UNIVERSITE DE NANCY I

U. E. R. DE MATHEMATIQUES

l'analyse dans le projet civa

thèse

pour l'obtention
du doctorat de spécialité
de mathématiques appliquées
(informatique)

soutenue le 27 juin 1974

par

norbert hertschuh

jury :

M.	C.	PAIR	Président
M.	J. C.	DERNIAME	Examinateurs
Mme	C.	ROLLAND	

UNIVERSITE DE NANCY I

U. E. R. DE MATHEMATIQUES

l'analyse dans le projet civa

thèse

pour l'obtention
du doctorat de spécialité
de mathématiques appliquées
(informatique)

soutenue le 27 juin 1974

par

norbert hertschuh

jury :

M.	C.	PAIR	Président
M.	J. C.	DERNIAME	Examinateurs
Mme	C.	ROLLAND	

Je remercie vivement Monsieur le Professeur PAIR, Directeur de l'Institut Universitaire de Calcul Automatique, pour tous les conseils qu'il m'a prodigués et pour l'honneur qu'il me fait en présidant ce Jury.

Mes remerciements vont aussi à Madame ROLLAND qui a accepté de participer au Jury.

Que Monsieur DERNIAME trouve, dans ce travail, l'expression de ma profonde et amicale reconnaissance pour l'attention, les précieux conseils et l'efficacité qu'il m'a apportés dans la direction de mes recherches.

J'adresse mes remerciements à l'Equipe CIVA, spécialement à Monsieur FIEGEL, pour leur collaboration.

Enfin, je remercie Madame Zannad, Mademoiselle Le Maréchal et Monsieur Deberdt pour la qualité de leur travail dans la réalisation matérielle de cette thèse.

E R R A T A

Page VI - 7 : lignes 28 et 29

Page VI - 8 : ligne 4

Page VI - 12 : lignes 13 et 24

Lire NØART au lieu de NØPRØD

Page VI - 9 : lignes 16 et 20

Lire FIT-VAL au lieu de FIT-VALOR

Page VI - 15 : ligne 12

Lire FIT-REGL au lieu de FI-DEC-STATS

SOMMAIRE

PREMIERE PARTIE : ORGANISATION LOGIQUE DES INFORMATIONS ET CINEMATIQUE DES FICHIERS

ANNEXES : ANALYSE EN CIVA D'UNE APPLICATION :"LA PAIE DU LAIT"

INTRODUCTION

Ce travail est composé de deux parties.

Dans la première, après avoir présenté l'organisation logique des données en entrée d'un programme, nous présentons une instruction d'itération logique dans CIVA et un métalangage dans Cobol, permettant de libérer le programmeur de la gestion des fichiers et réalisant ce qu'on appelle "la cinématique automatique des fichiers".

Dans la deuxième partie, nous proposons un certain nombre d'outils permettant de mener à bien l'analyse d'une application dans le cadre de CIVA.

Il pourrait sembler plus logique d'étudier d'abord les problèmes d'analyse au niveau d'une application pour terminer par les problèmes d'analyse au niveau du programme. Mais nous avons choisi cet ordre de présentation parce que plusieurs notions présentées dans la première partie seront nécessaires à la compréhension de la seconde partie.

IERE PARTIE

/ORGANISATION LOGIQUE DES INFORMATIONS ET CINEMATIQUE DES FICHIERS/

INTRODUCTION

Le programme définit une relation entre deux ensembles d'informations. L'un d'eux (les entrées) fait l'objet du traitement alors que l'autre en représente la finalité (résultats).

Cette relation est exécutée par un ensemble d'actions élémentaires qui sont liées entre elles selon une organisation logique. La nature de ces actions dépend de l'objectif du programme, c'est-à-dire des résultats qu'on veut obtenir. Par contre, l'organisation des actions dépend de la structure des données qui font l'objet du traitement. Nous allons étudier ce problème plus particulièrement pour les programmes de gestion dont les entrées sont constituées essentiellement par des fichiers traités séquentiellement.

Soit le problème de l'établissement de factures à partir du fichier des articles commandés par les clients. Chaque enregistrement comporte le numéro du client, le numéro de la commande, le numéro de l'article commandé, la quantité commandée et le prix unitaire de l'article commandé.
Les enregistrements du fichier sont triés par numéro client, numéro de commande et numéro d'article. On en déduit tout naturellement l'organisation logique des traitements qui est la suivante :

Pour chaque client {
- éditer en-tête facture ;
- éditer total facture ;

Pour chaque commande {
- éditer en tête commande ;
- calcul total facture ;
- éditer sous total commande ;

Pour chaque article {
- Montant = prix * quantité ;
- Calcul total commande ;
- Editer ligne article ;

On constate qu'il y a trois niveaux de traitements, chacun étant attaché à un ensemble de notions caractérisé par une notion appelée indicatif.

Ainsi l'indicatif"commande"caractérise l'ensemble de données dont les éléments sont les enregistrements ayant le même numéro de commande. Ces ensembles de données peuvent être inclus les uns dans les autres.
Ainsi l'ensemble des données caractérisant un article commandé est un élément de l'ensemble de données caractérisant une commande.

Malheureusement les langages de programmation actuels ne permettent pas une telle formulation des traitements et la programmation de cette classe de problèmes est fort délicate.

D'une part, on ne peut accéder aux ensembles de données à partir de la valeur des indicatifs.
D'autre part, les langages de programmation ne permettent pas facilement d'associer un traitement particulier ou un changement de traitement quand, en examinant successivement les données, on rencontre un changement de valeur pour un indicatif (que nous appellerons une rupture).

C'est alors au programmeur d'assurer la gestion de ces ruptures.

Par conséquent, avant d'effectuer le traitement concernant un article, il faut vérifier qu'il n'y a pas eu rupture sur le numéro de commande, c'est-à-dire qu'on traite toujours la même commande.

S'il y a rupture, il faut choisir la nouvelle valeur de numéro de commande en consultant les fichiers contenant les commandes.

Nous allons montrer, dans ce qui suit, comment on peut accéder aux informations à partir des indicatifs et comment on peut résoudre de manière systématique le problème des ruptures de niveau et de choix des valeurs d'indicatifs.
Puis nous verrons comment ces solutions ont été adaptées au langage de programmation CIVA.
Enfin, nous présenterons la réalisation de la cinématique automatique des fichiers en Cobol par génération des instructions de gestion des fichiers à partir d'un métalangage spécifique.

Chapitre I

ORGANISATION LOGIQUE DES INFORMATIONS ET STRUCTURE DU PROGRAMME

ORGANISATION LOGIQUE DES INFORMATIONS ET STRUCTURE DU PROGRAMME

I.1. - INTRODUCTION :

Certaines méthodes de programmation proposent une structure hiérarchique des programmes liée à la structure hiérarchique et répétitive des données (CANTOR (31), WARNIER (28)).

Nous avons eu l'occasion de les utiliser dans le cadre de notre enseignement et nous avons compris leur grand intérêt pédagogique (DUFOURD, HERTSCHUH (23)).

Ceci nous a conduit à étudier de manière plus approfondie l'organisation des données et leur conséquence sur la structure des programmes.
Et, partant de là, nous avons pu résoudre formellement les problèmes de lecture des fichiers, de ruptures de niveau et de choix de valeur des indicatifs que nous présentons dans ce chapitre.

I.2. - ORGANISATION LOGIQUE DES INFORMATIONS CONTENUES DANS UN FICHIER

I.2.1. - La notion de fichier :

Dans la majorité des applications, le système d'information doit gérer non des entités isolées mais des ensembles finis d'entités présentant des propriétés homogènes : ensemble des produits, ensemble des salariés, ensemble des clients, etc...

Cet ensemble d'entités sera caractérisé par un ensemble de notions (1) constituant une unité logique.
Un article sera l'ensemble des données associées à une réalisation de l'unité logique.
Un fichier sera défini comme une file d'articles.
Un enregistrement est l'unité d'information pouvant être traitée sur le support physique du fichier. Dans les fichiers internes CIVA, un enregistrement sera la représentation physique d'un article.

I.2.2. - Groupement logique et indicatif :

I.2.2.1. Définitions :

Soit F un fichier, c'est-à-dire un ensemble d'enregistrements.

Soit $A = \{ A_1, \ldots\ldots, A_n \}$ l'ensemble des notions caractérisant le fichier F.

Soit α un sous-ensemble de A, par exemple $\alpha = \{ A_1, A_2 \ldots, A_m \}$ avec $m < n$

Soit $v(A_i)$ l'ensemble des valeurs associé à la notion A_i et $v(\alpha) = v(A_1) \times v(A_2) \ldots\ldots\ldots \times v(A_m)$.

(1) Nous appelons notion un concept défini par l'utilisateur auquel on peut associer un ensemble de valeurs, chaque valeur est une réalisation de la notion et constitue une donnée (CHABRIER J.J. (5)).

S'il existe une fonction bijective $f : v(\alpha) \longrightarrow F$ et si tout sous-ensemble $\alpha_1 \in \alpha$ ne définit pas une fonction bijective $f : v(\alpha_1) \longrightarrow F$, on dit que α est un index pour le fichier F.

Les notions constituant l'index seront appelées les indicatifs de base du fichier F et permettent d'identifier chaque enregistrement du fichier.
On peut mettre en évidence dans le fichier F des sous-ensembles composés d'enregistrements ayant même valeur pour une notion $A_p \in A$. Ces sous-ensembles seront appelés groupements logiques d'indicatif A_p. L'indicatif A_p peut appartenir ou non au sous-ensemble α.

La nécessité de partager un fichier en groupements logiques apparaît dans l'analyse (voir 2ème partie "analyse dans CIVA") et répond aux besoins des traitements à effectuer sur les données.

Exemple :

Soit à éditer un état des salaires versés par une entreprise sur lequel on a pour chaque salarié : le nom et le salaire ; pour chaque service : le total des salaires versés ; pour chaque usine : le total des salaires versés.
Le fichier comporte les notions suivantes : numéro usine, numéro service, matricule, nom, salaire.
L'index de ce fichier est le matricule.
On a deux indicatifs : numéro usine, numéro service, permettant de partager le fichier en groupements logiques nécessaires pour effectuer les traitements associés à "service" et "usine".

Dans la suite de ce travail, nous appelons indicatif les indicatifs de base des fichiers ainsi que les notions permettant de définir des groupements logiques dans les fichiers.

1.2.2.2. Relations entre les groupements logiques :

1.2.2.2.1. Relation d'inclusion

Dans les fichiers en entrée d'un programme, on peut définir plusieurs groupements logiques d'indicatifs I_1, I_2,..., I_n. Entre ces différents gorupements logiques, on peut définir des relations d'inclusion.

Soit G_{k-1} l'ensemble des groupements logiques d'indicatif I_{k-1} dans le fichier F.

Soit $G_{k-1}(x)$ le groupement logique correspondant à une réalisation x de I_{k-1}.

Soit $G_k(x, y)$ le groupement logique correspondant à une réalisation x de I_{k-1} et y de I_k.

Si pour tout $x \in I_{k-1}$, $G_{k-1}(x) = \bigcup_{y \in I_k} G_k(x, y)$

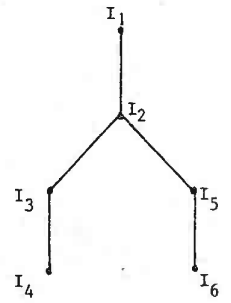
on dira que les groupements logiques d'indicatif I_k sont inclus dans les groupements logiques d'indicatif I_{k-1} et que l'indicatif I_k est d'ordre ou de niveau inférieur à I_{k-1}.

Le groupement logique d'indicatif I_k associé à une réalisation x de I_k est composé d'un ensemble d'enregistrements ayant même valeur pour les indicatifs I_1, I_2, ..., I_{k-1}, I_k.

Exemple :

Considérons le fichier des salariés d'une entreprise dont les indicatifs sont : numéro usine, numéro service, matricule. Le groupement logique associé à une réalisation de l'indicatif numéro usine est composé des groupements logiques d'indicatif numéro service ayant le même numéro usine.
Numéro usine sera un indicatif d'ordre supérieur au numéro service.

Cette relation permet de définir sur les indicatifs associés aux groupements logiques un ordre représenté par une arborescence.



Chaque branche de l'arborescence est constituée d'une suite d'indicatifs vérifiant la relation d'inclusion.

1.2.2.2.2. Relation d'affinité

Si nous posons :

$G_{k-1}(x)$: groupement logique associé à une réalisation x de l'indicatif I_{k-1}

$G_{k-2}(y)$: groupement logique associé à une réalisation y de l'indicatif I_{k-2}

$G_k(x, z)$: groupement logique associé à une réalisation x de I_{k-1} et z de I_k

$G_k(y, z)$: groupement logique associé à une réalisation y de I_{k-2} et z de I_k

Si $G_{k-1}(x) = \bigcup_{z \in I_k} G_k(x, z)$,

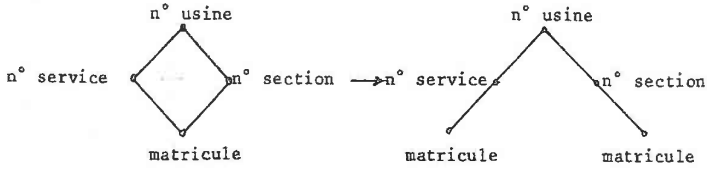
Si $G_{k-2}(y) = \bigcup_{z \in I_k} G_k(y, z)$

et, s'il n'y a pas de relation d'inclusion entre G_{k-1} et G_{k-2}, on dira qu'il y a relation d'affinité entre G_{k-1} et G_{k-2}.

La représentation des relations d'inclusion et d'affinité nous définit un réseau.

Exemple :

Soit le fichier des salariés dont les indicatifs sont numéro usine, numéro service, numéro section, matricule. Numéro service et numéro section sont d'ordre inférieur à numéro usine ; matricule est d'ordre inférieur à numéro service et à numéro section.
On a donc :



Il y a relation d'affinité entre les indicatifs numéro service et numéro section, car ils ont même indicatif de niveau inférieur matricule.

Comme nous travaillons sur des organisations classiques de fichiers, nous nous ramenons à une structure arborescente normale par duplication des groupements logiques ayant plus d'un ascendant dans le graphe.

1.2.3. - Le traitement des groupements logiques :

1.2.3.1. Ordre de rangement des fichiers :

A l'ensemble des groupements logiques d'indicatif I_k sera associé un ensemble de traitements qui seront exécutés pour chaque valeur de l'indicatif I_k.

Ceci suppose qu'on puisse accéder successivement à tous les enregistrements constituant le groupement logique associé à une valeur de I_k.
Or, nous voulons réaliser la cinématique des fichiers dans le cas de fichiers séquentiels, c'est-à-dire que pour accéder à un enregistrement donné, il faut lire tous les enregistrements le précédant sur le support physique.

Cette contrainte d'accès impose l'introduction d'un ordre de présentation des enregistrements sur le support du fichier afin que tous les enregistrements ayant même indicatif soient regroupés.

Ceci implique la nécessité de trier le fichier selon ses indicatifs avant d'effectuer tout traitement.
De plus, les groupements logiques définis sur un fichier doivent vérifier la relation d'inclusion, car un enregistrement ne peut pas appartenir simultanèment à deux groupements logiques non inclus l'un dans l'autre.

A tout fichier sera donc associée une suite d'indicatifs $I_1, I_2, \ldots\ldots, I_n$ ordonnée par la relation d'inclusion et définissant sur ce fichier des groupements logiques d'indicatifs $I_1, I_2, \ldots\ldots, I_n$.

I_1 sera appelé indicatif majeur du fichier ;

I_n caractérise les enregistrements du fichier et sera appelé indicatif mineur.

1.2.3.2. Groupements logiques simples et multiples :

- Groupements logiques simples

On dit d'un groupement logique qu'il est simple si les éléments de ce groupement logique ne proviennent que d'un seul fichier. Ceci est le cas chaque fois qu'un indicatif n'est associé qu'à un seul fichier.

La reconnaissance d'un groupement logique simple I_1 consiste à détecter les ruptures de valeur de I_1 sur le seul fichier concerné.

- Groupements logiques multiples

Un groupement logique I_m est multiple s'il est composé d'enregistrements provenant de plusieurs fichiers différents.
Ceci implique que l'indicatif I_m le caractérisant est associé à plusieurs fichiers.
La reconnaissance des groupements logiques multiple sera plus complexe, car elle nécessite le déroulement simultané de tous les fichiers ayant I_m comme indicatif et la détection des ruptures sur valeur de I_m dans tous ces fichiers.

1.3. - ORGANISATION LOGIQUE DU PROGRAMME :

Elle sera basée sur l'arbre de traitement traduisant l'organisation logique des informations contenues dans les fichiers séquentiels en entrée du programme.

1.3.1. - L'arbre de traitement (DU BEANDIEZ et LAPAUW (22)) :

1.3.1.1. Principe :

Il se compose d'une ou plusieurs branches, chaque branche comportant une suite d'indicatifs.
Cela signifie que le programme généré contiendra des opérations spécifiques de chacun des niveaux de chaque branche.

Soit une branche ayant pour indicatifs :

- client,
- produit,
- date de commande.

Dans ce cas, on aura un certain nombre d'opérations à réaliser pour chaque client, d'autres pour chacun des produits d'un client, d'autres enfin pour chaque date de commande de chaque produit de chaque client.
Ceci implique que toutes les informations en entrée dont les indicatifs et arguments de tri sont :

- soit client,
- soit client-produit,
- soit client-produit-date de commande,

seront lues et traitées séquentiellement.

Toutes les autres informations en entrée nécessaires à cette branche seront utilisées en accès direct.

Un programme peut être constitué de plusieurs branches, à condition que le graphe soit arborescent.

1.3.1.2. Exécution de l'arbre de traitement :

Les opérations spécifiques de chacun des niveaux de la branche seront divisées en deux parties :

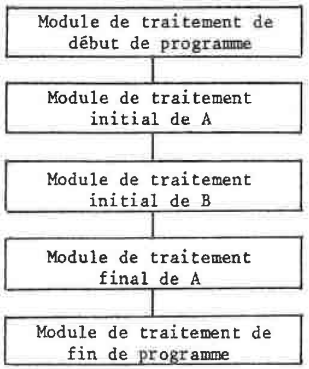
- la première sera exécutée avant que les opérations des niveaux inférieurs de la branche ne soient exécutées ; on l'appellera module de traitement initial ;

- la deuxième sera exécutée après que les opérations des niveaux inférieurs auront été exécutées. En effet, on peut avoir besoin des résultats obtenus aux niveaux inférieurs pour effectuer certains traitements du niveau supérieur ; on l'appellera module de traitement final.

Ainsi, avant d'éditer le total d'une commande, il faut avoir traité tous les produits de la commande. Seules les opérations spécifiques aux feuilles de l'arbre de traitement ne seront pas divisées en deux parties.

Exemple :

Soit un programme à une branche ayant pour indicatifs A et B.
Quel que soit le nombre de fichiers en entrée, les modules de traitement de ce programme seront organisés selon le schéma suivant :



Le déroulement du programme ci-dessus sera le suivant :

a) exécution du module "début de programme".

b) choix de la valeur de A à traiter sur l'ensemble des fichiers séquentiels.

c) exécution du module "traitement initial de A" pour la valeur de A choisie.

d) choix de la valeur de B à traiter sur l'ensemble des fichiers séquentiels dont la valeur de A a été préalablement choisie.

e) exécution du module "traitement de B" pour la valeur B choisie.

f) choix de la valeur suivante de B. S'il n'y a pas de rupture sur valeur de A, on va à e, sinon :

g) exécution du module "traitement final de A" pour l'ancienne valeur de A.

h) choix de la valeur suivante de A sur l'ensemble des fichiers séquentiels ; si pas fin de fichier sur l'ensemble, on va à c, sinon :

i) exécution du module "fin de travail".

Remarques :

1) L'exécution des modules n'est pas systématique aux moments indiqués ci-dessus. Si aucune valeur d'indicatif correspondant ne peut être choisie, faute d'exister, on n'exécute pas le module.

   Par exemple, il pourrait n'exister aucun enregistrement d'indicatif A et B ayant la valeur de A choisie ; dans ce cas, on n'exécutera pas le module B pour cette valeur de A.

2) Lorsque l'arbre de traitement comporte plusieurs branches, celles-ci seront exécutées l'une après l'autre, en allant de la gauche vers la droite.

3) Certains modules du schéma peuvent être vides. En effet, il peut arriver que la nature des résultats demandés n'implique aucun traitement dans certains modules.

1.3.1.3. Nature des opérations dans l'exécution de l'arbre de traitement :

On constate qu'il y a deux types d'opérations dans le programme qui traduit l'arbre de traitement :

1 - Les opérations spécifiques de chacun des niveaux des branches de traitement. Elle sont regroupées dans les modules de traitement. Ces modules sont au nombre de deux par niveau d'une branche, sauf pour le dernier niveau de chaque branche qui ne donne lieu qu'à un seul module.

2 - Les opérations dont la fonction est :
   . de lire et interclasser tous les fichiers à lecture séquentielle pour déterminer la valeur de l'indicatif à traiter à chaque niveau ;
   . de détecter les ruptures sur valeur d'indicatif et de gérer en conséquence l'exécution des modules de traitement.

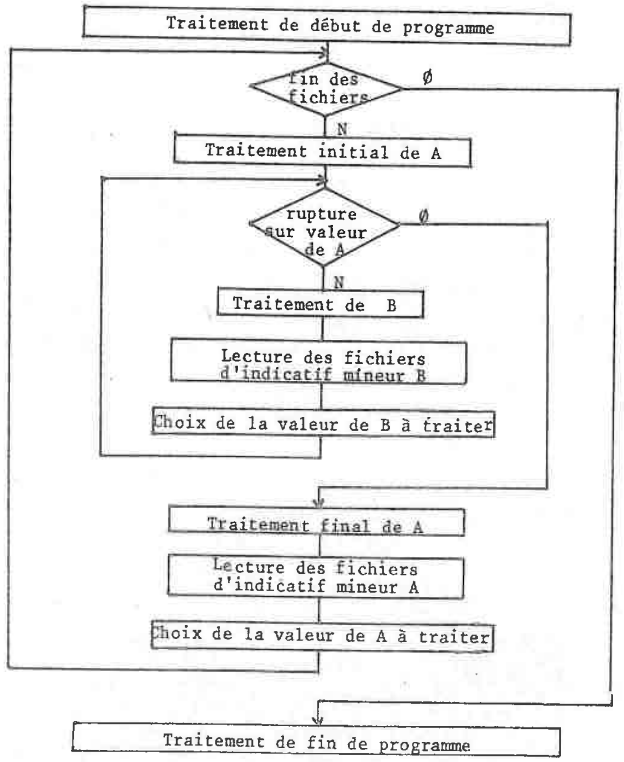
Ces opérations sont regroupées dans les modules de cinématique des fichiers que nous allons mettre en évidence dans l'organigramme d'exécution de l'arbre de traitement.

1.3.2. - Construction de l'organigramme d'exécution :

1.3.2.1. Le principe :

On part de l'arbre de traitement que l'on va linéariser et auquel on va ajouter les opérations de lecture de fichiers, de choix de valeur des indicatifs, de rupture sur valeur d'indicatif.

Soit l'exemple précédemment traité d'un programme comportant une branche ayant pour indicatifs A et B.



Dans le module "traitement de début de programme", on effectuera la lecture initiale de tous les fichiers et le choix des valeurs initiales de A et B à traiter sur l'ensemble des fichiers séquentiels.

On a placé le test "rupture sur valeur de A" entre le module "traitement initial de A" et le module "traitement de B", car ce test doit être effectué non seulement chaque fois qu'on a choisi une nouvelle valeur de B à traiter, mais également lorsqu'on a choisi une nouvelle valeur de A à traiter.

Cette constatation est valable pour le test "fin des fichiers" qui doit être effectué chaque fois que tous les fichiers ont été lus au moins une fois, mais également après la lecture initiale de tous les fichiers dans le module "traitement de début de programme".

Codification de l'organigramme d'exécution :

MTI_j : Module de traitement initial attaché à l'indicatif I_j.

$MTF_{j,1}$: Module de traitement final attaché à l'indicatif I_j. On utilise l'indice 1 s'il y a plusieurs modules de traitement final attachés à un indicatif I_j. Ceci est le cas lorsque l'indicatif I_j a plusieurs successeurs sur l'arbre de traitement (voir 1.3.2.3.).

MT_j : Module de traitement attaché à l'indicatif I_j qui est une feuille de l'arbre de traitement.

LCI_j : Lecture des fichiers d'indicatif mineur I_j, choix de la valeur de I_j à traiter.

$TR_{j,1}$: Test de rupture sur la valeur de I_j. On utilise l'indice 1 pour les mêmes raisons que $MTF_{j,1}$.

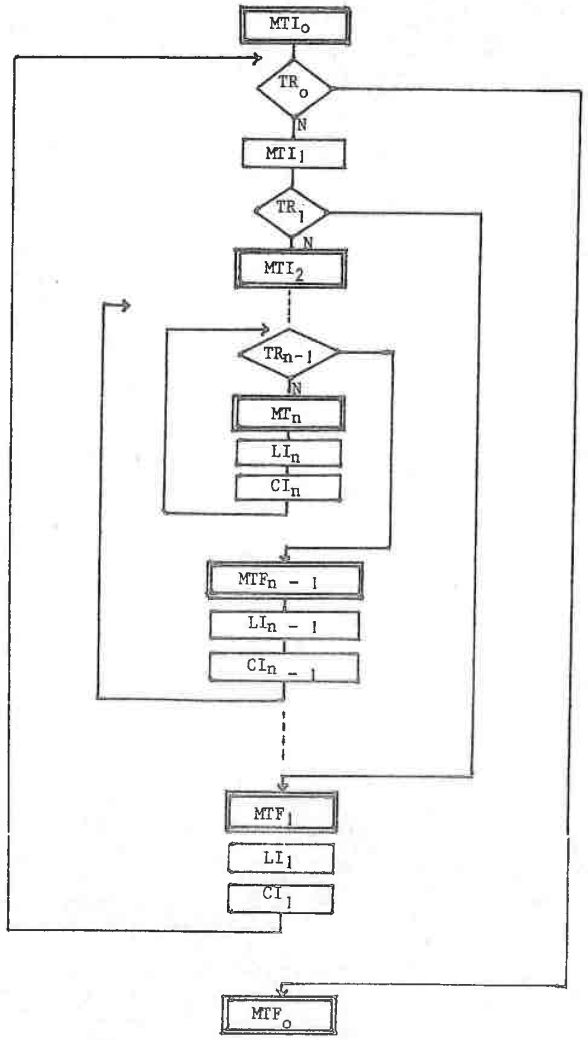
MTI_o : Module "traitement de début de programme".

MTF_o : Module "traitement de fin de programme".

TR_o : Test "fin des fichiers".

1.3.2.2. Arbre de traitement à une branche :

Soit un programme ayant une branche de traitement avec la suite d'indicatifs I_1, ..., I_n.
L'organigramme correspondant sera le suivant :



Les modules MTIO, MTFO, TRO ne sont pas attachés à un indicatif.

Les modules MTI_i, MTF_i, TR_i, LC_i pour i allant de 1 à n - 1 sont attachés à la suite d'indicatifs I1, ..., I_{n-1}

Les modules MT_n et LCI_n sont attachés à l'indicatif n, qui est l'indicatif mineur de l'ensemble des fichiers en entrée du programme.

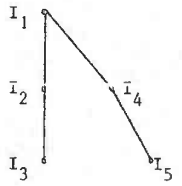
Tout programma constitué d'une branche avec la suite d'indicatifs I_1, ..., I_n, aura le même organigramme que ci-dessus.

Ce dernier est totalement indépendant du découpage en fichiers des informations en entrée.

Le découpage en fichiers intervient uniquement dans la construction des modules de cinématique des fichiers TR_i et LCI_i.

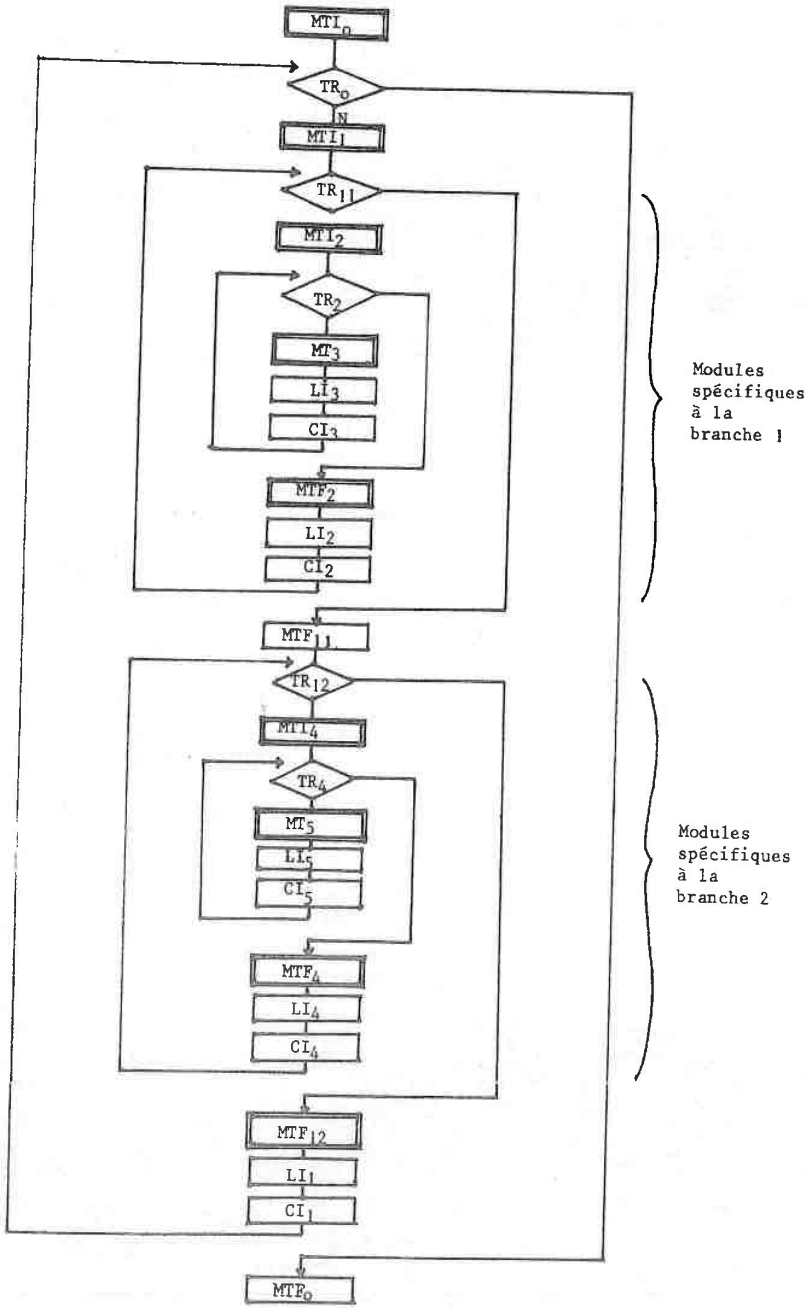
1.3.2.3. Arbre de traitement à plusieurs branches :

Soit un programme ayant un arbre de traitement à deux branches et la suite d'indicatifs I_1, I_2, I_3, I_4, I_5.



