Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Implémentation d'une application de gestion sur un réseau local

Vanoirbeek, Christine

Award date:
1981

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2026

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE



RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

FACULTES
UNIVERSITAIRES
N.-D. DE LA PAIX
NAMUR

Bibliothèque
FMB16
14/1

FM B16/1981/14/1

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

IMPLEMENTATION D'UNE APPLICATION

DE GESTION SUR UN RESEAU LOCAL

Mémoire présenté par

Christine Vanoirbeek

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

Année académique 1980-1981

LBS 3442282
77121

Je remercie les personnes qui, à titres divers, contribuèrent à la réalisation de ce mémoire.

Je tiens à exprimer ma reconnaissance à Monsieur Ph. VAN BASTELAER, promoteur de ce mémoire, pour les conseils apportés lors de la rédaction.

Je tiens également à témoigner ma gratitude à Monsieur J.D.NICOUD de m'avoir accueillie pour un stage à l'Ecole Polytechnique Fédérale de Lausanne.

Je remercie vivement l'ensemble des membres du Laboratoire de Micro-informatique pour leur acceuil au sein de leur département, et en particulier Monsieur C.PETITPIERRE dont les conseils judicieux m'ont permis de mener à bien la réalisation du projet.

Je tiens enfin à remercier Monsieur R.VERHAEGE pour sa constante disponibilité, ainsi que Mademoiselle F.VANOIRBEEK pour le soin et la diligence apportés à la rédaction.

P L A N G E N E R A L

- Plan général I
- Table des matières..... II
- Introduction..... V

PARTIE I : TENDANCE A LA REPARTITION

Chapitre 1: Cadre général..... 2
Chapitre 2: Situation du projet..... 19

PARTIE II : ELEMENTS EXISTANTS
Chapitre 3: Description du réseau local COBUS..... 29
Chapitre 4: Spécifications de l'application "PETIT-PAS"... 42

PARTIE III : REALISATION DU PROJET
Chapitre 5: Architectures organique et d'implémentation... 56
Chapitre 6: Primitives de gestion des fichiers..... 67
Chapitre 7: Primitives de communication..... 113
Chapitre 8: Réalisation concrète de l'application..... 114

- Conclusion..... 123
- Bibliographie..... 124

ANNEXES
Annexe A Proposition d'architecture organique
Annexe B Modèle conceptuel
Annexe C Diagramme des flux d'information
Annexe D Programmes des fonctions systèmes
Annexe E Programme application

T A B L E D E S M A T I E R E S

PARTIE I : Tendance à la répartition

CHAPITRE 1 : Cadre général

1.1. Introduction 2
1.2. Bref historique..... 2
  1.2.1. Première étape: connexion directe des terminaux à l'ordinateur central..... 3
  1.2.2. Deuxième étape: apparition des concentrateurs..... 4
  1.2.3. Troisième étape: apparition de frontaux..... 5
  1.2.4. Quatrième étape: réseaux généraux d'ordinateur..... 6
1.3. Centralisation et décentralisation..... 7
1.4. Caractéristiques des systèmes distribués..... 9
1.5. Rôle prépondérant de la micro-informatique..... 13
1.6. Réseaux locaux..... 14
1.7. Bases de données réparties..... 15
  1.7.1. Notion de base de données répartie..... 15
  1.7.2. Principaux avantages des bases de données réparties.. 16
  1.7.3. Problèmes spécifiques aux bases de données réparties. 18

CHAPITRE 2 : Situation du projet

2.1. But poursuivi..... 19
2.2. Présentation de l'application "PETIT-PAS"..... 20
  2.2.1. Présentation et organisation de la firme..... 20
  2.2.2. Objectifs de la direction générale..... 24
  2.2.3. Critique de l'existant..... 25
  2.2.4. Proposition d'automatisation..... 25
    2.2.4.1. Modification de la politique de l'entreprise..... 25
    2.2.4.2. Nouvelle organisation de la firme..... 25

PARTIE II : Eléments existants

CHAPITRE 3 : Description du réseau local COBUS

3.1. Remarque préliminaire..... 29
3.2. Généralités..... 29
3.3. Configuration du réseau..... 31
  3.3.1. Vue globale..... 31
  3.3.2. Eléments utiles dans le cadre du projet..... 33
3.4. Aspect hardware..... 33
3.5. Protocole de transmission..... 34
3.6. Système d'exploitation du réseau..... 37
  3.6.1. Introduction..... 37

3.6.2. Désignation des divers objets..... 37
3.6.3. Notion de canal..... 37
3.6.4. Création et destruction des objets..... 38
3.6.5. Ouverture et fermeture des objets..... 38
3.6.6. Ordres primitifs disponibles..... 38
3.6.7. Principales fonctions du multiplexeur..... 39

CHAPITRE 4 : Spécification de l'application "PETIT-PAS"
4.1. Considérations d'ordre méthodologique..... 42
4.1.1. Structure des données..... 42
4.1.1.1. Nomenclature des éléments..... 42
4.1.1.2. Mode de représentation..... 43
4.1.2. Structure des traitements..... 44
4.2. Acquit de l'analyse fonctionnelle..... 45
4.2.1. Modèle conceptuel global..... 45
4.2.2. Structure des traitements..... 47
4.2.2.1. Aspect statique..... 47
4.2.2.2. Aspect dynamique..... 53
4.2.2.3. Flux d'information..... 53
4.3. Modification apportée..... 53

PARTIE III : Réalisation du projet

CHAPITRE 5: Architecture organique et d'implémentation

5.1. Architecture organique..... 56
5.2. Architecture d'implémentation..... 57
5.2.1. Remarque préliminaire..... 57
5.2.2. Affectation des traitements aux processeurs..... 58
5.2.3. Répartition des données..... 61
5.2.4. Problème spécifique des priorités..... 62
5.2.5. Caractéristiques d'une implémentation dans l'optique réseau..... 65

CHAPITRE 6: Primitives de gestion des fichiers
6.1. Introduction..... 67
6.2. Spécification générale des programmes à réaliser..... 68
6.3. Caractéristiques et modes de représentation des fichiers sur la mémoire de masse..... 69
6.3.1. Quelques définitions..... 70
6.3.1.1. Aspect logique d'un fichier..... 70
6.3.1.2. Aspect physique d'un fichier..... 71
6.3.1.2.1. Classes de support..... 71
6.3.1.2.1. Organisation d'un fichier..... 72
6.3.2. Fichiers séquentiels..... 75
6.3.3. Fichiers indexés..... 75
6.3.3.1. Types d'accès implémentés..... 76
6.3.3.2. Structuration physique proprement dite..... 76
6.3.3.2.1. Organisation du fichier .INDX..... 78
6.4. Gestion des accès concurrents aux fichiers..... 81

6.4.1. Introduction..... 81
6.4.2. Notion de blocage..... 82
6.4.3. Principales situations imposant le blocage..... 83
6.4.4. Impact des mécanismes de contrôle de la base de données sur les programmes d'application..... 86
6.4.5. Granularité du blocage..... 86
6.4.6. Types de blocage..... 87
6.4.7. Solution retenue dans le cadre du projet..... 87
6.5. Examen des programmes..... 89
6.5.1. Mode de présentation des traitements..... 89
6.5.1.1. Notions de base..... 89
6.5.1.2. Concept d'arbre programmatique..... 90
6.5.2. Primitives implémentées..... 92
6.5.2.1. Généralités..... 92
6.5.2.2. Structure générale du programme gérant la mémoire de masse..... 96
6.5.2.3. Ordres implémentés..... 96
6.5.2.3.1. Introduction..... 96
6.5.2.3.2. Ordres communs aux deux types de fichiers..... 97
6.5.2.3.3. Ordres spécifiques aux fichiers séquentiels.... 99
6.5.2.3.4. Ordres spécifiques aux fichiers indexés..... 102

CHAPITRE 7: Primitive de communication..... 113

CHAPITRE 8: Implémentation

8.1. Réalisation concrète..... 114
8.2. Description des fichiers..... 116

INTRODUCTION

Que l'on soit pour ou contre, le phénomène de répartition dans le domaine informatique est devenu une réalité que l'on ne peut ignorer. L'objet de ce projet est de mettre en évidence certaines caractéristiques propres à l'implémentation d'une application de gestion dans une telle optique. Cette réalisation se situe dans le cadre particulier des réseaux locaux.

La conception générale du mémoire repose sur la distinction entre trois grandes parties.

- La première partie a pour but d'introduire le contexte général. A cet effet, un premier chapitre sera consacré à l'examen de la situation globale en matière d'informatique répartie. L'objectif poursuivi tout au long de cette démarche est double. Il s'agit d'une part de clarifier certains concepts de base et de relever les spécificités inhérentes aux systèmes répartis. Il s'agit d'autre part, de voir dans quelle mesure ceux-ci constituent un apport de solutions à des situations pré-existantes ainsi que de dégager les nouveaux problèmes auxquels ils donnent naissance. Le second chapitre décrit le cadre particulier dans lequel s'insère l'application à réaliser.

- La seconde partie comporte une description des éléments qui constituent la base du projet à mettre en oeuvre. Il s'agit essentiellement de donner une description de l'environnement technologique qui devra supporter l'application (c'est le rôle du chapitre trois) ainsi que des spécifications à respecter (c'est le rôle du chapitre quatre).

- La troisième et dernière partie est relative à la réalisation proprement dite. Le chapitre cinq propose et justifie une architecture d'implémentation. Les trois autres chapitres donnent une description des fonctions système et des fonctions utilisateur réalisées.

PREMIERE PARTIE

TENDANCE A LA REPARTITION

CHAPITRE 1

CADRE GENERAL

1.1. INTRODUCTION

L'objet de cette première partie est de faire un rapide tour d'horizon de ce que l'on appelle les systèmes répartis ou distribués. Encore faut-il s'entendre sur les données précises de ces deux termes. Certains ne font pas de distinction et utilisent indifféremment l'un pour l'autre, d'autres par contre, apportent une nuance. Ainsi, par exemple, trouve-t-on dans [20] la distinction suivante: le concept d'"informatique répartie" concerne les systèmes où on se limite aux échanges de lots de données en différé avec le système central et le concept d'"information distribuée" fait allusion aux systèmes où l'on distribue davantage l'intelligence-machine de façon à permettre des traitements locaux. En ce qui nous concerne, nous n'accorderons pas de signification intrinsèque à ces deux termes et les utiliserons seulement pour désigner une idée de décentralisation quelle qu'elle soit, le contexte dans lequel ils seront employés ne laissant subsister aucune équivoque.

La démarche adoptée dans cette première partie a pour but, d'une part, d'essayer de relever les facteurs qui ont présidé à une telle évolution et, d'autre part, de dégager quelles sont les principales caractéristiques propres aux systèmes centralisés et décentralisés.

Un bref historique permettra de mettre en évidence cette tendance à la répartition. Suite à cela, on examinera de façon générale ce que recouvrent les notions de centralisation et de décentralisation. On passera dès lors en revue les différentes caractéristiques propres aux systèmes distribués. Une attention particulière sera accordée à la micro-informatique et aux micro-ordinateurs eu égard à leur rôle prépondérant en matière de répartition. Et enfin, on abordera le problème spécifique des bases de données réparties.

1.2. BREF HISTORIQUE

L'architecture des systèmes informatiques a fortement évolué depuis une dizaine d'années. La référence [16] discerne quatre étapes essentielles dans cette évolution:

1. Connexion directe des terminaux à l'ordinateur central
2. Apparition des concentrateurs

3. Apparition des frontaux
4. Réseaux généraux d'ordinateurs.

Dans ce qui suit, on examinera succintement chacune de ces quatre étapes en essayant de mettre en évidence les principales raisons qui ont conduit à une telle répartition des fonctions entre les divers composants d'un système.

1.2.1. Première étape: Connexion directe des terminaux à l'ordinateur central

Les circuits logiques et la mémoire qui supportent "l'intelligence" en informatique ont longtemps été des éléments coûteux par rapport, en particulier, aux lignes de transmission et aux terminaux. On a, par conséquent, cherché à les rentabiliser au maximum en les mettant en commun entre le plus grand nombre d'utilisateurs. On en arrivait donc à des systèmes hyper-centralisés où se trouvait rassemblée toute l'intelligence. La distribution des fonctions au sein d'un système informatique était simple :

- le maximum dans le système central
- le minimum près du terminal

Pour assurer un niveau de service suffisant aux utilisateurs, il convient de mettre en oeuvre des techniques (connues) dont la complexité croît rapidement avec l'ampleur du système. Il s'agit d'une part, de permettre le traitement simultané d'un grand nombre de travaux (multiprogrammation, time-slice, réentrance de programmes...) Il s'agit d'autre part, de gérer de manière satisfaisante l'ensemble des données relatives à ces travaux (bases de données, contrôle d'accès et de concurrence...) Les problèmes de sécurité et de performance sont solubles par des techniques relativement complexes (reprises automatiques, problème de la cohérence des données dans un environnement transactionnel...) dépendant du niveau de fiabilité désiré.

Comme on peut le pressentir, ce foisonnement de software engendre l'exécution d'un grand nombre d'instructions parasites inutiles au traitement lui-même. C'est ce que l'on appelle parfois " l'overhead " des systèmes d'exploitation. On constate en effet que, depuis quelques années, l'accroissement de complexité des systèmes devient alarmant. Ils supposent maintenir un environnement considérable en compétence technique et en nombre. Les systèmes d'exploitation sont parfois imprégnés d'un nombre intolérable d'erreurs de software, surtout dans les systèmes de grande taille. Les opérations de reprise de travaux, pas toujours comprises dans le software constructeur, se révèlent de plus en plus difficiles au fur et à mesure qu'on mélange en multiprogrammation sur une même machine des modes de traitement (par lot, en temps réel, en temps partagé), puis qu'on exploite en ligne des bases de données qui doivent être reconstruites après lesdits effondrements, puis

qu'on greffe la mémoire virtuelle sur le tout. Chacune de ces étapes représente une escalade de complexité. Ce phénomène de centralisation, parfois aggravé par ce que l'on pourrait à la limite qualifier d' " impérialisme des informaticiens " a contribué à atteindre un degré de sophistication fréquemment dangereux. (Cfr. schéma 1)

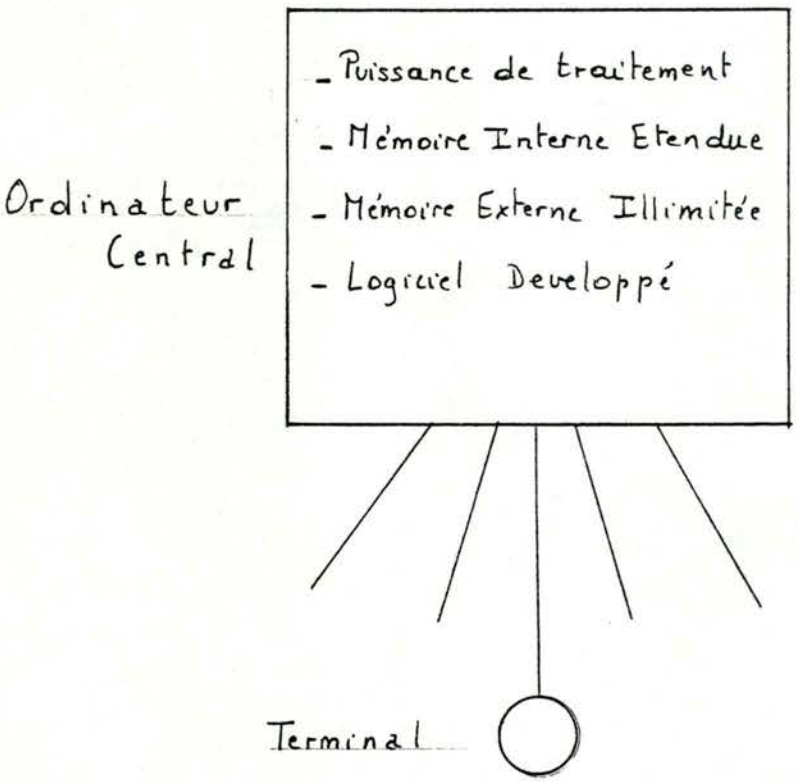


Fig. 1

1.2.2. Deuxième étape: Apparition des concentrateurs

La connexion directe d'un grand nombre de terminaux à l'ordinateur central devient problématique.
D'une part, cela s'avérait coûteux pour deux raisons principales:

- Le matériel nécessaire à la connexion d'un terminal sur l'ordinateur central représente un coût non négligeable et la gestion des transmissions pour un nombre important de terminaux monopolise la puissance de l'ordinateur central pour des tâches auxquelles il n'est pas adapté.
- Les lignes de transmission commencent à représenter un coût important dans l'ensemble de l'installation alors qu'elles ont un faible taux d' utilisation.

D'autre part, la complexité d'une telle organisation a eu pour conséquence irrémédiable une forte dégradation du temps de

réponse.

L'introduction des concentrateurs a permis d'apporter une réponse satisfaisante à ces deux points. Les concentrateurs assurent au moindre coût les fonctions de communication et de transport. Du fait de leur situation privilégiée (au contact des terminaux) et de leur nature (matériel programmable le plus souvent), ceux-ci sont capables de prendre en charge un certain nombre de fonctions simples liées aux applications. (Cfr. schéma 2).

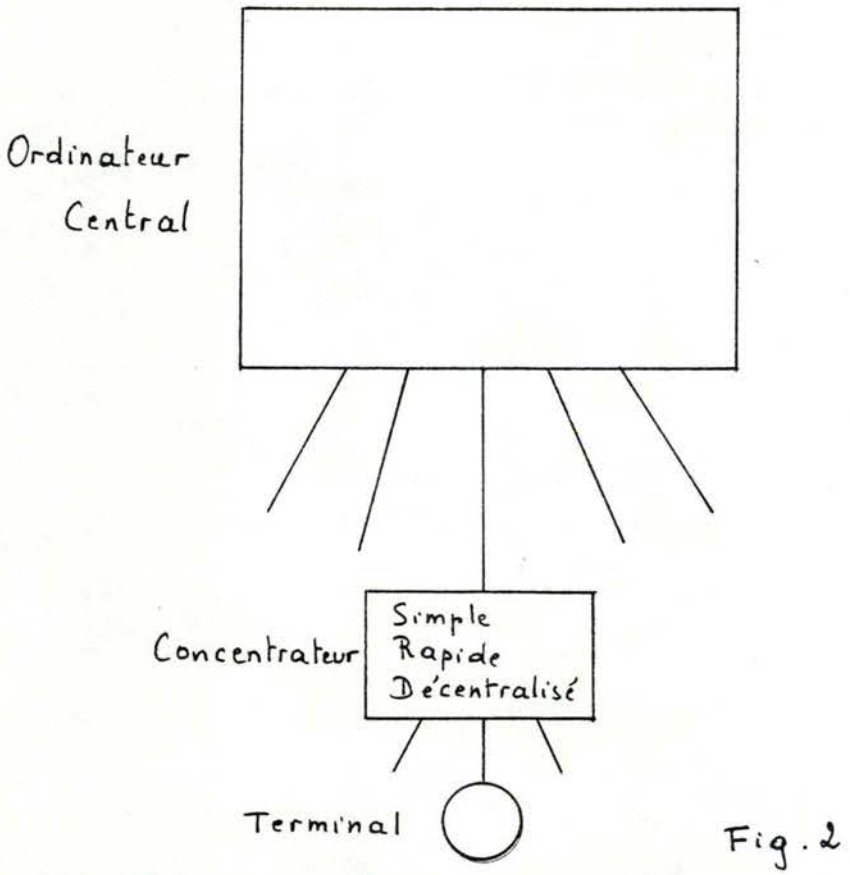


Fig. 2

1.2.3. Troisième étape: Apparition des frontaux

Dans les deux premières étapes, l'ordinateur central reste chargé de fonctions de gestion de transmission avec les terminaux (étape 1) ou avec les contentrateurs (étape 2) au moyen d'un contrôleur de communication câblé. L'apparition des mini-ordinateurs et l'expérience acquise avec les concentrateurs ont conduit à remplacer les contrôleurs de communication câblés par des contrôleurs programmés, dits frontaux. Contrairement au concentrateur, le frontal est intimement relié au central par une interface rapide et peut donc entièrement décharger le central de fonctions telles que la gestion des transmissions. Le processeur frontal réalise les fonctions de contrôle du réseau de télécommunication. Il gère automatiquement les ordres d'entrées-sorties vers les différentes stations. Les programmes fournis par le système pour gérer les échanges avec les

terminaux et rendre les programmes d'application indépendants des caractéristiques physiques des appareils sont exécutés par le frontal. Celui-ci gère également les incidents liés aux transmissions. Un processeur frontal constitue donc une extension du central dans lequel les opérations de gestion des communications sont décentralisées. Le fait que le frontal soit programmable permet de plus, de modifier les procédures de transmission sans avoir à intervenir sur le logiciel du central. (Cfr. schéma.3).

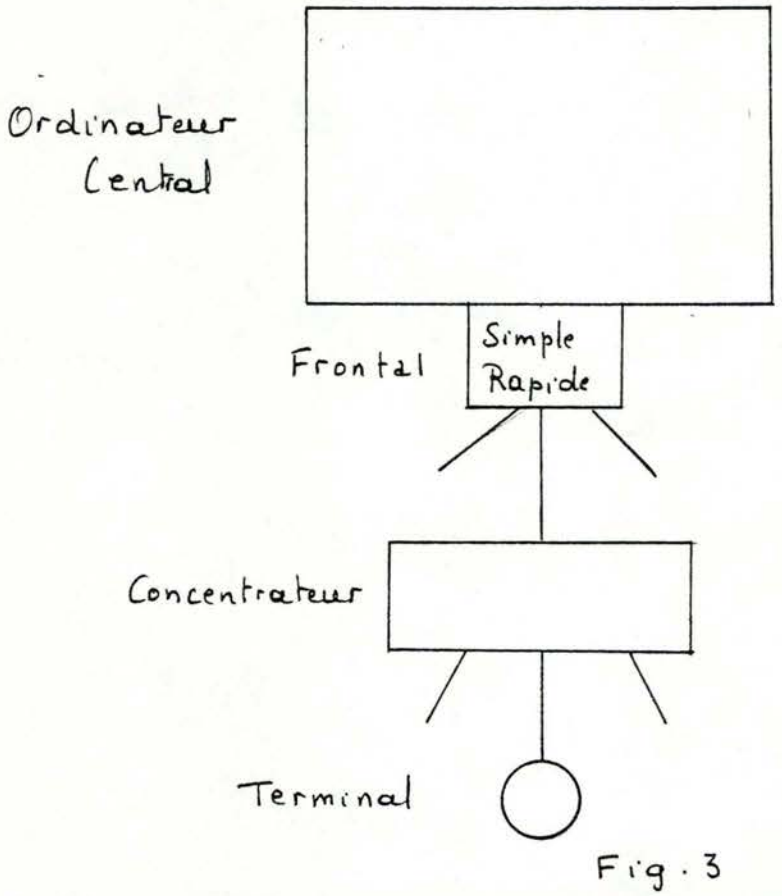


Fig. 3

1.2.4. Quatrième étape: Réseaux généraux d'ordinateurs

Dans les trois premières étapes, le système téléinformatique a une structure étoilée autour de l'ordinateur central. Dans cette dernière étape, on a affaire à des réseaux généraux d'ordinateurs dans lesquels:

- un terminal peut accéder à différents ordinateurs pour bénéficier d'une variété de services,

- les communications entre ordinateurs permettront de réaliser des applications informatiques distribuées.

Cette étape coincide également avec l'apparition des terminaux intelligents qui s'apparentent de plus en plus à des micro-ordinateurs. La tendance actuelle est de reporter les traitements à la périphérie du réseau dans les ordinateurs de

traitement et les terminaux intelligents, les intermédiaires étant uniquement chargés de transporter l'information sans la traiter. (Cfr. schéma.4).

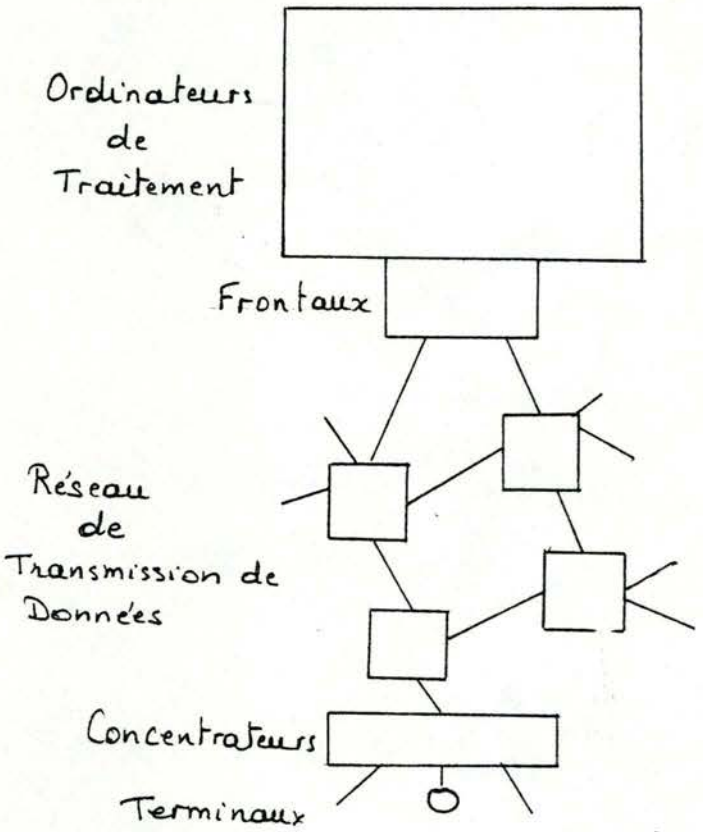


Fig. 4

Ici encore l'idée de répartition des fonctions n'est pas sans incidence sur l'architecture du réseau qui peut elle-même avoir un caractère centralisé ou décentralisé. Les deux organisations ont leurs avantages et leurs inconvénients.

- L'architecture hiérarchique centralisée permet au calculateur frontal d'optimiser le trafic total de messages qu'un réseau de lignes à débit donné peut supporter, ce qui donne l'occasion à l'utilisateur de faire des économies en coût de localisation de lignes spécialisées.

- L'architecture décentralisée maillée ne permet pas une telle optimisation car aucun contrôleur de noeud du réseau ne peut prévoir à l'avance les messages qui lui seront offerts par d'autres contrôleurs de noeuds. Par contre, ces réseaux offrent en général une ou plusieurs voies possibles pour le routage des messages et ces alternances permettent aux contrôleurs de noeuds d'équilibrer le trafic sur le réseau entier.

1.3. CENTRALISATION ET DECENTRALISATION

Comme on vient de le voir, dans le passé, des considérations d'ordre économique favorisèrent le partage d'un système central parmi un nombre important d'utilisateurs. Le système d'exploitation se voyait ainsi attribuer l'importante charge que représentait le contrôle d'un tel partage. Actuellement, les progrès technologiques ont contribué d'une part à une baisse considérable des prix des composants et des mémoires, et d'autre part, à une disponibilité croissante de moyens de communication et de télétraitement. Une conséquence de cet état de choses est l'aboutissement à une nouvelle organisation possible du software. Conjointement à cette avance technologique, on a pu discerner dans certaines entreprises un désir de plus en plus marqué de décentralisation. Que ce soit pour des raisons d'ordre politique,administratif voire psychologique, on tend actuellement à la distribution des responsabilités parmi les parties concernées de façon à leur assurer une indépendance maximale ainsi qu'à l'émiettement du pouvoir de décision de façon à motiver l'exécutant dans sa tâche.
On ne peut cependant pas généraliser ces structures d'organisation et, dans certains cas, le maintien d'une volonté de centralisation peut être un frein au choix de systèmes répartis qui menaceraient de remettre en question la structure de pouvoirs en place. La principale crainte dans un tel environnement est celle d'une décentralisation des décisions. L'éclatement des services informatiques risque, aux dires de certains, de conduire à une perte de contrôle par la direction générale de l'entreprise de la gestion informatique de celle-ci et à l'élaboration de logiciels de gestion parallèles et non compatibles les uns avec les autres.

On peut établir une première grande classification des architectures de systèmes possibles et ainsi, examiner quels sont leurs impacts quant à la répartition des traitements et du stockage des données.

Dans un système centralisé, les moyens de traitement et de stockage se trouvent situés dans un même lieu. Des terminaux géographiquement dispersés dialoguent avec le système central. Dans un tel contexte, l'ordinateur central doit être pourvu d'un logiciel de base très élaboré. Celui-ci devait en effet, d'une part assurer le contrôle du partage des ressources du système et d'autre part, supporter des défaillances partielles en mettant en oeuvre des mécanismes de reconfiguration automatique. Ceci explique que, de plus en plus, des chefs d'entreprises se sont plaints de la lourdeur de leur système informatique et de la lenteur qu'implique toute modification de procédure.

Comme alternative à de telles structures de nature centralisée apparaît actuellement la tendance à la décentralisation. Celle-ci repose sur le principe de l'intégrité de la gestion à un niveau élémentaire. En d'autres termes, toute cellule de travail d'une entreprise doit avoir les moyens d'assurer sa gestion, moyens qu'elle ne partage pas avec une autre unité. Dans de telles

architectures, le traitement et le stockage de l'information eux-mêmes placés plus près des utilisateurs sont éclatés sur plusieurs ordinateurs plus petits ou sur des terminaux dits intelligents (la limite entre ordinateur et terminal devient plus floue) qui peuvent échanger des données entre eux par l'intermédiaire de moyens de communication. La notion de système central disparaît complètement, ce qui introduit peut-être un autre type de complexité dans la mesure où il faut malgré tout veiller à assurer au système global sa cohérence et sa cohésion mais les services offerts sont probablement meilleurs (temps de réponse, disponibilité du système...). Cette approche est théoriquement devenue avantageuse avec le développement de la micro-informatique. Celle-ci fera l'objet d'un examen particulier ultérieurement. (Cfr. parag. 1.5) Il est à noter à ce stade du développement qu'il faut faire une distinction entre deux approches fondamentalement différentes quant à la conception des systèmes distribués.

- Une première façon de voir les choses consiste à assurer à chaque cellule de travail une autonomie complète dans la mesure où l'information est saisie, traitée et restituée dans les lieux où elle est produite et consommée. Cette optique est parfaitement compatible avec une structure d'organisation à tendance décentralisée. Une des raisons d'agir de la sorte repose sur 'hypothèse que le personnel responsable d'une entité se sent davantage motivé s'il a l'entière responsabilité d'un ensemble d'information et de son traitement.

- Une deuxième façon de voir les choses consiste à décentraliser les fonctions comme ci-dessus mais à répartir les données entre les divers utilisateurs de façon transparente, c'est-à-dire que ceux-ci accèdent aux données sans se préoccuper de leur localisation physique dans le réseau. Les utilisateurs n'ont accès qu'à des objets logiques. Ces techniques relèvent du domaine des bases de données réparties. (Ce sujet sera abordé au parag. 1.7)

Un système semi-centralisé constitue une solution intermédiaire entre les deux extrêmes évoqués ci-dessus. Il est caractérisé par le fait que la saisie de l'information peut être effectuée de manière autonome, éventuellement faire l'objet d'un premier traitement pour ensuite être transmise depuis le site distant vers le système central où elle sera traitée à nouveau et/ou exploitée par d'autres utilisateurs. On aboutit ainsi à une concentration d'un nombre relativement faible de données au niveau central et leur traitement peut s'effectuer sans avoir recours à la multiprogrammation.

1.4. CARACTERISTIQUES DES SYSTEMES DISTRIBUES

Les points suivants reprennent différentes caractéristiques imputables aux systèmes distribués. Cette liste est loin d'être exhaustive; elle a seulement pour but de mettre en évidence certains aspects spécifiques de ces systèmes et de voir dans quelle

mesure ils sont susceptibles d'apporter des èlèments de rèponse à certains problèmes cruciaux dègagès lors de l'analyse des systèmes dècentralisès.

1. Faible coût

Le faible coût est une caractèristique importante en ce sens qu'elle constitue peut-être une des raisons principales qui est à la base de cet essor croissant de l'informatique rèpartie. Cette èvolution n'a ètè possible , comme nous l'avons vu, que grâce aux progrès technologiques. La baisse des coûts des systèmes dèpend principalement des facteurs suivants:

- importante diminution de prix des composants et des mèmoires
- disparition de tâches gènèrèes par des systèmes centralisès et massifiès
- l'importance du marchè de l'informatique rèpartie qui autorise de sustantielles èconomies d'èchelle.

Il ne faut toutefois pas perdre de vue que, parallèlement à cette diminution des coûts, d'autres charges sont à prendre en considèration. Il est en effet probable que, dans un tel cor exte, les frais de personnel soient susceptibles de s'a croître suite à la rèpartition des èquipements sur diffèrents sit s. En outre, le coût des lignes peut, dans certains cas, être non nègligeable.

2. Meilleure utilisation des ressources

On peut dire que l'informatique rèpartie contribue à une meilleure utilisation des ressources en ce sens que dans un système centralisè une part importante du système d'exploitation est consacrèe à la gestion des problèmes dècoulant du fait que le système fait l'objet de requêtes nombreuses et diversifièes. Le système d'exploitation doit en effet remplir un grand nombre de fonctions visant à assurer au système sa cohèrence. Il doit, en outre, garantir un contrôle et des protections d'accès qui, dans certains contextes, sont loin d'être nègligeables.

Cet aspect complexe a comme consèquence immèdiate une dègradation des performances et cela explique la tendance de l'informatique rèpartie à regrouper les diffèrents types de travaux et à affecter leur traitement à des processeurs spècialisès. Entendons par là le fait que cette spècialisation est perceptible au niveau même du fonctionnement des ordinateurs. Contrairement à l'architecture des premiers ordinateurs qui comportaient un processeur central unique dont dèpendait l'exècution de l'ensemble des processus, on a tendance actuellement à le remplacer par une association de processeurs spècialisès (pocesseur d'entrèe/sortie, processeur de base de donnèes,...)

3. Modularitè

Il y a bien sûr diffèrentes façons d'envisager une dècoupe

plus un concept de localisation géographique.

5. Traitements simples au niveau local.

Il est probable que les traitements effectués aux niveaux décentralisés soient plus simples que ceux effectués au niveau central. Cela tient essentiellement au caractère spécialisé que revêt alors le traitement. Cette moindre complexité est également dûe au fait qu'à ce stade, il ne faudra pas tenir constamment compte de l'ensemble des cas de figures possibles dans le cas théorique général. Il est absurde de vouloir traiter automatiquement tous les cas possibles, quels qu'ils soient, à l'échelle d'une unité décentralisée. Les cas d'exception demandent en général une intervention humaine en fin de traitement d'où l'intérêt d'avoir des outils permettant une réelle interaction homme-machine. Dès lors, à la rencontre d'un cas particulier, l'individu peut décider, modifier ou adapter les opérations à réaliser (ce qui n'est pas le cas dans un système centralisé où les exceptions sont soit prises en compte après coup, soit traitées avec des retards parfois considérables.

6. Meilleurs services aux utilisateurs.

On a souligné ci-dessus la tendance en informatique à la spécialisation des traitements. Ce choix des processeurs les mieux appropriés à des travaux particuliers a pour conséquence des meilleurs temps de réponse. D'autres éléments des systèmes répartis contribuent à cet effet. A titre d'exemple, mentionnons:

- le simple fait que l'on peut transmettre des informations à un taux plus rapide
- la possibilité de réduire le temps d'exécution d'une transaction en effectuant au préalable à un niveau décentralisé un pré-traitement.

Les temps d'accès risquent également d'être nettement plus satisfaisants et ce , notamment à cause

- de volumes moindres de données manipulées
- d'un système d'exploitation moins complexe

Remarquons toutefois que dans certains cas où l'on a recours aux systèmes de gestion de bases de données, ces temps sont susceptibles d'augmenter singulièrement.

7. Problèmes de sécurité.

La notion de sécurité recouvre en fait deux concepts: la réaction du système en cas de panne et le problème de vulnérabilité.

- Réaction du système en cas de panne.

Dans un système distribué, on peut parler de conséquences

6. Meilleurs services aux utilisateurs.

On a souligné ci-dessus la tendance en informatique à la spécialisation des traitements. Ce choix des processeurs les mieux appropriés à des travaux particuliers a pour conséquence des meilleurs temps de réponse. D'autres éléments des systèmes répartis contribuent à cet effet. A titre d'exemple, mentionnons:

- le simple fait que l'on peut transmettre des informations à un taux plus rapide
- la possibilité de réduire le temps d'exécution d'une transaction en effectuant au préalable à un niveau décentralisé un pré-traitement.

Les temps d'accès risquent également d'être nettement plus satisfaisants et ce , notamment à cause

- de volumes moindres de données manipulées
- d'un système d'exploitation moins complexe

Remarquons toutefois que dans certains cas où l'on a recours aux systèmes de gestion de bases de données, ces temps sont susceptibles d'augmenter singulièrement.

7. Problèmes de sécurité.

La notion de sécurité recouvre en fait deux concepts: la réaction du système en cas de panne et le problème de vulnérabilité.

- Réaction du système en cas de panne.

  Dans un système distribué, on peut parler de conséquences réduites suite à la survenance d'incidents en ce sens que les effets seront généralement localisés. Ce sera le cas dans la mesure où l'on vise à distribuer les fichiers et encourager le traitement maximum au niveau local. Il faut toutefois attirer l'attention sur le fait que dans un environnement base de données réparties , il s'agira de mettre en oeuvre des techniques relativement complexes de reconfiguration automatique, ce qui peut remettre en cause le caractère local d'une panne.
  Un autre point intéressant des systèmes répartis réside dans le fait que l'on peut utiliser de telles architectures de différentes façons selon l'état de fonctionnement du système et du réseau. Ainsi, en cas de certaines pannes, on peut envisager de continuer un travail de saisie et de contrôle au niveau local grâce aux moyens de traitement et de stockage dont on dispose (cassettes, disques souples...). Cette information conservée localement pourra ensuite être " ré-injectée " dans le système en vue d'un traitement ultérieur

- Vulnérabilité.

L'un des corollaires de la concentration des moyens de traitement (et des fichiers qui leur sont associés) est une vulnérabilité croissante de l'entreprise à tout danger menaçant. Son talon d'Achille: le centre informatique. Celui-ci est en effet exposé à différents types de destruction tels que:

- la destuction physique (incendies, innondations, explosions....)
- la paralysie du système (incident d'alimentation électrique, grève...)
- le vol d'information
- la fraude massive

Les systèmes décentralisés peuvent dans une certaine mesure apparaître moins vulnérables du simple fait de leur dispersion, mais d'un autre côté ils semblent tout à fait assujettis à leurs moyens de transmisssion. Par conséquent, en agissant sur ces moyens, ne risque-t-on pas des problèmes équivalents de paralysie totale du système?

8. Facteurs psychologiques.

Sans accorder une importance excessive à ces facteurs d'ordre psychologique, il faut toutefois reconnaître que ceux-ci peuvent avoir une influence au sein d'une entreprise. Ainsi, dans un contexte décentralisé, le fait d'accorder à une unité une certaine autonomie et une certaine autorité quant à la gestion d'un ensemble d'informations peut avoir pour conséquence de dégager un sens accru des responsabilités et une meilleure attention au travail exécuté à ce niveau.

1.5. ROLE PREPONDERANT DE LA MICRO-INFORMATIQUE

L'apport de la micro-informatique en matière de systèmes répartis est considérable dans la mesure où elle permet d'atteindre une simplicité qui était exclue dans le contexte d'une grosse machine centrale. Il est intéressant à ce propos de relever quelques caractéristiques propres à ces outils de l'informatique répartie:

1. faible coût:

Cela est essentiellement dû, comme on l'a déjà mentionné, aux progrès technologiques. Soulignons deux facteurs en faveur de l'abaissement du coût des micro- ordinateurs:

- leur coût peut baisser plus vite que celui des gros ordinateurs du fait du jeu de la production de masse,
- les micro-ordinateurs n'ont besoin ni d'une grande puissance de calcul, ni d'une capacité de mémoire étendue, ni d'unités d'impression rapide.

2. performances adaptées aux besoins:

Les micro-ordinateurs ne sont pas conçus pour traiter des chaînes complexes de programmes sophistiqués mais pour résoudre des problèmes simples et spécifiques. (ex. facturation, gestion de stock, paye,...) Un autre point intéressant relatif à cet aspect des choses consiste à utiliser la micro-programmation. Disposer de machines micro-programmables peut être très avantageux dans la mesure où l'on peut atteindre une très grande souplesse d'adaptation. On peut en effet envisager de mettre au point un jeu d'instructions particulièrement adapté et efficace dans le cadre d'une application particulière. Bien qu'une telle réalisation est loin d'être évidente!

3. autonomie:

Les micro-ordinateurs doivent être vus comme des outils entièrement autonomes, et ce, à deux points de vue:

- autonomie physique:
  Les micro-ordinateurs sont des ordinateurs à part entière en ce sens qu'ils ne doivent nullement être connectés à un système central pour assurer leur fonctionnement.

- autonomie du software:
  On peut envisager de laisser une possibilité de modifier la programmation de façon relativement indépendante au niveau du micro-ordinateur. Il est évident que de tels changements ne peuvent se faire qu'en respectant certaines contraintes impératives en vue de ne pas compromettre l'intégrité du système global.

4. taille réduite:

Pour atteindre une souplesse d'utilisation,une taille réduite est indispensable. Les micro-ordinateurs ont l'avantage d'être facilement transportables et non pas cantonnés dans un local particulier: la fameuse " salle de l'ordinateur".

5. Maintenance réduite:

Cette caractéristique de maintenance réduite est essentielle au développement des micros-ordinateurs. La maintenance et le dépannage sont, en effet, des éléments cruciaux dont dépendent la bonne marche d'un système informatique et sa rentabilité.
Dans ce domaine, plusieurs facteurs jouent en faveur des micro-ordinateurs:

- Les composants utilisés pour leur fabrication sont extrêmement fiables.
- Les constructeurs s'efforcent de réduire au maximum les parties
  mécaniques dans les périphériques.
- La modularité du hardware permet de procéder à des échanges standard en cas de panne.

Un aspect important de la micro-informatique réside dans le fait qu'elle ouvre une porte à une nouvelle classe d'utilisateurs potentiels. Son avènement a pour conséquence de faire pénétrer l'informatique dans des domaines où sa présence était jusqu'alors impensable compte tenu du coût élevé de son matériel. De ce point de vue (commercial), il apparaît que les domaines de la petite gestion, de l'automatisme et l'instrumentation constituent les points forts, donc moteurs de cette future informatique.

1.6. RESEAUX LOCAUX

Un usage intéressant des micro-ordinateurs consiste à relier ceux-ci de façon à former un réseau local. Un tel réseau se caractérise essentiellement par le fait qu'il permet des transmissions à grande vitesse sur des distances relativement courtes. Il n'entre pas dans le cadre de cet exposé de se livrer à un examen approfondi de la technologie ni des différentes configurations existantes dans le cadre des réseaux locaux. Un travail de synthèse relatif à ce sujet figure dans [10].

Cette évolution relativement récente suscite un intérêt chez bon nombre d'utilisateurs. Pour s'en convaincre, il suffit de mentionner quelques systèmes exitants ainsi que les efforts consentis dans l'examen des possibilités offertes par de tels systèmes. A titre d'exemple, citons simplement:

- Le réseau MITRIX de la firma américaine MITRE [27].
- Le réseau ETHERNET de la firme américaine XEROX [5].
- Le réseau SPIDER de la firme américaine BELL TELEPHONE [14].
- Le réseau DCS (Distributed Computing System) de l'UNIVERSITY CALIFORNIA IRVINE [12].

- Le projet MICROBE qui est une réalisation d'un système de base de données relationnelle répartie sur un réseau local de micro-ordinateurs.[13].

1.7. BASES DE DONNEES REPARTIES

1.7.1. Notion de base de données réparties

L'approche préconisée dans [24] est de soutenir que la notion de base de données réparties (B.D.R.) est en fait un concept relativement ancien. Selon les auteurs, le caractère réparti d'un ensemble d'informations a été soumis à une réalité centralisatrice inhérente à des contraintes d'ordre technologique. L'atout technique qui a permis à l'informatique d'appréhender les bases de données réparties est très clairement le développement des réseaux. Cette situation associée au fait que les bases de données traditionnelles centralisées montrent aujourd'hui avec évidence les limites de leur utilisation contribuent à l'implémentation des bases de données réparties. A titre d'exemple, citons la référence [7] où l'on peut

constater les efforts entrepris pour adapter un système de gestion de base de donnée existant dans une optique de répartition.

Pour illustrer ce point, il suffit toujours selon les auteurs, de citer deux exemples représentatifs de la dégradation du niveau de service offerts aux utilisateurs dans un contexte de bases de données centralisées.

Le premier exemple concerne une entreprise comportant un siège central et une multiplicité de filiales géographiquement dispersées. (exemple: multinationales,...)
Dans une solution centralisée, la base de données représentant l'ensemble du système d'information de toute l'entreprise est installée et gérée dans le service informatique du siège central. Les filiales possèdent généralement une copie de la partie de la base de données qui leur est utile, copie sur laquelle elles font leur traitements locaux sans interférer avec le système central; les copies sont périodiquement remises au siège central afin de mettre à jour la base de données.
Imaginons alors que le système d'informations de l'entreprise s'enrichisse continuellement et que, conséquemment, la base de données du siège central s'accroisse. Trois critiques essentielles sont imputables à un tel type d'organisation.

6. La fiabilité

Il est prévisible que la probabilité de panne soit fonction de la complexité du système mis en oeuvre à moins que l'on investisse de manière conséquente dans la gestion des problèmes de fiabilité.

7. Augmentation des coûts non proportionelle à l'amélioration du service rendu.

Des considérations d'ordre économique tendant à prouver que si, sans pour autant changer la nature du problème, on amplifie le volume des données qu'il met en jeu au-delà d'un seuil critique, le coût de la solution peut augmenter en proportion et qu'à celui-ci s'ajoute le coût de l'infrastructure développée pour permettre la mise en place de la solution.

8. Inadéquation de l'outil informatique aux besoins de l'application qui l'emploie.

La tendance en gestion des entreprises allant essentiellement dans le sens de la décentralisation , il est probable qu'un outil informatique hautement centralisateur s'avère mal adapté.

Le deuxième exemple est relatif aux situations où un ensemble de personnes (physiques ou morales) possédant chacune leur propre système d'information décident à un certain moment de réunir leur capital d'information afin d'accroître le niveau de service rendu à chacun d'entre eux. Dans un tel exemple, il

est évident que la mise en commun des bases de données existantes ne doit nullement correspondre à une quelconque perte d'individualité des partenaires qui doivent rester maîtres en leur domaine. Seules des procédures de communication et synchronisation doivent être réalisées pour permettre ce nouveau service commun.

1.7.2. Principaux avantages des bases de données réparties.

Le but de ce paragraphe est de présenter un certain nombre d'avantages des bases de données réparties. Ces avantages sont regroupés en trois grandes classes que nous examinerons successivement: les aspects humains, les aspects relevant des organisations d'entreprises et enfin les aspects économiques.

1. Les aspects humains

- Résistance psychologique à un système centralisé.

  Si l'on examine le problème de l'implémentation d'une base de données centralisée dans une entreprise où chaque service dispose déjà d'un ensemble de fichiers structurés selon leurs critères particuliers, on voit que la constitution d'une bases de données boulverserait profondément les habitudes des utilisateurs et remettrait en cause la vision personnelle qu'ont ceux-ci de leurs données. Les bases de données réparties proposent une solution à ce problème en permettant à ces services de conserver leur individualité, leurs méthodes de travail, leurs façons de voir les choses, tout en permettant l'établissement d'une standardisation ainsi qu'un contrôle d'accès aux données.

- Perte de pouvoir associé au contrôle des données

  Dans le cadre d'une centralisation abusive, les divers services fournissent au centre de traitement de l'information des données et utilisent des résultats définis en dehors d'eux. Cette situation n'est en général pas bien supportée car elle constitue une perte réelle de contrôle et de décision. Elle a pour conséquence inévitable une dégradation de la qualité des données fournies.
  L'emploi d'une base de données répartie permet de retrouver un équilibre plus juste en redonnant à chaque service des pouvoirs (traitement et gestion des données locales au service) et en rendant néanmoins possible un partage aisé des informations.

- Enrichissement des tâches

  La prolifération des micro-ordinateurs ainsi que la disponibilité de logiciels évolués permettent à l'utilisateur d'avoir un contact direct avec son outil. Ce

phènomène de dèmocratisation de l'informatique peut dans une certaine mesure rendre possible pour l'utilisateur de concevoir et rèaliser dans sa quasi-totalitè une application de gestion. Il ne se contente plus de saisir uniquement des donnèes qui seront effectivement utilisèes par d'autres.

2. Les aspects organisationels

La nècessitè de rèpartir les informations, leur gestion ainsi que leur traitement existe dans la rèalitè vècue indèpendamment de toute application informatique. L'outil rèparti peut s'avèrer plus puissant que la simple somme des outils locaux qu'il intègre. Cela provient du fait que la mise en commun d'information de provenances distinctes produit un phènomène de synergie. Utilisèe pour une bonne cause, cette augmentation de puissance constitue un avantage indèniable. Il va de soi qu'on se trouve ainsi devant un dèbat à l'ordre du jour et qui est celui des dangers que reprèsente cette puissance vis-à-vis des libertès individuelles et collectives. Il n'entre pas dans notre propos d'adopter une position dans ce dèbat, mais il paraissait important d'en souligner l'existence.

3. Facteurs èconomiques

- Diminution de la vulnèrabilitè

  Dans une base de donnèes centralisèe, un incident quelconque (volontaire ou non) peut pènaliser la totalitè des utilisateurs et rendre inutilisable l'ensemble des moyens informatiques, Dans une base de donnèes rèpartie, un incident peut n'avoir qu'une portèe limitèe. Les informations constitutives de la base ètant dissèminèes sur plusieurs sites, un accès au sous-ensemble des informations localisèes sur les sites non perturbès demeure possible.

- Diminution des coûts

  Le simple fait d'attribuer un caractère local à certains fichiers peut rèduire le trafic d'information sur un rèseau. Dans une base de donnèes rèpartie, l'utilisateur a accès à un enregistrement que celui-çi soit local ou situè sur un site à distance. C'est donc cette faible quantitè d'information qui est vèhiculèe du site èmetteur au site rècepteur. Il n'en est pas toujours de même dans une organisation classique centralisèe où c'est souvent la totalitè du fichier qu'il est nècessaire de faire parvenir au site rècepteur pour en permettre l'exploitation.

1.7.3. Problèmes spècifiques aux bases de donnèes rèparties

Lorsqu'il s'agit de rèpartir des donnèes sur un rèseau, deux aspects essentiels sont à considèrer.

Un premier aspect important réside dans la façon dont les données seront stockées et quels seront les chemins d'accès. La répartition du contrôle impose le maintien des copies multiples de certaines informations. Il s'agit d'en conserver la cohérence tout en permettant un accès en mise à jour. Il est également nécesaire de se pencher sur le problème de la détermination au nombre optimal de copies et de leur localisation sur le réseau. Ce problème se ramène à un cas particulier de l'allocation optimale des ressources; des modèles mathématiques qui tiennent compte de l'évolution dynamique de l'allocation sont décrits dans la littérature. Il convient notamment à ce niveau d'assurer un temps de réponse raisonnable suite à la requête d'un utilisateur. Cela soulève un certain nombre de problèmes d'optimisation.

Un deuxième aspect tout aussi primordial concerne la cohérence des données. Il s'agit en effet de veiller à ce que

- les informations manipulées satisfassent aux contraintes d'intégrité
- seules les personnes autorisées réalisent certaines opérations de consultation et de mise à jour;
- les accès concurrents n'altèrent pas la cohérence de la base de données;
- des utilitaires adaptés gèrent la base de données en cas de panne logicielle ou matérielle.

CHAPITRE 2

SITUATION DU PROJET

2.1. BUT POURSUIVI

Le projet à réaliser dans le cadre de ce mémoire consistait à étudier les problèmes susceptibles de se poser au cours de l'implémentation d'une application de gestion. - l'application "PETIT-PAS"- sur un réseau local de micro-ordinateurs. Le réseau devant supporter l'application est le réseau COBUS développé au Laboratoire de micro-informatique de l'Ecole Polytechnique Fédérale de Lausanne,en Suisse. Une description détaillée en est donnée au chapitre 3. Les spécifications fonctionnelles du cas " PETIT-PAS " sont définies avec précision dans [26], l'essentiel en est rappelé au chapitre 4. Le travail proprement dit consistait à définir une architecture d'implémentation qui respecte les spécifications fonctionnelles et notamment la découpe établie en fonctions-utilisateurs. Pour l'architecture retenue, il était alors question de définir et réaliser les fonctions-systèmes sous-jacentes à la mise en oeuvre du projet.

On pouvait donc à priori, envisager la démarche suivante pour mener à bien cette réalisation:

1. Détermination d'une architecture d'implémentation

   Dans un contexte réparti, la détermination d'une architecture d'implémentation revêt deux aspects essentiels: d'une part, décider de l'affectation des différents traitements sur les différents processeurs,d'autre part,décider de la re'partition des données sur les mémoires auxiliaires. Ajoutons à cela la spécification de la nature des messages échangés entre les divers processus.

2. Définition des fonctions-système

   La définition des fonctions-système recouvre divers aspects. Cela comporte essentiellement:

   - la gestion des accès aux mémoires auxiliaires
   - la résolution de problèmes de synchronisation, d'exclusion mutuelle... caractéristiques de ce genre d'applications.
   - la gestion des échanges d'information
   - dans le cadre particulier de l'application , une éventuelle gestion d'écran

3. Examen des problèmes de sécurité

Il faudra en outre ne pas perdre de vue les aspects de protection, sécurité... qui assurent au système sa fiabilité. Une fois les termes du problème clairement définis, il s'avèrera opportun de porter un choix sur les parties qui feront réellement l'objet d'une implémentation. Soulignons enfin le fait que le but visé ne consistait pas à s'attarder sur des problèmes d'optimisation et de performance mais plutôt à mettre en évidence les difficultés typiques d'une telle implémentation.

2.2. PRESENTATION DE L'APPLICATION " PETIT-PAS "

Pour introduire l'application " PETIT-PAS " sans toutefois entrer dans des explications trop détaillées, limitons nous aux quelques considérations suivantes:

2.2.1. Présentation et organisation de la firme

La firme PETIT-PAS est spécialisée dans la vente par correspondance d'articles d'habillement. Sa structure d'organisation telle qu'elle se présentait préalablement à la décision d'informatiser est représentée schéma 1. Comme on peut le voir, la firme est organisée en départements. Chacun d'eux possède une direction et comprend différents services.

1. Direction générale (pour mémoire)

2. Département commercial.

   - Service "Vente et Marketing"
     Ce service s'occupe du marketing et de la politique de vente: définition du catalogue, choix des produits, fixation des prix de vente, publicité et promotions, etc.
   - Service "Gestion de la clientèle"
     Ce service analyse le profil de la clientèle : profil socio-économique, répartition géographique... Ses outils principaux sont un fichier des clients (suivi des ventes à chaque client) et un fichier des prospects (clients potentiels à qui est envoyé le catalogue).
   - Service "Gestion des produits"
     Ce service étudie l'évolution des ventes par produit et type de produit. Il est en contact très étroit avec les services "Vente et Marketing" et "Clientèle".

3. Département de la production.

   Le département de la production s'occupe des tâches allant de la réception des commandes à l'expédition des colis. Il comprend les services suivants:

- Service d'enregistrement des commandes.

  - contrôle des commandes,
  - création du bon de commande interne nécesssaire aux différentes opérations de traitement des commandes.

- Service de préparation des commandes.

  - constitution des colis, emballage et pesage.

- Service de facturation.

  - création des factures et documents d'expédition.

- Service d'expédition.

  L'expédition se fait par chemin de fer. La firme est liée à la Société des chemins de fer par un contrat aux termes duquel, notamment, la Société des chemins de fer se charge de la " prise à domicile " des colis et facture ses prestations globalement par quinzaine.

4. Département "Achats et Fournisseurs".

   - Service central des stocks.

     Le service est chargé de la centralisation des informations relatives à l'état des stocks, informations qu'il communique quotidiennement aux autres services de la firme.

   - Service acheteurs.

     Ce sont ces services qui prennent les décisions relatives au choix des fournisseurs, aux quantités à réserver, commander et réapprovisionner et qui s'occupent du suivi des livraisons-fournisseurs.

5. Service d'entreposage.

   Ce service est chargé de la réception des livraisons des fournisseurs, du contrôle de la marchandise, de l'approvisionnement du service de réparation des commandes clients.

6. Département financier.

   - Service comptable.

     Ce service est notamment chargé de la gestion des comptes clients et comptes fournisseurs.

   - Service de la Trésorerie.

- Service des investissements et financement à long terme.

  Actuellement, les investissements portent principalement sur l'appareil de production (chaînes de constitution des colis, service des commandes téléphonées, terminaux de consultation de l'état des stocks...) Cette automatisation des tâches de production vise à répondre aux objectifs fixés par la Direction Générale.

7. Département administratif.

   Ce département comprend entre autres le service du contentieux et le service du fichier central des adresses (adressographe fournissant aux différents services de la firme les étiquettes adresses préimprimées; également chargé de l'envoi du courrier préparé par les différents services).

8. Département du personnel (pour mémoire).

   Le détail des procédures utilisées dans le cadre de la solution manuelle est donnée dans [30]

Un point important est à retenir; il est expliqué ci-dessous. Dans cette solution, une commande reçue était satisfaite en fonction des disponibilités des stocks. Malheureusement, l'ensemble des commandes pouvant être honorées en une fois, s'avérait être seulement de l'ordre de 50%. Pour les commandes où il restait une partie à livrer, on procédait comme suit. Une première livraison partielle avait lieu, pour laquelle on facturait au client des frais de transport. Une seconde livraison avait ensuite lieu si dans un délai de 21 jours, on avait pu satisfaire totalement ou partiellement au restant de la commande. Quoiqu'il en soit, une fois ce délai expiré, le client recevait soit une livraison relative à une partie ou l'entièreté de la quantité restant à livrer, soit un justificatif de non-livraison. La commande était alors considérée comme apurée par la firme et le client prêt à attendre et désireux malgré tout de recevoir une éventuelle quantité pour laquelle il n'avait pas pu être satisfait, devait repasser commande.



- Service Vente & Marketing
- Service Gestion Clientèle
- Service Gestion Produits

- Service Enregistrement Cdes
- Service Préparation Cdes
- Service Facturation
- Service d'expédition

- Service central des stocks
- Service Acheteurs

- Service Comptable
- Service de la Trésorerie
- Service des Investissements pt Financement à long terme

- Service du Contentieux
- Service du Fichier Central des Adresses

2.2.2. Objectifs de la direction générale›

Le souhait de la Direction Générale est de mettre en place un système informatique en vue de traiter d'une part les commandes clients et les livraisons y afférant, d'autre part les commandes fournisseurs ainsi que les réapprovisionnements.

Les objectifs de gestion qu'elle poursuit sont de deux ordres:

- A. Un objectif financier

  Amélioration de la rentabilité financière à court terme. Essentiellement , cela implique les trois sous-objectifs suivants:

  A.1.Amélioration de la rotation des stocks

  A.2.Réduction du découvert financier

  A.2.1. Enregistrement plus rapide des factures
  A.2.2. Minimisation des coûts de fonctionnement en stock

  A.3.Réduction des coûts de fonctionnement

- B. Un objectif de marketing:

  Amélioration du service à la clientèle.

Les objectifs informationnels déductibles des objectifs de gestion sont repris ci-dessous:

- A.
  - A.1
    Minimisation du délai entre le réception d'une livraison fournisseur et son enregistrement dans le stock.
    Minimisation du temps de diffusion de l'information concernant l'état du stock.
    Meilleure définition des paramètres relatifs à la gestion du stock.
    Meilleure connaissance des ventes (panel des consommations...)
    Meilleure connaissance des délais fournisseurs
  - A.2
    Accélération du traitement des factures.
    C'est-à-dire:
    - réduction du temps de séjour d'une commande dans le système
    - meilleure gestion des débiteurs
    - amélioration des conditions de paiement aux fournisseurs.
  - A.3
    Diminution des coûts de rupture

C'est-à-dire:
- réduction du nombre des ruptures
- réduction du coût d'une rupture

- B
  - Diminution du délai de livraison d'une commande
  - Réduction du nombre des livraisons différées
  - Réduction des commandes refusées suite à une erreur de rédaction dans le texte de la commande.

Toujours dans la même optique, la firme envisage de mettre sur pied un service de réception des commandes par téléphone. Ce service sera doté de terminaux afin de pouvoir connaître instantanément l'état des stocks.

2.2.3. Critique de l'existant

Les principaux griefs imputables à la solution manuelle sont:

- un délai intolérable de mise à jour des stocks
- le fait que le fichier d'état des stocks soit tenu à deux endroits différents dans l'entreprise.

Sur base de ces considérations, une nouvelle solution a été élaborée.

