expression-T ›

‹ expression-T › ::= ‹ exp. de T › ‹ op. T › ‹ exp. de D ›

‹ exp. de T › ::= ‹var. de T› /‹valeur de D›

‹ var. de T › :: = ‹ var. de D › / **HEURE**

HEURE ::= fonction qui lit l'horloge de temps réel du système

‹ var. de D › :: = ‹ identificateur ›

‹ valeur de D › :: = ‹ valeur H › **h** / ‹ valeur H › **h** ‹ suite-D › /

‹ suite-D ›

‹ suite-D › :: = ‹ val. m.s › **m** / ‹ val. m.s › **m** ‹ val. m.s. › **s** /

‹ val. m.s. › **s**

‹ valeur H › :: = 00/01/.../24

‹ valeur m.s. › :: = 00/01/.../60

‹op. T› ::= +/-

III. 1.6.2. Instruction conditionnelle :

‹ inst. conditionnelle › :: = **COND** ‹ bloc condition-S › /

COND ‹ bloc conditions - T ›

‹ bloc condition - S › :: = ‹ variable › = ‹ expression ›

‹ bloc condition - T › :: = ‹ var. de T › = ‹ expression-T ›

III. 1.7. Les instructions composées :

‹ instruction composée › :: = ‹ inst. appel de section › ; /

‹ inst. répétitive › ; /

‹ inst. de déroulement › ;

‹ inst. appel de section › :: = ‹ appel de section › /

‹ appel de section › : ‹ inst. conditionnelle ›

‹ appel de section › :: = **SECT** ‹ identificateur ›

‹ inst. répétitive › ::= **REPETER SI** ‹ inst. conditionnelle ›‹bloc répétiton›

‹ bloc répétition › :: = ‹ intruction composée › **FIN_REPET**

‹ inst. de déroulement › :: = ‹ dérouler › ‹ suite de section ›

‹ dérouler › :: = **DEROULER_OU:** ‹suite_section_OU›/

DEROULER_ET: ‹suite_section_ET›

‹ suite_section_ET › :: = ‹ appel de section › /

‹ appel de section › , ‹ suite_section_ET ›

‹ suite_section_OU › :: = ‹ inst. appel de section › /

‹ inst. appel de section › , ‹ suite_section_OU ›

III. 2 SEMANTIQUE DU L.C.P.

III. 2.1 Déclarations

III 2.1.1. Déclaration d'E/S :

Une variable d'E/S se déclare comme suit :

identificateur (nbre de bits) ;

Le "**nbre de bits**" indique le nombre de bits de la variable qui sera dans notre étude <= 4 bits, pour des raisons pratiques liées au matériel. Cette limitation n'a rien de fondamental .

L'utilisateur connecte les organes commandés par groupe de 4 au plus, qui sont supportés par des variables de sorties, il en est de même pour les variables d'entrées.

Les variables d'**ENTREE-I**, sont appelées Entrées à Impulsion ; celles-ci sont mémorisées dans un latch lorsqu'elles se manifestent.

Au cas ou l'on envoie une sortie à impulsion, le système envoie d'abord un 1 (ou 0) sur le bit de sortie correspondant, puis il le remet à 0 (ou 1).

III 2.1.2. Déclaration des variables locales :

Les variables locales servent à mémoriser des données. Chaque variable locale sera présentée sur 4 bits au plus, et le nombre maximum des variables locales à priori quelconque.

La déclaration d'une variable locale se fait comme la déclaration d'une variable d'E/S.

III 2.1.3. Déclaration des durées :

Une variable de durée se déclare de la manière suivante :

identificateur ;

la nature de durée est exprimé en : heure, minute et seconde.

On dispose d'une fonction **HEURE** qui lit l'horloge de temps réel du système exprimée en heures, minutes et secondes après minuit, sous la forme HH MM SS.

III 2.1.4. Déclaration des sections :

Les sections sont des sortes de procédures qui n'admettent que des variables extérieurs.

Une section se déclare de la manière suivante :

SECTION ‹ identificateur › ;

DEB-SECT

‹ suite d'instruction ›

FIN-SECT ;

III 2.2. Les instructions:

III 2.2.1. Instructions d'affectations :

On peut imposer un jeu de valeurs binaires à une variables de sortie **Si:=101** ; ce qui signifie que trois organes sont connectés pour être commandés par ces 3 bits : le premier organe et le dernier sont en activités, et le second est mis au repos (si on admet qu'un organe est actif au "1" logique), la valeur binaire **101** ne sera pas mémorisée dans **Si**.

Les instructions de type **Li := exp.** (où Li est une variable locale), signifie que le résultat de l'expression "**exp.**" est mémorisé dans **Li**.

Les instructions de type **Li(k), Sj(1) := exp** (où Li (k) indique le bit n° k dans la variable locale Li, Sj(1) le bit n° 1 dans la variable de sortie Sj), signifient que le résultat de l'expression "exp.", qui est sur un seul bit, est envoyé sur le bit j de la sortie Sj et mémorisé dans le bit k de la variable locale Li.

Si dans l'expression "exp." il y a une varaible d'entrée ou une variable de sortie ou une variable d'entrée à impulsion, Ek, alors le système lit d'abord l'entrée Ek (ou la sortie), puis il effectue l'opération correspondante.

Les instructions du type Si, Sj := exp. sont acceptées.

Dans une instruction d'affectation, les variables doivent être de même dimension.

Les instructions de temps de type : **Di := HEURE +/- Ti (type de temps)**

(où : - **Di** est une variable de durée,

- **Ti** est une valeur de temps exprimé par le "type de temps " qui est H, M ou S) signifie que le résultat de l'opération **HEURE +/- Ti** est mémorisé dans **Di** .

L'initialisation d'une durée **Dj** peut se faire à l'aide de l'instruction de type : **Dj := HEURE** .