L'indicatif I_1 est commun aux deux branches.

L'organigramme se présentera ainsi :



On exécute d'abord les modules attachés aux indicatifs de la branche 1, puis on exécute les modules attachés aux indicatifs de la branche 2.

Il y a deux modules de traitement final : MTF_{11} et MTF_{12}, attachés à l'indicatif I_1 car celui-ci a deux successeurs I_2 et I_4 sur l'arbre de traitement.

Le premier module de traitement final MTF_{11} est exécuté lorsqu'il y a rupture sur valeur de I_1 dans TR_{11}, le second est exécuté lorsqu'il y a rupture sur valeur de I_1 dans TR_{12}.

Pour le reste, l'organisation de l'organigramme est identique à celle correspondant à un arbre de traitement à une branche.

1.3.3. - Organisation des modules de "cinématique des fichiers" :

1.3.3.1. Le module de lecture des fichiers : LI_j :

Dans le module LI_j seront lus tous les fichiers d'indicatif mineur I_j.

Exemple :

Si la suite des indicatifs du fichier FICH est I_1, I_2, I_3, il sera lu dans le module LI_3.

Un fichier d'indicatif mineur I_j ne sera lu que si la valeur de l'indicatif I_j et des indicatifs d'ordre supérieur sur la même branche est égale à la valeur qui vient d'être traitée pour chacun de ces indicatifs et si le fichier n'est pas encore terminé.

Exemple :

Soit le fichier FICH ayant pour indicatifs I_1, I_2. La valeur de I_2 dans l'enregistrement présent est égale à 30 et celle de I_1 est égale à 5.

Si la valeur de I_2 qui vient d'être traitée est égale à 29 et celle de I_1 est égale à 5, on ne lira pas le fichier.

En effet, si on le faisait, on ignorerait un enregistrement du fichier.

1.3.3.2. Le module de choix de valeur des indicatifs :

Soit le module de choix de valeur de l'indicatif I_j : CI_j.
Il consiste à déterminer, après la lecture des fichiers d'indicatif mineur I_j, quelle est la valeur de I_j et de tous les indicatifs d'ordre supérieur à I_j sur la branche en cours de traitement.

Pour ce faire, un ou plusieurs fichiers ayant I_j parmi leurs indicatifs vont jouer un rôle particulier du fait que toute valeur du fichier résultant doit figurer dans un de ces fichiers.

Soit la notation suivante :

$f \in E(I_j) \iff$ f est un fichier ayant I_j comme indicatif.

On appellera ensemble directeur pour l'indicatif I_j un ensemble $D(I_j)$ tel que $D(I_j) \subset E(I_j)$ et toute valeur de I_j figurant dans $T = \bigcup_{f \in E(I_j)} f$ figure dans D.

Le choix s'opère en commençant par les indicatifs majeurs :

- parmi toutes les valeurs d'un indicatif I_1 (I_1 étant un indicatif d'ordre supérieur sur la même branche que I_j) dans les enregistrements présents, choisir la plus petite (resp. la plus grande) si les fichiers sont classés dans l'ordre croissant (resp. décroissant) de cet indicatif ;

la valeur de I_1 choisie sera rangée dans la zone ZT_jI_1 de même type que I_1.

- choisir ensuite la valeur de l'indicatif d'ordre inférieur I_{1+1} parmi tous les enregistrements présents dont la valeur de I_1 est égale à la valeur de ZT_jI_1 (ceci devant être vérifié pour tous les indicatifs d'ordre supérieur à I_1 sur la même branche).

- choisir finalement la valeur de I_j parmi tous les enregistrements présents vérifiant la relation $I_1 = ZT_jI_1$ $\forall 1$ tel que $I_1 \in S(I_j)$, $S(I_j)$ étant l'ensemble des indicatifs d'ordre supérieur à I_j sur la même branche.

La valeur de I_j choisie sera rangée dans la zone de référence ZRI_j de même type que I_j, on traitera la valeur de I_j contenue dans ZRI_j.

1.3.3.3. Module de test de rupture :

Soit le module de test de rupture TR_j.
Suivant le résultat du test de rupture TR_j, on va exécuter soit le module de traitement MTI_j attaché à l'indicatif I_j, soit le module de traitement MTF_{j-1} attaché à l'indicatif I_{j-1} d'ordre supérieur à I_j sur la même branche.

Le module de traitement MTI_j sera exécuté sous deux conditions :

- si au moins un fichier parmi les fichiers directeurs pour l'indicatif I_j n'est pas terminé.

En effet, si tous les fichiers directeurs pour l'indicatif I_j sont terminés, on ne peut plus choisir de valeur pour cet indicatif ; par conséquent, on n'exécutera plus les modules de traitement attachés à cet indicatif jusqu'à la fin du programme. Pour tester la fin d'un fichier, on utilisera une

zone de travail de un caractère codifié FF_i (pour le ième fichier du programme) et qui sera positionnée à 1 dès qu'on a atteint la fin du fichier.

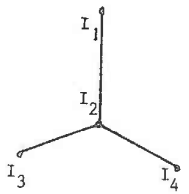
- si la valeur des indicatifs d'ordre supérieur à I_j, dans l'enregistrement sélectionné pour fournir la valeur de I_j, est égale à la valeur choisie pour cesindicatifs pour l'itération en cours :

$$ZRI_l = ZT_j I_l \quad (\forall l \quad : \quad I_l \in S(I_j)$$

$S(I_j)$ étant l'ensemble des indicatifs d'ordre supérieur à I_j sur la même branche.

1.3.3.4. Exemple d'organisation :

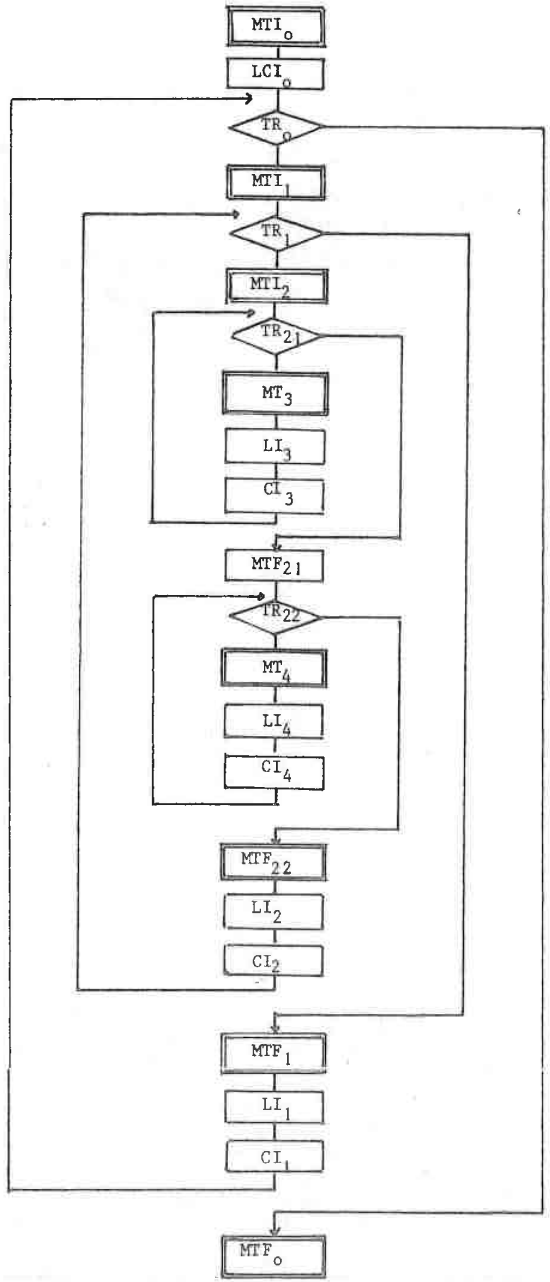
Soit un programme dont les traitements sont attachés aux indicatifs I_1, I_2, I_3, I_4 de type entier, dont la structure est donnée par l'arbre de traitement suivant :



Les fichiers en entrée du programme sont F_1, F_2, F_3, F_4.
F_1 et F_2 ont pour indicatifs I_1, I_2, I_3.
F_3 est compose des indicatifs I_1, I_2 et F_4 des indicatifs I_1, I_2, I_4.

Les fichiers sont supposés triés par I_1, I_2, I_3, I_4 croissant et sont tous directeurs.

La structure du programme sera représentée par le schéma suivant :



Les modules et les procédures de cinématique des fichiers sont définis ci-dessous :

```
Procédure LCI0 ;
premier X1 ; premier X2 ; premier X3 ; premier X4 ;
FF1 = 0 ; FF2 = 0 ; FF3 = 0 ; FF4 = 0 ;
CI4 ; CI3 ; CI2 ; CI1 ;
fin proc ;

Procédure LI4 ;
   Si I1 de X4 = ZR1 et I2 de X4 = ZR2 et I4 de X4 = ZR4
      alors X4 en X4 + 1 ;
            si ∉ debfile alors FF4 = 1 ; I1 de X4 = maxentier
            fsi
   fsi ; finproc ;

Procédure LI3 ;
   lecture 3 (X1, FF1) ; lecture 3 (X2, FF2)
fin proc ;

Procédure lecture 3 (X, FF) ;
   Si I1 de X = ZR1 et I2 de X = ZR2 et I3 de X = ZR3
      alors X en X + 1
            si ∉ debfile alors FF = 1 ; I1 de X = maxentier fsi
   fsi
fin proc ;

Procédure LI2 ;
   Si I1 de X3 = ZR1 et I2 de X3 = ZR2
      alors X3 en X3 + 1
            si ∉ debfile alors FF3 = 1 ; I1 de X3 = maxentier
                                                    fsi
   fsi ; finproc ;
```



```
Procédure LI1 ; co il n'y a pas de fichier à lire oc ;

Procédure CI3 ;
         ZT3 I1 = maxentier ;
         choix 3 (X1) ; choix 3 (X2)   fin proc ;

Procédure choix 3 (X) ;
```

I1 de X < ZT3I1	V	F	F
I1 de X = ZT3I1	F	V	V
I2 de X < ZT3I2		V	F
I2 de X = ZT3I2		F	V
I3 de X < ZR3			V
ZT3I1 = I1 de X	1		
ZT3I2 = maxentier	2		
ZT3I2 = I2 de X		1	
ZR3 = maxentier		2	
retable	3	3	
ZR3 = I3 de X			1

```
ft finproc ;
```

Dans ZT_3I_1 et ZT_3I_2 sont rangées les valeurs prises par les indicatifs I_1 et I_2 dans l'enregistrement choisi pour fournir la valeur suivante de I_3 à traiter.

```
Procédure CI4 ;
         ZR4 = I4 de X4 ; ZT4I2 = I2 de X4 ; ZT4I1 = I1 de X4
         finproc ;
```

Procédure CI_2 ;

ZT_2I_1 = maxentier ; ZR_2 = maxentier ;

choix 2 (X_4) ; choix 2 (X_1) ; choix 2 (X_2)

choix 2 (X_3)

finproc ;

Procédure choix 2 (X)

I_1 de X < ZT_2I_1	V	F
I_1 de X = ZT_2I_1	F	V
I_2 de X < ZR_2		V
ZT_2I_1 = I_1 de X	1	
ZR_2 = I_2 de X		1
retable	2	

ft finproc ;

On constate que la procédure choix 2 (X) est exécutée pour tous les fichiers directeurs pour l'indicatif I_2.

Procédure CI_1 ;

ZR_1 = maxentier ;

choix 1 (X_1) ; choix 1 (X_2) ; choix 1 (X_3) ;

choix 1 (X_4) ;

finproc ;

Procédure choix 1 (X) ;

ZR_1 = maxentier ;

si I_1 de X < ZR_1 alors ZR_1 = I_1 de X fsi

finproc ;

Dans le cas que nous traitons, comme les fichiers directeurs pour I_1 sont les mêmes que pour I_2, on peut écrire plus simplement :

Procédure CI_1 ;

$ZR_1 = ZT_2I_1$;

finproc ;

Procédure TR_o ;

Si (FF_1 = 0 ou FF_2 = 0 ou FF_3 = 0 ou FF_4 = 0)

alors TR_o = 0 ;

sinon TR_o = 1 fsi

finproc ;

Procédure TR_1 ;

Si (FF_1 = 0 ou FF_2 = 0 ou FF_3 = 0 ou FF_4 = 0) et

$ZR_1 = ZT_2I_1$

alors TR_1 = 0 ;

sinon TR_1 = 1 fsi

finproc ;

Procédure TR_{21} ;

Si (FF_1 = 0 ou FF_2 = 0) et $ZR_2 = ZT_3I_2$ et $ZR_1 = ZT_3I_1$

alors TR_{21} = 0 ;

sinon TR_{21} = 1 fsi

finproc ;

Procédure TR_{22} ;

Si FF_4 = 0 et $ZR_2 = ZT_4I_2$ et $ZR_1 = ZT_4I_1$

alors TR_{22} = 0 ;

sinon TR_{22} = 1 ;

finproc ;

En CIVA, le programme pourrait donc être décrit ainsi :

```
Module programme ;
   MII_o ; LCI_o ;
   tant que TR_o = 0 faire MTI_1 ;
        tant que TR_1 = 0 faire MTI_2 ;
             tant que TR_21 = 0 faire MT_3 ;
                                      LI_3 ;
                                      CI_3   fp ;

                                 MTF_21 ;
             tant que TR_22 = 0 faire MT_4 ;
                                      LI_4 ;
                                      CI_4   fp ;

                                 MTF_22 ;
                                 LI_2 ;
                                 CI_2    fp ;

                            MTF_1 ;
                            CI_1    fp ;

        MTF_o  finproc ;
```

La solution de cet exemple peut paraître complexe. Ceci est dû au fait qu'un groupement logique d'indicatif I_j est identifié par l'ensemble des indicatifs d'ordre supérieur et non par I_j seul.

1.3.3.5. Organisation dans un cas particulier :

Nous nous plaçons dans le cas où l'indicatif I_j à lui seul permet d'identifier le groupement logique d'indicatif I_j.

Exemple :

Soit le fichier des salariés d'une entreprise dont le traitement nécessite le regroupement des enregistrements par service et par usine.
Les indicatifs seront donc : numéro usine, numéro service, matricule.
On aura un numéro matricule propre pour chaque salarié de l'entreprise, un numéro service propre pour chaque service de l'entreprise, et un numéro usine pour chaque usine.
Le numéro matricule permet donc d'identifier les salariés et le numéro de service les différents services.

Nous allons voir que dans ce cas l'organisation des modules de cinématique des fichiers est beaucoup plus simple.

1.3.3.5.1. Organisation des modules de cinématique des fichiers

Lecture des fichiers :

Un fichier d'indicatif mineur I_j ne sera lu que si la valeur de l'indicatif I_j est égale à la valeur de I_j qui vient d'être traitée et si le fichier n'est pas encore terminé.

Choix de la valeur suivante de l'indicatif à traiter :

Pour déterminer le groupement logique d'indicatif I_j qu'on va traiter, il suffit de choisir la nouvelle valeur de I_j.

Parmi toutes les valeurs de I_j dans les enregistrements présents on choisira la plus petite (resp. la plus grande) si les fichiers sont classés dans l'ordre croissant (resp. décroissant) de cet indicatif ; la valeur de I_j choisie sera rangée dans ZRI_j. On rangera dans ZTI_{j-1} la valeur de I_{j-1} dans l'enregistrement choisi.

Test de rupture : TR_j.

La différence avec le cas général porte sur la deuxième condition d'exécution du module de traitement MTI_j qui devient : $ZRI_j = ZTI_j$.

1.3.3.5.2. Exemple d'organisation

Nous reprenons l'exemple précédemment traité en supposant que chaque indicatif I_1, I_2, I_3, I_4 permet à lui seul d'identifier les groupements logiques de même ordre.

Le schéma d'organisation générale du programme sera le même que précédemment.
Nous décrivons ci-dessous les modules et procédures réalisant la cinématique des fichiers.

```
Procédure LCI0 ;
    premier X1 ; premier X2 ; premier X3 ; premier X4 ;
    FF1 = 0 ; FF2 = 0 ; FF3 = 0 ; FF4 = 0 finproc ;

Procédure LI4 ;
    Si I4 de X4 = ZR4
       alors X4 en X4 + 1 ;
       si ∄ debfile alors FF4 = 1 ; I4 de X4 = maxentier fsi
    fsi ; finproc ;

Procédure LI3 ;
    Si I3 de X1 = ZR3
       alors X1 en X1 + 1 ;
            si ∄ debfile alors FF1 = 1 ; I3 de X1 = maxentier
            fsi
    fsi ;
```



```
    Si I3 de X2 = ZR3
       alors X3 en X3 + 1 ;
            si ∄ debfile alors FF2 = 1 ; I2 de X3 = maxentier fsi
    fsi ; finproc ;

Procédure LI2 ;
    Si I2 de X3 = ZR2
       alors X3 en X3 + 1 ;
            si ∄ debfile alors FF2 = 1 ; I2 de X3 = maxentier fsi
    fsi ; finproc ;

Procédure LI1 ; co pas de fichier à lire oc ; finproc ;
Procédure CI4 ;
    ZR4 = I4 de X4 ; ZT3 = I2 de X4 ;
    finproc ;

Procédure CI3 ;
    ZR3 = maxentier ;
    si I3 de X1 < ZR3 alors  ZR3 = I3 de X1 ;
                             ZT2 = I2 de X1 fsi
    si I3 de X2 < ZR3 alors  ZR3 = I3 de X2 ;
                             ZT2 = I2 de X2 fsi
    finproc ;

Procédure CI2 ;
    ZR2 = maxentier ;
    choix 2 (X1) ; choix 2 (X2) ; choix 2 (X3) ; choix 2 (X4)
    finproc ;

Procédure choix 2 (X) ;
    Si I2 de X ≤ ZR2 alors  ZR2 = I2 de X ;
                            ZT1 = I1 de X fsi
    finproc ;
```

```
Procédure CI1 ;
    ZR1 = maxentier ;
    choix I (X1) ; choix I (X2) ; choix I (X3) ; choix I (X4)
    finproc ;
Procédure choix I (X) ;
    Si I1 de X < ZR1 alors ZR1 = I1 de X fsi
    finproc ;
Procédure TRo ;
    Si (FF1 = 0 ou FF2 = 0 ou FF3 = 0 ou FF4 = 0)
      alors TRo = 0 ;
      sinon TRo = 1 fsi
    finproc ;
Procédure TR1 ;
    Si (FF1 = 0 ou FF2 = 0 ou FF3 = 0 ou FF4 = 0) et ZR1 = ZT1
      alors TR1 = 0 ;
      sinon TR1 = 1 fsi
    finproc ;
Procédure TR21 ;
    Si (FF1 = 0 ou FF2 = 0) et ZR2 = ZT2
      alors TR21 = 0 ;
      sinon TR21 = 1 fsi
    finproc ;
Procédure TR22 ;
    Si FF4 = 0 et ZR3 = ZT3 alors TR22 = 0 ;
                            sinon TR22 = 1 fsi
    finproc ;
```


En CIVA, le programme correspondant serait décrit ainsi :

```
Procédure programme ;
    MIo ; LCIo ;
    tantque TRo = 0 faire MTI1
          tantque TR1 = 0 faire MTI2 ;
                tantque TR21 = 0 faire MT3 ;
                                       LI3 ;
                                       CI3  fp ;
                                 MTF21 ;
                tantque TR22 = 0 faire MT4 ;
                                       LI4 ;
                                       CI4  fp ;
                                 MTF22 ;
                                 LI2 ;
                                 CI2  fp ;
                           MTF1 ;
                           CI1  fp ;
    MTFo finproc ;
```

Nous constatons que l'organisation générale du programme est identique dans les deux cas.
La différence porte uniquement sur le contenu des modules de cinématique des fichiers qui est beaucoup plus simple dans le cas où l'indicatif I_j permet d'identifier à lui seul les groupements logiques qui lui sont attachés.
En programmation, la cinématique des fichiers est souvent réalisée en se plaçant dans ce cas particulier soit que l'hypothèse restrictive sur les indicatifs est vérifiée, soit à tort dans les autres cas au risque d'avoir une cinématique des fichiers erronée.

1.3.4. - Exemples d'utilisation :

Exemple 1 :

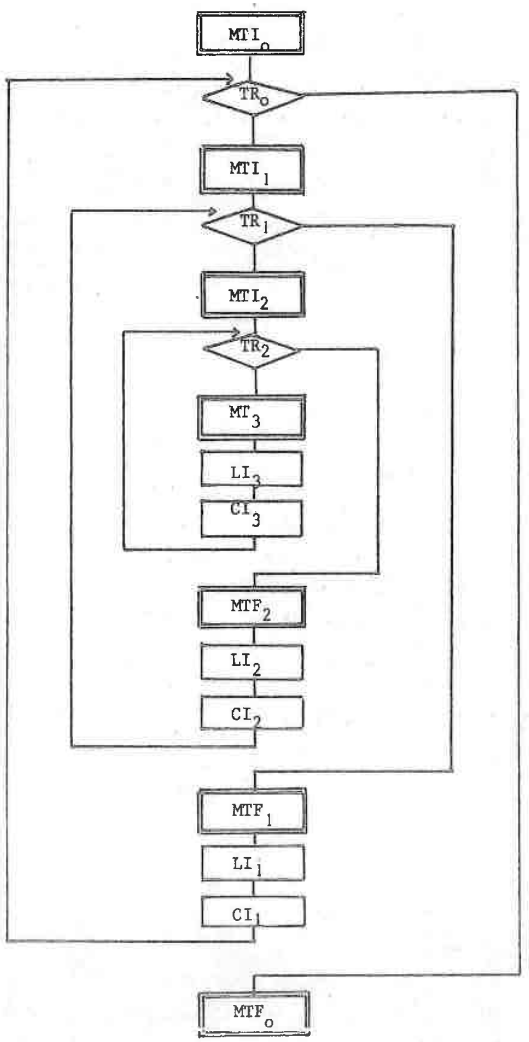
3 niveaux de traitement avec un fichier en entrée :

Soit le fichier des salariés indiquant pour chacun son nom et son salaire net et ayant pour indicatifs le matricule, le code service et le code usine.
On suppose le fichier trié par ordre croissant des indicatifs.
On veut obtenir en sortie un fichier contenant :

- pour chaque salarié la décomposition de son salaire en billets de 500 F, 100 F, 50 F, 10 F et en pièces de 5 F, 1 F, 0,50 F.

- pour chaque service et pour chaque usine, le cumul des salaires versés.

L'organigramme d'exécution sera le suivant :



Les modules de lecture LI_1 et LI_2 sont vides, car le seul fichier de ce programme sera lu en LI_3.

Toutes les valeurs des indicatifs seront fournies par l'unique fichier en entrée.

Le programme CIVA correspondant sera fort simple et se présentera ainsi.

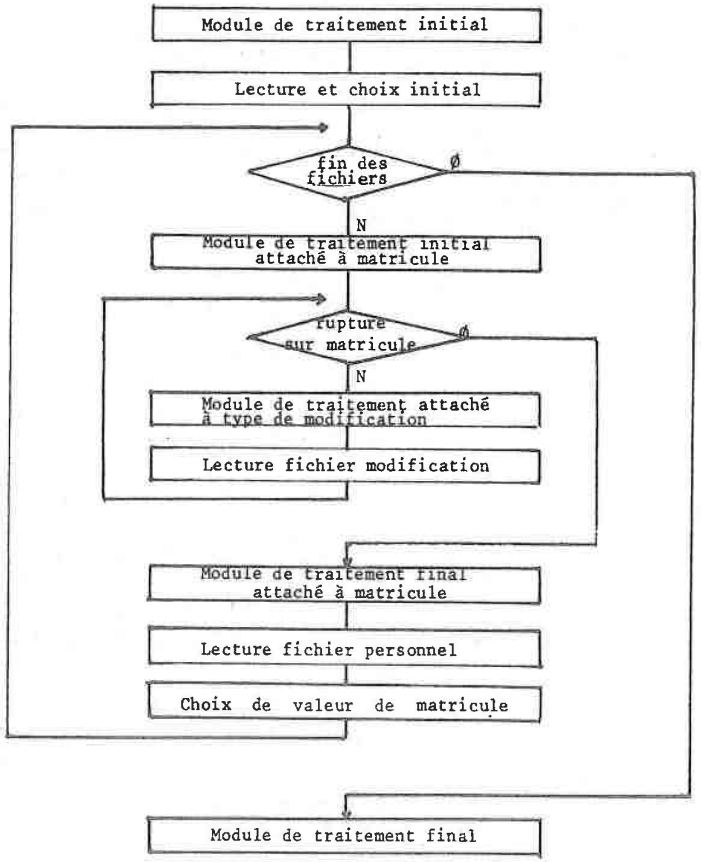
```
                Module traitement statistiques ;

MTI0    { utilise info ; co info est une classe contenant la description des
        { fichiers salariés et sorties et des variables de travail

        { premier X de salarié ; FF1 = 0 ;
LCI0    { ZR1 = usine de X ; ZR2 = service de X ;
        { ZR3 = matricule de X ;

TR0     { tantque FF1 = 0 faire

MTI1    {       total usine = 0 ;

TR1     {       tantque ZR1 = usine de X et FF1 = 0  faire

MTI2    {             total service = 0 ;

TR2     {             tantque ZR2 = service de X et FF1 = 0 faire

        {                   total service = total service + salaire de X ;
MT3     {                   décompte ; co procédure qui calcule le décompte
        {                   monétaire et construit le fichier de sortie oc ;

        {                   X en X + 1
LI3     {                   si ∄ debfile alors FF1 = 1 ;
                                        matricule de X = maxentier

CI3     {                   fsi ;
        {                   ZR3 = matricule de X       fp ;

        {             Cumul service de Y = total service ;
MTF2    {             service de Y = service de X ;
        {             total usine = total usine + total service ;

CI2     {             ZR2 = service de X   fp ;

        {       Cumul usine de Y = total usine ;
MTF1    {       usine de Y = usine de X ;
        {       Y en Y + 1 ;

CI1     {       ZR1 = usine de X   fp ;

MTF0    { finmodule ;
```


Dans cet exemple, la zone de référence I_3 est inutile, car il n'y a qu'un fichier au niveau I_3 qui sera donc lu à chaque itération sans condition restrictive.

Exemple 2 :

Mise à jour d'un fichier personnel à partir d'un fichier des modifications :

Le fichier personnel contenant par salarié son nom, prénom, adresse, sa situation dans l'entreprise, sa situation familiale, aura pour indicatif le matricule.
Le fichier modification comportant des adjonctions, des suppressions et des modifications des enregistrements du fichier personnel aura pour indicatifs le matricule et le type de modification.
On suppose qu'on peut avoir une adjonction suivie d'une ou de plusieurs modifications.
Les fichiers sont triés par matricule et type de modification croissants. Le fichier personnel mis à jour sera écrit dans fichier personnel nouveau.



Module mise à jour ;

Module traitement initial
{ utilise données ; co la classe données contient la déclaration des fichiers personnel, modification, nouveau personnel et des variables : ZR_1, FF_1, FF_2, SUPPRIME, MØD, STRUCT
N de nouveau personnel ;

Lecture et choix initial
{ premier P de personnel ; premier M de modification ;
$FF_1 = 0$; $FF_2 = 0$; co FF_1 et FF_2 sont attachés respectivement à personnel et à modification oc ;
Si matricule de P < Matricule de M
alors ZR_1 = Matricule de P ;
sinon ZR_1 = Matricule de M fsi ;

Fin des fichiers
{ tantque $FF_1 = 0$ ou $FF_2 = 0$ faire

Traitement initial matricule
{ SUPPRIME = F ; MØDIF = F
cø SUPPRIME, variable booléenne vraie si suppression ;
MØDIF variable booléenne vraie si modification oc ;

Rupture sur matricule
{ tantque ZR_1 = matricule de M et $FF_2 = 0$ faire

Traitement de modification
{ Modification ; co procédure de mise à jour ;
enregistrement personnel modifié ou créé est rangé dans STRUCT ; SUPPRIME et MODIF sont positionnés oc ;

Lecture fichier modification
{ M en M + 1 ;
Si ∄ debfile alors FF_2 = 1 fsi ;

Traitement final de Matricule
{ Si MØDIF alors si non SUPPRIME
alors N = STRUCT ;
N en N + 1 fsi ;
sinon N = P ; N en N + 1
fsi ;

Lecture fichier personnel
{ Si matricule de P = ZR_1
alors P en P + 1 ;
si ∄ debfile alors FF_1 = 1 ;
matricule de P = maxentier
fsi ;
fsi ;

Choix de valeur de matricule

```
Si matricule de P < matricule de M
   alors ZR1 = matricule de P ;
   sinon ZR1 = matricule de M  fsi ;
```

Traitement final

```
finmodule ;
```

Commentaires :

Il n'y a pas de choix de valeur pour type de modification car il n'y a qu'un fichier directeur pour cet indicatif.