2.2.4. Proposition d'automatisation

2.2.4.1. Modification de la politique de l'entreprise

En vue de répondre aux objectifs évoqués ci-dessus, la firme a modifié sa politique de livraison en ces termes. Elle a décidé que désormais le nombre d'expéditions différées pour une commande était à priori illimité. L'analyse de la nouvelle solution a permis de mettre en évidence qu'il était commercialement avantageux de procéder de la sorte. Les livraisons s'effectueront au fur et à mesure des réapprovisionnements,; la commande ne sera archivée qu'après son apurement complet ou épuisement complet des articles restant à livrer.

2.2.4.2. Nouvelle organisation de la firme.

La mise en oeuvre de la solution automatisée provoque certaines modifications structurelles. Globalement, celles-ci consistent d'une part à changer quelque peu l'organisation de certains départements, d'autre part à adjoindre un nouveau service: " le centre de traitement de l'information" qui sera placé directement sous l'autorité de la direction générale.

La nouvelle organisation de la firme est représentée schéma 2. Elle est la suivante:

1. La Direction Générale et les Départements Commercial, Financier , et du Personnel restent inchangés

2. Le département de Production sera organisé comme suit:

   - Service d'enregistrement des commandes
     Celui-çi procédera à l'aide de terminaux au contrôle des commandes , à la mise à jour du fichier clients ainsi qu'à l'enregistrement des commandes.

   - Préparation des commandes.
     Ce service effectuera la recherche en magasin par série de commandes.

   - Service emballage et expédition.
     Celui-çi procédera à l'aide de terminaux à la reconstitution de la commande, au lancement de la facturation et à l'emballage et l'expédition des commandes.

   - Inventaire permanent
     Tous les vendredi midi, le service de préparation des commandes, d'emballage et d'expédition cessent leur travail habituel et procèdent à un inventaire permanent. Ils rangent les rayons, comptabilisent le nombre d'articles de chaque rayon et signalent l'état des stocks à un gestionnaire des stocks qui corrige éventuellement la quantité en stock dans le "fichier d'état des stocks".

3. Département Achats et Fournisseurs.

   Ce département n'est plus constitué que du seul service des achats qui se charge d'effectuer la gestion des fournisseurs: choix des fournisseurs, commandes aux fournisseurs, suivi des commandes et des livraisons, analyse de l'état des stocks et des commandes client. Ce service est également doté de terminaux.

4. Département d'entreposage.

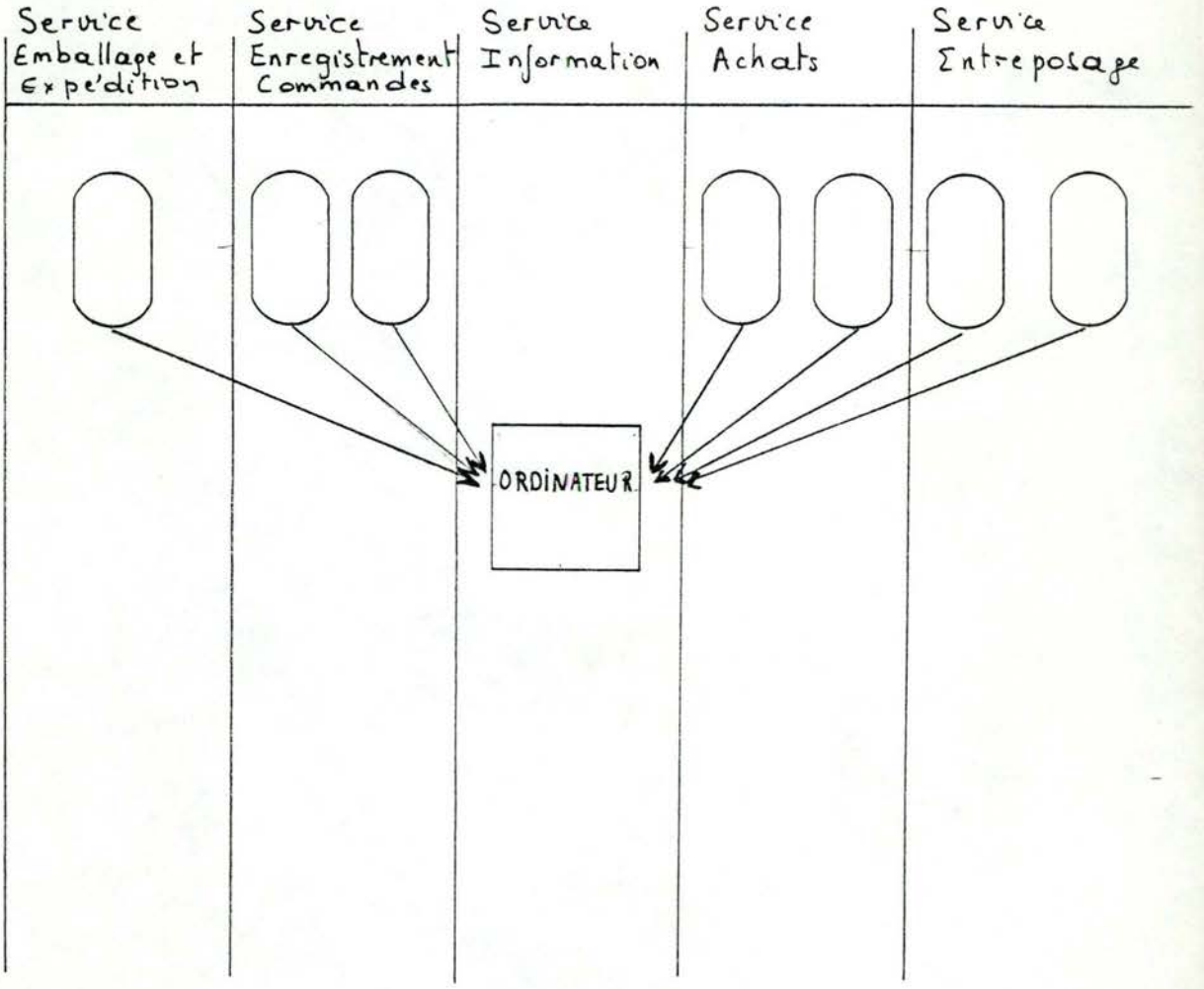
   Ce département s'occupe de la réception des livraisons fournisseurs, de leur contrôle et de leur enregistrement par terminal. Il est aussi responsable du rangement dans les magasins des marchandises reçues.

5. Le centre de traitement de l'information.
   Nouvellement créé, ce centre doté d'un matériel adéquat effectuera dorénavant:

- en liaison temps réel avec le service d'enregistrement des commandes client, le contrôle et l'enregistrement de ces commandes
- en liaison temps réel avec le service d'entreposage, le contrôle et l'enregistrement des livraisons fournisseurs
- en liaison temps réel avec le service d'emballage et d'expédition, l'édition de tous les documents d'expédition en ce compris la facture
- en liaison temps réel avec le service des achats, une aide à la préparation des commandes fournisseurs

Compte tenu de ses liaisons avec les autres services, il assume également les traitements suivants:

- mise à jour de l'état des stocks et des ressources
- gestion des commandes différées
- optimisation des recherches en magasin (ceci afin d'optimiser les parcours).



DEUXIEME PARTIE

ELEMENTS EXISTANTS

CHAPITRE 3

3.1. REMARQUE PRELIMINAIRE

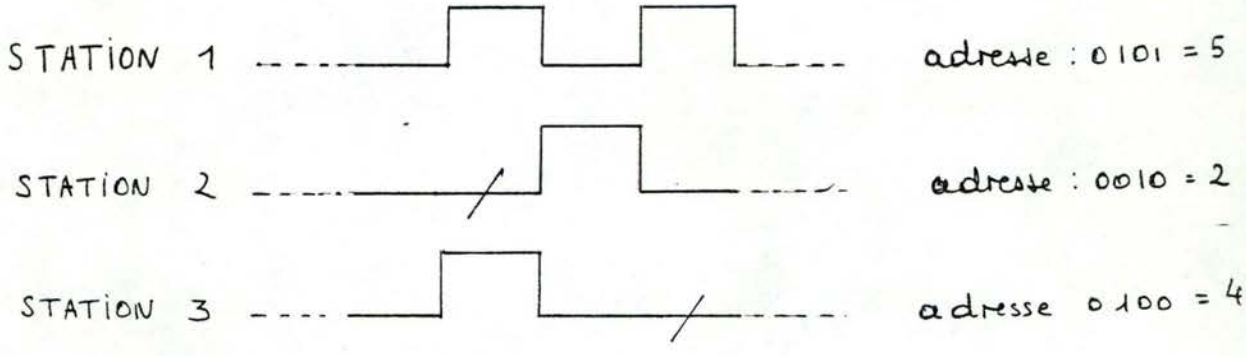
Le but de ce chapitre est de donner une vue synthétique du réseau afin de mettre en évidence pour un potentiel utilisateur, l'ensemble des possibilités qui lui sont offertes. A cet effet, on s'attardera en premier lieu sur quelques généralités. On fera ensuite apparaître la configuration du réseau et on donnera une description succinte des divers éléments qui y sont connectés. Une distinction sera faite entre la totalité des équipements disponibles et ceux qui seront utiles dans le cadre du projet . On examinera alors l'ensemble du réseau à divers points de vue: hardware, protocole de transmission et système d'exploitation.

3.2. GENERALITES

Le réseau local COBUS (COaxial BUS) a été mis au point au Laboratoire de Micro-informatique de l'Ecole Polytechnique Fédérale de Lausanne. Ce système à topologie de bus a pour but de permettre l'interconnexion de plusieurs stations intelligentes destinées à échanger de l'information. Le bus coaxial permet de relier jusqu'à 64 stations et peut atteindre une longueur maximale de 200m.
Les transmissions se font en série et sont contrôlé es de manière distribuée par un mécanisme de contention.
Les principes généraux de ce type de contrôle sont les suivants.
Toutes les stations se disputent l'accès au réseau de communication. Les intentions d'émettre étant tout à fait aléatoires, il peut arriver que plusieurs stations décident d'envoyer un message pratiquement au même instant; auquel cas, on dit qu'il y a collision.
Le contrôle par contention repose sur le fait que chaque station est capable de détecter une telle collision ou une destruction de message. Dès lors, une station décelant un événement de ce type arrête immédiatement son émission, attend pendant un certain intervalle de temps et retransmet son message. L'occupation du bus est généralement telle que les collisions soient assez rares. Cela provient du fait que, dans un réseau local, les lignes de transmission ont une grande largeur de bande.
Dans le cas précis de COBUS, le contrôle par contention est assuré comme suit: chaque station connectée sur le réseau écoute continuellement le bus. Toute station détecte donc le passage d'un message sur la ligne et n'émet qu'en l'absence de message détecté. Lorsqu'elle émet, elle échantillonne le bus et vérifie si ce

qu'elle reçoit correspond aux bits qu'elle vient d'émettre. Le résultat de son échantillonnage est en fait une fonction "OR" effectuée sur les bits d'adresse envoyés à cet instant sur le bus par l'ensemble des stations désirant émettre. Par conséquent, si elle est seule à émettre, elle ne constatera, sauf incident, aucune différence. Si, au contraire, une ou plusieurs autres stations essayent à ce moment d'envoyer un message, il arrivera un moment où l'une d'elles constatera une divergence. Dès lors, elle cessera d'émettre et attendra un certain temps avant de renvoyer son message dans sa totalité.
Le protocole assure qu'une seule station ne détectera pas de collision, celle-ci continuera donc à émettre comme si rien ne s'était passé. Il est à noter toutefois qu'il n'y a pas de station privilégiée quant à l'obtention du bus; il n'existe aucun mécanisme de priorité.
Les choses se passent en effet de la façon suivante.
Conformément au protocole (paragraphe 3.5.), une station désireuse de communiquer avec une autre doit établir un dialogue. Cela veut dire que les deux stations doivent échanger des informations dans les deux sens, pour se mettre d'accord, avant d'envoyer les données proprement dites. Pour ce faire, la station émettrice (appelée SOURCE) envoie en premier lieu une entête de message vers la station réceptrice (appelée DESTINATION). Les deux premiers bytes de cette entête sont respectivement l'adresse de la destination et l'adresse de la source. Dès lors, même si deux ou plusieurs stations qui entrent en collision s'adressent à la même destination, du moins les bytes de leur propre adresse différeront-ils. Par conséquent, après l'envoi de ces deux bytes, on peut être assuré que seule une des sources potentielles restera en dialogue avec la destination et que les deux stations achèveront de transmettre sans être dérangées.
Exemple:
Considérons le cas de trois stations émettant simultanément. Supposons qu'elles se soient adressées à la même destination et que le byte émis par chacune d'elle (et représenté fig.1)soit celui de leur propre adresse. On s'aperçoit que la station qui ne constate pas de divergence est celle d'adresse la plus élevée des stations concurrentes.

Fig. 1



3.3. CONFIGURATION DU RESEAU

3.3.1. Vue globale

La configuration du réseau telle qu'elle se présentait lors de mon arrivée au Laboratoire de Micro-informatique est schématisée fig.2

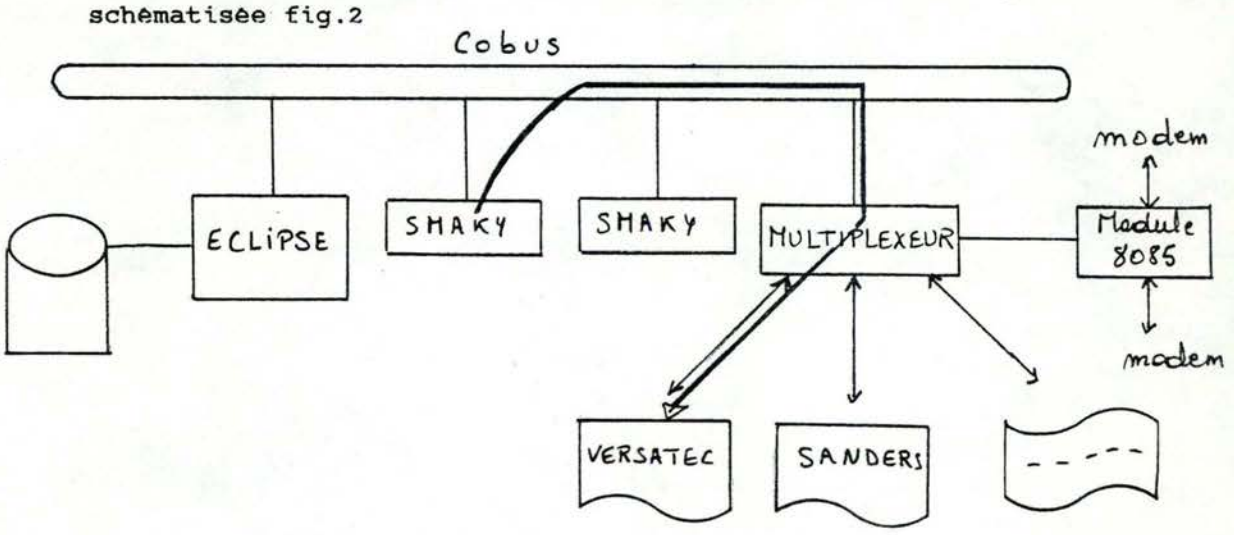


Fig. 2

Sont connectées au réseau:

- un certain nombre de stations terminales intelligentes (SMAKY6) constituées d'un microprocesseur, d'un écran à accès direct dans la mémoire et d'un clavier.
  Leurs principales caractéristiques sont les suivantes:

  - ces stations sont construites autour d'un microprocesseur Z80 de ZYLOG.
    Celui-ci travaille par mots de 8 bits.
    Il possède deux fois huits registres de 8 bits dont un remplit le rôle d'accumulateur.
    4 registres de 16 bits assument les fonctions suivantes:
    compteur ordinal, pointeur de pile et 2 autres registres à disposition de l'utilisateur.

  - La mémoire vive est de 64 K RAM

  - L'écran permet d'afficher 20 lignes de 64 caractères (majuscules ou minuscules accentuées) et environ 30.000 points (affichage graphique superposable de 120 fois 256 points).

- Le clavier est de type ASCII et comporte 7 touches supplèmentaires.

- un mini-ordinateur ECLIPSE de Data Général doté d'un disque de grande capacité qui constitue la mémoire de masse mise en commun pour les divers utilisateurs.

- un multiplexeur est également connecté au réseau.
  Il a pour but de gérer une liaison série venant d'un autre mini- ordinateur ou d'un terminal du centre de calcul ainsi que deux imprimantes (VERSATEC, SANDERS). Pour plus de détails, cfr. paragraphe 3.5.7.

Il était alors question d'adjoindre une seconde mémoire de masse au réseau, à savoir, un micro-disque WINCHESTER de MICROPOLIS mais les interfaces logiciels et matériels n'étaient pas encore réalisés.

Le micro-disque comporte 3 plateaux formatés de la façon apparaisant sur la fig.3.

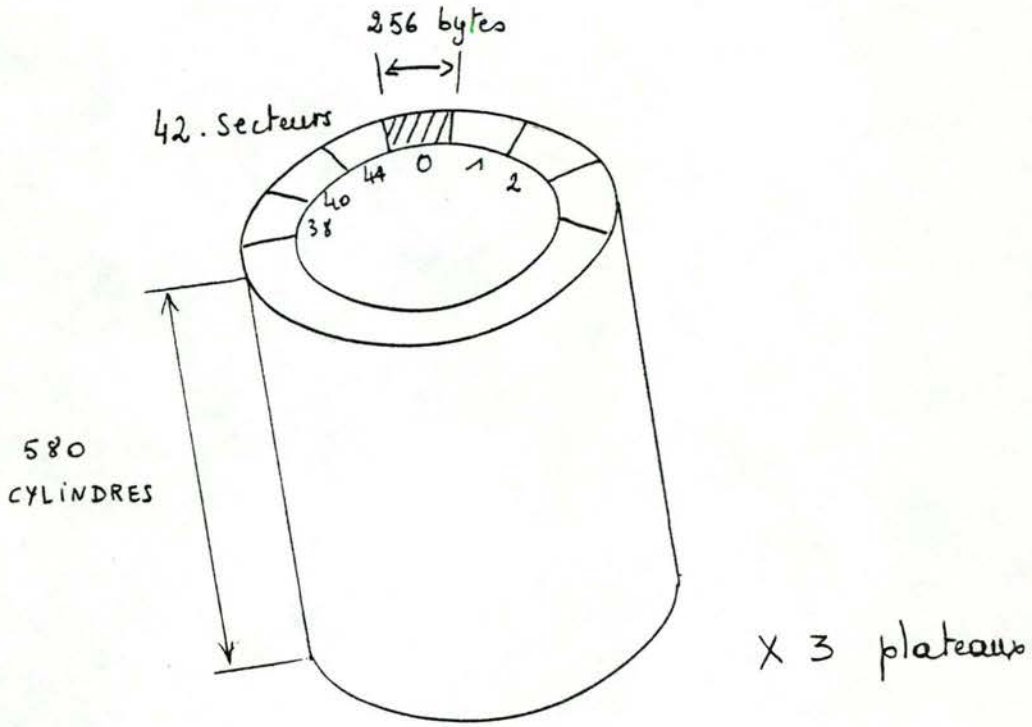


Fig. 3

Les capacités sont les suivantes:

256 * 42 = 10 752 bytes/cylindre
256 * 42 * 580 = 6 236 160 bytes/plateau
256 * 42 * 580 *3= 18 708 480 bytes au total

En conclusion, on peut donc relever l'idée de base présidant à une telle configuration du réseau.
Elle consiste à centraliser les ressources trop importantes ou trop coûteuses pour être réparties au niveau de chaque station et à les mettre à la disposition de tous les utilisateurs du réseau. C'est le cas des mémoires de masse (disques de grande capacité,...) et des équipements électromécaniques (imprimantes, perforateur/lecteur de papier...).
Un avantage inhérent à un tel réseau à topologie de bus réside dans l'indépendance de la position géographique des terminaux ainsi que dans l'extrême souplesse et la facilité d'adjoindre de nouveaux terminaux. Le réseau COBUS a prouvé son efficacité eu égard aux coûts engendrés. Les prix relativement peu élevés de ses interfaces le rend tout à fait adapté pour des petits réseaux locaux reliant des ordinateurs individuels.

3.3.2. Eléments utiles dans le cadre du projet.

Parmi l'ensemble des équipements connectés au réseau, il convient maintenant de faire la part de ceux qui seront explicitement requis pour le projet à développer. En fait, les éléments qui doivent retenir notre attention sont au nombre de deux:

- les stations terminales intelligentes (SMAKY 6)
- le micro-disque WINCHESTER

Les stations terminales intelligentes seront destinées à supporter les programmes constituant l'application de gestion à réaliser.
Le micro-disque comportera tout ou partie des fichiers nécessaires à la mise en oeuvre de l'application.
La gestion de cette seconde mémoire de masse sera assurée par une des stations terminales.

3.4. ASPECT HARDWARE.

Le niveau hardware ne sera abordé que de façon succinte étant donné que, dans le cadre de cet exposé, l'aspect utilisateur retient davantage l'attention. Le lecteur désireux d'obtenir des

détails à ce sujet se réfèrera à [10].

Signalons l'existence de 2 types d'interfaces spécifiques aux 2 types de stations connectées au réseau:

- un interface non intelligent orienté réseau (interface Cobus) utilisé avec le terminal intelligent.

- un interface intelligent (avec 2 microprocesseurs) orienté station pour le mini-ordinateur et qui utilise l'interface non intelligent vers le réseau.

Dans les stations terminales, c'est l'unique microprocesseur de la station qui active l'interface non intelligent. Pendant une transmission, le terminal ne peut pas effectuer d'autres processus. Une telle situation est acceptable pour ces stations qui travaillent en monoprogrammation.
Dans les stations de ressources, ce sont les microprocesseurs supplémentaires des interfaces intelligents qui dirigent les transmissions à travers l'interface orienté réseau, tandis que le processeur principal de la station peut continuer à effectuer d'autres processus en parallèle. Le composant principal de l'interface Cobus est un ACIA MC6850 de MOTOROLA (Asynchronous Communication Interface Adaptator). Celui-ci a pour but de sérialiser/désérialiser les bytes pour des communications de type asynchrone; il assure également un contrôle de parité au niveau du byte. Des circuits supplémentaires de type MSI SSI sont nécessaires pour assurer les fonctions de synchronisation, détection de porteuse et détection d'interférence.
Les divers composants de l'interface intelligent sont:

- 2 microprocesseurs Z80 de ZYLOG
- 2 mémoires mortes (ROM) associées aux deux micro- processeurs
- 1 mémoire vive (RAM) locale qui se partage logiquement en trois zones:
  une attribuée à chaque processeur en vue d'y stocker ses variables et une zone accessible aux deux microprocesseurs destinée au stockage intermédiaire des messages arrivant ou partant sur la ligne de transmission.
- 1 interface Cobus non intelligent orienté réseau qui est dirigé par le processeur de protocole
- 1 interface DMA entre la mémoire RAM locale et la mémoire centrale du mini-ordinateur et qui est sous contrôle du second microprocesseur.

3.5. PROTOCOLE DE TRANSMISSION

Un PROTOCOLE se définit comme l'ensemble des conventions qui permettent la coopération entre entités distinctes voulant participer à une tâche commune. Dans le cas particulier de Cobus, le protocole permet de transmettre un message d'une station quelconque à n'importe quelle autre et assure l'exactitude du

message transmis. Les échanges se font point à point. La technique de l'envoi simultané à plusieurs récepteurs n'est pas possible. Les trois niveaux habituels de protocole se retrouvent sur Cobus.

- Au niveau le plus bas, les blocs sont envoyés de façon aléatoire par les diverses unités connectées au réseau; un mécanisme relativement simple a été mis en oeuvre en vue de gérer les collisions. Les interférences sont détectées au niveau du bit.
  Cette simplicité impose en contrepartie 2 contraintes principales: un bus relativement court et une vitesse de transmission peu élevée (de l'ordre de 100 Kbits/sec.).

  Une station désirant envoyer des données vers une autre doit établir un dialogue; pour ce faire, quatre échanges sont nécessaires.(cfr.fig.4)

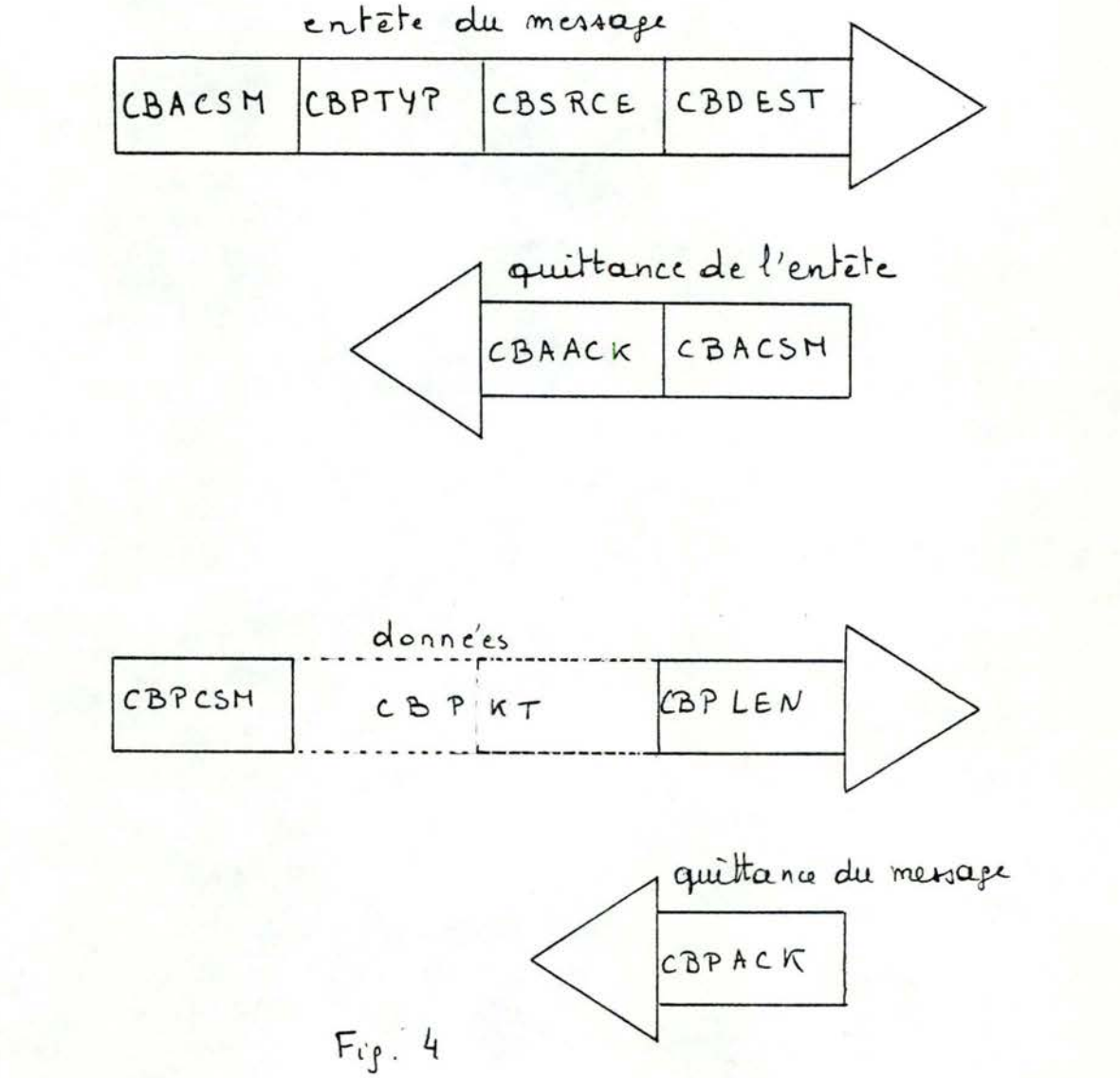


Fig. 4

1. Une entête de message est envoyée de la source vers la destination.
Elle contient:

- 1 byte CBDEST qui est l'adresse de la station destination, (de 1 à 377 octal)
- 1 byte CBSRCE qui est l'adresse de la station source , (de 1 à 377 octal)
- 1 byte CBPTYP qui indique la vitesse de l'émetteur, le type du message et la présence d'un duplicata
- 1 byte CBACSM de somme de contrôle longitudinal pour protéger les trois premiers bytes.

2. Toutes les stations connectées sur le bus lisent l'information comprise dans l'entête de message ; l'unité qui reconnaît son adresse renvoie une quittance d'entête vers la station source.
Celle-ci contient:

- 1 byte CBAACK qui contient les informations suivantes:
    a. vitesse du récepteur
    b. disponibilité du récepteur
    c. utilité du duplicata
- i byte CBASCM de somme de contrôle longitudinal pour protéger le byte précédent.

3. Une connexion point à point est dès lors établie et la vitesse de transmission peut être augmentée de façon à être en accord avec la vitesse proposée.
Les données proprement dites sont transmises de la station source vers la station destination.
Elles contiennent:

- 1 byte CBPLEN qui indique le nombre de bytes de données (jusqu'à 400 bytes octals)
- de 1 à 400 bytes octals de données CBPKT
- 1 byte CBPCSM de somme de contrôle longitudinal pour protéger les deux bytes précédents.

4. Une quittance du message est renvoyée par la station destination à la station source.
Elle contient:

- 1 byte CBPACK qui indique la bonne réception de tout le message et termine le dialogue (code fixe).

- Le second niveau de protocole garantit la validité du message échangé entre les deux stations.

- Le troisième niveau met à la disposition de l'utilisateur des appels systèmes lui permettant de faire des transferts de fichiers.

3.6. SYSTEME D'EXPLOITATION DU RESEAU.

3.6.1. Introduction.

Le système d'exploitation comprend les routines de communication à travers le réseau, la partie située dans le gestionnaire de la mémoire de masse et celle qui lui fait écho dans le processeur local.
Les objets manipulés par ce système sont les fichiers , les périphériques et les " boîtes aux lettres ".
Tous ces objets sont traités de façon similaire, par le programme d'application, qu'ils soient logés dans la station où se trouve le programme ou dans une autre station.
Les ordres principaux permettant de traiter ces objets sont:

- la création,
- l'effacement,
- le changement de nom,
- l'ouverture,
- la lecture,
- l'écriture,
- la fermeture.

3.6.2. Désignation des divers objets.

Chaque station ou (noeud) du réseau possède un nom correspondant à un répertoire ou sous-répertoire.
Les objets leur appartenant sont également dotés d'un nom; pour les référencer, il faudra donc les préfixer par le nom du directoire ou sous- directoire auquel ils appartiennent.
Exemples:

MULTIPLEXEUR:IMPRIMANTE
MEMOIRE:REPERTOIRE:FICHIER

Les noms de fichiers se trouvent dans le répertoire de la mémoire de masse.
Les noms des périphériques et des boîtes aux lettres sont conservés en mémoire dans une même table.

3.6.3. Notion de canal

Toutes les informations relatives à un fichier ouvert se trouvent consignées dans une table.
Pour les opérations postérieures à l'ouverture, le fichier sera désigné par un numéro d'entrée dans la table et non plus par son nom. C'est ce concept qui a reçu le nom de canal.
Chaque élément de la table des canaux contient les renseignements suivants:

- le début de la file de processus en attente de libération de l'objet concerné

- les adresses des tampons contenant les données qui font l'objet d'un transfert
- les caractéristiques du mode d'utilisation de l'objet (temporaire ou permanente , protégée en lecture ou en écriture, accès multiples autorisés ou non ,...)
- le nom et le statut de l'objet (fichier, périphérique, boîte aux lettres)
- les adresses des différentes procédures effectuant les liaisons avec le matériel.

3.6.4. Création et destruction des objets

La création d'un fichier se fait par l'adjonction de son nom dans un répertoire.
La création d'une boîte aux lettres se fait par la préparation d'une table contenant son nom ainsi que l'adresse du début d'une liste de messages.
La création d'un périphérique consiste à préparer cette table et à charger le driver ou à faire la connexion avec ce dernier.
En principe, les périphériques sont créés à l'initialisation du système.

3.6.5. Ouverture et fermeture des objets

La procédure qui ouvre un fichier, un périphérique ou une boîte aux lettres s'assure que cette ouverture correspond aux caractéristiques que l'appelant a défini dans ses paramètres.

La procédure qui ferme un objet s'assure que tous les tampons ont été vidés avant de libérer le canal ou de donner le contrôle à un processus éventuellement en attente.

3.6.6. Ordres primitifs disponibles

Les ordres primitifs permettant la manipulation de fichiers dans la station de ressources sont mentionnés ci-dessous.

CREATE	pour créer un nouveau fichier
DELETE	pour effacer un fichier
RENAME	pour changer le nom d'un fichier
OPEN	pour ouvrir un fichier
CLOSE	pour fermer un fichier
RESET	pour fermer tous les fichiers que cette station a ouverts
RDBYTE	pour lire un certain nombre de bytes dans un fichier
RDL	pour lire une ligne dans un fichier
WRBYTE	pour écrire un certain nombre de bytes dans un fichier
GTOD	pour obtenir l'heure

GDAY	pour obtenir la date.

Le tableau suivant mentionne les paramètres d'entrée et de sortie des procédures.

APPEL	PARAMETRES D'ENTREE	PARAMETRES DE SORTIE
CREATE	nom de fichier	
DELETE	nom de fichier	
RENAME	ancien nom, nouveau nom	
OPEN	nom de fichier	canal
CLOSE	canal	
RESET		
RDBYTE	canal, nombre, zone mémoire	nombre
RDL	canal, zone mémoire	nombre
WRBYTE	canal, nombre, zone mémoire	nombre
WRL	canal, zone mémoire	nombre
GTOD		heures, minutes, secondes
GDAY		jour, mois, année

L'utilisateur peut à partir de ses programmes demander l'exécution de ces ordres dans la station des ressources en faisant des appels standardisés au système d'exploitation.
Les routines correspondant à ces appels envoient l'ordre à la station de ressources qui l'exécute et renvoie ensuite une réponse.

3.6.7. Principales fonctions du multiplexeur

Les programmes du multiplexeur sont supportés par un système de programmation parallèle, multiprocessus, écrits en assembleur CALM Z80. Deux sortes d'ordres sont implémentés:

- des ordres simples gérant des périphériques.
  Ils se subdivisent en deux groupes:

  - des ordres purement logiciels : CREATE, OPEN et CLOSE.
    Les ordres ne dépendant pas du temps de réaction des interfaces, ils sont donc traités séquentiellement de façon synchrone et la réponse fournie immédiatement.

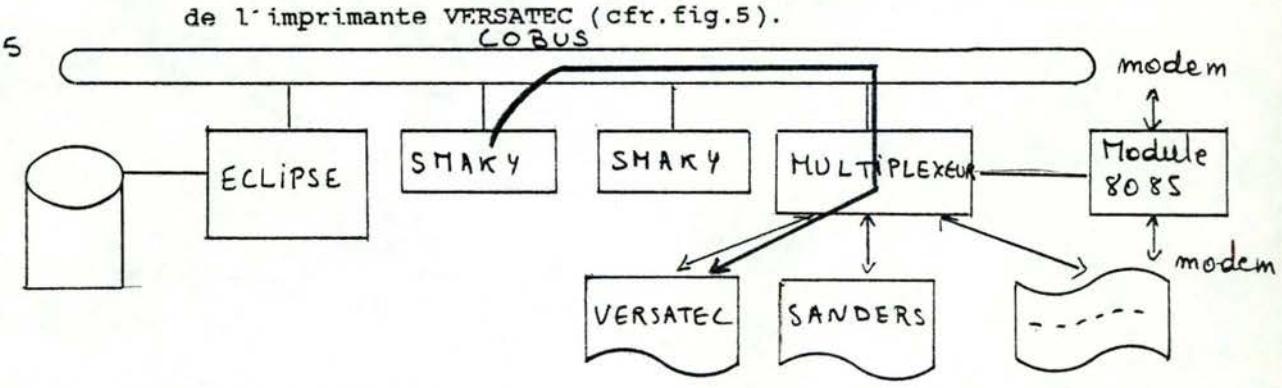
  - des ordres mettant en oeuvre le matériel READ et WRITE.
    Ceux-ci sont réalisés de façon asynchrone. Ils sont traités par interruptions et leur exécution est supportée par un processus.
    Il y a donc un processus affecté à chaque périphérique.

- des ordres programmés déclenchant des processus logés dans le multiplexeur.
  Ceux-ci utilisent les ordres simples et des appels gérant la mémoire de masse reliée à cobus. Ils sont envoyés sous forme de chaîne de caractères ASCII, ceci en vue de standardiser les ordres qui seraient traités dans différentes sortes de stations.

Les divers transferts d'information possibles sont mentionnés ci-dessous:

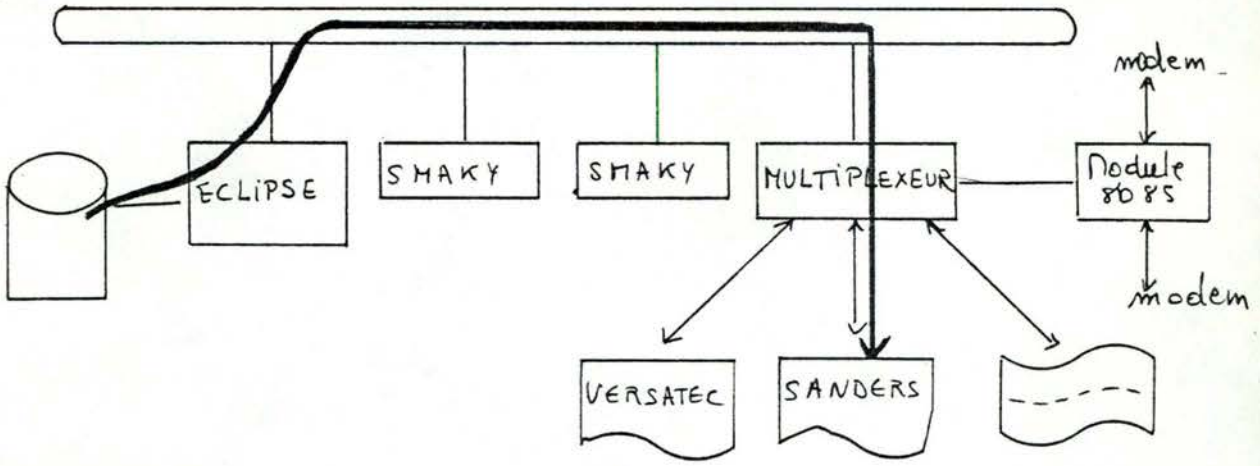
- Transfert d'information à partir d'un SMAKY à destination de l'imprimante VERSATEC (cfr.fig.5).

Fig.5

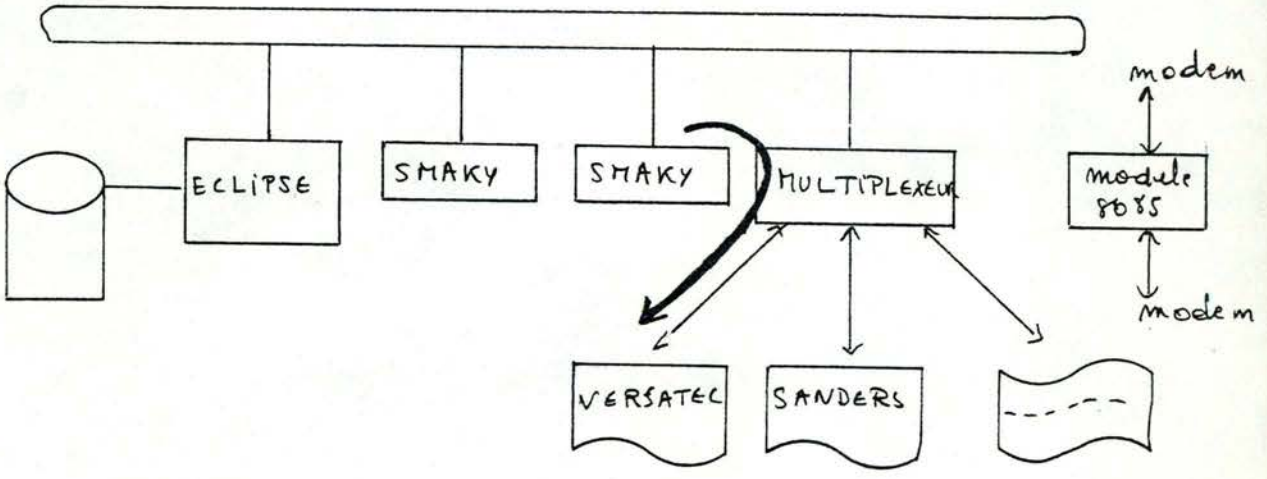


- Un SMAKY donne l'ordre au multiplexeur de transférer un fichier de la mémoire de masse sur l'imprimante SANDERS (cfr.fig.6)

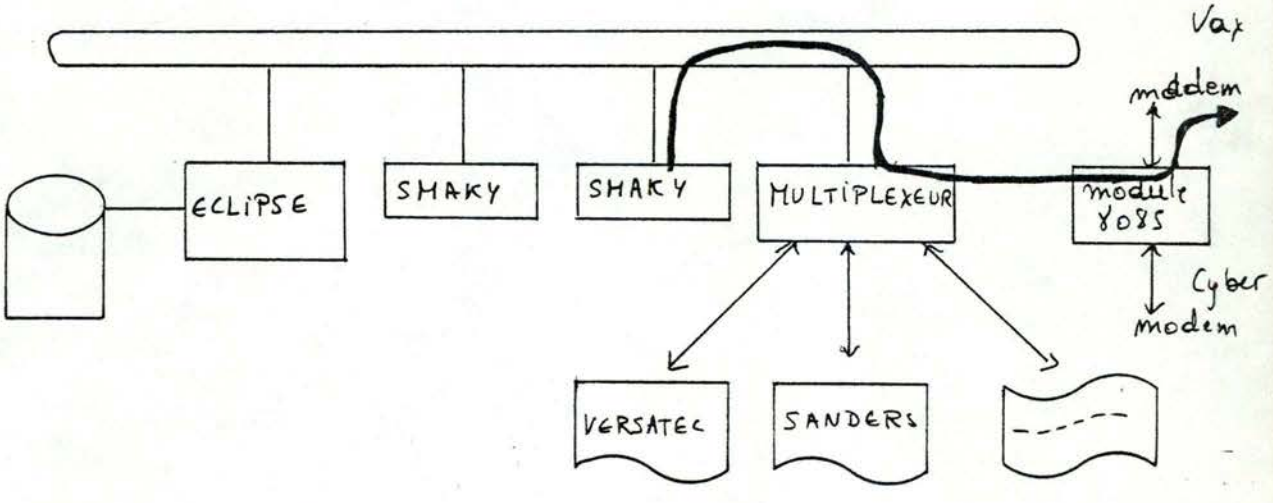
Fig. 6



- Un ordinateur extérieur au réseau envoie un fichier sur la VERSATEC par voie série (cfr.fig7).



- Une liaison point à point est établie entre un SMAKY et un MODEM aboutissant à un gros ordinateur. Le SMAKY apparaît donc comme s'il était un terminal connecté à l'ordinateur (cfr.fig.8).



CHAPITRE 4

SPECIFICATIONS DE L'APPLICATION "PETIT-PAS"

4.1. CONSIDERATIONS D'ORDRE METHODOLOGIQUE

Les spécifications fonctionnelles sur lesquelles s'appuie l'implémentation de l'application à réaliser sont le fruit d'une démarche procédant par raffinements successifs. [3]
On distingue deux composantes fondamentales dans la structure fonctionnelle proposée: la structure fonctionnelle des données et celle des traitements. Exposons brièvement les points importants de ces deux structures; la description se fera dans les deux cas à partir des éléments teminaux de la hiérarchie.

4.1.1. Structure des données

4.1.1.1. Nomenclature des éléments

Les éléments de la structure logique des données répondent aux définitions suivantes.

- Au plus bas niveau de la hiérarchie de la structure des données, on distingue les DONNEES ELEMENTAIRES.
  Celles-ci peuvent être considérées comme des éléments atomiques dans la mesure où, du point de vue de l'analyse, il n'existe pas de constituant plus élémentaire.
  Une donnée élémentaire n'a pas d'existence intrinsèque, elle dépend de l'usage qui en est fait au sein de l'organisation.

- Une INFORMATION ELEMENTAIRE
  définit la configuration de données élémentaires, qui désigne une propriété d'usage au sein de l'organisation associée à un objet au cours d'une période
  de temps et en un lieu défini.
  Les informations élémentaires sont regroupées dans ce qu'il convient d'appeler un DICTIONNAIRE DES INFORMATIONS ELEMENTAIRES.
  Le dictionnaire reprend pour chaque information élémentaire un ensemble de caractéristiques telles que
  - son nom, sa définition
  - les désignations équivalentes au sein de l'entreprise

- les types de données
- les contraintes d'intégrité associées aux données et aux informations élémentaires
- la référence aux unités d'information, structures logiques et fonctionnelles relatives à cette information élémentaire.

- On appellera UNITE D'INFORMATION le regroupement d'informations élémentaires relatives à un même objet et utilisées par un traitement logique homogène.

- Une STRUCTURE LOGIQUE est l'ensemble structuré des unités d'information dont on a besoin pour la réalisation d'un traitement homogène.

- La STRUCTURE CONCEPTUELLE DES DONNEES intègre les structures logiques particulières associées aux traitements logiques qui réalisent les applications définies dans un projet cadre.

4.1.1.2. Mode de représentation

Il existe de nombreux modèles de représentation de structures de données. Celui qui retient notre attention est connu sous le nom de modèle ENTITE/ASSOCIATION.
Il met en jeu les concepts suivants:

ENTITE:

> "Une entité est ce qu'un individu ou groupe voit comme un tout, ayant une existence propre. Une entité est caractérisée par un ensemble de propriétés quantitatives et qualitatives et un comportement permanent. Le nombre de propriétés ainsi que la permanence sont fonction du point de vue choisi."

ASSOCIATION:

> "Une association est un ensemble de deux ou plusieurs entités où chacune assume un rôle donné. Une association peut posséder plusieurs propriétés. L'existence d'une association est contingente à l'existence des entités qu'elle relate.

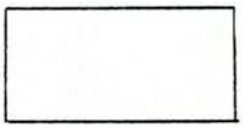
PROPRIETE-VALEUR:

> "Une propriété appartenant à une entité ou à une association une qualité que les individus attribuent à cette entité ou cette association. L'existence des propriétés

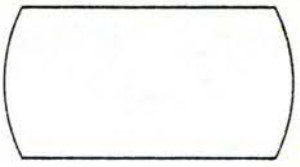
attribuées est contingente à l'existence des entités ou de l'association concernée."

Le modèle ci-dessus donne lieu au graphisme suivant.

Un type d'entité est représenté par un rectangle.



Un type d'association est représenté par la figure suivante.



On complètera cette représentation en mentionnant les propriétés qui caractérisent une entité ou une association .

4.1.2. Structure des traitements

La structure des traitements comporte les éléments suivants:

- Une FONCTION correspond au niveau d'individualisation élémentaire considéré du point de vue de l'analyse fonctionnelle.
  Remarquons que ce niveau est arbitraire et reflète le degré de modularité choisi par la programmation.

- Une PHASE est un traitement possédant une unité spatio-temporelle d'exécution.
  Elle est exécutée dans le contexte d'une cellule d'activité. Entendons par là un centre d'activité homogène dans le temps et l'espace doté de ressources et pourvu de règles de comportement nécessaires à son fonctionnement.

- Une APPLICATION décrit l'enchaînement des phases relatives à un flux d'information, flux homogène et permanent dans le temps.

- Un SOUS-SYSTEME regroupe des applications non-indépendantes dans le cadre des interactions " exécution-gestion"; il regroupe les applications d'exécution (appartenant au flux) et les activités de gestion (appartenant au flux de décision) qui définissent le comportement des premières.
  Le lien entre les applications d'un sous-projet est constitué par des informations situationelles (fichiers permanents) communes à ces différentes applications.

- Le SYSTEME INFORMATIQUE est constitué de l'ensemble des flux et fichiers d'information ainsi que des processus de traitement de l'information décrivant le fonctionnement et l'état global de l'organisme.

On peut établir une correspondance entre la structure des données et la structure des traitements.

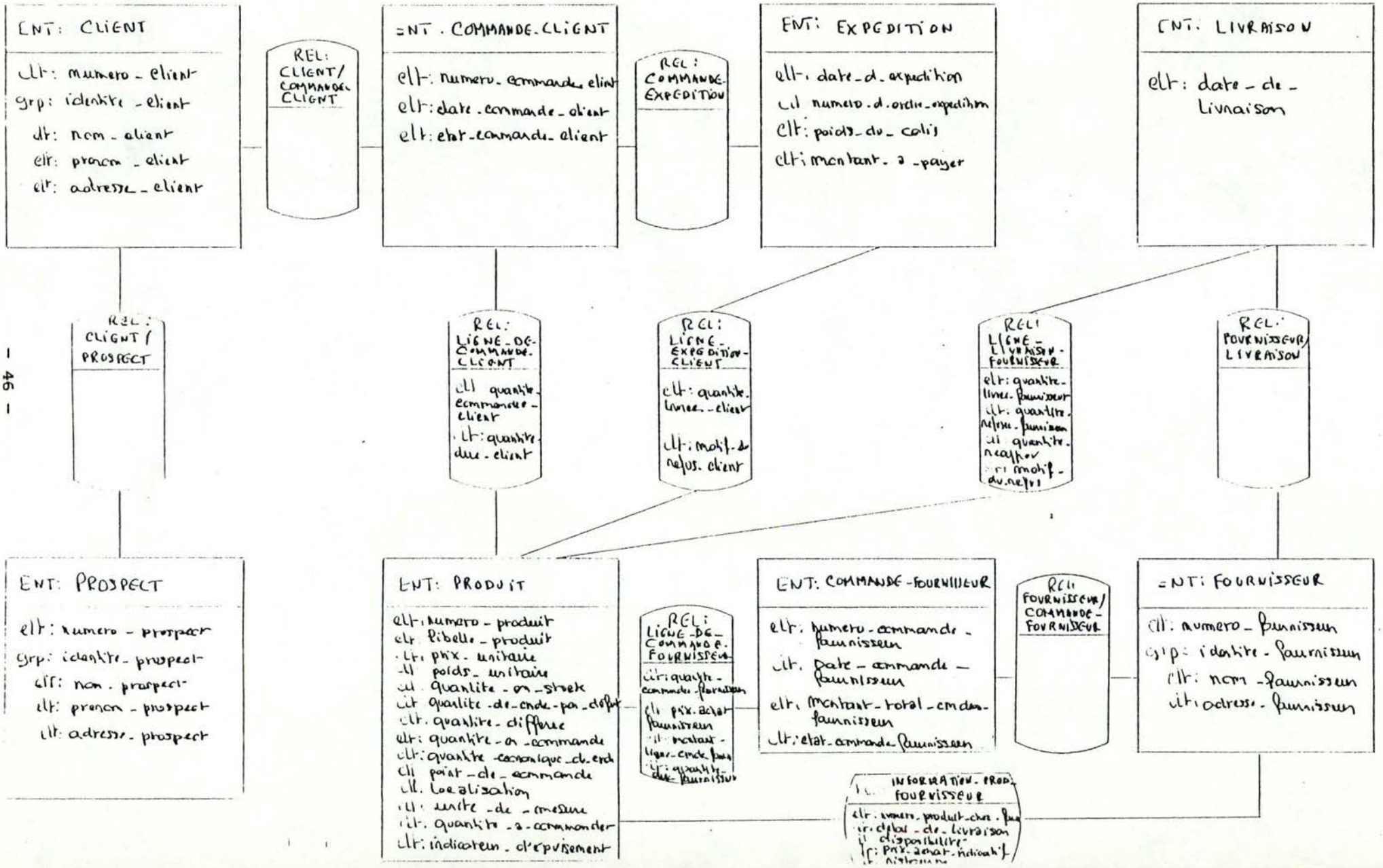
- une application est définie au niveau de la structure fonctionnelle des données.
- Une phase travaille sur une structure logique de données.
- Les unités d'informations constituent des unités d'accès pour les fonctions qui travaillent sur cette structure.

4.2. ACQUIT DE L'ANALYSE FONCTIONNELLE

4.2.1. Modèle conceptuel global

La figure 1 représente le modèle conceptuel relatif à l'application entière. La structure logique associée à chacune des phases se trouve en annexe.

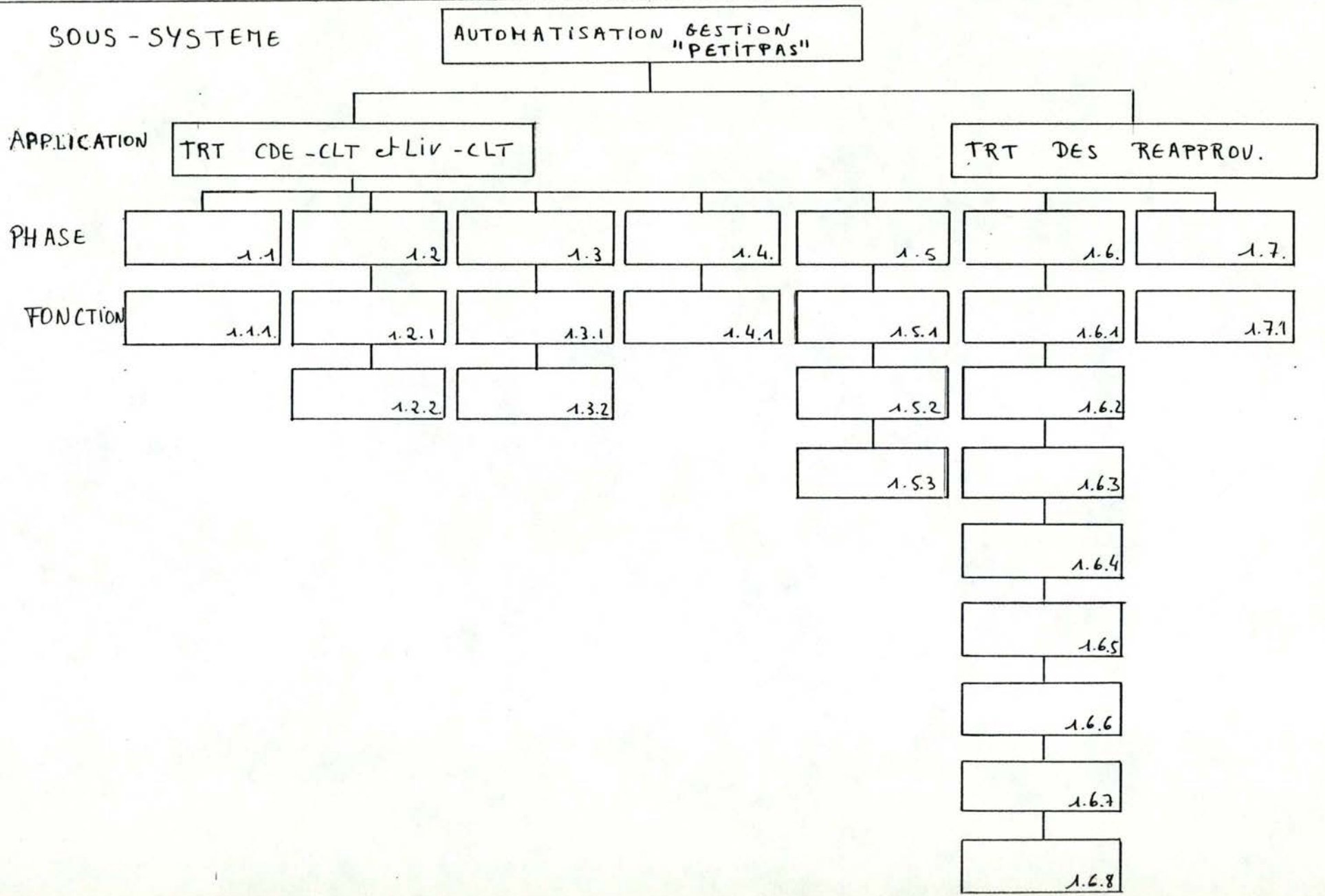
MODELE CONCEPTUEL



4.2.2. Structure des traitements

4.2.2.1. Aspect statique

La description suivante des traitements ne tient compte que de l'aspect statique. Elle est conforme à la découpe suggérée au paragraphe 4.1.2. . La situation globale est schématisée fig. 2


SOUS - SYSTEME
AUTOMATISATION "GESTION "PETITPAS"
APPLICATION
TRT CDE -CLT et LIV -CLT
TRT DES REAPPROV.
PHASE
1.1
1.2
1.3
1.4.
1.5
1.6.
1.7.
FONCTION
1.1.1.
1.2.1
1.2.2.
1.3.1
1.3.2
1.4.1
1.5.1
1.5.2
1.5.3
1.6.1
1.6.2
1.6.3
1.6.4
1.6.5
1.6.6
1.6.7
1.6.8
1.7.1

PHASE 1.1. Préparation-bon de commande client

fonction 1.1.1. Vérification signature

message E : Bon-de-commande-client
message S : Bon-de-commande-client-signé
traitement : Vérifier si le bon de commande client reçu est signé ou non
S'il l'est, un message est généré (Bon-de- commande-client-signé) à destination de la fonction 1.2.1.
S'il ne l'est pas, un message est généré (Bon-de-commande-client-non signé) à destination de la fonction 1.3.2.

PHASE 1.2. Enregistrement-bon-commande-client

fonction 1.2.1. Vérification-identité-client

message E : Bon-de-commande-client-signé
message S : Bon-de-commande-client-signé-accepté
Bon-de-commande-signé-refusé
traitement: Le traitement effectué ici a pour objet de mettre à jour le fichier des clients en fonction des informations figurant sur le bon de commande du client.

fonction 1.2.2. Vérification-corps-commande

message E : Bon-de-commande-client-signé-accepté
message S : Commande-candidate-à-mettre-à-jour
Bon-de-commande-client-signé-accepté-non validé
traitement: Le traitement effectué a pour objet de vérifier si les informations constituant le corps de la commande sont correctes et d'y apporter des rectifications le cas échéant.
Si la commande est correcte ou peut être corrigée: un message (commande-candidate-à-mettre à jour) est généré à destination de la fonction 1.4.1.
Si la commande n'est pas correcte et ne peut être rectifiée, un message (Bon-commande-client-signé-accepté-non-valide) est généré à destination de la fonction 1.3.1.

PHASE 1.3. Traitement des cas litigieux

fonction 1.3.1. Correction-bon-de-commande

message E : Bon-de-commande-client-signé-refusé
Bon-de-commande-client-accepté-non valide
message S : Bon-de-commande-client-signé
Bon-de-commande-client-à-renvoyer
traitement: Les bons de commande arrivant à cette cellule font l'objet d'un examen qui a pour but de permettre leur éventuel recyclage.
Si le bon de commande peut être corrigé: un message est généré (Bon-de-commande-client-signé) à

destination de la fonction 1.2.1.
Dans le cas contraire: un message est gènerè (Bon-de-commande- client-à-renvoyer) à destination de la fonction 1.3.2.

fonction 1.3.2.

message E : Bon-de-commande-client-non-signè
Bon-de-commande-à-renvoyer
message S : justificatif-de-refus
traitement: Renvoyer au client
son bon de commande
un justificatif de refus
un bon de commande vierge

PHASE 1.4. Mise-à-jour-moins-du-stock

fonction 1.4.1.

message E : Commande-candidate-à-mettre-à-jour
message S : Commande-client-à-livrer
traitement: Le but de ce traitement est de livrer au client dans la mesure du possible l'ensemble des produits qu'il a commandès.
Ce traitement est relatif à une commande qui vient d'être enregistrèe ou à une commande diffèrèe sèlectionnèe suite à la rentrèe en stock d'un produit.

PHASE 1.5.Ordonnancement-parcours-rayon

fonction 1.5.1. Constitution d'une sèrie

message E : Commande-client-à-livrer
message S : Liste-commandes-à-ordonnancer
traitement: Le but de ce traitement est l'accumulation de cent commandes diffèrentes. Une fois cette sèrie constituèe, le message (liste-commandes-à- ordonnancer) est gènèrè à destination de la fonction 1.5.2.

fonction 1.5.2. Ordonnancement

message E : Liste-commandes-à-ordonnancer
message S : Bon-par-casier
Bon-par-sèrie
traitement: Ce traitement a pour objet d'optimiser le parcours du magasinier dans les rayons.
A cet effet, il gènère une liste (bon par sèrie) reprenant les produits à prendre dans l'ordre du prèlèvement ainsi que la quantitè à dèposer dans chaque casier. Un second bordereau est gènèrè qui spècifie par casier les numèros des dix commandes

qui s'y trouveront regroupées.

fonction 1.5.3. Déstockage

message E : Bon-par-série
traitement: en suivant les rayons dans l'ordre spécifié par le bon par série, le magasinier prélève dans les rayons les quantités inscrites sur son bordereau et les dépose dans les casiers prévus.
Au terme de ce processus, le magasinier a donc rempli les dix casiers avec les marchandises correspondant à cent commandes.