III. 2.2.2. Instructions conditionnelles :

L'instruction conditionnelle s'écrit par exemple :

COND Ei (j) = exp. ;

Si on a déclaré Ei comme variable d'entrée de dimension k, où k □ j, la condition porte sur le j ème bit de Ei :

Si le j ème bit de Ei est égal à "exp." **alors** on passe à l'instruction suivante "c'est-à-dire la condition est vraie" ; **sinon** on se bloque sur la condition.

On peut exprimer des conditions sur les durées "Di", où sur l'heure **HEURE** ; par exemple :

COND Di = tj, signifie qu'à partie du moment où Di égale le temps tj, la condition est vraie.

COND HEURE = h, la condition est vrai si l'heure actuelle **HEURE** égale à h, où h exprimé en heures, minutes et secondes.

On peut imposer des conditions de types :

* **COND Di = Dj +/- Dk;**

* **COND Di = Dj +/- Ti (type de temps);**

* **COND HEURE = Dj +/- Dk;**

* **COND Di = Dj +/- Ti (type de temps);**

III 2.2.3. Instructions composées :

III 2.2.3.1. Appel de section :

L'appel de section se fait comme suit :

* **SECT < ident. >** ; signifie que la section < ident. > est appelée.

* **SECT < ident. > : < condition >** ;

L'insertion de la section < ident. > ne sera effective que si la condition **«condition »** est vraie, dans le cas contraire on passe à l'instruction suivante. Cela nous permet d'utiliser la **recursivité** qui ne présente aucune complication dans le cas d'existance de la condition, car l'arrêt de la **recursivité** est contrôlé par la condition **«condition »**, dans le cas contraire (absence de la condition), la recursivité est interdite.

Deux formes de recursivité existent en L.C.P. :

- **La recursivité intrinsèque** survient lorsqu'une section contient une instruction qui fait appel à cette section.

- **La recursivité mutuelle** survient lorsqu'une section A contient une instruction qui appelle la section B, cependant que B contient un appel à A.

La récursivité mutuelle peut d'ailleurs impliquer davantage que deux sections.

III. 2.2.3.2. Instructions répétitive :

Un bloc de répétition commence par **REPETER**, et s'achève par **FIN_REPET.**

On peut avoir plusieurs blocs répétitifs amboîtés.

On exprimer une condition sur **REPETER** auquel cas on ne rentre pas dans le bloc si elle n'est pas vérifiée.

III 2.2.3.3. Instructions de déroulement :

On peut construire une instruction qui déroule en parallèle plusieurs sections, à priori quelconque, et là on a deux cas :

- **DEROULER_ET** : dérouler en parallèle toutes les sections jusqu'à la fin des sections déroulées; c'est-à-dire que la section qui s'achève attend les autres.

- **DEROULER_OU** : cette instruction déroule en parallèle les sections qui ont des conditions"vrai", et on arrête le déroulement lorsqu'une section déroulée se termine .

Dans le cas où on veut dérouler les sections qui ont des conditions vraies, et le déroulement s'achève lorsque toutes les sections s'achèvent, cette instruction on peut l'exprimer de la manière suivante :

Si on doit dérouler les sections A1,A2 et A3 avec les conditions COND A1, COND A2 et COND A3 respectivement, cette instruction on peut l'exprimer par la suite des insructions suivantes:

L1 := 0; L2 := 0; L3 := 0;

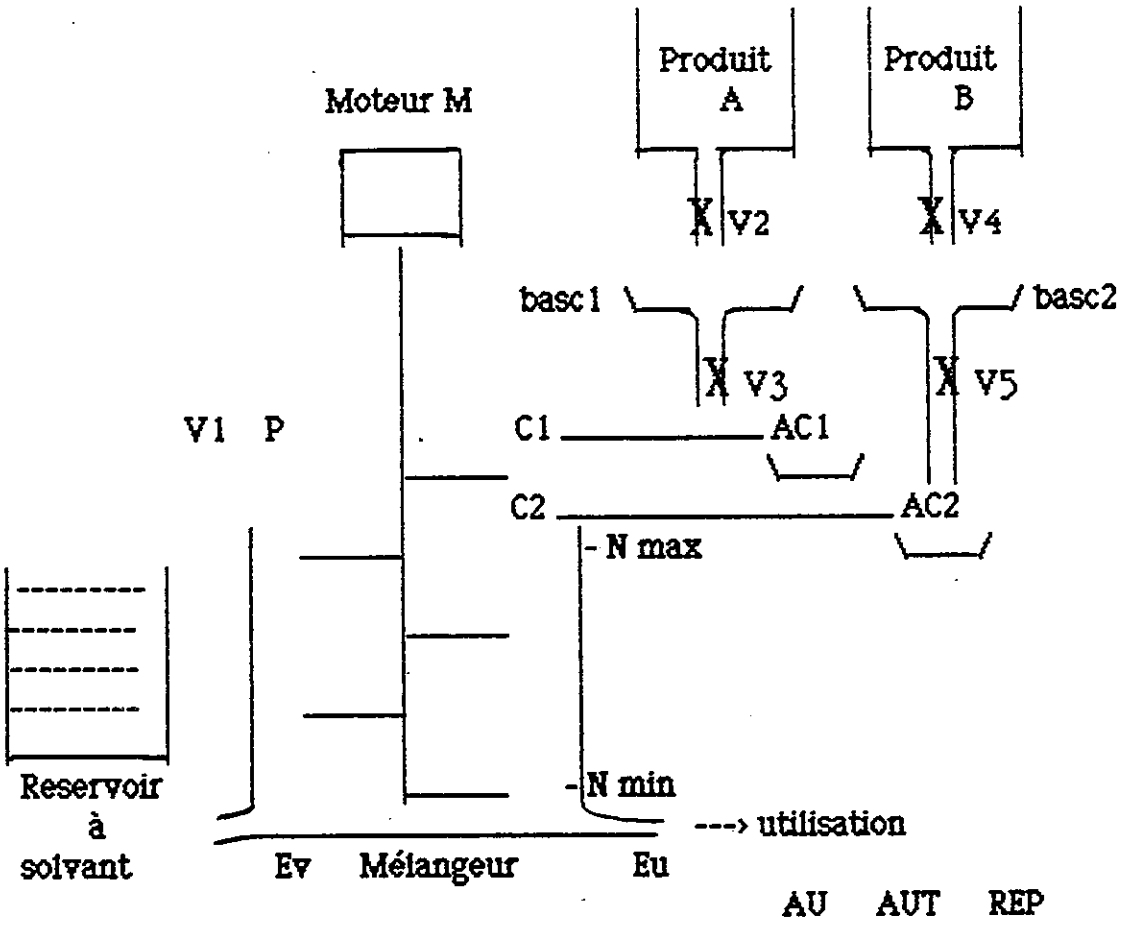
DEROULER_OU : SECT B1:COND A1,SECT B2:COND A2,SECT B3:COND A3;
où :

```
SECTION B1 ;          SECTION B2 ;          SECTION B2 ;
DEB-SECT              DEB-SECT              DEB-SECT
    L1 := 1;              L2 := 1 ;             L3 := 1;
    inst. SECT A1 ;       inst. SECT A2 ;       inst. SECT A3 ;
    L1 = 0 ;              L2 = 0 ;              L3 := 0;
    COND L2 = 0 ;         COND L1 = 0;          COND L1 = 0;
    COND L3 = 0 ;         COND L3 = 0;          COND L2 = 0;
FIN-SECT;             FIN-SECT;             FIN-SECT;
```