1.3.5. - Conclusion :

En programmation de gestion, les problèmes essentiels sont ceux liés à la cinématique des fichiers :

où et quand lire les fichiers ?
où et comment choisir la valeur des indicatifs à traiter ?
où et comment réaliser les ruptures de niveau ?

Bien souvent, dans la pratique, on résout ces problèmes de manière empirique et par "approximations successives".
Dans ce chapitre, nous avons présenté des outils permettant de résoudre systématiquement ces problèmes, connaissant certains paramètres :

. la structure des informations en entrée,
. les fichiers et leurs indicatifs.

Nous avons pu constater, dans le cadre de notre enseignement, que leur utilisation diminue les erreurs de programmation commises et permet une analyse modulaire et beaucoup plus rapide des problèmes traités.

Chapitre 2

LA CINEMATIQUE AUTOMATIQUE DES FICHIERS

LA CINEMATIQUE AUTOMATIQUE DES FICHIERS

2.1. - INTRODUCTION

Nous venons de présenter quelques notions qui permettent de décrire les problèmes liés à la cinématique des fichiers. Il nous a semblé que l'utilisation de ces notions devait permettre de réaliser la cinématique automatique des fichiers pour en libérer totalement le programmeur, qui n'aurait plus qu'à écrire les modules de traitement proprement dits.

Ceci peut être réalisé de deux manières différentes :

- soit concevoir un langage de programmation utilisant les concepts précédemment définis : notion d'indicatif, notion de groupement logique, notion d'ordre de traitement, et possédant les instructions adéquates pour les mettre en oeuvre.

  Cette solution a été adoptée dans le langage de programmation CIVA.

- soit compléter un langage existant par un métalangage spécialement conçu pour résoudre ces problèmes et qui sera traduit par un générateur en une suite d'instructions appartenant à ce langage de programmation.

  Nous avons réalisé un tel générateur en Cobol.

2.2. - LA CINEMATIQUE AUTOMATIQUE DANS CIVA (DERNIAME J.C. (1))

2.2.1. - La déclaration des indicatifs :

Si l'on veut attacher une liste d'indicatifs à un fichier pour toute la durée de vie de celui-ci (c'est le cas le plus général), on les déclare dans la même classe que ce fichier ou dans une classe ayant même durée de vie.

Si, au contraire, les traitements concernant un fichier ne considèrent pas toujours les indicatifs dans le même ordre, ou n'associent pas toujours les mêmes indicatifs au fichier, il est préférable de faire la déclaration des indicatifs dans les modules de traitement concernés.

Une déclaration d'indicatifs s'écrit :

[< indicatif> [<sens>]] [,<indicatif> [<sens>]]* indicatif de
<identificateur de fichier> .

<indicatif> :: = <identificateur> .

<sens> :: = croi | décroi.

La déclaration d'indicatifs indiquera que I, champ d'une structure déjà déclarée est un indicatif pour le fichier désigné et qu'on l'utilisera pour découper ce fichier en groupements logiques. Un sens peut être associé à chaque indicatif pour indiquer dans quel ordre on veut traiter ces groupements logiques.

Si l'indicatif ne constitue pas un critère dans le préordre associé au fichier ou si le sens n'est pas le même, on triera le fichier en prenant l'indicatif comme critère. L'ordre des indicatifs dans la déclaration définit l'ordre des groupements logiques dans le fichier désigné. La déclaration d'indicatifs se fait dans une classe ou dans un module. Une déclaration d'indicatifs pour un fichier annule et remplace la déclaration précédente.

Il existe une forme simplifiée de la déclaration :

indicatif de < identificateur d'ensemble >.

Elle permet de déclarer pour un fichier donné une liste vide d'indicatifs en remplacement des indicatifs déjà déclarés.

Cette suppression d'indicatifs sera locale à l'unité dans laquelle se trouve cette déclaration.

Ceci permet donc de déclarer des indicatifs différents pour le même fichier suivant les traitements qui sont envisagés.

2.2.2. - Instruction d'itération logique :

Elle traduit la structure de l'organigramme d'exécution et décrit les traitements à effectuer sur les groupements logiques définis sur les différents fichiers en entrée.
La syntaxe de cette instruction sera :

pour chaque <identificateur d'indicatif> <fin de pour>
< fin de pour> :: = faire < instruction > [; < instruction>]*
fpc

Ainsi l'instruction d'itération logique :

pour chaque k faire i1 ; i2 ; ..... ; in fpc ;

permet de demander l'exécution des instructions situées entre faire et fpc pour chacun des groupements logiques d'indicatif k définis sur le ou les fichiers intéressés.

En effet, lorsqu'on utilise l'instruction d'itération logique, on n'indique pas le nom des fichiers sur lesquels porte l'itération, mais uniquement l'indicatif auquel est attaché cette instruction.
Au moment de l'évaluation de l'instruction d'itération logique, il faudra déterminer quels sont, parmi les fichiers pouvant être accédés,

ceux qui ont k comme indicatif et auxquels s'appliquera l'instruction d'itération logique.
Comme les autres instructions d'itération dans CIVA, les instructions d'itération logique peuvent être imbriquées sans se chevaucher.
L'instruction externe correspond à un indicatif de niveau supérieur à celui de l'instruction interne.

2.2.2.1. Instruction d'itération logique simple :

On dira qu'une instruction d'itération est une instruction d'itération logique simple si ses indicatifs définissent des groupements logiques dans un seul fichier.
Elle va vérifier que le fichier est rangé selon un préordre compatible avec les indicatifs utilisés.
Si les indicatifs sont aussi des critères de tri et si les sens indiqués sont identiques, il n'y a pas de tri du fichier, sinon on effectue un tri pour ranger les enregistrements du fichier selon le préordre donné par les indicatifs et leur sens associé.

Le fichier sera lu dans l'itération la plus interne.

2.2.2.2. Instruction d'itération logique multiple :

On dira qu'une instruction d'itération est une instruction d'itération logique multiple si ses indicatifs définissent des groupements logiques dans plusieurs fichiers.
Elle va vérifier que les fichiers attachés aux indicatifs de l'instruction d'itération sont triés selon des préordres compatibles. Si cela n'est pas vérifié, on effectuera les tris nécessaires en prenant comme critères les indicatifs et leur sens associé.
L'ordre des critères sera l'ordre de présence des indicatifs sur les différentes branches de l'arbre de traitement.
Chaque fichier sera lu dans l'itération attachée à l'indicatif mineur de ce fichier.

2.2.3. - Exemples d'utilisation :

Exemple 1 :

Soit un fichier des salariés contenant pour chaque salarié : numéro usine, numéro service, matricule, nom, prénom, salaire net.

On veut éditer un état statistique donnant, pour chaque usine, pour chaque service, la masse des salaires versés et pour toute la société, le cumul général.

```
Classe  Info ;
début service struct (libel 1 file (10) car (service n° :) ;
                      numéro 1 file (3) entier) ;
fin service   struct (libel 2 file (14) car (total service :) ;
                      cumul service décimal 9 (6) V 9 (2)) ;
début usine   struct (libel 3 file  (9) car (usine n° :) ;
                      numéro 3 file (2) entier) ;
fin usine     struct (libel 4 file (12) car (total usine :) ;
                      cumul usine décimal 9 (8) V 9 (2)) ;
fin générale  struct (libel 5 file (14) car (total général :) ;
                      cumul général décimal 9 (8) V 9 (2) ;

Fichsal ensemble struct (usine file (2) entier ; service file (3)
        entier ; lignesol struct (matricule file (5) entier ; nom
        file (20) car ; prenom file (15) car ; salaire
        décimal 9 (5) V 9 (2))) ;
ligne struct (matric file (5) entier ; nom 1 file (20) car ;
             prénom 1 file (15) car ; salaire 1 décimal 9 (5) V 9 (2)) ;
fin classe ;

Module cumuls ;
   utilise Info ; X de FICHSAL ;
   usine croi, service croi, matricule croi indicatif de fichsal ;
   cumul général = 0 ;
```

```
Pour chaque usine faire
        cumul usine = 0 ;
        numéro usine = usine ;
        éditer (début usine) ;
      Pour chaque service faire
            cumul service = 0 ;
            numéro service = service
            éditer (début service) ;
          Pour chaque matricule faire
                ligne = lignesal ;
                cumul service = cumul service + salaire de X
                                                        fpc ;
            éditer (fin service) ;
            cumul usine = cumul usine + cumul service fpc ;
      éditer (fin usine) ;
      cumul général = cumul général + cumul usine  fpc ;
éditer (fin générale) fin module ;
```

Exemple 2 :

On veut éditer le même état qu'auparavant, sauf pour l'en-tête service où l'on veut indiquer le nom du service et du chef de service.
On aura plusieurs fichiers en entrée.
Un fichier donnant pour chaque service le numéro usine, le numéro service, le nom du service, le nom du chef de service.
Deux fichiers salariés ayant la même structure que le fichier unique précédent.
Le partage entre les deux fichiers se fait sur le critère du service.

```
Module cumuls ;
   utilise Info ;
      fichsal 1 ensemble type fichsal ;
      fichsal 2 ensemble type fichsal ;
      fichser ensemble struct (usine file (2) car ; ligneser struct (service
            file (3) car ; nomser file (10) car ; nom chef file (20) car)) ;
      F 1 de fichsal 1 ; F 2 de fichsal 2 ; S de fichser ;
```

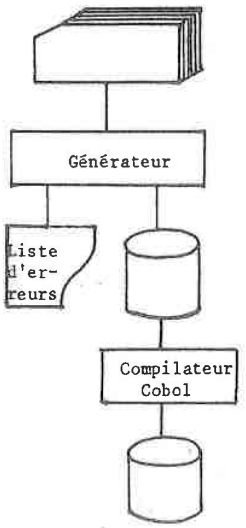


```
usine croi, service croi, matricule croi, indicatif de fichsal 1 ;
usine croi, service croi, matricule croi, indicatif de fichsal 2 ;
usine croi, service croi  indicatif de fichser ;

Pour chaque usine faire
        cumul usine = 0
        si usine de F 1 ⩽ usine de F 2
            alors numéro usine = usine de F 1 ;
            sinon numéro usine = usine de F 2  fsi ;
        éditer (début usine) ;
        cumul usine = 0 ;
      Pour chaque service faire
          si service de S = service de F 1 ou service de S = service de
                                                              F 2
              alors éditer (ligneser) ;
                   cumul service = 0 fsi ;
            Pour chaque matricule faire
                si matricule de F 1 < matricule de F 2
                   alors ligne = lignesal de F 1 ;
                         cumul service = cumul service + salaire de F 1 ;
                   sinon ligne = lignesal de F 1 ;
                         cumul service = cumul service + salaire de F 2 ;
                         ligne = lignesal de F 2
                fsi fpc ;
            éditer (fin service) ;
            cumul usine = cumul usine + cumul service fpc ;
      éditer (fin usine) ;
      cumul général = cumul général + cumul usine fpc ;
éditer (fin générale) fin module ;
```

L'introduction de deux fichiers supplémentaires a légèrement compliqué la partie traitement de notre programme.
En ce qui concerne la cinématique des fichiers, elle a nécessité la déclaration des deux nouveaux fichiers et des indicatifs associés à ces fichiers, mais elle n'a nullement modifié l'instruction d'itération logique. Or il est évident que la cinématique des fichiers dans ce cas est nettement plus complexe que dans l'exemple avec un fichier, et dans les langages de programmation usuels, cela aurait nécessité un effort d'analyse et de programmation non négligeable.

2.3. - CINEMATIQUE AUTOMATIQUE DES FICHIERS EN COBOL

2.3.1. - Schéma général de réalisation :



Programme Cobol initial + texte en métalangage cinématique.

Programme Cobol final (comprenant les instructions de cinématique générées).

Programme traduit et prêt à être exécuté.

Nous allons étudier les conditions d'utilisation et le principe de fonctionnement du générateur.

2.3.1.1. Les données en entrée du générateur :

Les données transmises au générateur le sont dans l'ordre présenté ci- dessous :

IDENTIFICATION DIVISION

DATA DIVISION

PROCEDURE DIVISION } Déclarations et instructions du métalangage.

MODULE INITIAL

STOP RUN. } Modules de traitement écrits par l'utilisateur.

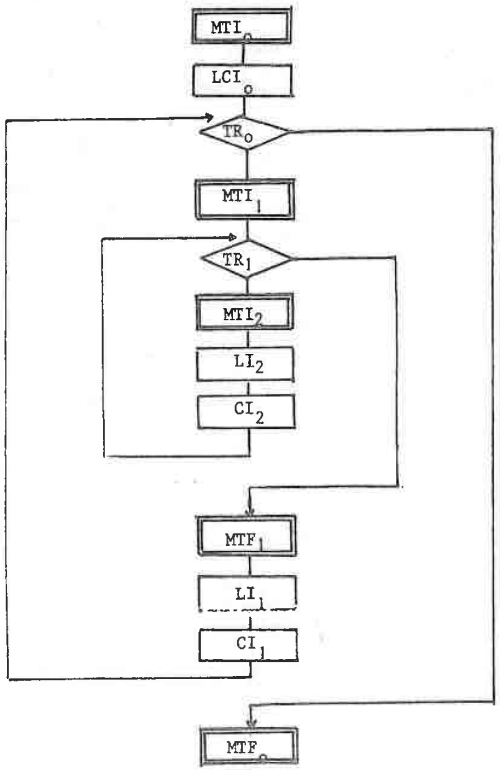
Le générateur lit et copie sur un fichier disque les cartes contenant l'IDENTIFICATIØN DIVISIØN et la DATA DIVISIØN.
Puis, après la carte PRØCEDURE DIVISIØN, il analyse les cartes contenant les métadéclarations et les métainstructions.
L'analyse étant terminée, il génère les instructions Cobol correspondantes et les copie sur le fichier disque.
Puis il lit les cartes Cobol contenant les instructions de traitement et les ajoute au fichier disque. Simultanèment à l'analyse du métalangage, le générateur imprime la liste des erreurs de syntaxe détectées. La phase génération n'est entreprise que s'il n'y a aucune erreur de syntaxe.

Le programme Cobol écrit sur le disque peut être compilé sans aucune autre intervention.

Nous verrons dans le chapitre 3 la conception et la réalisation du générateur de cinématique des fichiers.

2.3.1.2. Organisation du programme Cobol généré :

Soit un programme dont l'organigramme d'exécution se présente ainsi :



Les modules encadrés d'un double trait sont les modules de traitement écrits par l'utilisateur en Cobol.

Les modules encadrés d'un trait simple et les tests TR_0 et TR_1 sont générés à partir du métalangage.

Le programma Cobol final (généré) correspondant à cet organigramme se présentera ainsi :

```
IDENTIFICATIØN DIVISIØN.
DATA DIVISIØN.
WØRKING-STØRAGE SECTIØN.
                                  } Génération de la déclaration de variables
                                    locales à la partie du programme Cobol
                                    générée.
PRØCEDURE DIVISIØN.
```


Instructions Cobol générées à partir du métalangage.

```
PRØ0. PERFORM MTI0 THRU F-MTI0
      co lecture initiale de tous les fichiers, choix initial des
      valeurs à donner aux indicatifs, initialisations de variables
      propres à la partie générée du programme Cobol oc.
TR0 . co test de fin des fichiers et exécution des modules MTI1 ou
      MTFØ selon résultat du test oc.
LI1 . co lecture de tous les fichiers attachés à I1 oc.
CI1 . co choix de la nouvelle valeur de I1 à traiter oc.
RT1 . GØ TØ TR0.
TR1 . co test de rupture attaché à I1 et exécution des modules MTI2
      ou MTF  selon le résultat du test oc.
LI2 . co lecture des fichiers attachés à I2 oc.
CI2 . co choix de la nouvelle valeur de I2 à traiter oc.
RT2 . GØ TØ TR1.
```

Paragraphes écrits par utilisateur.

```
MTI0 . co le module MTI0 est constitué de tous les paragraphes compris
       entre MTI0 et F-MTI0 oc.
F-MTI0. EXIT.
MTI1 . co de module MTI1 est constitué de tous les paragraphes compris
       entre MTI1 et F-MTI1 oc.
F-MTI1. EXIT.
  ¦
  ¦
  ¦
```

2.3.1.2.1. Working-storage Section

Elle comportera un certain nombre de lignes générées.
La génération des déclarations de variables a pour but de déclarer toutes les variables locales à la partie générée de la "procédure division". On y trouvera en particulier la déclaration des variables du type ZRIj, ZTIj et FFn (code de fin de fichier). Ces variables seront totalement ignorées du programmeur qui n'en aura jamais besoin dans les modules de traitement écrits par lui.

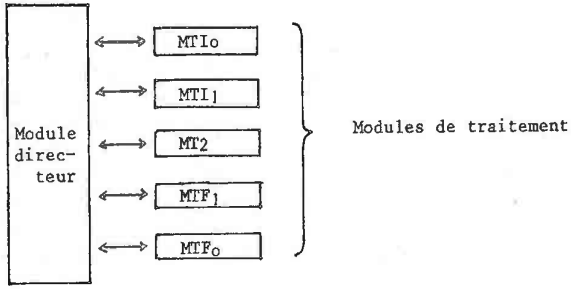
2.3.1.2.2. Procédure division

Elle est divisée en deux parties dont la première est générée et la deuxième est écrite par le programmeur.

. Le module directeur :

La partie générée constitue le module directeur du programme, car en plus de la cinématique des fichiers, elle réalise le contrôle de l'exécution du programme.

Ceci peut être schématisé ainsi :



Le module directeur constitue le squelette du programme chargé des phases communes, il appelle les différents modules de traitement pour exécution.
Sa structure traduit fidélement la structure de l'organigramme d'exécution.
Les modules de traitement sont élaborés sous forme de sous-programmes fermés ; le contrôle est donné au module de traitement depuis le module directeur.
Il est composé des modules réalisant la cinématique des fichiers : LI_j, CI_j, TR_{j-1} et LCI_o (dans notre exemple j allant de 1 à 2).
Chacun de ces modules constitue dans le programme Cobol généré un paragraphe dont le nom sera le nom du module.

Nous avons ajouté des paragraphes de nom RT_j, afin que les paragraphes de nom CI_j ne contiennent pas d'instruction de branchement GØ TØ et comportent uniquement les instructions effectuant le choix de la nouvelle valeur de I_j à traiter.

. Les modules de traitement :

Les modules de traitement sont élaborés sous forme de sous-programmes fermés ; le contrôle est donné au module de traitement depuis le module directeur et rendu, après exécution, au module directeur.
Un module de traitement sera constitué d'un paragraphe ou d'une suite de paragraphes dont le premier et le dernier ont leurs noms déclarés dans le métalangage.

Cette contrainte nous a été imposée par les conditions d'utilisation de l'instruction d'appel d'un sous-programme fermé en Cobol :

PERFØRM nom de paragraphe THRU nom de paragraphe.

2.3.2. - Définition du métalangage :

Le métalangage que nous avons défini est indépendant du contexte dans lequel il est utilisé. En conséquence, il ignore toutes les déclarations et instructions figurant dans le programme Cobol dans lequel il est utilisé.
Ceci a un avantage certain dans la mesure où le métalangage pourra être utilisé pour générer la cinématique des fichiers dans d'autres langages de programmation que le Cobol. Il suffira alors de modifier uniquement la partie génération tout en conservant la partie analyse syntaxique du générateur.

L'inconvénient qui en découle est qu'il faudra définir dans le cadre du métalangage tous les éléments nécessaires pour générer des instructions Cobol compatibles avec le reste du programme. Ceci implique la déclaration dans le métalangage des fichiers et des indicatifs bien que cela fasse double emploi avec les déclarations faites dans la "file section" du programme Cobol initial.

2.3.2.1. Déclaration des fichiers et des indicatifs :

Pour un ou plusieurs fichiers ayant mêmes indicatifs, on aura la déclaration suivante :

<déclaration indicatifs> :: = INDICATIF DE <fichier> [<fichier>]*, <indicatif> [,<indicatif]*;

<fichier> :: = <identificateur de fichier> [(identificateur d'enregistrement)]

<indicatif> :: = <identificateur> [<sens>] [<taille>]

<sens> :: = croi | décr

<taille> :: = X (<n>)

<n> :: = <entier>

Les identificateurs des fichiers, des enregistrements et des indicatifs déclarés dans le métalangage devront être identiques à ceux déclarés dans la "file section" du programme Cobol.

L'identificateur d'enregistrement est optionnel et n'est utilisé que dans les versions de Cobol où l'on ne peut pas prendre le nom du fichier comme qualificateur.
De plus, la longueur maximale autorisée pour les identificateurs sera de 10 caractères.
On ne déclare que les fichiers pour lesquels on veut la génération de la cinématique automatique, tous les autres fichiers devront être gérés par le programmeur.
La taille de l'indicatif est fournie dans la déclaration afin de réserver des zones de travail en "Working-storage Section" pour les variables locales au module directeur.
Pour des raisons de compatibilité, l'indicatif devra être défini ou redéfini dans la "file section" comme étant du type alphanumérique.

La déclaration d'un indicatif I pour un fichier F aura pour effet d'indiquer que I champ d'un enregistrement déjà déclaré (en "file section") est un indicatif pour le fichier désigné et permet de diviser ce fichier en groupements logiques.
Il faudra que le programmeur veille à ce que les groupements logiques du fichier soient rangés dans l'ordre selon lequel on veut les traiter.
Le groupement logique de plus bas niveau, c'est-à-dire celui attaché à l'indicatif mineur du fichier constitue l'enregistrement du fichier.

Exemple :

Indicatif de FICHSAL, service croi X (3), matricule croi X (7) ;

2.3.2.2. Instruction d'itération logique :

Elle permet de traduire la structure de l'organigramme d'exécution du programme et de définir les modules de traitement écrits par l'utilisateur en Cobol.
Elle s'écrit :

<module> <structure itérative> <module>

<structure itérative> :: = Pour chaque <indicatif> faire <liste d'éléments pour > fpc

<liste d'éléments pour> :: = <module> | <module> pour chaque <indicatif> faire <liste d'éléments pour> fpc
<module> | <module> pour chaque <indicatif> faire <liste d'éléments pour> fpc [[<module>]
pour chaque <indicatif> faire <liste d'éléments pour> fpc]* <module>

<indicatif> :: = <identificateur>

<module> :: = <identificateur> [jusque < identificateur>].

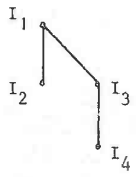
Cette instruction permet de construire l'arbre de traitement, associé au programme, qui définit une relation d'ordre entre les différents indicatifs. Elle associe à chaque indicatif les modules de traitement qui sont exécutés pour chaque réalisation de cet indicatif.
Un module est défini par les noms du premier et du dernier paragraphe de la suite des paragraphes le constituant.

Exemple :

Soit l'instruction d'itération logique suivante :

```
Pour chaque I1 faire init 1
        Pour chaque I2 faire trait 2  fpc
        Pour chaque I3 faire init  3
                Pour chaque I4 faire trait 4   fpc
                        fin 3   fpc
                fin 1   fpc
```

L'arbre de traitement correspondant sera :



Nous verrons dans le chapitre suivant l'algorithme nous permettant de construire cette arborescence.

Elle permet également d'indiquer le ou les deux modules de traitement attachés à chaque indicatif.
Ainsi, si nous prenons l'exemple ci-dessus, on constate que traitement2 est attaché à l'indicatif I_2, c'est-à-dire que pour chaque valeur de I_2 on exécutera le module de traitement "trait2".

L'instruction d'itération logique est composée d'instructions d'itération qui peuvent être imbriquées, mais ne doivent pas se chevaucher. L'instruction externe correspond à un indicatif de niveau supérieur à celui de l'instruction interne.
Soit l'instruction d'itération associée à l'indicatif I_1. Pour chaque valeur de I_1 on exécutera le module de traitement init 1, puis on exécutera toutes les instructions d'itération internes (celles associées aux indicatifs I_2, I_3 et I_4), enfin, on exécutera fin 1 qui est le deuxième module de traitement attaché à l'indicatif I_1.

2.3.2.3. Exemples d'utilisation :

Exemple 1 :

Nous allons reprendre l'exemple que nous avions traité dans "la cinématique automatique dans CIVA" et réalisant un état statistique des salaires à partir d'un fichier.
Nous ne présentons ici que la "procédure division" du programme Cobol initial où se trouvent les métadéclarations et les métainstructions..

```
Métalangage.
        Procédure division
        Indicatif de Fichsal, usine croi X (2), service
                croi X (3), matricule croi X (5) ;
        mtio jusque f-mtio
        Pour chaque usine faire mti 1 jusque f-mti 1
                Pour chaque service faire mti 2 jusque f-mti 2
                        Pour chaque matricule faire mt 3 fpc
                                mtf 2 fpc
                        mtf 1 fpc
        mtfo ;
```

Modules de traitement Cobol.

mtio. co ce module effectue les initialisations du programme oc.

f-mtio.exit.

mti1. co ce module effectue initialisations usine oc.

f-mti1.exit.

mti2. co ce module effectue initialisations service oc.

f-mti2.exit.

mt3. co ce paragraphe effectue traitement d'un salarié oc.

mtf2. co édition cumul service oc.

mtf1. co édition cumul usine oc.

mtfo. co édition cumul général oc.

Exemple 2 :

Nous allons réaliser le même état statistique, mais en utilisant 3 fichiers (2 fichiers salariés et un fichier service) :

```
Procédure division

Indicatif de fichsal 1, usine croi X (2), service croi
            X (3), matricule croi X (5) ;

Indicatif de fichsal 2, usine croi X (2), service croi
            X (3), matricule croi X (5) ;

Indicatif de fichser, usine croi X (2), service croi X (3) ;
mtio jusque f-mtio

Pour chaque usine faire mti 1 jusque f-mti 1

          Pour chaque service faire mti 2 jusque f-mti 2

                    Pour chaque matricule faire mt 2 fpc

                              mtf 2 fpc

                         mtf 1 fpc

mtfo ;
```


Les modules de traitement Cobol écrits par l'utilisateur auraient la même organisation que précédemment.
Nous vérifions par cet exemple que le fait de traiter le même problème avec trois fichiers au lieu d'un ne modifie en rien l'énoncé de l'instruction d'itération logique et l'organisation des modules de traitement écrits en Cobol.

Il suffit de déclarer dans le cadre du métalangage les fichiers utilisés et leurs indicatifs.

Le générateur fera le lien entre les indicatifs et les fichiers.

Nous constatons, par cet exemple, que l'utilisation du métalangage permet une programmation totalement indépendante du découpage en fichiers des informations traitées.

Chapitre 3

LE GENERATEUR DES INSTRUCTIONS DE CINEMATIQUE DES FICHIERS EN COBOL

LE GENERATEUR DES INSTRUCTIONS DE CINEMATIQUE DES FICHIERS EN COBOL

3.1. - INTRODUCTION

Le générateur est un programme qui analyse les instructions et les déclarations du métalangage insérées dans la "procédure division" d'un programme Cobol et génère les instructions "Cobol" correspondantes.

Le langage de programmation utilisé pour écrire le générateur est le Cobol. Nous avons choisi de l'écrire en langage évolué afin que son utilisation sur différents ordinateurs soit possible sans avoir à réécrire complètement le programme.

Actuellement, il est opérationnel sur CII 10070.
Nous ne donnerons pas ici une analyse complète du générateur, nous nous limiterons à certains modules qui montrent bien son fonctionnement et qui seront écrits en CIVA pour en faciliter la lecture.

Pour l'analyse complète et détaillée, on pourra consulter FIEGEL C. (9) qui décrit la réalisation de ce générateur.

3.2. - DESCRIPTION DU GENERATEUR

```
Module générateur ;
    utilise table fichiers, table indicatifs.
        co il comporte deux parties :
            . la première analyse les métadéclarations et la métainstruction et
              construit une table des fichiers, une table des enregistrements
              et une table comportant pour chaque indicatif différents
              renseignements.
            . la deuxième génère à partir de ces tables les instructions Cobol
              oc ;
    analyse des métadéclarations ;
    analyse de la métainstruction ;
    génération ;
fin module ;
```

3.2.1. Module analyse des métadéclarations ;

co Pour chaque fichier on note son nom ainsi que celui de ses enregistrements dans la table des fichiers et dans la table des enregistrements. Pour chaque indicatif on note dans la table des indicatifs son nom, sa longeur, son sens, les fichiers qui le possèdent, et les fichiers pour lesquels il est l'indicatif mineur oc.

3.2.2. Module analyse de la métainstruction ;

```
utilise travail ;
```

co On complète la table des indicatifs par les noms des modules attachés aux indicatifs. On construit l'arbre de traitement du programme sur l'ensemble des indicatifs en utilisant la représentation par un double chaînage (lien vertical et lien horizontal) (PAIR (3)).
On appelle lien vertical de x : lv (x) le premier point de la liste des successeurs de x.
On appelle lien horizontal de x : lh (x) le point suivant dans la liste des successeurs où figure x.