PHASE 1.6. Constitution du colis

fonction 1.6.1. Affichage-expédition

message E : Bon-par-casier
traitement: Afficher à l'écran l'image d'un bon de livraison

fonction 1.6.2. Reconstitution-commande

message E : Bon-par-casier
message S : Bon-par-casier-annoté
traitement: Le magasinier reconstitue le colis en puisant dans le casier et en suivant à l'écran les quantités à prélever pour chaque produit. Si le casier contient suffisamment de chaque produit pour reconstituer le colis, le magasinier ne passe aucune marque sur le bon par casier.
Si le casier ne contient pas suffisamment de chaque produit pour reconstituer le colis, le magasinier abandonne la commande, met les marchandises en attente et porte une marque en regard de la commande incomplète sur le Bon-par-casier. Ce bon-par-casier annoté est destiné à la fonction 1.6.4.

fonction 1.6.3. Facturation

message S : Bon-de-livraison-client-ar
Etiquette-adresse
Bordereau-sncb-ar
Bordereau-poste-ar
traitement: Lorsque le colis correspondant à une commande est reconstitué, on déclenche la facturation dont l'objet est d'éditer l'ensemble des documents d'accompagnement de la commande.

fonction 1.6.4. Parcours-correction

message E : Bon-par-casier-annoté

traitement: D'une part remettre en magasin les marchandises excédentaires dans un casier. D'autre part, retirer si possible les marchandises manquantes dans un casier.

fonction 1.6.5. Affichage-expédition-correction

message E : Bon-par-casier-annoté
traitement: idem fonction 1.6.1.

fonction 1.6.6. Reconstitution-commande-correction

message E : Bon-par-casier-annoté
traitement: Le manutentionnaire, en suivant à l'écran et en puisant dans le casier reconstitue la commande.
Si celle-ci est complète: tant mieux pour le client
Sinon: il décide de l'expédier telle qu'elle.

fonction 1.6.7. Correction-du-système

traitement: Le traitement ici réalisé a pour objectif de mettre à jour le système pour qu'il représente au maximum la réalité (c'est-à-dire de rectifier une éventuelle différence entre stock-papier et stock-ordinateur)

fonction 1.6.8. Emballage-et-expédition

message E : Bon-de-livraison-client-ar
Bordereau-sncb-ar
Bordereau-poste-ar
Etiquette-adresse
message S : Bon-de-livraison-client
Bordereau-sncb
Bordereau-poste
traitement: Emballer un colis avec son bon de livraison, y coller l'étiquette adresse et y joindre le bordereau d'accompagnement.

PHASE 1.7. Sélection-commandes-différées

fonction 1.7.1. Sélection-commandes-différées

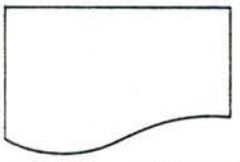
message E : Produit-rentré-en-stock
message S : Commande-candidate-à-mettre-à-jour
traitement: Le traitement a pour but de sélectionner les commandes différées pour lesquelles un produit est rentré en stock.

4.2.2.2. Aspect dynamique

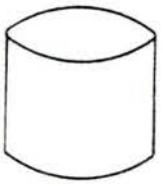
La description de l'aspect dynamique des traitements est réalisé dans [29] conformément aux principes exposés dans [4].

4.2.2.3. Flux d'information

Le flux d'information (cfr.annexe B) donne une description spatiale et temporelle des points de naissance, de transformation et de disparition des informations dans le système.
Les informations circulantes sont schématisées comme suit:



Les ensembles d'information sont schématisés comme suit:



4.3. MODIFICATION APPORTEE

Il n'entre pas dans le cadre de ce travail de remettre en question les spécifications fonctionnelles proposées dans [26]. Nous suggérons toutefois, dans une optique simplificatrice, la modification suivante du statut de l'entité EXPEDITION.

Nous proposons de considérer l'entité EXPEDITION comme étant le pendant de l'entité COMMANDE. Une fois enregistrée dans le système l'entité COMMANDE verra ses éléments varier en fonction de l'activation des divers processus. A cet effet, elle se trouve affectée d'un statut (dont la signification précise se trouve dans [26]). Celui-çi donne une vision de l'état courant de la commande dans le système depuis son apparition jusqu'à sa disparition. Les informations reprises sur la commandes témoignent des quantités commandées par le client mais restant à livrer eu égard au dernier examen des stocks. De façon analogue, les informations consignées sur l'expédition re représentent les quantités que l'entreprise peut livrer au client conformément au dernier examen des stocks. On modifie ainsi le statut de l'entité EXPEDITION en ce sens que maintenant on aura une association bi-univoque entre l'entité COMMANDE et l'entité EXPEDITION. Une fois qu'une série a été constituée, cela matérialise l'intention au'a la firme d'effectuer réellement la livraison. Dans ce cas, on peut considérer que les quantités figurant sur l'expédition ont été satisfaites et donc les "supprimer". Le fichier des BONS DE LIVRAISON constituent un état transitoire qui permet un éventuel retour en arrière

consécutivement à une opération de rectification du stock.

TROISIEME PARTIE

REALISATION DU PROJET

CHAPITRE 5

ARCHITECTURES ORGANIQUE ET D'IMPLEMENTATION

5.1. ARCHITECTURE ORGANIQUE

L'architecture organique sur laquelle se base l'implémentation de l'application s'inspire très fortement de celle proposée en annexe B et représentée fig.1. Relevons ci-dessous quelques caractéristiques qui lui sont propres.

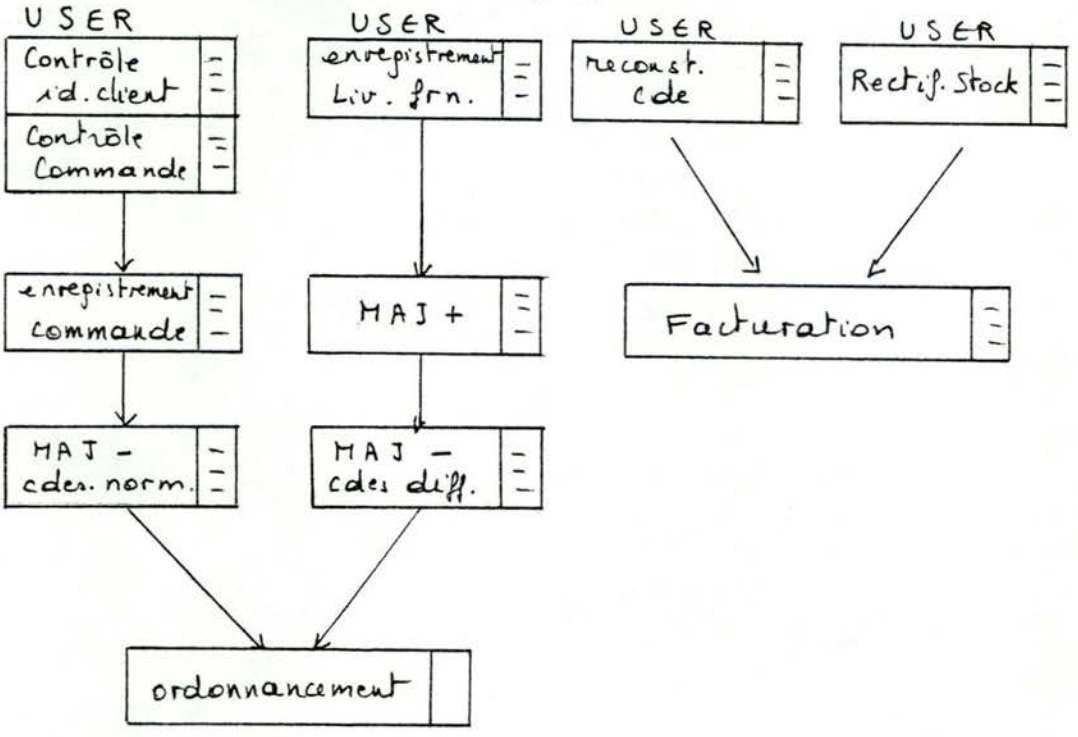


Fig. 1

- Un premier point important spécifique à cette organisation de programmes réside dans le fait qu'elle respecte la fonction en fonction-utilisateur (dégagée dans le cadre de l'analyse fonctionnelle) en associant un programme à chacune des fonctions.
- Cette façon de procéder a l'avantage d'assurer un certain degré de transparence par rapport au matériel qui devra supporter l'application et par ailleurs, permet un maximum de parallélisme entre les processus.
- L'aspect utilisation de la structure de données par les traitements est pris en compte par le fait que, en regard de chaque processus, sont portés les accès aux données requis dans le cadre de l'exécution.

- La différence fondamentale que présente cette architecture par rapport à celle figurant en annexe réside dans le fait qu'elle ne fait pas intervenir de module coordinateur en vue de gérer les priorités imposées par les spécifications fonctionnelles de base. Ce point particulier fera l'objet d'un examen détaillé (Cfr.parag.5.2.4.).

Attirons dès à présent l'attention sur le fait que cette décision a été réellement prise au niveau de l'implémentation de l'application. Le matériel utilisé est en effet déterminant dans le choix de la solution retenue.

5.2. ARCHITECTURE D'IMPLEMENTATION

Entendons par architecture d'implémentation la concrétisation de l'architecture organique en tenant compte de l'aspect matériel. Il s'agira donc, dans l'optique qui nous concerne, à savoir l'optique réseau, de considérer deux composantes fondamentales au problème: l'affectation des traitements aux divers processeurs et la répartition des données sur le réseau.

5.2.1. Remarque préliminaire

Il est intéressant de remarquer que la configuration générale du réseau local COBUS influe fortement sur la façon dont devra être implémentée l'application réseau est la mise à disposition des divers utilisateurs des ressources coûteuses telles que notamment les mémoires de masse (Cfr.parag.3.3.1.). La décision d'adjoindre un micro-disque de type WINCHESTER rencontre ces préoccupations. Les primitives de gestion des fichiers localisés sur cette unité se doivent donc d'être générales. Cette situation a donc des conséquences à deux niveaux: celui de la répartition des traitements et celui de la répartition des données.

1. Conséquence au niveau des traitements

La réalisation d'un projet quel qu'il soit, doit être compatible avec la philosophie de l'architecture du réseau évoquée ci-dessus. A cet effet, il n'est pas question d'introduire au niveau de la station gérant les fichiers résidant sur le micro-disque du logiciel spécifique à une application. Nous verrons ultérieurement que cette considération sera d'importance dans le problème des priorités spécifique au cas" PETITPAS".

2. Conséquence au niveau des données

Il est naturel, dans le contexte dans lequel on travaille, de regrouper une part importante des données manipulées sur le micro-disque.

Diverses raisons influent dans cette direction.

- Le fait qu'un fichier soit partageable entre plusieurs utilisateurs justifie sa présence sur le micro-disque. L'ensemble des requêtes émises par les divers utilisateurs en vue d'une quelconque opération sur le fichier seront traitées séquentiellement. On ne travaille pas dans le contexte particulier des bases de données réparties qui supposent la présence de copies multiples et, par conséquent, des problèmes complexes induits par leurs mises à jour.
- Le caractère volumineux d'un fichier est également un aspect influant dans ce sens.
- Un élément non négligeable est également le développement de primitives d'accès direct aux enregistrements de ces fichiers. (Cfr.parag.6.2.)

5.2.2. Affectation des traitements aux processeurs

Une première façon d'envisager l'affectation des processus aux divers processeurs est schématisée fig.2

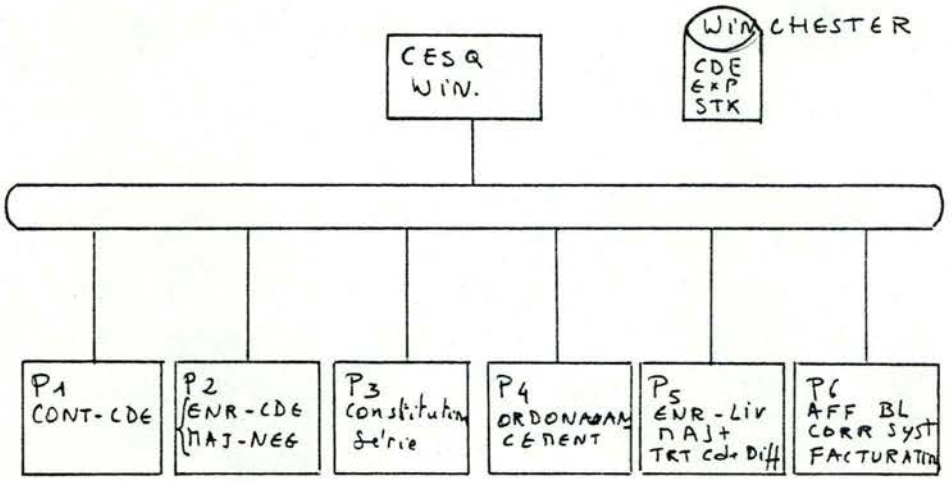


Fig. 2

- Le processeur P1 est spécialisé dans le contrôle de validité de la commande client c'est-à-dire qu'il a pour objet de vérifier si celle-ci est syntaxiquement correcte et recevable. Les vérifications sont essentiellement de deux ordres:

- un premier contrôle est relatif à l'ensemble des propriétés identifiant le client c'est-à-dire son nom, son adresse,...
En cas de divergence avec les informations consignées dans le fichier client, celui-ci est mis à jour.
- un second contrôle porte sur l'ensemble du corps de la commande et consiste essentiellement à vérifier si les références des produits commandés sont correctes ainsi que les prix mentionnés par le client.

- Le processeur P2 assure l'enregistrement d'une commande sur laquelle ont été effectués avec succès l'ensemble des contrôles de validité. Il répercute également dans le stock l'effet consécutif à la prise en considération de la commande par le système.

- Le processeur P3 a pour seule et unique fonction de constituer une série de cent commandes différentes et de déclencher en conséquence le processus d'ordonnancement portant sur la dite série.

- Le processeur P4 supporte l'exécution d'un programme d'ordonnancement qui a pour objet d' optimiser le parcours en magasin du magasinier chargé de constituer l'ensemble des colis relatifs à une série de cent commandes.

- Le processeur P5 assure l'exécution du programme dont le but est d'enregistrer une livraison fournisseur en la répercutant dans l'état des stocks .
Ce processeur effectue également le traitement des commandes différées sélectionnées suite à l'enregistrement de rentrées de produits en stock.

- Le processeur P6 permet d'afficher des bons-de-livraison et de déclencher une éventuelle phase de correction du stock afin que le "stock ordinateur" coincide avec le "stock physique" réel. La facturation s'effectue également sur ce processeur.

Examinons ci-dessous les principales raisons qui ont présidé à une telle répartition des traitements.

Remarquons dès à présent qu'une telle organisation présente un gros défaut: une sous-utilisation du matériel.
Ce sont des considérations d'ordre technologique et" software" qui imposent cet état de choses.

Notons qu'il est , en effet, impossible d'affecter le traitement ‹constitution d'une série› au processeur P1 ou P4 du simple fait que le système n'est pas interruptible. Ce processus est en fait susceptible d'être activé à partir de deux sources différentes: l'enregistrement d'une commande que l'on vient de recevoir et le traitement d'une commande différée. Par conséquent, si on l'associe au processeur P1, celui-ci risque en cours d'exécution de devoir accepter une requête de la part du

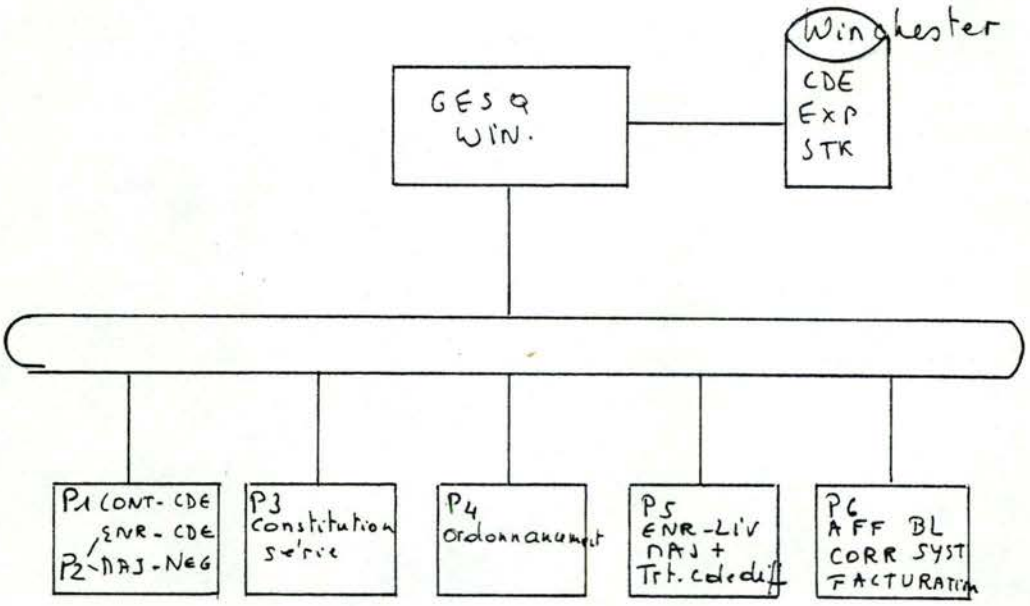
processeur P5 suite à la sélection et au traitement d'une commande différée.
Si on décide de l'associer au processeur P4, le problème est analogue par rapport au déroulement du programme d'ordonnancement.

La raison pour laquelle le programme ‹contrôle-commande› peut être séparé de l'exécution de celui de l' ‹enregistrement-commande› et ‹MAJ-› est la suivante.
Il convient de remarquer qu'une commande ne peut être acceptée par le système, c'est-à-dire enregistrée, que si elle est entièrement correcte. Au niveau fichier, une commande est relative à plusieurs enregistrements physiquement différents; un pour l'entête et un pour chaque ligne de la commande.
En outre, l'enregistrement d'une commande implique , dans le cas général, pour chaque ligne une mise à jour du fichier expédition et une mise à jour du fichier stock soit au total trois accès disque.
Ainsi, par exemple, l'enregistrement d'une commande de dix lignes se traduira par trente accès disque.
Cette situation risque de donner un mauvais temps de réponse entre la demande d'acceptation d'une commande et son enregistrement.
Mais, une exigence de ce type est-elle réaliste?
Cela dépend en fait du taux de commandes à enregistrer par jour ainsi que du nombre moyen de lignes par commande.

Une raison tend à regrouper sur un même processeur les fonctions de contrôle et d'enregistrement de la commande.
Remarquons, en effet, que si l'on envisage de prévoir plusieurs postes de contrôle des commandes et que l'on décide de scinder en deux les opérations de vérification et celle d'enregistrement proprement dit, un problème apparaît relevant de la non interruptibilité du système. En effet, le processus P1 communique avec le processeur P2 par l'envoi de la totalité d'une commande. Dès lors, si on multipliait les processeurs de type P2, cela augmenterait les communications envers le processeur P1 qui, en cours d'exécution, devrait accepter et gérer correctement les requêtes émanant des processeurs de type P2, ce qui est actuellement impossible.

Attirons enfin l'attention sur le fait que les spécifications prévoient la présence de deux postes de contrôle des commandes.

Sur base de ces considérations, on préferera donc, à la première organisation proposée, celle représentée fig.3



Cette structuration a l'avantage d'utiliser de manière plus satisfaisante le matériel informatique , mais a pour inconvénient une dégradation du temps de réponse.

5.2.3. Répartition des données

Pour les raisons précisées plus haut, la majorité des données utiles à l'application se trouveront concentrées sur le micro-disque. Ce caractère relativement centralisé des informations sur le réseau est donc en quelque sorte imposé.
Remarquons par ailleurs, que si l'on se réfère à la nomenclature proposée des éléments d'un système informatique, on se situe au niveau de l'application et que les différentes phases et fonctions qui la composent manipulent une structure logique commune . Il n'apparaît pas, dans ces conditions, d'un grand intérêt de décentraliser davantage ces informations.

Un seul effort de répartition s'avère être intéressant qui consiste à isoler les fichiers SIGNALETIQUE CLIENT et NOMENCLATURE PRODUIT qui sont seulement utilisés par les processus opérant des contrôles de validité sur les commandes.

Cela ne va toutefois pas sans poser certains problèmes.
En effet, dans la situation qui nous préoccupe, isoler ces fichiers revient à les placer sur disquettes, seuls autres supports envisageables pour contenir des informations.
Les problèmes soulevés sont de deux ordres:

- d'une part, le volume des deux fichiers concernés risque d'être un obstacle à leur localisation sur une disquette dont la capacité reste malgré tout limitée.
- d'autre part, si on envisage que plusieurs processeurs soient assignés au contrôle des commandes, ceux-ci ne peuvent travailler sur un même ensemble de clients car les mises à jour faites par l'un ne pourraient se répercuter facilement chez les autres sans poser des problèmes relativement complexes.

Une solution envisageable allant dans ce sens pourrait être dictée par l'organisation de l'entreprise.
Supposons que nous ayons affaire à une entreprise d'importance moyenne (et ce , en vue de satisfaire aux conditions relatives à la taille des fichiers) et que cette firme travaille par catégories de clients. On peut dès lors de façon réaliste imaginer une classification des clients sur base d'un facteur géographique, linguistique...
Pour ne citer qu'un exemple, on procédera à une telle décomposition dans une compagnie d'assurance en classant les clients d'après le type de police d'assurance dont ils sont détenteurs et qui est fonction du risque couvert (incendie, vol, ...)
En procédant de la sorte, on s'assure donc que les divers processeurs travaillent chacun sur des sous-ensembles distincts du fichier SIGNALETIQUE CLIENT global et qu'il n'y a donc aucun risque d'interférence. De façon plus générale, et en débordant du contexte particulier d'un réseau local, on adoptera ce mode d'organisation lorsqu'une compagnie possède des sièges disperses géographiquement.
Un exemple caractéristique relève du domaine bancaire où chaque agence manipule des informations qui sont essentiellement relatives à ses propres clients et ne communique pas de manière permanente avec le système central.

Dans le cadre particulier de l'application" PETITPAS", placer les deux fichiers SIGNALETIQUE CLIENT et NOMENCLATURE PRODUITS au niveau du processeur de contrôle de validité, possède l'avantage de rendre celui-ci quasiment autonome du réseau.
On pourra par exemple tirer parti de cette indépendance en autorisant un mode d' exploitation" dégradé". Entendons par là, le fait que, consécutivement à un incident paralysant le réseau ou une partie des équipements nécessaires, il sera possible de continuer la vérification des commandes et d'en assurer un stockage intermédiaire en vue de la réinjecter ultérieurement dans le système.

5.2.4. Problème spécifique des priorités

Pour bien comprendre les raisons qui ont motivé le choix retenu en matière de gestion des priorités, il convient en tout premier lieu, d'exposer concrètement quelles étaient les contraintes de priorité à respecter, et d'examiner quelle était

leur incidence quant au comportement global du système.
Les priorités à respecter sont les suivantes:

1. La rectification du stock doit être prioritaire par rapport à toute opération sur le stock.
2. La ‹mise à jour positive› du stock doit être prioritaire par rapport à la ‹mise à jour négative› du stock consécutive à la sélection d'une commande différée.
3. La ‹mise à jour› du stock relative à une commande différée doit être prioritaire par rapport à la même mise à jour effectuée suite à l'enregistrement d'une commande normale.

Pour bien dégager l'importance relative de ces trois contraintes, raisonnons par l'absurde et examinons quelles seraient les conséquences de leur non-respect.

A.-Si on ne tient pas compte de la première contrainte, les diverses situations suivantes sont possibles:

MAJ + avant une correction. Cela aura comme conséquence une surestimation du stock; une divergence sera de toute façon remarquée lorsqu'une commande aura été satisfaite à tort par le système qui se base sur le" stock ordinateur". La rectification se faisant maintenant de façon relative, cette situation ne pose aucun problème et ne diffère pas fondamentalement de celle où la correction du système aurait été faite avant. L'inconsistance de stock aurait seulement été détectée un peu plus tôt, mais ça n'aurait aucun impact grave au niveau du traitement des commandes.

MAJ - avant une correction

Une commande enregistrée provoquera une mise à jour moins du stock et la réquisition de tout ou partie de marchandises qui , en fait, n'existent pas. Cet état de choses sera détecté lors de l'affichage du bon de livraison.

MAJ - suite à un différé avant correction

Situation analogue à celle décrite ci-dessus.

Ce qu'il faut en fait se demander, c'est si ce problème n'a pas un caractère quelque peu fictif.
Il faut en effet, bien se rendre compte que la gestion de priorité de ce type ne sera effective que si des requêtes conflictuelles sont effectuées dans un intervalle de temps relativement petit. Or, l'apparition de celles-ci dépend d'évènements extérieurs sur lesquels on n'a aucune influence.

- La MAJ + du stock dépend d'une arrivée de marchandises.
- La détection d'une correction à effectuer dans le stock dépend du moment où le magasinier affiche le bon de livraison.
- L'enregistrement d'une nouvelle commande dans le stock dépend du moment où l'opératrice envoie explicitement sa

requête au terminal.

D'autre part, comme on l'a déjà suggéré plus haut, il est impossible pour des raisons de généralité, d'introduire une gestion de priorités au niveau même de la station gérant les demandes d'accès aux fichiers sur le micro-disque.
Dans ces conditions, on se verrait contraint , pour traiter ce problème d'introduire un processeur supplémentaire qui serait spécialisé dans une gestion de files d'attente. Ce processeur recevrait les diverses demandes d'opérations sur le stock, et rangerait chacune d'elles dans une file d'attente, regroupant des demandes de même type. Chacune des files serait vidée de façon FIFO (First In First Out). Le choix d'une file est effectué en fonction des priorités imposées par l'application.
Selon toute vraissemblance, la mise en oeuvre d'une telle solution s'avérerait coûteuse eu egard aux bénéfices retirés.

Relevons ci-dessous quelques éléments susceptibles d'aller à l'encontre de cette solution.

- Il est probable que ces files d'attente seraient vides la majeure partie du temps.
  Pour que ce ne soit pas le cas, il faudrait que les demandes d'opération sur le stock soient très fréquentes ce qui risque d'être fort peu le cas dans un contexte interactif.
- Pour que la gestion de ces priorités aie un intérêt en soi, il faut évidemment qu'elles soient relatives à un même produit et dans un intervalle de temps assez réduit.
  Une telle probabilité est discutable bien qu'elle risque d'exister. On peut par exemple imaginer qu'un même produit soit fortement sollicité suite à une campagne publicitaire. Ce genre de situation est évidemment fort peu prévisible, et peut-être , s'avérerait-il intéressant d'effectuer une simulation pour obtenir des renseignements intéressants.

En conséquence, on peut donc dire qu'il faut ici raisonner en termes d'analyse coût/efficacité.
La solution consistant à introduire un processeur supplémentaire en vue de traiter ce problème spécifique des priorités est-elle justifiée? Quels seraient les bénéfices retirés compte tenu du coût supplémentaire qu'elle occasionne?

B. Si on relâche la contrainte 2, on risque tout au plus de différer quelque peu la prise en considération d'une commande normale par le système.
Mais, ici encore, on s'invente un problème, car tout dépend en fait du moment où se fera la livraison en stock.

C. Si on relâche la contrainte 3, les conséquences au niveau du traitement des commandes sont inacceptables.
En agissant de la sorte, on prend en effet une initiative qui va à l'encontre des désidératats exprimés dans la politique commerciale de l'entreprise.

Mais, on remarquera d'autre part, que le problème peut être résolu de façon très simple en réservant lors de la réception des marchandises la quantité nécessaire pour satisfaire l'ensemble des commandes différées en attente de ce produit, et en ne rendant disponible pour les commandes du jour que la quantité restante après cette réquisition.

D. La contrainte suivante est d'égale importance et sera respectée tout simplement en traitant les commandes différées selon l'ordre chronologique.

Pour les raisons mentionnées ci-dessus, on accordera seulement de l'importance aux contraintes de type 3 et 4 qui seront satisfaites par des mécanismes très simples.
On attirera enfin l'attention sur le fait que si, ultérieurement la gestion des priorités de type 1 et 2 s'avère indispensable, l'introduction d'un processeur spécialisé dans la gestion des files d'attente est excessivement simple et ne provoque aucune modification par rapport au système existant. Il s'agira en effet simplement d'adresser les requêtes à ce processeur qui assurera leur transmission dans l'ordre requis au gérant de la mémoire de masse.

5.2.5. Caractéristique d'une implémentation dans l'optique réseau.

Trois points importants retiendront notre attention quant aux caractéristiques que l'on peut dégager d'une telle implémentation.

Un premier aspect à souligner réside dans le fait que la réalisation de l'application" PETIT-PAS" sur un réseau local ne semble pas impliquer de différences fondamentales par rapport à une implémentation sur une machine centrale. Entendons par là le fait que la conception logique des modules ne s'en trouve pas affectée. Il faut en effet remarquer, que la distinction entre les deux solutions envisageables (optique centralisée ou optique réseau) reste localisée à un niveau relativement technologique et n'influe pas directement sur l'architecture d'implémentation. L'intérêt d'une telle comparaison aurait probablement été plus grand s'il s'était agi de réaliser un projet de taille plus importante sur un réseau général d'ordinateurs. Dans un tel contexte seulement, les deux composantes fondamentales du problème posé dans le cadre d'une solution répartie (à savoir la répartition des traitements et celle des données) auraient eu leur pleine signification. Comme on l'a déjà fait remarquer, la seule tentative de répartition envisageable au niveau des données consistait essentiellement à reléguer au niveau local des contrôles de validité. Quant aux traitements,le fait que le processeur soit localisé dans un micro-ordinateur ou dans la machine centrale est une différence purement technique.

Les deux points suivants jouent en faveur d'une implémentation réseau.

Un avantage imputable à une implémentation de type réseau se situe au niveau des pannes, et ce, principalement pour deux raisons.La première a déjà été évoquée au paragraphe 1.5. point 7. Il s'agit de la possibilité de faire tourner le système en mode"dégradé". Cette façon d'agir permet donc d'utiliser une partie du réseau dans le cas où l'un des composants n'est plus opérationnel. La seconde d'ordre purement technique et est due au caractère relativement standard des divers composants utilisés.

La modularité du système est également probablement supérieure dans le cadre d'un réseau local où l'adjonction de nouveaux processeurs (dans le cadre des limites technologiques) est sans grande incidence sur le reste du système.

CHAPITRE 6

PRIMITIVES DE GESTION DES FICHIERS

6.1. INTRODUCTION

Une première tâche à réaliser dans le cadre du projet consistera à implémenter pour la seconde station de ressources qu'on avait décidé d'adjoindre au réseau COBUS l'ensemble des ordres primitifs nécessaires à la manipulation des fichiers localisés sur le micro-disque WINCHESTER.
Deux types de fichiers devaient être disponibles pour l'utilisateur sur cette unité: des fichiers à accès séquentiel et des fichiers à accès aléatoire.
La gestion de ces fichiers doit se faire dans une station liée directement à la mémoire de masse si l'on veut réduire le trafic sur le réseau et améliorer le temps de réponse.
Tous les systèmes d'exploitation de micro-ordinateurs permettent de faire les opérations classiques suivantes sur les fichiers: ouverture, écriture, lecture et fermeture.

Dans le contexte ainsi défini, les ordres à exécuter par le gérant de la mémoire de masse comporte donc:

- des ordres analogues à ceux existant pour la première station de ressources et détaillés en 3.6.6.
- des ordres supplémentaires relatifs aux fichiers de type indexé à savoir: des ordres d'écriture/lecture en fonction de valeur de clé et des ordres de modification et destruction d'enregistrement.

Attirons enfin l'attention sur quelques points inhérents au fait que l'on désire avoir des accès multiples au gérant de la mémoire de masse:

- au niveau logiciel

  Il faut prévoir des mécanismes assurant l'intégrité des données manipulées par les divers utilisateurs de façon partagée.
  La principale difficulté technique est évidemment d'empêcher que les mises à jour faites par un usager ne viennent interférer avec les lectures et mises à jour d'un autre utilisateur.

- au niveau système

  Il est intéressant que l'interface avec le réseau soit asynchrone (c'est-à-dire travaille par interruption). Il faut donc assurer une gestion de tampons destinés à

recevoir les blocs et les renvoyer si l'on veut être assez rapide et ne pas rater de blocs.

Il faut que le système soit capable de maintenir plusieurs fichiers ouverts simultanément et de les étendre indépendamment les uns des autres.

Mises à part les deux contraintes mentionnées ci-dessus, le système n'a pas d'autres exigences particulières. Il ne doit pas être multi-tâche, les ordres étant traités séquentiellement.
L'installation à mettre en oeuvre utilisera le système UCSD (Université Californienne de San Diego). Ce système est moins puissant que RDOS de Data General (utilisé au LAMI) et permettant de maintenir ouverts 64 directoires ainsi que 64 canaux) car il est mono-utilisateur mais il possède toutefois des possibilités d'entrées/sorties asynchrones.

6.2. SPECIFICATIONS GENERALES DES PROGRAMMES A REALISER.

Pour bien mettre en évidence la nature des programmes à implémenter, il convient de détailler quelque peu la situation globale. Celle-ci est schématisée fig.1 PI 15 En fait, le but à atteindre est de fournir à l'utilisateur un certain nombre de primitives lui permattant de faire des opérations sur les fichiers séquentiels et indexés résidant sur le micro-disque WINCHESTER.
Il faut donc, d'une part, considérer les stations terminales des utilisateurs qui enverront des demandes d'accès aux fichiers et d'autre part, une station spécialisée dans la gestion des fichiers et que, dans la suite de l'exposé on nommera gérant de la mémoire de masse.

- Les fichiers de type séquentiel ne seront accessibles que par un utilisateur à la fois.
  Les primitives à développer pour ce type se résument essentiellement aux ordres de CREATION, OUVERTURE, FERMETURE, LECTURE de l'article suivant, ECRITURE de l'article suivant.
- Les fichiers de type indexé quant à eux, seront partageables par plusieurs utilisateurs simultanément. Les principaux ordres à développer seront l'OUVERTURE, la FERMETURE, LECTURE en fonction de clé, ECRITURE, MISE A JOUR et DESTRUCTION d'enregistrement.

(Remarquons que la création d'un fichier de type séquentiel se réduit à un ordre simple, il n'en ira pas de même dans le cas du fichier indexé qui s'avère être une opération quelque peu plus complexe. Il s'agira de mettre au point un programme réalisant cette opération).

Examinons dès à présent, un peu plus en détail la façon dont les choses se passent lorsqu'un utilisateur désire faire une opération quelconque sur un fichier. De façon à assurer une certaine compatibilité avec les primitives existant sur le réseau , les ordres ont été implémentés comme suit.

La requête exprimée par l'utilisateur faisant appel à une primitive se traduit par l'envoi d'un bloc sur le réseau à destination du gérant de la mémoire de masse. Ce bloc spécifie essentiellement deux types d'information:

- le code identifiant la nature de la requête de l'utilisateur (c'est-à-dire s'il s'agit d'un ordre d'ouverture, lecture, fermeture,...)
- un ensemble de paramètres spécifiques à l'ordre invoqué (par exemple pour un ordre d'ouverture, on mentionnera le nom du fichier).

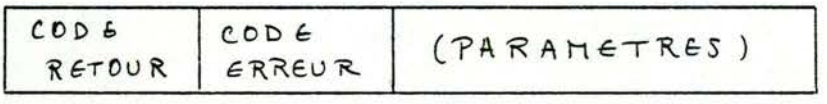
On peut donc, en toute généralité, schématiser un tel bloc comme suit:



La réponse à la requête de l'utilisateur se présente également sous forme d'un bloc envoyé par le gérant de la mémoire de masse vers la station terminale de l'utilisateur.
De façon générale, ce bloc contient:

- un code retour indiquant si la requête a pu ou non être satisfaite,
- un code erreur spécifiant, le cas échéant, pourquoi la requête n'a pu être satisfaite.
- éventuellement un certain nombre de paramètres caractéristiques de la requête.

Le susdit bloc aura donc la forme suivante:



On peut, par conséquent, discerner deux types de programmes à traiter:

1. Le programme destiné à tourner sur le gérant de la mémoire de masse.
   Il a pour but de traiter successivement ls ordres primitifs émis par les divers utilisateurs sur le réseau.
2. Les routines appelées par l'utilisateur via ses programmes.
   Celles-ci consistent à envoyer sur le réseau la requête de l'utilisateur à destination du gestionnaire de fichiers.
   Ce message comprend
   - le code correspondant à la nature de l'opération demandée
   - les paramètres nécessaires

6.3. CARACTERISTIQUES ET MODES DE REPRESENTATION DES FICHIERS SUR LA MEMOIRE DE MASSE

6.3.1. Quelques définitions

Avant d'aborder les problèmes de représentation physique des fichiers, il convient de clarifier certains concepts qui leur sont relatifs afin de s'entendre sur les termes employés par la suite. Ceux-ci peuvent en effet recouvrir des notions différentes en fonction du contexte où ils sont utilisés.
Fondamentalement, on distinguera deux aspects à un fichier:

- un aspect logique découlant de l'analyse fonctionnelle et qui précise les informations comprises dans le fichier ainsi que les contraintes et propriétés auxquelles elles satisfont. A ce niveau, on ne fait donc aucunement référence à une quelconque forme de représentation de ces données.

- un aspect technique désignant la façon dont sera organisé le fichier sur son support matériel. Ce niveau recouvre donc une notion relativement technique de réalisation du fichier.

6.3.1.1. Aspect logique d'un fichier [18]

Un fichier peut être perçu comme une collection logique d'informations répondant aux définitions suivantes.

Soient A_1, A_2, ..., A_m des ensembles de valeurs

Soient $a^i \equiv (a_1^i, a_2^i, \ldots a_R^i)$ des ensembles de valeurs prise chacune dans un ensemble A_i.
$a_j^i \in A_j$
Soit $A \equiv \{a^1, a^2, \ldots, a^m\}$ tout ensemble d'éléments a^i de même forme.

On appelle A_j : Attribut
a_j^i : valeur de l'attribut A_j
A : fichier de la forme $A\ (A_1, A_2, \ldots, A_R)$
a^i : un article du fichier A de type $A\ (A_1, \ldots, A_R)$

Les attributs possèdent diverses propriétés.

Ils peuvent être:

ELEMENTAIRES:c.a.d. ayant atteint un niveau atomique de décomposition
ou
DECOMPOSABLES:c.a.d. pouvant être subdivisés en éléments conservant une signification

SIMPLES:c.a.d. comportant une seule valeur
ou
REPETITIFS:c.a.d. susceptibles de comporter plusieurs

valeurs, auquel cas, le nombre d'occurences peut être fixe ou variable.

OBLIGATOIRES:c.a.d. comportant impérativement une valeur
ou
FACULTATIFS:c.a.d. susceptibles de ne pas comporter de valeur.

IDENTIFIANTS:c.a.d.que dans le fichier un seul article possèdera cette valeur d'attribut.

6.3.1.2. Aspect physique d'un fichier

6.3.1.2.1. Classes de support

Les propriétés technologiques des supports matériels déterminent certaines classes de support. Ceux-ci sont dès lors caractérisés par le mode d'accès aux informations qu'ils contiennent. On distingue:

- Les supports séquentiels

  Un tel support est fait d'un espace mémoire où les informations sont disposées consécutivement. Cet espace est doté d'un dispositif de lecture/écriture capable d'une part, de lire les données suivant celles qui viennent d'être lues, d'autre part, d'écrire des données à la suite de celles qui viennent d'être écrites. L'alternance entre ces deux types d'opérations n'est généralement pas possible. (Pas tellement pour des raisons techniques mais essentiellement pour des raisons technologiques).

- Les supports adressables

  De façon schématisée, on peut concevoir l'espace mémoire sur un support adressable comme étant constitué d'un ensemble de cellules destinées à mémoriser des informations. Chacune d'elle est munie d'une adresse permettant de la référencer univoquement parmi la totalité des cellules. Cet espace est doté d'un dispositif de lecture/écriture capable de se positionner directement sur une cellule dont on spécifie l'adresse en vue d'une opération d'écriture ou de lecture.

On appelle ENREGISTREMENT (record), l'ensemble des données indivisibles obtenu ou fourni par chaque opération d'accès au fichier.

On appelle BLOC l'ensemble des données faisant réellement l'objet d'un transfert entre la
mémoire centrale et la mémoire auxiliaire.

6.3.1.2.2. Organisation d'un fichier

La manière dont est obtenue l'adresse d'un enregistrement sur un support adressable définit ce que nous appellerons l'organisation d'un fichier. Il est usuel de distinguer trois grands types d'organisation: séquentielle, relative et indexée.

- Un fichier d'organisation séquentielle est caractérisé par le fait que les enregistrements y sont simplement rangés les uns à la suite des autres. Il n'existe aucun moyen de localiser un enregistrement particulier sur le support. Un fichier séquentiel ne peut par conséquent être traité en mode d'accès séquentiel.

- Un fichier d'organisation relative est caractérisé par le fait qu'un enregistrement peut y être identifié sur base de son numéro d'ordre dans le fichier.

- Un fichier d'organisation indexée est caractérisé par le fait qu'un enregistrement peut y être référencé sur base d'une valeur d'un de ses attributs.

Il s'agit donc d'associer à un attribut un mécanisme permettant d'accéder rapidement aux articles possédant une valeur déterminée pour cet attribut.
On peut imaginer diverses méthodes permettant de mettre au point un tel mécanisme d'accès. Les techniques les plus courantes peuvent être regroupées en trois grandes catégories qui s'inspirent des principes suivants:

- la recherche dichotomique
- l'indexation
- les fonctions de hash-code (accès calculé)

1. La recherche dichotomique

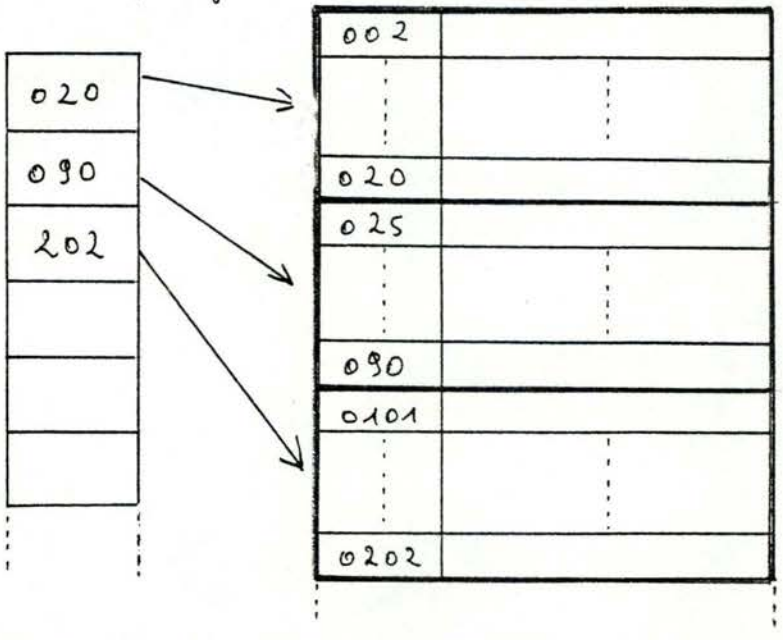
   La technique de la recherche dichotomique présuppose que l'on travaille sur une liste simple d'éléments adressables triés selon un certain ordre. Globalement, la méthode consiste à diviser initialement l'intervalle de recherche. Vérifier si l'élément recherché se trouve à gauche ou à droite de la borne ainsi déterminée. Si l'élément se trouve à gauche (droite), on répète le même procesus à gauche (droite). Et ce processus se poursuit jusqu'à ce qu'on ait trouvé l'élément recherché ou que la recherche se soit soldée par un échec.
   Des algorithmes détaillés existent dans la littérature.
   Concrètement, plusieurs solutions s'inspirant de cette méthode sont envisageables.

- On peut par exemple agir directement sur le fichier des enregistrements, ce qui suppose qu'on le maintienne continuellement trié.
- On peut encore associer au fichier de base contenant les enregistrements un second fichier contenant les valeurs triées des attributs ainsi qu'un pointeur vers l'article. La recherche dichotomique opérera donc sur cette liste.

2. L'indexation

La technique de l'indexe consiste à associer au fichier de base un second petit fichier (le fichier dit d'indexe) contenant des valeurs d'attributs limites permettant de référencer des articles dans le fichier de base.
En toute généralité, on peut représenter une structure d'indexe comme sur la fig. 2.

Fig. 2

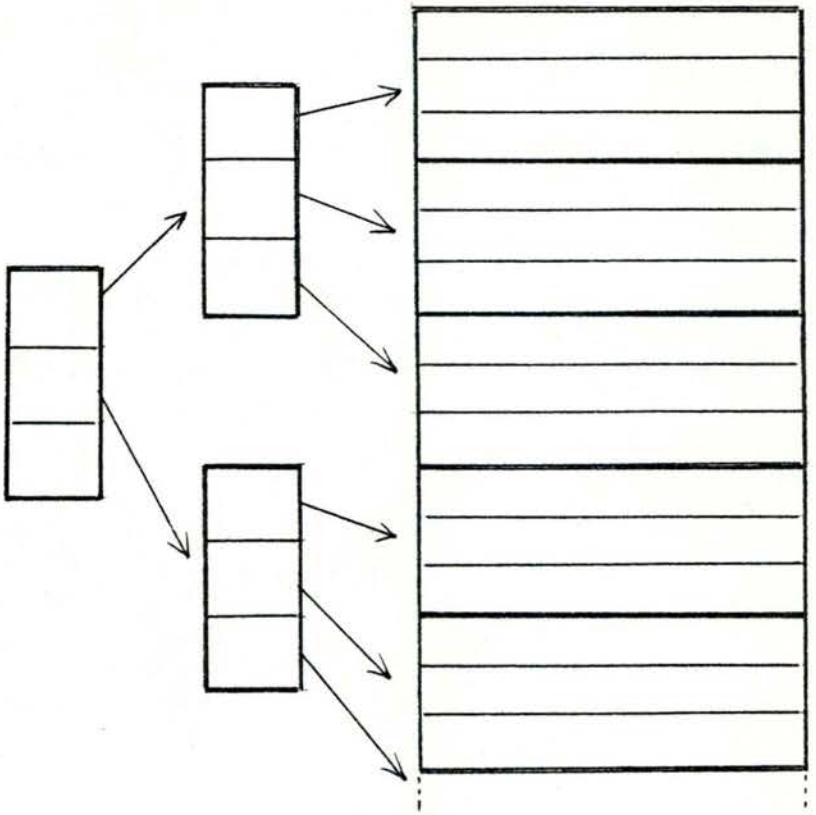


-

On distingue essentiellement deux types d'indexe:

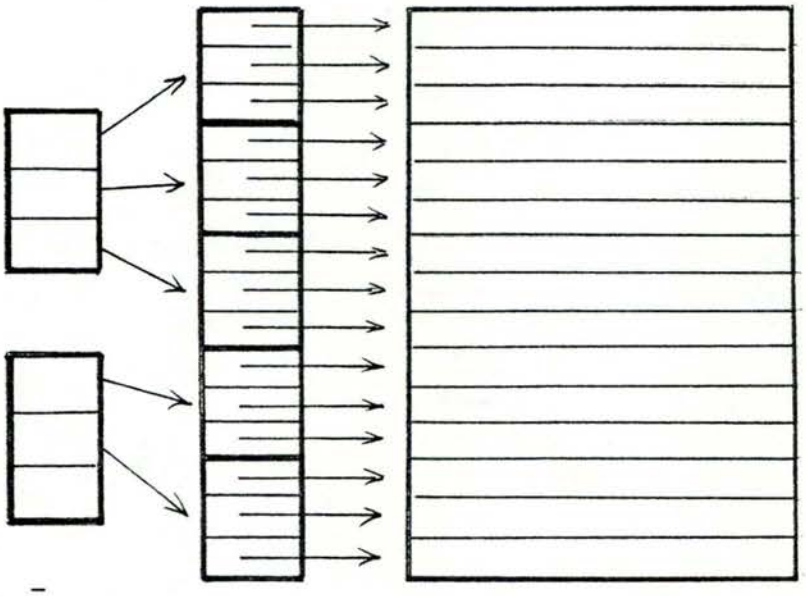
- un indexe non dense:
  Cela consiste, sur la base établie ci-dessus, à appliquer le même principe sur les blocs d'indexe existant et à constituer chaque fois un niveau d'indexe supérieur.(cfr. fig. 3)

Fig. 3



- un indexe dense: On a affaire à un tel type d'index lorsqu'il y a autant de sorties au dernier niveau d'indexe qu'il y a d'articles dans le fichier de base.(cfr. fig. 4)

Fig. 4



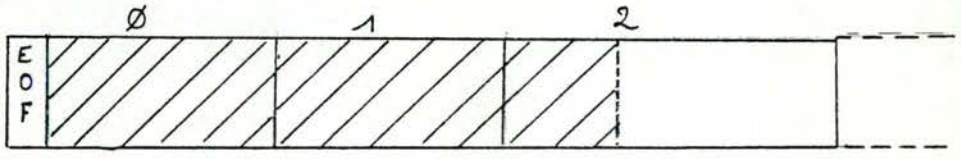
3. Les fonctions de Hash-code

Cette méthode consiste à associer une adresse à chaque valeur de clé par le biais d'une fonction. Le problème consiste donc à choisir judicieusement une fonction f telle que

adresse = f (valeur de clè)

6.3.2. Fichiers sequentiels

La structuration physique des fichiers de type sequentiel est evidemment très simple.Elle est representee ci-dessous:



- les trois bytes du premier bloc du fichier contiennent la position de la fin du fichier.(EOF)
- Les deux premiers bytes sont la representation d'un entier indiquant un numero de bloc de 512 bytes.
- Le troisième byte repère un bloc de 2 bytes dans un bloc de 512 bytes. (un byte = 256 valeurs possibles)

Lors de chaque écriture, les donnees sont ajoutees à la suite des precedentes. Si cela s'avère nécessaire, on ajoute de nouveaux blocs. L'aire hachurée represente la partie occupée du fichier.

6.3.3. Fichiers indexés

6.3.3.1. Types d'accès implèmentés

Remarquons ici que le choix des types d'accès à implèmenter n'a pas été exclusivement dicté par les besoins de l'application de gestion à réaliser dans le cadre de ce projet mais plutôt par le souci d'être suffisamment général. Il faut en effet attirer l'attention sur le fait que les primitives développées sont relatives à la gestion d'une station de ressources destinée à être employée par divers utilisateurs dans le cadre de leurs applications particulières.

Il s'avérait donc primordial d'assurer un certain niveau de service en mettant à leur disposition des outils relativement standard. C'est sur cette base que furent prises les décisions suivantes:

- Caractèristiques des clès d'accès

Une clè primaire doit être désignée. Celle-ci doit être univoque (identifiante) c'est-à-dire qu'à chacune de ses valeurs doit correspondre dans le fichier un seul article. Il s'agit donc d'un mécanisme d'accès associé à un attribut obligatoire identifiant et non répétitif.(cfr.paragraphe 6.3.1.1.) Plusieurs clés secondaires peuvent être désignées, celles-ci peuvent être associées à des attributs non identifiants et répétitifs.(cfr.6.3.1.1.)

- Types d'accès implémentés

  De façon générale, les opérations de lecture et consultation peuvent utiliser pour la recherche une clé quelconque (nous la nommerons clé de référence), les opérations d'écriture et de mise à jour ne peuvent utiliser que la clé primaire.
  Les types d'accès disponibles pour l'utilisateur sont détaillés au paragraphe 6.5.2.3.

6.3.3.2. Structuration physique proprement dite

La création d'un fichier de type indexé implique en fait la création de deux fichiers physiquement différents:

1. Un fichier de données contenant les enregistrements proprement dits qui seront manipulés par l'utilisateur dans son programme d'application.

2. Un fichier d'indexe contenant pour chaque clé créée la structure d'indexe y afférant.

La fig.5 représente globalement l'organisation de deux types de fichiers. Chaque bloc a une longueur fixe de 512 bytes. Pour être tout-à-fait complet, mentionnons encore l'existence de deux blocs spécifiques dans le fichier d'indexe.

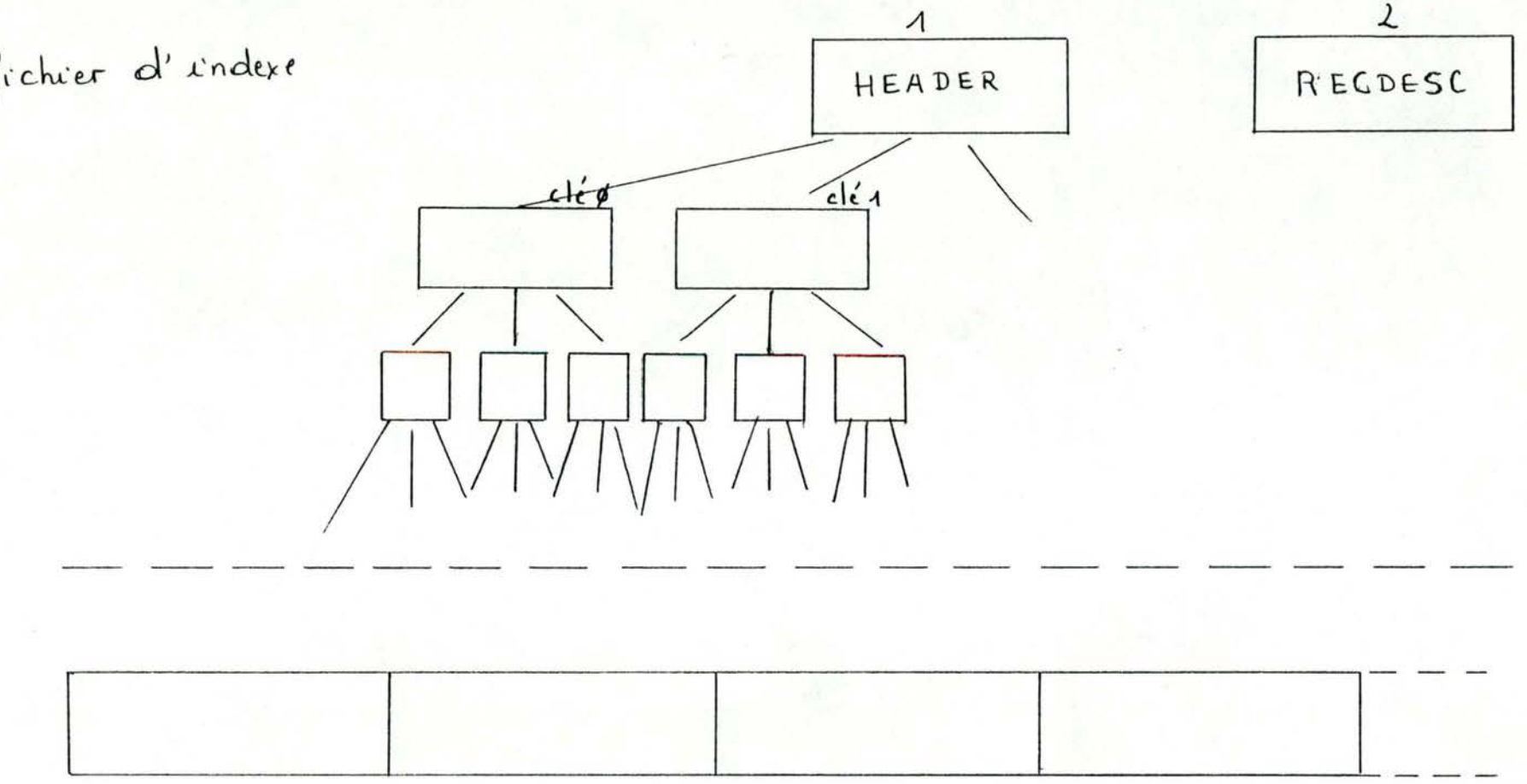

fichier d'index
1
HEADER
2
REGDESC
clé 0
clé 1
fichier de données

fig. 5

- un premier bloc, appelé HEADER contient essentiellement deux grands types d'information

  - les positions courantes de chacun des deux fichiers

  - un ensemble de caractéristiques relatives à chacune des clés implémentées (un pointeur vers le bloc représentant le sommet de la structure d'indexe la longueur de la clé,...)

Ces informations seront utiles au programme tournant sur le gestionnaire de la mémoire de masse afin d'assurer la bonne maintenance du fichier d'indexe.

- un second bloc, appelé RECDESC (record description) donne une description précise des zones des enregistrements. Ces renseignements seront exploités par les primitives appelées par l'utilisateur.

Rem.: Les deux premiers blocs du fichier d'indexe seront systématiquement réservés pour le HEADER et le RECDESC. Le détail des informations consignées dans ces blocs figurent en annexe avec la description des programmes.
Nous allons maintenant examiner plus en détail l'organisation physique des deux fichiers.

6.3.3.2.1. Organisation du fichier .INDX

Il convient de distinguer la structure d'indexe assobiée à la clé primaire identifiante de celle associée à une clé secondaire non identifiante.

A. Structure d'indexe associée à la clé identifiante.

Notons préalablement que l'entièreté des valeurs de clés existantes se trouvent reprises à ce niveau. On a donc affaire à un indexe de type dense (cfr.6.3.1.2.2.)

La structure d'indexe relative à la clé identifiante est schématisée fig. 6.

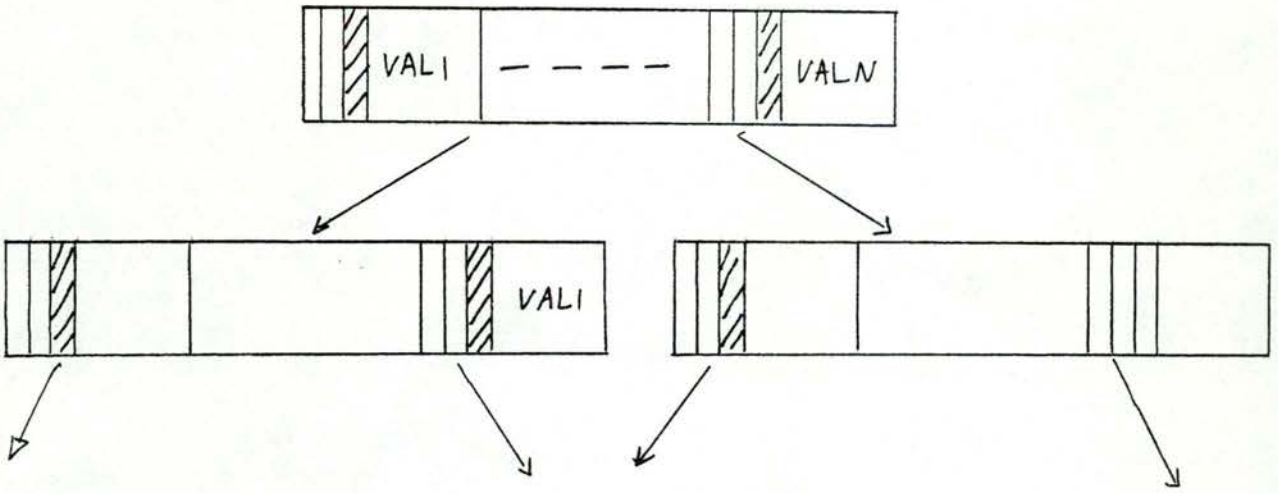
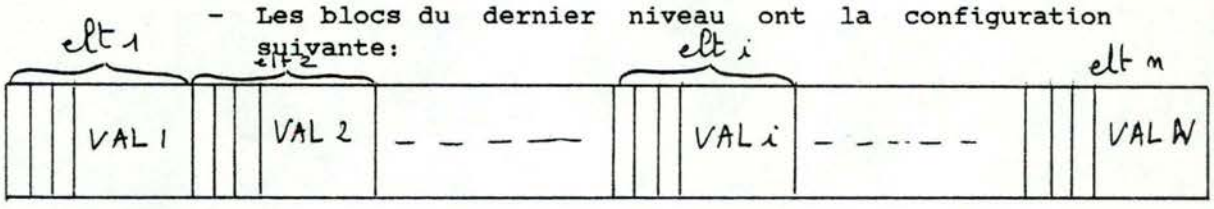


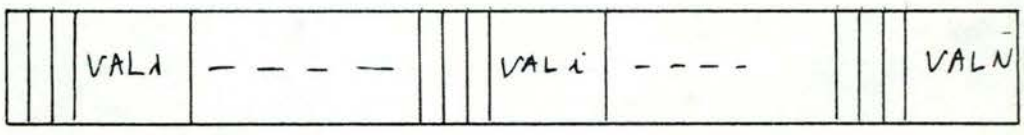
Fig. 6

- Les blocs du dernier niveau ont la configuration suivante:



Notons préalablement que l'entièreté des valeurs de clés existantes se trouvent reprises à ce niveau. On a donc affaire à un indexe de type dense (cfr.6.3.1.2.2.)
Un élément i du bloc comprend deux choses:
- un pointeur PTi
- une valeur de clé VALI Le pointeur PTi (3 bytes) permet de localiser dans le fichier de données l'enregistrement dont la valeur de clé primaire vaut VALI.

- Les blocs des autres niveaux ont la configuration suivante:



Un élément i du bloc de niveau k comprend également les deux types d'informations:
- PTi

- VALi
  mais leur signification est quelque peu différente
  .
  VALi représente la valeur de clé constituant la borne supérieure des valeurs de clés présentes dans le bloc de niveau k+1 désigné par PTi (2bytes) de clés présentes dans le bloc de niveau k+1 désigné par PTi (2bytes)

Les aires hachurées ne sont pas utilisées. Cela provient du fait qu'au dernier niveau, PTi requiert 3 bytes alors qu'aux autres niveaux 2 bytes seulement sont nécessaires. Cette façon de procéder occasionne une perte de place mais présente l'avantage de conserver une certaine compatibilité dans la structure des blocs manipulés à tous les niveaux et par conséquent, facilite le traitement.