Dans le déroulement en parallèle **(DEROULER_ET/DEROULER_OU)**, la récursivité est interdite.

IV. APPLICATION A L'AUTOMATISATION D'UNE STATION DE MELANGE.

Soit à automatiser une station de mélange représentée sur la figure ci-dessus . /4./



Il s'agit de réaliser les opérations suivantes:

- Si le niveau minimum de solution dans le mélangeur est atteint, et à condition d'avoir l'autorisation AUT de démarrer le processus, on ouvre les vannes V1,V2,V4 et l'on met la pompe P en marche.

- Quand une quantité B1 et B2 de produit à dissoudre s'est déversée sur le plateau de la bascule basc 1 et basc 2, on ferme la vanne V2 et V4.

- Lorsque le niveau de remplissage maximum du mélangeur est atteint, on arrête la pompe P et on ferme la vanne V1.

- Les opérations précédentes étant achevées, on met en fonction le moteur M du dispositif de mélange en même temps qu'on ouvre les vannes V3 et V5.

- Après un temps t1 on arrête les convoyeurs C1 et C2 et on ferme les vannes V3 et V5.

- Cela étant fait, après une temporisation d'un temps t2 on arrête le moteur du mélangeur M.

- Après incident et sur l'ordre de reprise REP, le système doit vidanger le mélangeur par la vanne Eu jusqu'à Nmin. (Il mettra d'autre part les convoyeurs en marche arrière AC1 et AC2 pendant le temps t1 de manière à récupérer les restes de produit qui pourraient s'y trouver).

- Pour des raisons de sécurité, on prévoit un arrêt d'urgence AU qui, s'il est reçu pendant les phases de remplissage, arrête simplement l'action en cours qui reprendra sur l'ordre REP.

S'il est émis en période de mélange, il entraîne un arrêt complet, mais de ce fait, rend le produit impropre à l'utilisation.

La procédure de reprise consiste à vidanger le mélangeur et à récupérer les produits sur les convoyeurs comme dans la phase d'initialisation.

Le schéma fonctionnel se trace en distinguant, par analyse du cahier des charges les variables de conduite du processus : M, P, C1, C2, AC1, AC2, V1, V2, V3, V4, V5, Eu et Ev.

Les variables donnant une "mesure" de l'état du processus:

B1, B2, Nmin, Nmax et enfin les variables caractérisant les grandeurs d'entrée du système automatisé : AUT, AU et REP.

```
PROGRAMME STATION_MELANGE ;

    ENTREE :      E1(4);

    ENTREE_I :    EI1(3);

    SORTIE :      S1(4), S2(4), S3(3), S4(2);

    LOCALE :      L1(4), L2(4), L3(3), L4(2), L5(2)

    DUREE :       D1, D2, D3;

    SECTION Remplis_M ;              "Pompe P  + V1"
      DEB_SECT
          S2 , L2 := L2 + 1001;  " Ouverture de la vanne V1 et la Pompe
P"
          COND E1(4) = 1 ;           " < N max  atteint > ? "
          S2 , L2 := L2 . 0110 ; " Fermeture de la vanne V1 et la Pompe
P"
      FIN_SECT;

    SECTION Prod_B1 ;              "Produit B1 + V2"
      DEB_SECT
          S1 , L1 := L1 + 0001 ;       " Ouverture de la vanne V2 "
          COND E1(1) = 1 ;             " SI B1 = 1 ? "
          S1 , L1 := L1 . 1110 ;       " Fermeture de la vanne V2 "
      FIN_SECT;

    SECTION Prod_B2 ;              "Produit B2 + V4"
      DEB_SECT
          S1 , L1 := L1 + 0100 ;       " Ouverture de la vanne V4 "
          COND E1(2) = 1 ;             " SI B2 = 1  ? "
          S1 , L1 := L1 . 1011 ;       " Fermture de la vanne V4 "
      FIN_SECT;

    SECTION Remplissage ;
      DEB_SECT
          DEROULER_ET : SECT Remplis_M, SECT Prod_B1,SECT Prod_B2;
      FIN_SECT;
```

```
SECTION Init ;                    " Phase d'initialisation "
  DEB_SECT
    L5(1) := 1 ;                  " Variable de contrôle pour un
                                      arrêt urgent ? "
    COND EI1(3) = 1;   " Attendre l'ordre de reprise (REP à 1)"
    S2 , L2 := L2 + 0010 ;   "Ouverture de la vanne Eu "
    L4, S4 := 11 ;                " Mettre en marche AC1 et AC2 "
    D1 := HEURE ;                 " Initialisation la durée D1 "
    REPETER SI L5(1) = 0;                 " Attendre un temps
        COND HEURE = D1 + t1 m ;              t1 exprimé
        L5(1) := L5(2) ;                      en minute      "
    FIN_REPET;
    L4 , S4 := 00 ;               " Fermeture AC1 et AC2 "
    COND E1(3) = 1 ;              " Attendre jusqu' au niveau N min "
    S2 , L2 := L2 . 1101 ;        " Fermeture Eu "
  FIN_SECT;

SECTION Arrêt_Init ;" Arrêt pendant la phase d'initialisation "
  DEB_SECT
    REPETER
      COND EI1(1) =1 ;                " AU =?= 1 "
      S4 := 00 ;                    " Fermeture AC1 et AC2 "
      S2(2) := 0 ;                      " fermeture Eu "
      D3 := HEURE ;          "   Initialisation de la durée D3 "
      L5(2) := 0 ;            " Bit de contrôle pour signaler
                                  l'arrêt urgent AU "
      COND EI1(3) =1 ;   " Attendre l'ordre de la reprise REP "
      S4 := L4 ;         " Envoyer l'ancienne valeur de AC1 et AC2"
      S2(2) := L2(2) ;       " Envoyer l'ancienne valeur de Eu"
```

```
        D3 := HEURE - D3 ; " Calculer le temps de l'arrêt urgent qui
        D1 := D1 + D3 ;          s'est produit puis l'ajouter à D1 "
        L5(2) := 1 ;
    FIN_REPET ;

FIN_SECT;
```