Comme les indicatifs, lv et lh sont représentés par des tableaux à une dimension, les liens de chaînage seront les indices de lignes auxquels ils renvoient. Pour l'analyse et l'établissement du double chaînage, nous utiliserons une pile oc ;

```
zone file car ;
remplir (zone) ; co ce module range les éléments constitutifs de la
                    métainstruction dans "zone" oc ;
```

Sélection

zone =	Pour chaque <identificateur>	faire <module> fpc	faire <module>	<module> fpc	<module>
Actions	Traitement pour	traitement faire 1	traitement faire 2	traitement module 1	traitement module 2

ft ;

```
co <module> : = identificateur 1 [ jusque identificateur 2 ]
oc ;
fin module ;
module traitement pour ;
    recherche (ni) ; co on recherche l'indice de l'identificateur dans la
                       table des indicatifs oc ;
    si h ≠ 0 alors si dp = vrai alors i = pile (h) ;
                                        lv (i) = ni ;
                  fsi ;
                                  sinon lh (i) =ni ;
                                        dp = vrai
    fsi ;
    h = h + 1 ; pile (h) = ni ;
fin module ;
module traitement faire 1 ;
    traitement faire 2 ;
    dp = faux ; i = pile (h) ;
    h = h - 1 ;
fin module ;
```

```
module traitement faire 2 ;
        tmodinit 1 (ni) = identificateur 1 ;
        si identificateur 2 existe alors tmodinit 2 (ni) = identificateur 2
                                                                    fsi ;
fin module ;
module traitement module 2 ;
        tmodinter 1 (ni) = identificateur 1 ;
        si identificateur 2 existe alors tmodinter 2 (ni) = identificateur 2
                                                                    fsi ;
fin module ;
module traitement module 1 ;
        tmodfinal 1 (i) = identificateur 1 ;
        si identificateur 2 existe alors tmodfinal 2 (i) = identificateur 2
                                                                    fsi ;
        h = h - 1 ;
        i = pile (h)
fin module ;
```

3.2.3. Module génération ;

co il va explorer l'arbre de traitement en utilisant une pile appelée pile attachée à l'information arborescente (PAIR (15)).
L'intérêt de son utilisation est d'avoir dans la pile l'ensemble des indicatifs de niveau supérieur à l'indicatif figurant au sommet de la pile. Or il nous faut tous ces indicatifs pour effectuer les tests de rupture, la lecture des fichiers et le choix de valeur des indicatifs.
Pour chaque noeud de l'arborescence on génère les modules de cinématique attachés à l'indicatif.
Pour ce faire, on utilise un module :générer (x)où x est une file de caractères de taille variable.
Le rôle de ce module est de générer les caractères figurant entre les " " et le contenu des variables identifiées par une suite de caractères entre [ ].

Exemple :
Générer ("GØ TØ [eti] ")
si la variable eti contient les caractères :
r, e, t, o, u, r
on générera : GØ TØ RETØUR oc

```
        utilise travail ;
        génération déclarations ;
                co ce module génère les déclarations des variables locales
                   au module généré dans la "Working-storage section" oc ;
        début programme ;
                co ce module génère toutes les instructions de début de
                   programme : lecture initiale de tous les fichiers, ini-
                   tialisation de toutes les variables locales à la partie
                   générée, choix initial de valeur pour tous les
                   indicatifs oc ;
        i = racine ; h = 0 ;
        exploration arborescence (i) ;
        générer (" GØ TØ [modfinal 1] ") ;
fin module ;
module exploration arborescence (i) ;
        h = h + 1 ;
        pile (h) = i ;
        tr (i) ;
```

Décision				
	lv (i) $\neq$ 0	V	F	F
	lh (i) $\neq$ 0	-	V	F
	h 0	-	-	V
Actions				
	i = lv (i)	1		
	reinst	2	4	
	li (i) h = h - 1 ci (i)		1	1
	inter (i)		2	
	mtf (i)			2
	i = lh (i)		3	
	i = pile (h)			3
	retable			4

Fin table

```
fin module ;
```

```
Module tr (i) ;          co génère les instructions effectuant les tests de fin de
                            fichier et de rupture sur valeur de i et des indicatifs
                            de niveau supérieur ainsi que l'appel des modules de
                            traitement à exécuter oc ;
        x de pile ;
        générer ("TR [ i] ") ;
        premier y de fichdir (i) ;
        générer ("IF (FF [y] = 1") ;
        Pour chaque y de fichdir (i, 2 : 5) tant que y ≠ 0 faire
                    générer ("AND FF [y]  = 1") fpc
        générer (")") ;
        si i ≠ racine alors pour chaque x faire
                    générer ("ØR ZT [i] I [x] ≠ ZR [x] ") ; fpc ; fsi ;
        générer (" GØ TØ ST [ i] .") ;
        générer ("PERFØRM [ tmodinit 1 (i) ] ") ;
        si tmodinit 2 (i) ≠ 0 alors générer ("THRU [ tmodinit 2 (i) ] .") fsi ;
fin module ;
Module li (i) ;          co génère les instructions effectuant la lecture des fi-
                            chiers ayant i comme indicatif mineur oc ;
        z de fichlu (i) ; x de pile ;
        générer ("LI [ i] .")
        si premier z de fichlu  = 0 alors générer ("EXIT.") fsi ;
    Pour chaque z tant que z ≠ 0 faire
        générer ("IF FF [z] = 0");
        pour chaque x faire
            générer ("AND [ indic (x)] ØF [enreg (z)]  = ZR [x] ") fpc ;
        générer ("PERFØRM LF [ z ] ") ;
        premier x ;
        générer ("IF FF [z] = 1 MØVE [valeur (sens (x))] TØ [ indic (x)]
                ØF [enreg (z)] .")
    fpc ;
fin module ;

Module ci (i) ;          co génère les instructions effectuant le choix de la
                            valeur de l'indicatif indic (i) pour le ranger dans
                            ZR (i), ainsi que le choix des valeurs pour les indica-
                            tifs indic (j) de niveau supérieur à indic (i) sur la
                            branche de traitement pour les ranger dans ZTiIj.
```



```
                            Nous ne présentons pas le détail de l'analyse de ce
                            module ; le principe est le même que pour les modules
                            ci (i) et 1i (i) oc ;
fin module ;
Module inter (i)         co génère les instructions d'appel d'un module de traite-
                            ment lorsque indic (i) a un lien horizontal oc ;
fin module
Module mtf (i) ;         co génère les instructions d'appel du module de traitement
                            final attaché à l'indicatif indic (i) oc ;
fin module ;

        classe table fichiers ;
            nbfich entier ;
                        co nombre de fichiers dans la table des fichiers oc ;
            tabfich file (20, 10) car ;
                        co nom des fichiers déclarés oc ;
            enreg file   (20, 10) car ;
                        co nom des enregistrements des fichiers oc ;
        fin classe ;

        classe table indicatifs ;
            nbindic entier ;
                        co nombre d'indicatifs déclarés oc ;
            modinit 1 file (10) car ;
                        co contient le nom du premier paragraphe du module
                           de traitement initial du programme oc ;
            modinit 2 file (10) car ;
                        co contient le nom du dernier paragraphe du module
                           de traitement initial du programme oc ;
            modfinal 1 file (10) car ;
                        co nom du premier paragraphe du module de traitement
                           final du programme  oc ;
            modfinal 2 file (10) car ;
                        co nom du dernier paragraphe oc ;
            racine entier ;
                        co numéro d'indice de la racine de l'arborescence oc ;
            tabindic  file (15) struct (indic file (10)
                        co identificateur de l'indicatif oc
```

```
        l-indic entier,
                co taille de l'indicatif en caractères oc
        ordre entier,
                co = 1 si indicatif par ordre croissant sinon = 2 oc
        lv entier,
                co lien vertical oc
        lh entier,
                co lien horizontal oc
        tmodinit 1 file (10) car,
                co nom du 1er paragraphe du module de traitement
                   initial attaché à l'indicatif i oc
        tmodinit 2 file (10) car,
                co nom du dernier paragraphe oc
        tmodfinal 1 file (10) car,
                co nom du premier paragraphe du module de traitement
                   final attaché à l'indicatif i oc
        tmodfinal 2 file (10) car,
                co nom du dernier paragraphe oc
        tmodinter 1 file (10) car,
                co nom du premier paragraphe du module de traitement
                   intermédiaire attaché à l'indicatif i oc
        tmodinter 2 file (10) car,
                co nom du dernier paragraphe oc
        fichlu file (5) entier,
                co numéro des fichiers lus pour l'indicatif i oc
        fichdir file (5) entier) ;
                co numéro des fichiers possédant l'indicatif i oc
fin classe
Classe travail ;
      h entier ; i entier ; ni entier ;
      pile file entier ;
      valeur file (2, 10) car (high value, low values) ;
      dp booléen ;
fin classe ;
```


3.3. - EXEMPLE DE MISE EN OEUVRE

Soit à éditer un état statistique donnant pour chaque région :

- la liste des clients avec le montant des achats effectués,
- le total des achats,
- la liste des représentants et le montant des ventes réalisées par chacun,
- le chiffre d'affaires total réalisé par les représentants de la région.

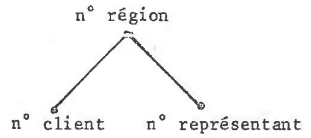
Si le total des achats est différent du chiffre d'affaires total pour la région, on imprime un message d'erreur.

On aura deux fichiers en entrée :

- un fichier des clients trié par numéro région et numéro client croissants et comportant pour chaque client : REGIØN (numéro région), CLIENT (numéro client), N-REGCLI (nom de la région), N-CLI (nom du client), CA-CLI (montant des achats).

- un fichier des représentants trié par numéro région et numéro représentant croissants et comportant pour chaque représentant : REGIØN (numéro région), REPRESEN-T (numéro représentant), N-REGREP (nom de la région), N-REP (nom du représentant), CA-REP (chiffre d'affaires réalisé).

L'arbre de traitement associé à ce programme est le suivant :



PROGRAMME CØBØL SØURCE + METALANGAGE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEST
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. CII-10070.
OBJECT-COMPUTER. CII-10070.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT F-CLI ASSIGN TO CLI.
       SELECT F-REP ASSIGN TO REP.
DATA DIVISION.
FILE SECTION.
FD  F-CLI
       LABEL RECORD IS STANDARD DATA RECORD IS ENRCLI.
01  ENRCLI.
    02  REGION     PIC X(2).
    02  N-REGCLI   PIC X(10).
    02  CLIENT     PIC X(5).
    02  N-CLI      PIC X(10).
    02  CA-CLI     PIC 9(6).
    02  FILLER     PIC X(47).
FD  F-REP
    .  LABEL RECORD IS STANDARD DATA RECORD IS ENRREP.
01  ENRREP.
    02  REGION     PIC X(2).
    02  N-REGREP   PIC X(10).
    02  REPRESAN-T PIC X(5).
    02  N-REP      PIC X(10).
    02  CA-REP     PIC 9(6).
    02  FILLER     PIC X(47).
WORKING-STORAGE SECTION.
77  CUM-REP        PIC 9(7).
77  CUM-CLI        PIC 9(7).
```



```
PROCEDURE DIVISION.
* CINEMATIQUE
    INDICATIF DE F-CLI (ENRCLI), REGION X(2) CRUI, CLIENT
              X(5) CRUI;
    INDICATIF DE F-REP (ENRREP), REGION X(2) CRUI, REPRESAN-T
              X(5) CRUI;
      TRAITINIT
    POUR CHAQUE REGION FAIRE INITREGION
              POUR CHAQUE CLIENT FAIRE TR-CLI FPC
                   TOT-CLI
              POUR CHAQUE REPRESAN-T FAIRE TR-REP FPC
                   FINREGION FPC
      TRAITFIN;
FIN DE LA CINEMATIQUE
TRAITINIT.
    OPEN INPUT  F-CLI F-REP.
INITREGION.
    MOVE 0  TO CUM-REP CUM-CLI.
    IF REGION OF ENRCLI < REGION OF ENRREP
          DISPLAY  ' REGION  ' REGION OF ENRCLI '  ' N-REGCLI
      ELSE  DISPLAY ' REGION  ' REGION OF ENRREP '  ' N-REGREP.
TR-CLI.
    ADD  CA-CLI  TO CUM-CLI.
    DISPLAY '  ' CLIENT '    ' N-CLI '    ' CA-CLI.
TR-REP.
    ADD  CA-REP  TO CUM-REP.
    DISPLAY '  ' REPRESAN-T '  ' N-REP '    ' CA-REP.
TOT-CLI.
    DISPLAY '        TOTAL CLIENTS        ' CUM-CLI.
```

Métalangage.

```
FINREGION.
    IF CUM-REP NOT = CUM-CLI
       DISPLAY  'ERREUR : TOTAUX DIFFERENTS'.
    DISPLAY '        TOTAL REPRESENTANTS '  CUM-REP.
TRAITFIN.
    CLOSE F-CLI F-REP.
    STOP RUN.
```

```
                    **** C O B O L  S I R I S 7   VERSION 004 ****
           IDENTIFICATION DIVISION.
           PROGRAM-ID. TEST
           ENVIRONMENT DIVISION.
           CONFIGURATION SECTION.
           SOURCE-COMPUTER. CII-10070.
           OBJECT-COMPUTER. CII-10070.
           INPUT-OUTPUT SECTION.
           FILE-CONTROL.
                  SELECT F-CLI ASSIGN TO CLI.
                  SELECT F-REP ASSIGN TO REP.
           DATA DIVISION.
           FILE SECTION.
           FD  F-CLI
                  LABEL RECORD IS STANDARD DATA RECORD IS ENRCLI.
           01  ENRCLI.
               02  REGION     PIC X(2).
               02  N-REGCLI   PIC X(10).
               02  CLIENT     PIC X(5).
               02  N-CLI      PIC X(10).
               02  CA-CLI     PIC 9(6).
               02  FILLER     PIC X(47).
           FD  F-REP
                  LABEL RECORD IS STANDARD DATA RECORD IS ENRREP.
           01  ENRREP.
               02  REGION     PIC X(2).
               02  N-REGREP   PIC X(10).
               02  REPRESAN-T PIC X(5).
               02  N-REP      PIC X(10).
               02  CA-REP     PIC 9(6).
               02  FILLER     PIC X(47).
           WORKING-STORAGE SECTION.
           77  CUM-REP        PIC 9(7).
           77  CUM-CLI        PIC 9(7).
           01  ZZ-TR.
               02 FF01  PIC 9.
               88  F01  VALUE 0.
               88  V01  VALUE 1.
               02 FF02  PIC 9.
               88  F02  VALUE 0.
               88  V02  VALUE 1.
               02  ZR01      PIC X(2) .
               02  ZT02I01   PIC X(2) .
               02  ZR02      PIC X(5) .
               02  ZT03I01   PIC X(2) .
               02  ZR03      PIC X(5) .
```

déclaration de variables locales aux modules de cinématique.

III - 12

```
           PROCEDURE DIVISION.
           PR00.
                  PERFORM  TRAITINIT                    .
               MOVE 0 TO FF01 FF02                                .
           LF01.
               READ F-CLI       AT END MOVE 1 TO FF01.
           LF02.
               READ F-REP       AT END MOVE 1 TO FF02.
           CI00.
                  PERFORM  CI01                         .
                  PERFORM  CI02                         .
                  PERFORM  CI03                         .
           TR01.
```

B B B L S O U R C E P R O G R A M A N D D I A G N O S T I C L I S T I N G 2

```
               IF ( V01 AND V02 )                              GO TO  ST01.
                  PERFORM  INITREGION                .
           TR02.
               IF ( V01 )
                 OR ZR01 NOT = ZT02I01                         GO TO  ST02.
                  PERFORM  TR-CLI                    .
           LI02.
               IF F01 AND REGION     OF ENRCLI     = ZR01
                      AND CLIENT     OF ENRCLI     = ZR02
                  PERFORM  LF01
                  IF V01   MOVE HIGH-VALUES TO REGION     OF ENRCLI     .
           CI02.
                 MOVE REGION     OF ENRCLI     TO ZT02I01.
                 MOVE CLIENT     OF ENRCLI     TO ZR02   .
           RT02.
               GO TO TR02.
           ST02.
                  PERFORM  TOT-CLI                   .
           TR03.
               IF ( V02 )
                 OR ZR01 NOT = ZT03I01                         GO TO  ST03.
                  PERFORM  TR-REP                    .
```

instructions "Cobol" réalisant la cinématique des fichiers.

III - 13

```
LIO3.
    IF FO2 AND REGION      OF ENRREP      = ZRO1
           AND REPRESAN-T OF ENRREP      = ZRO3
       PERFORM  LFO2
       IF VO2   MOVE HIGH-VALUES TO REGION      OF ENRREP     .
CIO3.
      MOVE REGION      OF ENRREP      TO ZTO3IO1.
      MOVE REPRESAN-T OF ENRREP      TO ZRO3    .
RTO3.
    GO TO TRO3.
STO3.
       PERFORM  FINREGION                    .
LIO1.
    EXIT.
CIO1.
      MOVE HIGH-VALUES TO ZRO1    .
    IF     REGION      OF ENRCLI     <  ZRO1
      MOVE REGION      OF ENRCLI     TO ZRO1    .
    IF     REGION      OF ENRREP     <  ZRO1
      MOVE REGION      OF ENRREP     TO ZRO1    .
RTO1.
    GO TO TRO1.
STO1.
    GO TO TRAITFIN  .
TRAITINIT.
    OPEN INPUT  F-CLI F-REP.
INITREGION.
    MOVE 0  TO CUM-REP CUM-CLI.
    IF REGION OF ENRCLI < REGION OF ENRREP
            DISPLAY  ' REGION  ' REGION OF ENRCLI '  ' N-REGCLI
      ELSE  DISPLAY ' REGION  ' REGION OF ENRREP '  ' N-REGREP.
TR-CLI.
    ADD  CA-CLI  TO CUM-CLI.
    DISPLAY '  ' CLIENT '    ' N-CLI '    ' CA-CLI.
TR-REP.
    ADD  CA-REP  TO CUM-REP.
    DISPLAY '  ' REPRESAN-T '   ' N-REP '    ' CA-REP.
TOT-CLI.
```