B. Structure d'indexe associée à la clé non identifiante.

La structure relative à une clé non identifiante est en fait analogue à la précédente si ce n'est qu'on ajoute un niveau de pointeur supplémentaire au dernier niveau de l'arbre.
Globalement, la situation est présentée fig. 7.

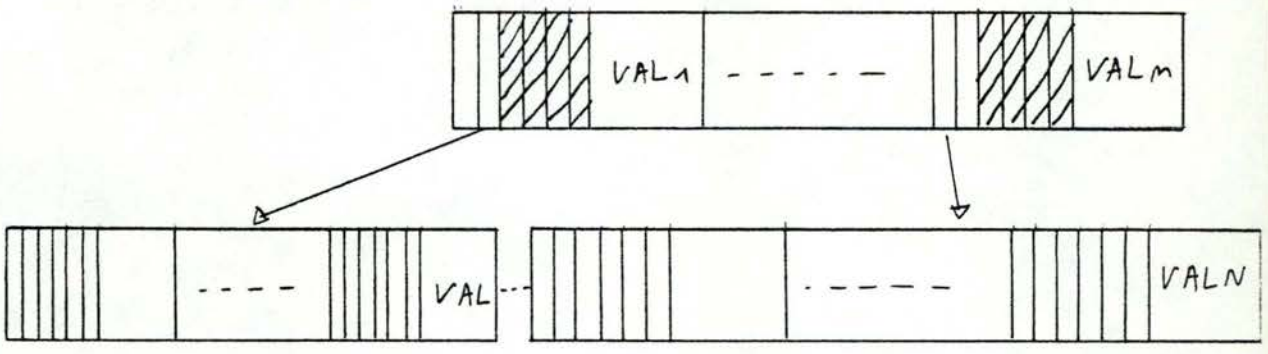
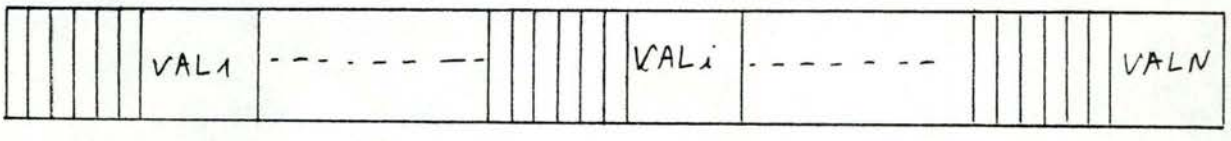


Fig. 7

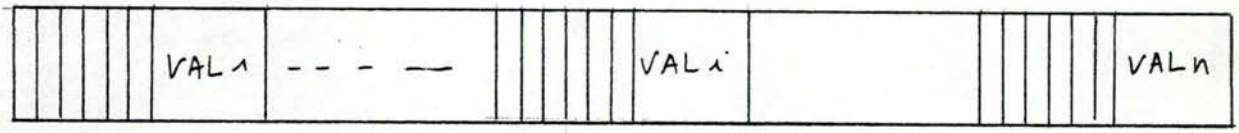
-

- Les blocs du dernier niveau ont la configuration suivante:



- PTi1 pointe vers le début d'une chaîne de pointeurs, chaque élément de cette chaîne pointant vers un enregistrement du fichier possédant VALi comme valeur pour la clé secondaire.
- PTi2 pointe vers la fin de la chaîne de pointeurs évoquée ci-dessus.

- Les blocs des autres niveaux ont la configuration suivante:



Un élément i du bloc contient les mêmes informations que celles mentionnées dans le cas de la clé identifiante.

Pour les mêmes raisons que celles expliquées au pt.A , l'aire hachurée n'est pas employée.

6.4. GESTION DES ACCES CONCURRENTS AUX FICHIERS

6.4.1. Introduction

Le but de ce paragraphe est une tentative de synthèse des types de problèmes susceptibles de se poser dans un environnement base de données partagée.
On peut percevoir un tel système comme étant constitué de N utilisateurs, (chacun d'eux concrétisé par le processus qu'il exécute) en compétition dynamique pour l'obtention de ressources parmi N (en l'occurence des unités d'information dans la base de données). Dans un tel contexte , des mécanismes de contrôle et de protection s'imposent en vue de garantir l'intégrité de la base de données. Garantir cette intégrité recouvre en fait différents aspects. On distingue essentiellement trois types de problèmes:

- la protection de l'existence des données dans la base grâce à des mesures de reprise, sauvetage...
- le maintien de la qualité des informations en veillant à résoudre les problèmes d'interférence entre les divers processus..
- le maintien du caractère privé des informations grâce à une gestion des accès aux données.

Nous nous limiterons dans le cas présent à analyser les problèmes du second type.
Pour ce faire, nous nous efforcerons de mettre en évidence les

diverses situations problèmatiques susceptibles de se produire et, à partir de là, nous brosserons un éventail des solutions envisageables. Nous motiverons dès lors les choix retenus dans le cadre du projet en vue de garantir l'intégrité des données partagées.
Remarquons que ces problèmes très complexes mais relativement classiques ont déjà fait couler beaucoup d'encre et occupent une part importante dans la littérature.
Notons enfin que cette complexité va croissant dans un environnement réparti, mais que nous n'y accorderons pas un intérêt particulier eu égard au caractère centralisé des informations dans le cadre du projet à mettre en oeuvre.

6.4.2. Notion de blocage.

Une solution immédiate vient à l'esprit pour résoudre les problèmes évoqués ci-dessus qui consiste à introduire une forme d'usage exclusif des ressources. Cela peut s'obtenir par un BLOCAGE des ressources en question.
Remarquons néanmoins, dès à présent, que cette solution est elle-même source d'un nouveau problème non négligeable : celui des INTERBLOCAGES.
Une telle situation se produit dans les circonstances suivantes:
Soit P et Q, deux proccessus en compétition pour l'usage de deux ressources A et B .
Supposons alors la séquence d'évènements suivante:

1. Le processus P demande et acquiert le contrôle exclusif de la ressource A,
2. le processus Q demande et acquiert le contrôle exclusif de la ressource B,
3. Le processus P demande l'usage de la ressource B mais ne pourra l'obtenir que lorsqu'elle sera relâchée par le processus Q,
4. Le processus Q demande l'usage de la ressource A, mais de façon analogue, il ne pourra l'obtenir que lorsqu'elle sera relachée par le processus P.

Diverses stratégies sont envisageables en vue de résoudre ce nouveau problème. On peut les classer en trois grandes catégories:

1. Ignorer les interblocages

   On laisse les situations d'interblocage se produire et on les découvre par des moyens extérieurs.
   Cette position était défendable au début de l'informatique, mais ne l'est plus du tout actuellement à cause des tendances récentes telles que temps réel, large base de données partagée...

2. Détecter les interblocages

Plusieurs auteurs ont présenté diverses méthodes de détection des situations d'interblocage. La seule solution lorsqu'un événement de ce type se produit est de retirer des ressources à un ou plusieurs processus et laisser les autres poursuivre leur exécution en les leur affectant. Ceci pose un nouveau problème qui est celui du "déroulement en sens inverse" (roll back) d'un processus.

3. Empêcher les interblocages

Cette stratégie s'avère relativement complexe dans le cadre particulier du partage des données. En effet, une solution consisterait à pouvoir déclarer avant l'exécution d'un processus l'ensemble des ressources dont il aura besoin , mais ceci est pratiquement impossible en ce qui concerne les données, du moins si l'on désire un niveau de blocage relativement fin ,ce qui est quasiment une contrainte dans un contexte interactif.

6.4.3. Principales situations imposant le blocage

Le blocage est rendu indispensable essentiellement par trois types de problèmes majeurs:

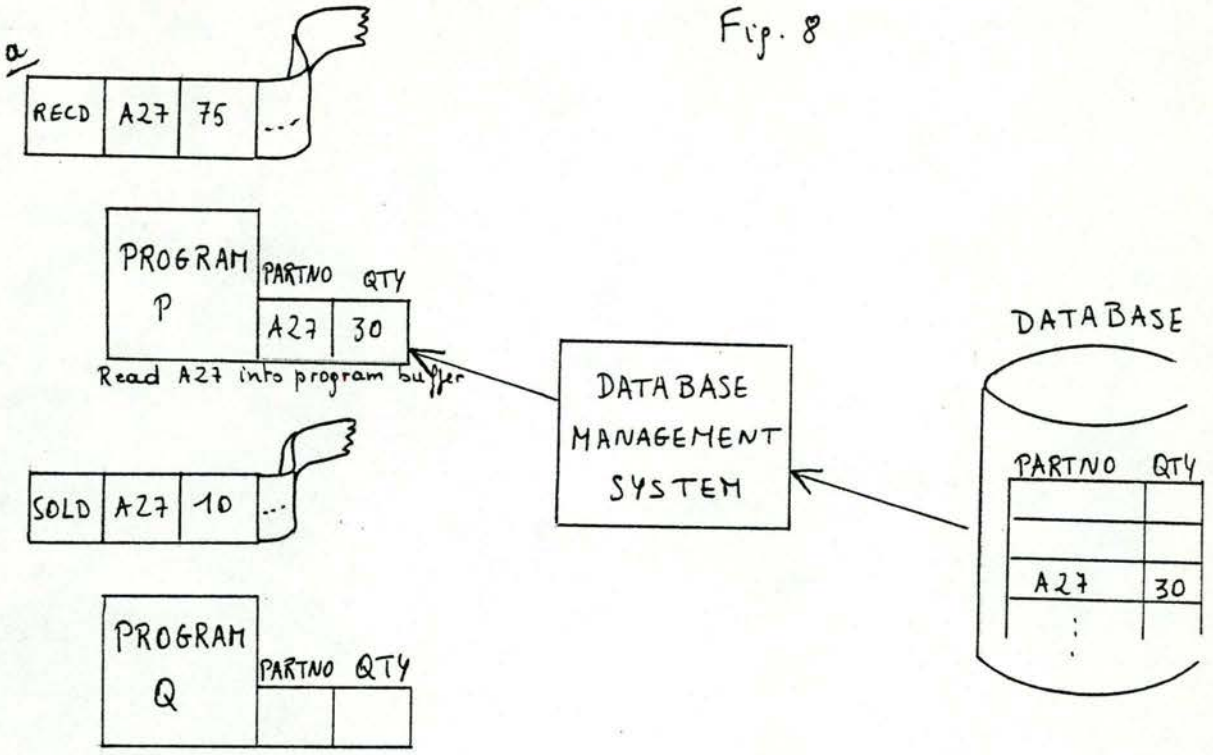
1. Le problème des mises à jour perdues, Nécessité de " défaire " un processus, Le problème de l'intégrité d'un processus.

1. Le problème des mises à jour perdues

Ce problème relativement classique résulte d'une séquence d'événements telle que la suivante:

Supposons que deux processus concurrents P et Q désirent mettre à jour le le même enregistrement A (Cfr. fig. 8)

Fig. 8



PROGRAM BUFFER:

DATABASE CONTENT:

b

PROGRAM P | A27 | 105 | (= 30 + 75)

Process transaction and update A27 in buffer

PROGRAM Q | A27 | 30 |

Read A27 into buffer

c

PROGRAM P | A27 | 105 |

Write updated A27 record to the database

PROGRAM Q | A27 | 20 | (= 30 - 10)

Process transaction and update A27 in buffer

d

PROGRAM P | A27 | 105 |

PROGRAM Q | A27 | 20 |

Write updated A27 record to the database

Supposons que les divers accès à la base de données s'opèrent selon la séquence ci-dessous:

1. Le processus P lit l'article A et en obtient donc une copie dans son buffer.
2. Le processus Q lit également l'article A et en obtient une copie dans son buffer.
3. Le processus P modifie l'article A et en demande la réécriture dans la base de données.
4. Le processus Q modifie sa copie de l'article A et en demande à son tour la réécriture dans la base de données.

Conséquence: La mise à jour effectuée par le processus P n'a pas été prise en considération.

2. Nécessité de "défaire" une transaction

Il peut parfois être indispensable de " défaire " un processus. Comme on l'a déjà mentionné, cela se produit lorsqu'on désire résoudre une situation d'interblocage.
Ce sera également une obligation lorsqu'un processus, pour quelque raison que ce soit , se termine anormalement. Cette action de retour en arrière implique d'"effacer" tous les changements effectués dans la base de données par le processus en question. Les exemples ci-dessous prouvent que cette opération n'est pas sans incidence sur les autres processus manipulant les mêmes données.

Exemple 1: (Cfr. fig.9)

TRANSACTION A	time	TRANSACTION B
–	t1	–
change R	t2	–
unlock R	t3	–
–	t4	find R
** ROLLBACK **	t5	–
–		–
–		–

en t1 : le processus A bloque l'enregistrement R (lecture)
en t2 : le processus A modifie l'enregistrement R
en t3 : le processus A débloque l'enregistrement R (écriture)
en t4 : le processus B lit l'article R
en t5 : pour une raison quelconque, le processus A doit être défait

Conséquence: Le processus B a lu un article qui n'a pas réellement existé.

Exemple 2: (Cfr. fig. 10)

Fig. 10

Transaction A	time	Transaction B
–	t_0	–
change R	t_1	–
unlock R	t_2	–
–	t_3	FINDX R
–	t_4	change R
ROLLBACK	t_5	–
–		–
–		–

Cet exemple est analogue au précédent, si ce n'est que cette fois, le processus B ne s'est pas contenté de lire l'enregistrement R, mais l'a modifié.
Si la première situation peut être tolérable dans certaines circonstances, ce n'est pas du tout le cas dans l'exemple 2.
Une règle simple peut être déduite:
"Les changements non confirmés doivent rester bloqués jusqu'à la fin de l'exécution du processus."

3. Problème de l'intégrité d'un processsus

Il se peut qu'un processus exige que les données qu'il manipule soient bloquées vis à vis de mises à jour effectuées par d'autres processus , même s'il ne fait que des opérations de consultation.

Exemple: (Cfr. fig. 11)

ACC 1	ACC 2	ACC 3
40	50	30

Transaction A	time	transaction B
–	t_0	–
FIND ACC 1 (40) Sum = 40	t_1	–
FIND ACC 2 (50) Sum = 90	t_2	–
–	t_3	FIND ACC 3 Substract 10
–	t_4	FIND ACC 4 add 10
FIND ACC 3 (20) Sum = 110, not 120	t_5 t_6	COMMIT – –
–		–

Considérons deux processus A et B sur un ensemble d'enregistrements représentant des comptes bancaires.
Le processus A a pour objet de faire la somme des soldes de l'ensemble des comptes. Sur les comptes proprement dits, il ne fait donc que des opérations de stricte consultation.

Le processus B quant à lui, a pour objet de faire des tranferts d'un compte vers un autre.
Par conséquent, si l'on se réfère à la fig. 11, on se rend compte que le résultat produit par A (110) au temps t7 est manifestement faux.

6.4.4. Impact des mécanismes de contrôle de la base de données sur les programmes d'application

L'impact des techniques mises en oeuvre en vue de garantir l'intégrité d'une base de données partagée sur les programmes d'application est souvent passée sous silence car, idéalement, ces moyens devraient être transparents à l'utilisateur. Mais, pour qu'il en soit ainsi, c'est souvent au détriment des performances. En effet, il n'est pas souhaitable que l'entièreté des données utilisées par l'application (ou pire, susceptibles d'être utilisées par l'application) soit bloquée durant tout le temps de l'exécution du programme utilisateur, car cela réduirait fortement les possibilités de concurrence.
Dans cette optique, le système doit être pourvu de règles et protocoles établissant sur quel ensemble de données doit porter le blocage et pendant quel intervalle de temps ce blocage doit opérer.
En toute généralité, il est souhaitable que les mécanismes soient conçus de façon telle qu'ils

- maximisent le degré de concurrence entre les processus s'exécutant,
- aient un impact minimal sur la conception logique des programmes.

En fait, ces objectifs sont contradictoires et les mécanismes instaurés dans l'un ou l'autre système assurent un compromis entre les deux. Des préoccupations de cet ordre influent sur deux aspects fondamentaux dans les méthodes proposées : la granularité du blocage et le type de blocage. Ces deux points sont examinés ci-dessous.

6.4.5. Granularité du blocage

Théoriquement, les volumes d'information concernés par un mécanisme de blocage peut varier très fort entre les deux extrêmes suivants: le fichier entier ou une zone dans l'enregistrement. Les systèmes implémentés mettent évidemment en oeuvre une solution mitigée. Ce que l'on convient d'appeler la granularité du blocage porte donc sur le degré de finesse de l'information faisant l'objet d'une mesure de blocage.

6.4.6. Type de blocage

Fondamentalement, on distingue deux types de blocages. La raison principale présidant à cette différenciation est la suivante.
Supposons un processus P qui ne fait que des opérations de consultation et qui a requis un blocage des données qu'il manipule en vue de garantir l'intégrité de son exécution (Cfr. parag. 6.4.3. pt 3). S'il n'existe qu'un seul type de blocage, un autre processus désirant lui aussi uniquement faire des opérations de consultation, se verra inutilement refuser l'accès à ces données. Ceci explique pourquoi il est utile de disposer des deux types de blocages suivants:

LOCK R (blocage en lecture)

Cette requête émane d'un processus qui a l'intention de lire des données et qui désire que les accès concurrents soient interdits s'ils ont pour objet de modifier ces données. Dans ce cas, d'autres processus requérant un LOCKR recevraient l'autorisation d'accès.

LOCK U (blocage en mise à jour)

Cette requête émane d'un processus qui a l'intention d'aller modifier des données. Dans ce cas, il faut qu'il n'y ait aucun blocage préalable de type LOCKR R ou LOCK U afférant à ces données.

Divers blocages plus nuancés " sont envisageables qui font intervenir à la fois les notions de granularité et type qui viennent d'être examinées. A titre d'exemple, mentionnons la référence [17] qui propose 5 types de blocage: S, X, IS, IX, SIX.

6.4.7. Solution retenue dans le cadre du projet.

Dans le contexte particulier qui nous intéresse, examinons l'impact des mécanismes de blocage envisageables à la lumière de deux critères: la granularité du blocage et l'intervalle de temps du blocage.

En ce qui concerne la granularité, le choix s'impose entre deux alternatives: le blocage au niveau du fichier ou le blocage au niveau de l'enregistrement. Pour chacune de ces deux solutions, examinons les possibilités subsistant au niveau de l'intervalle de temps du blocage et dégageons dans tous les cas la part de responsabilité laissée au programmeur ainsi que le niveau de performance atteint.

Première hypothèse : blocage au niveau du fichier

Si l'on envisage de faire porter le mécanisme de blocage sur la totalité du fichier, il est hors de question que le blocage opère durant tout le temps de l'exécution du processus pour des raisons évidentes de performance. Il convient donc de fournir au programmeur des outils lui permettant de limiter le temps du blocage à une séquence d'exécution pour laquelle des interférences de la part d'autres processus sont intolérables si l'on veut garantir la validité des données manipulées et du processus exécuté. Sur cette base, on peut proposer à l'utilisateur de définir une transaction en l'insérant entre deux primitives: LOCK R et LOCK U (cfr.parag.6.4.6.)

Avantage:

Cette solution laisse une responsabilité limitée au programmeur. Il lui suffit de demander le blocage du fichier sur lequel il travaille durant le temps de l'exécution de son processus pour être assuré de l'intégrité des données qu'il manipule.

Inconvénient:

Un inconvénient majeur de cette solution réside dans le fait qu'elle n'est pas du tout adaptée dans un contexte interactif. Imaginons en effet un système où N utilisateurs travaillent en mode interactif sur un même fichier. Il suffit que l'un d'eux, pour quelque raison que ce soit, laisse un délai important s'écouler entre la consultation d'un enregistrement et sa réécriture dans le fichier pour empêcher l'ensemble des autres utilisateurs d'accéder au fichier.
Une telle solution est donc susceptible de provoquer une dégradation intolérable du temps de réponse. Cette raison, à elle seule, est jugée suffisamment importante pour justifier le rejet de cette solution.

Deuxième hypothèse: blocage au niveau de l'enregistrement

Une deuxième solution consiste à fournir un mécanisme permettant de bloquer un article du fichier pendant tout le temps nécessaire à sa mise à jour; c'est-à-dire, à partir du moment où un processus en demande la lecture jusqu'au moment où il réécrit la version modifiée dans le fichier. A un niveau aussi fin, il n'est pas indispensable de prévoir les deux types de blocages mentionnés au parag.6.4.6. . Lorsqu'un article est bloqué, il est inaccessible aux autres processus qui désirent mettre à jour ce même enregistrement.
Pour mettre en oeuvre cette solution, il s'agit de mettre au point les deux primitives suivantes:

RDMOD < identifiant - article >
REWRITE < identifiant - article >

Un niveau de blocage aussi fin risque de ne pas être suffisant pour toutes les applications. Il est intéressant de prévoir une possibilité de bloquer plusieurs articles simultanément. Evidemment, il faut décider au préalable d'un nombre maximum d'articles.
Soulignons toutefois que le fait d'introduire cette possibilité

risque de faire surgir un problème déjà évoqué: celui des interblocages. Ce problème peut être évité de deux façons:

1. on donne la possibilité au programmeur de demander en une seule requête l'ensemble des ressources dont il a besoin.
2. L'ordre de réquisition des ressources de la part des divers processus doit être le même. (Ce qui n'est pas une évidence lorsqu'il s'agit de données).

Avantage:

Au niveau des performances, cette solution est de loin préférable à la précédente. Un article ou un ensemble d'articles ne sera bloqué que durant le temps de leur exploitation par un processus, ce qui est le minimum requis.

Inconvénient:

L'accroissement des performances apporté par cette deuxième solution va de paire avec l'augmentation de la responsabilité du programmeur dans la conception logique de ces programmes. Celui-ci doit en effet accorder une plus grande attention à ces problèmes et doit être conscient des contraintes qui lui sont imposées.

Remarque:

Les deux primitives proposées ne suffisent pas pour résoudre le problème de l'intégrité d'un processus. Il conviendrait pour apporter une solution à ce problème de réaliser en outre une possibilité de blocage au niveau de l'entièreté du fichier.

6.5. EXAMEN DES PROGRAMMES

6.5.1. Mode de présentation des traitements

6.5.1.1. Notions de base

Avant d'entamer la description des programmes, il convient de dire un mot au sujet du formalisme employé pour exposer les traitements. Le mode de représentation adopté repose sur ce qu'en théorie des graphes, on nomme des arborescences. De tels graphes sont fréquemment utilisés dans le cadre de diverses applications et leur compréhension ne nécessite pas de développement particulier.

6.5.1.2. Concept d'arbre programmatique [19]

L'arbre programmatique est un graphe de type arborescent figurant la structuration de l'ensemble des traitements. La recherche de cette structuration se conduit par décompositions successives de la fonction en sous-fonctions , niveau par niveau.
On obtiendra par exemple une structure du type de celle représentée fig.12

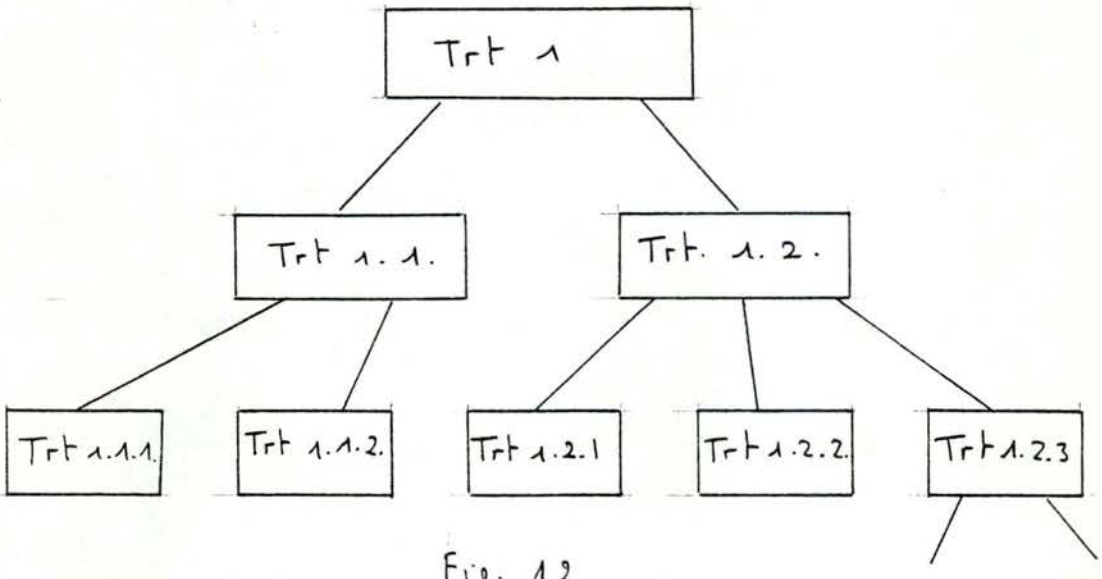
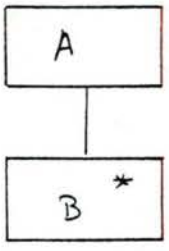


Fig. 12

Un arbre de ce type décrit incomplètement les traitements à effectuer, car il ne mentionne pas la fréquence d'incidence de chacun d'eux. Il s'agit d'inclure cette possibilité.
On distingue comme structure de base:

- la structure répétitive ou itérative
- la structure conditionelle
- la structure sélective.

La structure répétitive est représentée comme suit:

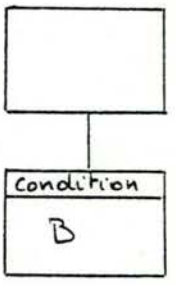


Le signe "*" indique que le traitement A consiste en plusieurs occurences du traitement B.

Notons ici la distinction entre deux types possibles d'itération:

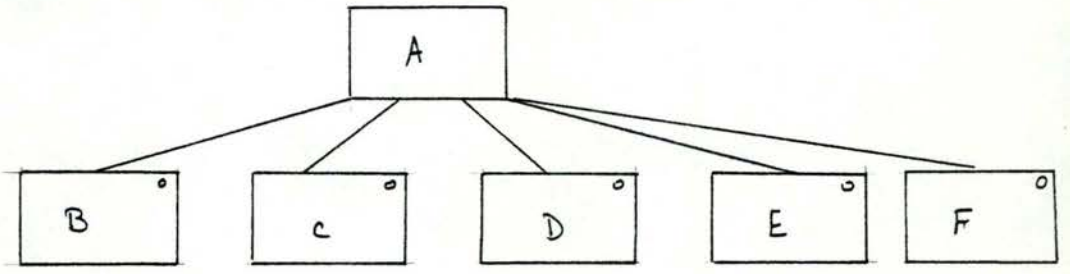
- une première possibilité consiste à tester la condition de fin du traitement itératif avant chaque exécution potentielle du dit traitement,
- une seconde possibilité consiste à tester l'arrêt en fin d'exécution du traitement itératif.

Le langage de programmation PASCAL autorise les deux formes; il s'agit respectivement du " WHILE condition DO trt " et du " REPEAT traitement UNTIL condition ".
Dans certaines circonstances, la distinction entre ces deux formes peut avoir son intérêt. Nous prendrons la liberté, pour distinguer ces deux possibilités, d'adopter le formalisme suivant:
La structure conditionnelle est représentée comme suit:

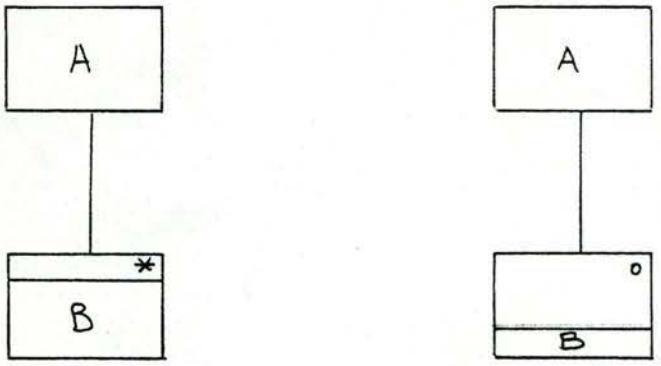


Le traitement B n'est exécuté que si la condition spécifiée dans la partie supérieure du cadre est satisfaite.
Dans le cas contraire, le traitement est simplement omis.

La structure sélective est représentée comme suit:



Le signe "o" indique que le traitement A consiste en une sélection parmi les traitements B, C, D, E et F. Cette sélection s'opère sur base de la condition exprimée dans la partie supérieure de la case représentant le traitement.
Un programme est une combinaison des deux figures représentatives des deux structures de base.
Cette combinaison se fait par imbrication entraînant une augmentation du nombre de niveaux dans l'arborescence ou par concaténation à un même niveau. A un même niveau, l'ordre d'exécution va de la gauche vers la droite. Dans un arbre programmatique, l'axe horizontal donne l'ordre d'exécution et l'axe vertical correspond au niveau d'imbrication. (cfr.fig.13)



6.5.2. Primitives implèmentèes.

6.5.2.1. Gènèralitès

Le programme destinè à être exècutè par la station gèrant le micro-disque fut le premier à être mis au point. Il a pour objet le traitement sèquentiel des requêtes èmises par les divers utilisateurs en vue d'une opèration sur un des fichiers. Pour assurer une bonne gestion de l'ensemble du système, ce programme tient à jour deux tables: la table des fichiers et la table des canaux. Examinons le contenu de ces deux tables.

1. La table des fichiers.

Celle-ci gère l'ensemble des fichiers mis à la disposition des utilisateurs du système. Un èlèment de cette table comprend les renseignements suivants:

BUSY	USERCOUNT

BUSY : variable de type boolèen indiquant si le fichier correspondant est libre ou occupè.

USERCOUNT : variable de type entier spècifiant le nombre d'utilisateurs faisant des opèrations sur le fichier.

Notons que:

usercount = 1 pour un fichier de type sèquentiel

usercount > 1 pour un fichier de type indexè car dans ce cas on autorise le partage d'un même fichier pour plusieurs utilisateurs.

2. La table des canaux.

Le concept de canal a dèjà ètè èvoquè (cfr.parag.3.6.3.), rappelons brièvement en quoi il consiste. A chaque ouverture de fichier, l'utilisateur se voit attribuer en retour un numèro appelè " numèro de canal ". En fait, ce numèro correspond à une entrèe dans la table des canaux où se trouveront diverses informations utiles au programme de gestion des fichiers. La notion de canal est donc relative à la fois à un utilisateur et à un fichier.

A titre d'exemple, considèrons la situation suivante:

Soit les utilisateurs USER-1, USER-2
Soit les fichiers FILE-1, FILE-2, FILE-3 de type indexè et donc susceptibles d'être ouverts par plusieurs utilisateurs simultanèment.

Supposons que

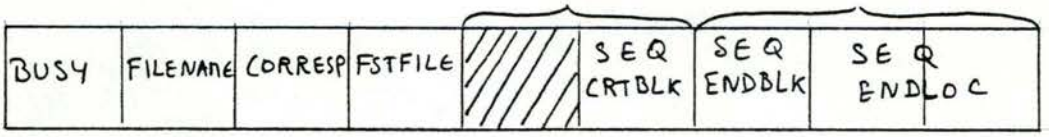
l'utilisateur USER-1 ouvre les fichiers FILE-1 et FILE-2
l'utilisateur USER-2 ouvre les fichiers FILE-1 et FILE-3

Les canaux suivants seront attribuès:

canal i pour le fichier FILE-1 de l'utilisateur USER-1
canal j pour le fichier FILE-2 de l'utilisateur USER-1
canal k pour le fichier FILE-1 de l'utilisateur USER-2
canal l pour le fichier FILE-3 de l'utilisateur USER-2

Les renseignements contenus dans un èlèment de la table des canaux diffèrent quelque peu avec la nature du fichier ouvert par l'utilisateur (sèquentiel ou indexè). Examinons ce qu'il en est dans les deux cas.

- Cas d'un fichier sèquentiel



BUSY : variable de type booléen témoignant du caractère libre ou occupé du canal.

FILENAME : chaîne de caractères correspondant au nom du

fichier ouvert par l'utilisateur.

CORRESP.: variable de type entier identifiant l'utilisateur (il s'agit en fait du numèro de la station de l'utilisateur).

FSTFILE : variable de type entier contenant le numèro du fichier à l'utilisateur (ce numèro correspond à une entrée dans la table des fichiers).

SEQCRTBLK: sequential current block
SEQCRTLOC: sequential current location
variables de type entier.
Leur combinaison spècifie la position courante de l'utilisateur dans le fichier.

SEQENDBLK: sequential end block
SEQENDLOC: sequential end location.
variables de type entier.
Leur combinaison spècifie la position de la fin
du fichier.
Le système PASCAL travaille par blocs de 512 bytes;
à l'intèrieur de ces blocs, une dècoupe en blocs de
2 bytes est effectuèe par le programme.
Dans ces conditions, pour dèfinir univoquement un
endroit dans le fichier, on spècifie:
- le numèro du bloc de 512 bytes
- le numèro du bloc de 2 bytes à l'intèrieur.

- Cas du fichier indexè

Remarque prèliminaire: L'ouverture d'un fichier de type indexè correspond en fait à l'ouverture de deux fichiers physiquement diffèrents: un fichier avec l'extension . DATA, l'autre avec l'extension . INDX (cfr.paragraphe 6.3.3.2.)

BUSY	FILENAME	CORRESP	FSTFILE	SEGFILE	INDX CRTBLK	INDX CRTLOC	CRT KEY VAL	CRT KEY OCC	- - - -

BUSY: variable de type booléen tèmoignant du caractère libre ou occupè du canal.

FILENAME : chaîne de caractères correspondant au nom du fichier ouvert par l'utilisateur (nom avec

l'extension .INDX).

CORRESP : variable de type entier identifiant l'utilisateur (il s'agit en fait du numéro de la station utilisatrice).

FSTFILE : (first file) variable de type entier contenant le numéro du Ce numéro correspond à une entrée dans la table des fichiers).

SECFILE : (Second file) Idem avec le fichier .DATA

INDXCRTBLK: index current blok
INDXCRTLOC: index current location
variables de type entier
Ces deux informations combinées indiquent la dernière position du fichier d'indexe.

CRTKEYVAL : current key value
CRTKEYOCC : current key occurence
La combinaison de ces deux informations indique la dernière valeur de clé référencée par l'utilisateur.

Si l'on représente les deux tables dans leur totalité, on obtient la situation représentée fig.14
(1) l'utilisateur du canal 1 a ouvert un fichier séquentiel
(2) l'utilisateur du canal 4 a ouvert un fichier indexé

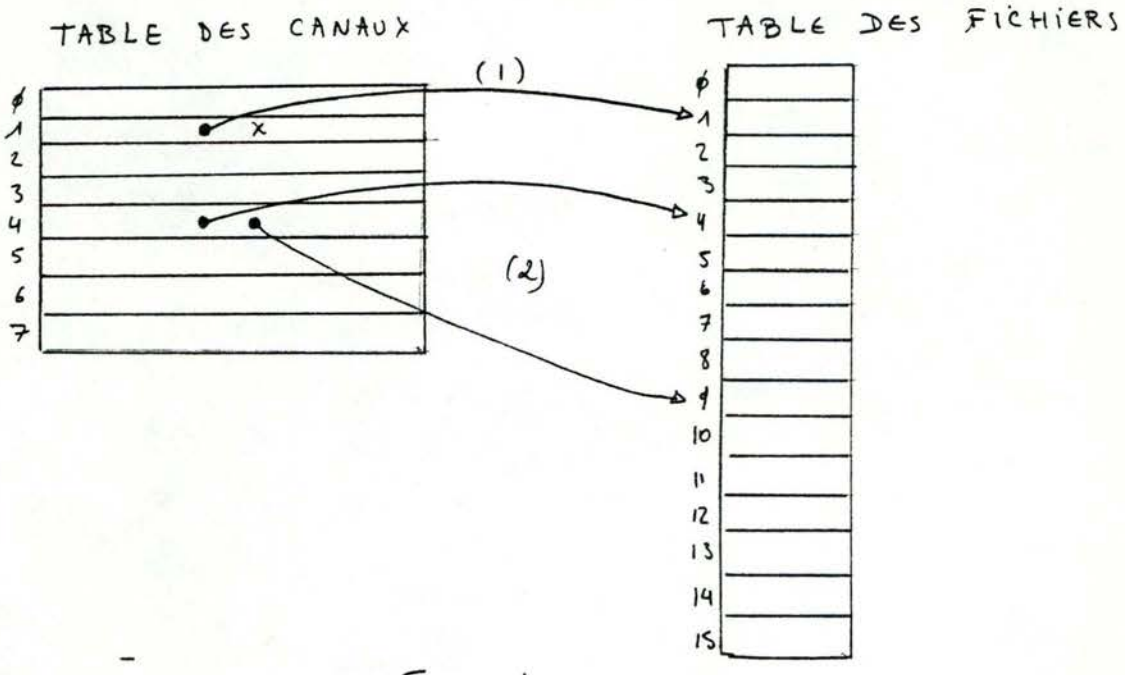


Fig. 14

Remarque.

Dans sa version actuelle, le programme autorise au maximum l'ouverture de 16 fichiers simultanément. Ce nombre a été fixé arbitrairement car cela s'avérait suffisant à ce moment. Néanmoins, il s'agit là d'un nombre que l'on peut modifier facilement. Signalons toutefois, que la relation suivante est obligatoire pour être compatible avec la façon dont est écrit le programme. Si LC représente la longueur de la table des canaux et LT celle de la table des fichiers, on doit avoir

LT = 2*LC

En effet, on pourra avoir , dans le cas où chaque utilisateur travaille avec un fichier indexé différent, à ouvrir 2 * LC fichiers simultanément.

6.5.2.2. Structure générale du programme gérant la mémoire de masse.

La structure générale du programme gérant les fichiers est donc la suivante:

A l'initialisation du système
- initialiser la table des canaux
- initialiser la table des fichiers

Ensuite, le programme itère sur le traitement suivant:
- lecture d'un bloc sur le réseau
- en fonction du code reçu, déclenchement du traitement approprié
- renvoi à l'utilisateur du bloc contenant la réponse à sa requête.

Les seules actions élémentaires autorisées sur les fichiers par le système UCSD sont les suivantes:

CREATION
OUVERTURE SO LECTURE d'un bloc de 512 bytes en fonction de son numéro relatif
ECRITURE d'un bloc de 512 bytes en fonction de son numéro relatif
FERMETURE

6.5.2.3. Ordres implémentés

6.5.2.3.1. Introduction.

Nous allons maintenant passer en revue l'ensemble des primitives implémentées en vue de gérer les fichiers de la mémoire de masse. Comme nous l'avons vu, la requête d'un utilisateur se traduit par l'envoi sur le réseau d'un bloc contenant le code identifiant sa demande suivi et un

certain nombre de paramètres.

Pour chaque primitive, nous détaillerons le contenu des informations contenues dans le bloc adressé au gérant de la mémoire de masse (BLOC REQUETE) ainsi que celles contenues dans le bloc renvoyé à la station terminale (BLOC REPONSE).
Notons que certains des ordres implémentés sont communs aux deux types de fichiers (entendons par là que la requête est identifiée par un même code) alors que d'autres sont spécifiques à chaque type.
Les ordres actuellement disponibles pour l'utilisateur sont les suivants:

- Ordres communs.

  OPEN CLOSE

- Ordres spécifiques aux fichiers séquentiels.

  READSEQ
  WRITESEQ

  Ordres spécifiques aux fichiers indexés.

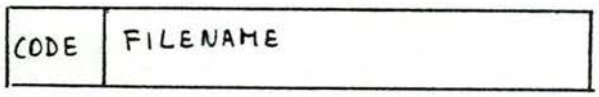
  WRITEINDX
  READINDX
  READNEXT
  READMOD
  REWRITE
  DELETE

Pour chacune des primitives mise en oeuvre, nous adopterons le schéma général suivant de description:
- Structure du bloc requête.
- Structure du bloc réponse.
- Exposé succint de la fonction réalisée.
- Explication détaillée du traitement sous forme d'arbre programmatique (cfr.annexe D).

6.5.2.3.2. Ordres communs aux deux types de fichier

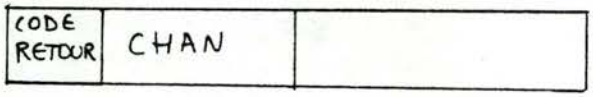
---- O P E N F I L E ---

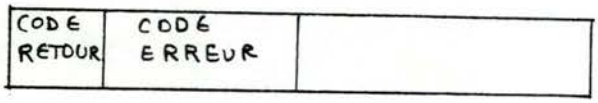
Bloc requête



Bloc réponse

I





Fonction

La procédure a pour effet d'aller ouvrir le fichier de nom FILENAME en vue de permettre à l'utilisateur d'y faire des opérations d'entrées/sorties.
Si FILENAME comporte l'extension " INDX ", cela signifie que le fichier à ouvrir est de type indexé et que par conséquent, il faut ouvrir deux fichiers: le fichier d'indexe et le fichier de données.
Dans le cas contraire, il s'agit d'un fichier séquentiel et il n'y a donc qu'une seule ouverture de fichier.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CHAN = numéro de canal attribué à l'utilisateur (conformément au principe exposé parag. 3.5.3.)

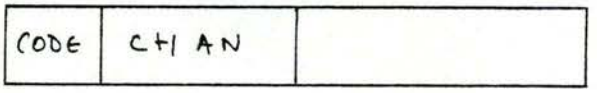
Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

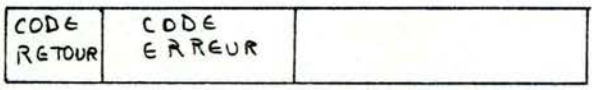
CODE ERR = entier spécifiant le type d'erreur

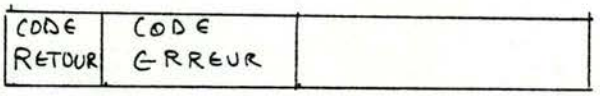
— C L O S E F I L E ——

Bloc requête



Bloc réponse





Fonction

La procédure a pour effet de libérer le canal de numéro CHAN attribué à l'utilisateur. Logiquement, cette requête équivaut à une demande de fermeture du fichier et après exécution de cet ordre, ledit fichier est inaccessible à l'utilisateur. (Physiquement, il se peut que le fichier ne soit pas réellement fermé. Ce sera le cas d'un fichier partageable.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur

6.5.2.3.3. Ordres spécifiques aux fichiers séquentiels

---- W R I T E S E Q ----

Bloc requête

CODE	CHAN	DATA LGTH	DATA

Bloc réponse

CODE RETOUR	CODE ERREUR	

CODE RETOUR	CODE ERREUR	

Fonction

La procédure a pour effet d'aller écrire séquentiellement DATA-LGTH bytes à la suite de ceux déjà écrits dans le fichier associé au canal de numéro CHAN. Les bytes à écrire sont contenus dans DATA.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur

---- R E A D S E Q ----

Bloc requête

CODE	CHAN	DATA LGTH	

Bloc réponse

CODE RETOUR	CODE ERREUR	DATA-RD	DATA

Fonction

La procédure a pour objet de lire dans le fichier séquentiel associé au canal de numéro CHAN les DATA-LGTH bytes suivant ceux qui ont déjà été lus.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

DATA-RD = nombre de bytes effectivement lus

DATA = zone mémoire contenant les données lues

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur

6.5.2.3.4. Ordres spécifiques aux fichiers indexés

Les ordres suivants utilisent tous une même procédure récursive qui a pour objet de gérer l'arbre d'indexe. Il est par conséquent intéressant d'expliquer en quoi consiste cette procédure avant d'examiner les traitements spécifiques à chacun des ordres.
Globalement, l'exécution de cette routine récursive consiste à

- parcourir l'arbre d'indexe depuis la racine afin d'y rechercher une valeur de clé passée comme référence lors de l'appel.
- Déclencher un éventuel traitement sur le fichier des données lorsque le niveau inférieur de l'arbre est atteint.
- Effectuer un parcours ascendant de l'arbre en vue de regagner le sommet.

Au cours de cette troisième étape, il se peut qu'un traitement soit à effectuer consécutivement à l'adjonction ou la suppression d'une valeur de clé dans la structure d'indexe. Examinons ci-dessous ces deux éventualités.

- 1. Insertion d'une valeur de clé

La politique d'insertion est la suivante.
Elle consiste à insérer la nouvelle valeur de clé dans le bloc dont la borne supérieure est > à la valeur à insérer et dont la borne inférieur est < à la valeur à insérer. Lorsqu'on insère un élément dans un bloc, deux cas peuvent se produire.

- Le nouveau bloc n'est pas plein.
  Dans ce cas, il suffit d'ajouter la nouvelle valeur de clé à la bonne place dans le bloc.
  Supposons, par exemple, que l'on veuille ajouter la valeur de clé 25 et que la situation au dernier niveau de l'arbre (à savoir le niveau n) soit celle schématisée fig. 15

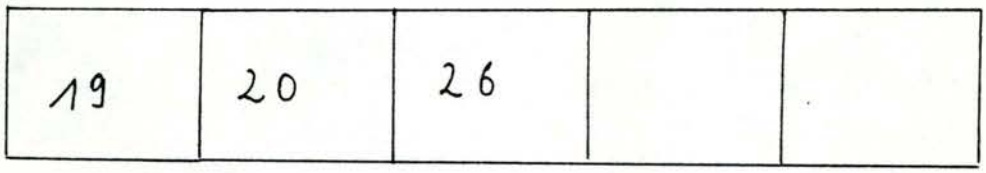


Fig. 15

L'effet produit par l'ajonction de cette nouvelle valeur reste limité au niveau n et est représenté fig. 16

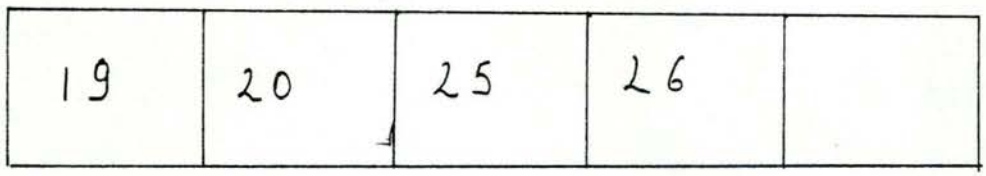


Fig. 16

-

N.B.: Il n'y a pas de conséquence aux autres niveaux car on ne modifie jamais la borne supérieure d'un bloc.

- Le nouveau bloc est plein.
  Dans ce cas, on ne se contente pas d'ajouter à la bonne place la nouvelle valeur, mais on décide de répartir sur deux blocs l'ensemble des valeurs comprises dans le bloc plein.
  Supposons que l'on veuille insérer la valeur de clé 25 et que la situation au niveau n soit

celle schématisée fig. 17

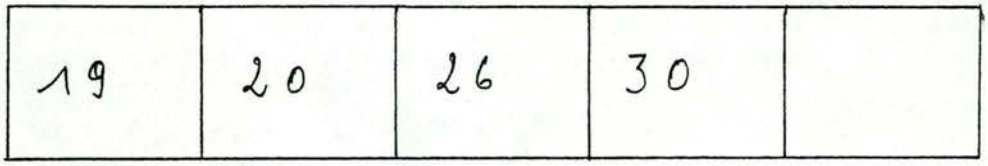


Fig. 17

-

L'adjonction de la nouvelle valeur provoque la création d'un nouveau bloc au niveau n qui elle-même provoque l'ajout d'une nouvelle valeur de clé au niveau n-1. Ce phénomène est décrit fig. 18

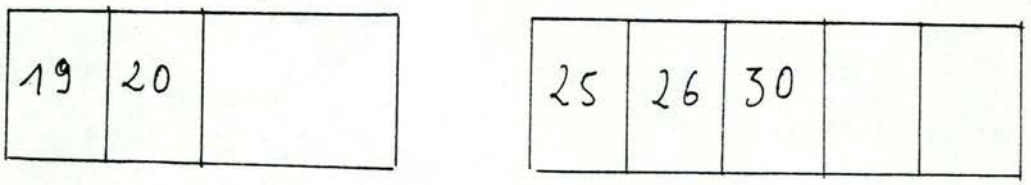


Fig. 18

-

Attirons également l'attention sur le fait que conjointement, il faut également veiller à la gestion des pointeurs. Cela apparaît de façon significative sur la fig.

Le processus qui vient d'être décrit est évidemment susceptible de se répercuter à d'autres niveaux. En toute généralité, l'adjonction d'un bloc au niveau i provoque l'adjonction d'un bloc au niveau i-1. Cette gestion est assurée automatiquement par le jeu de la récursivité. La seule chose à tester au retour de la procédure est, si il y a lieu de créer un nouveau niveau à l'arbre. Ce

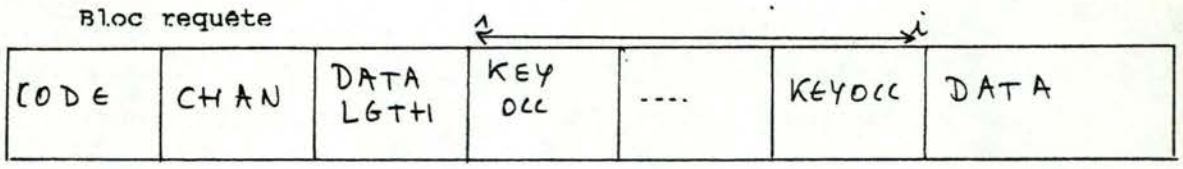
traitement particulier est assuré par une autre procédure.

- 2. Suppression d'une valeur de clé

La politique de suppression devrait être analogue à celle d'insertion si on voulait garantir un bon taux d'occupation du fichier d'indexe. Elle consisterait donc à détecter le fait que deux blocs consécutifs soient remplis à moitié et à réaliser une fusion de ces deux blocs. Actuellement, pour des raisons de simplicité, un bloc n'est réellement supprimé que lorsqu'il est entièrement vide.

---- W R I T E I N D X ----

Bloc requête



Bloc réponse

CODE RETOUR	CODE ERREUR	

Fonction

La procédure a pour effet d'aller écrire dans le fichier de type indexé associé au canal de numéro CHAN l'enregistrement compris dans la zone DATA dont la longueur est DATA-LGTH. KEYOCC [i] représente le nombre d'occurences de la i ième clé dans l'enregistrement DATA.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur

---- R E A D I N D X ----

Bloc requête

CODE	CHAN	DATA-LGTH	KEY-NO	KEY-LGTH	KEY-VAL

Bloc réponse

CODE RETOUR	CODE ERREUR	DATA RD	DATA

Fonction

La procédure a pour effet d'aller lire dans le fichier indxé associé au canal de numéro CHAN l'enregistrement dont la KEY-NO ième clé (de longueur KEY-LGTH) a pour valeur KEY-VAL. Si la clé n'est pas identifiante, c'est le premier article satisfaisant à la condition qui est lu.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

DATA-RD = nombre de bytes effectivement lus

DATA = zone mémoire contenant les données lues

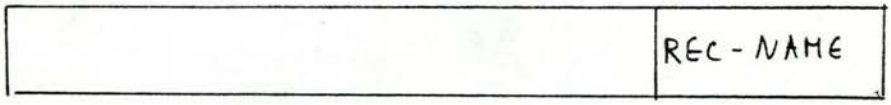
Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

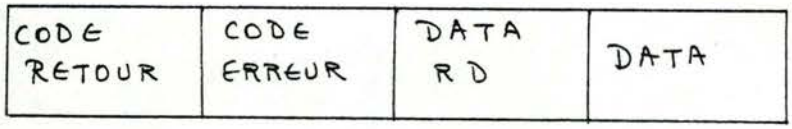
CODE ERR = entier spécifiant le type d'erreur

---- R E A D N E X T ----

Bloc requête



Bloc réponse



Fonction

Cette procédure permet de lire le fichier associé au canal de numéro CHAN séquentiellement sur base des valeurs croissantes d'une clé. Elle a ppur objet d'aller lire l'enregistrement dont la valeur de la KEY-NOième clé est > (clé identifiante) ou > (clé non identifiante) à celle de celui qui a été lu précédemment. KEY-LGTH contient la valeur de la longueur de clé.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur

DATA-RD = nombre de bytes effectivement lus

DATA = zone mémoire contenant les données lues

---- R E A D M O D ----

Bloc requête

	REC-NAME

Bloc réponse

CODE RETOUR	CODE ERREUR	DATA RD	DATA

Fonction

Cette procédure a exactement la même fonction que READINDX, à la seule différence près, que dans ce cas, la lecture est effectuée avec l'intention d'éventuellement modifier l'article. Si l'article à lire existe, il ne sera effectivement lu que s'il ne fait pas l'objet d'un blocage par un autre processus (cfr.) Auquel cas, il sera lu et bloqué et ne sera disloqué que lorsqu'un ordre rewrite sera demandé par l'utilisateur. Plusieurs enregistrements peuvent être bloqués simultanément; il convient d'associer à ces articles un nom les identifiant, c'est le rôle du paramètre REC-NAME.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

DATA-RD = nombre de bytes effectivement lus

DATA = zone mémoire contenant les données lues

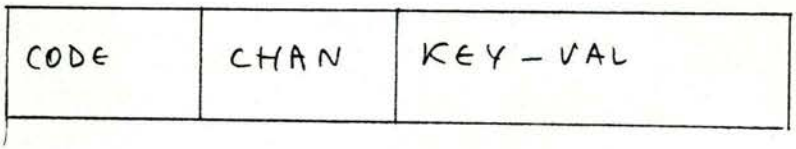
Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

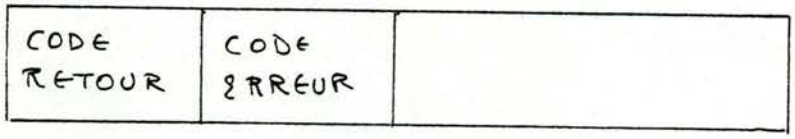
CODE ERR = entier spécifiant le type d'erreur

---- R E W R I T E ---

Bloc requête



Bloc réponse



Fonction

La procédure a pour objet de détruire (dans le fichier associé au canal de numéro CHAN) l'enregistrement dont la valeur de clé primaire vaut KEY-VAL.
Cette opération ne sera évidemment effectuée que si l'article existe et qu'il ne fait l'objet d'aucun blocage.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur
(cfr. liste des erreurs).

---- D E L E T E ----

Bloc requête

CODE	CHAN	KEY-VAL

Bloc réponse

CODE RETOUR	CODE ERREUR	

Fonction

La procédure a pour objet de détruire (dans le fichier associé au canal de numéro CHAN) l'enregistrement dont la valeur de clé primaire vaut KEY-VAL. Cette opération ne sera évidemment possible que si l'article existe bien et qu'il ne fait l'objet d'aucun blocage.

Si la requête était correcte et que tout s'est bien passé, le programme utilisateur reçoit un bloc de type I:

CODE RETOUR = 0

CODE ERREUR = 0

Dans le cas contraire, le programme utilisateur reçoit un bloc de type II:

CODE RETOUR = 1

CODE ERR = entier spécifiant le type d'erreur

CHAPITRE 7

PRIMITIVES DE COMMUNICATION

7.1. PRIMITIVES DE COMMUNICATION

Les primitives de communication implémentées sont en fait excessivement simples car utilisant des outils pré-existants. Il suffisait en effet seulement d'utiliser deux routines écrites en assembleur qui ont respectivement pour fonction l'envoi et la réception d'un bloc sur le réseau.
Le texte de ces deux routines (CBSEND et CBRECV) est fourni en annexe D.
Le bloc transitant sur le réseau doit avoir le format suivant:

	CORR	LOCAL	BLK LOW	BLK HIGH	MESSAGE

- LOCAL : adresse de la station sur laquelle s'exécute la procédure (c.a.d. adresse de la station qui envoie si on se trouve dans la procédure CBSEND et adresse de la station qui reçoit si on se trouve dans la procédure CBRECV).
- CORR : adresse de la station correspondant avec celle sur laquelle s'exécute la procédure (c.a.d. dans la procédure CBSEND, adresse de la station à laquelle le message est destiné et dans la procédure CBRECV, adresse de la station dont on attend un message).
- BLK-LOW et BLK-HIGH : Ces deux bytes représentent respectivement le byte de poids fort et le byte de poids faible d'un entier contenant la longueur du message.
- MESSAGE : Texte proprement dit du message transmis.

Sur cette base, il s'agissait donc pour répondre aux besoins de l'application, de mettre au point des primitives répondant aux spécifications suivantes:

1. Primitive d'envoi d'un message

   SEND MESS (MESSAGE, MESS-LGTH, CORR, LOCAL).

   Cette procédure a pour effet d'envoyer le message MESSAGE de longueur MESS-LGTH à destination de la station d'adresse CORR à partir de la station d'adresse LOCAL.

2. Primitive de réception d'un message

   RECVMESS (MESSAGE, MESS-LGTH, CORR, LOCAL).

Cette procédure a pour effet d'assurer la réception par la station d'adresse LOCAL dans une zone mémoire MESSAGE de longueur MESS-LGTH le message envoyé par la station d'adresse CORR.

Tout le problème de l'intégrité des données transmises est résolu au niveau du protocole (cfr.parag. 3.5.) et ne doit donc plus être pris en considération.

CHAPITRE 8

REALISATION CONCRETE DE L'APPLICATION

8.1. REALISATION

Dans l'état actuel, l'application est réalisée conformément à la première proposition faite au parag. 5.2.2.

Le contrôle de la commande est assuré au niveau du PROCESSEUR P1. Les opérations se déroulent comme suit. Un premier menu est proposé à l'opératrice lui demandant de manifester son intention d'enregistrer une nouvelle commande ou d'arrêter le processus. Les opération de vérification comportent deux aspects.
Un premier contrôle porte sur l'identité du client. A cet effet, un second menu est proposé à l'opératrice lui donnant la possibilité d'interroger le fichier des clients sur base de leur numéro client ou de leurs nom et prénom. Consécutivement au résultat de sa requête, l'opératrice peut prendre certaines décisions en tapant un caractère de code significatif. Elle peut notamment, à ce moment, décider d'aller mettre à jour le fichier des clients. Un programme de gestion d'écran a été mis au point en vue de permettre une introduction aisée des données.
Le second contrôle porte sur le caractère valide du corps de la commande. L'opératrice traite la commande ligne par ligne. Chaque ligne contrôlée et validée par l'opératrice est envoyée au processeur P2.

Au terme du traitement de la commande sur le processeur P1, un message est envoyé à destination du PROCESSEUR P2 lui signifiant si les éléments qu'il vient de recevoir sont à enregistrer ou non. Si la commande a été jugée recevable par l'opératrice, le processus d'enregistrement de la commande est alors déclenché sur le processeur P2. Celui-ci consiste essentiellement à

- enregistrer la commande dans le fichier des commandes créer l'expédition correspondante
- mettre à jour le fichier d'état des stocks.

Ces opération réalisées, le numéro de la commande nouvellement créée est envoyé au PROCESSEUR P3 dont l'unique fonction est de constituer une série de cent commandes et d'envoyer un message au PROCESSEUR P4 en vue d'y déclencher le processus d'ordonnacement.

Les spécifications précises de ce traitement n'étant pas fournies, celui-ci n'a d'autre objet que

- d'ajouter, dans le fichier des bons de livraison, la série de bons relative aux cent commandes qui viennent d'être traitées
- d'éditer les bons par série
- d'éditer les bons par casier

sans faire intervenir de critère de localisation physique des articles dans le stock.

Le PROCESSEUR P5 a pour objet

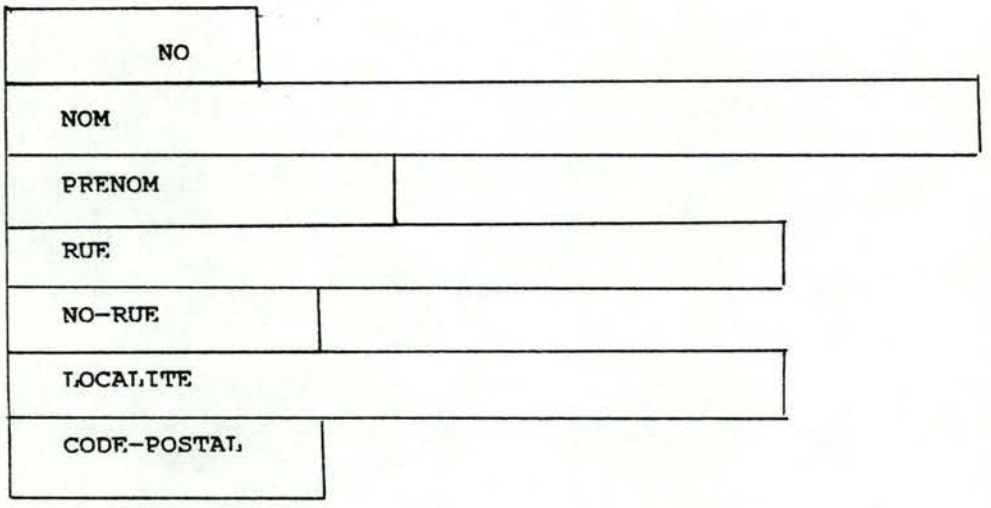
- d'enregistrer une livraison fournisseur
- de mettre à jour le stock
- de déclencher le traitement des commandes différées en attente pour les produits rentrés

Celles-ci sont traitées en priorité par rapport aux commandes du jour par le simple fait que la quantité nécessaire pour satisfaire l'ensemble des commandes différées est réservée d'office.