SECTION Arrêt_remp; " **Arrêt pendant la phase de remplissage**"

```
DEB_SECT
    REPETER
        COND EI1(1) =1 ;          " AU =?= 1 "
        S1 := L1 . 1010 ;   " Fermeture des vannes V2 et V4 "
        S2 := L2 . 0110 ;        " Fermeture de V1 et P "
        COND EI1(3) =1 ;  " Attendre l'ordre de la reprise REP "
        S1 := L1 ;   " Envoyer l'ancienne valeur de V2, V3, V4 et
V5"

        S2 := L2 ;   " Envoyer l'ancienne valeur de V1, Eu, Ev et P"
    FIN_REPET;
FIN_SECT;
```

SECTION Mélange ; " **Phase du mélange** "

```
DEB_SECT
    D1 := HEURE ;
    L5(2) := 0 ;
    COND L5(2) = 0 ;
    S3 , L3 := 111 ;        " Mettre le moteur M et les convoyeurs
                                    C1 et C2 en marche "
    S1 , L1 := L1 + 1010 ;      " Ouvrire les vannes V3 et V5 "
    COND HEURE = D1 + t1 m ;
    COND L5(2) =0 ;
    S1 , L1 := L1 . 0101 ;     " Fermeture des vannes V3 et V5 "
    COND HEURE = D1 + t2 m ;
    COND L5(2) =0 ;
    S3 , L3 := 000 ;        " Arrêt du moteur M et les convoyeurs
                                    C1 et C2 "
FIN_SECT;
```

```
SECTION Arrêt_Mélange ; " Arrêt pendant la phase du mélange

DEB_SECT
    COND EI 1(1) =1 ;                " AU =?= 1 "
    L5(2) := 1 ;
    S3 , L3 := 000 ;     " Fermeture du moteur M et les convoyeurs
                                              C1 et C2 "
    S1 , L1 := L1 . 0101 ; " Fermeture des vannes V3 et V5 "
    COND EI 1(3) =1 ;     " Attendre l'ordre de la reprise REP "
    S2(3) , L2(3) := 1 ;          " Ouverture de la vanne Ev "
    COND EI(3) = 1 ;          " < le niveau N min est atteint > ? "
    S2(3) , L2(3) := 0 ;          " Fermeture de la vanne Ev "

FIN_SECT;


DEBUT                    " PROGRAMME PRINCIPAL "
    REPETER
        COND EI 1(2) = 1 ; " Attendre l'autorisation de démarrage "
        L1 , L2 := 0000 ;
        L3 := 000 ;
        L4 , L5 := 00 ;
        DEROULER_OU : SECT Init, SECT Arrêt_Init ;
        DEROULER_OU : SECT Remplissage, SECT Arrêt_remp ;
        DEROULER_OU : SECT Mélange, SECT Arrêt_Mélange ;
    FIN_REPET ;

FIN .
```

BIBLIOGRAPHIE :

/1./ BIANCO E.
Etude et formalisation d'une classe de systèmes
Doctorat d'état, Paris 1969

/2./ BIANCO E.
De la machine de turing aux ordinateurs modernes
Birkhauser Verlag 1979

/3./ ISOARDI P.
LE GRAFCET ETUDE ET REFLEXIONS
Mars 1983

/4./ BOSSY J.C., BRARD P., FAUGERE P., MEKLAUD C.
LE GRAFCET, sa pratique et ses applications
1981

/5./ KRAKOWIAK
Principes des système d'exploitation des ordinateurs
DUNOD 1985

/6./ TIBERGHIEN J.
LE GUIDE DE PASCAL SYBEX
1982

/7./ KERNIGHAN B. W., RITCHIE D. M.
LE LANGAGE C Masson
1988

/8./ LE BEUX P.
INTRODUCTION A ADA SYBEX
1982

SYSTèME UNIVERSEL

E. Bianco

C. R. Subject Classification Informatics : C53 C54 D31 D41

Résumé.

Le système est ici décrit dans son fonctionnement en équilibre. En admettant que la machine de distribution prend en compte tous les traitements asynchrones, alors on montre que, disposant de l'outil de description adéquat l'algorithme d'un système aussi complèxe demeure d'une remarquable simplicité.