```
      DISPLAY '        TOTAL CLIENTS        '  CUM-CLI.
FINREGION.
    IF CUM-REP NOT = CUM-CLI
        DISPLAY  'ERREUR : TOTAUX DIFFERENTS'.
    DISPLAY '        TOTAL REPRESENTANTS '  CUM-REP.
TRAITFIN.
    CLOSE F-CLI F-REP.
    STOP RUN.
```

2EME PARTIE

/LES OUTILS D'ANALYSE DE CIVA/

Chapitre 4

INTRODUCTION

INTRODUCTION

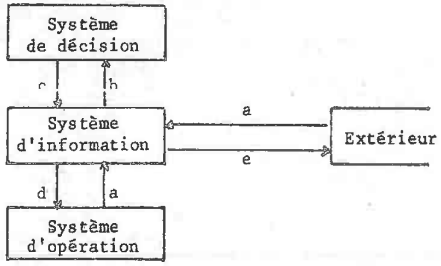
4.1. - ROLE DU SYSTEME DE TRAITEMENT DE L'INFORMATION DANS UNE ORGANISATION (REIX (26), MELESE (16)).

Toute organisation (entreprise, administration) est un assemblage de trois types de systèmes :

- le système de décision : il perçoit des informations, les analyse, les combine suivant un processus défini pour créer de nouvelles informations : les décisions.
  (Exemple : le système de décision relatif à l'approvisionnement : à partir d'informations telles que niveau du stock, prévision des ventes futures, délai de livraison, il va déterminer la quantité à commander du produit x).

- le système d'opération : il est chargé de réaliser les actions correspondant aux décisions élaborées par le système de décision.
  (Exemple : l'ouvrier qui va usiner x pièces d'un type donné à partir du bon de travail qui lui a été remis).

- le système d'information : il est chargé de relier les deux systèmes précédents (voir schéma ci-dessous) :



a) Le système d'information perçoit et stocke des informations provenant du système d'opération et de l'extérieur.
Ces informations ont pour origine des événements internes ou externes à l'organisation (Exemple : bordereau de relevé de lait, bordereau d'analyse, etc...).

b) Le système d'information fournit des informations au système de décision. Ces informations sont synthétiques et élaborées en vue d'une prise de décision. Ce sont également des comptes rendus d'actions exécutées par le système d'opération ou des indicateurs signalant l'état du système d'opération (Exemple : statistique mensuelle de paie du lait).

c) Le système de décision fournit des informations au système d'information :

. des ordres à transmettre aux systèmes d'opérations et à l'extérieur,
. des informations concernant le traitement de l'information (Exemple : définition d'un modèle de gestion des stocks).

d) Le système d'information fournit des informations au système d'opération. Il s'agit de messages d'exécution indiquant les opérations à réaliser et les conditions de leur réalisation (Exemples : un bon de travail indique une opération de fabrication à réaliser, un état de décompte monétaire indique les sommes à décaisser et le nom des bénéficiaires).

e) Le système d'information fournit des informations à l'extérieur. Elles traduisent soit des opérations effectuées avec l'extérieur (Exemple : la facture traduisant la vente), soit des obligations imposées par l'extérieur (Exemple : état fiscal annuel).

4.2. - QUELQUES CONCEPTS PERMETTANT DE DECRIRE LE SYSTEME DE TRAITEMENT DE L'INFORMATION

. Zone naturelle d'information (Z.N.I.) :

Elle recouvre un domaine d'activité de l'entreprise traitant des informations concernant une même population.
Exemple : La Z.N.I. "personnel" comprend tous les traitements concernant l'ensemble des salariés de l'entreprise.

Le découpage en Z.N.I. coïncide plus ou moins bien avec le découpage en fonctions utilisé en gestion.

. Le concept de procédure (MALLET (24)) :

On appelle procédure un ensemble de traitements d'information donnant réponse à un événement, tendant à une fin définie utile et indispensable au fonctionnement de l'entreprise, et d'exécution assez fréquente pour qu'on ait pu en arrêter les modalités.
(Exemples : la paie du personnel, la facturation).

On appelle événement un ensemble d'informations à l'origine d'une ou plusieurs procédures.
Ce peut être :

- une information venant de l'extérieur de l'entreprise : un bon de commande.
- une information émanant d'un service de l'entreprise : un bon de sortie "matière".

La famille de procédures est un ensemble de procédures appartenant à une même Z.N.I., déclenchées par des événements de même nature et nécessaires pour l'obtention des mêmes sorties.

Une application sera constituée de la partie de la famille de procédures qui sera automatisée.

Pour représenter les différentes procédures d'une entreprise, on pourra utiliser le schéma de circulation des documents. Celui-ci permet de visualiser les différents documents et services intervenant dans le déroulement d'une procédure.

En annexe, figure le schéma de circulation de l'application "paie du lait".

.3. - L'ANALYSE PRELIMINAIRE

Le point de départ de l'analyse est constitué par ce qu'on appelle l'analyse préliminaire ou étude d'opportunité.

Celle-ci consiste à recenser toutes les procédures existantes et à les décrire à l'aide des schémas de circulation de documents. Puis, après avoir fait une étude des volumes de données traitées et évalué les coûts, on va décider de l'automatisation ou non du système de traitement de l'information de l'entreprise. Cette phase de l'analyse qui ressort plus des problèmes d'organisation n'a pas été abordée dans le cadre de notre travail. On pourra se reporter à différents ouvrages qui traitent abondamment de ce sujet (BAUVIN (19), REIX (26)).

L'automatisation étant décidée, il faut passer à la phase d'analyse qui consiste à automatiser les différentes applications de l'entreprise.

.4. - ANALYSE FONCTIONNELLE ET ANALYSE ORGANIQUE

Le but du projet CIVA est de fournir un ensemble d'outils permettant de passer, sans heurts, de l'analyse à l'exploitation. Il s'agit essentiellement de définir un langage unique permettant de décrire à la fois les résultats de l'analyse, les programmes et leur exploitation et de construire un système (traducteur et ensemble de services) acceptant ce langage (DERNIAME J.C. (1)).

Pour ce faire, nous proposons à l'utilisateur un certain nombre d'outils lui permettant, d'une part de mener à bien l'analyse d'une application, d'autre part de décrire les résultats de l'analyse en utilisant la modularité et le langage CIVA, évitant ainsi de tout remettre en cause lors du passage à la programmation.

Un chapitre sera consacré à l'analyse fonctionnelle qui étudie les traitements et les informations indépendamment des contraintes matérielles.

Dans le chapitre suivant, nous tiendrons compte de ces contraintes pour déterminer le découpage en programmes et la représentation des données sur des fichiers. Nous rappelons que tous les problèmes liés à l'acquisition et au contrôle des données, ainsi que les problèmes d'édition, ont été présentés dans CHABRIER J.J. (5).

Pour présenter les outils de l'analyse, nous nous appuierons sur une application concrète qui est traitée en annexe en utilisant le langage CIVA.

A cette occasion, nous tenons à remercier la Société MARCILLAT, qui nous a autorisé à présenter une de ses applications et qui nous a fourni tous les renseignements demandés, ainsi que B. JARAY et J.F. DUFOURD, qui ont élaboré cette étude de cas dans le cadre de l'enseignement de l'analyse à l'I.U.T. Informatique de NANCY.

Chapitre 5

ANALYSE FONCTIONNELLE

ANALYSE FONCTIONNELLE

5.1. - INTRODUCTION

L'analyse fonctionnelle consiste à définir la logique générale d'une application d'une façon indépendante des moyens qui en assureront le fonctionnement.

Les sorties demandées par les services "consommateurs" d'informations sont le point de départ.

On décrit ensuite les divers modes d'obtention des notions constituant les sorties et on récapitule l'ensemble des notions d'entrée nécessaires.

L'ensemble des notions d'entrée sont regroupées selon certains critères en sous-ensembles logiques appelés : segments logiques.

Dans ce chapitre, nous présentons, d'une part un certain nombre de concepts et d'outils permettant d'effectuer ces tâches, d'autre part leur mise en oeuvre dans le cadre du projet CIVA.

.2. - DEFINITION DES CONCEPTS UTILISES DANS L'ANALYSE FONCTIONNELLE

5.2.1. Sortie, application et chaîne de traitement :

Une sortie est une collection de données fournie par le système de traitement de l'information et destinée à un autre système ou au même système dans un cycle ultérieur.

L'ensemble des notions, dont les valeurs constituent la sortie, est déterminé par le système destinataire (et sera appelé unité logique de sortie).

5.2.1.1. Les sorties destinées à un système autre que le système de traitement de l'information :

Elles constituent l'essentiel des sorties des applications en informatique. Ainsi, dans l'application traitée en annexe, toutes les sorties répertoriées sauf une sont de ce type.

Le support de ces sorties devra être celui qui est le mieux adapté au destinataire (imprimé, message affiché sur écran, message oral, etc...).
En général, ce support sera l'imprimé.

Dans le cadre de l'analyse fonctionnelle, nous considérons que les sorties seront représentées en format interne sur un support magnétique.
Le changement de format (interne-externe) et de support des fichiers internes CIVA sera réalisé par les services d'édition de fichiers externes de CIVA (CHABRIER J.J. (5)).

On peut mettre en évidence dans l'ensemble des notions d'une sortie des sous-ensembles qui sont édités sous une même condition et qui constituent un groupe logique de sortie.

Il sera indispensable de sélectionner une ou plusieurs notion(s) présentes dans le groupe, soit alors d'en introduire une (ou plusieurs) particulière(s) de telle façon que chaque réalisation de cette (ces) notion(s) permette(nt) d'identifier sans ambiguïté toute réalisation du groupe logique de sortie.

Une telle notion est appelée indicatif (1).

Ces groupes logiques de sortie vérifient des relations d'inclusion qui permettent de définir un ordre sur les indicatifs associés.
Cette relation d'ordre sur les indicatifs sera représentée par une arborescence (arbre d'édition de la sortie).
Nous appellerons unité logique de sortie l'ensemble des groupes logiques constituant la sortie.
Une unité logique sera caractérisée par les indicatifs communs à tous les groupes logiques de sortie figurant en majeur sur l'arborescence et constitueront l'index du fichier de sortie (voir 1ère partie)

Cet index permet d'identifier chaque article du fichier de sortie.

(1) Les indicatifs de groupe logique n'interviennent pas dans la cinématique des fichiers.
Celle-ci utilise uniquement les indicatifs constituant l'index du fichier et ceux définissant des groupements logiques d'enregistrements (voir chapitre 1).

Exemple :

Nous présentons ici le document de sortie "Bordereau de paie du lait" qui figure en annexe.
On y trouve également une table des symboles à laquelle on pourra se reporter pour la signification des identificateurs.

NØAGR
NØM
PRENØM
ADRESSE

TØUR
RELEVE
MAT-GRAS
MAT-AZ
QU-BAC
TM-GRAS
TM-AZ
TM-BAC
PR-GRAS
PR-AZ
PR-BAC
PR-BASE
PR-CTR
PR-TEMP
PR-PAT
PR-MØN
PR-LAIT
TØTLI
VALHT
TVA
NET-LAIT

} Groupe logique de sortie 1 ; indicatif : NØAGR

NØART
QTE
DESIGN
PRIX
TVA 7
TVA 19
MTTC

} Groupe logique de sortie 2 ; indicatifs : NØAGR, NØPRØD

STVA 7
STVA 19
NAP

} Groupe logique de sortie 3 ; indicatif : NØAGR

CØDIV
LIBDIV
PDIV
MDIV

} Groupe logique de sortie 4 ; indicatifs : NØAGR, CØDIV

LI-MØIS
MØDRFG
NFL
NFP
ACØMPTES
SMDIV
SPDIV
APAYER
REMB

} Groupe logique de sortie 5 ; indicatif : NØAGR.

Avant d'aborder le cas des sorties destinées au système de traitement de l'information dans un cycle ultérieur, définissons d'abord quelques concepts qui nous sont utiles pour la suite.

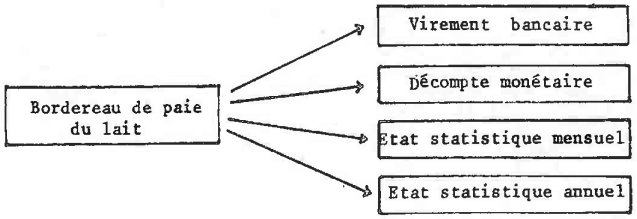
5.2.1.2. La notion d'application et de chaîne de traitement :

Une application A est un ensemble fini de sorties S_i.
On peut définir sur A une relation binaire que nous noterons $\cup$ et qui se lit "utilisé pour le calcul de".

Etant donné S_i, S_j $(i \neq j) \in A$, on aura par exemple $S_i \cup S_j$ (S_i est utilisé pour le calcul de S_j).

Cette relation permet de construire un graphe sur l'ensemble A.
Ce graphe doit être sans circuit.

Dans l'exemple traité en annexe, on a :



Pour établir le bordereau des virements bancaires, l'état de décompte monétaire, l'état statistique mensuel, l'état statistique annuel, on utilise des données qui ont été calculées pour établir le récapitulatif de paie du lait.

Une chaîne de traitement A_i est un sous-ensemble de A tel que les $S_j \in A_i$ aient même périodicité d'obtention. Les sous-ensembles A_i constituent une partition de A et on vérifie que

$$A = \bigcup_{i=1}^{n} A_i$$

Dans l'exemple traité en annexe, on a deux sous-ensembles qui sont respectivement :

- A_1 : "Bordereau de paie du lait", "bordereau des virements bancaires", "état statistique mensuel" ;
  la périodicité de ces sorties est le mois.

- A_2 : "état statistique annuel".

5.2.1.3. Les sorties destinées au système de traitement de l'information dans un cycle ultérieur :

a) Les données obtenues en sortie d'une chaîne de traitement de périodicité donnée et destinées à une autre chaîne de traitement appartenant ou non à la même application.

Dans l'exemple traité en annexe, nous avons, dans la chaîne de traitement mensuelle, une sortie de ce type qui est destinée à la chaîne de traitement annuelle.

En effet, il faut conserver certaines données calculées chaque mois afin d'établir l'état statistique annuel.

b) Les données obtenues en sortie d'une chaîne de traitement à la période n - 1 et destinées à la même chaîne de traitement lors de sa mise en oeuvre à la période n.

Prenons une chaîne de traitement ayant à établir chaque mois un état des quantités en stock de chaque produit.
Pour déterminer la quantité en stock à la fin de la période n, il faut connaître la quantité en stock à la fin de la période n - 1 et les mouvements durant la période n.

La quantité en stock à la fin d'une période constitue une sortie du type précédemment défini.

5.2.2. Mode d'obtention d'une sortie :

5.2.2.1. Sous-chaîne, phase de traitement, opération :

5.2.2.1.1. Sous-chaîne

C'est l'ensemble des opérations permettant d'obtenir une sortie. On peut fusionner plusieurs sous-chaînes en une seule si l'ensemble S de leurs sorties vérifie la règle suivante :

Pour toute sortie S_i (élément non majorant du graphe défini sur S par la relation $\cup$), il existe une sortie $S_j \in S$ telle que :

- $S_j \cup S_i$,
- I_i (ensemble des indicatifs de S_i) $\subset I_j$,
- Tout indicatif commun doit occuper le même rang et avoir les mêmes prédécesseurs sur les deux arbres d'édition.

Exemple :

Nous calculons dans la même sous-chaîne la sortie "bordereau paie du lait" et le fichier de sortie "récapitulatif agriculteur".

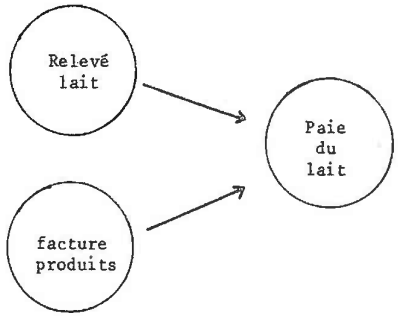
5.2.2.1.2. Phase de traitement

Elle consiste en un partage de la sous-chaîne en sous-ensembles traduisant la logique du problème telle qu'elle est perçue par l'analyste. Une phase de traitement sera caractérisée par les données transmises à la phase de traitement suivante ou par le fichier de sortie pour la dernière phase de traitement d'une sous-chaîne.

Ces données sont l'ensemble des réalisations d'un ensemble de notions constituant un segment transmis (voir 5.3.2.1.3.). et les notions le caractérisant seront appelées notions transmises.

Dans l'exemple traité en annexe, nous avons considéré que la sous-chaîne : "paie du lait" constitue une phase de traitement. Supposons qu'on ait découpé cette sous-chaîne en trois phases de traitement :

- une phase "relevé lait" qui calcule la 1ère partie du bordereau "paie du lait" ;
- une phase "facture produits" qui calcule la 2ème partie du bordereau "paie du lait" ;
- une phase "paie du lait" qui calcule la partie "relevé de compte" du bordereau "paie du lait" et qui crée les articles du fichier de sortie de la sous-chaîne.



Les notions constituant les segments transmis à la phase de traitement "paie du lait" seront :

- pour la phase de traitement "relevé de lait" :
  numéro agriculteur, nom, prénom, numéro tournée, adresse, 31 relevés journaliers, 3 taux de matière grasse, 3 taux de matière azotée, 3 taux de qualité bactériologique, taux moyen de matière azotée, taux moyen de qualité bactériologique, taux moyen de matière grasse, prime de qualité bactériologique, prime de matière azotée, prime de matière grasse, etc... ;

- pour la phase de traitement "facture produits" :

n fois { numéro agriculteur, numéro produit, désignation, prix unitaire, montant hors taxes, montant T.V.A. 7, montant T.V.A. 19, montant taxes comprises,

total T.V.A. 7, total T.V.A. 19, total montant hors taxes, total montant taxes comprises.

La phase de traitement "paie du lait" crée le fichier de sortie de la sous-chaîne "paie du lait" à partir des deux segments transmis et d'autres données supplémentaires.

5.2.2.1.3. Opération

C'est la transformation appliquée à une ou plusieurs notions appelées notions d'entrée et permettant d'obtenir une notion différente appelée notion résultat.

Exemple :

Montant hors taxe = Prix unitaire * quantité

notion résultat — notions d'entrée

5.2.2.2. Les caractéristiques des notions d'entrée d'une phase de traitement :

. Notion indexée :

est une notion caractérisant un ensemble fini d'entités.

Si N est une notion indexée caractérisant un ensemble fini d'entités F d'index I :
on a une fonction surjective f : I ⟶ N.

Soit l'ensemble des agriculteurs apportant leur lait à une coopérative :
NØM, PRENØM, ADRESSE seront des notions indexées ; NUMERØ = "numéro de l'agriculteur" sera l'index de cet ensemble fini d'entités.

. Notion non indexée :

est une notion ne caractérisant pas un ensemble fini d'entités et à laquelle on ne peut associer qu'une seule réalisation

taux de T.V.A. à 7 % : T.V.A. 7 = {7}

. Classification des notions indexées :

- Les notions transmises

Ce sont les notions figurant dans le segment transmis par la phase de traitement précédente à la phase de traitement dont on étudie les notions d'entrée.

- Les notions intermédiaires

Notions résultat ne figurant pas dans un segment transmis ou un fichier de sortie et servant au calcul d'une notion transmise ou sortie.

Exemple :

Dans la phase de traitement "bordereau de paie du lait" on a les opérations :

APAYER = SØLDE si SØLDE > 0
REMB = SØLDE si SØLDE < 0
SØLDE = NET-LAIT - NAP - ACØMPTES + SPDIV - SMDIV

SØLDE est une notion intermédiaire car elle ne figure pas dans un fichier transmis ou une sortie.
APAYER, REMB sont des notions résultat figurant dans une sortie.

- Les notions permanentes

a) Ce sont des notions dont l'ensemble des valeurs est stable dans le temps. L'ensemble des valeurs introduit une seule fois dans le système peut y être stocké pour être utilisé dans la phase de traitement pendant plusieurs périodes. Il faudra néanmoins prévoir la mise à jour périodique de cet ensemble à partir des modifications l'affectant (adjonction, suppression, modification d'une valeur).

Les notions NØM, PRENØM, ADRESSE caractérisant l'ensemble d'entités "agriculteurs" sont de ce type.

b) On range également dans cette classe les notions dont l'ensemble des valeurs en entrée de la phase de traitement est remplacé par un ensemble de valeurs nouvelles en sortie de la phase de traitement (voir b) de 5.2.1.3.).

Les notions permanentes de ce type traduisent l'état d'un ensemble d'entités à un instant donné.

La quantité en stock est une notion de ce type.

- Les notions neuves

Elles caractérisent un événement d'origine extérieure au système de traitement de l'information, qui, introduit dans celui-ci, déclenche les traitements d'une application.

L'ensemble des valeurs associé à une notion neuve varie à chaque mise en oeuvre de l'application considérée et devra donc être introduit à chaque fois dans le système de traitement de l'information.

Ainsi, les 31 relevés journaliers constituent une notion neuve.

5.2.2.3. Classification des opérations :

La classification que nous présentons est à rapprocher de celle proposée par PAIR C. (25).

5.2.2.3.1. Les opérations simples

. Opération de transfert :

L'ensemble des valeurs associé à la notion résultat sera égal à l'ensemble des valeurs associé à la notion d'entrée.

Lors de l'établissement du bordereau "paie du lait", on aura pour chaque agriculteur :

NØM (sur état de sortie) = NØM (dans fichier "agriculteur").

. Opération de calcul :

Exemple :

	MØNTANT	=	PRIX	X	QUANTITE
indicatifs :	NØAGR, NØART		NØART		NØAGR, NØART

Les indicatifs associés aux notions d'entrée doivent être totalement ordonnés selon la relation d'inclusion définie sur les groupements logiques mis en évidence dans les ensembles finis d'entités auxquels appartiennent ces notions (voir chapitre 1).

Les notions d'entrée doivent avoir au moins un indicatif commun.

Une des notions d'entrée, de même que la notion résultat, doivent posséder l'ensemble des indicatifs associés aux notions d'entrée.

L'opération de calcul sera effectuée pour chaque réalisation de cette notion d'entrée.

Ainsi, dans notre exemple, l'opération sera effectuée pour chaque réalisation de la notion QUANTITE.

Remarque :

Il peut arriver qu'une réalisation de QUANTITE soit associée à une réalisation de NØART n'ayant pas de correspondant parmi les réalisations de PRIX. On est alors dans un cas d'erreur et l'opération ne sera pas exécutée.

. Opération itérative :

- de type cumul :

$$\underbrace{NAP}_{N\emptyset AGR} = \underbrace{NAP}_{N\emptyset AGR} + \underbrace{MTTC}_{N\emptyset AGR,\ N\emptyset ART}$$

indicatifs :

Une valeur de la notion résultat est obtenue à partir de plusieurs valeurs d'une notion d'entrée.
Cette opération définit une partition d'un ensemble d'entités en sous-ensembles (appelés groupements logiques d'entités) ayant même valeur pour une notion appelée indicatif de traitement (voir chapitre 1).
Ainsi, dans notre exemple, une valeur de NAP est obtenue à partir d'un sous-ensemble de réalisations de MTTC ayant même valeur pour NØAGR.

- de type itération sur une notion indicée :

Une notion indicée est une notion dont chaque réalisation est une table (un ensemble V de valeurs, un ensemble I d'indices et une fonction f de I dans V) (PAIR C. (15)), dont l'ensemble des indices est un intervalle de la suite des entiers positifs, négatifs ou nuls.

Exemple :

RELEVE file (31) décimal 9 (4) ;

L'opération itérative sur la notion indicée sera :

Pour I de 1 à 31 faire $\underbrace{TØTLI}_{N\emptyset AGR}$ = $\underbrace{TØTLI}_{N\emptyset AGR}$ + $\underbrace{RELEVE\ (I)}_{N\emptyset AGR}$

indicatifs :

Une exécution de cette opération porte sur l'ensemble des valeurs associé à une réalisation de la notion indicée.

5.2.2.3.2. Les opérations conditionnelles

La suite des opérations simples permettant de calculer une notion résultat sera choisie en fonction des propriétés d'une ou plusieurs notions sur lesquelles porte la condition.

Exemple :

Si TQBA $\geqslant$ 8 alors PQBA = PRI-QBA

Si TQBA $\leqslant$ 4 alors PQBA = REF-QBA

sinon PQBA = 0

La notion PQBA (prime de qualité bactériologique) peut être obtenue de trois façons différentes suivant la valeur de la notion TQBA (taux de qualité bactériologique) sur laquelle porte la condition.

5.3. - MISE EN OEUVRE DE L'ANALYSE FONCTIONNELLE

5.3.1. - Description des opérations d'une phase de traitement :

Elle se fait dans un document appelé grille d'évaluation.

5.3.1.1. Présentation de la grille d'évaluation :

NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS ENTREE		
			TYP	IDENT.	INDICATIFS
(1)	(2)	(3)	(4)	(5)	(6)

Colonne (1) : identificateurs des notions résultat de la phase de traitement.

Colonne (2) : description de l'opération permettant d'obtenir la notion résultat figurant en colonne (1). Cette colonne est vide lorsque la notion résultat est obtenue par une opération de transfert.

Colonne (3) : décrit les conditions devant être satisfaites pour que l'opération de la colonne (2) soit exécutée.

Colonne (4) : on y fait figurer un caractère alphabétique indiquant à quel type appartient la notion d'entrée inscrite en colonne (5).

Ce caractère peut prendre les cinq valeurs suivantes :

. N : notion neuve,
. P : notion permanente,
. T : notion transmise,
. I : notion intermédiaire,
. C : notion non indexée.

Colonne (5) : identificateurs de notions entrée.

Colonne (6) : indicatifs attachés à la notion figurant en colonne (5).

5.3.1.2. Exemple d'utilisation :

Nous allons présenter la première partie de la phase de traitement P1 : "Bordereau de paie du lait" (Voir en annexe).

NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS ENTREE		
			TYP	IDENT.	INDICATIFS
NØM			P	NØM	NØAGR
ADRESSE			P	ADRESSE	NØAGR
TØUR			P	TØUR	NØAGR
RELEVE			M	RELEVE	NØAGR
MAT-GRAS			M	MAT-GRAS	NØAGR
TM-GRAS	$\sum_{i=1}^{3}$ MAT-GRAS (i)/3				
QU-BAC			M	QU-BAC	NØAGR
TM-BAC	$\sum_{i=1}^{3}$ QU-BAC (i)				
MAT-AZ			M	MAT-AZ	NØAGR
TM-AZ	$\sum_{i=1}^{3}$ MAT-AZ (i)/3		C	F-GRAS	
PR-GRAS	(TM-GRAS - F-GRAS) × V-GRAS		C	V-GRAS	

NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS ENTREE		
			TY	IDENT.	INDICATIFS
			C	F-AZ	
PR-AZ	(TM-AZ - F-AZ)×V-AZ		C	V-AZ	
PR-BASE			C	PR-BASE	
PR-BAC	PRI-BAC	Si TM-BAC ⩾ 8	C	PRI-QBA	
	REF-BAC	Si TM-BAC ⩽ 4	C	REF-QBA	
	0	Sinon			
PR-CTR	V-CTR	Si CTR = 1	P	CTR	NØAGR
	0	Sinon	C	V-CTR	
PR-TEM	V-TEM	Si TEMP = 1	P	TEMP	NØAGR
	0	Sinon	C	V-TEMP	
	V-PAT	Si PAT = 1	P	PAT	NØAGR
	0	Sinon	C	V-PAT	
PR-MØN	V-MØN	Si MØN = 1	P	MØN	NØAGR
	0	Sinon			

5.3.1.3. Description des opérations avec les tables de décision :

En effet, si les conditions de calcul de certaines notions sont nombreuses et imbriquées, il vaut mieux utiliser une table de décision pour décrire les opérations nécessaires à l'obtention de ces notions.

Il suffira d'indiquer dans la colonne "opérations" de la grille d'évaluation le nom du module dans lequel est décrite la table de décision.

Pour l'exemple traité dans le paragraphe précédent, on aura :

NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS ENTREE		
			TY	IDENT.	INDICATIFS
PR-BAC	Module TAB1		C	PRI-QBA	
			C	REF-QBA	
PR-CTR	V-CTR	Si CTR = 1	P	CTR	NØAGR
	0	Sinon	C	V-CTR	
PR-TEM	V-TEM	Si TEM = 1	P	TEM	NØAGR
	0	Sinon	C	V-TEM	
PR-PAT	V-PAT	Si PAT = 1	P	PAT	NØAGR
	0	Sinon	C	V-PAT	
	V-MØN	Si MØN = 1	P	MØN	NØAGR
	0	Sinon	C	V-MØN	

Module TAB 1 ;

Sélection

NØTIØNS RESULTAT	TM-BAC ⩾ 8	TM-BAC ⩽ 4	Sinon
Actions			
PR-BAC = PRI-QBA	1		
PR-BAC = REF-QBA		1	
PR-BAC = 0			1

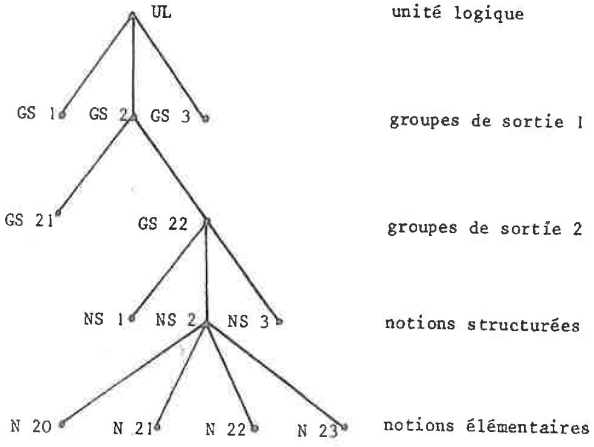
fin table

fin module ;

On peut également, au moment de l'établissement de la grille d'évaluation, ne pas vouloir préciser les opérations permettant d'obtenir certaines notions résultat. Il suffira alors d'indiquer dans la grille d'évaluation le nom du module dans lequel seront décrites ces opérations.

5.3.1.4. Les conditions d'existence des sorties :

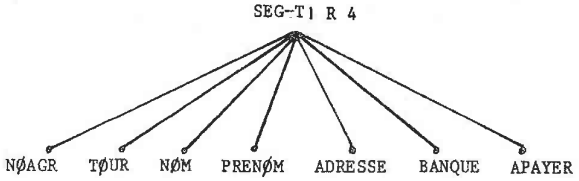
Les résultats d'une phase de traitement ont une structure pouvant être représentée par une arborescence traduisant les relations d'inclusion existant entre des sous-ensembles de notions résultat.



Dans la grille d'évaluation, nous avons uniquement décrit les opérations permettant de calculer les notions élémentaires. Or ces notions élémentaires peuvent appartenir à des notions structurées, à des groupes logiques, à une unité logique.
En conséquence, une réalisation de la notion élémentaire n'existera que si les réalisations des notions structurées, des groupes logiques et de l'unité logique existent. On appelle condition d'existence la condition à laquelle est attachée l'existence sur le fichier de sortie d'une notion structurée, d'un groupe logique, de l'unité logique.

Exemple :

Soit la phase de traitement P1. On y calcule en particulier des données transmises à la phase suivante et dont la structure très simple se présente ainsi :



La grille d'évaluation correspondant à la création de ce fichier de sortie est la suivante :

NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS ENTREE		
			TY	IDENT.	INDICATIFS
	existence	MØDREG = "BA"	P	MØDREG	NØAGR
NØAGR			P	NØAGR	NØAGR
TØUR			P	TØUR	NØAGR
NØM			P	NØM	NØAGR
PRENØM			P	PRENØM	NØAGR
ADRESSE			P	ADRESSE	NØAGR
BANQUE			P	BANQUE	NØAGR
APAYER	SØLDE	Si SØLDE > 0	I	SØLDE	

Pour décrire une condition d'existence sur la grille d'évaluation, on indique le nom de la notion structurée en colonne (1), le libellé "existence" en colonne (2), la condition en colonne (3), la notion entrée nécessaire pour exprimer la condition dans les trois dernières colonnes.

Dans notre exemple, on ne crée un article pour le segment transmis SEG-T1 R4 que si MØDREG (mode de réglement) = "BA", c'est-à-dire si l'agriculteur a choisi d'être réglé par virement bancaire.

5.3.2. - La description des informations :

Après l'établissement des grilles d'évaluation pour toutes les phases de traitement d'une application, on possède la liste des notions d'entrée nécessaires pour obtenir les sorties.

Ces notions d'entrée éparses vont être regroupées suivant certaines propriétés pour constituer des segments logiques. Il faut également rendre communes à l'ensemble des phases de traitement d'une application les descriptions de propriétés des objets créés : segments logiques, notions structurées, notions élémentaires.

Nous verrons en 5.3.2.2. que ceci peut se faire dans une classe associée à l'application, appelée classe répertoire, qui contient la description de ces types.

La classe répertoire fait partie de la classe d'application.

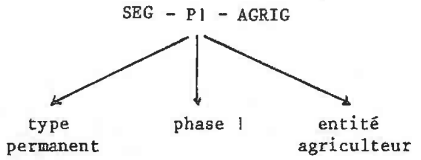
5.3.2.1. Les segments logiques :

Un segment logique est un ensemble de notions figurant dans une phase de traitement vérifiant certaines propriétés.
La première propriété considérée sera évidemment le type de la notion. Cette partition en sous-ensembles de notions de même type sera affinée en utilisant d'autres propriétés propres à chaque type de notion.

5.3.2.1.1. Les segments permanents

Il sera composé de l'ensemble des notions de type permanent caractérisant une même entité. Un segment permanent sera identifié par le type de ses notions, le numéro de phase et par l'entité qu'il caractérise.

Ainsi, dans la phase P1 "bordereau de paie du lait" on aura un segment identifié :



5.3.2.1.2. Les segments mouvements

Un segment mouvement sera composé de l'ensemble des notions de type neuves qui apparaissent en même temps pour caractériser un événement.
En effet, pour que la saisie des données de ce type s'effectue dans les meilleures conditions possibles, il y a lieu de fractionner l'ensemble des notions de ce type en sous-ensembles homogènes au plan du lieu et du temps de saisie.

Un segment mouvement sera identifié par son type, le numéro de phase et par l'événement qu'il caractérise.

Dans la phase P1 "bordereau paie du lait", on aura quatre segments mouvement :

SEG - M1 - RAMASSAGE (bordereau de ramassage)
SEG - M1 - ANALYSE (bordereau d'analyse du lait)
SEG - M1 - DIVERS (bordereau des opérations diverses)
SEG - M1 - ACHATS (bordereau des achats effectués).

5.3.2.1.3. Les segments transmis

Les notions transmises ont deux origines :

- ce sont des notions obtenues dans une phase d'une sous-chaîne et réutilisées dans une ou plusieurs autre(s) sous-chaîne(s) appelée(s) réceptrice(s).

Il y aura autant de segments logiques que de sous-chaînes réceptrices.

Ainsi, la sous-chaîne "calcul paie du lait" qui transmet des notions aux trois autres sous-chaînes de l'application crée trois segments logiques.

- le découpage d'une sous-chaîne en phases de traitement implique la création par une phase de notions résultat utilisées par une ou plusieurs autres lui succédant.
  Dans une phase de traitement, il y aura autant de segments logiques transmis que de phases de traitement réceptrices des notions transmises.

Un segment transmis sera identifié par le type de ses notions, le numéro de la phase origine, et le numéro de la phase réceptrice.

La phase de traitement "bordereau de paie du lait" qui est aussi une sous-chaîne aura plusieurs segments de ce type.

SEG - T1 R3 (notions transmises à la sous-chaîne "décompte monétaire" (Phase 2)).

SEG - T1 R2 (notions transmises à la sous-chaîne "état statistique mensuel" (Phase 3)).

SEG - T1 R4 (notions transmises à la sous-chaîne "virement bancaire" (Phase 4)).

5.3.2.1.4. Les segments intermédiaires

On a un segment de ce type par phase de traitement. Il regroupe toutes les notions intermédiaires de cette phase.

Il sera identifié par son type et le numéro de phase.

Ainsi SEG - I2 contient toutes les notions intermédiaires de la phase de traitement 2.

5.3.2.1.5. Les segments de constantes

Pour chaque phase de traitement, on crée un segment des constantes qui contient toutes les notions non indexées de la phase.

Il sera identifié par le type de ses notions et le numéro de phase.

Par exemple, SEG - C2 contient toutes les notions non indexées de la phase de traitement 2 .

5.3.2.2. La classe répertoire :

Elle permet de rendre communes à toute l'application des descriptions de propriétés des objets créés durant l'analyse fonctionnelle.
En effet, de nombreuses notions figurant dans différents segments et étant utilisées dans différentes phases de traitement, elle évite de déclarer plusieurs fois la même notion.
Dans cette classe ne sont autorisées que les déclarations de type, car, à ce stade d'analyse, le découpage en unités de traitement n'étant pas réalisé et tous les modules n'ayant pas encore été déterminés, le partage des objets créés entre les modules n'est pas possible.

Elle est également la base de la documentation et permet de construire la lexique de l'application (BARTHELEMY C, PHILIPPE (3)).

Dans cette classe seront décrites les notions élémentaires, les notions structurées et les segments logiques mis en évidence au cours de l'analyse fonctionnelle.

Exemple :

Nous prenons quelques éléments de la classe répertoire associée à l'application traitée en annexe.