Le PROCESSEUR P6 permet au magasinier d'afficher le contenu du bon de livraison relatif à la commande qu'il est en train de reconstituer. S'il a suffisamment dans son casier pour constituer le colis, le processus de facturation peut être déclenché. Dans le cas contraire, une procédure d'exception peut être invoquée dont l'objet est d'effectuer la correction du système de façon à ce que celui-ci reflète au maximum la situation réelle.

8.2. DESCRIPTION DES FICHIERS

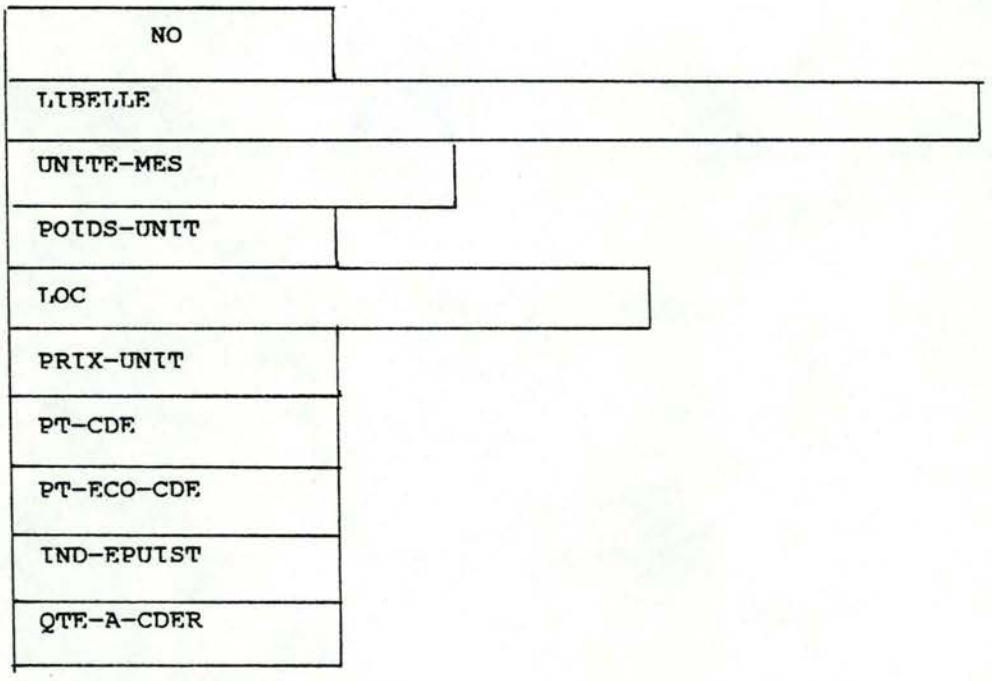
8.2.1. FICHIER CLIENT



Clé primaire : NO

Clé secondaire : NOM+PRENOM

8.2.2. FICHIER PRODUIT



Clé primaire : NO

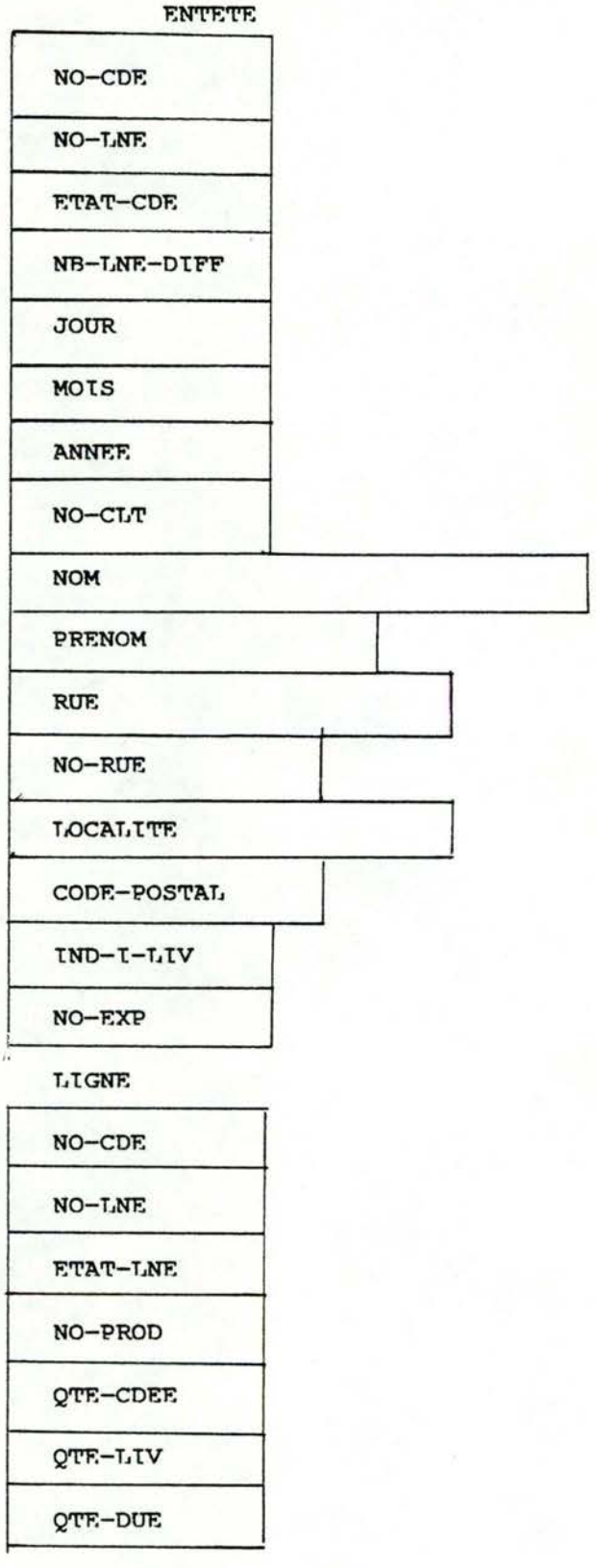
Clé secondaire : LIBELLE

8.2.3. FICHIER STOCK

NO
INDIC-EPUIST
QTE-DISP
QTE-DIFF

Clé primaire : NO

8.2.4. FICHIER COMMANDES



Clé primaire : NO-CDE+NO-LNE

Clé secondaire : ETAT-LNE+NO-PROD

Clé secondaire : NO-EXP

8.2.5. FICHIER EXPEDITION

ENTETE

NO-EXP
NO-LIGNE
NO-CDE
JOUR
MOIS
ANNEE
POIDS-COLIS
MONTANT

Clé primaire: NO-EXP+NO-LIGNE

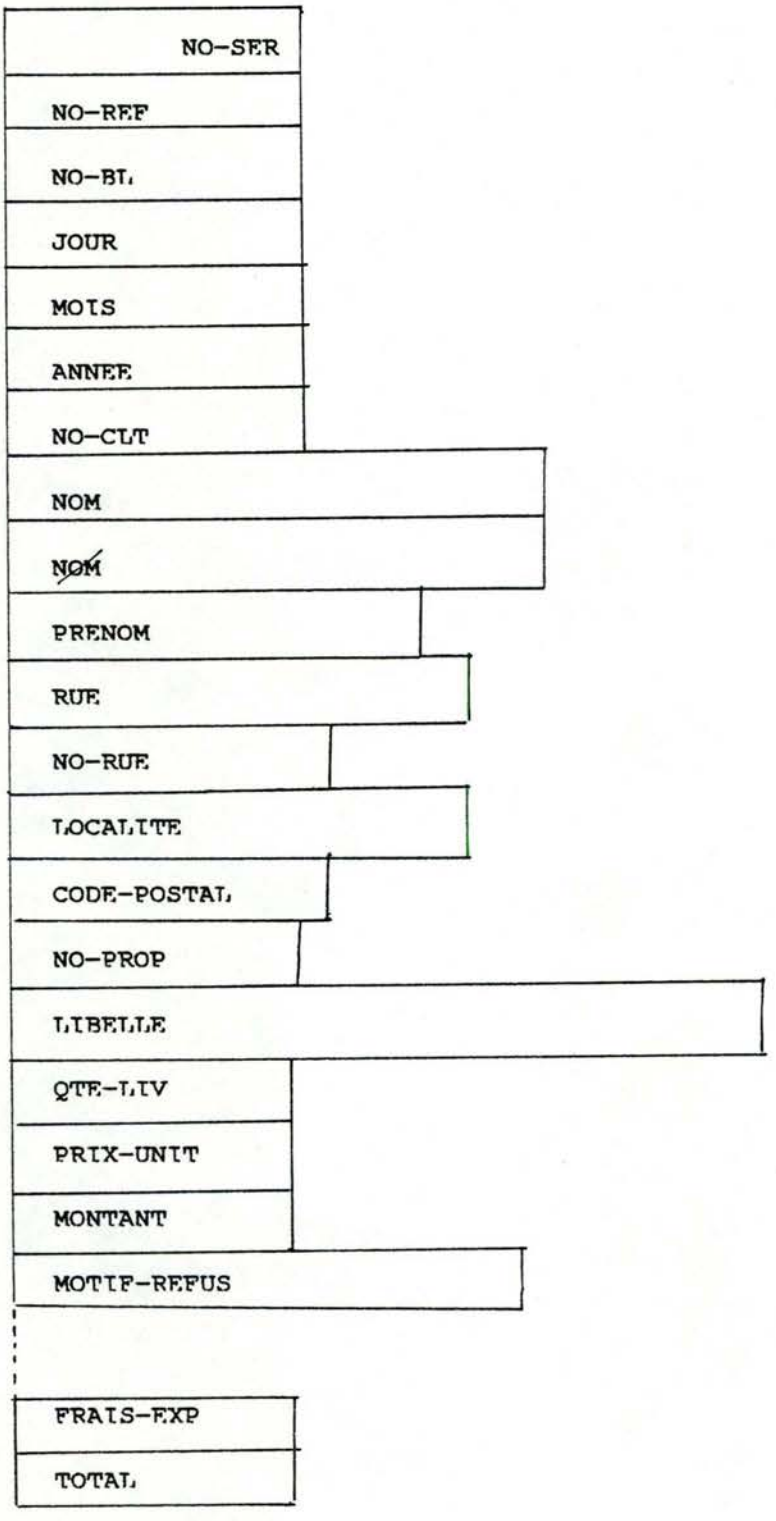
Clé secondaire : NO-CDE

LIGNE

NO-EXP
NO-LNE
NO-PROD
LIBELLE
PRIX-UNIT
POIDS-UNIT
QTE-A-LIV
MOTIF-REFUS

Clé primaire:NO-EXP+NO-LIGNE

8.2.6. FICHIER BONLIV



Clé primaire : NO-SER+NO-REF

CONCLUSION

Le but de ce mémoire était d'étudier les possibilités offertes ainsi que les contraintes imposées par l'implémentation d'une application de gestion sur un réseau local.
Comme on a pu le constater, on s'est un peu éloigné de l'objectif premier car une large part du travail a été consacrée à la réalisation de primitives d'accès pour une mémoire auxiliaire. Ce problème présentait deux aspects essentiels. D'une part, il convenait d'établir une structuration des données; d'autre part, il était question de gérer de manière satisfaisante les accès concurrents aux fichiers partageables de cette mémoire.
Les options importantes prises au niveau de l'application proprement dite, ainsi que l'élaboration de l'architecture d'implémentation ont été mises en évidence et justifiées. Malheureusement, pour des contraintes d'ordre essentiellement temporel, la mise au point de jeux de tests significatifs validant l'application n'a pu être réalisée comme on était en droit de le souhaiter au départ.

En conséquence, si on établit un bilan global de la situation telle qu'elle se présente au terme du projet, il y a lieu de dégager les éléments suivants.

- Le caractère faisable de l'application dans une optique réseau a été prouvé et les principaux problèmes rencontrés exposés.
- L'ensemble des fonctions systèmes supportant l'application a été réalisé.
- Un point qu'il conviendrait de développer pour mettre un point final au projet est l'examen des problèmes de sécurité et de fiabilité.

B I B L I O G R A P H I E

[1] ANCEAU F., Systèmes fonctionnellement distribués, Actes du Congrès de l'AFCET, 1979, Editions Hommes et Techniques.

[2] BARON R.J., SHAPIRO L.G., Data structure and their implementation, Van Nostrand Reinhold, University Computer Science Gering.

[3] BODART F., Concepts, méthodes et outils de l'analyse fonctionnelle, notes de cours de 1 ère licence, FNDP,1978-79.

[4] BODART F., PIGNEUR Y., A model and a language for functional specification and evaluation of information system dynamics, Oxford, april 1979.

[5] BOGGS D.R., METCALFE R.M., Ethernet: Distributed Packet Switching for local Computer Network,Palo Alto Research Center, novembre 1975.

[6] BOUHOT J.P., FRANCE-LANORD B., LUSSATO B., La micro-informatique: introduction aux systèmes répartis, Editions d'informatique, 1974.

[7] BRES J., IDMS s'ouvre à la répartition, 01 informatique,no.145, nov.1980.

[8] DATE C.J., Locking and recovery in a shared database system: an application programming tutorial, IBM General Products Division, San Jose, California,USA,1979.

[9] DOWN J.P., TAYLOR F.E., Why distributed computing?, NCC Publications,1976..

[10] DUMONT R., Un réseau local intelligent, Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique, FNDP, 1978-79.

[11] EVEREST G.C., Concurrent update Control and database integrity, University of Minnesota, USA.

[12] FARBER D.J., A ring network, Datamation, february 1975.

[13] FERNANDEZ F., FERRAT L. et NGUYEN GIA TOAN et SERGEANT G., Actes du congrès de l'Afcet, 24-27 nov.1980, Editions Hommes et Techniques.

[14] FRASER A.G., A virtual Channel Network, Datamation,february 1975.

[15] TRAIGER I.L., GRAY J.N., GALTIERI C.A., LINDSAY B.G., Transactions and consistency in distributed database systems, IBM Research Division, San Jose Laboratory, California.

[16] MACCHI C., GUILBERT J.F., Teleinformatique: transport et traitement de l'information dans les réseaux et systèmes téléinformatiques, Bordas et C.N.E.T.-E.N.S.T, Paris, 1979.

[17] GRAY J.N., LORIE P.A., PUTZOULU C.R., Granularity of locks in a

large shared data base, Proc.Int.Conf.on VLDB, sept.1975.

[18] HAINAUT J.L., Fichiers et banques de données, Cours de 1ère et 2ème licences, FNDP,1978-80.

[19] JACKSON M.A., Principles of program design, Academic Press London, New York, San Francisco,1975.

[20] LAMOND F., Révolution chez IBM (2ème partie), L'ouverture à l'informatique distribuée, 01 informatique, no 136, nov.1979.

[21] NCC, Management strategy for distributed computing, Oxford Road, Manchester, England, 1979.

[22] NICOUD J.D., Benefits of a working local network, Swiss Federal institute of Technology, Lausanne, Switzerland.

[23] PETITPIERRE C., Mltiplexeur implante sur un réseau, EPFL-Lausanne Section de Calculatrice Digitale.

[24] ROCHFELD A., SPACCAPIETRA S.,Introduction à : Bases de données réparties, Monographie d'informatique de l'AFCET, Groupe de travail AFCET-TTI sur les bases de données, animé par S. SPACCAPIETRA, Editions Hommes et Techniques, 1978.

[25] SOMMER R., Cobus, a firmware controlled data transmission system, Mini and Microcomputer Laboratory, Swiss Federal Institute of Technology, Lausanne, Euromicro, 1976, North Holland Publishing Company.

[26] VERHOFSTAD J.S.M., Recovery Technics for Database systems,BNSR, Toronto,Ontario Canada.

[27] WILLARD D.G., Mitrix: A sophisticated Digital Cable Communication System, Proceeding of the National Telecommunication Conference, novembre 1973.

[28] WILMS P., Les applications réparties dans les réseaux généraux d'ordinateurs, Mathématiques appliquées et informatique, Université scientifique et médicale et institut national polytechnique de Grenoble, Laboratoire associé au CNRS no.7, Rapport de synthèse, mai 1978.

Rapports internes aux Facultés Notre Dame De La Paix (F.N.D.P.)

[29] Automatisation de la gestion des commandes dans une firme de vente par correspondance.

[30] Rapport d'entretien.

[31] Proposition d'automatisation.

BUMP
0 0 3 4 4 2 2 8 2
*FM B16/1981/14/1

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)
INSTITUT D'INFORMATIQUE

IMPLEMENTATION D'UNE APPLICATION DE GESTION SUR UN RESEAU LOCAL

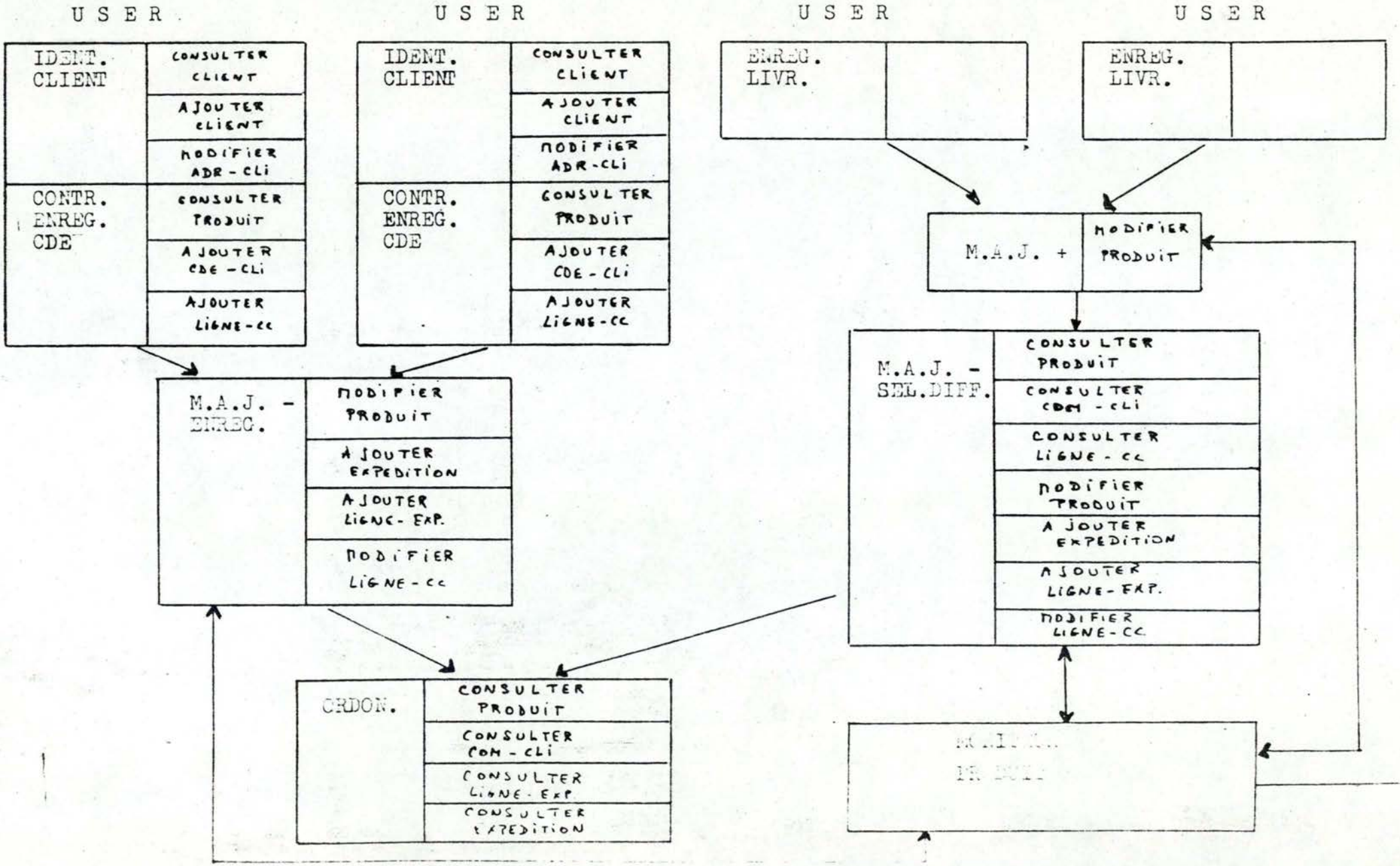
(ANNEXES)

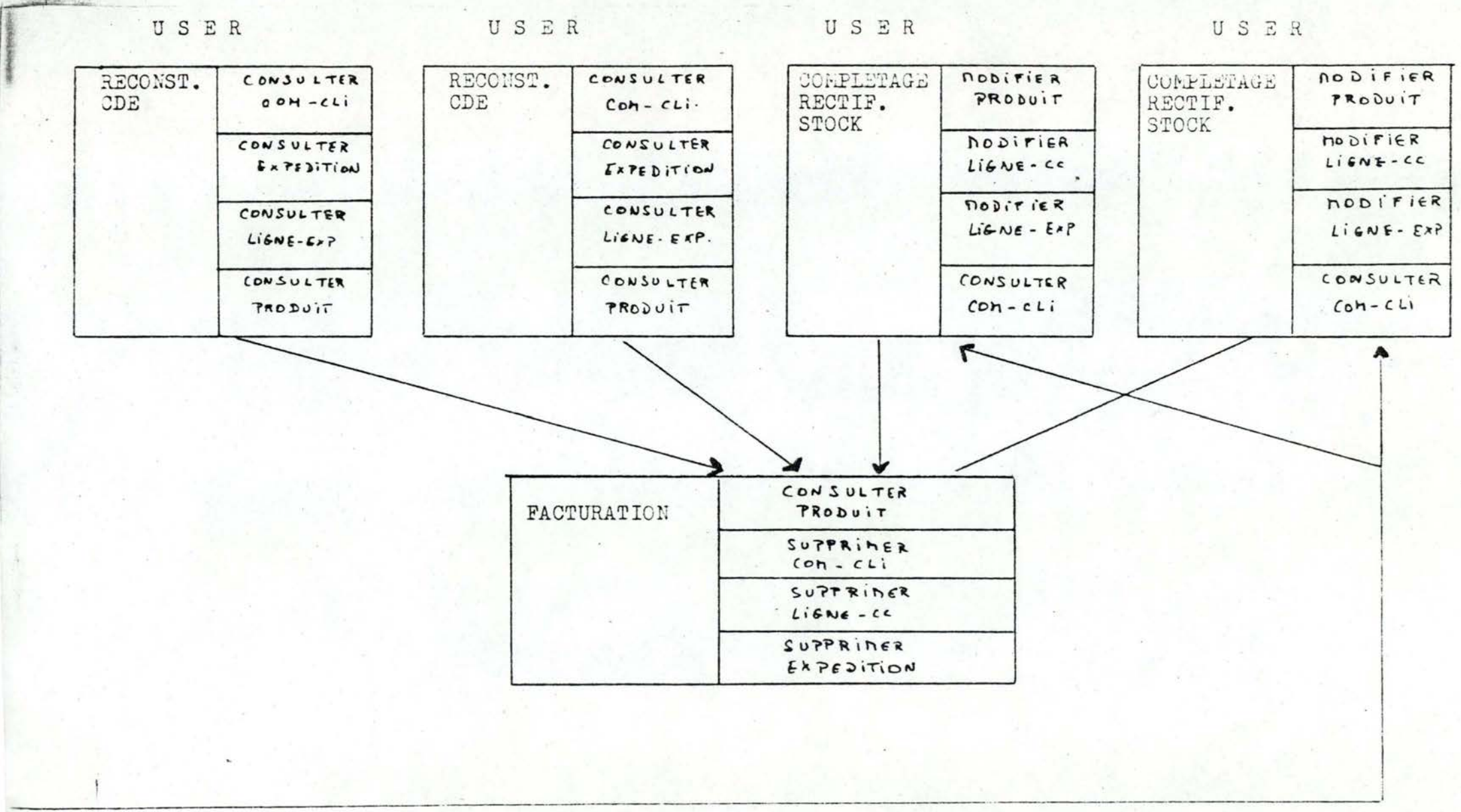
Mémoire présenté par
Christine Vanoirbeek
en vue de l'obtention du titre de Licenciée et Maître en Informatique

Année académique 1980-1981

ANNEXE A

Proposition d'architecture organique

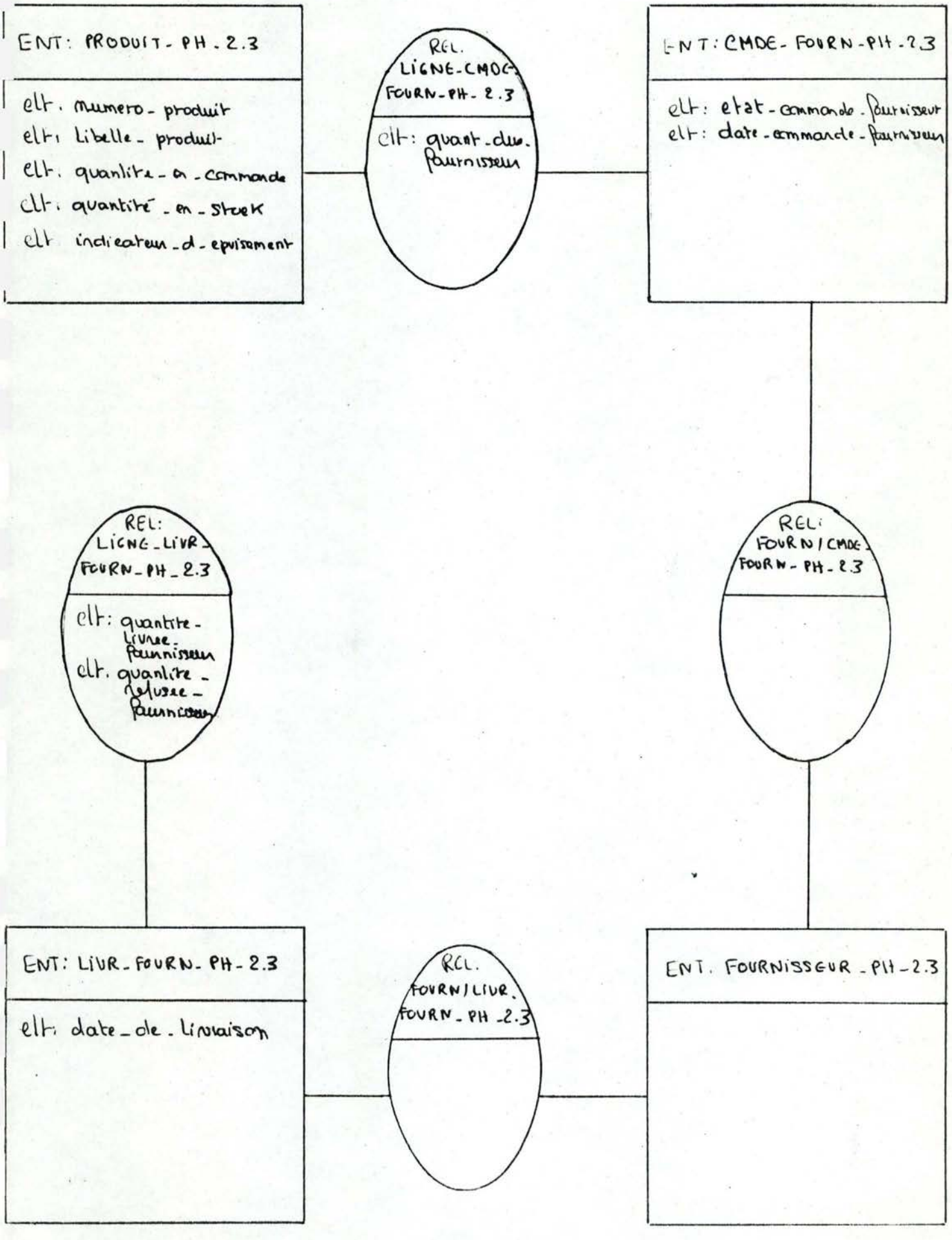

USER
USER
USER
USER
IDENT. CLIENT
CONSULTER CLIENT
AJOUTER CLIENT
MODIFIER ADR-CLI
CONTR. ENREG. CDE
CONSULTER PRODUIT
AJOUTER CDE-CLI
AJOUTER LIGNE-CC
IDENT. CLIENT
CONSULTER CLIENT
AJOUTER CLIENT
MODIFIER ADR-CLI
CONTR. ENREG. CDE
CONSULTER PRODUIT
AJOUTER CDE-CLI
AJOUTER LIGNE-CC
ENREG. LIVR.
ENREG. LIVR.
M.A.J. +
MODIFIER PRODUIT
M.A.J. - ENREG.
MODIFIER PRODUIT
AJOUTER EXPEDITION
AJOUTER LIGNE-EXP.
MODIFIER LIGNE-CC
M.A.J. - SEL.DIFF.
CONSULTER PRODUIT
CONSULTER CDE-CLI
CONSULTER LIGNE-CC
MODIFIER PRODUIT
AJOUTER EXPEDITION
AJOUTER LIGNE-EXP.
MODIFIER LIGNE-CC
ORDON.
CONSULTER PRODUIT
CONSULTER CDE-CLI
CONSULTER LIGNE-EXP
CONSULTER EXPEDITION


U S E R
U S E R
U S E R
U S E R
RECONST. CDE
CONSULTER COM-CLI
CONSULTER EXPEDITION
CONSULTER LIGNE-EXP
CONSULTER PRODUIT
RECONST. CDE
CONSULTER COM-CLI
CONSULTER EXPEDITION
CONSULTER LIGNE-EXP.
CONSULTER PRODUIT
COMPLETAGE RECTIF. STOCK
MODIFIER PRODUIT
MODIFIER LIGNE-CC
MODIFIER LIGNE-EXP
CONSULTER COM-CLI
COMPLETAGE RECTIF. STOCK
MODIFIER PRODUIT
MODIFIER LIGNE-CC
MODIFIER LIGNE-EXP
CONSULTER COM-CLI
FACTURATION
CONSULTER PRODUIT
SUPPRIMER COM-CLI
SUPPRIMER LIGNE-CC
SUPPRIMER EXPEDITION

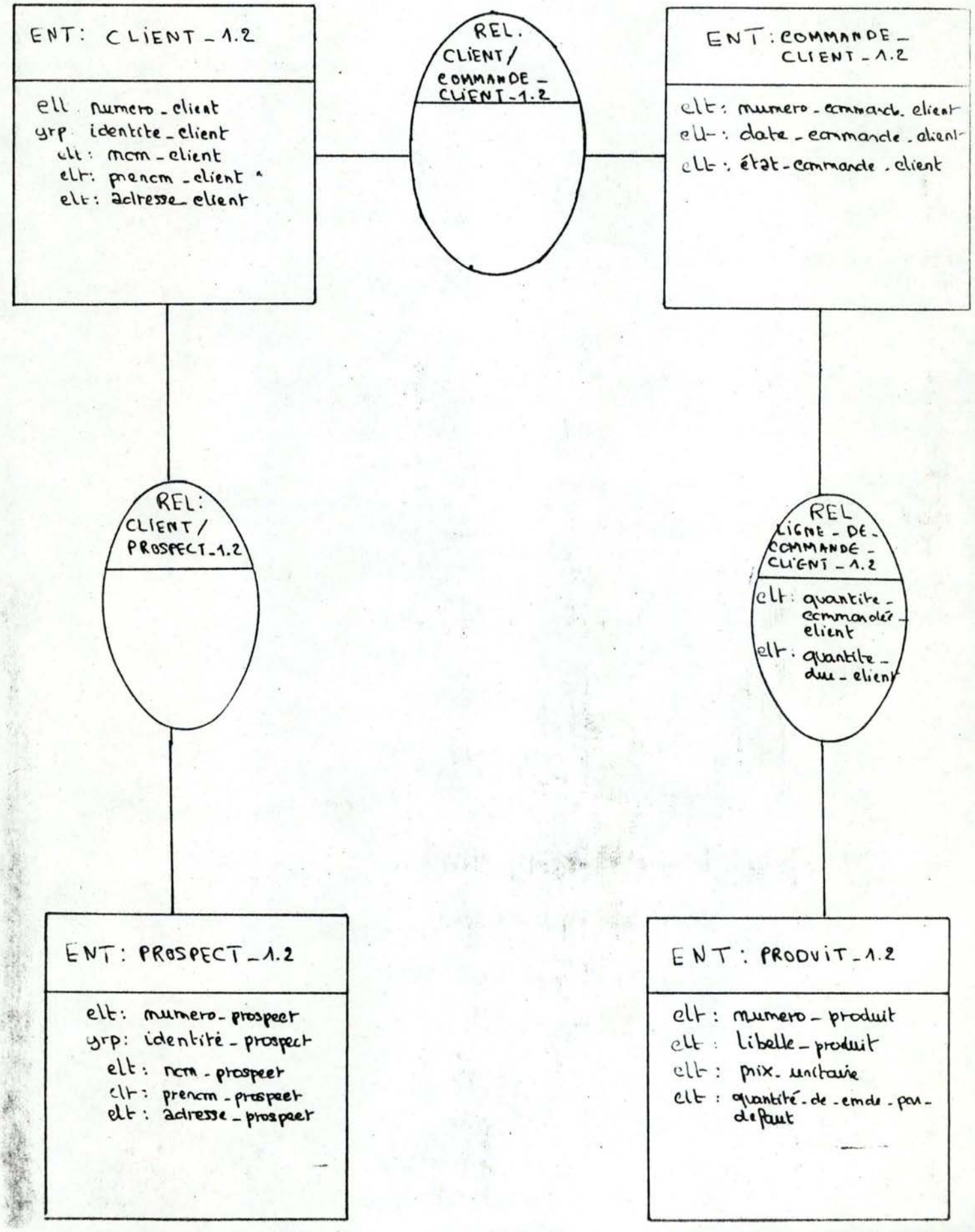
ANNEXE B

Modèle conceptuel par phase

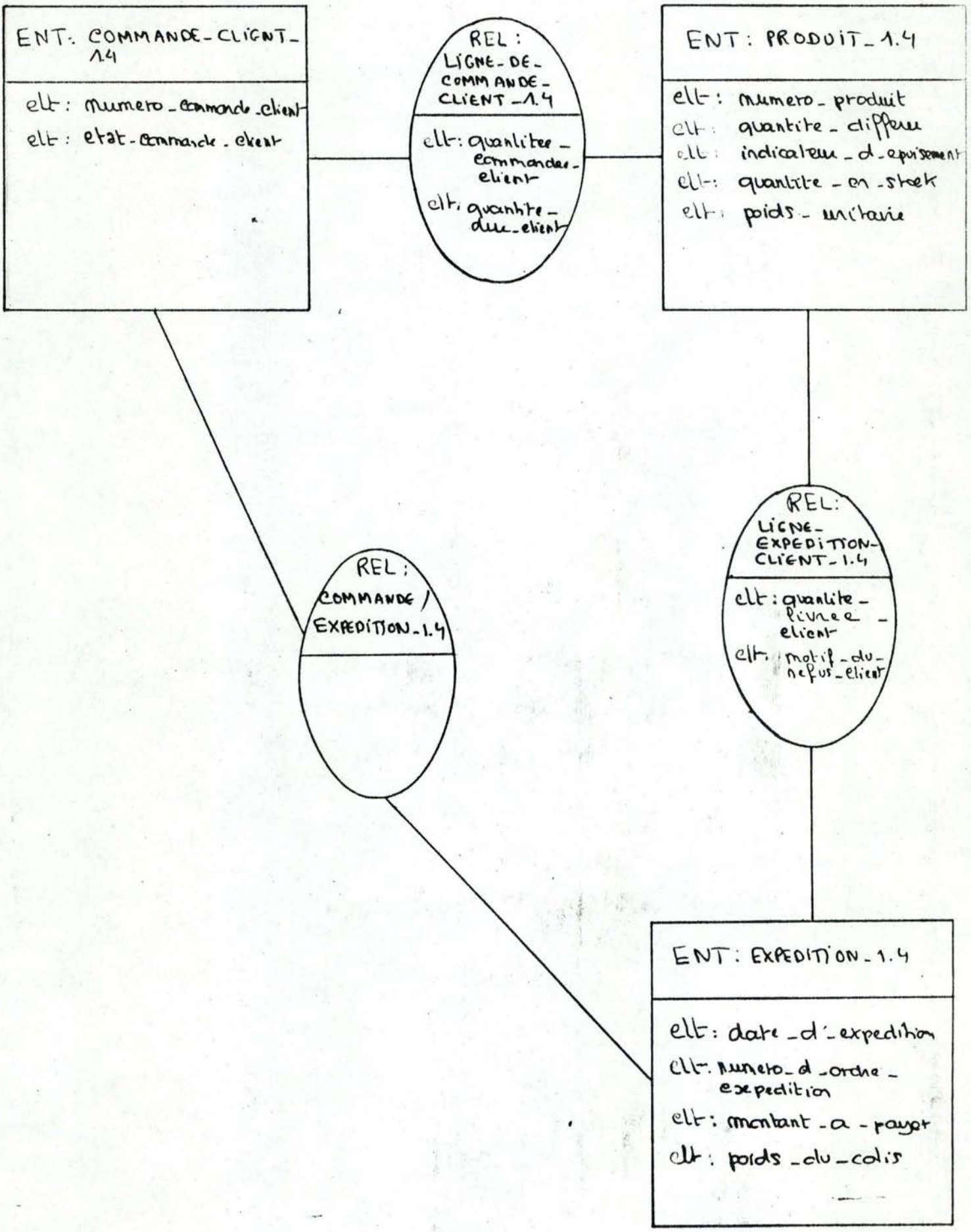
STRUCTURE LOGIQUE PHASE 2.3



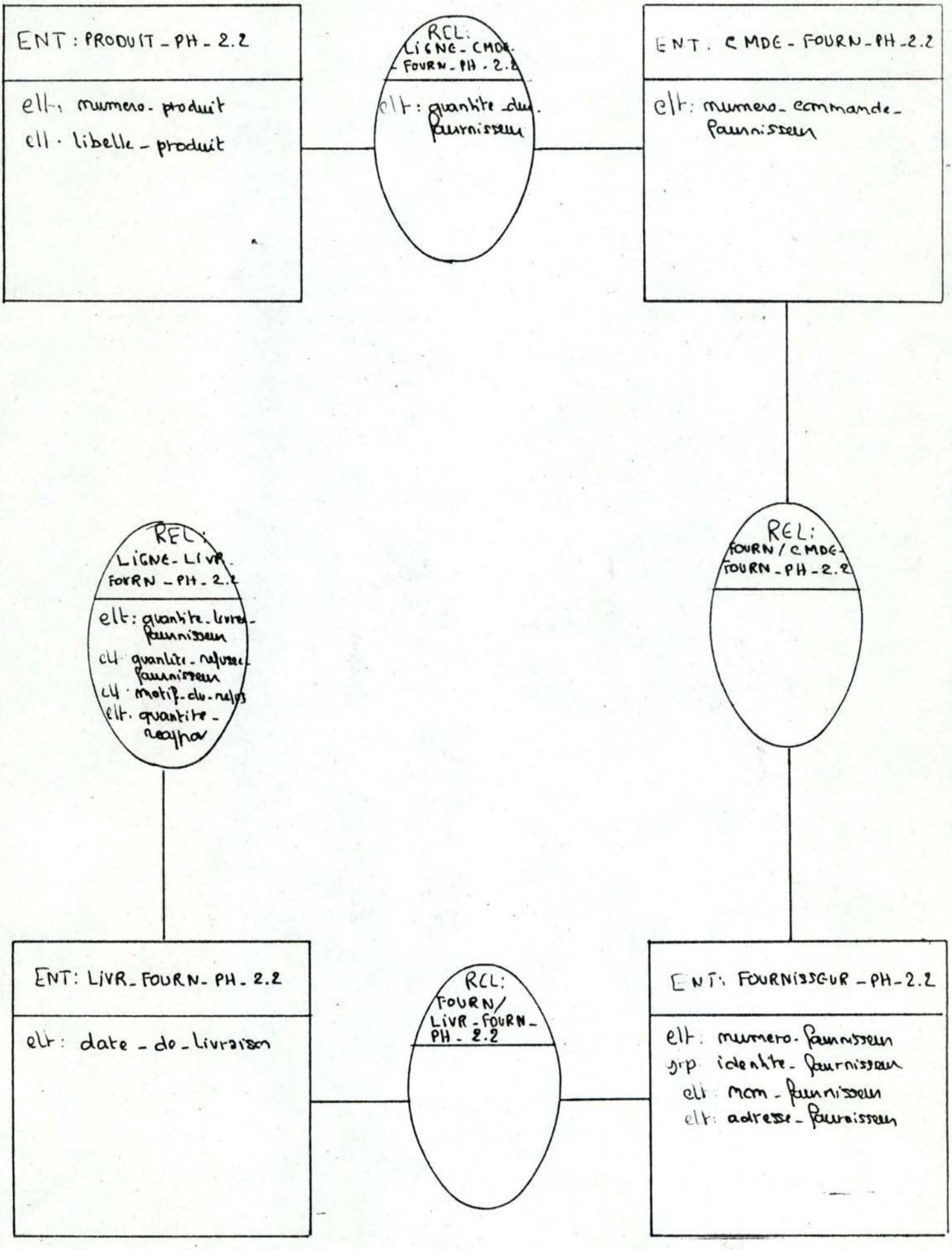
STRUCTURE LOGIQUE PHASE 1.2



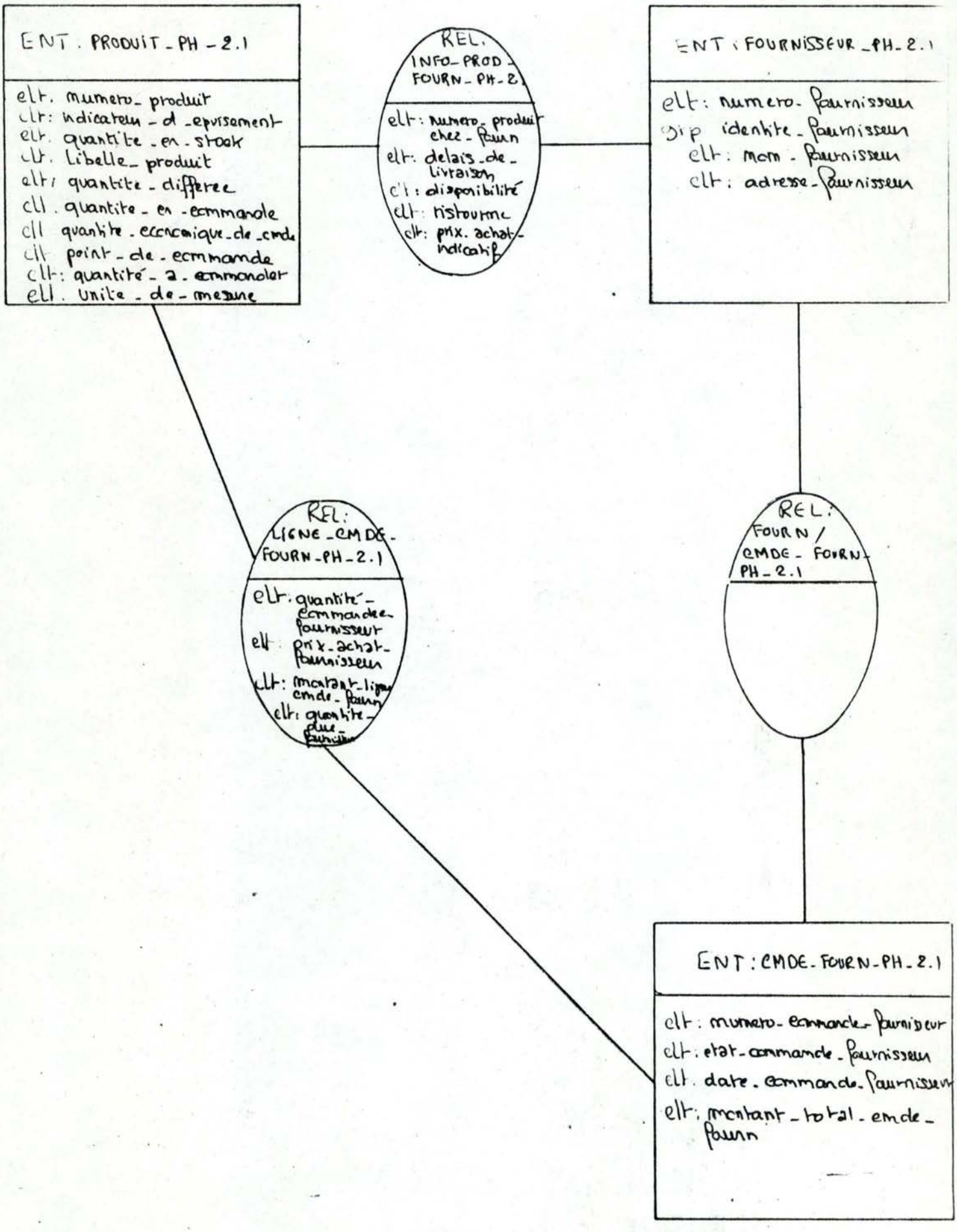
STRUCTURE LOGIQUE PHASE 1.4



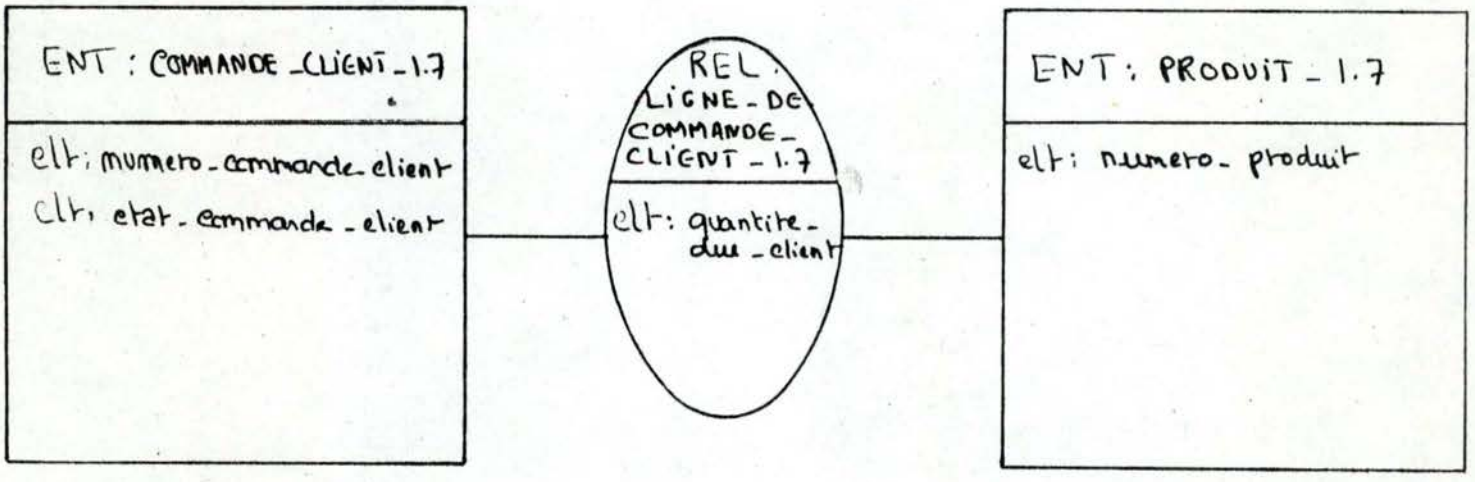
STRUCTURE LOGIQUE PHASE 2.2



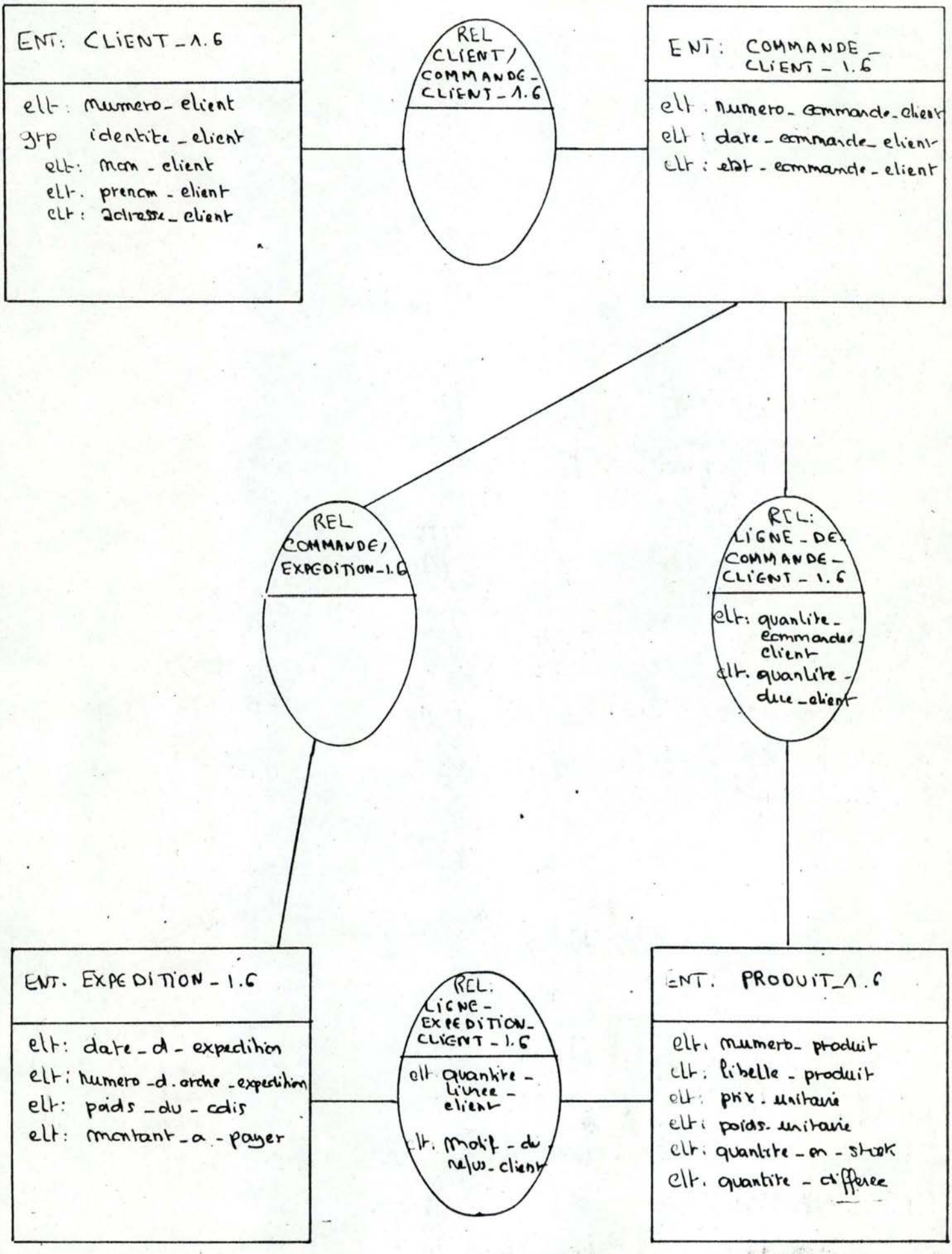
STRUCTURE LOGIQUE PHASE 2.1



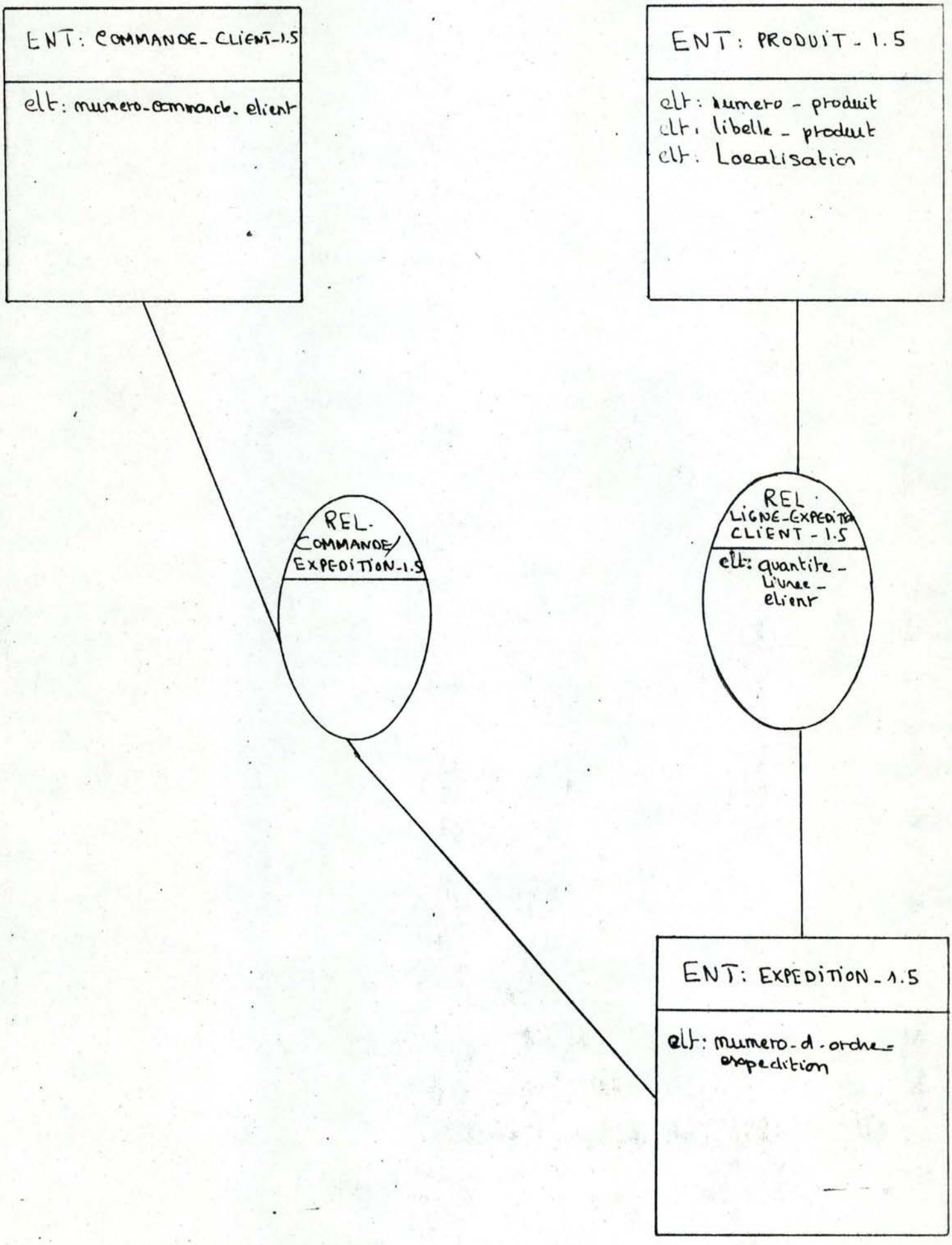
STRUCTURE LOGIQUE PHASE 1.7



STRUCTURE LOGIQUE PHASE 1.6

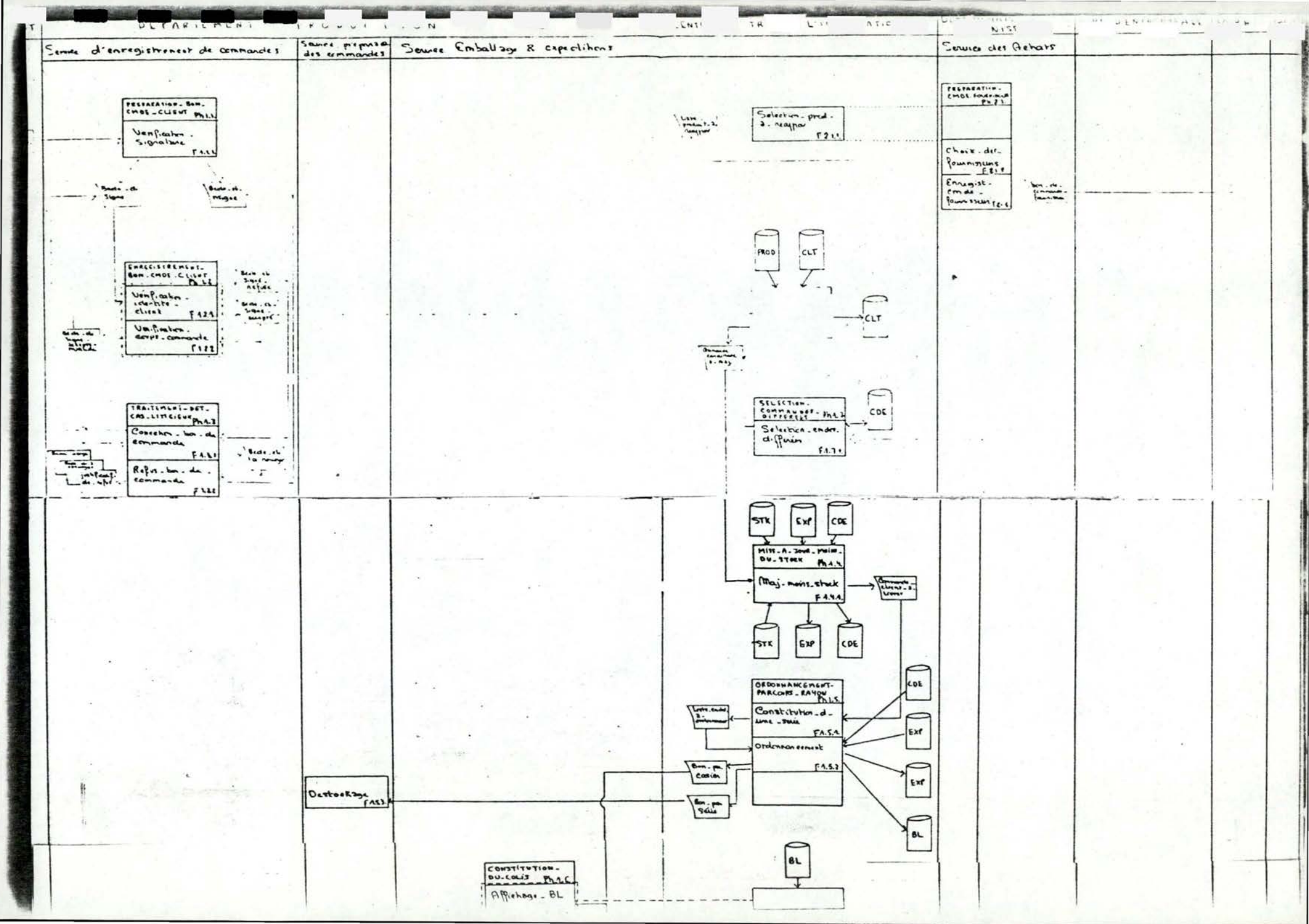


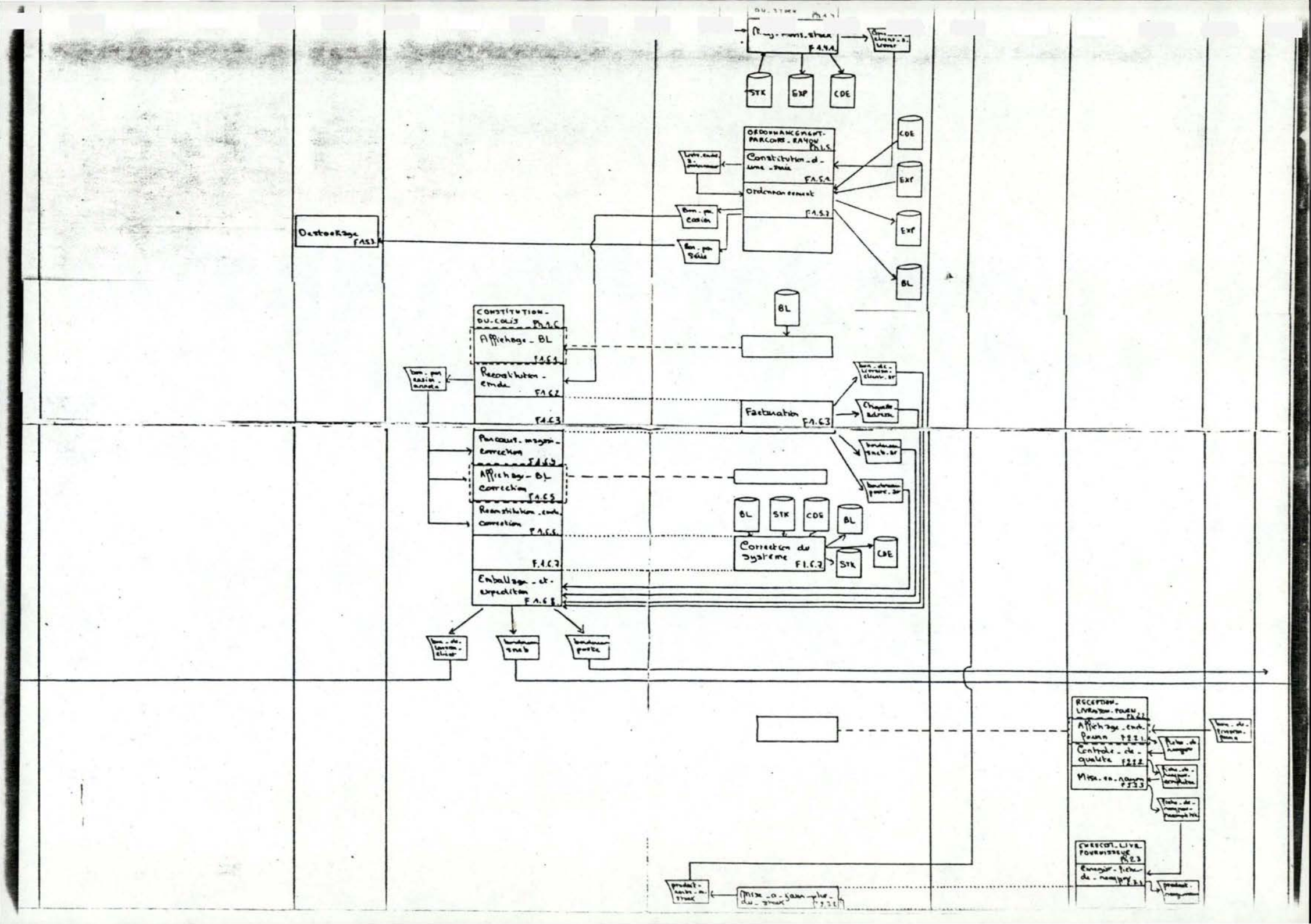
STRUCTURE LOGIQUE PHASE 1.5



ANNEXE C

Diagramme des flux d'information


Service d'enregistrement de commandes
Service préparation des commandes
Service Emballage & expeditions
Service des Achats
PREPARATION - BON CMDE - CLIENT
Vérification - signature
F 1.1.1
ENREGISTREMENT BON CMDE CLIENT
Vérification - solvabilité client
F 1.2.1
Vérification - compt - commande
F 1.2.2
TRAITEMENT DES CAS LITIGIEUX
Correction - bon de commande
F 1.3.1
Répercussion - bon de commande
F 1.3.2
Selection - prod - à - rangger
F 2.1.1
PROD
CLT
CLT
SELECTION COMMANDES DIFFEREES
Selection - cmdes diffusion
F 1.7.1
CDE
STK
EXP
CDE
MISE A JOUR MOUVEMENT DU STOCK
Maj - mouvements - stock
F 4.4.1
STK
EXP
CDE
ORDONNANCEMENT PARCOURS - RAYON
Constitution - d - une - tournée
F 4.5.1
Ordonnancement
F 4.5.2
CDE
EXP
EXP
BL
Destockage
F 4.3.3
CONSTITUTION DU COLIS
Affichage - BL
BL
PREPARATION CMDE FOURNISSEUR
Choix des fournisseurs
F 8.1.1
Enregistrement cmde fournisseur
F 8.1.2