SYSTEME UNIVERSEL

Je vais essayer de donner une version plus complète du système que j'ai déja décrit partiellement. Dans le déroulement des opérations il y a la phase la plus courante, celle qui correspond à l'attribution d'un quantum de calcul à une tâche parmi l'ensemble de toutes celles qui sont en cours de déroulement. Mais à côté de cette partie essentielle du travail qui représente la partie directement utile, il existe nombre de traitements importants qui absorbent à leur tour une quantité de travail pas toujours négligeable et indirectement utile.

Ainsi, phénomène relativement rare, il y a l'opération d'intégration d'un sous-système. Un ensemble de compilateurs sont construits comme de simples programmes, mais de telle manière qu'ils soient compatibles et cohérents, ce qui exige à la fois un jeu de langages compatibles dans leurs divers aspects, langage de programmation, pour décrire des algorithmes, un langage de configurations, pour structurer les données, et un langage de communication pour traiter globalement les productions issues des deux premiers.

Chaque compilateur doit pouvoir émettre des messages qui sont délivrés à l'utilisateur dans un délai minimum, par exemple pour avertir d'une erreur. Ce sont ces messages, qui coupent l'activité normale en assurant le contact conversationnel.

Enfin tout langage de communication doit permettre de demander des échanges avec la mémoire de masse, donc de lancer un échange.

Je me place dans l'hypothèse de la Machine de Distribution, qui prend en compte tous les échanges asynchrones

Le système est chargé de gérer les tâches en cours de déroulement mais également de contrôler tous les échanges, en même temps qu'il assure la répartition de la place disponible en mémoire centrale. Je vais décrire les moyens employés pour lui permettre de réaliser l'ensemble du traitement.

TYPOLOGIE DES FILES.

Il existe plusieurs sortes de files selon l'emploi sémantique qui en est fait. Pour un programme, une file peut être "externe", dans la mesure où c'est dans une telle file qu'apparaissent les paramètres, données ou résultats. Une file peut également être locale pour des valeurs qui n'ont aucun intérêt en dehors du champ d'application du programme.

Il existe également des files dont le contenu est externe mais non fonction du calcul, ce sont les files qui contiennent les messages. Ces files font partie de la catégorie générale des files de données qui sont inséparables des

algorithmes codés. C'est le cas des files de variables qui sont jointes au code du programme quand on désire un déroulement avec trace. Alors le dérouleur est construit pour établir un lien entre la variable telle que l'a déclarée l'utilisateur et son image compilée.

Dans le cadre d'un sous-système, chaque compilateur dispose de ses propres files locales, mais les files externes peuvent être communes à plusieurs d'entre eux. Ainsi dans l'exemple suivant qui pourrait s'appliquer à un langage de programmation du type LAC, je considère un ensemble complet minimum avec LP le langage des algorithmes, LE le langage des configurations, et LC le langage de communication.

A chaque langage correspond deux compilateurs: le traducteur et le dérouleur. Le tableau ci-après montre la mise en commun des files externes.

Files	LC	DC	LE	DE	LP	DP	
ϕc : phrase lue	0	0	0	0	0	0	
ϕdc : phrase du dérouleur		0					
variables de LC	0	0					
messages de LC	0						
file locale LC	0						
messages DC		0					
file locale DC		0					
ψc: code de LC			0	0			
ϕe: phrase lue	0	0	0	0	0	0	
ϕde: phrase du dérouleur				0			
variables de LE	0	0	0	0			
messages de LE			0				
file locale LE			0				
messages DE				0			
file locale DE				0			
ψe: code de LE	0	0	0	0			
ϕp: phrase lue	0	0	0	0	0	0	
ϕdp: phrase du dérouleur						0	
variables de LP	0	0			0	0	
messages de LP					0		
file locale LP					0		
message DP						0	
file locale DP						0	
ψp: code de LP	0	0			0	0	

Dans ce tableau on perçoit clairement que certaines files peuvent être communes à quatre compilateurs, comme c'est le cas pour une file de variables ou une file qui contient du code. En effet, le contenu de ces files confectionné par le traducteur, est utilisé par le dérouleur pour calculer, mais est également traité en bloc au moyen du langage de communication entre mémoire centrale et mémoire de masse.

MATERIALISATION DES FILES.

De cela je tire une structure adaptée pour le VIIF global. Chaque VIIF local est attaché à l'un des compilateurs du sous-système. Dans le VIIF global je retrouve donc une première ligne pour l'en-tête de ce VIIF G, suivie d'autant de lignes qu'il y a de VIIF locaux, une dernière ligne contient les descriptifs de chacune des files du sous-système.

{(N°ssyst),(ηtâche),(k),(N°comp),(N°term)} , {VIIF 1} , {VIIF 2} , {VIIF 3} , ...
1 2 3 4 5
TSS DIS DISC

... , {VIIF k} , {(t1),(λ1),(v1) , (t2),(λ2),(v2) , ... , (tq),(λq),(vq)}
F
DISF

Tout ceci constitue une image de tâche. Chaque fois qu'un utilisateur demande à travailler, il doit préciser avec quel sous-système, le système construit une telle image de tâche dans une file que j'appelle **file de distribution**, et qui contient toutes les images des tâches en cours de déroulement.

C'est l'exploration de la file de distribution qui permet au système de gérer tout le fonctionnement de l'ordinateur. La référence TSS donne l'accès au VIIF global, les références DIS et DISC sont calculées pour donner les accès aux divers VIIF locaux, et enfin F et DISF servent pour les descriptifs des files du sous-système.

Les cinq éléments de la première ligne du VIIF G indiquent dans l'ordre: le numéro du sous-système en fonction pour la tâche. Ce numéro repère dans la liste de tous les sous-systèmes que possède le système, celui qui est lié à la tâche. Ensuite l'élément N°2 est l'état de la tâche, le 3 représente le nombre de VIIF locaux, c'est-à-dire le nombre de compilateurs que comporte le sous-système. Le 4 c'est le numéro du compilateur en fonction, au démarrage c'est donc celui du premier compilateur à mettre en ligne. Enfin l'élément 5 est le numéro repère du terminal qui a demandé la tâche en question.