```
Classe répertoire ;
  co description d'un article du fichier de sortie "décompte monétaire" oc
     type ART-DEC struct GR-LOG 1 struct (SITUATIØN, APAYER, T-ESP file (9)
                                                        décimal 99),
                         GR-LØG 2 struct (CUM-T-ESP file (9) décimal 9 (3)),
                         GR-LØG 3 struct (CUM-G-ESP file (9) décimal 9 (4))) ;
  co description de segments logiques oc
  type SEG-M 1-RAMASSAGE struct (NOAGR,
                                 RELEVE file (31) décimal 9 (4)) ;
  type SEG-P1 -PRØDUIT   struct (
              NØART      décimal 9 (4),
              DESIGN     file (15) car,
              PRIX       décimal 9 (4) V 99,
              CØ-TVA     décimal 9) ;
  .........
  .........
fin classe ;
```

Toutes les notions structurées et élémentaires sont décrites une seule fois, bien qu'elles soient répétées dans plusieurs segments logiques ou fichiersde sortie.

Nous avons appliqué la règle suivante :

Toutes les notions structurées et élémentaires figurant dans un segment permanent seront décrites dans ce segment.

Toutes les notions figurant dans un segment mouvement et non encore décrites le seront dans ce segment.

Toutes les autres notions seront décrites dans les segments transmis, les fichiers de sortie, les segments intermédiaires, et les segments de constantes suivant leur ordre d'apparition durant l'analyse.

Chapitre 6

ANALYSE ORGANIQUE

ANALYSE ORGANIQUE

6.1. - INTRODUCTION

L'analyse organique a pour but de rechercher la meilleure façon de traiter les informations pour satisfaire les demandes des utilisateurs tout en respectant les contraintes imposées par le matériel et le logiciel utilisés.

Le point de départ est constitué par les éléments fournis par l'analyse fonctionnelle : les opérations, les phases de traitement et les segments logiques.

Elle consiste à prendre un certain nombre d'options en ce qui concerne :

- l'organisation des informations : découpage en fichiers ;

- l'organisation des traitements : découpage en chaîne, en unités de traitement et en modules. Nous proposons dans ce chapitre, plusieurs règles permettant de passer des phases de traitement aux unités de traitement.

6.2. - CONSTITUTION DES FICHIERS

Toutes les données utilisées dans le cadre d'une application doivent être regroupées selon certains critères pour constituer des fichiers. Il faut alors choisir leur support physique et la structure de leur représentation en mémoire. C'est cette structure qui détermine dans quelles conditions il sera possible d'accéder aux données ainsi stockées.

Ces données sont des réalisations des notions décrites au cours de l'analyse fonctionnelle. On peut donc leur appliquer la classification établie pour les notions et on aura donc des données permanentes, neuves, intermédiaires, transmises et constantes.

Les données intermédiaires n'étant utilisées que dans une phase de traitement, ne sont pas enregistrées sur un support physique.

Les fichiers de sortie, quant à eux, sont déterminés dès l'analyse fonctionnelle.

6.2.1. Les fichiers permanents :

Ils sont constitués uniquement de données permanentes et peuvent être déterminés selon deux critères.

6.2.1.1. Regroupement sur un ensemble fini d'entités :

Le fichier sera constitué à partir de l'ensemble des notions caractérisant un même ensemble fini d'entités.
Un tel fichier ne peut être conçu au niveau d'une application, mais de toutes les applications utilisant le même ensemble fini d'entités.

Ainsi, un fichier "clients" comprendra non seulement toutes les notions nécessaires à l'application "facturation" : NØM, PRENØM, ADRESSE, REPRESENTANT, REMISE, etc..., mais également celles nécessaires à l'application "comptabilité clients" : CØMPTE, DEBIT, CREDIT, SØLDE, etc...

Un fichier conçu dans une telle optique sera volumineux et posera des problèmes de représentation sur les supports physiques. De plus, lors de sa consultation par une unité de traitement, on n'aura en général besoin que d'une petite partie des notions le constituant.

6.2.1.2. Regroupement sur le traitement :

Il nous amène à limiter le regroupement selon le premier critère en tenant compte des traitements effectués sur l'ensemble fini d'entités.

On peut ainsi évaluer l'intensité des relations entre les différentes notions en mesurant la fréquence de leur présence simultanée dans les différentes sous-chaînes de traitement (REIX (26)).

Ceci permet de mettre en évidence des sous-ensembles de notions ayant des relations plus intenses et pouvant constituer des fichiers.

Mais ce critère de regroupement sur les traitements doit être appliqué avec modération afin d'éviter la multiplication des fichiers.

Dans l'exemple traité en annexe, on a deux fichiers permanents associés aux deux ensembles finis d'entités "agriculteurs" et "produits".
Nous avons appliqué dans ce cas le seul critère de regroupement sur les entités.

Il faut effectuer la mise à jour des fichiers permanents avant toute utilisation. Dans les systèmes de programmation classique ceci était réalisé par une unité de traitement spécifique. Dans CIVA ceci se fera dans le cadre du service d'acquisition en écrivant un métamodule ad hoc.

6.2.2. Les fichiers mouvements :

Les fichiers mouvements seront constitués des notions neuves mises en évidence dans l'application.

Ils représentent des ensembles de données qui vont être introduites pour la première fois dans le système de traitement de l'information. Pour ces fichiers, le problème essentiel est celui de la saisie de l'information. En conséquence, on fractionnera l'ensemble des notions neuves en différents fichiers homogènes au plan du lieu et du temps de saisie. Les notions qui apparaissent ensemble sur un document de base parce qu'elles traduisent le même événement constitueront les articles d'un fichier mouvement.

Dans l'application traitée en annexe, nous aurons quatre fichiers mouvement obtenus à partir de quatre documents de base, traduisant chacun un événement déterminé.

document de base	fichier mouvement
bordereau de ramassage	FIM-RELEVE
bordereau d'analyse	FIM-ANALYSE
bordereau des divers	FIM-DIVERS
bordereau des achats	FIM-ACHATS

6.2.3. Les fichiers transmis :

Les fichiers transmis ne seront définitivement constitués qu'après le découpage de la chaîne de traitement en unités de traitement. A ce stade, on peut considérer que chaque segment logique transmis constituera un fichier transmis.

6.3. - ORGANISATION DES TRAITEMENTS

6.3.1. Constitution des unités de traitement :

L'analyse fonctionnelle nous fournit pour chaque chaîne de traitement un ensemble d'opérations regroupées par phase de traitement et permettant d'obtenir les sorties à partir d'un ensemble de notions d'entrée.

Pour que cette analyse devienne opérationnelle, il faut aboutir à un découpage de la chaîne en unités de travail pouvant être mises en oeuvre sur un ordinateur.

Ces unités de travail sont appelées unités de traitement.
On peut en donner les deux définitions suivantes :

- Fraction des travaux composant la chaîne qui peut être régie par un programme et qui correspond en général à un seul passage du flux d'informations à travers la mémoire centrale (REIX (26)).

- Unité de fonctionnement de la machine définie par des fichiers d'entrée, des fichiers de sortie et un programme transformant les données en résultats (MALLET (24)).

Dans la suite de ce paragraphe, nous allons constituer les unités de traitement en deux étapes :

- dans une première étape, nous allons tenir compte des contraintes d'accès aux fichiers et regrouper au maximum les opérations pour minimiser le nombre d'accès aux fichiers.

- dans une deuxième étape, nous allons limiter ce regroupement en tenant compte des contraintes imposées par le matériel et de l'objectif de simplicité de mise en oeuvre des programmes.

6.3.1.1. Les phases de traitement organiques :

A ce stade, les fichiers ont été constitués, leur représentation sur des supports physiques et les méthodes d'accès (accès sélectif ou séquentiel) ont été fixées pour chacun d'entre eux.

De plus, à chaque fichier est attachée une liste d'indicatifs permettant de définir des groupements logiques sur ce fichier et impliquant un ordre de rangement de ses articles selon ces indicatifs (cf. chapitre 1 en 1.2.3.1.).

Cette liste d'indicatifs est obtenue au cours de l'analyse fonctionnelle lorsqu'on définit les opérations sur les notions d'entrée pour obtenir les notions résultat. Ceci va nous permettre de dégager trois règles de compatibilité devant être vérifiées par les fichiers à accès séquentiel en entrée d'une phase de traitement :

Soit n fichiers à accès séquentiel F_1, F_2, ..., F_n en entrée d'une phase de traitement.

Soit le fichier F_j ayant j indicatifs : I_1, I_2, ..., I_1, ... I_{j-1}, I_j ordonnés par la relation d'inclusion (cf. chapitre 1 en 1.2.2.2.) (on dira que I_1 occupe le rang 1 dans cette suite).

Règle 1 :

L'indicatif I_1 est commun à tous les fichiers et occupe le rang 1.

Règle 2 :

Si $M_1 \in F$ est l'ensemble des fichiers ayant I_1 parmi leurs indicatifs, il faut que I_1 ait le même rang dans tous ces fichiers et que les indicatifs d'ordre supérieur à I_1 leur soient communs et occupent également le même rang.

Règle 3 :

Soit S l'ensemble des indicatifs associés au fichier de sortie de la phase de traitement. L'ensemble des indicatifs associés à F_j ($1 \leq j \leq n$) doivent appartenir à S et occuper le même rang dans F_j et sur l'arbre d'édition de la sortie.

Chaque fois qu'un fichier ne vérifie pas les règles de compatibilité, il faut ajouter une phase de traitement ayant pour fonction de créer un fichier transmis vérifiant les règles de compatibilité.
Ce fichier transmis sera obtenu à partir d'un ou plusieurs fichiers parmi lesquels figure celui qui ne vérifie pas la règle de compatibilité.

Nous appelons phases organiques les phases de traitement obtenues après application des règles de compatibilité.

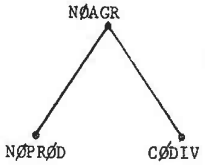
Nous allons appliquer ces règles de compatibilité à la phase de traitement "bordereau de paie du lait".

Voici un extrait de la grille d'évaluation pour cette phase de traitement :

NØTIØNS RESULTAT		ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
				TY	IDENT.	INDICATIFS
QUANTITE	D				QUANTITE	NØAGR, NØART
DESIGN	D			P	DESIGN	NØART
PUHT	D			P	PUHT	NØART
MTHT	C	PUHT * QTE				
TVA 7	C	MTHT * TAUX 7	si CØTVA = 7	P	CØTVA	NØART
TVA 19	C	MTHT * TAUX 19	si CØTVA = 19	C	TAUX 7	
MTTC	C	MTHT + TVA 7	si CØTVA = 7			
	C	MTHT + TVA 19	si CØTVA = 19			

Fichiers en entrée de la phase	Indicatifs
FIM-RELEVE	NØAGR
FIM-ANALYSE	NØAGR
FIM-DIVERS	NØAGR, CØDIV
FIM-ACHATS	NØAGR, NØPRØD
FIP-PRØDUIT	NØPRØD
FIP-AGRIC	NØAGR

L'arborescence associée au fichier de sortie de cette phase de traitement :



On constate que le fichier FIP-PRØDUIT ne vérifie pas les règles de compatibilité. Il possède un indicatif, mais celui-ci n'a pas le rang 1 dans les autres fichiers en entrée et ne constitue pas la racine de l'arbre d'édition.
On va donc introduire une phase supplémentaire "Valorisation des bons de commandes", qui va créer un fichier transmis. Ce fichier transmis figurera en entrée de la phase de traitement "Bordereau de paie du lait" et respectera les règles de compatibilité. Sur ce fichier, on aura pour chaque article commandé : QUANTITE (quantité commandée), DESIGN (désignation du produit), PUHT (prix unitaire hors taxe).

Phase de traitement "Valorisation des bons de commande".

NØTIØNS RESULTAT		ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
				TY	IDENT.	INDICATIFS
QUANTITE	D			M	QUANTITE	NØART, NØAGR
DESIGN	D			P	DESIGN	NØART
PUHT	D			P	PUHT	NØART

Le fichier créé lors de cette phase de traitement aura pour indicatifs : NØART, NØAGR.

Ce fichier sera utilisé par la phase de traitement "bordereau de paie du lait" et aura pour indicatifs NØAGR, NØART.
Ceci implique d'effectuer un tri préalable.

Phase de traitement "Bordereau de paie du lait" après modification.

NØTIØNS RESULTAT		ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
				TY	IDENT.	INDICATIFS
QUANTITE	D			T	QUANTITE	NØAGR, NØART
DESIGN	D			T	DESIGN	NØAGR, NØART
PUHT	D			T	PUHT	NØAGR, NØART
MTHT	C	PUHT * QUANTITE		T	CØTVA	NØAGR, NØART
TVA 7	C	MTHT * TAUX 7	si CØTVA = 1	C	TAUX 7	
TVA 19	C	MTHT * TAUX 19	si CØTVA = 2	C	TAUX 19	

Fichiers en entrée de la phase	Indicatifs
FIM-RELEVE	NØAGR
FIM-ANALYSE	NØAGR
FIM-DIVERS	NØAGR, CØDIV
FIP-AGRIC	NØAGR
FIT-VALØR (fichier transmis par phase de traitement "Valorisation bons de commande").	NØAGR, NØART

Le fichier FIT-VALØR remplace les deux fichiers FIM-ACHATS et FIP-PRØDUIT. Il respecte les règles de compatibilité.

6.3.1.2. Constitution d'unités de traitement maximales :

6.3.1.2.1. Graphe d'enchaînement des phases organiques

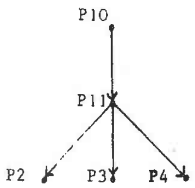
La chaîne de traitement est un ensemble C de phases organiques.

Etant donné P_i, P_j $(i \neq j) \in C$, le fait que la phase P_i transmette un fichier à la phase organique P_j implique une relation de précédence de P_i par rapport à P_j. Cette relation que nous noterons P_i T P_j se lit "P_i précède P_j", elle est antisymétrique.

Nous appellerons graphe d'enchaînement le graphe représentant cette relation.
Ce graphe doit être sans circuit.

Exemple 1 :

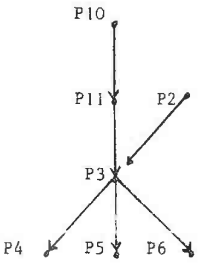
Soit le graphe d'enchaînement de la chaîne de traitement traitée en annexe :



P10 : Valorisation des bons de commande.
P11 : Bordereau de paie du lait.
P2 : Décompte monétaire.
P3 : Etat statistique mensuel.
P4 : Etat de virement bancaire.

Exemple 2 :

Soit le graphe obtenu à partir d'un découpage différent de la même chaîne en phases de traitement lors de l'analyse fonctionnelle (voir 5.1.2.1.2.).



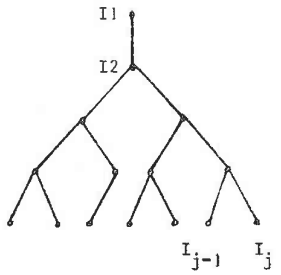
P10 : Valorisation des bons de commande.
P11 : Facture achats.
P2 : Facture relevé.
P3 : Bordereau paie du lait.
P4 : Décompte monétaire.
P5 : Etat statistique.
P6 : Virement bancaire.

6.3.1.2.2. Règles de regroupement des phases organiques en unités de traitement

Pour que le regroupement de plusieurs phases organiques en une unité de traitement soit possible, il faut que les fichiers en entrée de ces phases vérifient certaines règles.

Soit P_1, P_2, ..., P_{n-1}, P_n l'ensemble des phases organiques d'une chaîne de traitement.

Soit I_1, ..., I_{j-1}, I_j l'ensemble des indicatifs associés à une phase et pouvant être représentés sur une arborescence appelée arbre de traitement (cf. chapitre 1).



Une unité de traitement est un ensemble de phases organiques vérifiant les règles suivantes :

. Toutes les phases organiques d'une unité de traitement ont le même indicatif comme racine de leur arbre de traitement.

. Si deux phases organiques ont plusieurs indicatifs communs, ceux-ci doivent figurer en majeur et occuper le même rang sur leurs arbres de traitement respectifs, c'est-à-dire que, si un indicatif I_j est commun à deux phases organiques, tous les indicatifs d'ordre supérieur sur cette branche doivent leur être communs et occuper le même rang.

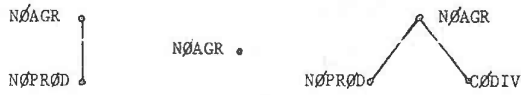
. S'il y a un chemin entre deux phases organiques d'une même unité de traitement, tous les sommets de ce chemin doivent vérifier les règles précédentes.

. Si deux phases organiques appartenant à la même unité de traitement ont même ascendant dans le graphe d'enchaînement, il faut qu'ils aient le même arbre de traitement.

Si nous reprenons l'exemple 1 du paragraphe précédent, nous constatons que seules P2 et P3 vérifient les 4 règles précédentes et seront donc regroupées pour constituer une unité de traitement. Les autres phases organiques constituent chacune une unité de traitement.
Les phases P10 et P11 ne vérifient pas la règle 1 ; en effet la racine de l'arbre de traitement de P10 est NØPRØD (numéro produit) alors que celle de P11 est NØAGR (numéro agriculteur). Il en est de même pour P11 et P2/P3, P11 et P4, P4 et P2/P3.

Si nous reprenons l'exemple 2 du paragraphe précédent, il y a 2 regroupements de phases organiques en unités de traitement. Le premier est réalisé avec P4 et P5, il est identique à celui de l'exemple 1.
Le deuxième est réalisé avec P11, P2, P3, qui proviennent de la partition en trois sous-ensembles de l'ensemble des opérations contenues dans la phase organique P2 de l'exemple 1.

Les arbres de traitement des phases P11, P2, P3 sont respectivement :



P11, P2, P3 ont même racine.
P11 et P3 ont deux indicatifs communs, qui occupent le même rang dans les deux arbres de traitement.

En conséquence, le regroupement de ces trois phases est possible. Nous constatons que, malgré un découpage différent en phases de traitement au niveau de l'analyse fonctionnelle, l'application des règles de regroupement aboutit à un même découpage de la chaîne en unités de traitement.

6.3.1.2.3. Les différents types de regroupement des phases organiques

- Regroupement par suppression de données transmises.
  On a un regroupement de ce type chaque fois que figurent dans la même unité de traitement deux phases organiques reliées par un arc dans le graphe G_E. En effet les données transmises par la première phase à la seconde n'auront plus à être représentées sur un fichier et deviennent des données intermédiaires.

  Dans l'exemple 2 présenté auparavant, le regroupement de P2 et P3 appartient à ce type.

- Regroupement des données transmises.
  On est dans ce cas lorsque les deux phases organiques ont même ascendant dans le graphe G_E.
  Au lieu d'avoir deux fichiers transmis à deux phases organiques différentes, on aura un fichier transmis à une unité de traitement regroupant les deux phases organiques précédentes.

  Dans l'exemple 1, le regroupement de P2 et P3 appartient à ce type.

- Regroupement des traitements.
  Tous les regroupements n'appartenant pas aux deux types précédents peuvent être classés dans ce troisième type de regroupement.
  Le regroupement de P11 et P2 dans l'exemple 2 en fait partie.
  Sa caractéristique essentielle est de ne pas entraîner automatiquement la fusion des données transmises par les deux phases organiques en un seul fichier.

6.3.1.2.4. Organisation des unités de traitement dans la chaîne

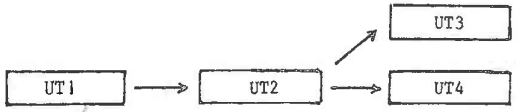
Soit G_U le graphe défini comme suit :

. les sommets de G_U représentent les unités de traitement

. un arc joint deux sommets de G_U dès qu'une phase organique de la première unité de traitement est dans G_E l'extrêmité initiale d'un arc dont l'extrêmité terminale est une phase organique de la seconde.

G_U représente les contraintes d'antériorité entre les unités de traitement et doit être sans circuit.

Dans la chaîne de traitement "paie du lait" on a le graphe suivant :



La composition des unités de traitement pour l'exemple 1 sera la suivante :

UT1 : P1O
UT2 : P11
UT3 : P2 et P3
UT4 : P4

6.3.1.2.5. Constitution des fichiers transmis

Chaque fois qu'un arc joint deux sommets de G_U, il y aura un fichier transmis entre les deux unités de traitement correspondantes. Le fichier transmis sera obtenu par la réunion des segments transmis par les phases organiques figurant dans la première unité de traitement vers des phases organiques figurant dans la deuxième unité de traitement.

Exemple :

Constitution du fichier transmis par UT2 à UT3.

Il sera obtenu par la réunion des segments transmis par P11 (UT2) respectivement à P2 et P3 (UT3).

Segment transmis par P11 à P2
SEG-T11-R2-DECØMPTE struct (NØM, PRENØM, ADRESSE, TØUR, APAYER) ;

Segment transmis par P11 à P3
SEG-T11-R3-STATS struct (NØM, PRENØM, ADRESSE, TØUR, TØTLI, VALHT, TVA, NET-LAIT) ;

Le fichier transmis par UT2 à UT3 est obtenu par la réunion des deux segments précédents :

FI-DEC-STATS ensemble struct (SITUATIØN, S-LAIT, APAYER) ;

6.3.1.3. Détermination des unités de traitement définitives :

On va faire intervenir à ce moment deux éléments qui vont limiter le regroupement des phases organiques effectué précédemment.

6.3.1.3.1. Simplicité des programmes

Cela conduit en général à limiter le nombre des opérations effectuées dans une unité de traitement pour faciliter la réalisation, la mise en oeuvre et la maintenance de cette unité de traitement. Cet objectif est celui visé par CIVA en proposant la modularité de la description des actions et des informations.
Il suffit donc de découper l'unité de traitement en modules qui seront simples à réaliser, à mettre en oeuvre et à maintenir, tout en conservant l'unité de traitement initiale comme unité de fonctionnement de la machine.

6.3.1.3.2. Configuration de l'ordinateur

- Contraintes liées au nombre de périphériques.

Il peut arriver que certaines unités de traitement nécessitent plus de périphériques d'un type donné que la configuration n'en dispose.
Par exemple, le traitement effectué dans une unité de traitement nécessite cinq dérouleurs de bandes, alors que la machine n'en possède que quatre.
Il faut dans ce cas procéder à un découpage de l'unité de traitement et à la création d'un fichier transmis.

- Contraintes imposées par la taille de la mémoire centrale.

En effet, un programme ne peut fonctionner sur une machine que dans la mesure où l'on dispose d'une place en mémoire suffisante pour le charger.
Là également, la description modulaire des informations permet de n'avoir à un instant donné en mémoire centrale que les zones de travail nécessaires au module en cours d'exécution et diminue donc l'occupation en mémoire centrale des programmes.
Ce problème a été traité dans DENDIEN (6) et DERNIAME (1).

6.3.2. Organisation interne d'une unité de traitement :

L'unité de traitement est une unité de description particulière des actions car elle correspond à l'unité de fonctionnement de l'ordinateur mais elle n'est pas la seule. C'est ce que nous allons voir dans la suite de ce paragraphe.

6.3.2.1. Les modules de traitement indexés :

6.3.2.1.1. Définition

A chaque unité de traitement est associé un ensemble d'indicatifs ordonnés selon un arbre de traitement (voir chapitre 1).

A chaque indicatif qui est une feuille de l'arbre de traitement est associé un seul module de traitement.
A tout indicatif autre qu'une feuille de l'arbre de traitement sont associés deux ou plusieurs modules de traitement :

. un module de traitement initial,

. un ou plusieurs modules de traitement final (il y en a plusieurs chaque fois que l'indicatif a plusieurs successeurs dans l'arbre de traitement).

Ces modules sont exécutés pour chaque réalisation de l'indicatif associé et seront appelés modules de traitement indexés.

Soit un programme ayant deux indicatifs :

numéro usine

numéro service

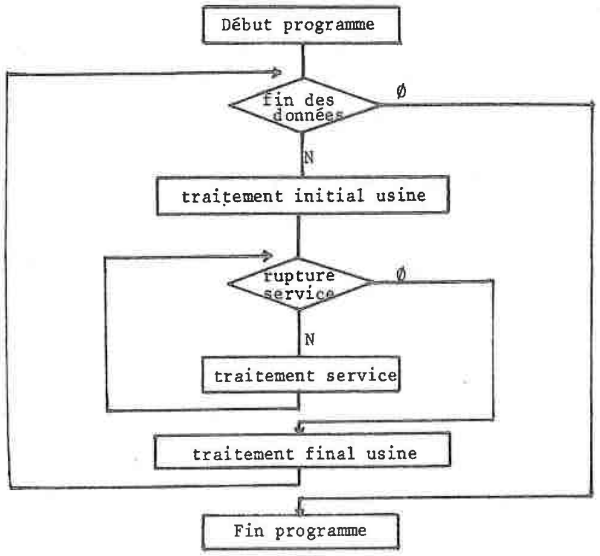
On a un module de traitement indexé associé à "numéro service".
On a deux modules de traitement indexés associés à "numéro usine" :

. le module initial qui sera exécuté pour chaque valeur de "numéro usine" avant l'exécution du module associé à "numéro service" pour tous les "numéros service" appartenant à cette usine.

. le module de traitement final qui sera exécuté après le module associé à "numéro service".

On a enfin deux modules, l'un exécuté une fois au début du programme, l'autre exécuté une fois à la fin du programme.

La structure du programme peut être représentée par le schéma suivant :



6.3.2.1.2. Affectation des opérations aux modules de traitement indexés

Le problème va consister à partager entre les différents modules de traitement indexés d'une unité de traitement les opérations définies lors de l'analyse fonctionnelle.
La règle d'affectation des opérations est la suivante :

a) Opérations autres que les opérations de transfert :

Chaque opération est affectée au module attaché à l'indicatif de plus bas niveau parmi les indicatifs des notions en entrée de l'opération.
Si cet indicatif n'est pas une feuille de l'arbre de traitement, l'opération sera affectée en principe au module de traitement initial, sauf si cette opération utilise une notion d'entrée obtenue dans un module de niveau inférieur.

b) Opérations de transfert :

Une opération de transfert est affectée au module attaché à l'indicatif de plus bas niveau parmi les indicatifs de la notion résultat de l'opération.
Elle sera placée systématiquement dans le module de traitement initial.

Exemple :

Nous allons appliquer les règles d'affectation aux opérations de la phase organique P3 qui figure dans l'unité de traitement UT3.
La branche de traitement de l'UT3 est la suivante :

TØUR

NØAGR

et on aura donc les trois modules de traitement indexés :

1 - module de traitement initial attaché à TØUR ;
2 - module de traitement attaché à NØAGR ;
3 - module de traitement final attaché à TØUR.

Grille d'évaluation de la phase organique P3 :

	NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
				TY	IDENT.	INDICATIFS
1	TØUR			T	TØUR	TØUR, NØAGR
2	NØM			T	NØM	TØUR, NØAGR
2	PRENØM			T	PRENØM	TØUR, NØAGR
2	ADRESSE			T	ADRESSE	TØUR, NØAGR
2	TØTLI			T	TØTLI	TØUR, NØAGR
2	VAL-HT			T	VAL-HT	TØUR, NØAGR
2	TVA			T	TVA	TØUR, NØAGR
2	NET-LAIT			T	NET-LAIT	TØUR, NØAGR
2	CUM-LI	CUM-LI + TØTLI				

	NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
				TY	IDENT.	INDICATIFS
2	CUM-VAL-HT	CUM-VAL-HT + VAL-HT				
2	CUM-TVA	CUM-TVA + TVA				
2	CUM-NET	CUM-NET + NET-LAIT				

Le fichier obtenu en sortie de P3 a la structure suivante :

NØM
PRENØM
ADRESSE
TØTLI
VAL-HT
TVA
NET-LAIT

} Groupe de sortie 1, indicatifs : TØUR, NØAGR.

TØUR
CUM-LI
CUM-VAL-HT
CUM-TVA
CUM-NET

} Groupe de sortie 2, indicatif : TØUR.

A gauche de la grille d'évaluation, nous avons mis le numéro du module de traitement indexé auquel sera affectée l'opération.

TØUR étant obtenu par transfert figure dans le module de traitement initial attaché à l'indicatif du groupe de sortie dans lequel figure TØUR.

NØM, PRENØM, ADRESSE, TØTLI, VAL-HT, TVA, NET-LAIT, sont obtenus par transfert et sont affectés au module de traitement associé à l'indicatif de plus bas niveau du groupe de sortie les regroupant : NØAGR.

CUM-LI, CUM-VAL-HT, CUM-TVA, CUM-NET sont obtenus par un calcul et on leur applique la règle d'affectation b).

6.3.2.1.3. Organisation interne d'un module de traitement indexé

L'ensemble des opérations constituant un module de traitement indexé peut être décomposé en sous-ensembles indépendants qui seront décrits dans des modules appelés par le module de traitement indexé.

Ces modules aurons souvent été mis en évidence lors de l'analyse fonctionnelle (voir 5.2.1.3.).

De plus, les unités de traitement provenant d'un regroupement de phases organiques, on peut vouloir regrouper sous forme d'un module les opérations appartenant à une même phase organique. Cela serait fort utile du point de vue de la maintenance et de la documentation, car toute modification d'une opération d'une unité de traitement pourrait être représentée dans la phase de traitement correspondante au niveau de l'analyse fonctionnelle.

6.3.2.2. Opérations logiques et opérations technologiques :

Nous appelons opérations logiques les opérations décrites lors de l'analyse fonctionnelle.

Les opérations technologiques, ce sont les opérations qui gèrent le transfert des données entre la mémoire centrale et un fichier.

Il existe trois catégories d'opérations technologiques :

- Les opérations technologiques d'entrée :

  Elles réalisent le transfert des données entre un fichier d'entrée et la mémoire et comprennent les opérations de lecture des fichiers.
  Elles seront réalisées grâce aux instructions de cinématique des fichiers que nous avons définies dans la première partie et qui, de plus, réalisent la gestion des modules de traitement indexés.
  Ces instructions figureront en général dans le module directeur de l'unité de traitement (un exemple est donné ci-dessous).

- Les opérations technologiques de sortie :

Elles gèrent le transfert des informations de la mémoire vers un fichier de sortie et l'édition.

Tout ce qui concerne l'édition des fichiers de sortie est réalisé par les services d'édition de CIVA (CHABRIER J.J. (5)).

Il suffira donc de créer le fichier de sortie qui sera édité par la suite. Ceci impose donc de prévoir pour chaque fichier de sortie une opération d'écriture de ses articles sur le support de sortie.

Cette opération est réalisée par l'instruction de progression de l'élément courant attaché au fichier dans le cas d'un traitement séquentiel (DERNIAME J.C. (1)).

Exemple :

Soit le fichier de sortie S.
X de S (co déclaration de l'élément courant X oc)
L'instruction : X en X + 1 ; écrit un nouvel article sur le fichier de sortie S à la suite des autres.

Cette instruction devra figurer dans le module de traitement final attaché à l'indicatif mineur du fichier de sortie.

Dans l'exemple du paragraphe 2.2.1.2. l'index (ensemble des indicatifs permettant d'identifier un article d'un fichier) du fichier de sortie est : TØUR.
Par conséquent, l'instruction de progression de l'élément courant figure dans le module de traitement final attaché à l'indicatif : TØUR.

- Les opérations technologiques annexes :

. L'ouverture et la fermeture des fichiers.

L'ouverture d'un fichier est effectuée dès qu'un module utilise la classe contenant la déclaration de ce fichier ; la fermeture du fichier est effectuée à la fin de l'exécution de ce module.

. Le déroutement et la reprise de l'exécution d'une unité de traitement lors de la détection d'une anomalie ou d'une erreur.

Il est prévu, au niveau de la classe système, une action par défaut à entreprendre systématiquement dès qu'un tel événement se réalise, tout en laissant la possibilité de décrire d'autres séquences de récupération si on le désire (DERNIAME J.C. (1) en 10.42).

6.3.2.3. Déclaration des informations utilisées dans une unité de traitement :

6.3.2.3.1. Déclaration des fichiers

Les fichiers contiennent des données utilisées par tous les modules d'une ou plusieurs unités de traitement. En conséquence, ils seront déclarés dans des classes utilisées par les modules directeurs.

Un fichier sera composé d'un ou plusieurs segments logiques mis en évidence au cours de l'analyse fonctionnelle.

Exemple :

Le fichier transmis par UT2 à UT3 est obtenu à partir de deux segments logiques :

SEG-T12-DECØMPTE struct (NØM, PRENØM, ADRESSE, TØUR, APAYER) ;
SEG-T12-STATS struct (NØM, PRENØM, ADRESSE, TØUR, TØTLI, VAL-HT, TVA, NET-LAIT) ;

et sera déclaré ainsi :

FI-DEC-STATS ensemble DEC-STATS struct (SEG-T12-STATS, APAYER).

Un des avantages de la description modulaire des informations indépendamment de leur utilisation est de permettre le partage de la description des fichiers entre toutes les unités de traitement utilisatrices.

De plus, la description est fortement simplifiée car il suffit de donner la liste des segments logiques, notions structurées et notions élémentaires le constituant et qui ont été décrits dans la classe d'application au moment de l'analyse fonctionnelle.

6.3.2.3.2. Déclaration des autres informations

Elle concerne d'une part les notions intermédiaires et les notions non indexées utilisées au cours de l'analyse fonctionnelle, d'autre part les variables de travail nécessaires lors de l'écriture des modules en langage source CIVA.

Il y a les informations qui ne sont utilisées que dans un seul module et qui peuvent être déclarées à l'intérieur du module concerné.
Mais si l'on veut rendre communes à plusieurs modules des déclarations d'identificateurs et donc les descriptions de propriétés d'objets créés, il faut les effectuer dans une classe utilisée par ces modules. La description modulaire des objets n'est pas seulement nécessaire pour la description elle-même mais également pour le partage des objets. La déclaration d'un objet écrite dans une classe permet aux modules utilisant cette classe de communiquer des valeurs pendant leur exécution. La durée de vie de cet objet dépend de la durée d'exécution de l'ensemble des modules qui utilisent la classe qui les déclare.

Exemple :

Soit deux modules M1 et M2 utilisant la classe C définie par :

```
Classe C  :  S entier ; A entier ; B entier ;
```


a) soit les modules suivants :

```
Module  CALC1  ;
    utilise C ;
    S = A + B ;          fin module ;

Module  CALC2  ;
    utilise C ;
    S = S/2 ;            fin module ;
```

Il y a mise en commun entre CALC1 et CALC2 de la description de S, mais on est en présence de deux objets différents désignés par S à des instants différents. Il n'y a pas de communication de valeur entre les deux modules.

b) soit les deux modules CALC1 et CALC2 et un module CALC défini ainsi :

```
Module  CALC ;
    utilise C ;
    CALC1 ; CALC2 ;      fin module ;
```

Il y a dans ce cas un seul objet désigné par S dans CALC1 et CALC2 et communication de valeur entre les deux modules. La durée de vie de S est égale à la durée d'activation du module CALC.

c) il y a une relation "appelle" entre CALC1 et CALC2 :

```
Module  CALC1 ;
    utilise C ;
    S = A + B ;
    CALC2 ;              fin module ;