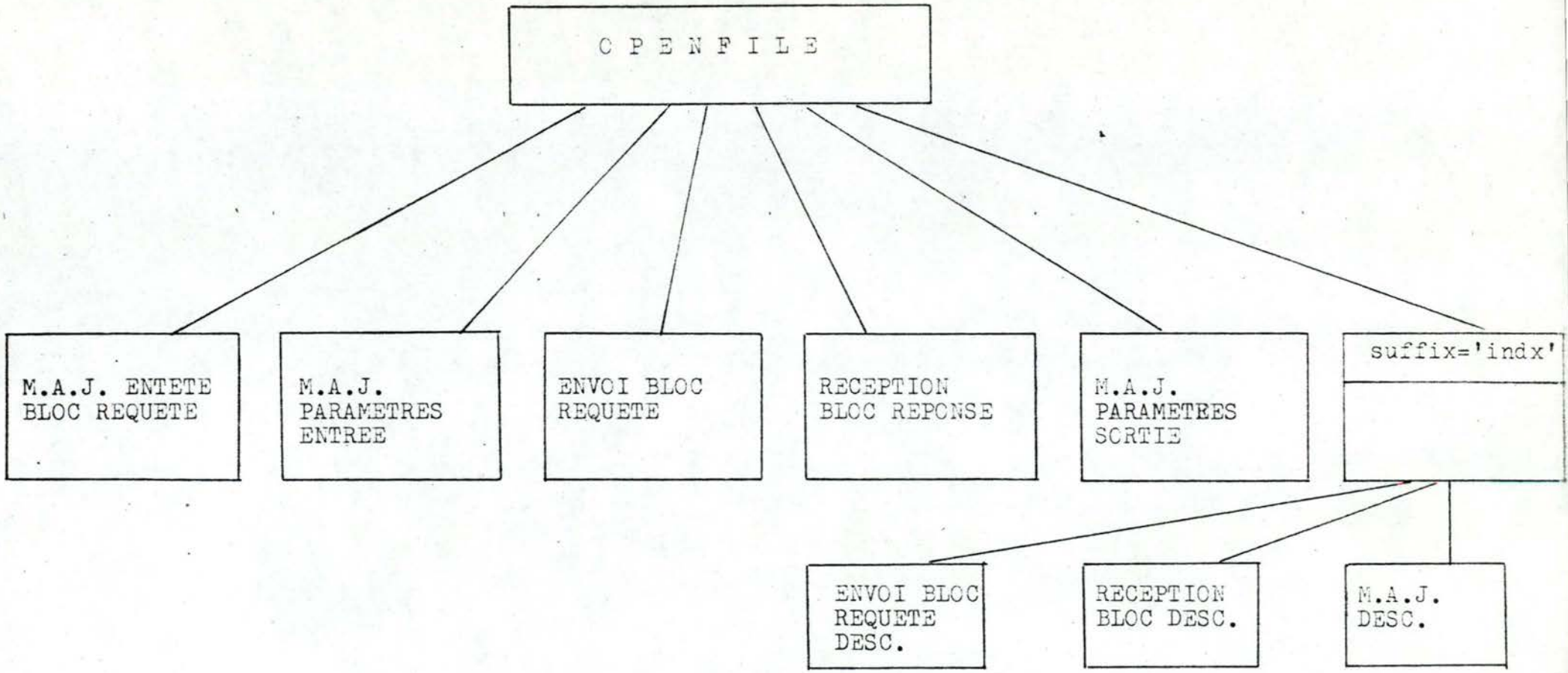

STK
EXP
CDE
ORDONNANCEMENT-PARCOURS-RAYON
Constitution-d-une-vague
Ordonnancement
DestockAge
CONSTITUTION-DU-COLIS
Affichage-BL
Recapitulation-emds
Facturation
Parcourt-magasin-correction
Affichage-BL correction
Reconstitution-emds correction
Correction du Système
Emballage-et-expedition
BL
RECEPTION-LIVRAISON-FOURN
Affichage-emds-Param
Contrôle-de-qualité
Mise-en-rayons
ENREGIST-LIVR-FOURNISSEUR
Enregistr-fiche-de-mvmt

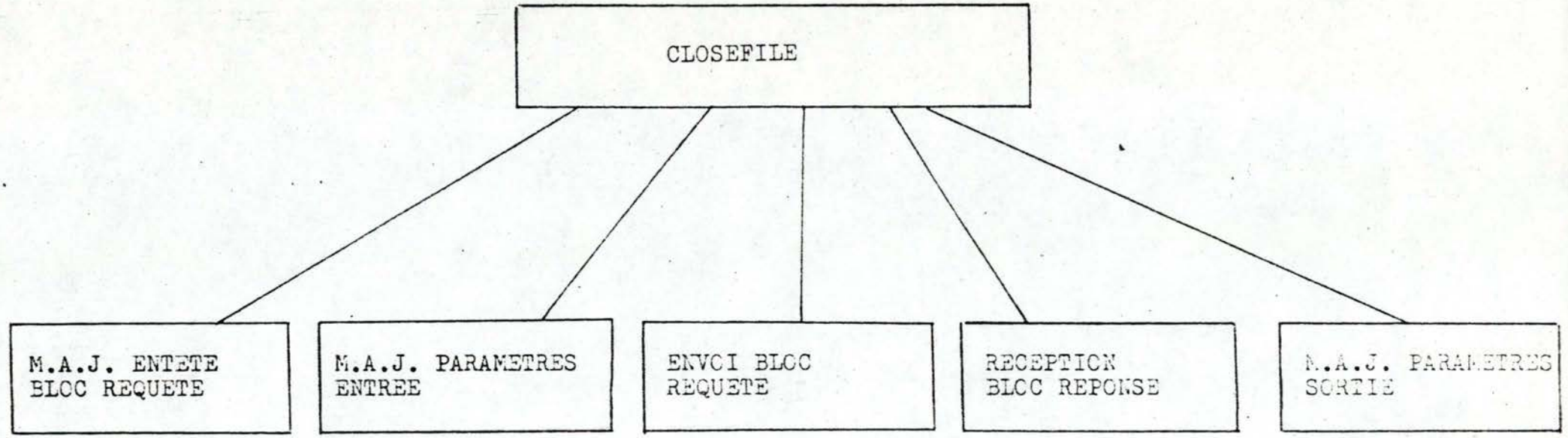
ANNEXE D

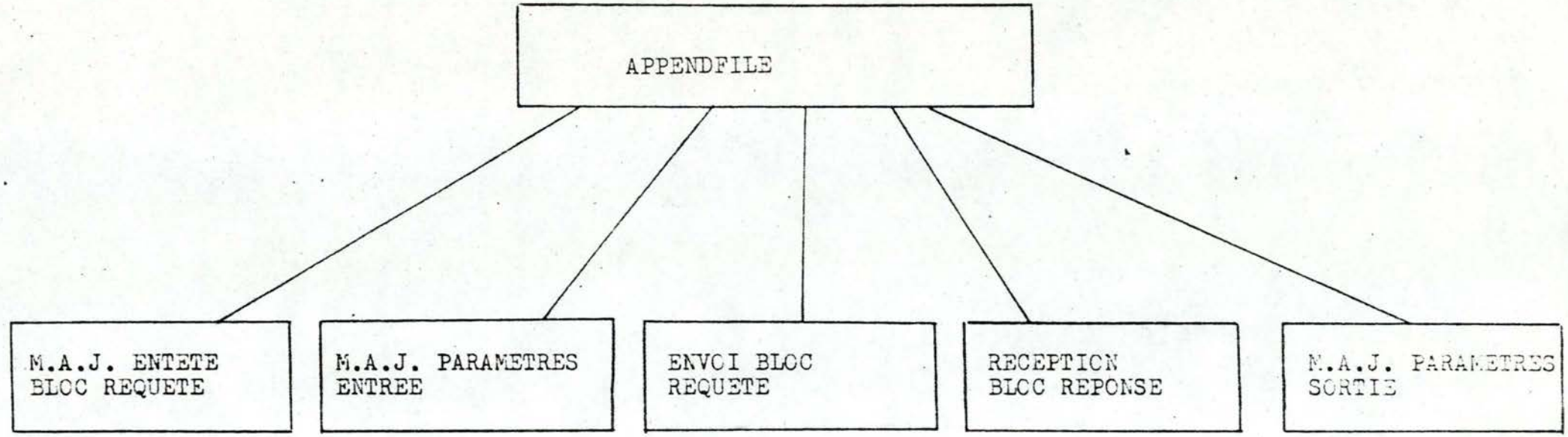
Programmes des fonctions système

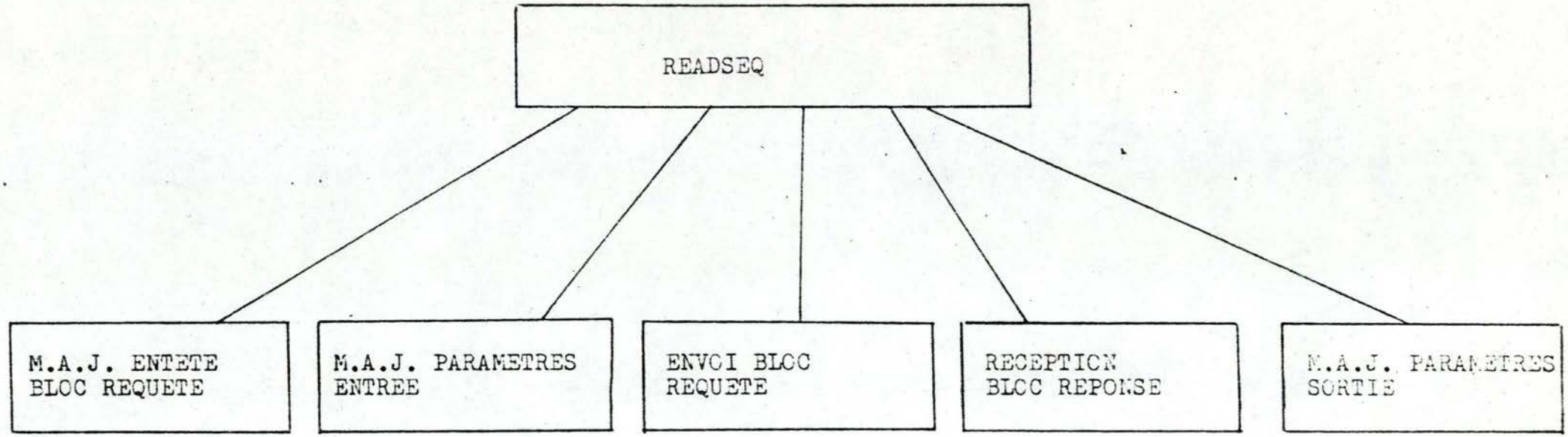
- programmes exécutés dans les stations terminales
- programme exécuté sur le gérant de la mémoire de masse
- roùtines de communication

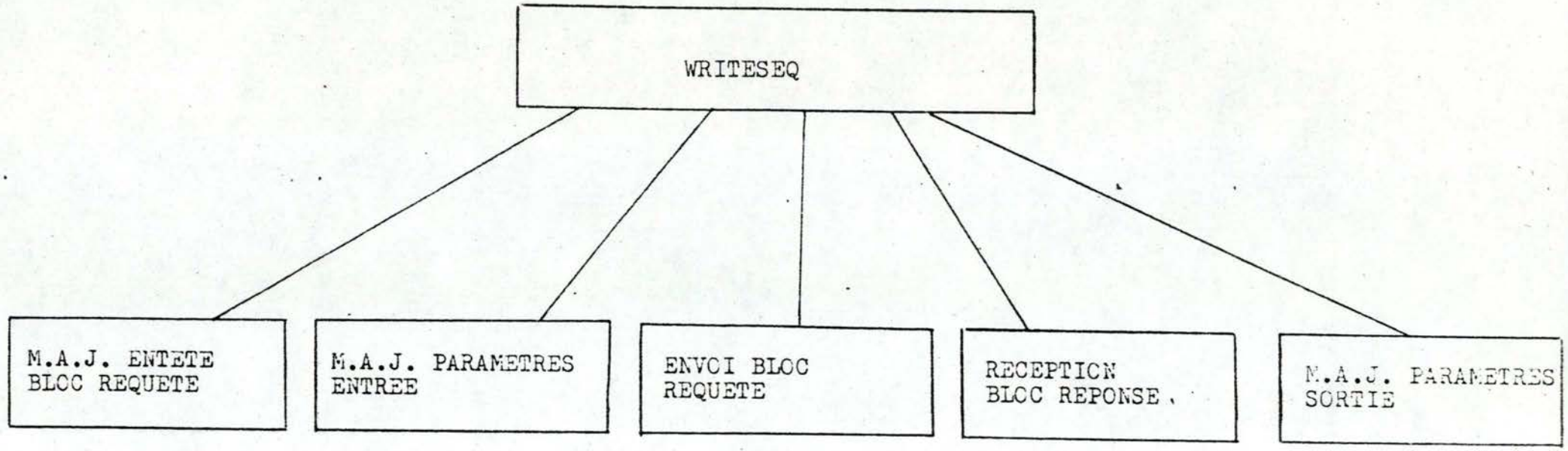
Programmes exécutés dans les stations terminales

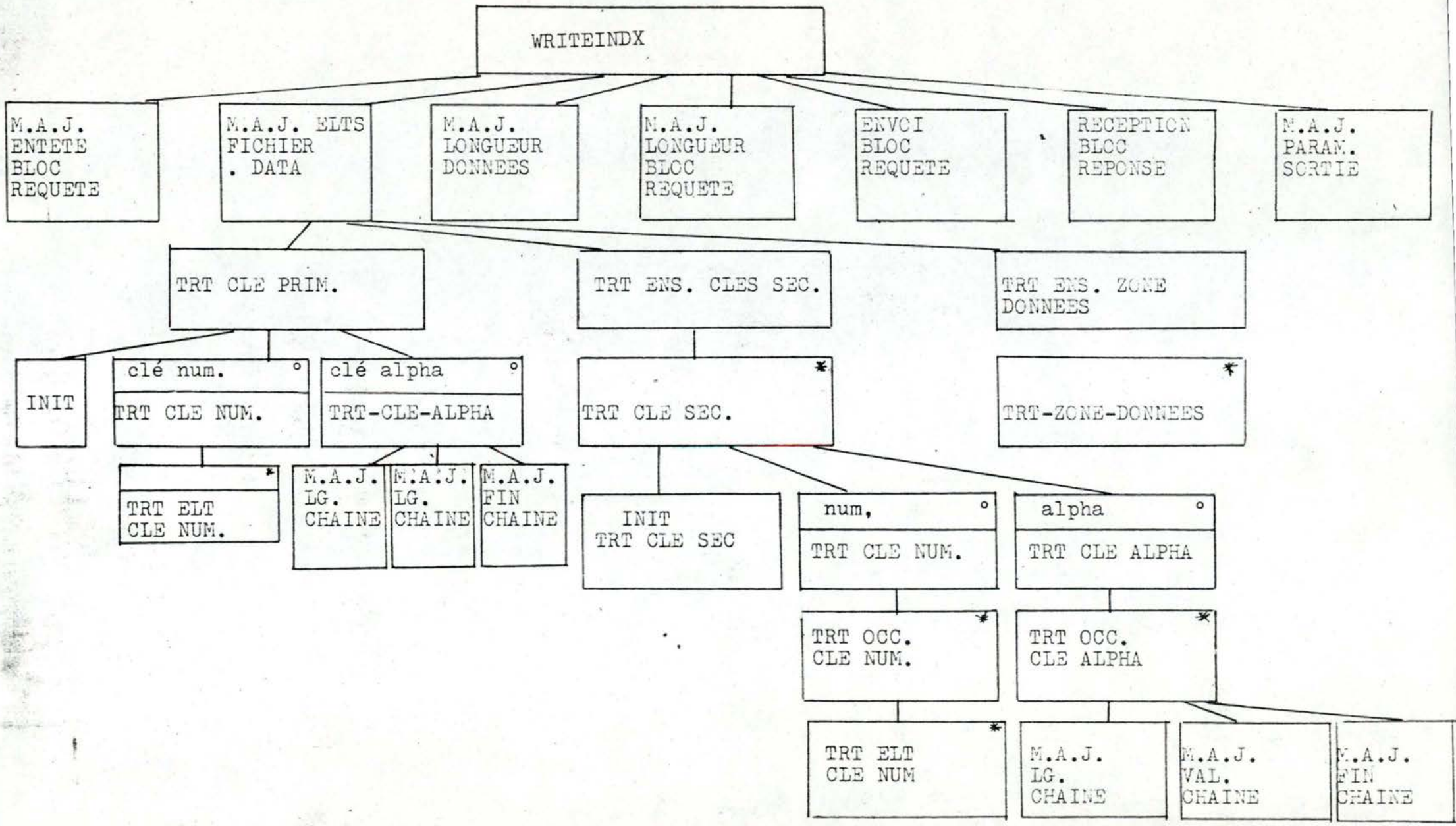

C P E N F I L E
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETEES SORTIE
suffix='indx'
ENVOI BLOC REQUETE DESC.
RECEPTION BLOC DESC.
M.A.J. DESC.

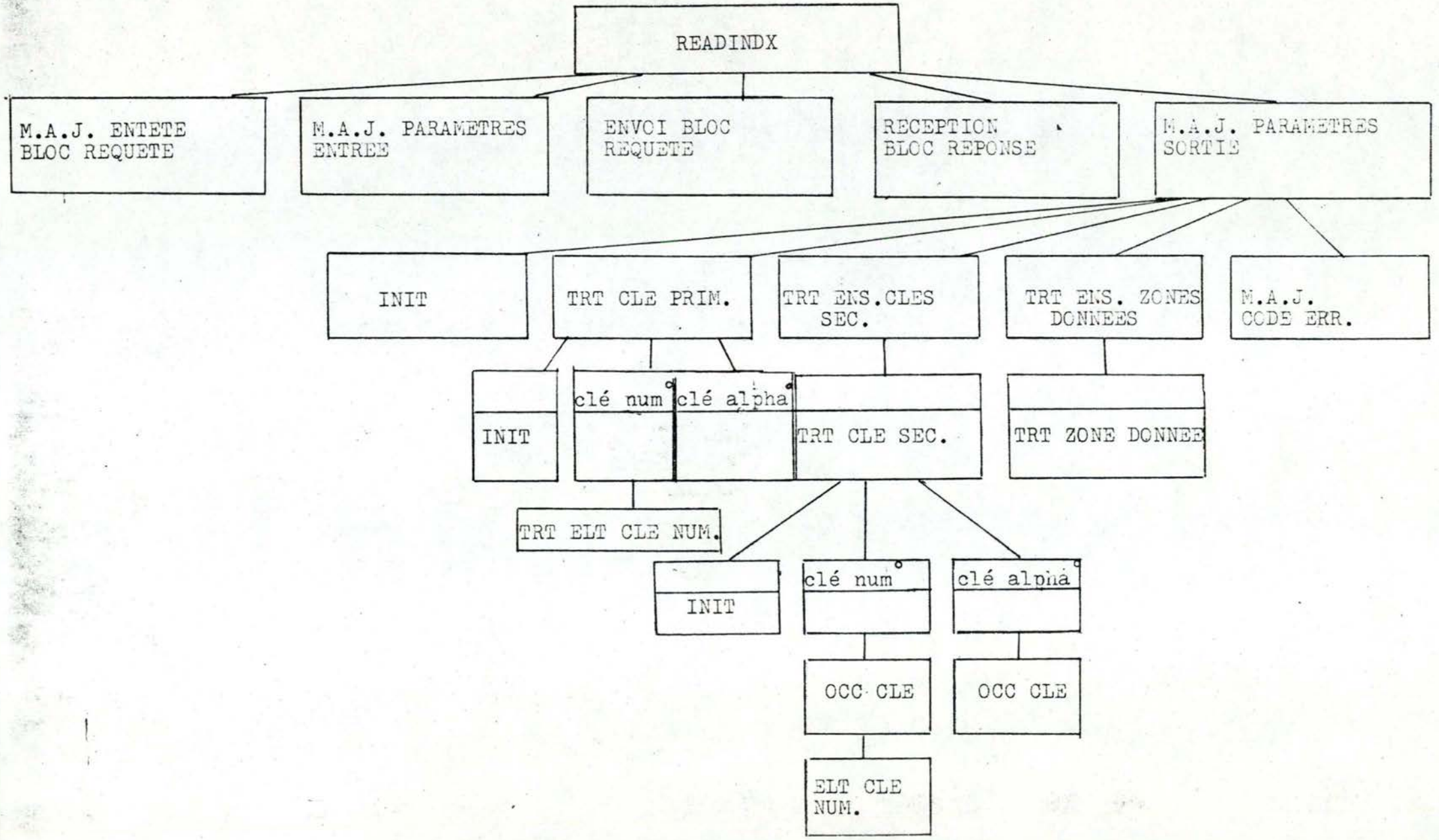

CLOSEFILE
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETRES SORTIE

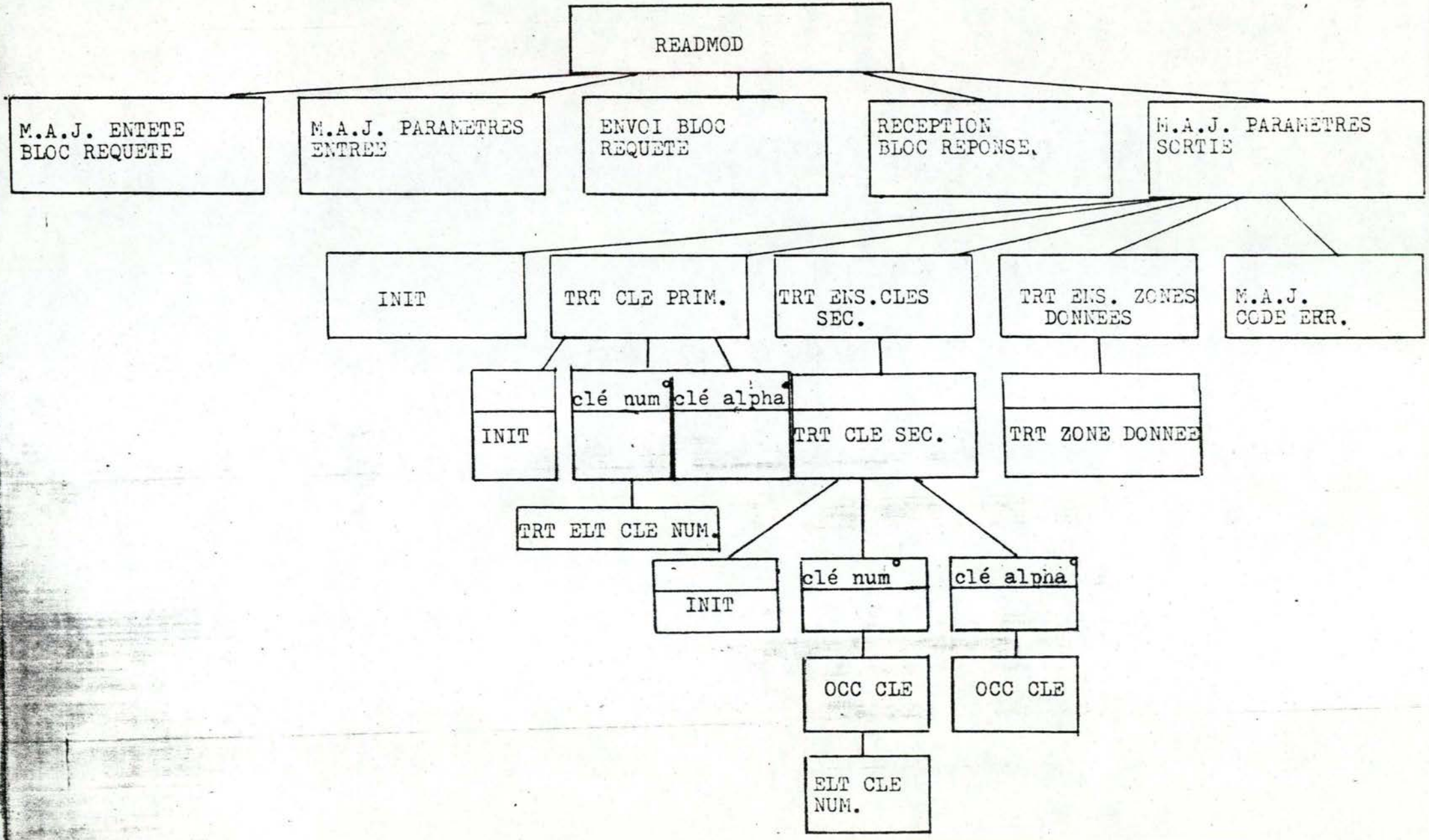

APPENDFILE
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETRES SORTIE

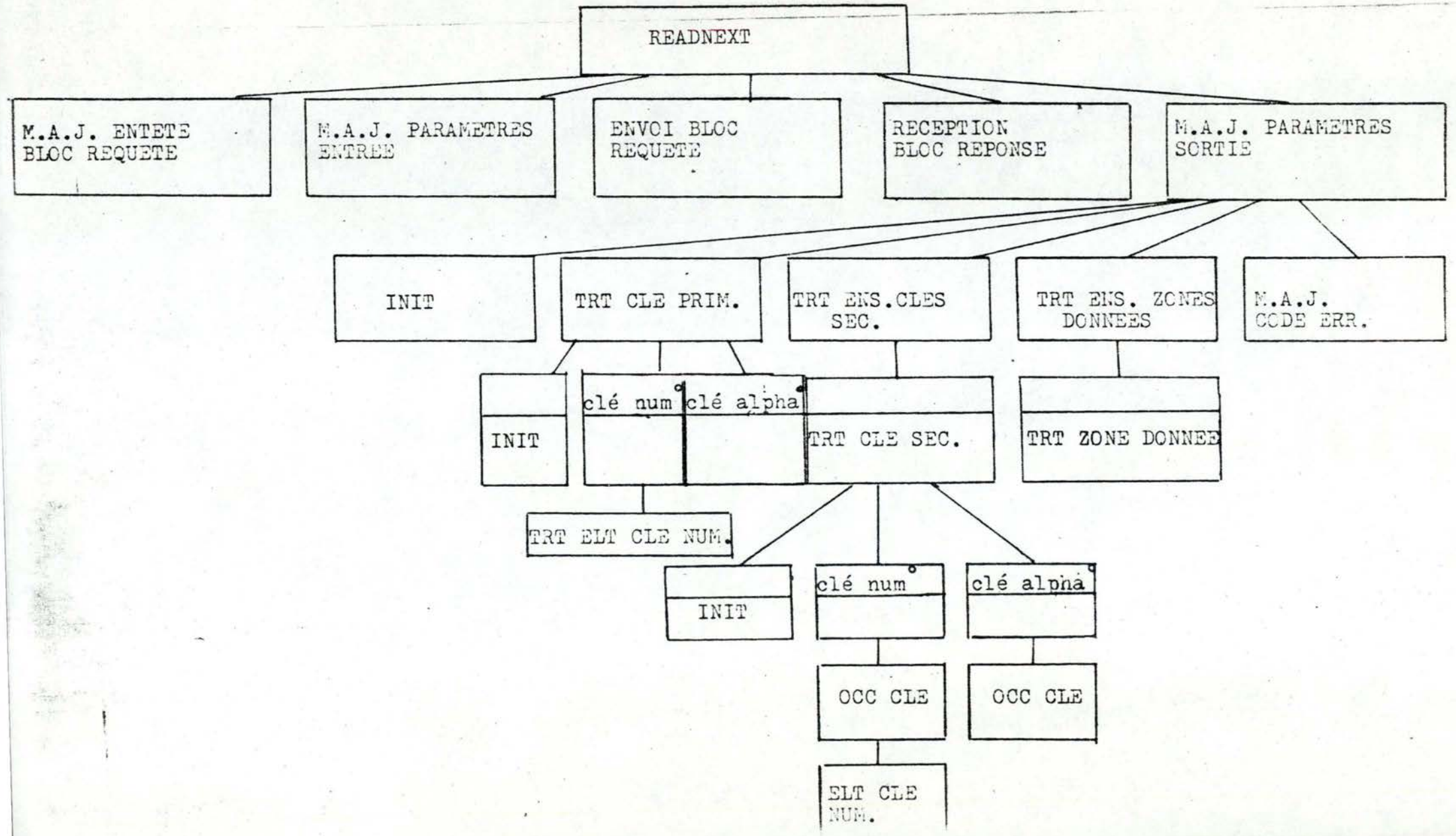

READSEQ
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETRES SORTIE

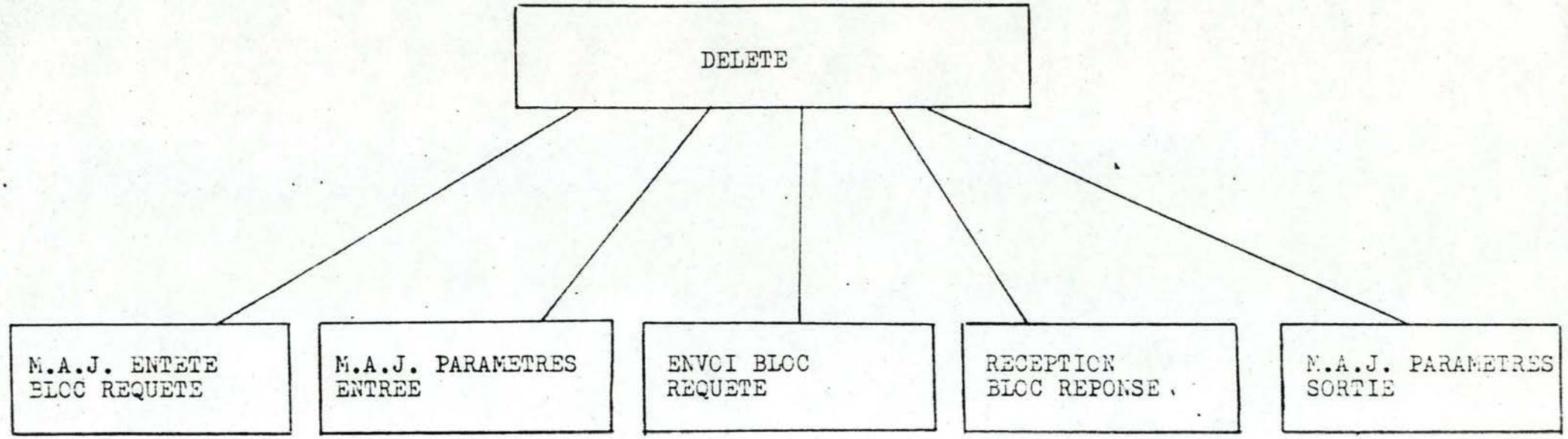

WRITESEQ
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE.
M.A.J. PARAMETRES SORTIE

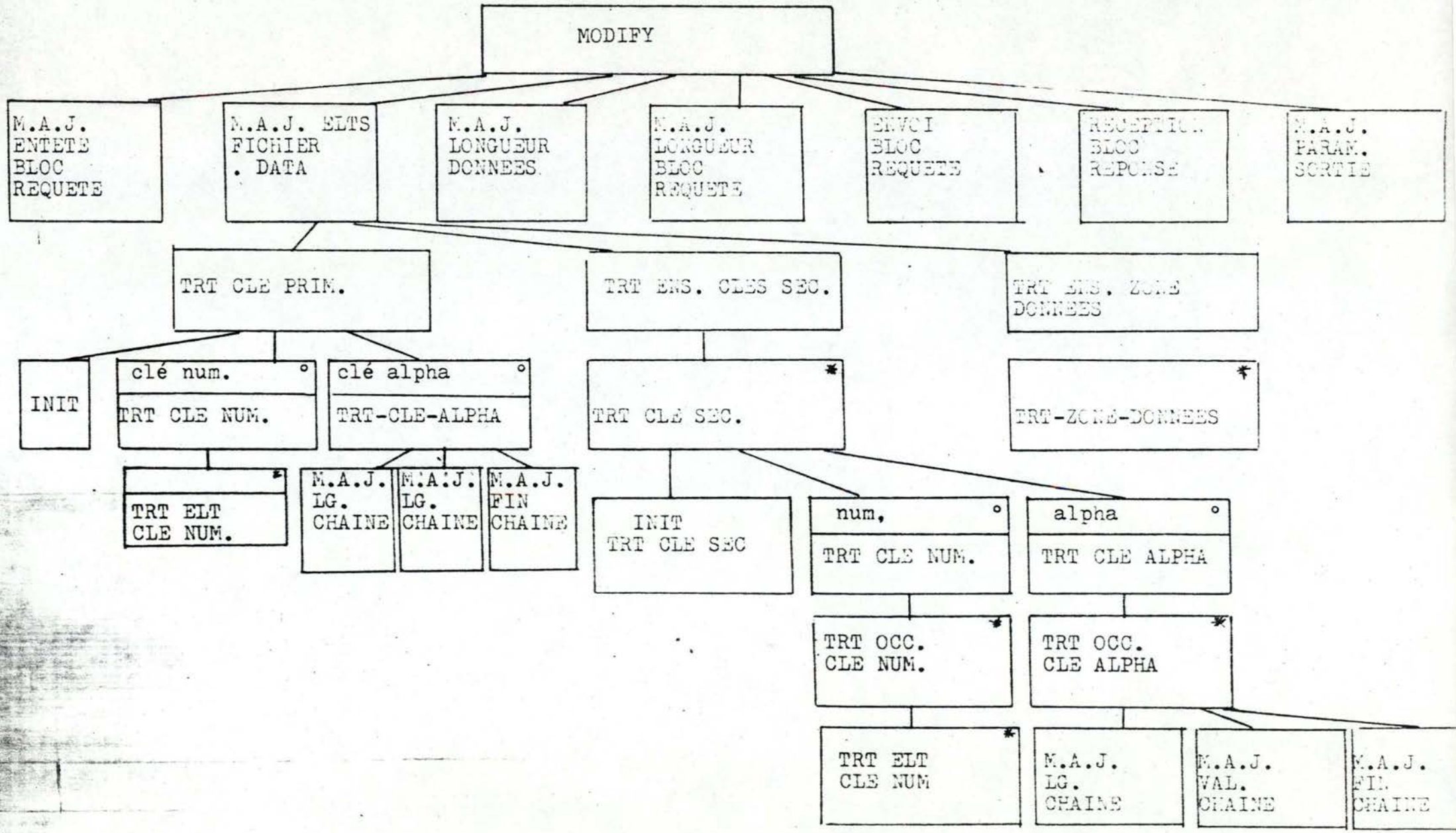

WRITEINDX
M.A.J. ENTETE BLOC REQUETE
M.A.J. ELTS FICHIER . DATA
M.A.J. LONGUEUR DONNEES
M.A.J. LONGUEUR BLOC REQUETE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAM. SORTIE
TRT CLE PRIM.
TRT ENS. CLES SEC.
TRT ENS. ZONE DONNEES
INIT
clé num.
TRT CLE NUM.
clé alpha
TRT-CLE-ALPHA
TRT CLE SEC.
TRT-ZONE-DONNEES
TRT ELT CLE NUM.
M.A.J. LG. CHAINE
M.A.J. LG. CHAINE
M.A.J. FIN CHAINE
INIT TRT CLE SEC
num,
TRT CLE NUM.
alpha
TRT CLE ALPHA
TRT OCC. CLE NUM.
TRT OCC. CLE ALPHA
TRT ELT CLE NUM
M.A.J. LG. CHAINE
M.A.J. VAL. CHAINE
M.A.J. FIN CHAINE


READINDX
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETRES SORTIE
INIT
TRT CLE PRIM.
TRT ENS.CLES SEC.
TRT ENS. ZONES DONNEES
M.A.J. CODE ERR.
INIT
clé num
clé alpha
TRT CLE SEC.
TRT ZONE DONNEE
TRT ELT CLE NUM.
INIT
clé num
clé alpha
OCC CLE
OCC CLE
ELT CLE NUM.


READMOD
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE.
M.A.J. PARAMETRES SORTIE
INIT
TRT CLE PRIM.
TRT ENS.CLES SEC.
TRT ENS. ZONES DONNEES
M.A.J. CODE ERR.
INIT
clé num
clé alpha
TRT CLE SEC.
TRT ZONE DONNEE
TRT ELT CLE NUM.
INIT
clé num
clé alpha
OCC CLE
OCC CLE
ELT CLE NUM.


READNEXT
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETRES SORTIE
INIT
TRT CLE PRIM.
TRT ENS.CLES SEC.
TRT ENS. ZONES DONNEES
M.A.J. CODE ERR.
INIT
clé num
clé alpha
TRT CLE SEC.
TRT ZONE DONNEE
TRT ELT CLE NUM.
INIT
clé num
clé alpha
OCC CLE
OCC CLE
ELT CLE NUM.


DELETE
M.A.J. ENTETE BLOC REQUETE
M.A.J. PARAMETRES ENTREE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAMETRES SORTIE