La variable ηtâche prend les diverses valeurs:

"demandé" quand, sous le contrôle du langage hors-texte se trouve créée l'en-tête du VIIF G.

"en cours" quand le système rencontre "demandé", il crée le VIIF G à partir du numéro de sous-système qui lui est réclamé et de sa liste des sous-systèmes.

"terminé" le système liquide alors ce qui appartient à la tâche, et récupère la place en mémoire centrale, il élimine l'image de tâche de la file de distribution.

Le descriptif des files du sous-système contient trois valeurs pour chaque file:

ti qui représente le type de la file.

λi localisation de la file sur la file support: cette valeur donne le rang.

vi donne le volume de la file en question, selon ce qu'en est le type.

Le type lui-même est une variable qui prend les valeurs:

"définitif" quand le volume en est déclaré, par exemple comme file locale dans son compilateur.

"calculé" quand le volume ne peut pas être déclaré dans le compilateur, mais doit absolument être calculé au déroulement, comme une file dont l'encombrement est laissé à l'appréciation de l'utilisateur.

"provisoire" quand le volume est déclaré, mais peut être corrigé par l'utilisateur.

Je reprends rapidement l'organisation du VIIF local en tenant compte de cette structure du VIIF G:

(p1),(d1),(η1),(D1),(N°succ),(FR1),(η^imess),(η^iech),(),(),(),(),(),(),(),(), ...
DISC image du cartouche

(nb.f.loc: li),(lj1),(lj2), ... , (ljli)
[di]

(nb.f.ext: ei),(Jf1),(Jf2), ... , (Jfei)
[Di]

(nb.f.res: s),(m1),(K1),(v'1) , (m2),(K2),(v'2) , ... , (ms),(Ks),(v's)
[FRi]

Cette ligne comporte donc 19+li+ei+s éléments. Les Ip, Jq, et Kr sont les numéros des triplets descripteurs de files dans la ligne à laquelle on atteint par les références F et DISF.

La file de distribution est destinée à contenir les VIIP G qui correspondent à chacune des tâches en cours de déroulement. Les tâches en état de travail conversationnel sont mises en relation avec le terminal qui leur

correspond avec le numéro de terminal. C'est à l'extrémité de la file de distribution que, d'abord le langage hors-texte, puis ensuite le système, font apparaître sur demande de l'utilisateur, le nouveau VIIF G.

C'est pendant la création d'un nouveau VIIF G que le système répartit la place disponible, quand il rencontre "demandé" en [TSS,2 . Par contre, lorsqu'il trouve "terminé" comme valeur de cet élément, il récupère toute la place qu'occupait la tâche.

Le système doit tenir registre de l'occupation des lignes de la file support. Pour cela il dispose d'un tableau dont chaque ligne contient trois informations:

l'état de la ligne, libre ou occupée, le rang de la ligne, ainsi que son volume.

Dans la description de l'algorithme du système, je ne tiendrai pas compte de toutes les techniques possibles de gestion de la mémoire. Bien qu'il s'agisse là d'un problème très important, il sort nettement du cadre de cet exposé. Je veux montrer comment un système universel peut être construit avec les moyens que je me donne, il apparaît clairement qu'un choix de gestion de place s'intègre dans la partie logicielle évolutive du système sans implications sur les points fondamentaux de la construction.

TRANSFERT DES PARAMETRES.

Deux compilateurs se communiquent de l'information par l'intermédiaire des files déclarées externes. Une file externe doit donc être déclarée dans plusieurs contextes. C'est la raison pour laquelle l'ensemble des caractéristiques ne sont déclarées en fait qu'en un seul contexte. Seuls les statuts sont répétés partout, car ils sont utiles partout. Les caractéristiques d'occupation peuvent n'être déclarées, si elles sont indispensables, qu'en un seul contexte.

Par contre, les files ainsi constituées se présentent comme un 'pool' commun à l'ensemble des compilateurs du sous-système, et c'est lors de l'exploitation des instructions d'arrêt que se réalise le transfert des paramètres effectifs.

Il se révèle ainsi nécessaire de disposer dans la phrase symbolique, de d'un repère avec lequel on puisse sans ambigüité désigner la file support de transfert. Pour cela il suffit de numéroter les files externes exactement comme on numérote des statuts, en observant toutefois qu'une même file doit porter le même numéro dans chacun des contextes où elle est déclarée.

La conséquence immédiate de ceci se retrouve dans la construction de la table des sous-systèmes à usage du système. Dans l'opération d'**intégration** d'un ensemble de **compilateurs satellites**, c'est-à-dire d'un sous-système on constitue un statut du pool des files, dans lequel le système doit trouver tout ce dont il a besoin pour réserver de la place dans sa file support. En l'occurence:

le numéro de la file, si elle est externe, puis, le type, la dimension et le mode. Pour les files locales, type et dimension suffisent.

L'ALGORITHME DU SYSTèME.

Je vais, pour l'instant donner une version encore incomplète de la section initiale. J'en donnerai la version définitive ultérieurement en présentant en même temps l'opération d'intégration d'un sous-système dans le système.

Je ne décris donc ici que ce qui est indispensable pour cette version.