Module  CALC2 ;
    utilise C
    S = S/2 ;            fin module ;
```

S désigne un seul objet dans CALC1 et CALC2, car l'objet créé dans CALC1 a une durée de vie égale à la durée d'activation de ce module et recouvre la durée d'activation du module CALC2.

On constate donc que, si on veut rendre communes à plusieurs modules des descriptions de propriétés d'objets créés, il suffit de les décrire dans une classe utilisée par ces modules.

Mais si on veut réaliser le partage des objets créés dans une classe entre les modules utilisant cette classe, il faudra vérifier la durée de vie de ces objets.

Ces problèmes de conditions de validité des identificateurs et de durée de vie des objets sont traitées dans DERNIAME J.C. (1).

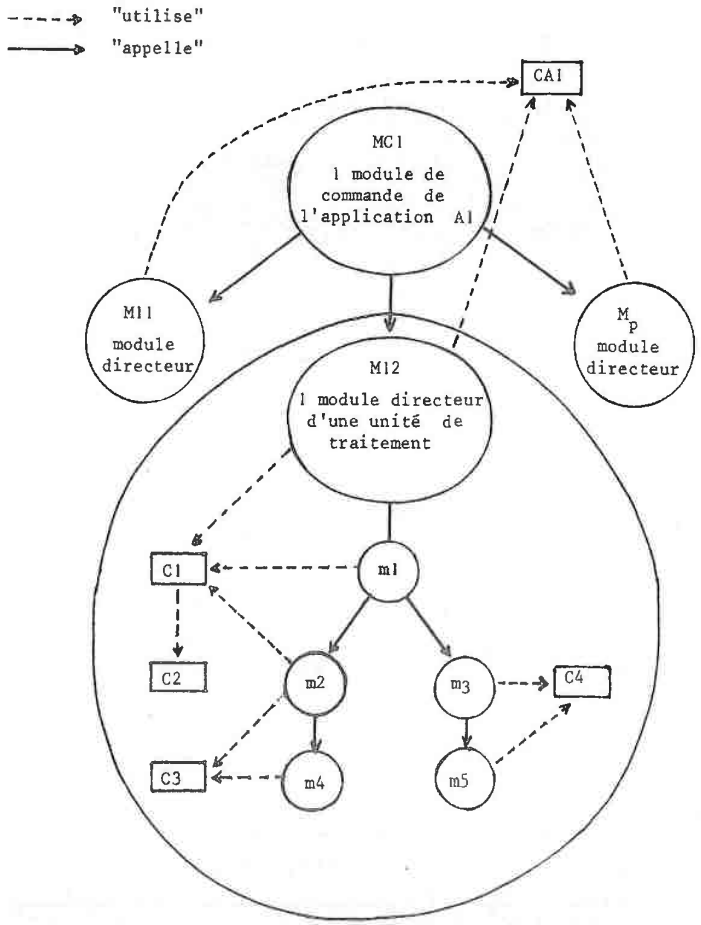
6.3.3. La mise en oeuvre de la chaîne de traitement et des unités de traitement (DERNIAME J.C. (1), VIAULT D. (12)) :

La mise en oeuvre de la chaîne de traitement va consister à demander l'exécution successive des unités de traitement la constituant.

Cela se fait grâce au module de commande dans lequel on aura une suite d'appels de modules qui sont les différentes unités de traitement de la chaîne.

Les modules appelés par le module de commande sont les premiers modules de traitement, nous dirons aussi les modules directeurs.

Cela peut être schématisé ainsi :



Exemple :

Soit la chaîne de traitement présentée en annexe ; son module de commande sera le suivant :

Module MC1 ;
Application paie du lait ;
UT1 ; UT2 ; UT3 ; UT4 fin module ;

La mise en oeuvre d'une unité de traitement va consister en une suite d'appels de modules telle qu'elle est représentée sur le schéma ci-dessus.

Exemple :

Soit l'unité de traitement UT3 de la chaîne traitée en annexe.
On aura le module directeur :

```
Module UT3 ;
    utilise
    DEBUT-UT1 ;
    Pour chaque TØUR faire TI-TØUR ;
                Pour chaque NØAGR faire TR-AGRIC   fpc ;
                            TF-TØUR   fpc ;
    FIN-UT1    fin module ;
```

Dans ce module directeur, on réalise la cinématique des fichiers et on appelle les différents modules de traitement indexés mis en évidence lors de l'analyse organique.

6.4. - CONCLUSION

Ce chapitre nous a permis essentiellement de montrer comment on peut passer des phases de traitement (analyse fonctionnelle) aux unités de traitement d'une chaîne.

Ceci est réalisé en appliquant un ensemble de règles en deux étapes :

- les règles de compatibilité des fichiers en entrée d'une phase de traitement qui vont nous amener à partager une phase de traitement en plusieurs phases organiques en fonction des indicatifs associés aux différents fichiers (voir 6.3.1.1.).

- les règles de regroupement des phases organiques qui permettent de regrouper dans une même unité de traitement différentes phases organiques en se basant sur certaines propriétés de leurs arbres de traitement (voir 6.3.1.2.2.).

CONCLUSION

Les outils d'analyse que nous avons présentés ont été essayés sur plusieurs applications.

Nous avons constaté que le passage de l'analyse à la programmation ne posait aucun problème et que les descriptions des informations et des actions faites au niveau de l'analyse fonctionnelle pouvaient être réutilisées lors de la programmation. Il en est ainsi pour les descriptions des segments logiques et des notions structurées et élémentaires figurant dans la classe répertoire, de même que pour les opérations décrites dans les modules d'analyse fonctionnelle et dans les tables de décision.

Pour que l'utilisation de ces outils devienne opérationnelle, il faudra concevoir un certain nombre de documents standards (du type "grille d'évaluation") pour guider l'analyste lors de son travail.

Enfin, il nous semble que ces outils pourraient être développés dans le sens d'une utilisation de la machine dès l'analyse.
Ainsi, les problèmes de vérification des règles de compatibilité, de regroupement des phases organiques en unités de traitement, d'affectation des opérations aux modules de traitement indexés, de génération des opérations technologiques et des instructions d'itération logique pourraient certainement être résolus automatiquement à partir de la grille d'évaluation établie lors de l'analyse fonctionnelle.

Ce développement des outils d'analyse est un travail auquel nous pensons nous attacher.

ANNEXES

ANALYSE EN CIVA D'UNE APPLICATION

"L A P A I E D U L A I T"

I - PRESENTATION DE L'APPLICATION.

II - SCHEMA DE CIRCULATION ET DOCUMENTS DE SORTIE.

III - ANALYSE FONCTIONNELLE.

IV - ANALYSE ORGANIQUE.

I - PRESENTATION DE L'APPLICATION

INTRODUCTION :

Cette application provient de la Société MARCILLAT, qui est une entreprise de transformation laitière qui collecte le lait des agriculteurs et le transforme en divers produits laitiers.

L'application concerne plus particulièrement ce qu'on appelle "la paie du lait" et qui consiste à établir mensuellement pour chaque agriculteur, un document (bordereau de paie du lait) comportant :

- le relevé des quantités de lait livré, sa qualité, son prix ;
- la facture éventuelle des produits rétrocédés achetés par l'agriculteur à la laiterie ;
- le relevé du compte de l'agriculteur.

En plus du bordereau de paie du lait, on établit un état de décompte monétaire, un état de virement bancaire, un état statistique mensuel, un état statistique annuel.

DESCRIPTION DES PROCEDURES :

1) - Le ramassage du lait :

Il est quotidien et effectué en plusieurs tournées. Chaque fournisseur de lait appartient de façon permanente à une tournée de ramassage. Par tournée, le collecteur relève quotidiennement sur un bordereau de ramassage la quantité livrée par chaque fournisseur. Il y a environ 40 tournées.

2) - Le contrôle de la qualité :

Trois fois par mois, le ramasseur prélève un peu de lait de chaque fournisseur et la laiterie l'envoie à un laboratoire d'analyses.

Pour chaque fournisseur, le laboratoire note sur un bordereau les résultats des trois analyses du mois.

Il y a trois sortes d'analyses à chaque prélévement :

- teneur en matière grasse (nombre de grammes de matière grasse par litre) ;
- teneur en matière azotée (nombre de grammes d'extraits secs par litre) ;
- qualité bactériologique (une note valant 1, 2 ou 3).

3) - Calcul du prix du lait :

Eléments intervenant dans ce calcul :

a) Le prix de base du litre de lait fixé par la Commission Intersyndicale (agriculteurs et industriels) pour un an.
Un prix est prévu pour chaque mois.

b) Le contrat passé entre la laiterie et le fournisseur. Le fournisseur peut choisir entre une subvention du Ministère de l'Agriculture ou vendre la lait T.T.C. avec taux de T.V.A. de 7,5 %.

c) Les primes calculées à partir des résultats de l'analyse :

- la prime ou la réfaction par gramme pour un taux moyen de matière grasse supérieur ou inférieur à 34 g.
- la prime ou la réfaction par gramme pour une teneur moyenne en matière azotée supérieure ou inférieure à 31 g.
- la prime pour les notes 9 et 8 de la qualité bactériologique et la réfaction pour les notes 4 et 3.
Ces notes sont obtenues en faisant la somme des 3 notes obtenues lors des 3 relevés.

d) Les primes par litre pour des laits ayant des propriétés particulières :

- prime de contrôle laitier.
- prime de basse température si le lait est réfrigéré en tanks.
- prime d'étable patentée si l'étable est contrôlée régulièrement par un vétérinaire.
- prime de montagne si l'agriculteur habite une zone de montagne.

4) - Facturation des produits rétrocédés :

La laiterie vend à ses fournisseurs un certain nombre de produits :

- des produits laitiers fabriqués par la laiterie,
- du matériel laitier et des aliments de bétail.

Le prix de ces produits vient en déduction de la somme due au titre du lait.

5) - Divers financiers :

Ce sont des opérations financières entre la laiterie et le fournisseur :

- remboursement d'acomptes ou d'avances,
- saisie-arrêt,
- retenues pour achat de tanks,
- dûs antérieurs.

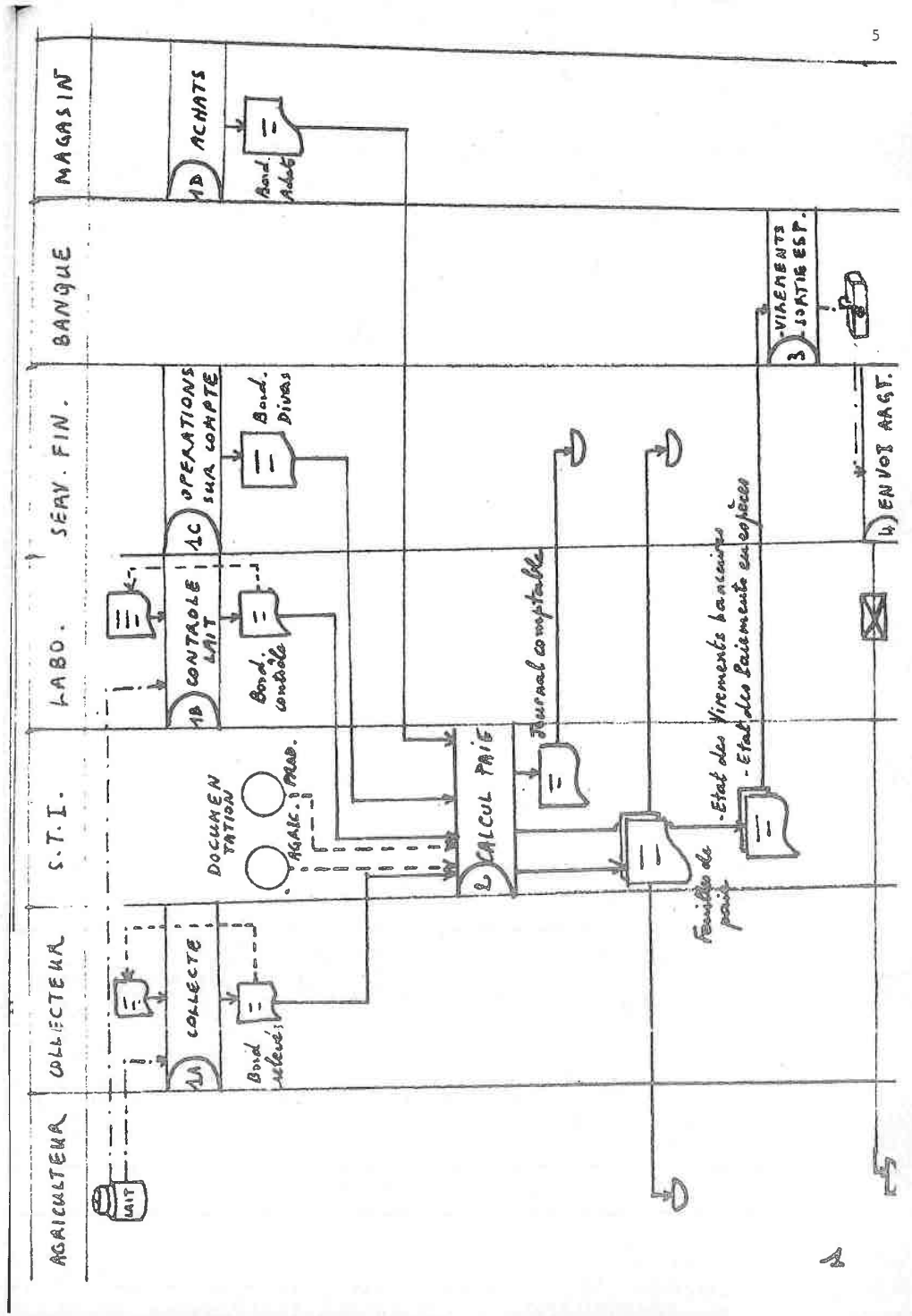
LISTE DES DOCUMENTS D'ENTREE ET DE SORTIE :

1) - Documents d'entrée :

- bordereau de ramassage,
- bordereau d'analyse,
- bordereau des opérations financières
- bordereau des achats effectués par l'agriculteur.

2) - Documents de sortie (voir pages suivantes) :

- bordereau de paie du lait,
- état de décompte monétaire,
- état de virement bancaire,
- état statistique mensuel,
- état statistique annuel.



arcillat s. a. DOIVENT A M.

POUR ENLÈVEMENTS DU LAIT ENTIER
DU MOIS DE

NOM — ADRESSE	RÉPAR	TOURN.	N° PRODUCTEUR

ELEVÉ DE LAIT

DU LITRE H. T. =
4 + 5 + 6 + 7 + 8 + 9 + 10

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MATIÈRE GRASSE				QUALITÉ BACTÉRIOLOGIQUE				MATIÈRE AZOTÉE				1 PRIX DE BASE	2 ± MATIÈRE GRASSE	3 ± QUALITÉ BACTÉRIO.	4 ± MATIÈRE AZOTÉE
1	2	3	TAUX REPÈRE	1	2	3	TAUX REPÈRE	1	2	3	TAUX REPÈRE	C	C	C	C

5 CONTRÔLE LAITIER	6 BASSE TEMPÉR.	7 ÉTABLE PATENTÉE	8 MONTAGNE	9	10 RÉGULARITÉ LITRAGE RÉFÉRENCE	%	PRIME	PRIX DU LITRE H. T.	TOTAL LITRAGE	VALEUR DU LAIT H. T.	DIVERS H. T. (1)	MONTANT H. T.	T.V.A. 7,00	NET A PAYER T.T.C.
C	C	C	C	C			C	C						

(1)

---

arcillat S.A.

DE 4.272.600 FRANCS
CORCIEUX
AINT-DIÉ 54 8 11 —
ONE : 87.06.29
433 88 115 1001
NANCY 303-27

CTURE

QUANTITÉS (Pièces)	DÉSIGNATION	PRIX UNITAIRE HORS TAXE	MONTANT HORS TAXE	T.V.A. 7 %	T.V.A. 19 %		NET A PAYER T. T. C.

N° ____ MOIS DE ____

---

arcillat s. a.

DIVERS	+	−

RELEVÉ DE COMPTE

IDENTIFICATION			MOIS DE	MODE DE RÈGLEMENT	VOTRE FACTURE DE LAIT	A DÉDUIRE		SOLDE	
Cde	TOURN.	PRODUCT.				VOTRE FACTURE DE PRODUITS	ACOMPTES	A REPORTER	A PAYER

ETAT STATISTIQUE MENSUEL

N° PROD.	NOM PRENOM ADRESSE	LITR.	VAL. H.T.	T.V.A.	NET

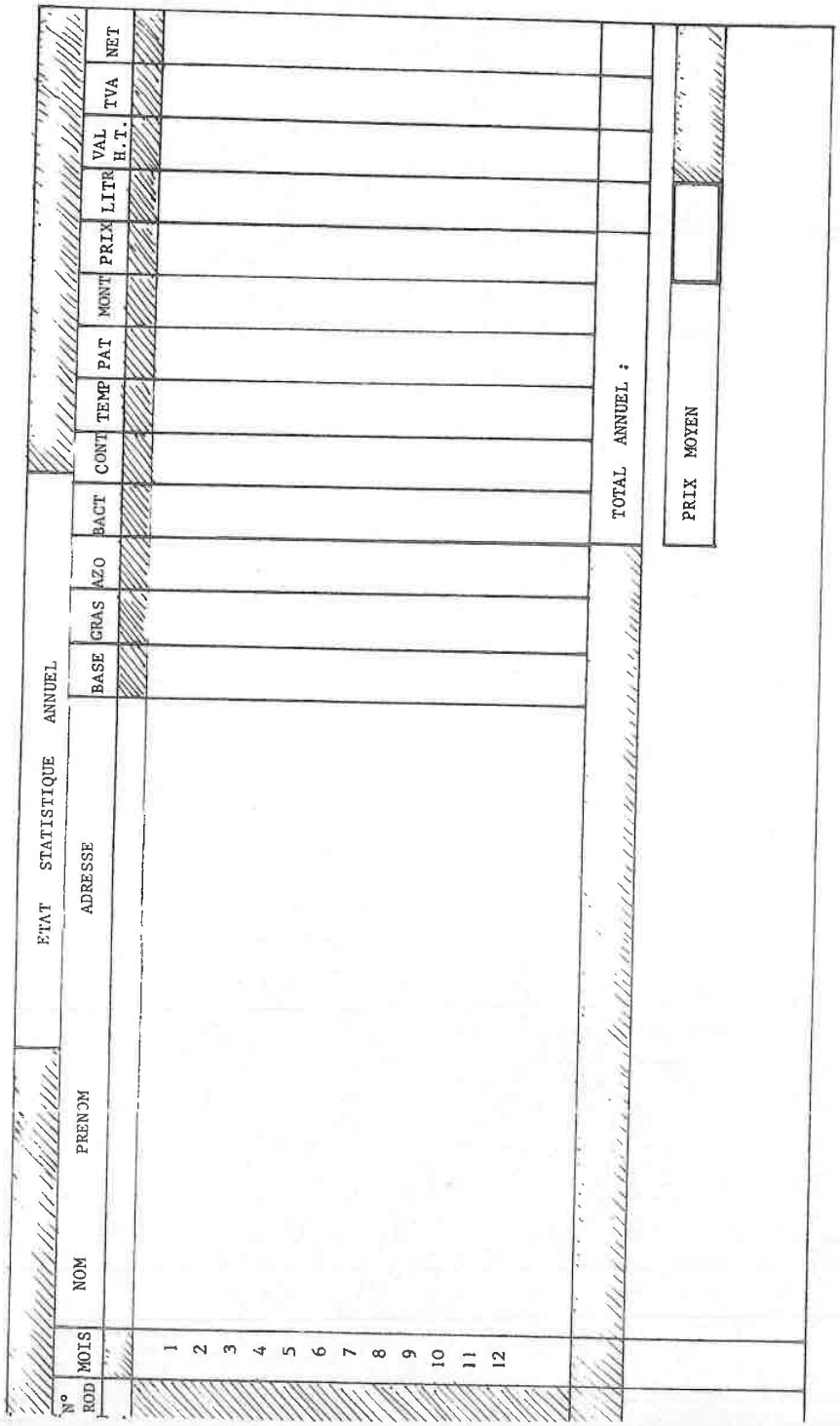
TOTAL TOURNEE N°

ETAT DE VIREMENT BANCAIRE

N° OD.	NOM PRENOM ADRESSE	DOMICILIATION BANCAIRE			
		BANQUE	AGENCE	N° COMPTE	A PAYER
	TOTAL AGENCE				

DECOMPTE MONETAIRE

N° OD.	NOM PRENOM ADRESSE	A PAYER	500 F	100 F	50 F	10 F	5 F	1 F	0,50 F	0,20 F	0,10 F
	TOTAL TOURNEE N°										
	TOTAL GENERAL										



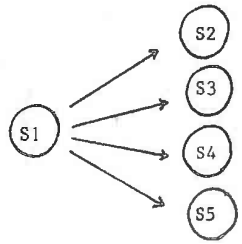
III - ANALYSE FONCTIONNELLE

1) - Définition des sorties de l'application "paie du lait" :

a) Les sorties fournies par le système de traitement de l'information à l'extérieur.

- Bordereau "paie du lait"(S1) : il est envoyé aux producteurs.
- Etat de décompte monétaire (S2).
- Etat de virement bancaire (S3).
- Etat statistique mensuel (S4).
- Etat statistique annuel (S5).

b) Le graphe défini sur les sorties sera le suivant :



c) Découpage en chaînes de traitement :

La chaîne de traitement mensuelle établit les sorties suivantes : S1, S2, S3, S4.

La chaîne de traitement annuelle établit S5.

d) Sorties réutilisées par le système de traitement de l'information dans un cycle ultérieur :

- Fichier créé par chaîne mensuelle et utilisé par chaîne annuelle (pour calculer S5) : S6

2) - Description des fichiers de sortie de l'application dans la classe répertoire :

a) Bordereau de sortie "paie du lait" (SI) :

```
type ART-PAIE struct (GR-LØG 1 struct (SITUATIØN struct
                     (NØAGR, TØUR, NØM, PRENØM, ADRESSE),
                     ANALYSE struct (MAT-GRAS, MAT-AZ, QU-BAC),
                     RELEVE,
                     TM struct (TM-GRAS décimal 999 V 9,
                                TM-AZ   décimal 999 V 9,
                                TM-BAC  décimal  9),
                     PR-BASE,
                     PR struct (PR-GRAS décimal  9 V 99,
                                PR-AZ   décimal  9 V 99,
                                PR-BAC  décimal  9 V 99,
                                PR-CTR  décimal  9 V 99,
                                PR-TEMP décimal  9 V 99,
                                PR-PAT  décimal  9 V 99,
                                PR-MØN  décimal  9 V 99,
                                PR-LAIT décimal  99 V 99),
                     S-LAIT struct (TØTLI décimal 9 (5),
                                    VALHT décimal 9 (5) V 99,
                                    TVA   décimal 9 (4) V 99,
                                    NET-LAIT décimal 9 (5) V 99)),
```

Groupe logique de sortie 1 indicatif : NØAGR

```
                     GR-LØG 2 struct (LIGNE struct (QTE, DESIGN,
                                      PRIX, MTHT, NØART),
                     S-LIGNE  struct (TVA 7 décimal 999 V 99,
                                      TVA 19 décimal 999 V 99,
                                      MTTC   décimal 9 (4) V 99),
```

Groupe logique de sortie 2 indicatifs : NØAGR, NØART

```
                     GR-LØG 3 struct (STVA 7 décimal 9 (4) V 99,
                                      STVA 19 décimal 9 (4) V 99,
                                      NAP     décimal 9 (5) V 99),
```

Groupe logique de sortie 3 indicatif : NØAGR

```
                     GR-LØG 4 struct (DIVERS struct (CØDIV,
                              LIBDIV, MDIV, PDIV),
```

Groupe logique de sortie 4 indicatifs : NØAGR, CØDIV


```
                     GR-LØG 5 struct (LI-MØIS file (10) car,
                                      MØDREG  file  (2) car,
                                      VFL     type NET-LAIT,
                                      NFP     type NAP,
                                      ACØMPTES décimal 9 (4) V 99,
                                      SMDIV   décimal 9 (4) V 99,
                                      SPDIV   décimal 9 (4) V 99,
                                      APAYER  décimal 9 (5) V 99,
                                      REMB    décimal 9 (5) V 99)) ;
```

Groupe logique de sortie 5 indicatif : NØAGR

b) Etat de décompte monétaire (S2) :

```
type ART-DEC struct GR-LØG 1 struct (SITUATIØN, APAYER,
                    T-ESP file (9) décimal 99),
```

Groupe logique de sortie 1 indicatifs : TØUR, NØAGR

```
                    GR-LØG 2 struct (CUM-T-ESP file (9) décimal 9 (3)),
```

Groupe logique de sortie 2 indicatif : TØUR

```
                    GR-LØG 3 struct (CUM-G-ESP file (9) décimal 9 (4))) ;
```

Groupe logique de sortie 3

c) Etat de virement bancaire (S3) :

```
type ART-VIR struct (GR-LØG 1 struct (SITUATIØN, APAYER, BANQUE),
                    GR-LØG 2 struct (CUM-GUI décimal 9 (6) V 99),
                    GR-LØG 3 struct (CUM-GEN décimal 9 (8) V 99)) ;
```

Groupe logique de sortie 1 indicatifs : CØGUI, NØAGR

Groupe logique de sortie 2 indicatif : CØGUI

Groupe logique de sortie 3

d) Etat statistique mensuel (S4) :

```
type ART-STAT struct
                    (GR-LØG 1 struct (SITUATIØN, S-LAIT),
```

Groupe logique de sortie 1 indicatifs : TØUR, NØAGR

Groupe logique de sortie 2 indicatif : TØUR

```
GR-LØG 2 struct (TØUR,
                 CUM-LI décimal 9 (8),
                 CUM-VALHT décimal 9 (8) V 99,
                 CUM-TVA décimal 9 (7) V 99,
                 CUM-NET décimal 9 (8) V 99)) ;
```

e) Etat statistique annuel (S5) :

Groupe logique de sortie 1 indicatifs : NØAGR, MØIS

```
type FIS-AN struct (GR-LØG 1 struct (SITUATIØN, PR, S-LAIT),
```

Groupe logique de sortie 2 indicatif : NØAGR

```
GR-LOG 2 struct (PR-MØY décimal  99 V 99,
                 C-TØTLI décimal  9 (6),
                 C-VALHT décimal  9 (6) V 99,
                 C-TVA  décimal   9 (5) V 99,
                 C-NET-LAIT décimal 9 (6) V 99)) ;
```

f) Fichier "Récapitulatif agriculteur" (S6) :

C'est le fichier obtenu en sortie de la chaîne de traitement mensuelle et réutilisé par la chaîne de traitement annuelle.