MODIFY
M.A.J. ENTETE BLOC REQUETE
M.A.J. ELTS FICHIER . DATA
M.A.J. LONGUEUR DONNEES
M.A.J. LONGUEUR BLOC REQUETE
ENVOI BLOC REQUETE
RECEPTION BLOC REPONSE
M.A.J. PARAM. SORTIE
TRT CLE PRIM.
TRT ENS. CLES SEC.
TRT ENS. ZONE DONNEES
INIT
clé num.
TRT CLE NUM.
clé alpha
TRT-CLE-ALPHA
TRT CLE SEC.
TRT-ZONE-DONNEES
TRT ELT CLE NUM.
M.A.J. LG. CHAINE
M.A.J. LG. CHAINE
M.A.J. FIN CHAINE
INIT TRT CLE SEC
num,
TRT CLE NUM.
alpha
TRT CLE ALPHA
TRT OCC. CLE NUM.
TRT OCC. CLE ALPHA
TRT ELT CLE NUM
M.A.J. LG. CHAINE
M.A.J. VAL. CHAINE
M.A.J. FIN CHAINE

```
(*$S+*)
UNIT UNITA;

INTERFACE

CONST

        (* liste des codes envoyes dans le bloc requete *)

        APPENDF =  6;
        OPENF   =  5;
        CLOSEF  =  7;
        WRSEQ   = 11;
        RDSEQ   =  9;
        WRINDX  = 40;
        RDINDX  = 41;
        RDNEXT  = 42;
        RDMOD   = 43;
        DLREC   = 44;
        MDREC   = 45;

        FL_SVR_AD  = 0;
        TERM_ST_AD = 0;

        (* valeur de positions dans les blocs requete et reponse *)

        I_CORR         = 4;
        I_LOCAL        = 5;
        I_BLOCK_LOW    = 6;
        I_BLOCK_HIGH   = 7;
        I_CODE         = 8;
        I_FL_NM_BEG    = 9;
        I_FL_NM_END    = 32;
        I_TERM         = 33;
        I_RET_CODE     = 8;
        I_ERR_CODE     = 9;
        I_CHAN         = 9;
        I_RET_CHAN     = 10;
        I_DATA_ID      = 10;
        I_DATA_HIGH    = 11;
        I_DATA_LOW     = 12;
        I_DATA_BEG     = 13;
        I_KEY_NO       = 10;
        I_KEY_VAL      = 11;
        I_KEY_OCC_BEG  = 13;

        KEY_MAX_LGTH   = 20;   { longueur maximale d'une cle}
        KEY_MAX_NB     = 10;   { nombre maximum de cles }
        MAX_REC_NB     = 7;    { nombre maximum d'enregistrement }
        DATA_MAX_NB    = 10;   { nombre maimum de zones donnees }

TYPE
        BYTE           = 0..255;

        WORD           = PACKED RECORD
                           CASE BOOLEAN OF
                           TRUE : (INT:INTEGER);
                           FALSE: (ARR:PACKED ARRAY [0..1] OF BYTE);
                         END;
```

```
    T_FILENAME    = PACKED RECORD
                      CASE BOOLEAN OF
                      TRUE :(STR:STRING[24]);
                      FALSE:(ARR:PACKED ARRAY [Ø..24] OF BYTE);
                    END;

    K_TYPE        = (PRM,SEC);

    CB_BLOCK      = PACKED ARRAY [Ø..263] OF BYTE;

    KEY_DESC      = RECORD
                      KEY_POS:INTEGER;
                      KEY_LGTH:INTEGER;
                      KEY_KIND:CHAR;
                    END;

    DATA_DESC     = RECORD
                      DATA_POS :INTEGER;
                      DATA_LGTH:INTEGER;
                    END;

    REC           = RECORD
                      CASE BOOLEAN OF
                      TRUE: (BUF :CB_BLOCK);
                      FALSE:(DESC:RECORD
                        ENT          : PACKED ARRAY [Ø..7] OF BYTE;
                        SEC_KEY_NB   : INTEGER;
                        KEY_DESC_TB  : ARRAY
                          [Ø..KEY_MAX_NB] OF KEY_DESC;
                        DATA_NB      : INTEGER;
                        DT_DESC_TB: ARRAY
                          [1..DATA_MAX_NB] OF DATA_DESC;
                        MAX_REC_LGTH: INTEGER;
                       END);
          END;

    KEY_KIND      = (INT,ALN,ARR);

    KEY           = RECORD
                      CASE KEY_KIND OF
                      INT:(INT:INTEGER);
                      ALN:(ALN:STRING[1ØØ]);
                      ARR:(ARR:PACKED ARRAY [Ø..99] OF BYTE);
                    END;

    DATA_ARRAY    = PACKED ARRAY [Ø..255] OF BYTE;

    KEY_ARRAY     = PACKED ARRAY [1..1Ø] OF BYTE;

VAR
    BLOCK         : CB_BLOCK;

    REC_ELT       : REC;

    REC_TB        : ARRAY [Ø..MAX_REC_NB] OF REC;

PROCEDURE appendfile ( VAR channel : INTEGER;
                       filename    : t_filename;
                       VAR err_code: INTEGER);
```

```
PROCEDURE openfile   ( VAR channel : INTEGER;
                       filename    : t_filename;
                       VAR err_code: INTEGER);

PROCEDURE closefile  ( channel : INTEGER;
                       VAR err_code : INTEGER);

PROCEDURE writeindx  ( channel : INTEGER;
                       sec_key_occ : key_array;
                       data : data_array;
                       VAR err_code : INTEGER);

PROCEDURE readindx   ( channel : INTEGER;
                       key_no  : INTEGER;
                       key_val : key;
                       VAR data : data_array;
                       VAR err_code : INTEGER);

PROCEDURE readmod    ( channel : INTEGER;
                       key_no  : INTEGER;
                       key_val : key;
                       VAR data_id : INTEGER;
                       VAR data : data_array;
                       VAR err_code : INTEGER);

PROCEDURE delete     ( channel : INTEGER;
                       key_val : key;
                       VAR err_code : INTEGER);

PROCEDURE modify     ( channel : INTEGER;
                       data_id : INTEGER;
                       sec_key_occ : key_array;
                       data : data_array;
                       VAR err_code : INTEGER);

PROCEDURE writeseq   (channel      : INTEGER;
                      data_lgth    : INTEGER;
                      VAR data     : data_array;
                      VAR err_code : INTEGER);

PROCEDURE readseq    (channel   : INTEGER;
                      data_lgth : INTEGER;
                      data_wr   : INTEGER;
                      err_code  : INTEGER);

IMPLEMENTATION

(*$I UNITB.TEXT *)

(*$I UNITC.TEXT *)

(*$I UNITD.TEXT *)

(*$I UNITE.TEXT *)
```

```
(*********************************************)
PROCEDURE writeindx (*channel       : INTEGER;
                     sec_key_occ    : key_array;
                     data           : data_array;
                     VAR err_code : INTEGER *);
(*********************************************)

VAR

  pos_key_occ : INTEGER;
  data_lgth   : INTEGER;
  crt_pos_blk : INTEGER;
  crt_pos_rec : INTEGER;


  (*******************)
  PROCEDURE prm_key_trt;
  (*******************)

  CONST

    blank = 32;

  VAR

    i         : INTEGER;   (indice)
    key_lgth : INTEGER;
    str_lgth : INTEGER;

  BEGIN

    (* initialisation*)

    crt_pos_rec := Ø;

    key_lgth := rec_tb [channel].desc.key_desc_tb [Ø].key_lgth;

    CASE rec_tb [channel].desc.key_desc_tb [Ø].key_kind OF

    'N','n' : BEGIN

                (* trt cle numerique *)

                FOR i := 1 TO rec_tb [channel].desc.
                              key_desc_tb [Ø].key_lgth DIV 2
                DO

                BEGIN

                  (* trt elt cle num *)

                  block[crt_pos_blk] := data [crt_pos_rec + 1];
                  block[crt_pos_blk + 1] := data [crt_pos_rec];

                  crt_pos_blk := crt_pos_blk + 2;
                  crt_pos_rec := crt_pos_rec + 2;
                  data_lgth   := data_lgth + 2;

                END;

              END;
```

```
    'A'..'a' : BEGIN
               (* trt cle alphanum *)

                 (* mise a jour longueur de la chaine *)

                 block [crt_pos_blk] := data [crt_pos_rec];
                 str_lgth            := data [crt_pos_rec];

                 crt_pos_blk := crt_pos_blk + 1;
                 crt_pos_rec := crt_pos_rec + 1;
                 data_lgth   := data_lgth + 1;

                 (* mise a jour valeur de la chaine *)


                 FOR i := 1 to str_lgth DO
                   BEGIN

                     block [crt_pos_blk] := data [crt_pos_rec];

                     crt_pos_blk := crt_pos_blk + 1;
                     crt_pos_rec := crt_pos_rec + 1;
                     data_lgth   := data_lgth + 1;

                   END;

                 (* mise a jour fin de la chaine *)

                 FOR i := 1 TO key_lgth - str_lgth DO
                   BEGIN

                     block [crt_pos_blk] := blank;

                     crt_pos_blk := crt_pos_blk + 1;
                     crt_pos_rec := crt_pos_rec + 1;
                     data_lgth   := data_lgth + 1;

                   END;



               END;

END;

END;

(*********************)
PROCEDURE sec_key_trt;
(*********************)

CONST

  blank = 32;

VAR

  i,j,k : INTEGER;
  key_lgth : INTEGER;
  str_lgth : INTEGER;

BEGIN
```

```
(* trt de l'ensemble des cles secondaires *)
FOR i := 1 TO rec_tb [channel].desc.sec_key_nb DO
BEGIN
  (* initialisation *)
  block [pos_key_occ] := sec_key_occ [i];
  crt_pos_rec := rec_tb [channel].desc.key_desc_tb [i].key_pos;
  key_lgth := rec_tb [channel].desc.key_desc_tb [i].key_lgth;
  CASE rec_tb [channel].desc.key_desc_tb [i].key_kind OF

    'N','n' : BEGIN
                (* trt cle numerique *)
                FOR j := 1 TO block [pos_key_occ] DO
                BEGIN
                (* trt d'une occ de cle numerique *)
                FOR j := 1 TO rec_tb [channel].desc.
                             key_desc_tb [i].key_lgth DIV 2
                DO
                BEGIN
                  (* trt d'un elt de la cle numerique *)
                  block[crt_pos_blk] := data [crt_pos_rec + 1];
                  block[crt_pos_blk + 1] := data [crt_pos_rec];
                  crt_pos_blk := crt_pos_blk + 2;
                  crt_pos_rec := crt_pos_rec + 2;
                  data_lgth   := data_lgth + 2;
                END;
                END;
              END;

    'A','a' : BEGIN
                (* trt de la cle alphanumerique *)
                FOR j := 1 TO block [pos_key_occ] DO
                BEGIN
                (* trt d'une occ de la cle alphanumerique *)
                  (* mise a jour longueur de la chaine *)
                  block [crt_pos_blk] := data [crt_pos_rec];
                  str_lgth            := data [crt_pos_rec];
                  crt_pos_blk := crt_pos_blk + 1;
```

```
                crt_pos_rec := crt_pos_rec + 1;
                data_lgth   := data_lgth + 1;

                (* mise a jour valeur de la chaine *)


                FOR k := 1 to str_lgth DO
                  BEGIN

                    block [crt_pos_blk] := data [crt_pos_rec];

                    crt_pos_blk := crt_pos_blk + 1;
                    crt_pos_rec := crt_pos_rec + 1;
                    data_lgth   := data_lgth + 1;

                  END;

                (* mise a jour fin de la chaine *)

                FOR k := 1 TO key_lgth - str_lgth DO
                  BEGIN

                    block [crt_pos_blk] := blank;

                    crt_pos_blk := crt_pos_blk + 1;
                    crt_pos_rec := crt_pos_rec + 1;
                    data_lgth   := data_lgth + 1;

                  END;

              END;

            END;

  END;

  pos_key_occ := pos_key_occ + 1; ( on passe a la cle suivante)

END;

END;

(*****************)
PROCEDURE data_trt;
(*****************)

VAR

  i,j : INTEGER;

BEGIN

  FOR i := 1 TO rec_tb [channel].desc.data_nb DO

  BEGIN

    (* trt de l'ensemble des zones de donnees *)

    crt_pos_rec := rec_tb [channel].desc.dt_desc_tb [i].data_pos;

    FOR j := 1 TO rec_tb [channel].desc.dt_desc_tb [i].data_lgth DO

    BEGIN
```

```
            (* trt d'une zone de donnee *)

            block [crt_pos_blk] := data [crt_pos_rec];

            crt_pos_blk := crt_pos_blk + 1;
            crt_pos_rec     := crt_pos_rec + 1;
            data_lgth   := data_lgth + 1;

          END;

        END;

      END;


BEGIN

  (* mise a jour entete bloc requete *)

  block [i_corr] := fl_svr_ad;
  block [i_local]:= term_st_ad;
  block [i_code] := wrindx;
  block [i_chan] := channel;

  (* mise a jour des elements a ecrire dans le fichier .DATA *)

  pos_key_occ := i_key_occ_beg;
  data_lgth   := rec_tb [channel].desc.sec_key_nb + 2;
  crt_pos_blk := i_key_occ_beg +
                 rec_tb [channel].desc.sec_key_nb;

  prm_key_trt;

  sec_key_trt;

  data_trt;

  (* mise a jour longeur des donnees *)

  block [i_data_high] := data_lgth DIV 256;
  block [i_data_low]  := data_lgth MOD 256;

  (* mise a jour longueur du bloc requete *)

  block [i_block_low]  := (i_key_occ_beg + data_lgth) MOD 256;
  block [i_block_high] := (i_key_occ_beg + data_lgth) DIV 256;

  (* envoi du bloc requete *)

  UNITWRITE (18,block [4],1,Ø,Ø);

  (* reception du bloc reponse *)

  UNITREAD (18,block [4],1,Ø,Ø);

  (* mise a jour parametre de sortie *)

  err_code := block [i_err_code];

END;
(*****************************************)
PROCEDURE readindx (*channel       : INTEGER;
                    key_no        : INTEGER;
                    key_val       : key;
```

```
                    VAR data     : data_array;
                    VAR err_code : INTEGER*);
{***********************************************}
CONST

  blank  = 32;

VAR

  i,j,K              : INTEGER;
  key_lgth,str_lgth  : INTEGER;
  crt_pos_blk        : INTEGER;
  crt_pos_rec        : INTEGER;
  pos_key_occ        : INTEGER;

  {********************}
  PROCEDURE prm_key_trt;
  {********************}

  VAR

    i : INTEGER;
    key_lgth : INTEGER;

  BEGIN

    crt_pos_rec := rec_tb [channel].desc.key_desc_tb[0].key_pos;

    key_lgth := rec_tb[channel].desc.key_desc_tb[0].key_lgth;

    CASE rec_tb[channel].desc.key_desc_tb [0].key_kind OF

    'N','n': BEGIN

               FOR i := 1 TO key_lgth DIV 2 DO
               BEGIN

                 data [crt_pos_rec] := block [crt_pos_blk + 1];
                 data [crt_pos_rec + 1] := block[crt_pos_blk];

                 crt_pos_blk := crt_pos_blk + 1;
                 crt_pos_rec := crt_pos_rec + 1;

               END;

             END;

    'A','a': BEGIN

               FOR i := 1 TO key_lgth + 1 DO
               BEGIN

                 data [crt_pos_rec] := block [crt_pos_blk];

                 crt_pos_blk := crt_pos_blk + 1;
                 crt_pos_rec := crt_pos_rec + 1;

               END;

             END;

  END;
```

```
END;

(*********************)
PROCEDURE sec_key_trt;
(*********************)

VAR

  i,j,k     : INTEGER;
  key_lgth : INTEGER;

BEGIN


  FOR i := 1 to rec_tb [channel].desc.sec_key_nb DO
  BEGIN

    crt_pos_rec := rec_tb [channel].desc.key_desc_tb [i].key_pos;
    key_lgth := rec_tb[channel].desc.key_desc_tb[i].key_lgth;

    CASE rec_tb[channel].desc.key_desc_tb[i].key_kind OF

  'N','n': BEGIN


             FOR j := 1 to block [pos_key_occ] DO
             BEGIN

               FOR k := 1 TO key_lgth DIV 2 DO
               BEGIN

               data [crt_pos_rec] := block [crt_pos_blk+1];
               data [crt_pos_rec+1] := block [crt_pos_blk];

               crt_pos_rec := crt_pos_rec + 2;
               crt_pos_blk := crt_pos_blk + 2;

               END


             END;


       END;

    'A','a': BEGIN

             FOR j := 1 TO block [pos_key_occ] DO
             BEGIN

               FOR k := 1 TO key_lgth + 1 DO
               BEGIN

                 data [crt_pos_rec] := block [crt_pos_blk];

                 crt_pos_rec := crt_pos_rec + 1;
                 crt_pos_blk := crt_pos_blk + 1;

               END

             END;

       END;
```

```
      END;
      pos_key_occ := pos_key_occ + 1;

    END;

  END;

  (*****************)
  PROCEDURE data_trt;
  (*****************)


  VAR

    i,j : INTEGER;

  BEGIN

    FOR i := 1 TO rec_tb [channel].desc.data_nb DO
    BEGIN

      crt_pos_rec := rec_tb [channel].desc.dt_desc_tb

                     [i].data_pos;

      FOR j := 1 TO rec_tb[channel].desc.dt_desc_tb
                    [i].data_lgth DO
      BEGIN


        data [crt_pos_rec] := block [crt_pos_blk];
        crt_pos_rec := crt_pos_rec + 1;
        crt_pos_blk := crt_pos_blk + 1;

      END;

    END;

  END;




BEGIN
  (* mise a jour entete bloc requete *)

  block [i_corr] := fl_svr_ad;
  block [i_local] := term_st_ad;
  block [i_code] := rdindx;
  block [i_block_low] := 3 + key_max_lgth;
  block [i_block_high] := 0;

  (* mise a jour parametres entree *)

  block [i_chan] := channel;
  block [i_key_no] := key_no;
```

```
key_lgth := rec_tb [channel].desc.key_desc_tb[key_no].key_lgth;
CASE rec_tb [channel].desc.key_desc_tb [key_no].key_kind OF

'N','n': BEGIN

           i := i_key_val;
           j := 1;

           FOR k := 1 TO key_lgth DIV 2 DO
           BEGIN

             block [i] := key_val.arr [j];
             block [i+1] := key_val.arr [j-1];
             i := i+2;
             j := j+2;

           END;

         END;

'A','a': BEGIN

           i := i_key_val;

           block [i] := key_val.arr [0];
           str_lgth  := key_val.arr [0];
           i := i+1;

           FOR j := 1 TO str_lgth DO
           BEGIN
             block [i] := key_val.arr [j];
             i := i+1;
           END;

           FOR j := i TO i + key_lgth - str_lgth - 1 DO
           BEGIN
             block [j] := blank;
           END;

         END;

    END;

(* envoi du bloc requete *)

UNITWRITE (18,block [4],1,0,0);

(* reception du bloc reponse *)

UNITREAD (18,BLOCK [4],1,0,0);

(* mise a jour des parametres de sortie *)

crt_pos_blk := i_key_occ_beg + rec_tb[channel].desc.
               sec_key_nb;

pos_key_occ := i_key_occ_beg;

prm_key_trt;

sec_key_trt;

data_trt;
```

```
err_code := block [i_err_code];
END;
```

```
{*****************************************}
PROCEDURE readmod (*channel      : INTEGER;
                   key_no        : INTEGER;
                   key_val       : key;
                   VAR data_id   : INTEGER;
                   VAR data      : data_array;
                   VAR err_code : INTEGER*);
{*****************************************}

CONST

  blank  = 32;

VAR

  i,j,K               : INTEGER;
  key_lgth.str_lgth : INTEGER;
  crt_pos_blk         : INTEGER;
  crt_pos_rec         : INTEGER;
  pos_key_occ         : INTEGER;

  {********************}
  PROCEDURE prm_key_trt;
  {********************}

  VAR

    i : INTEGER;
    key_lgth : INTEGER;

  BEGIN

    crt_pos_rec := rec_tb [channel].desc.key_desc_tb[Ø].key_pos;

    key_lgth := rec_tb[channel].desc.key_desc_tb[Ø].key_lgth;

    CASE rec_tb[channel].desc.key_desc_tb [Ø].key_kind OF

    'N','n': BEGIN

               FOR i := 1 TO key_lgth DIV 2 DO
               BEGIN

                 data [crt_pos_rec] := block [crt_pos_blk + 1];
                 data [crt_pos_rec + 1] := block[crt_pos_blk];

                 crt_pos_blk := crt_pos_blk + 1;
                 crt_pos_rec := crt_pos_rec + 1;

               END;

             END;

    'A','a': BEGIN

               FOR i := 1 TO key_lgth + 1 DO
               BEGIN

                 data [crt_pos_rec] := block [crt_pos_blk];

                 crt_pos_blk := crt_pos_blk + 1;
                 crt_pos_rec := crt_pos_rec + 1;

               END;
```

```
          END;
END;

END;

(*********************)
PROCEDURE sec_key_trt;
(*********************)

VAR

  i,j,k     : INTEGER;
  key_lgth  : INTEGER;

BEGIN


  FOR i := 1 to rec_tb [channel].desc.sec_key_nb DO
  BEGIN

  i crt_pos_rec := rec_tb [channel].desc.key_desc_tb [i].key_pos;
    key_lgth := rec_tb[channel].desc.key_desc_tb[i].key_lgth;

    CASE rec_tb[channel].desc.key_desc_tb[i].key_kind OF

    'N','n': BEGIN


               FOR j := 1 to block [pos_key_occ] DO
               BEGIN

                 FOR k := 1 TO key_lgth DIV 2 DO
                 BEGIN

                 data [crt_pos_rec] := block [crt_pos_blk+1];
                 data [crt_pos_rec+1] := block [crt_pos_blk];

                 crt_pos_rec := crt_pos_rec + 2;
                 crt_pos_blk := crt_pos_blk + 2;

                 END


               END;


         END;
     'A','a': BEGIN

                FOR j := 1 TO block [pos_key_occ] DO
                BEGIN

                  FOR k := 1 TO key_lgth + 1 DO
                  BEGIN

                    data [crt_pos_rec] := block [crt_pos_blk];

                    crt_pos_rec := crt_pos_rec + 1;
                    crt_pos_blk := crt_pos_blk + 1;

                  END
```

```
            END;
        END;

        END;
        pos_key_occ := pos_key_occ + 1;

      END;

    END;

    (*****************)
    PROCEDURE data_trt;
    (*****************)


    VAR

      i,j : INTEGER;

    BEGIN

      FOR i := 1 TO rec_tb [channel].desc.data_nb DO
      BEGIN

        crt_pos_rec := rec_tb [channel].desc.dt_desc_tb

                       [i].data_pos;

        FOR j := 1 TO rec_tb[channel].desc.dt_desc_tb
                      [i].data_lgth DO
        BEGIN


          data [crt_pos_rec] := block [crt_pos_blk];
          crt_pos_rec := crt_pos_rec + 1;
          crt_pos_blk := crt_pos_blk + 1;

        END;

      END;

    END;



BEGIN
  (* mise a jour entete bloc requete *)

  block [i_corr] := fl_svr_ad;
  block [i_local] := term_st_ad;
  block [i_code] := rdmod;
  block [i_block_low] := 3 + key_max_lgth;
  block [i_block_high] := 0;
```

```
(* mise a jour parametres entree *)

block [i_chan] := channel;
block [i_key_no] := key_no;

key_lgth := rec_tb [channel].desc.key_desc_tb[key_no].key_lgth;
CASE rec_tb [channel].desc.key_desc_tb [key_no].key_kind OF

'N','n': BEGIN

           i := i_key_val;
           j := 1;

           FOR k := 1 TO key_lgth DIV 2 DO
           BEGIN

             block [i] := key_val.arr [j];
             block [i+1] := key_val.arr [j-1];
             i := i+2;
             j := j+2;

           END;

         END;

'A','a': BEGIN

           i := i_key_val;

           block [i] := key_val.arr [Ø];
           str_lgth  := key_val.arr [Ø];
           i := i+1;

           FOR j := 1 TO str_lgth DO
           BEGIN
             block [i] := key_val.arr [j];
             i := i+1;
           END;

           FOR j := i TO i + key_lgth - str_lgth - 1 DO
           BEGIN
             block [j] := blank;
           END;

         END;

     END;

(* envoi du bloc requete *)

UNITWRITE (18,block [4],1,Ø,Ø);

(* reception du bloc reponse *)

UNITREAD (18,BLOCK [4],1,Ø,Ø);

(* mise a jour des parametres de sortie *)

crt_pos_blk := i_key_occ_beg + rec_tb[channel].desc.
               sec_key_nb;

pos_key_occ := i_key_occ_beg;

prm_key_trt;
```

```
  sec_key_trt;

  data_id := block [i_data_id];

  data_trt;

  err_code := block [i_err_code];

END;


(*********************************************)
PROCEDURE delete  (*channel       : INTEGER;
                   key_val       : key;
                   VAR err_code : INTEGER*);
(*********************************************)

CONST

  blank  = 32;

VAR

  i,j,K              : INTEGER;
  key_lgth,str_lgth : INTEGER;



BEGIN

  (* mise a jour entete bloc requete *)

  block [i_corr] := fl_svr_ad;
  block [i_local] := term_st_ad;
  block [i_code] := direc;
  block [i_block_low] := 3 + key_max_lgth;
  block [i_block_high] := 0;

  (* mise a jour parametres entree *)

  block [i_chan] := channel;

  key_lgth := rec_tb [channel].desc.key_desc_tb[0].key_lgth;
  CASE rec_tb [channel].desc.key_desc_tb [0].key_kind OF

  'N','n': BEGIN

             i := i_key_val;
             j := 1;

             FOR k := 1 TO key_lgth DIV 2 DO
             BEGIN

               block [i] := key_val.arr [j];
               block [i+1] := key_val.arr [j-1];
               i := i+2;
               j := j+2;

             END;
```

```
            END;

    'A','a': BEGIN

               i := i_key_val;

               block [i] := key_val.arr [Ø];
               str_lgth  := key_val.arr [Ø];
               i := i+1;

               FOR j := 1 TO str_lgth DO
               BEGIN
                 block [i] := key_val.arr [j];
                 i := i+1;
               END;

               FOR j := i TO i + key_lgth - str_lgth - 1 DO
               BEGIN
                 block [j] := blank;
               END;

             END;

        END;

  (* envoi du bloc requete *)

  UNITWRITE (18,block [4],1,Ø,Ø);

  (* reception du bloc reponse *)

  UNITREAD (18,BLOCK [4],1,Ø,Ø);

  (* mise a jour des parametres de sortie *)

  err_code := block [i_err_code];

END;



(*********************************************)
PROCEDURE modify   (*channel       : INTEGER;
                    data_id       : INTEGER;
                    sec_key_occ   : key_array;
                    data          : data_array;
                    VAR err_code  : INTEGER *);
(*********************************************)

VAR

  pos_key_occ : INTEGER;
  data_lgth   : INTEGER;
  crt_pos_blk : INTEGER;
  crt_pos_rec : INTEGER;

  (********************)
  PROCEDURE prm_key_trt;
  (********************)
```

```
CONST
  blank = 32;
VAR
  i         : INTEGER;   (indice)
  key_lgth  : INTEGER;
  str_lgth  : INTEGER;
BEGIN
  (* initialisation*)
  crt_pos_rec := Ø;
  key_lgth := rec_tb [channel].desc.key_desc_tb [Ø].key_lgth;
  CASE rec_tb [channel].desc.key_desc_tb [Ø].key_kind OF
  'N','n' : BEGIN
  i           (* trt cle numerique *)
              FOR i := 1 TO rec_tb [channel].desc.
                            key_desc_tb [Ø].key_lgth DIV 2
              DO
              BEGIN
                (* trt elt cle num *)
                block[crt_pos_blk] := data [crt_pos_rec + 1];
                block[crt_pos_blk + 1] := data [crt_pos_rec];
                crt_pos_blk := crt_pos_blk + 2;
                crt_pos_rec := crt_pos_rec + 2;
                data_lgth   := data_lgth + 2;
              END;
            END;
  'A','a' : BEGIN
            (* trt cle alphanum *)
              (* mise a jour longueur de la chaine *)
              block [crt_pos_blk] := data [crt_pos_rec];
              str_lgth            := data [crt_pos_rec];
              crt_pos_blk := crt_pos_blk + 1;
              crt_pos_rec := crt_pos_rec + 1;
              data_lgth   := data_lgth + 1;
  |           (* mise a jour valeur de la chaine *)
  |
  |           FOR i := 1 to str_lgth DO
  |             BEGIN
  |
  |               block [crt_pos_blk] := data [crt_pos_rec];
```

```
                crt_pos_blk := crt_pos_blk + 1;
                crt_pos_rec := crt_pos_rec + 1;
                data_lgth   := data_lgth + 1;

              END;

            (* mise a jour fin de la chaine *)

            FOR i := 1 TO key_lgth - str_lgth DO
              BEGIN

                block [crt_pos_blk] := blank;

                crt_pos_blk := crt_pos_blk + 1;
                crt_pos_rec := crt_pos_rec + 1;
                data_lgth   := data_lgth + 1;

              END;



        END;
END;

END;

(*********************)
PROCEDURE sec_key_trt;
(*********************)

CONST

  blank = 32;

VAR

  i,j,k : INTEGER;
  key_lgth : INTEGER;
  str_lgth : INTEGER;

BEGIN

  (* trt de l'ensemble des cles secondaires *)

  FOR i := 1 TO rec_tb [channel].desc.sec_key_nb DO

  BEGIN

    (* initialisation *)

    block [pos_key_occ] := sec_key_occ [i];

    crt_pos_rec := rec_tb [channel].desc.key_desc_tb [i].key_pos;

    key_lgth := rec_tb [channel].desc.key_desc_tb [i].key_lgth;

    CASE rec_tb [channel].desc.key_desc_tb [i].key_kind OF

      'N','n' : BEGIN

                  (* trt cle numerique *)
```

```
            FOR j := 1 TO block [pos_key_occ] DO
            BEGIN

            (* trt d'une occ de cle numerique *)

            FOR j := 1 TO rec_tb [channel].desc.
                          key_desc_tb [i].key_lgth DIV 2
            DO

            BEGIN

              (* trt d'un elt de la cle numerique *)

              block[crt_pos_blk] := data [crt_pos_rec + 1];
              block[crt_pos_blk + 1] := data [crt_pos_rec];

              crt_pos_blk := crt_pos_blk + 2;
              crt_pos_rec := crt_pos_rec + 2;
              data_lgth   := data_lgth + 2;

            END;

            END;

          END;

'A','a' : BEGIN

            (* trt de la cle alphanumerique *)

            FOR j := 1 TO block [pos_key_occ] DO
            BEGIN

            (* trt d'une occ de la cle alphanumerique *)

              (* mise a jour longueur de la chaine *)

              block [crt_pos_blk] := data [crt_pos_rec];
              str_lgth            := data [crt_pos_rec];

              crt_pos_blk := crt_pos_blk + 1;
              crt_pos_rec := crt_pos_rec + 1;
              data_lgth   := data_lgth + 1;

              (* mise a jour valeur de la chaine *)

              FOR k := 1 to str_lgth DO
                BEGIN

                  block [crt_pos_blk] := data [crt_pos_rec];

                  crt_pos_blk := crt_pos_blk + 1;
                  crt_pos_rec := crt_pos_rec + 1;
                  data_lgth   := data_lgth + 1;

                END;

              (* mise a jour fin de la chaine *)

              FOR k := 1 TO key_lgth - str_lgth DO
                BEGIN
```

```
                    block [crt_pos_blk] := blank;

                    crt_pos_blk := crt_pos_blk + 1;
                    crt_pos_rec := crt_pos_rec + 1;
                    data_lgth   := data_lgth + 1;

                  END;

              END;

            END;

    END;

    pos_key_occ := pos_key_occ + 1; { on passe a la cle suivante}

  END;

  END;

  (*****************)
  PROCEDURE data_trt;
  (*****************)

  VAR

    i,j : INTEGER;

  BEGIN

    FOR i := 1 TO rec_tb [channel].desc.data_nb DO

    BEGIN

      (* trt de l'ensemble des zones de donnees *)

      crt_pos_rec := rec_tb [channel].desc.dt_desc_tb [i].data_pos;

      FOR j := 1 TO rec_tb [channel].desc.dt_desc_tb [i].data_lgth DO

      BEGIN

        (* trt d'une zone de donnee *)

        block [crt_pos_blk] := data [crt_pos_rec];

        crt_pos_blk := crt_pos_blk + 1;
        crt_pos_rec     := crt_pos_rec + 1;
        data_lgth   := data_lgth + 1;

      END;

    END;

  END;

BEGIN

  (* mise a jour entete bloc requete *)

  block [i_corr] := fl_svr_ad;
  block [i_local]:= term_st_ad;
```

```
    block [i_code] := dlrec;
    block [i_chan] := channel;

    (* mise a jour des elements a ecrire dans le fichier .DATA *)

    pos_key_occ := i_key_occ_beg;
    data_lgth   := rec_tb [channel].desc.sec_key_nb + 2;
    crt_pos_blk := i_key_occ_beg +
                   rec_tb [channel].desc.sec_key_nb;

    prm_key_trt;

    sec_key_trt;

    data_trt;

    (* mise a jour longeur des donnees *)

    block [i_data_high] := data_lgth DIV 256;
    block [i_data_low]  := data_lgth MOD 256;

    (* mise a jour longueur du bloc requete *)

    block [i_block_low]  := (i_key_occ_beg + data_lgth) MOD 256;
    block [i_block_high] := (i_key_occ_beg + data_lgth) DIV 256;

    (* envoi du bloc requete *)

    UNITWRITE (18,block [4],1,Ø,Ø);

    (* reception du bloc reponse *)

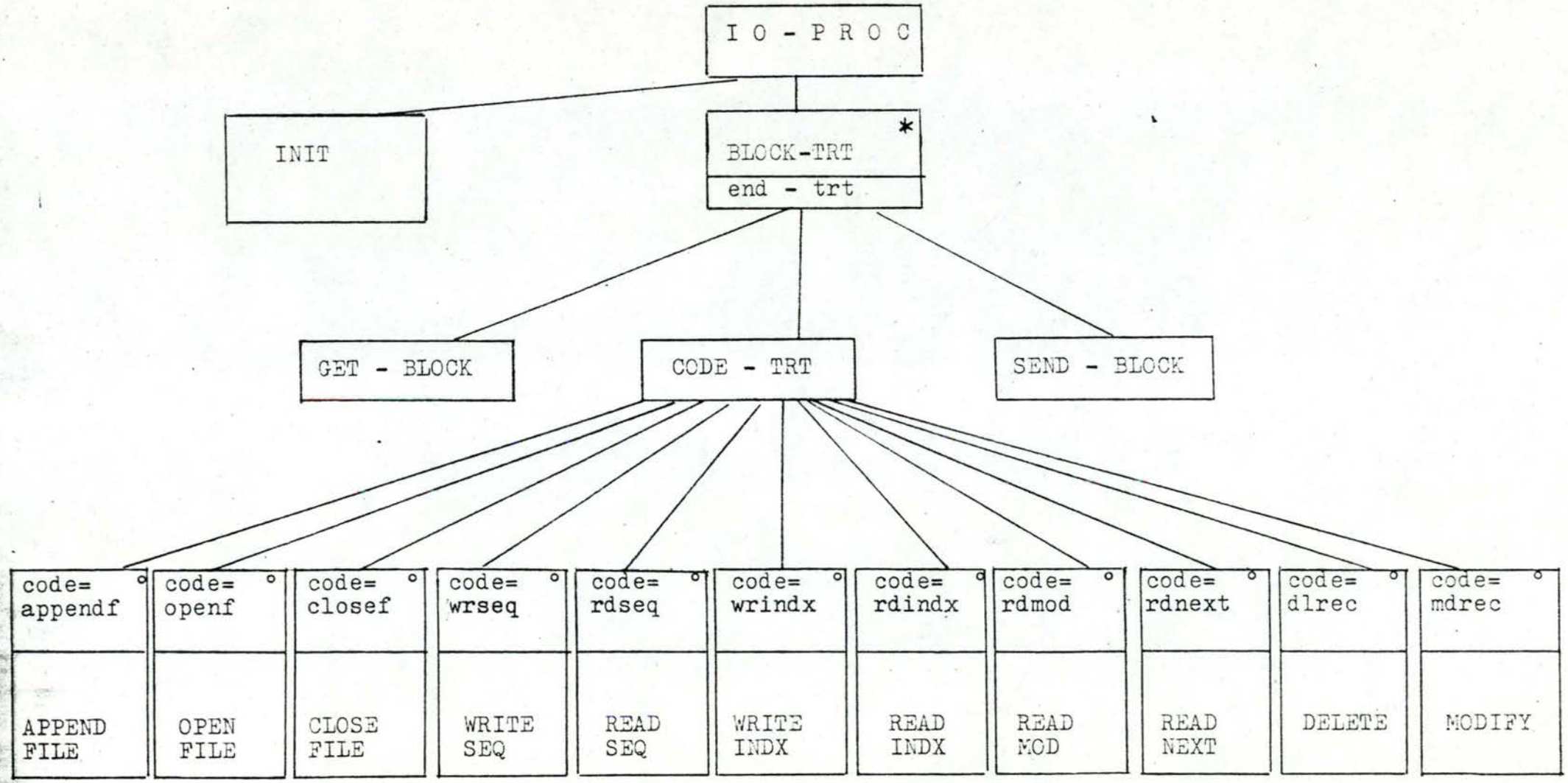
    UNITREAD (18,block [4],1,Ø,Ø);

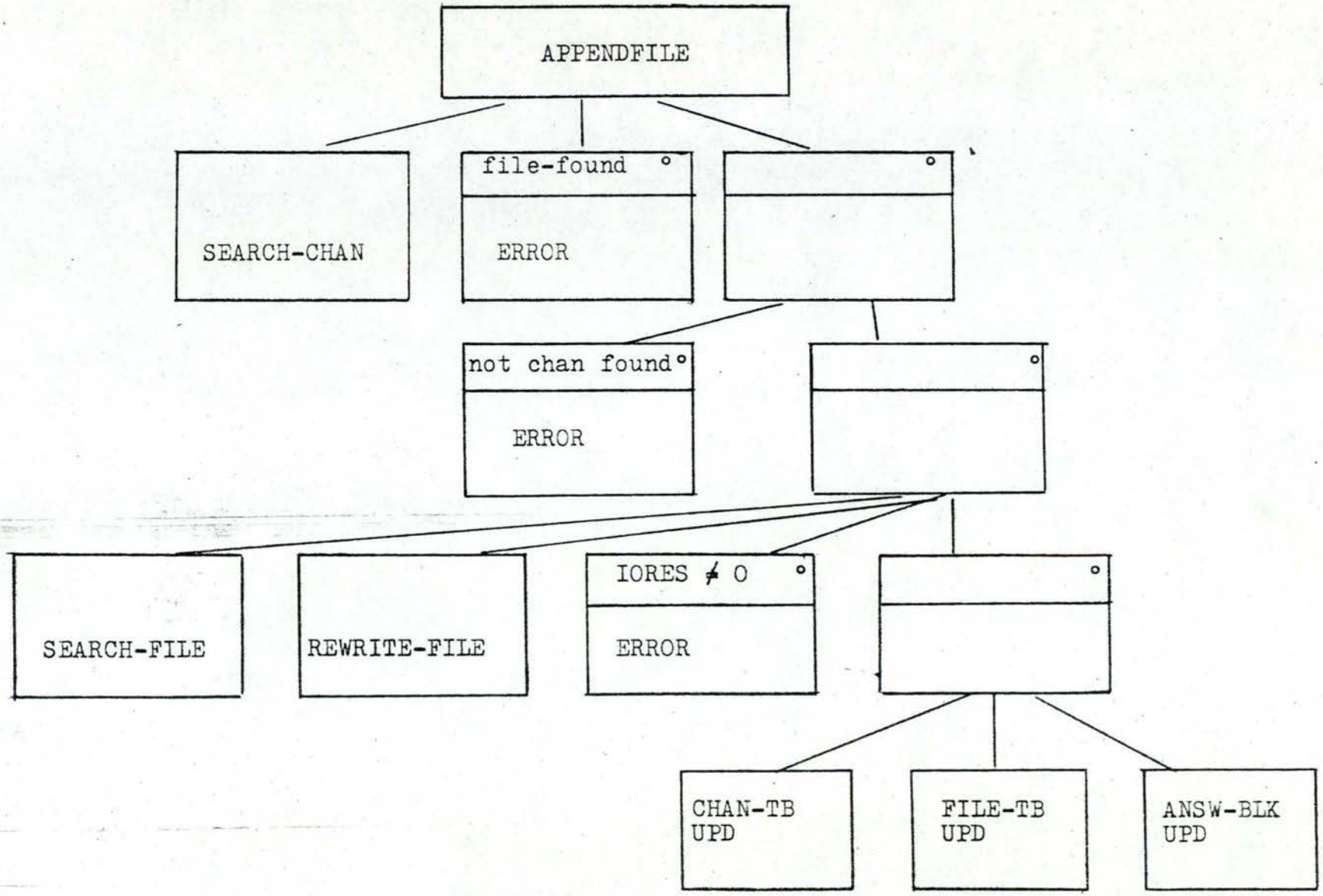
    (* mise a jour parametre de sortie *)

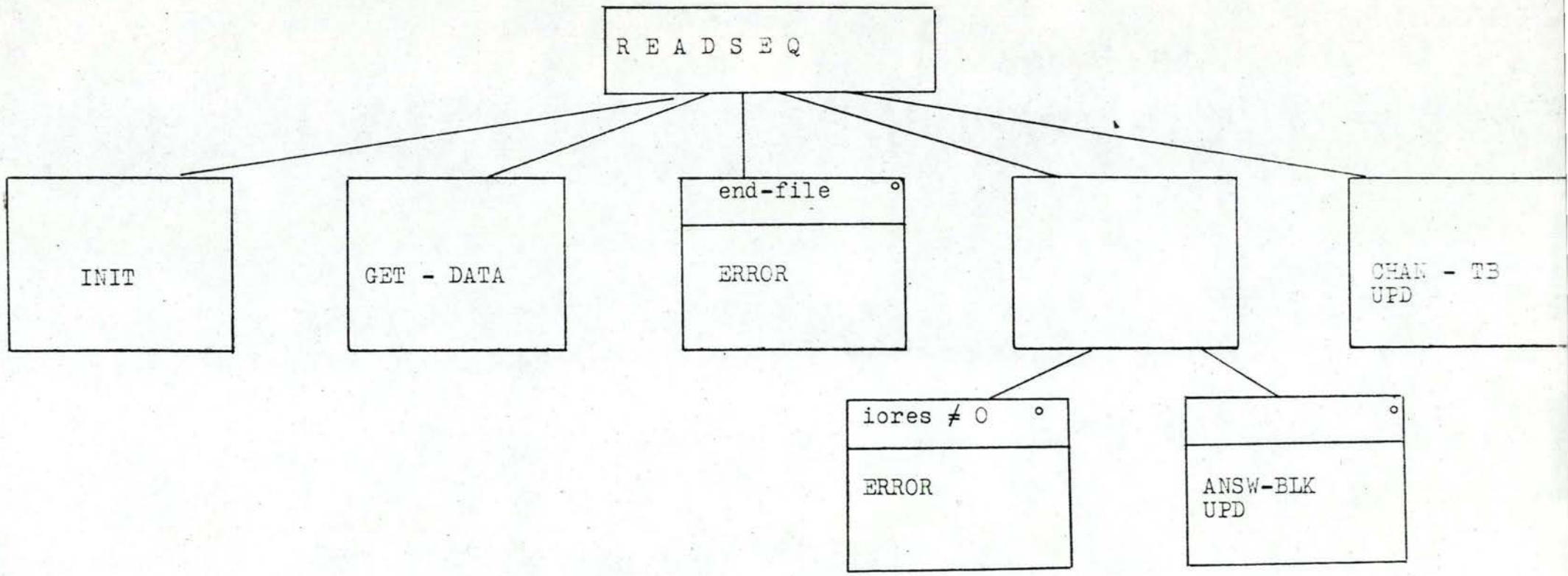
    data_id  := block [i_data_id];
    err_code := block [i_err_code];
END;

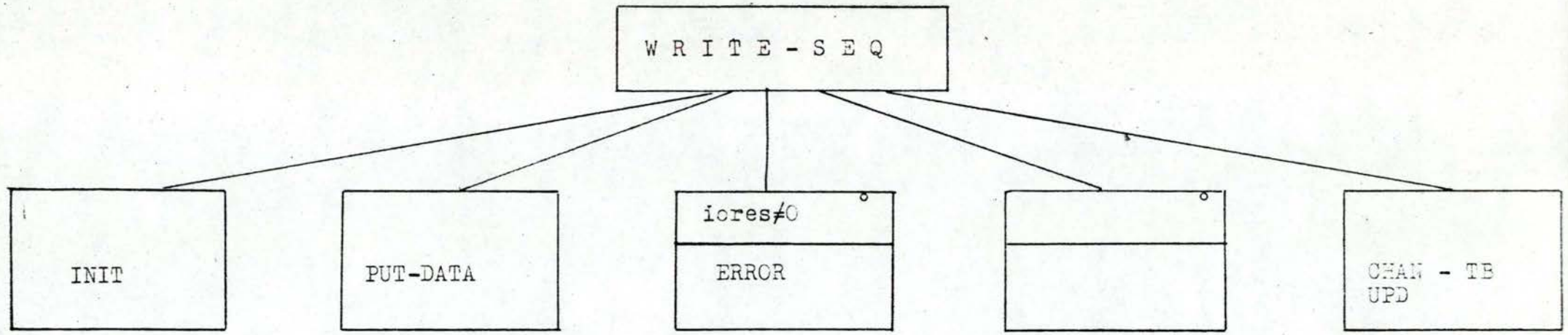
END. (* UNITA *)
```

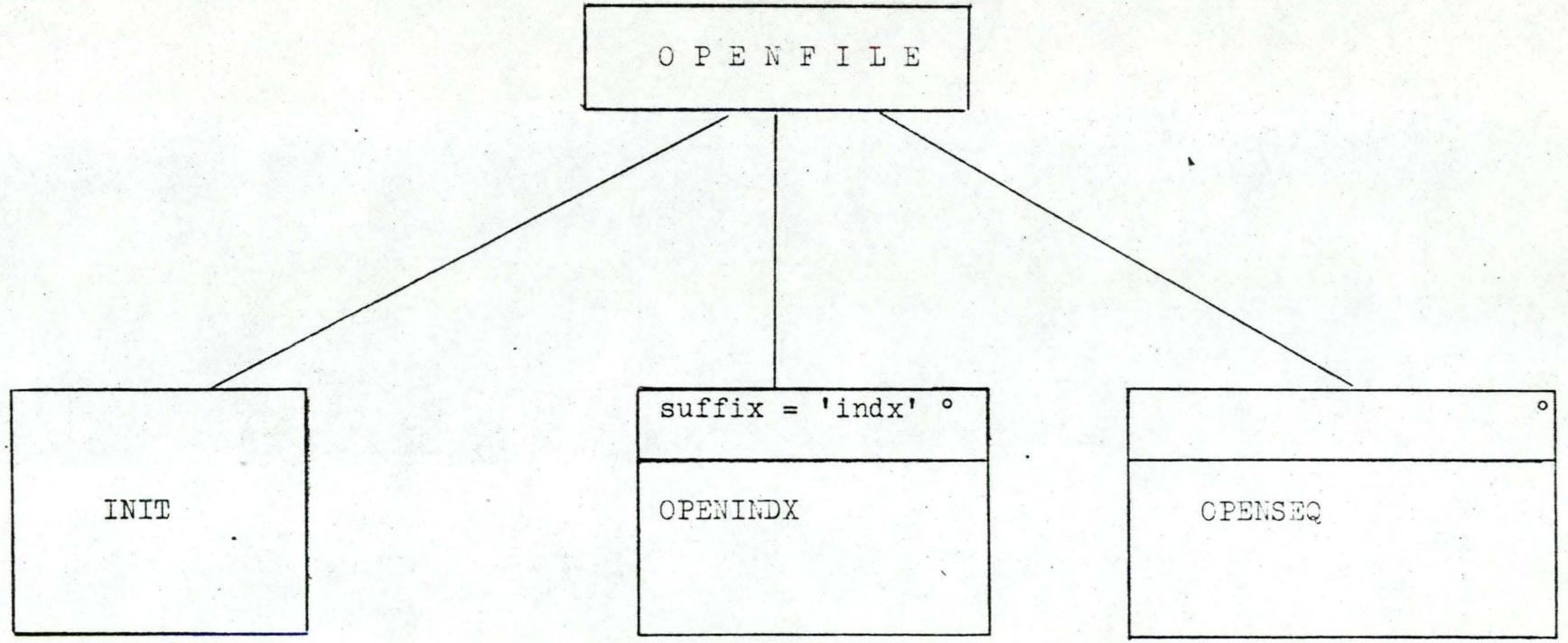
Programme exécuté sur le gérant de la mémoire de masse

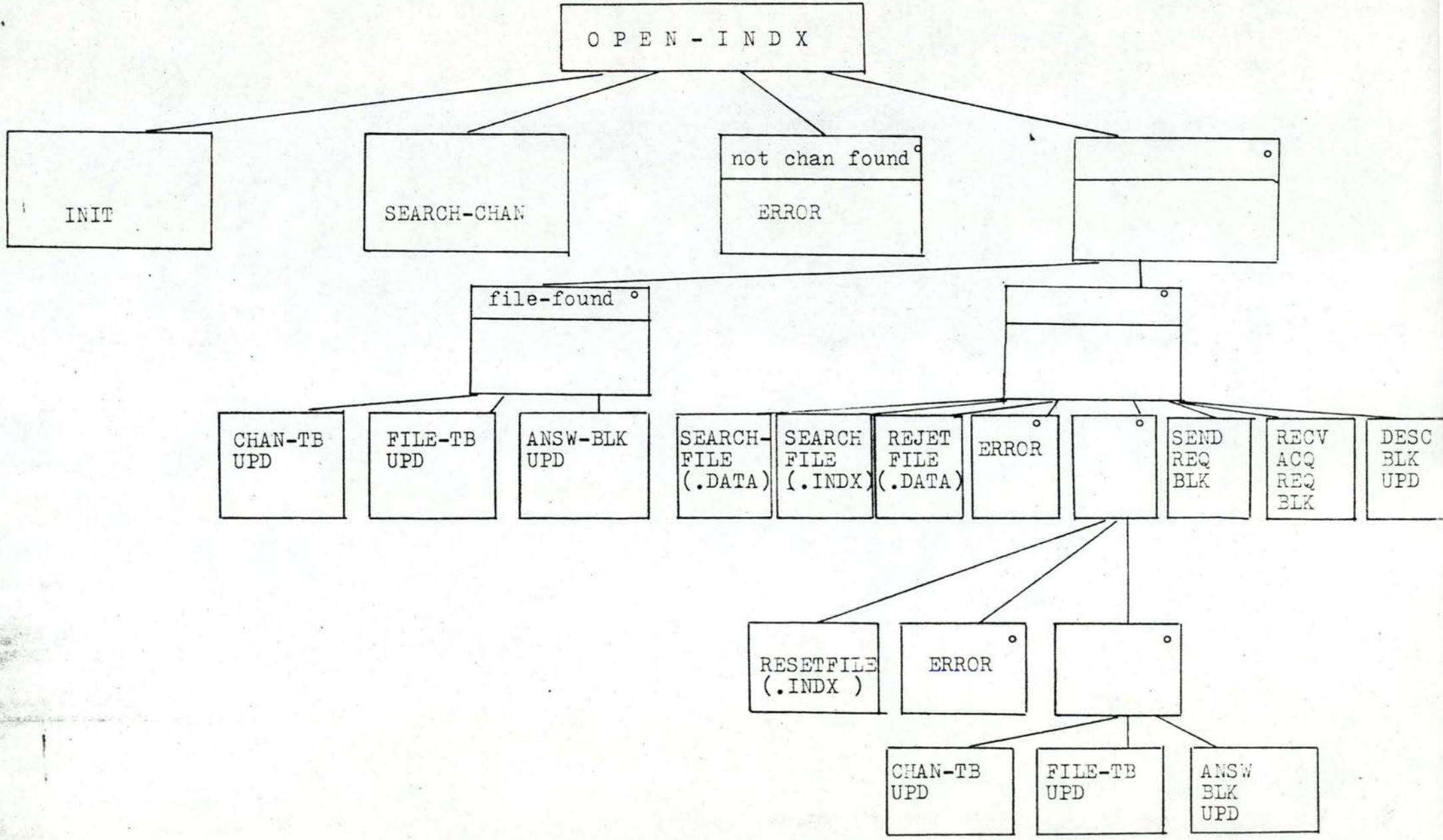

I O - P R O C
INIT
BLOCK-TRT
*
end - trt
GET - BLOCK
CODE - TRT
SEND - BLOCK
code= appendf
code= openf
code= closef
code= wrseq
code= rdseq
code= wrindx
code= rdindx
code= rdmod
code= rdnext
code= dlrec
code= mdrec
APPEND FILE
OPEN FILE
CLOSE FILE
WRITE SEQ
READ SEQ
WRITE INDX
READ INDX
READ MOD
READ NEXT
DELETE
MODIFY

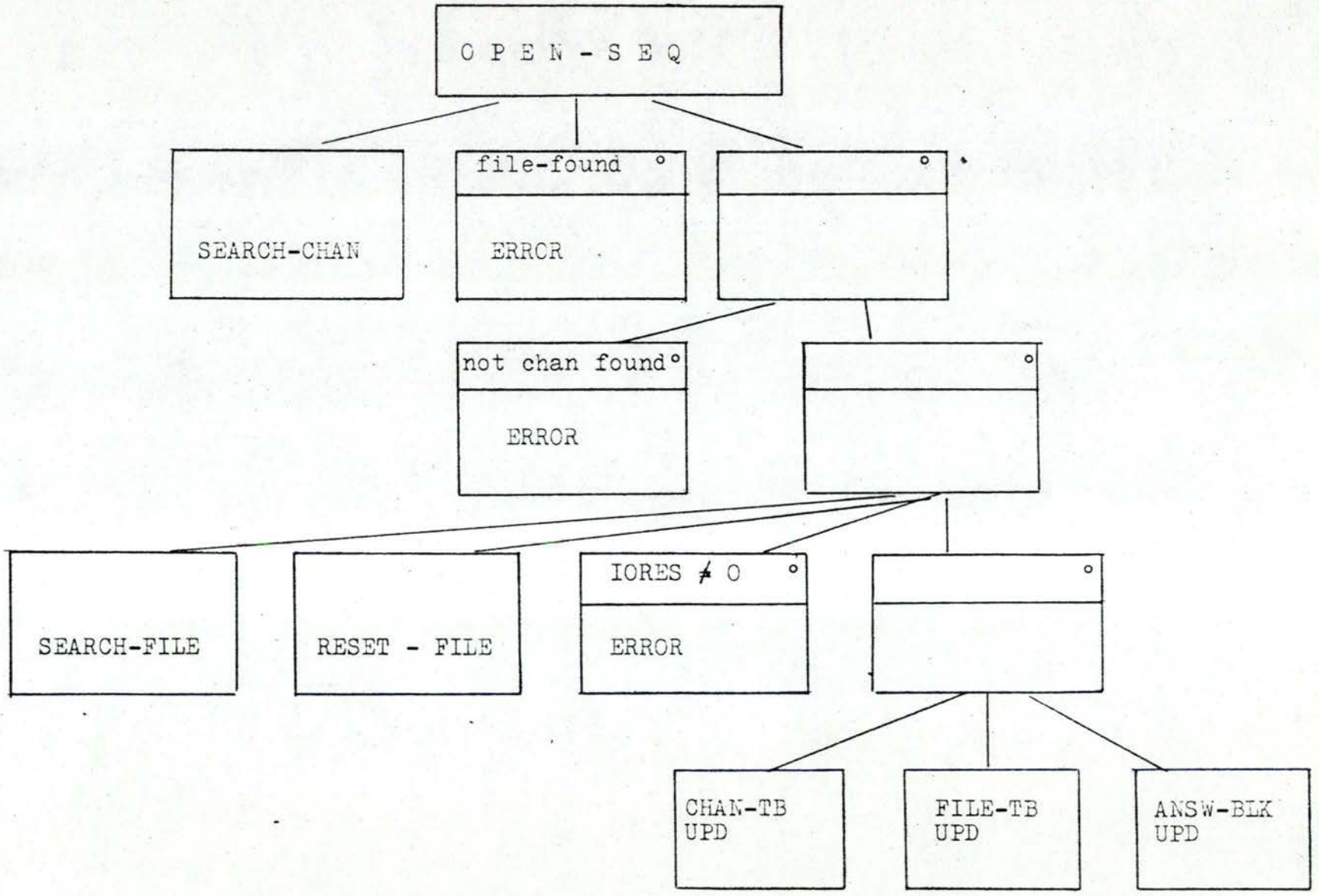

APPENDFILE
SEARCH-CHAN
file-found °
ERROR
°
not chan found°
ERROR
°
SEARCH-FILE
REWRITE-FILE
IORES ≠ 0 °
ERROR
°
CHAN-TB
UPD
FILE-TB
UPD
ANSW-BLK
UPD

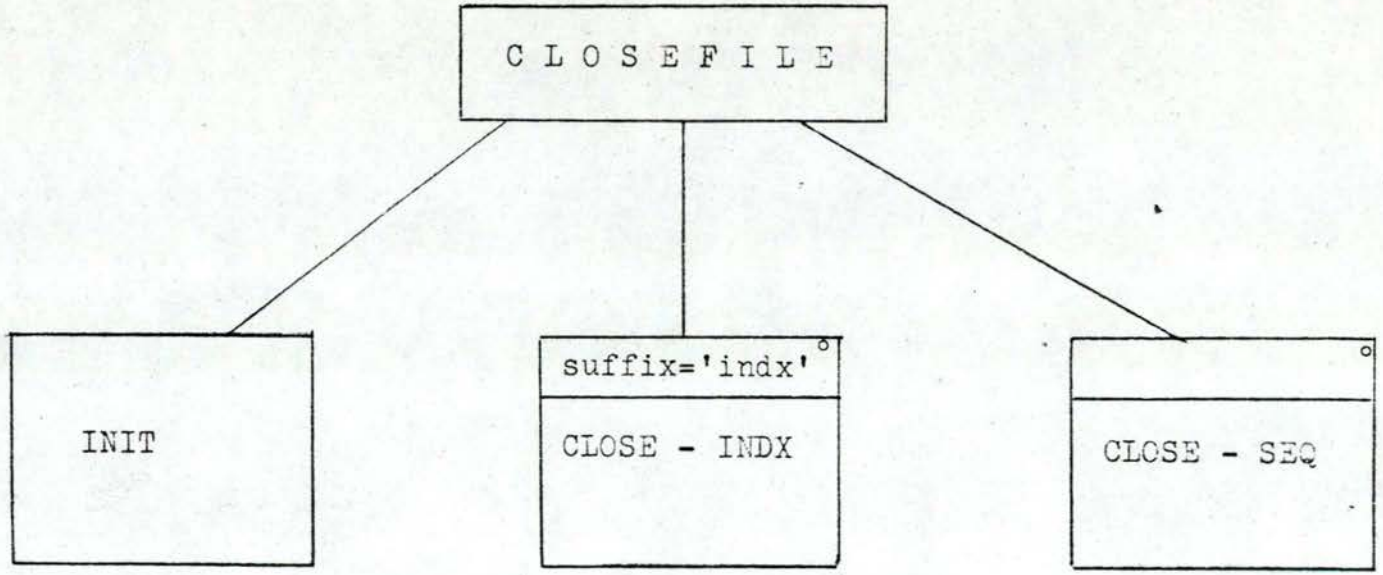

R E A D S E Q
INIT
GET - DATA
end-file
ERROR
CHAN - TB
UPD
iores ≠ 0
ERROR
ANSW-BLK
UPD

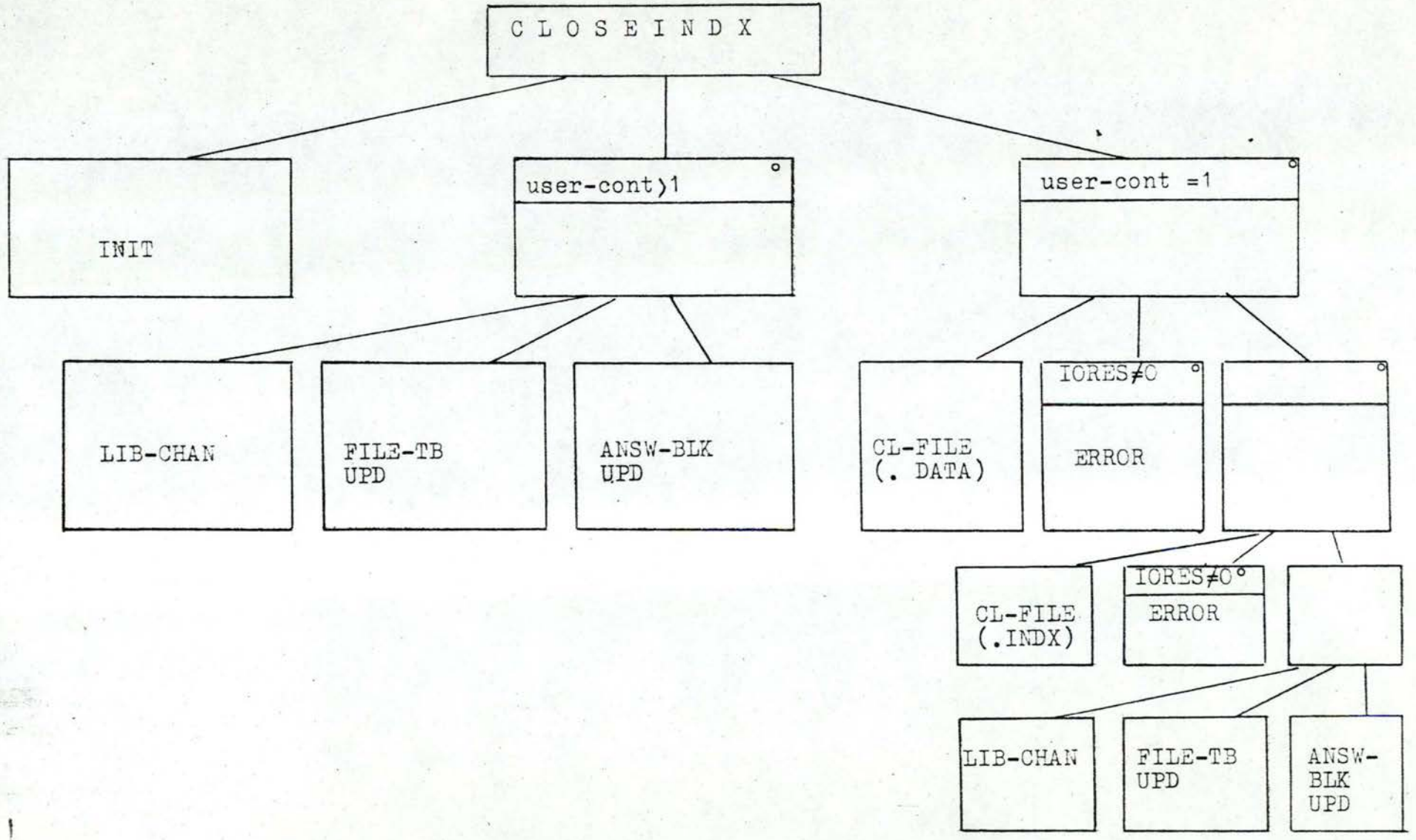

W R I T E - S E Q
INIT
PUT-DATA
iores≠0
ERROR
CHAN - TB
UPD

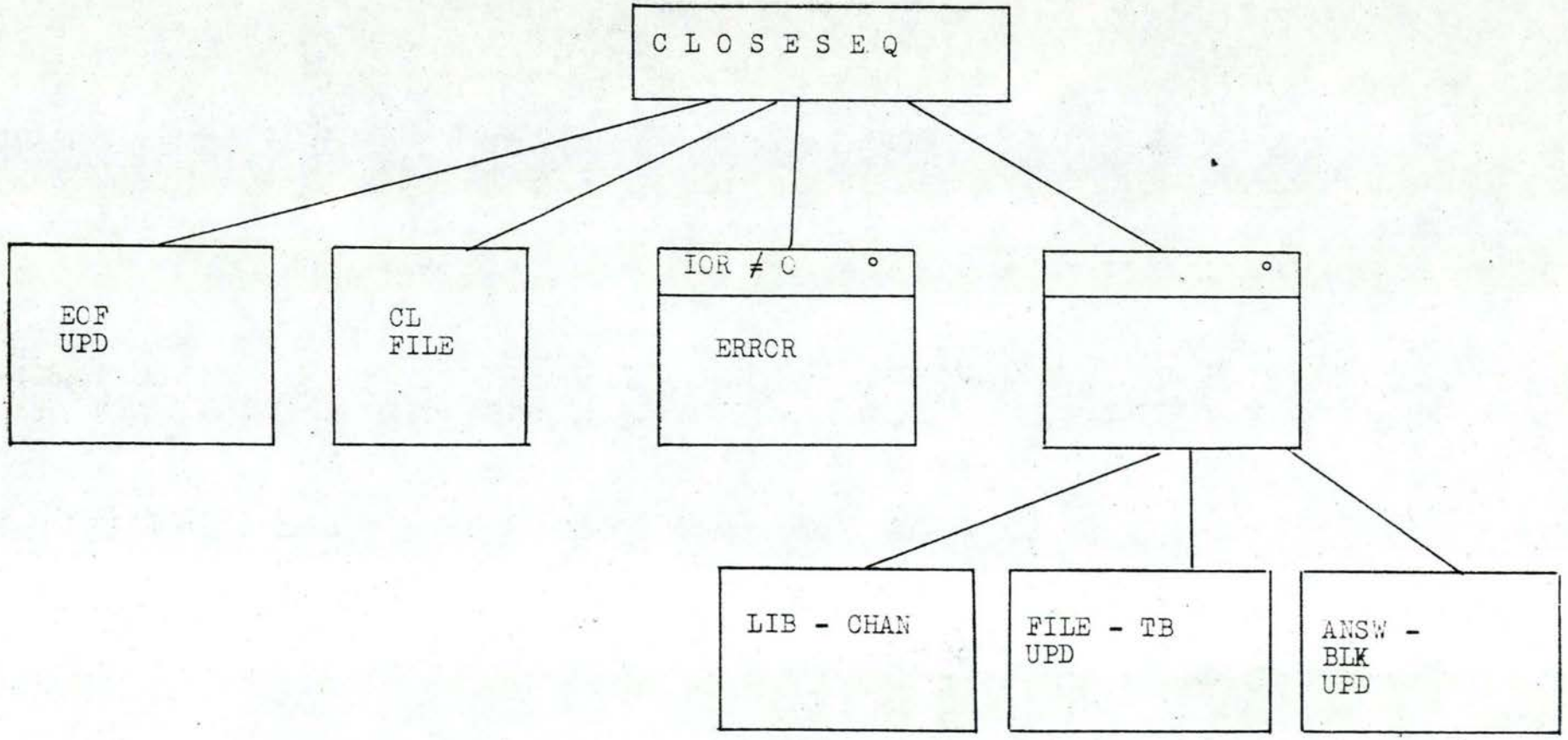

O P E N F I L E
INIT
suffix = 'indx' °
OPENINDX
°
OPENSEQ

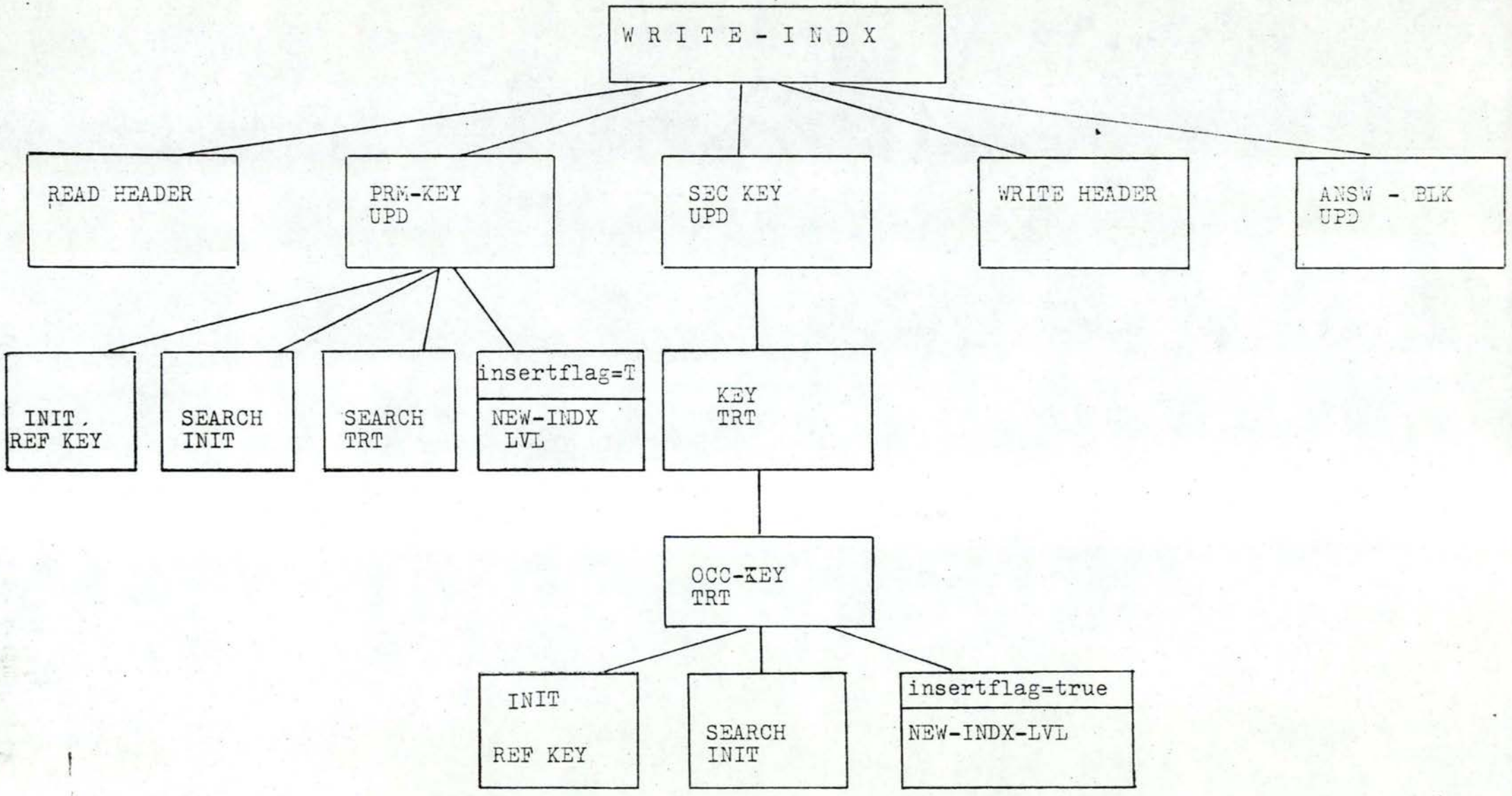

O P E N - I N D X
INIT
SEARCH-CHAN
not chan found
ERROR
file-found
CHAN-TB UPD
FILE-TB UPD
ANSW-BLK UPD
SEARCH-FILE (.DATA)
SEARCH FILE (.INDX)
REJET FILE (.DATA)
ERROR
SEND REQ BLK
RECV ACQ REQ BLK
DESC BLK UPD
RESETFILE (.INDX)
ERROR
CHAN-TB UPD
FILE-TB UPD
ANSW BLK UPD

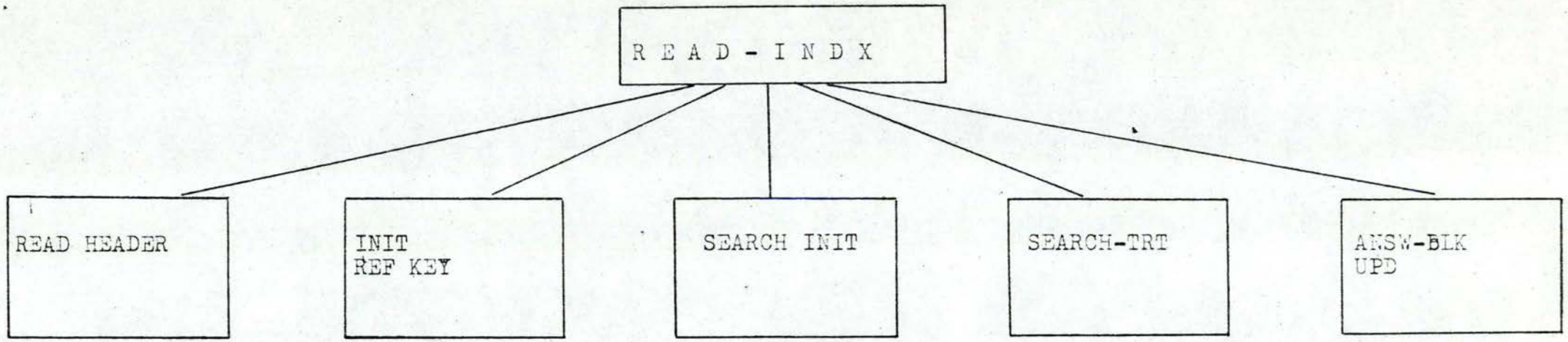

O P E N - S E Q
SEARCH-CHAN
file-found °
ERROR
°
not chan found°
ERROR
°
SEARCH-FILE
RESET - FILE
IORES ≠ 0 °
ERROR
°
CHAN-TB
UPD
FILE-TB
UPD
ANSW-BLK
UPD

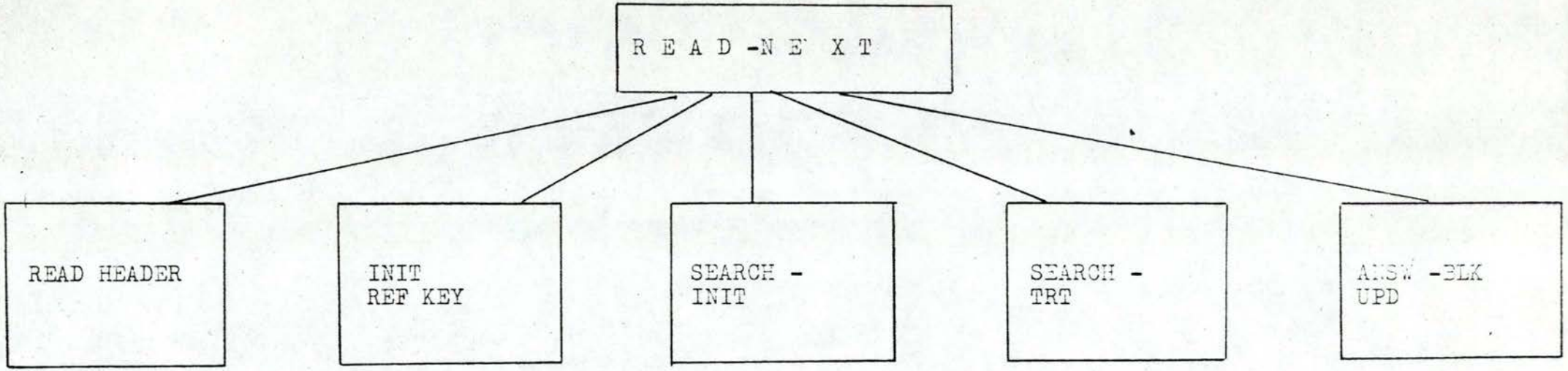

C L O S E F I L E
INIT
suffix='indx'
CLOSE - INDX
CLOSE - SEQ

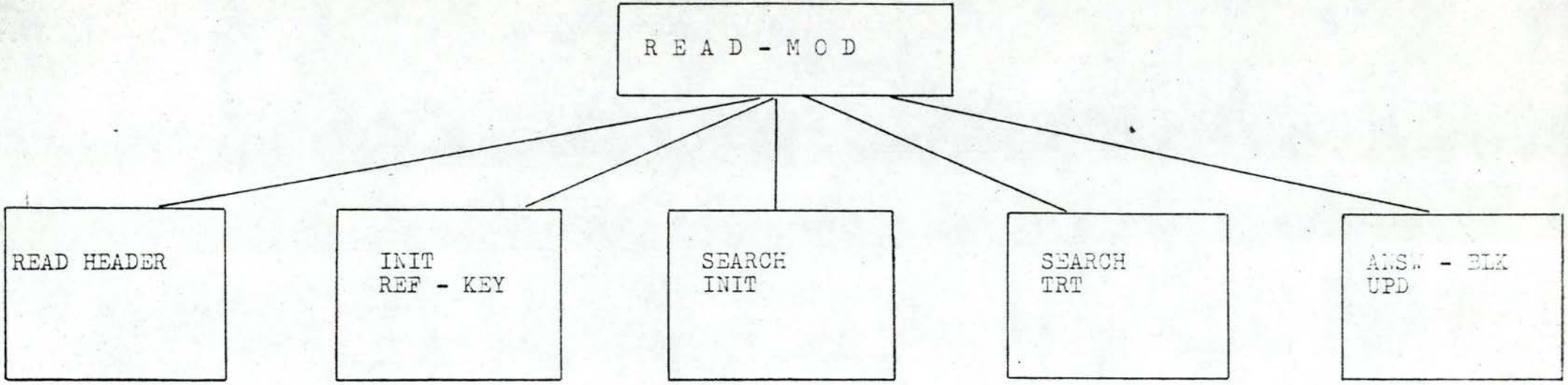

C L O S E I N D X
INIT
user-cont)1
user-cont =1
LIB-CHAN
FILE-TB
UPD
ANSW-BLK
UPD
CL-FILE
(. DATA)
IORES≠0
ERROR
CL-FILE
(.INDX)
IORES≠0
ERROR
LIB-CHAN
FILE-TB
UPD
ANSW-
BLK
UPD

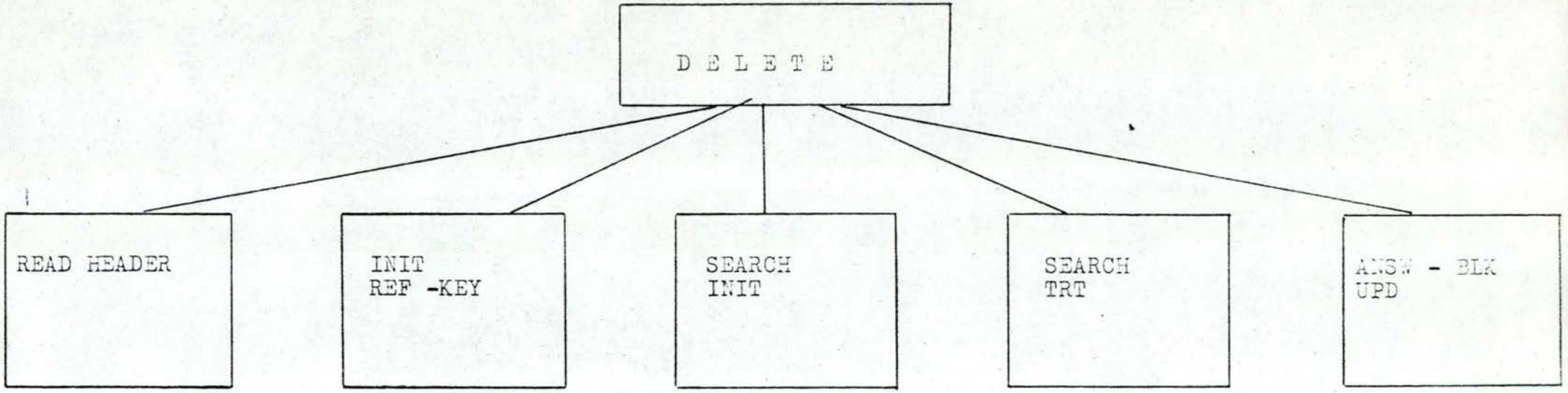

C L O S E S E Q
ECF
UPD
CL
FILE
IOR ≠ 0
ERROR
LIB - CHAN
FILE - TB
UPD
ANSW -
BLK
UPD

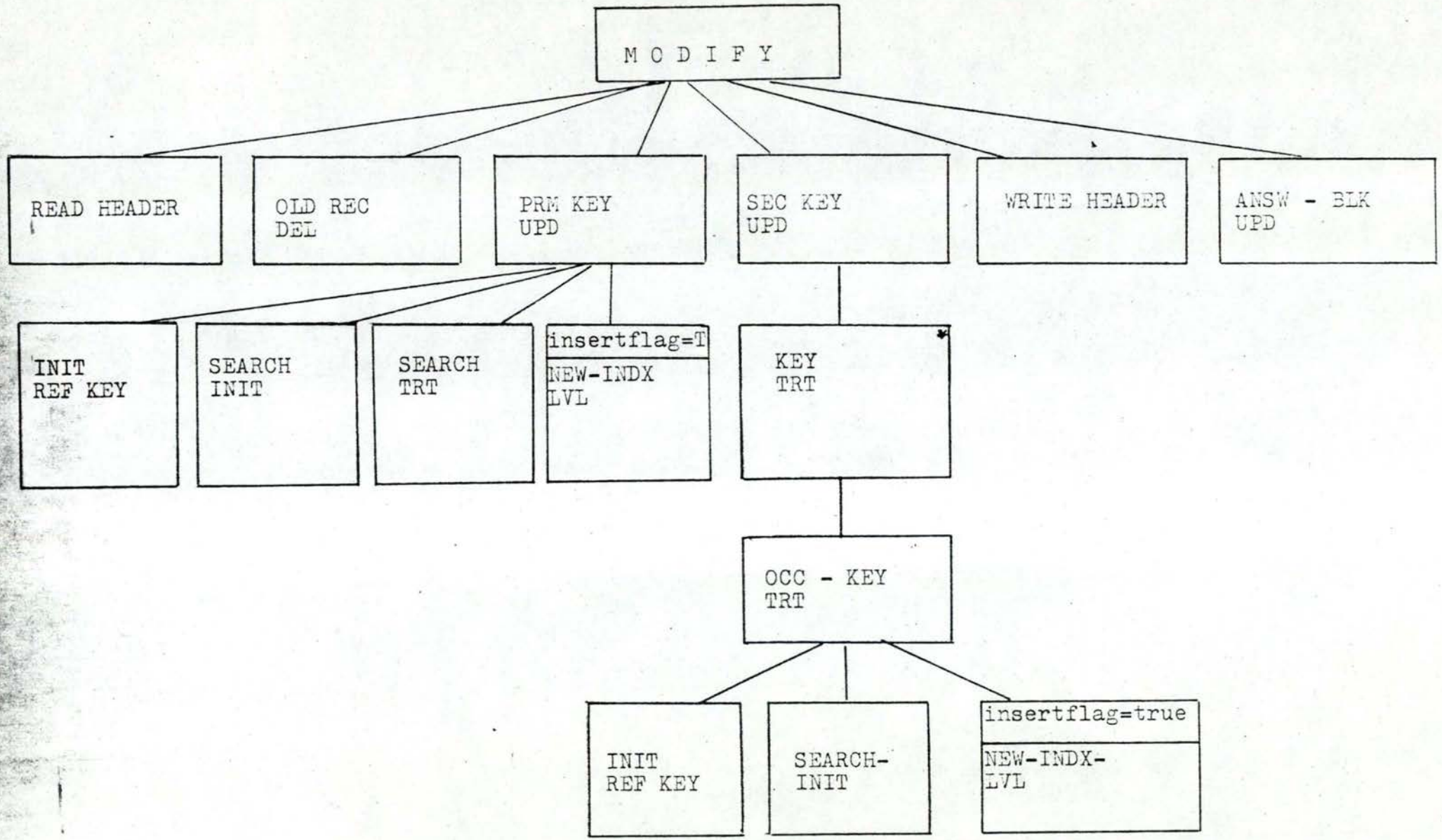

W R I T E - I N D X
READ HEADER
PRM-KEY UPD
SEC KEY UPD
WRITE HEADER
ANSW - BLK UPD
INIT. REF KEY
SEARCH INIT
SEARCH TRT
insertflag=T
NEW-INDX LVL
KEY TRT
OCC-KEY TRT
INIT
REF KEY
SEARCH INIT
insertflag=true
NEW-INDX-LVL