M-PROGRAMME

100 000, 10 000 **dynamique** (per, Fc, Nr) , 1 ((2), (1), 3(2)),
2 (2(2), (1), 3(2), 5(1), 3(2), (3), (2), d(2), D(2), FR(2)),
3 (F(2), F(2), F(2)): TSS : DIS : DISC : DISF : TSSFin ;

10 000 **tableau** (type(1), lieu(2), vol(2)) : libre : occupé : libres ;

1 **tableau** (dispo(2), entrée(2)) : Réserve ;

1 000 000 **support** ( ) : P : supplib : conf ;

50 **cartouche** (état(1), sens(1), type(1), Nde(2), Np(2), Ns(2), ADR(3),
vol(2),) : carte : carts : cartc ;

50 **tableau** (typel(1), numéro(2),) : cartlib : cartocc : cartocc2 ;

1 **tableau** (man2(2), tp2(2)) : manoeu ;

section init INITIALISATION ;

Réserve := initial

Libre := initial

TSS := initial

TSSFin := initial (Référence du premier élément non occupé de la file)

cartlib := initial

Manoeu := initial
[Manoeu, man2 := 1 ;
[Manoeu,tp2 := 1 ;

```
        itère Contenu ( cartocc de cartlib à ultime ) ;

        [cartocc := [Manoeu ;
        [Manoeu,man2 := [Manoeu,man2 + 1 ;
        [Manoeu,tp2 := [Manoeu,tp2 + 1 ;

        répétition Contenu ;

fin init SYSTEME ;

                ♦♦♦♦♦♦♦♦♦♦♦♦        ♦♦♦♦♦♦♦♦♦♦♦♦
section SYSTEME ;

        itère TâCHES ;

        DISF := [TSS,3 → TSS ;
        DISF := → DISF ;

        aiguillage FONCTION ( [TSS,2 = ( "dem" : lancement , "en cours" : calcul ,
                "terminé" : stop , "finDIS" : recommencer ; ) )

        lancement : insérer COMMENCER ;
          TSS := [TSS,3 → TSS ;
          TSS := 2 → TSS ;
          sortie lancement ;

        calcul : insérer QUANTUM ;
          TSS := [TSS,3 → TSS ;
          TSS := 2 → TSS ;
          sortie calcul ;

        stop : insérer SUPPRESS ;
          TSS := [TSS,3 → TSS ;
          TSS := 2 → TSS ;
          sortie stop ;

        recommencer : TSS := initial ;
          sortie recommencer ;

        répétition TâCHE ;

fin section SYSTEME ;
```

♦♦♦♦♦♦♦♦♦♦♦♦ ♦♦♦♦♦♦♦♦♦♦♦♦

```
section QUANTUM ;

    DISC := [TSS,4 → TSS ;

    aiguillage MESSAGE ( [DISC,7 = ( "dem" : décroche , "en cours" : suite ,
          "terminé" : continue , "nil" : analyse ; )

    décroche : insérer lancemess ;
       [DISC,7 := "en cours" ;
       sortie décroche ;

    suite : sortie suite ;

    continue : [DISC,7 := "nil" ;
       insérer récupération ;
       sortie continue ;

    analyse : insérer ECHANGE ;
       sortie analyse ;

    noeud MESSAGE ;

fin section QUANTUM ;
```

♦♦♦♦♦♦♦♦♦♦♦♦ ♦♦♦♦♦♦♦♦♦♦♦♦

```
section ECHANGE ;

    aiguillage INOUT ( [DISC,8 = ( "dem" : prep , "en cours" : suite1 ,
          "terminé" : reprise , "nil" : recalcul ; )

    prep : insérer LANCEMESS ;
       [disc,8 := "en cours" ;
       sortie prep ;

    suite1 : sortie suite1 ;

    reprise : [DISC,8 := "nil" ;
       insérer RECUP ;
       sortie reprise ;

    recalcul : insérer COMPUT ;
```

```
        sortie recalcul ;

    noeud INOUT ;

fin section ECHANGE ;

♦♦♦♦♦♦♦♦♦♦♦♦        ♦♦♦♦♦♦♦♦♦♦♦♦

section COMPUT ;

    aiguillage DEROULT ( [DISC,3 = ( "dem" : START , "en cours" : suitcal ,
          "terminé" : compsuiv ) ) ;

    START : DIS := [TSS,4→ TSS ;
      insertion fractionnée DIS ;
      sortie START ;

    suitcal : DIS := [TSS,4→ TSS ;
      insertion fractionnée DIS ;
      sortie suitcal ;

    compsuiv : Nr := [DISC,6 ;
      Par := Nr + 1 ;

    itère PARAMÈTRES ( de 1 à [DISC,Nr ) ;

    aiguillage MODE ( [DISC,Par = ( "transfert" : simple , "contract" : réduct ,
          "expansion" : extension ) ;

    simple : Par := Par + 3 ;
      sortie simple ;

    réduction : Par := Par + 1 ;
      Fc := [DISC,Par - 1 ;
      Fc := Fc * 3 ;
      Fc := Fc + 2 ;
      occupé := [DISC,Fc → initial ;
      libre := [réserve,dispo → initial ;
      [DISF,Fc := [libre,lieu ;
      Par := Par + 1 ;
      Fc := Fc + 1 ;
      [DISF,Fc := [DISC,Par ;
      [occupé,type := "λ" ;
```