```
type ART-RECAP struct (SITUATIØN, PR, S-LAIT) ;
```

3) - Découpage en phases de traitement :

a) chaîne mensuelle :

. Phase 1 : établit le bordereau "paie du lait" (S1) et le fichier de sortie (S6).
Elle transmet des données aux sous-chaînes calculant les sorties : S2, S3, S4.

. Phase 2 : établit l'état statistique mensuel (S2) à partir des données transmises par la phase P1.

. Phase 3 : établit l'état de décompte monétaire (S3) à partir des données transmises par P1.

. Phase 4 : établit l'état de virement bancaire (S4) à partir des données transmises par P1.

b) chaîne annuelle :

. Phase 5 : établit l'état statistique annuel à partir du fichier de sortie S6 obtenu dans la chaîne mensuelle.

4) - Description des opérations des différentes phases de traitement :

a) les grilles d'évaluation :

PHASE DE TRAITEMENT P1 : "BORDEREAU DE PAIE DU LAIT" :

NØTIONS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
			TY	IDENTIFICATEURS	INDICATIFS
NØM			P	NØM	NØAGR
PRENØM			P	PRENØM	NØAGR
ADRESSE			P	ADRESSE	NØAGR
TØUR			P	TØUR	NØAGR
MAT-GRAS			M	MAT-GRAS	NØAGR
MAT-AZ			M	MAT-AZ	NØAGR
QU-BAC			M	QU-BAC	NØAGR
RELEVE			M	RELEVE	NØAGR
TM-GRAS	module calcul				
TM-AZ	taux moyen				
TM-BAC					
PR-BASE			C	PR-BASE	
PR-GRAS	(TM-GRAS - F-GRAS) x V-GRAS		C	F-GRAS	
			C	V-GRAS	
PR-AZ	(TM-AZ - F-AZ) x V-AZ		C	F-AZ	
			C	V-AZ	
PR-BAC	module TAB-BAC		C	PRI-BAC	
			C	REF-BAC	
PR-CTR	V-CTR	si CTR = 1	C	V-CTR	
			P	CTR	NØAGR
PR-TEMP	V-TEMP	si TEMP = 1	C	V-TEMP	
			P	TEMP	NØAGR
PR-PAT	V-PAT	si PAT = 1	C	V-PAT	
			P	PAT	NØAGR
PR-MØN	V-MØN	si MON = 1	C	V-MØN	
			P	MØN	NØAGR
PR-LAIT	PR-BASE + PR-GRAS + PR-AZ + PR-BAC + PR-CTR + PR-TEMP + PR-PAT + PR-MØN				
TØTLI	module total litrage				
VALHT	TØTLI x PR-LAIT				
TVA	VALHT x TAUX	si C-TAUX = 1	P	C-TAUX	NØAGR
	0	sinon	C	TAUX	

NØTIONS RESULTATS	ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
			TY	IDENTIFICATEURS	INDICATIFS
NET-LAIT	VALHT + TVA				
QTE			M	QTE	NØAGR, NØART
DESIGN			P	DESIGN	NØART
PRIX			P	PRIX	NØART
MTHT	PRIX x QTE				
TVA 7			P	CØ-TVA	NØART
TVA 19			C	TAUX 7	
MTTC	module TAB-FA		C	TAUX 19	
STVA 7					
STVA 19					
NAP	NAP + MTTC				
LIBDIV			M	LIBDIV	NØAGR, CØDIV
PDIV			M	PDIV	NØAGR, CØDIV
MDIV			M	MDIV	NØAGR, CØDIV
MØDREG			P	MØDREG	
ACØMPTES	MDIV	si CØDIV = 5			
SPDIV	SPDIV + PDIV				
SMDIV	SMDIV + MDIV				
APAYER	SØLDE	si SØLDE > 0			
REMB	SØLDE	si SØLDE < 0			
SØLDE	NET-LAIT - NAP - ACØMPTES + SPDIV - SMDIV		I	SØLDE	NØAGR
BANQUE			P	BANQUE	NØAGR
SEG-T1-R4	existence	si MØDREG = "BA"	P	MØDREG	NØAGR
SEG-T1-R3	existence	si MØDREG = "ES"	P		
SEG-T1-R2	existence	si NØAGR de SEG-P1-ACRIC =NØAGR de SEG-M1-ACHATS			

PHASE DE TRAITEMENT P3 : ETAT STATISTIQUE MENSUEL

NOTIONS RESULTAT	OPERATIONS	CONDITIONS	NØTIØNS D'ENTREE		
			TY	IDENTIFI-CATEURS	INDICATIFS
NØM			T	NØM	TØUR, NØAGR
PRENØM			T	PRENØM	TØUR, NØAGR
ADRESSE			T	ADRESSE	TØUR, NØAGR
TØTLI			T	TØTLI	TØUR, NØAGR
VALHT			T	VALHT	TØUR, NØAGR
TVA			T	TVA	TØUR, NØAGR
NET-LAIT			T	NET-LAIT	TØUR, NØAGR
TØUR			T	TØUR	TØUR, NØAGR
CUM-LI	CUM-LI + TØTLI				
CUM-VALHT	CUM-VALHT + VALHT				
CUM-TVA	CUM-TVA + TVA				
CUM-NET	CUM-NET + NET-LAIT				

PHASE DE TRAITEMENT P2 : DECOMPTE MONETAIRE

NOTIONS RESULTAT	OPERATIONS	CONDITIONS	NØTIØNS D'ENTREE		
			TY	IDENTIFI-CATEURS	INDICATIFS
NØM			T	NØM	TØUR, NØAGR
PRENOM			T	PRENOM	TØUR, NØAGR
ADRESSE			T	ADRESSE	TØUR, NØAGR
TØUR			T	TØUR	TØUR, NØAGR
APAYER			T	APAYER	TØUR, NØAGR
T-ESP	module calcul 1		I	RESTE	TØUR, NØAGR
CUM-T-ESP	module calcul 2		I	S	
CUM-G-ESP	module calcul 3				

PHASE DE TRAITEMENT P4 : VIREMENT BANCAIRE

NOTIONS RESULTAT	OPERATIONS	CONDITIONS	NOTIONS D'ENTREE		
			TY	IDENTIFI-CATEURS	INDICATIFS
NØM			T	NØM	CØGUI, NØAGR
PRENØM			T	PRENØM	CØGUI, NØAGR
ADRESSE			T	ADRESSE	CØGUI, NØAGR
APAYER			T	APAYER	CØGUI, NØAGR
DOM-BANC			T	DØM-BANC	CØGUI, NØAGR
CUM-GUI	CUM-GUI + APAYER				
CUM-GEN	CUM-GEN + CUM-GUI				

PHASE DE TRAITEMENT P5 : ETAT STATISTIQUE ANNUEL

NOTIONS RESULTAT	OPERATIONS	CONDITIONS	NOTIONS D'ENTREE		
			TY	IDENTIFICATEURS	INDICATIFS
NØM			P	NØM	NØAGR
PRENØM			P	PRENØM	NØAGR
ADRESSE			P	ADRESSE	NØAGR
PR-BASE			T	PR-BASE	NØAGR, MOIS
PR-GRAS			T	PR-GRAS	NØAGR, MOIS
PR-AZ			T	PR-AZ	NØAGR, MOIS
PR-BAC			T	PR-BAC	NØAGR, MOIS
PR-CTR			T	PR-CTR	NØAGR, MOIS
PR-TEMP			T	PR-TEMP	NØAGR, MOIS
PR-PAT			T	PR-PAT	NØAGR, MOIS
PR-MON			T	PR-MØN	NØAGR, MOIS
PR-LAIT			T	PR-LAIT	NØAGR, MOIS
TØTLI			T	TØTLI	NØAGR, MOIS
VALHT			T	VALHT	NØAGR, MOIS
TVA			T	TVA	NØAGR, MOIS
NET-LAIT			T	NET-LAIT	NØAGR, MOIS
PR-MØY	C-VALHT/12				
C-TOTLI	C-TOTLI + TOTLI				
C-VALHT	C-VALHT + VALHT				
C-TVA	C-TVA + TVA				
C-NET-LAIT	C-NET-LAIT + NET-LAIT				

b) Description des modules utilisés par les phases de traitement :

Phase de traitement P1 :

module CALCUL TAUX MOYEN ;

TM-AZ = TM-BAC = TM-GRAS = 0 ;

Pour I de 1 à 3 faire TM-GRAS = TM-GRAS + MAT-GRAS (I) ;

TM-AZ = TM-AZ + MAT-AZ (I) ;

TM-BAC = TM-BAC + QU-BAC (I) fp ;

TM-GRAS = TM-GRAS/3

TM-AZ = TM-AZ/3

fin module ;

module TAB-BAC ;

Sélection

Actions	TQBA ≥ 8	TQBA ≤ 4	sinon
PQBA = PRI-BAC	X		
PQBA = REF-BAC		X	
PQBA = 0			X

ft

fin module ;

module TAB-FA ;

Si CØ-TVA = 1 alors TVA7 = MTHT × TAUX7 ;

MTTC = MTHT + TVA7 ;

TVA19 = 0 ;

STVA7 = STVA7 + TVA7 ;

sinon TVA19 = MTHT × TAUX19 ;

TVA7 = 0 ;

MTTC = MTHT + TVA19 ;

STVA19 = STVA19 + TVA 19 fsi ;

fin module ;

```
module TOTAL LITRAGE ;
Pour I de 1 à 31 faire TOTLI = TOTLI + RELEVE (I) fp
fin module ;

Phase de traitement P3 :
module calcul 1 ;
RESTE réel ; S réel ;
RESTE = APAYER ;
        Pour I de 1 à 9 faire T-ESP(I) = RESTE/BILLET(I) ;
        S = T-ESP(I) × BILLET(I) ;
        RESTE = RESTE - S  fp
fin module ;
module CALCUL 2 ;
        Pour I de 1 à 9 faire CUM-T-ESP(I) = CUM-T-ESP(I) + T-ESP(I)  fp ;
fin module ;
module CALCUL 3 ;
        Pour I de 1 à 9 faire CUM-G-ESP(I) = CUM-G-ESP(I) + CUM-T-ESP(I)
        fp ;
fin module ;
```

5) Description des segments logiques d'entrée dans la classe répertoire :

· Les segments logiques mouvement :

```
type SEC-M1 RAMASSAGE struct (NØAGR,
                              RELEVE file (31) décimal 9 (4)) ;
type SEC-M1-ANALYSE   struct (NØAGR,
          ANALYSE     struct (MAT-GRAS file (3) décimal 99V9,
                              MAT-AZ file (3) décimal 99V9,
                              QU-BAC file (3) décimal 9)) ;
```



```
type SEC-M1-DIVERS struct (NØAGR,
          DIVERS struct (CODIV décimal 99,
                         LIBDIV file (20) car,
                         PDIV décimal 9 (4) V99,
                         MDIV décimal 9 (4) V99)) ;

type SEC-M1-ACHATS struct (NØAGR,
          LICØ   struct (NOART décimal 9 (4),
                         QTE décimal 9 (3))) ;
```

· Les segments logiques permanents :

```
type SEC-P1-AGRIC struct (SITUATION struct(
     NØAGR décimal 9 (4)
     TØUR décimal 9 (2),
     NØM file (20) car,
     PRENØM file (15) car,
     ADRESSE struct (RUE file (30) car,
                     VILLE file (15) car,
                     CØDE struct (DEP décimal 9 (2),
                                  PØS décimal 9 (3)))),
     CONTROLE struct (CTR file (1) car,
                      TEMP file (1) car,
                      PAT file (1) car,
                      MØN file (1) car),
     C-TAUX file (1) car,
     BANQUE struct (COGUI décimal 99,
                    DØM-BANC file (50) car),
     MODREG file (2) car) ;

type SEC-P1-PRODUIT struct (
          NØART décimal 9 (4),
          DESIGN file (15) car,
          PRIX décimal 9 (4) V99,
          CØ-TVA décimal 9 ) ;
```

· Segments transmis entre deux phases de traitement :

1) Segment transmis entre P1 et P2.

type SEG-TIR2 struct (SITUATIØN, S-LAIT) ;

2) Segment transmis entre P1 et P3.

type SEG-TIR3 struct (SITUATIØN, APAYER) ;

3) Segment transmis entre P1 et P4.

type SEG-TIR4 struct (SITUATION, APAYER, BANQUE) ;

· Segments intermédiaires :

```
type SEG-I struct (SØLDE décimal 9 (5) V99,
                   RESTE décimal 9 (5) V99,
                     S   décimal 9 (5) V99) ;
```

Etant donné le nombre peu élevé de notions intermédiaires, on les a toutes regroupées dans un seul segment.

· Segments regroupant les notions de type constante :

```
type SEG-CI struct PARAM struct (PR-BASE décimal 99V99,
                                 F-GRAS  décimal 99V9,
                                 V-GRAS  décimal 99V999,
                                 F-AZ    décimal 99V9,
                                 V-AZ    décimal 9V999,
                                 PRI-BAC décimal 9V999,
                                 REF-BAC décimal 9V999,
                                 V-CTR   décimal 9V999,
                                 V-TEMP  décimal 9V999,
                                 V-PAT   décimal 9V999,
                                 V-MØN   décimal 9V999),
                       AUTRES struct (TAUX   décimal 99V99,
                                 TAUX7   décimal 99V99,
                                 TAUX19 décimal 99V99,
                                 LI-MOIS file (10) car)) ;
```


LEXIQUE DES NØTIØNS

SYMBØLES	SIGNIFICATIØN
ACOMPTES	montant des acomptes touchés par agriculteur durant le mois.
ANALYSE	notion structurée = MAT-GRA, MAT-AZ, QU-BAC.
APAYER	somme due par la laiterie à l'agriculteur.
C-NET-LAIT	valeur du lait livré durant une année par l'agriculteur.
CØDE	code postal figurant dans l'adresse de l'agriculteur.
CØDIV	code permettant d'identifier une opération financière.
CØGUI	code de l'agence bancaire de l'agriculteur.
CØNTRØLE	notion structurée : CTR, TEMP, PAT, MØN.
CØ-TVA	code TVA des produits (1 ou 2).
C-TAUX	code option TVA sur lait (si C-TAUX = 1, lait vendu TTC).
C-TØTLI	total litrage du lait livré durant une année par agriculteur.
CTR	code prime de contrôle laitier (CTR = 1 si agriculteur y a droit).
C-TVA	total TVA sur lait livré durant l'année par agriculteur.
CUM-G-ESP	cumul général pour toutes les tournées de CUM-T-ESP.
CUM-GUI	cumul par agence des virements bancaires du mois.
CUM-GEN	cumul général des virements bancaires du mois.
CUM-LI	cumul litrage du lait livré par tous les agriculteurs d'une tournée durant un mois.
CUM-NET	cumul NET-LAIT pour une tournée.
CUM-TVA	cumul TVA pour une tournée.
CUM-T-ESP	cumul T-ESP pour une tournée.
DESIGN	désignation des produits.

LEXIQUE DES NØTIØNS (suite)

SYMBØLES	SIGNIFICATIØN
DIVERS	notion structurée : CØDIV, LIBDIV, MDIV, PDIV.
DØM-BANC	domiciliation bancaire de l'agriculteur.
F-AZ	valeur de base du taux de matière azotée.
F-GRAS	valeur de base du taux de matière grasse.
LIBDIV	libellé de l'opération financière.
LICØ	notion structurée : NØART, QTE.
LIGNE	notion structurée : QTE, DESIGN, PRIX, MTHT, NØART.
LI-MØIS	libellé du mois.
MAT-AZ	3 taux de matière azotée obtenus par analyse.
MAT-GRAS	3 taux de matière grasse obtenus par analyse.
MDIV	montant à déduire au titre d'une opération financière.
MØDREG	mode de réglement ("BA" = virement bancaire "ES" = en espèces).
MØN	code prime de montagne (MØN = 1 si agriculteur y a droit).
MTTC	montant toutes taxes comprises d'une ligne de commande.
NAP	total facture des produits rétrocédés.
NFP	total facture des produits rétrocédés.
NET-LAIT	valeur taxes comprises du lait livré par agriculteur durant le mois.
NØAGR	numéro de l'agriculteur.
NØART	numéro de produit.
NØM	nom de l'agriculteur.
PAT	code prime étable patentée (PAT = 1 si agriculteur y a droit).
PDIV	montant à ajouter au titre d'une opération financière.
PR	notion structurée : toutes les primes par litre de lait.
PR-AZ	prime de matière azotée par litre.

LEXIQUE DES NØTIØNS (suite)

SYMBØLES	SIGNIFICATIØN
PR-BAC	prime de qualité bactériologique par litre.
PR-CTR	prime de contrôle laitier par litre.
PR-GRAS	prime de matière grasse par litre.
PRI-BAC	valeur fixée pour la prime de qualité bactériologique par litre.
PR-BASE	prix de base fixé pour le litre de lait.
PRENØM	prénom de l'agriculteur.
PRIX	prix unitaire d'un produit.
PR-LAIT	prix du litre de lait, primes comprises.
PR-MØN	prime de montagne par litre.
PR-MØY	prix moyen au litre touché par agriculteur pour toute une année.
PR-PAT	prime étable patentée par litre.
PR-TEMP	prime basse température par litre.
QTE	quantité commandée d'un produit.
QU-BAC	3 notes de qualité bactériologique obtenues par analyse.
REF-BAC	valeur fixée pour la réfaction au titre de la qualité bactériologique.
RELEVE	31 relevés journaliers de lait livré par agriculteur.
REMB	somme due par agriculteur à la laiterie.
RESTE	variable de travail.
RUE	n° et nom de la rue figurant dans adresse de l'agriculteur.
S	variable de travail.
SITUATION	notion structurée : NØAGR, TØUR, NØM, PRENØM, ADRESSE.
S-LAIT	notion structurée : TØTLI, VALHT, TVA, NET-LAIT.
S-LIGNE	notion structurée : TVA7, TVA19, MTTC.

LEXIQUE DES NØTIØNS (suite)

SYMBØLES	SIGNIFICATION
SMDIV	somme des MDIV pour un agriculteur.
SPDIV	somme des PDIV pour un agriculteur.
SØLDE	variable de travail.
STVA7	somme de TVA7 pour un agriculteur.
STVA19	somme de TVA19 pour un agriculteur.
TAUX	taux de TVA sur le lait (= 7,5 %).
TAUX7	taux de TVA sur produit (= 7 %).
TAUX19	taux de TVA sur produit (= 19 %).
TEMP	code prime de basse température (= 1, si agriculteur y a droit).
T-ESP	tableau de 9 éléments. Donne pour chaque agriculteur la décomposition de APAYER en billets de 500 ; 100 ; 50 ; 10 ; et en pièces de 5 ; 1 ; 0,50 ; 0,20 ; 0,10.
TM	notion structurée : TM-GRAS, TM-AZ, TM-BAC.
TM-AZ	taux moyen de matière azotée.
TM-GRAS	taux moyen de matière grasse.
TM-BAC	note de qualité bactériologique.
TØTLI	total litrage du mois pour un agriculteur.
TØUR	n° de tournée.
TVA	montant TVA sur lait vendu par un agriculteur.
TVA7	montant TVA à 7 % sur une ligne de commande.
TVA19	montant TVA à 19 % sur une ligne de commande.
VALHT	valeur hors taxe du lait livré pour un mois par un agriculteur.
V-AZ	valeur du gramme de matière azotée.

LEXIQUE DES NØTIØNS (suite)

SYMBØLES	SIGNIFICATIØN
V-GRAS	valeur du gramme de matière grasse.
VFL	voir NET-LAIT.
V-CTR	valeur de la prime de montagne.
VILLE	nom de ville figurant dans ADRESSE.
V-MØN	valeur de la prime de montagne.
V-PAT	valeur de la prime étable patentée.
V-TEMP	valeur de la prime basse température.

IV - ANALYSE ØRGANIQUE

1) - Constitution et déclaration des fichiers :

a) Les fichiers permanents :

Classe FICH 1 ;

FIP-AGRIC ensemble SEG-P1-AGRIC ;

fin classe ;

Classe FICH 2 ;

FIP-PRØD ensemble SEG-P1-PRØDUIT ;

fin classe ;

b) Les fichiers mouvements :

Classe FICH 3 ;

FIM-RELEVE ensemble SEG-M1-RAMASSAGE ;

FIM-ANALYSE ensemble SEG-M1-ANALYSE ;

FIM-DIVERS ensemble SEG-M1-DIVERS ;

fin classe ;

Classe FICH 4 ;

FIM-ACHATS ensemble SEG-M1-ACHATS ;

fin classe ;

c) Les fichiers de sortie (chaîne de traitement mensuelle) :

Classe FICH 5 ;

FIS-PAIE ensemble ART-PAIE ;

fin classe ;

Classe FICH 6 ;

FIS-DECØMPTE ensemble ART-DEC ;

fin classe ;

Classe FICH 7 ;

FIS-VIR ensemble ART-VIR ;

fin classe ;

Classe FICH 8 ;

FIS-STAT ensemble ART-STAT ;

fin classe ;

Classe FICH 9 ;

FIS-RECAP ensemble ART-RECAP ;

fin classe ;

2) - Vérification des règles de compatibilité dans les différentes phases de traitement :

La phase de traitement P1 ne les vérifie pas, car le fichier permanent des produits FIP-PRØD n'a pas le même indicatif majeur que les autres fichiers d'entrée et le fichier de sortie.

On va donc créer une phase organique P10 qui créera un fichier transmis à la phase P11 (ancienne phase P1) vérifiant les règles de compatibilité.

Phase organique P10 : Valorisation des bons de commande.

NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
			TY	IDENTITE	INDICATIFS
QTE			M	QTE	NØART, NØAGR
DESIGN			P	DESIGN	NØART
PRIX			P	PRIX	NØART
CØ-TVA			P	CØ-TVA	NØART
MTHT	PRIX × QTE				

Modifications apportées à l'ancienne phase de traitement P1 codifiée P11.

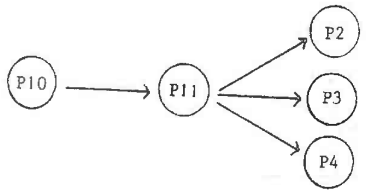
NØTIØNS RESULTAT	ØPERATIØNS	CØNDITIØNS	NØTIØNS D'ENTREE		
			TY	IDENTITE	INDICATIFS
QTE			T	QTE	NØAGR, NØART
DESIGN			T	DESIGN	NØAGR, NØART
PRIX			T	PRIX	NØAGR, NØART
MTHT			T	MTHT	NØAGR, NØART

Le découpage de la phase de traitement P1 en deux phases organiques P10 etP11 implique la création d'un nouveau segment transmis dans la classe répertoire :

SEG-T1OR11 struct (QTE, DESIGN, PRIX, CØ-TVA, MTHT) ;

3) - Regroupement des phases organiques en unités de traitement (chaîne de traitement mensuelle) :

a) Graphe d'enchaînement des phases organiques :



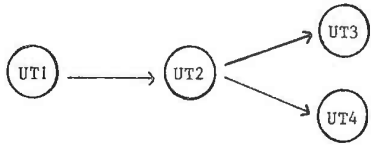
b) Règles de regroupement :

Seules les phases organiques P2 et P3 vérifient les règles de regroupement en unités de traitement.

On aura donc les unités de traitement suivantes :

UT1 : P10
UT2 : P11
UT3 : P2 et P3
UT4 : P4

c) Graphe représentant les contraintes d'antériorité entre unités de traitement :



d) Déclaration des fichiers transmis :

On aura un fichier transmis : FIT-REGL qui sera obtenu par réunion des segments transmis de P11 à P2 et de P11 à P3.

Les autres fichiers transmis sont la représentation des segments créés au cours de l'analyse fonctionnelle.

Classe FICH 10 ;

FIT-REGL ensemble struct (SITUATION, S-LAIT, APAYER) ;

fin classe ;

Classe FICH 11 ;

FIT-VAL ensemble SEG-T1OR11 ;

fin classe ;

Classe FICH 12 ;

FIT-VIR ensemble SEG-T1R4 ;

fin classe ;

4) - Mise en oeuvre de la chaîne mensuelle et des unités de traitement :

Module de commande de la chaîne ;

Module MC1 ;

Application Paie du lait ;

UT1 ; UT2 ; UT3 ; UT4 ; fin module ;

Description complète en Civa de l'unité de traitement UT2 :

```
module UT2 ; co module directeur oc ;
        utilise FICH1, FICH3, FICH5, FICH9, FICH10, FICH11, FICH12 ;
        P de FIP-AGRIC ; S de FIS-PAIE ; T11 de FIT-VAL ;
        Utilisation (FIP-AGRIC, FIT-VAL, FIM-RELEVE, FIM-ANALYSE,
                     FIM-DIVERS) = "lecture" ;
        Utilisation (FIS-PAIE, FIT-REGL, FIT-VIR, FIS-RECAP) = "écriture" ;
        T3 de FIT-REGL ; T4 de FIT-VIR ; S1 de FIS-RECAP ;
        M1 de FIM-RELEVE ; M2 de FIM-ANALYSE ; M3 de FIM-DIVERS ;
        I entier ;
        co la déclaration des paramètres de l'unité de traitement est
        faite dans la classe d'application. On leur attribue des valeurs
        si cela est nécessaire dans le module de commande oc
        NØAGR croi indicatif de FIP-AGRIC, FIM-RELEVE, FIM-ANALYSE ;
        NØAGR croi, NØART croi indicatif de FIT-VAL ;
        NØAGR croi, CØDIV croi indicatif de FIM-DIVERS ;

        Pour chaque NØAGR faire TI-AGRIC ;
                    Pour chaque NØART faire TR-ART fpc ;
                    Pour chaque CØDIV faire TR-DIV fpc ;
                        TF-AGRIC fpc ;

fin module ;

module TI-AGRIC ; co création du groupe logique de sortie 1 oc ;
        SITUATION de S = SITUATION de P ;
        ANALYSE   de S = ANALYSE   de M1 ;
        RELEVE    de S = RELEVE    de M2 ;
        CALCUL TAUX MOYEN ; co module analyse fonctionnelle oc ;
```



```
        PR-BASE de S = PR-BASE de PARAM ;
        PR-GRAS = (TM-GRAS - F-GRAS)V-GRAS ;
        PR-AZ = (TM-AZ - F-AZ)V-AZ ;
        TAB-BAC ; co module analyse fonctionnelle oc ;
        Si CTR  = 1 alors PR-CTR  de S = V-CTR  sinon Pr-CTR  de S = 0 fsi ;
        Si TEMP = 1 alors PR-TEMP de S = V-TEMP sinon PR-TEMP de S = 0 fsi ;
        Si PAT  = 1 alors PR-PAT  de S = V-PAT  sinon PR-PAT  de S = 0 fsi ;
        Si MØN  = 1 alors PR-MØN  de S = V-MØN  sinon PR-MØN  de S = 0 fsi ;

        PR-LAIT de S = PR-BASE de S + PR-AZ de S + PR-BAC de S
                     + PR-CTR de S + PR-TEMP de S + PR-PAT de S
                     + PR-MØN de S ;

        TØTLI de S = 0 ;
        TØTAL LITRAGE ; co module analyse fonctionnelle oc ;
        VALHT de S = TØTLI de S × PR-LAIT de S ;
        Si C-TAUX = 1 alors TVA de S = VALHT de S × TAUX ;
                            NET-LAIT de S = VALHT de S + TVA de S ;
                      sinon NET-LAIT de S = VALHT de S fsi ;
        NAP = 0 ; SPDIV = 0 ; SMDIV = 0 ; STVA7 = 0 ; STVA19 = 0 ;
fin module ;

module TR-ART ; co création du groupe logique de sortie 2 oc ;
        LIGNE de S = LIGNE de T11 ;
        TAB-FA ; co module analyse fonctionnelle oc ;
        NAP = NAP + MTTC ;
fin module ;
```

```
module TR-DIV ; co création du groupe logique de sortie 4 oc ;
        Si CØDIV = 5  alors  ACØMPTES = MDIV
                      sinon  DIVERS de S = DIVERS de M3 ;
                             SPDIV = SPDIV + PDIV ;
                             SMDIV = SMDIV + MDIV fsi ;
fin module ;

module TF-AGRIC ; co création du groupe logique de sortie 5,
                    création des fichiers transmis et du fichier
                    de sortie réutilisé par chaîne annuelle oc ;
        SØLDE réel ;
        SØLDE = NET-LAIT de S - NAP - ACØMPTES - SMDIV + SPDIV ;
        Si SØLDE ≥ 0  alors  APAYER de S = SØLDE
                      sinon  REMB = SØLDE fsi ;
        S en S + 1 ;
        CREATION-FUT3 ; co module de création des articles du fichier
                         transmis à UT3 oc ;
        Si SØLDE > 0 et si MØDREG de P = 'BA' alors faire CREATION-FUT4 ;
                      co module de création des articles du fichier
                      transmis à UT4 oc ;
        CREATION-SØRT ; co création des articles du fichier réutilisé par
                      chaîne annuelle oc ;
fin module ;

module CREATION-FUT3 ;
        SITUATION de T3 = SITUATIØN de P ;
        MØDREG    de T3 = MØDREG    de P ;
```



```
        APAYER    de T3 = APAYER    de S
        S-LAIT    de T3 = S-LAIT    de S ;
        T3 en T3 + 1
fin module ;

module CREATION-FUT4 ;
        SITUATION de T4 = SITUATION de P ;
        APAYER    de T4 = APAYER    de S ;
        BANQUE    de T4 = BANQUE    de P ;
        T4 en T4 + 1
fin module ;

module CREATIØN-SØRT ;
        SITUATION de S1 = SITUATION de S ;
        PR de S1 = PR de S ;
        S-LAIT de S1 = S-LAIT de S ;
        S1 en S1 + 1 ;
fin module ;
```

BIBLIOGRAPHIE

Publications CIVA

(1) DERNIAME J.C. — Le projet CIVA, un système de programmation modulaire. Doctorat d'Etat. Université NANCY 1. Janvier 1974.

(2) AUBRY B. — Traduction des tables de décision. Thèse de 3ème cycle. Université NANCY 1. Mars 1973.

(3) BENAMGHAR L. — Instruction d'affectation et définition d'un métalangage dans le projet CIVA. Doctorat d'ingénieur. Université NANCY 1. Juin 1973.

(4) BARTHELEMY C., PHILIPPE — Problèmes de documentation dans le projet CIVA. Thèse de 3ème cycle. A paraître.

(5) CHABRIER J.J. — Acquisition et édition des fichiers, analyse des données dans le projet CIVA. Thèse de 3ème cycle. Université NANCY 1. Décembre 1973.

(6) DENDIEN J. — Gestion statique de mémoire dans un système de programmation modulaire. Doctorat d'ingénieur. Université NANCY 1. Mars 1973.

(7) DUCLOY J. — Compilation dans le projet CIVA. Doctorat d'ingénieur. Université NANCY 1. Mars 1973.

(8) FIEGEL C. — Traduction des instructions d'itération dans le projet CIVA. Thèse de 3ème cycle. A paraître.

(9) FIEGEL C. — Réalisation du générateur d'instructions de cinématique des fichiers en Cobol. Document interne CIVA. Mai 1974.

(10) LION F. — Gestion des objets de taille variable dans le projet CIVA. Thèse de 3ème cycle. A paraître.

(11) MANSUY D. — Implantation des identificateurs et codification dans CIVA. Thèse de 3ème cycle. A paraître.

(12) VIAULT D. — Définition et interprétation des modules de commande dans le projet CIVA. Thèse de 3ème cycle. A paraître.

Structure d'information

(13) CODD E.F. — A relationnal Model of Data for large Shared Data Bank. ACM 1970.

(14) DELOBEL C. — Aspects théoriques sur la structure de l'information dans une banque de données. RIRO 1971.

(15) PAIR C. — Les structures d'information et leur représentation en mémoire. Cours polycopié. Université NANCY 1. 1972.

Les systèmes

(16) MELESE J. — La gestion par les systèmes. Hommes et techniques Ed., 1968.

(17) MELESE J. — L'analyse modulaire des systèmes. Hommes et techniques Ed., 1972.

(18) BLUMENTHAL S.C. — Les systèmes informatiques de gestion. Entreprise moderne d'édition, 1971.

Analyse et programmation

(19) BAUVIN G. — L'informatique de gestion. Hommes et techniques Ed., 1968.

(20) COUGER J.D. — Evolution of Business System analysis techniques. ACM Computins surveys. Septembre 1973.

(21) DASSE — Analyse informatique. MASSON et Cie, 1972.

(22) DU BEAUDIEZ et LAPAUW — Principes généraux d'une analyse des problèmes de gestion en vue d'une génération automatique des programmes. RIRO. Janvier 1973.

(23) DUFOURD J.F., HERTSCHUH N. — Programmation modulaire appliquée aux problèmes de gestion. Document interne. I.U.T. Informatique (1972-1973).

(24) MALLET R.A. — La méthode informatique. Ed. HERMANN, 1971.

(25) PAIR C. — Séminaires sur une méthode de programmation déductive. Université NANCY 1. Novembre 1973.

(26) REIX R. — L'analyse en informatique de gestion (2 tomes). DUNOD, 1971.

(27) THOMAS A. — Techniques d'analyse en informatique de gestion. DUNOD, 1970.

(28) WARNIER J.D. — Les procédures de traitement et leurs données. Ed. d'Organisation, 1973.

Méthodes d'analyse et de programmation

(29) ARIANE — (GAMMA).

(30) ARMIN PARM — (PROMO INFORMATIQUE).

(31) CANTOR 2 — (ICL).

(32) CORIC — (CGI).

(33) MINOS — (CEGOS).

(34) PROTEE — (SIS).

NOM DE L'ETUDIANT : HERTSCHUH Norbert

NATURE DE LA THESE : Doctorat de Spécialité en Mathématiques Appliquées (Informatique)

VU, APPROUVE

& PERMIS D'IMPRIMER

NANCY, le 12 Juin 1974

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE DE NANCY I
Le Président

J.R. HELLUY