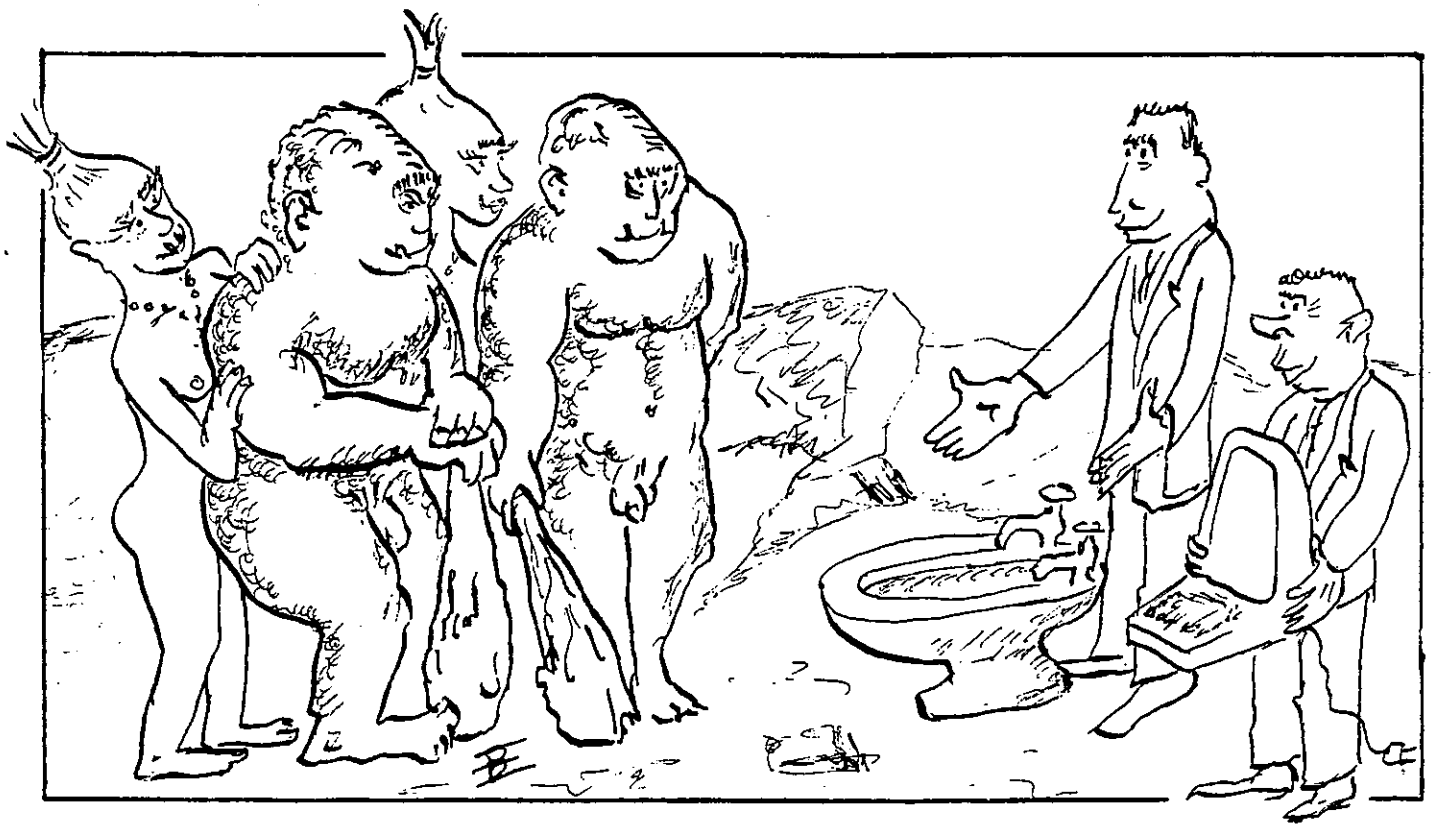
```
   [libre,vol := [DISC,Par ;
   suplib := [libre,lieu → initial
file suplib , [DISC,Par ;                    Création de la nouvelle file.
   libres := → libre ;
   [libres,lieu := [libre,lieu ;
   libre := → libre ;
   conf := [occupé,lieu → initial ;
   [suplib := [conf ;
   Par := Par + 1 ;

   sortie réduction ;

extension : Par := Par + 1 ;
   Fc := [DISC,Par - 1 ;
   Fc := Fc * 3 ;
   Fc := Fc + 2 ;
   occupé := [DISF,Fc → initial ;
   [DISF,Fc := [libre,lieu ;
   [occupé,type := "λ" ;
   Par := Par + 1 ;
   [libre,vol := [occupé,vol + [DISC,Par ;
   Fc := Fc + 1 ;
   [DISF,Fc := [libre,vol ;
   libres := → libre ;
   [libres,lieu := [libre,lieu ;
   suplib := [libre,lieu → initial ;
file suplib , [libre,vol ;
   libre := → libre ;
   Par := Par + 1 ;
   conf := [occupé,lieu → initial ;
   [suplib := [conf ;

   sortie extension ;

noeud MODE ;

répétition PARAMèTRES ;

   [TSS,4 := [DISC,5 ;
   DIS := [TSS,4 → TSS ;
   [DIS,3 := "demandé" ;
```

```
    sortie compsuiv ;

    noeud DéROULT ;

fin section COMPUT ;

♦♦♦♦♦♦♦♦♦♦♦♦        ♦♦♦♦♦♦♦♦♦♦♦♦

section RéCUP ;

        cartlib := ← cartlib ;
        cartocc := [DISC,5 → initial ;
        cartocc2 := [cartocc,type1 → initial ;
        [Manoeu,Man2 := [cartocc2,numéro ;
        [Manoeu,tp2 := [cartocc,type1 ;
        [cartocc2,numéro := [cartlib,numéro ;
        [cartlib,numéro := [Manoeu,num2 ;
        [cartocc,type1 := [cartlib,type1 ;
        [cartlib,type1 := [Manoeu,tp2 ;

fin section RéCUP ;

♦♦♦♦♦♦♦♦♦♦♦♦        ♦♦♦♦♦♦♦♦♦♦♦♦

section LANCEMESS ;

        cartc := [cartlib,numéro → initial ;
        [DISC,5 := [cartlib,numéro ;
        [cartc := [DISC,9 : 16          Ecriture abrégée pour le transfert de 8
cases.
        [cartc,type := "demandé" ;
        cartlib := → cartlib ;

fin section LANCEMESS ;
```

J'ai laissé de côté un certain nombre de questions à reprendre ultérieurement. Le fonctionnement du langage hors-texte avec la gestion des entrées conversationnelles, l'intégration des sous-systèmes, et la création de la file de distribution, seront repris en détail et viendront compléter cette organisation de système. Quant au problème soulevé par la gestion de la place en file support cela fera l'objet de quelques hypothèses.

Notre civilisation expliquée par les jeux.

Torture : jeu de maux .
Partouze : jeu de maints, jeu de vilains.
Savant construisant la bombe atomique et râlant qu'on l'utilise sans lui dire : jeu de lumière.
Goulags : jeux d'ombres.
Des rapports avec l'Administration : jeux d'O.
Se faire sauter en s'étirant dans son lit : jeux de ponts.



Heu ... je vois que nous arrivons à temps ... le progrès ...