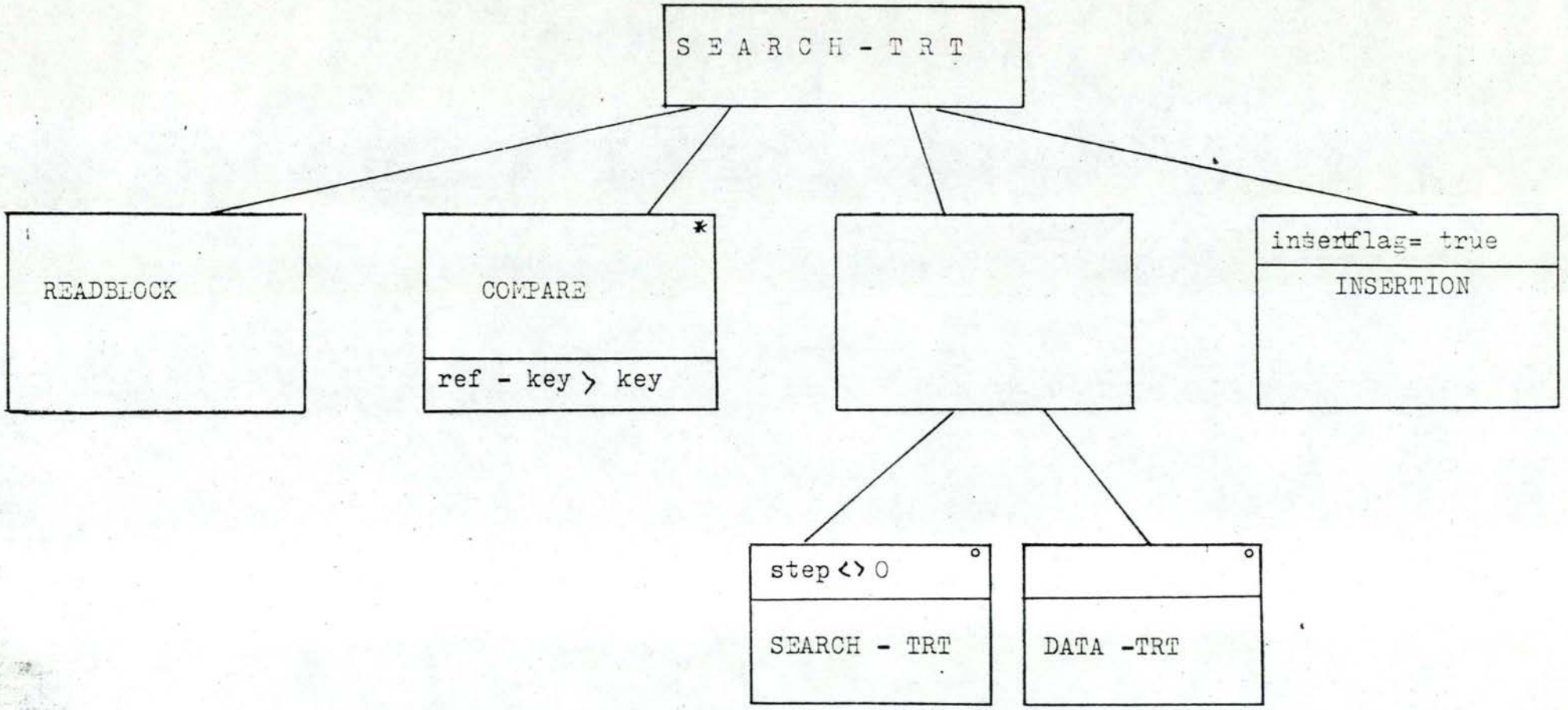

R E A D - I N D X
READ HEADER
INIT
REF KEY
SEARCH INIT
SEARCH-TRT
ANSW-BLK
UPD

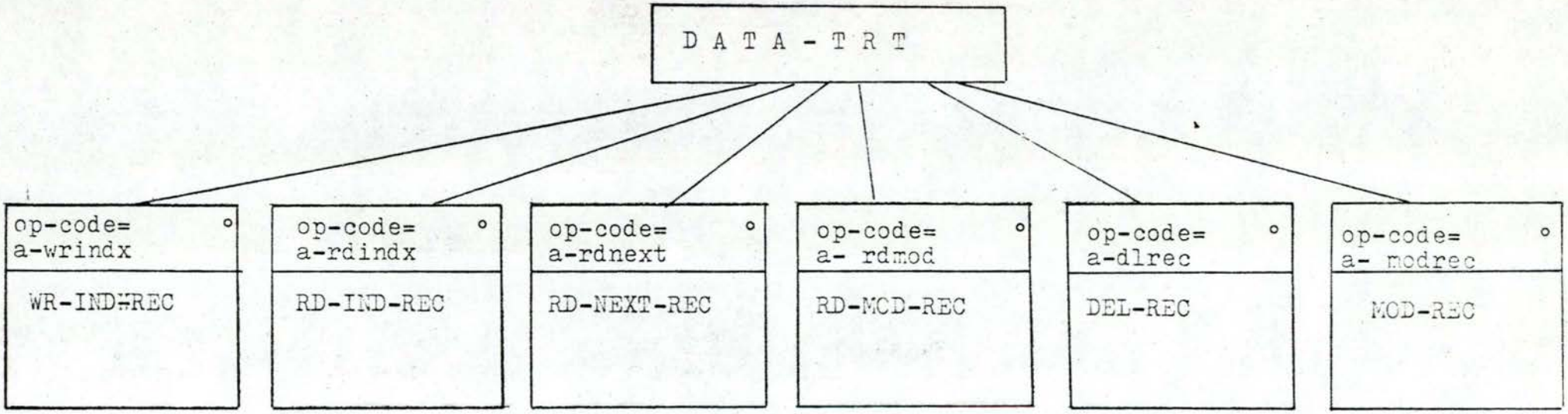

R E A D -N E X T
READ HEADER
INIT
REF KEY
SEARCH -
INIT
SEARCH -
TRT
ANSW -BLK
UPD

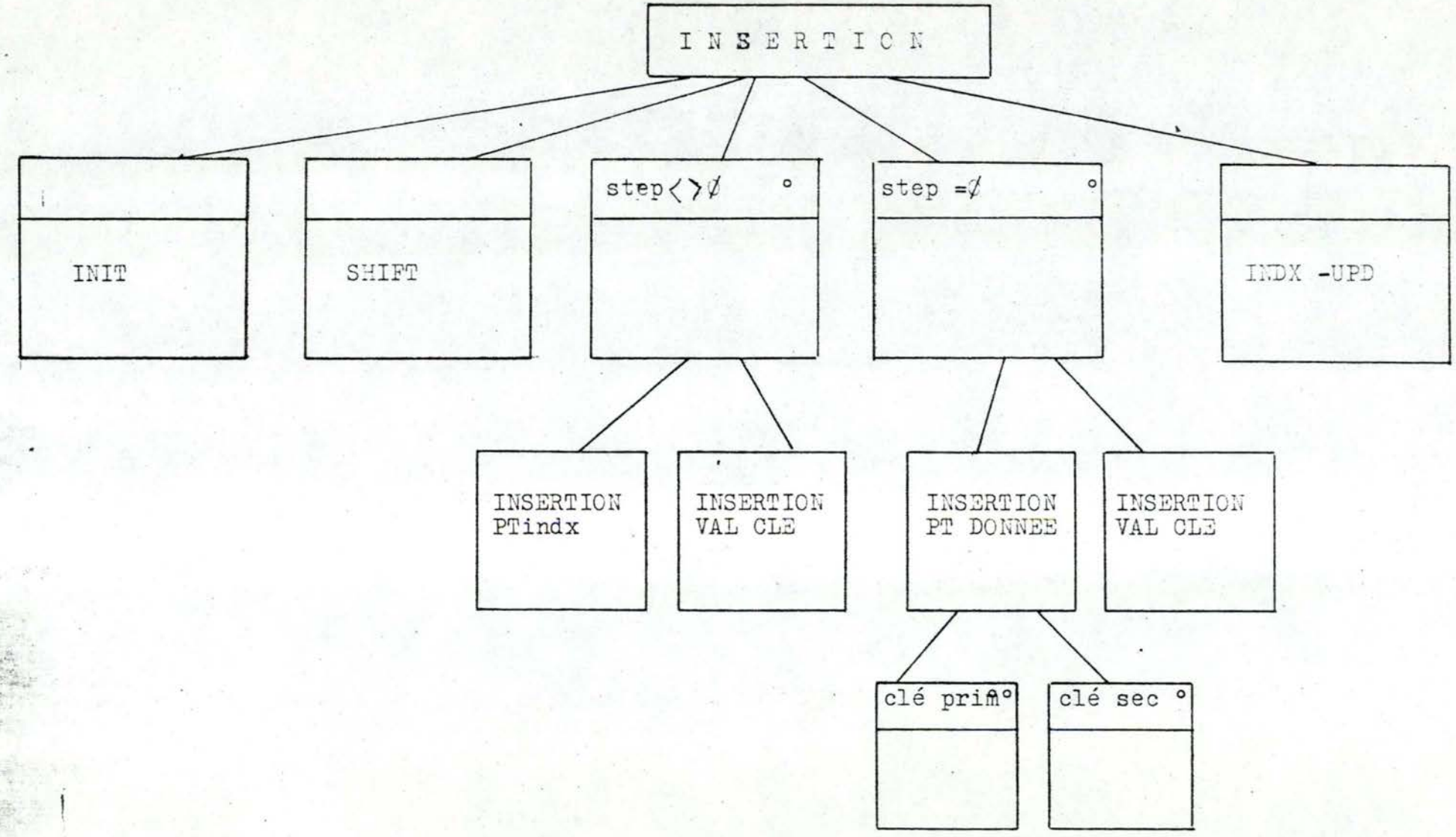

R E A D - M O D
READ HEADER
INIT
REF - KEY
SEARCH
INIT
SEARCH
TRT
ANSW - BLK
UPD

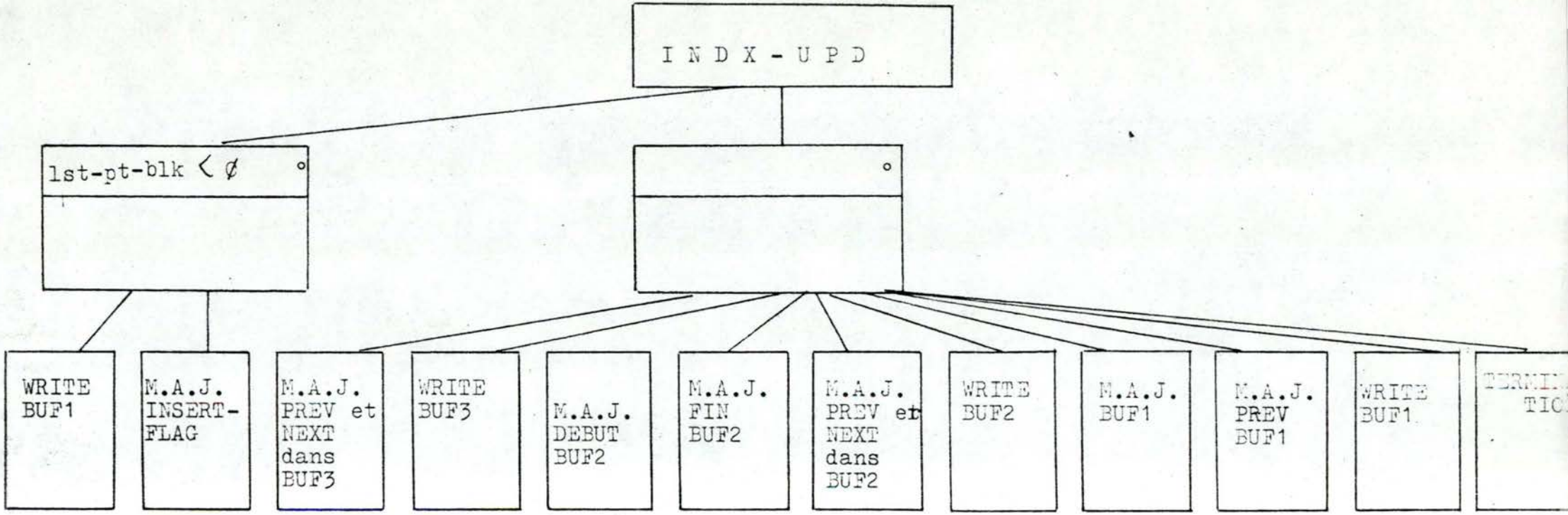

D E L E T E
READ HEADER
INIT
REF -KEY
SEARCH
INIT
SEARCH
TRT
ANSW - BLK
UPD

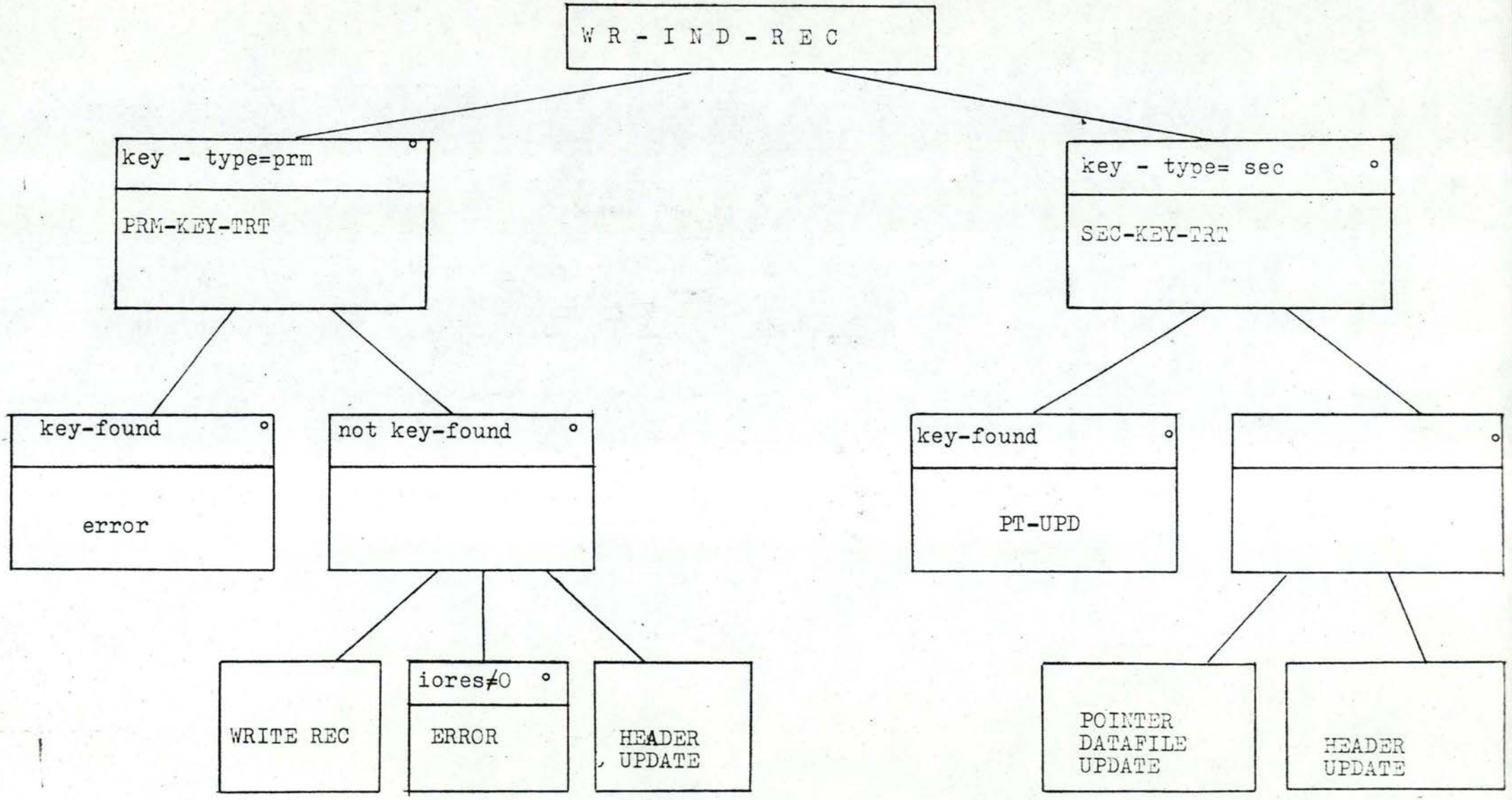

M O D I F Y
READ HEADER
OLD REC
DEL
PRM KEY
UPD
SEC KEY
UPD
WRITE HEADER
ANSW - BLK
UPD
INIT
REF KEY
SEARCH
INIT
SEARCH
TRT
insertflag=T
NEW-INDX
LVL
KEY
TRT
OCC - KEY
TRT
INIT
REF KEY
SEARCH-
INIT
insertflag=true
NEW-INDX-
LVL

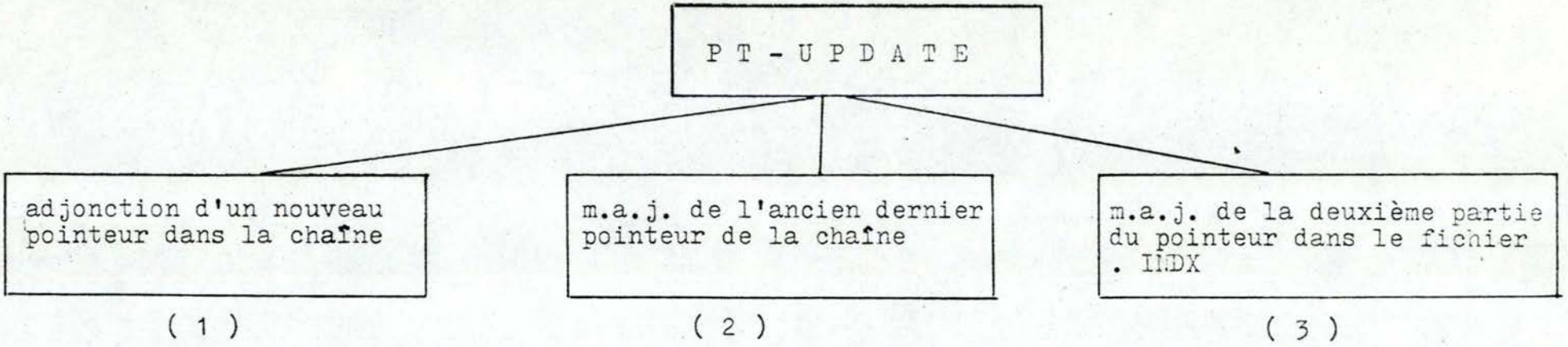

S E A R C H - T R T
READBLOCK
COMPARE
ref - key > key
insertflag= true
INSERTION
step <> 0
SEARCH - TRT
DATA -TRT

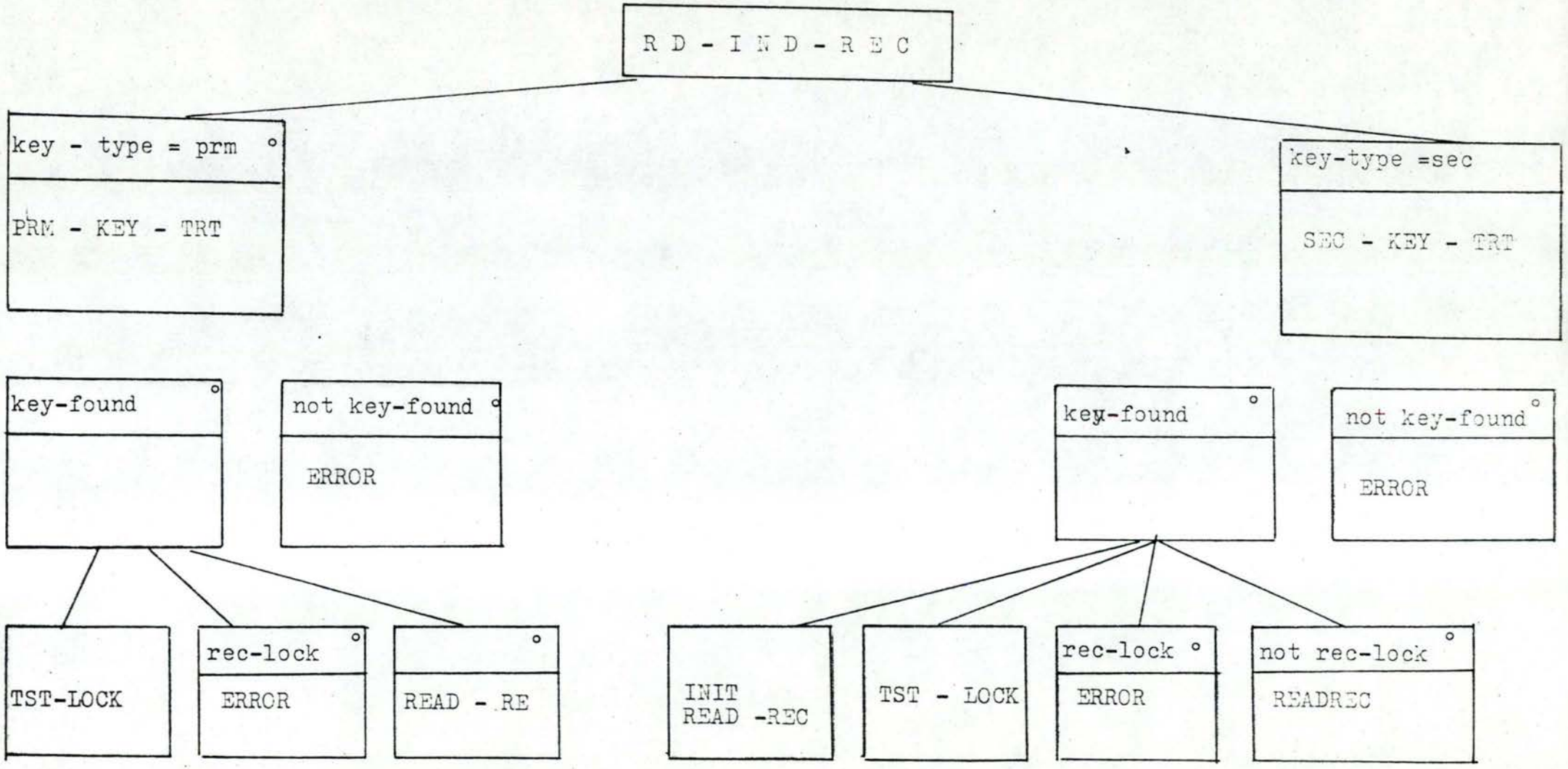

D A T A - T R T
op-code=
a-wrindx
WR-IND#REC
op-code=
a-rdindx
RD-IND-REC
op-code=
a-rdnext
RD-NEXT-REC
op-code=
a- rdmod
RD-MOD-REC
op-code=
a-dlrec
DEL-REC
op-code=
a- modrec
MOD-REC

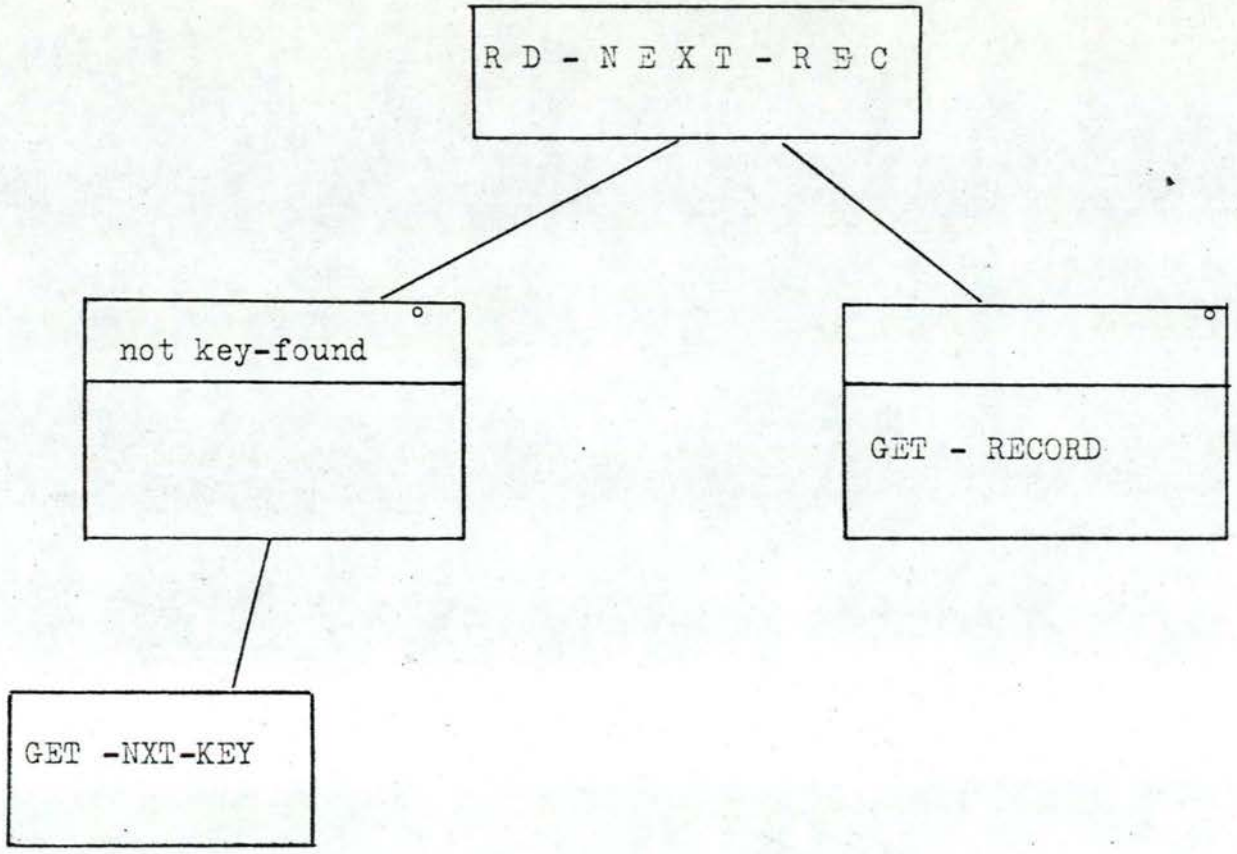

INSERTION
INIT
SHIFT
step<>∅ °
step =∅ °
INDX -UPD
INSERTION PTindx
INSERTION VAL CLE
INSERTION PT DONNEE
INSERTION VAL CLE
clé prim°
clé sec °

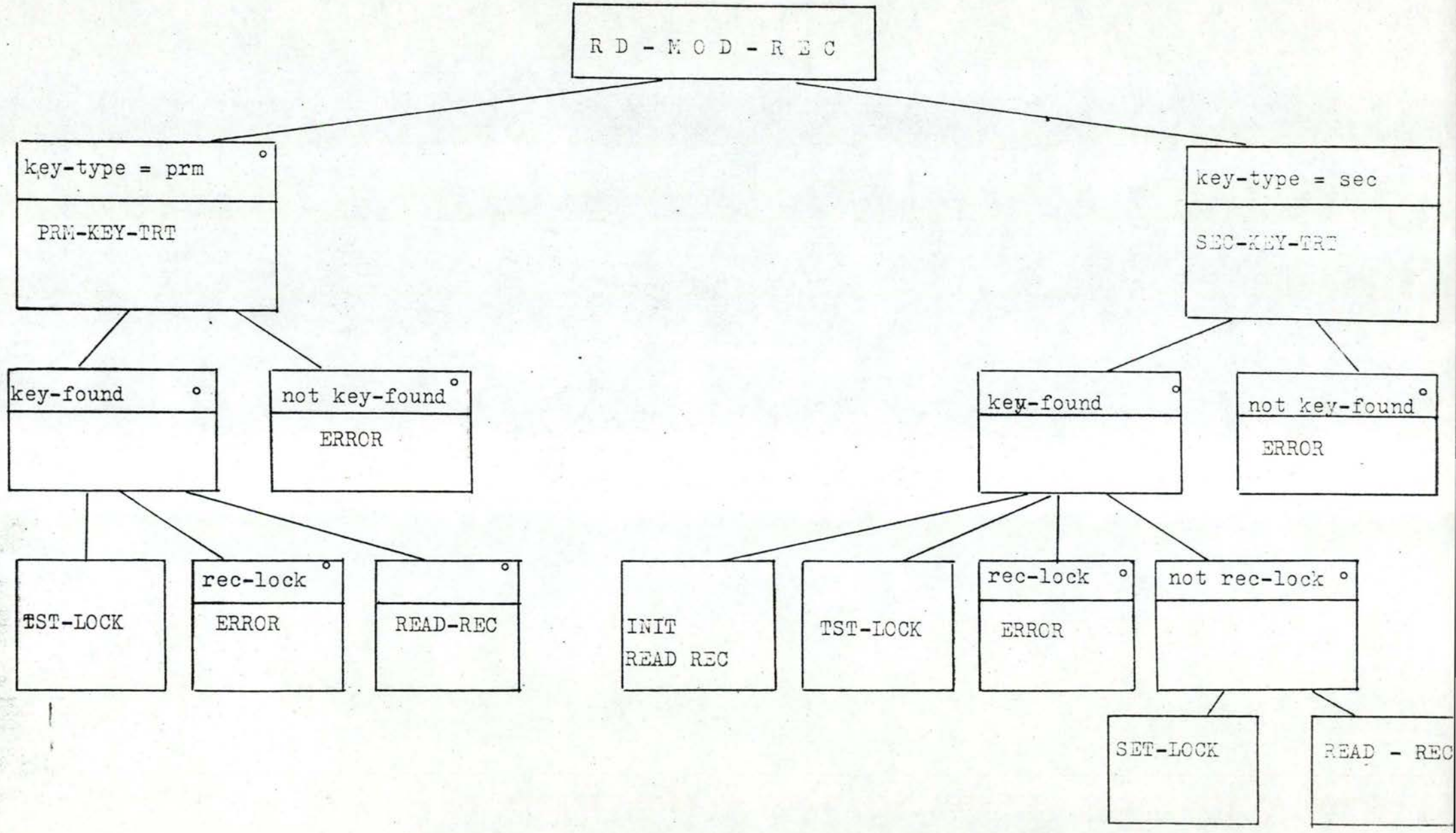

I N D X - U P D
lst-pt-blk < ∅
°
°
WRITE BUF1
M.A.J. INSERT-FLAG
M.A.J. PREV et NEXT dans BUF3
WRITE BUF3
M.A.J. DEBUT BUF2
M.A.J. FIN BUF2
M.A.J. PREV et NEXT dans BUF2
WRITE BUF2
M.A.J. BUF1
M.A.J. PREV BUF1
WRITE BUF1

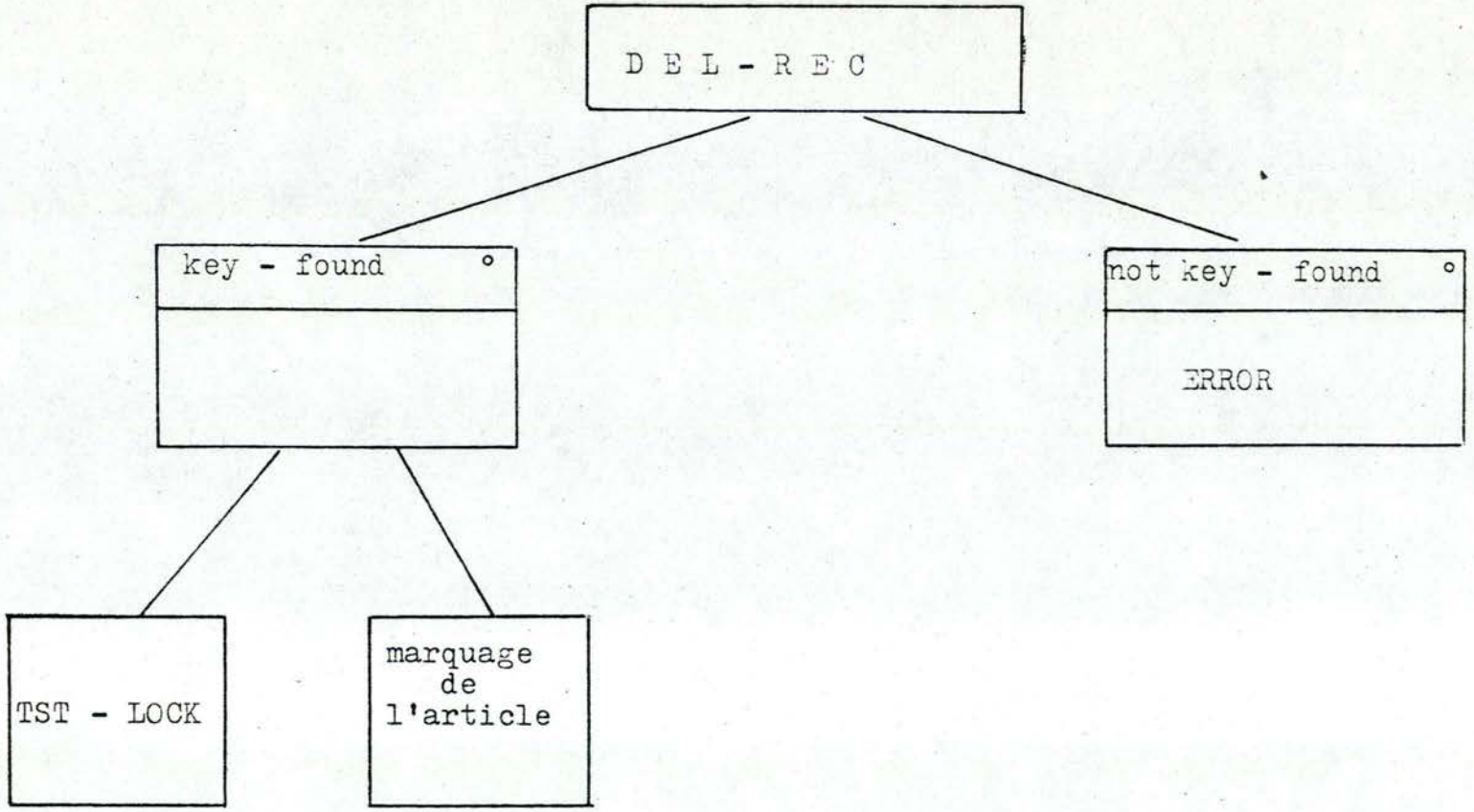

W R - I N D - R E C
key - type=prm
PRM-KEY-TRT
key - type= sec
SEC-KEY-TRT
key-found
error
not key-found
key-found
PT-UPD
WRITE REC
iores≠0
ERROR
HEADER
UPDATE
POINTER
DATAFILE
UPDATE
HEADER
UPDATE

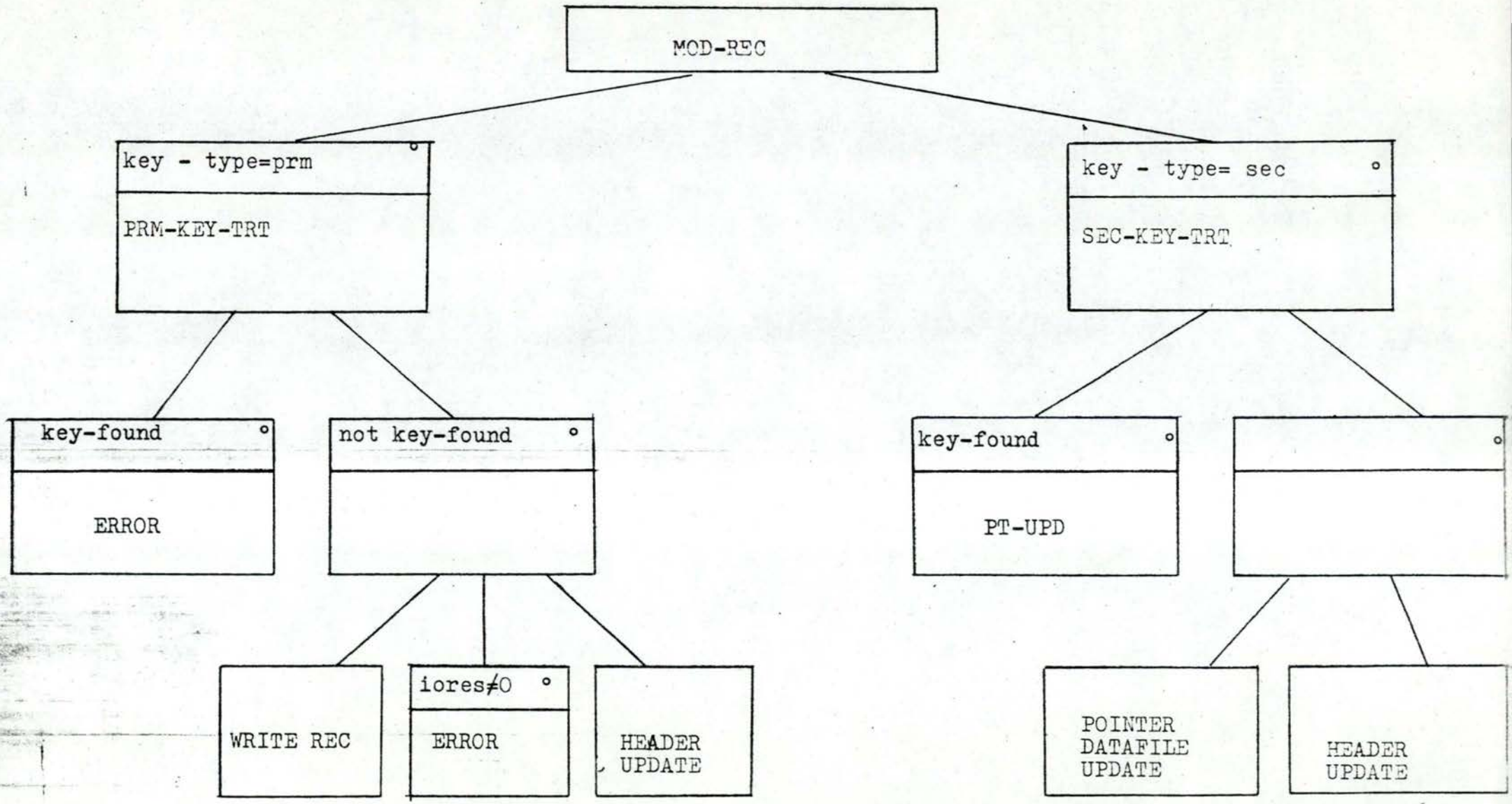

P T - U P D A T E
adjonction d'un nouveau pointeur dans la chaîne
m.a.j. de l'ancien dernier pointeur de la chaîne
m.a.j. de la deuxième partie du pointeur dans le fichier . INDX
(1)
(2)
(3)


R D - I N D - R E C
key - type = prm °
PRM - KEY - TRT
key-type =sec
SEC - KEY - TRT
key-found °
not key-found °
ERROR
key-found °
not key-found °
ERROR
TST-LOCK
rec-lock °
ERROR
°
READ - RE
INIT
READ -REC
TST - LOCK
rec-lock °
ERROR
not rec-lock °
READREC


R D - N E X T - R E C
not key-found
GET - RECORD
GET -NXT-KEY


RD-MOD-REC
key-type = prm
PRM-KEY-TRT
key-type = sec
SEC-KEY-TRT
key-found
not key-found
ERROR
key-found
not key-found
ERROR
TST-LOCK
rec-lock
ERROR
READ-REC
INIT
READ REC
TST-LOCK
rec-lock
ERROR
not rec-lock
SET-LOCK
READ - REC


D E L - R E C
key - found °
not key - found °
ERROR
TST - LOCK
marquage
de
l'article


MOD-REC
key - type=prm
PRM-KEY-TRT
key - type= sec
SEC-KEY-TRT
key-found
ERROR
not key-found
key-found
PT-UPD
WRITE REC
iores≠0
ERROR
HEADER
UPDATE
POINTER
DATAFILE
UPDATE
HEADER
UPDATE

```
(*$S+*)
PROGRAM IOPROC;

USES

        (*$U UNIT3.CODE*)
        UNIT3,

        (*$U UNIT1.CODE*)
        UNIT1,

        (*$U UNIT4.CODE*)
        UNIT4,

        (*$U UNIT2.CODE*)
        UNIT2,

        (*$U UNIT5.CODE*)
        UNIT5,

        (*$U UNIT6.CODE*)
        UNIT6;


(****************************)
(*PROCEDURE INITIALISATION;*)
(****************************)

VAR

  K,L:INTEGER;  (indices )

  {-------------------------------------------------------}
  {Cette procedure a pour effet d'initialiser             }
  {       - la table des canaux c.a.d.                    }
  {             - assurer la disponibilite de chacune des }
  {               entrees dans la table                   }
  {             - remplir certaines zones avec des valeurs}
  {               conventionnelles de base                }
  {       - la table des fichiers (idem)                  }
  {-------------------------------------------------------}

BEGIN (*procedure initialisation*)

  (* initialisation de la table des canaux *)

  FOR K := Ø TO CHAN_TB_LGTH DO
      BEGIN
        CHAN_TB[K].BUSY      := FALSE;
        CHAN_TB[K].FILE_NAME := '                         ';
        CHAN_TB[K].CORR      := -1;

        FOR L := 1 TO MAX_LOCK_REC DO
        WITH CHAN_TB [K].LOCK_REC_TB [L] DO
        BEGIN
          BUSY := FALSE;
          LOCK_REC_BLK := Ø;
          LOCK_REC_LOC := Ø;
        END;
      END;

  (* initialisation de la table des fichiers *)
```

```
  FOR K := Ø TO FILE_TB_LGTH DO
      BEGIN
        FILE_TB[K].BUSY := FALSE;
        FILE_TB[K].USERCOUNT := Ø;
      END;
END; (* procedure initialisation*)

(*******************************************************)


(*********************)
PROCEDURE BLOCK_TRT;
(*********************)

  (*********************)
  PROCEDURE GET_BLOCK;
  (*********************)

    {-----------------------------------------------------}
    {Cette procedure a pour effet d'assurer la reception du }
    {bloc requete envoye par l'utilisateur sur le reseau    }
    {depuis sa station terminale.                           }
    {-----------------------------------------------------}

  BEGIN (*procedure getblock*)

    BLOCK[I_LOCAL] := FL_SVR_AD;
    BLOCK[I_CORR]  := Ø;   (* en donnant Ø comme adresse de
                           correspondant, on fait en sorte
                           que la station gerant le micro-
                           disque accepte le bloc qui lui
                           est adresse quelle que soit son
                           origine *)

    UNITREAD (18,BLOCK[4],1,Ø,Ø);

    CORR           := BLOCK[CORR];

  END; (*procedure getblock *)

  (*******************************************************)

  (*********************)
  PROCEDURE CODE_TRT;
  (*********************)

    {-----------------------------------------------------}
    {Cette procedure a pour effet d'analyser le code fourni }
    {par l'utilisateur dans le bloc requete et de declencher }
    {le traitement approprie.                               }
    {-----------------------------------------------------}

  BEGIN (*procedure codetrt*)

    (* Les specifications precises des diverses procedures
       invoquees ici se trouvent detaillees dans le texte
       meme de ces procedures *)

    CASE BLOCK[I_CODE] OF

      APPENDF: APPENDFILE;
      OPENF  : OPENFILE;
      CLOSEF : CLOSEFILE;
      WRSEQ  : WRITESEQ;
```

```
        RDSEQ   : READSEQ;
        WRINDX  : WRITEINDX;
        RDINDX  : READINDX;
        RDNEXT  : READNEXT;
        RDMOD   : READMOD;
        DLREC   : DELETE;
        MDREC   : MODIFY;

      END;

    END; (*procedure codetrt*)

    (*************************************************************)

    (***********************)
    PROCEDURE SEND_BLOCK;
    (***********************)

    BEGIN (*procedure sendblock*)

      (-------------------------------------------------------)
      (Cette procedure a pour effet d'assurer l'envoi du bloc )
      (reponse a la station terminale d'ou emanait la requete )
      (-------------------------------------------------------)

      (*BLOCK_LGTH est une variable de type entier qui
      contient la longeur du block a renvoyer. Cette
      valeur doit etre mentionnee dans le bloc reponse de
      la facon suivante: 1) le byte de poids fort
                         2) le byte de poids faible *)

      BLOCK [I_BLOCK_LOW] := BLOCK_LGTH MOD 256;
      BLOCK [I_BLOCK-HIGH]:= BLOCK_LGTH DIV 256;

      UNITWRITE (18,BLOCK[4],1,Ø,Ø);

    END;(* procedure sendblock*)

    (*************************************************************)

  BEGIN (*procedure blocktrt*)

    GET_BLOCK;
    CODE_TRT;
    SEND_BLOCK;

  END; (*procedure blocktrt*)

  (*************************************************************)

  BEGIN (*program principal*)

    INITIALISATION;
    REPEAT
      BLOCK_TRT
    UNTIL BLOCK[I_CODE] = END_TRT;

  END. (*program principal*)
```

```
(*$S+*)
UNIT UNIT1;

INTERFACE

USES
        (*$U UNIT3.CODE*)
        UNIT3;


PROCEDURE SEARCH_CHAN (SEARCH_NAME                  : FILETITLE;
                       VAR FILE_FOUND,CHAN_FOUND    : BOOLEAN;
                       VAR FREE_CH_NO,EX_CH_NO      :INTEGER);

PROCEDURE SEARCH_FILE (VAR FILE_FOUND:BOOLEAN;
                       VAR FILE_NO:INTEGER);

PROCEDURE APPEND_FILE;

PROCEDURE OPEN_FILE;

PROCEDURE CLOSE_FILE;


IMPLEMENTATION

(*************************************************************)
PROCEDURE SEARCH_CHAN (*SEARCH_NAME                  : FILE_TITLE;
                       VAR FILE_FOUND,CHAN_FOUND    : BOOLEAN;
                       VAR FREE_CH_NO,EX_CH_NB      : INTEGER*);
(*************************************************************)

  {-------------------------------------------------------}
  {Cette procedure effectue deux choses                   }
  {                                                       }
  {     - elle recherche dans la table des canaux si le   }
  {       fichier de nom SEARCH_NAME est deja ouvert      }
  {         si oui : FILE_FOUND = TRUE                    }
  {                  EX_CH_NO   = numero de canal dans la }
  {                               table correspondant a   }
  {                               ce fichier              }
  {         si non : FILE_FOUND = FALSE                   }
  {                  EX_CH_NO   = indetermine             }
  {                                                       }
  {     - elle recherche dans la table des canaux s'il    }
  {       existe un canal de libre                        }
  {         si oui : CHAN_FOUND = TRUE                    }
  {                  FREE_CH_NO = numero de canal libre   }
  {         si non : CHAN_FOUND = FALSE                   }
  {                  FREE_CH_NO = indetermine             }
  {-------------------------------------------------------}
VAR

 K:INTEGER;  { indice dans la table des canaux }

BEGIN (*procedure searchan*)
```

```
  REPEAT

      IF NOT FILE_FOUND
      THEN
        BEGIN
          FILE_FOUND := CHAN_TB[K].FILE_NAME=SEARCH_NAME;
          EX_CH_NO   := K;
        END;

      IF (NOT CHAN_FOUND) AND (CHAN_TB[K].BUSY=FALSE)
      THEN
        BEGIN
          CHAN_FOUND:= TRUE;
          FREE_CH_NO:= K;
        END;

      K := K+1;

  UNTIL (CHAN_FOUND AND FILE_FOUND) OR (K > CHAN_TB_LGTH);

END; (*procedure searchan*)

(*************************************************************)

(*****************************************************)
  PROCEDURE SEARCH_FILE (*VAR FILE_FOUND: BOOLEAN;
                          VAR FILE_NO   : INTEGER*);
(*****************************************************)

  {-----------------------------------------------------------}
  {Cette procedure a pour objet de parcourir la table des     }
  {canaux en vue d'y trouver un fichier libre                 }
  {  En cas de succes : FILE_FOUND = TRUE                     }
  {                     FILE_NO    = indice, dans la table    }
  {                                  des canaux, correspon-   }
  {                                  dant au fichier libre    }
  {                     L'entree dans la table est alors      }
  {                     requisitionnee                        }
  {  En cas d'echec   : FILE_FOUND = FALSE                    }
  {                     FILE_NO    = indetermine              }
  {-----------------------------------------------------------}

VAR

  K :INTEGER;  {indice dans la table des canaux}

BEGIN (*procedure searchfile*)

  K:= Ø;

  REPEAT
    BEGIN
      FILE_FOUND := FILE_TB[K].BUSY = FALSE;
      K := K+1;
    END
  UNTIL (FILE_FOUND) OR (K > FILE_TB_LGTH);

  FILE_NO := K-1;
  FILE_TB[FILE_NO].BUSY := TRUE;
```

```
(************************)
  PROCEDURE APPEND_FILE;
(************************)

  {-------------------------------------------------------------}
  {Cette procedure a pour objet de creer, a la requete de      }
  {l'utilisateur, un fichier de type sequentiel et de le       }
  {ouvert en vue de lui permettre ulterieurement d'y faire des}
  {operations d'entrees/sorties                                }
  {-------------------------------------------------------------}

CONST

  BLANK = 32;

VAR

  K           : INTEGER;
  L           : INTEGER;
  EOF_STR     : BOOLEAN;
  FILE_FOUND  : BOOLEAN;
  CHAN_FOUND  : BOOLEAN;
  FREE_CH_NO  : INTEGER;
  EX_CH_NO    : INTEGER;
  SEQ_FL_NO   : INTEGER;

  (*********************************************)
  PROCEDURE REWRITE_FILE (FILE_NO  :INTEGER;
                          FILE_NAME:FILETITLE);
  (*********************************************)

    {-------------------------------------------------------}
    {Cette procedure a pour effet d'aller creer un fichier  }
    {de nom FILE_NAME dont l'identifiant est donne par      }
    {FILE_NO                                                }
    {-------------------------------------------------------}


  BEGIN (*procedure rewritefile*)
    CASE FILE_NO OF
      0: REWRITE (FILE0,FILE_NAME);
      1: REWRITE (FILE1,FILE_NAME);
      2: REWRITE (FILE2,FILE_NAME);
      3: REWRITE (FILE3,FILE_NAME);
      4: REWRITE (FILE4,FILE_NAME);
      5: REWRITE (FILE5,FILE_NAME);
      6: REWRITE (FILE6,FILE_NAME);
      7: REWRITE (FILE7,FILE_NAME);
      8: REWRITE (FILE8,FILE_NAME);
      9: REWRITE (FILE9,FILE_NAME);
     10:REWRITE (FILE10,FILE_NAME);
     11:REWRITE (FILE11,FILE_NAME);
     12:REWRITE (FILE12,FILE_NAME);
     13:REWRITE (FILE13,FILE_NAME);
     14:REWRITE (FILE14,FILE_NAME);
     15:REWRITE (FILE15,FILE_NAME);
    END;
  END; (*procedure rewritefile*)
```

```
BEGIN (*procedure appendfile*)

  (* initialisation *)


  (* Cette phase d'initialisation a pour objet de ranger
     de ranger dans la variable FILE_NAME le nom de fichier
     passe par l'utilisateur dans son bloc requete, nom
     constitue d'une chaine de caracteres Ascii *)

  K:= Ø; L:= I_FL_NM_BEG;

  REPEAT
    BEGIN

      EOF_STR :=      (BLOCK[L]= BLANK)
                  OR (BLOCK[L]=13)
                  OR (BLOCK[L]=Ø);

      IF NOT EOF_STR
      THEN
        BEGIN
          FILE_NAME.ARR[K]:= BLOCK[L];
          K := K+1;
          L := L+1;
        END
      ELSE
        FILE_NAME.ARR[-1] := K;
    END

  UNTIL EOF_STR;


  SEARCH_CHAN (FILE_NAME.STR,
               FILE_FOUND,CHAN_FOUND,
               FREE_CH_NO,EX_CH_NO);

  IF FILE_FOUND
  THEN
    ERROR (1ØØ)
  ELSE
    IF NOT CHAN_FOUND
    THEN
      ERROR (1Ø1)
    ELSE
      BEGIN
        SEARCH_FILE(FILE_FOUND,SEQ_FL_NO);
        REWRITE_FILE(SEQ_FL_NO,FILE_NAME.STR);
        IF IORESULTS <> Ø
        THEN
          ERROR (1Ø1)
        ELSE
          BEGIN

            (* mise a jour de la table des canaux *)

            CHAN_TB [FREE_CH_NO] .BUSY         := TRUE;
            CHAN_TB [FREE_CH_NO] .FILE_NAME    := FILE_NAME.STR;
            CHAN_TB [FREE_CH_NO] .CORR         := CORR;
            CHAN_TB [FREE_CH_NO] .FST_FL_NO    := SEQ_FL_NO;
            CHAN_TB [FREE_CH_NO] .SEC_FL_NO    := -1;
```

```
            * mise a jour de la table des fichiers *)

            FILE_TB[SEQ_FL_NO] .BUSY      := TRUE;
            FILL_TB[SEQ_FL_NO] .USERCOUNT := 1;

            (* mise a jour du block reponse *)

            BLOCK_LGTH         := 3;
            BLOCK[I_RET_CODE] := Ø;
            BLOCK[I_ERR_CODE] := Ø;
            BLOCK [I_RET_CHAN]  := FREE_CH_NO;
          END
      END
  END; (*procedure appendfile*)

  (*************************************************************)

(**********************)
  PROCEDURE OPEN_FILE;
(**********************)

  {-----------------------------------------------------------}


CONST

  BLANK = 32;


VAR

  K          : INTEGER;
  L          : INTEGER;
  EOFNAME    : INTEGER;
  EOF_STR    : BOOLEAN;
  TWO_POINTS: BOOLEAN;
  SUFFIX     : STRING[24];

  (*****************************************)
  PROCEDURE RESET_FILE (FILE_NO   : INTEGER;
                        FILE_NAME : FILETITLE);
  (*****************************************)

    {-----------------------------------------------------}
    {Cette procedure a pour effet d'ouvrir un fichier      }
    {existant de nom FILE_NAME identifie par FILE_NO       }
    {-----------------------------------------------------}

  BEGIN (*procedure resetfile*)

    CASE FILE_NO OF
      Ø: RESET (FILEØ,FILE_NAME);
      1: RESET (FILE1,FILE_NAME);
      2: RESET (FILE2,FILE_NAME);
      3: RESET (FILE3,FILE_NAME);
      4: RESET (FILE4,FILE_NAME);
      5: RESET (FILE5,FILE_NAME);
      6: RESET (FILE6,FILE_NAME);
      7: RESET (FILE7,FILE_NAME);
      8: RESET (FILE8,FILE_NAME);
```

```
    12:RESET (FILE12.FILE_NAME);
    13:RESET (FILE13.FILE_NAME);
    14:RESET (FILE14.FILE_NAME);
    15:RESET (FILE15.FILE_NAME);
  END;
END; (*procedure resetfile*)

(*************************************************************)

(*********************)
  PROCEDURE OPEN_SEQ;
(*********************)

VAR
  SEQ_FL_NO   : INTEGER;
  FILE_FOUND  : BOOLEAN;
  CHAN_FOUND  : BOOLEAN;
  FREE_CH_NO  : INTEGER;
  EX_CH_NO    : INTEGER;
  BUF         : T_BUFFER;
  SQENDLOC    : WORD;
  SQENDBLK    : WORD;

BEGIN

  SEARCH_CHAN (FILE_NAME.STR,
               FILE_FOUND,CHAN_FOUND,
               FREE_CH_NO,EX_CH_NO);

  IF FILE_FOUND
  THEN
    ERROR (101)
  ELSE
    IF NOT CHAN_FOUND
    THEN
      ERROR (102)
    ELSE
      BEGIN

        SEARCH_FILE (FILE_FOUND,SEQ_FL_NO);

        RESET_FILE (SEQ_FL_NO,FILE_NAME.STR);

        IF IORESULTS <> 0
        THEN
          ERROR (103)
        ELSE
          BEGIN

            (* mise a jour de la table des canaux *)

            CHAN_TB [FREE_CH_NO].CORR := CORR;
            CHAN_TB [FREE_CH_NO].FILE_NAME := FILE_NAME.STR ;

            WITH CHAN_TB[FREE_CH_NO] DO
            BEGIN
              BUSY             := TRUE;
              FST_FL_NO        := SEQ_FL_NO;
              SEC_FL_NO        := -1;
              SEQCRTBLK        := 0;
              SEQCRTLOC        := 4;
              READBLOCK (SEQ_FL_NO,BUF,0);
              SQENDLOC,SQENDBLK := BUF.STATUS;
```

```
            SEQ_END_BLK      := SQENDBLK.INT;
            SQENDLOC.ARR[Ø]  := BUF.DATA[2];
            SQENDLOC.ARR[1]  := BUF.DATA[3];
            SEQ_END_LOC      := SQENDLOC.INT;
          END;

          (* mise a jour de la table des fichiers *)

          FILE_TB[FREE_CH_NO].BUSY       := TRUE;
          FILE_TB[FREE_CH_NO].USER_COUNT := 1;

          (* mise a jour du bloc reponse *)

          BLOCK_LGTH         := 3;
          BLOCK[I_RET_CODE]  := Ø;
          BLOCK[I_ERR_CODE]  := Ø;
          BLOCK[I_RET_CHAN]    := FREE_CH_NO;
        END
    END
END; (*procedure openseq*)

(*************************************************************)

(*$I UNIT1B.TEXT *)
(*$I UNIT1C.TEXT *)
```

```
(**********************)
  PROCEDURE OPEN_INDX;
(**********************)

VAR

  DATA_FL_TLE   : FILE_TITLE;
  INDX_FL_TLE   : FILE_TITLE;
  FILE_FOUND    : BOOLEAN;
  CHAN_FOUND    : BOOLEAN;
  FREE_CH_NO    : INTEGER;
  EX_CH_NO      : INTEGER;
  DATA_FL_NO    : INTEGER;
  INDX_FL_NO    : INTEGER;
  I             : INTEGER;
  BUF           : T_BUFFER;

BEGIN

  (* initialisation *)

  (* Cette phase d'initialisation a pour effet de ranger
     respectivement dans les variables INDX_FILE et
     DATA_FILE le nom du fichier.INDX et celui du fichier
     .DATA*)

  INDX_FL_TLE := FILE_NAME.STR;
  DATA_FL_TLE := FILE_NAME.STR;
  DELETE (DATA_FL_TLE,POS('.',DATA_FL_TLE)+1,4);
  INSERT ('DATA',DATA_FL_TLE,POS('.',DATA_FL_TLE)+1);

  SEARCH_CHAN (INDX_FL_TLE,
               FILE_FOUND,CHAN_FOUND,
               FREE_CH_NO,EX_CH_NO);

  IF NOT CHAN_FOUND
  THEN
    ERROR (100)
  ELSE
    IF FILE_FOUND
    THEN
      BEGIN

        (* mise a jour de la table des canaux *)

        DATA_FL_NO := CHAN_TB[EX_CH_NO].FST_FL_NO;
        INDX_FL_NO := CHAN_TB[EX_CH_NO].SEC_FL_NO;

        CHAN_TB [FREE_CH_NO].BUSY       := TRUE;
        CHAN_TB [FREE_CH_NO].FILE_NAME  := INDX_FL_TLE;
        CHAN_TB [FREE_CH_NO].CORR       := CORR;
        CHAN_TB [FREE_CH_NO].FST_FL_NO  := DATA_FL_NO;
        CHAN_TB [FREE_CH_NO].SEC_FL_NO  := INDX_FL_NO;

        (* mise a jour de la table des fichiers *)

        FILE_TB[DATA_FL_NO].USER_COUNT :=
        FILE_TB[DATA_FL_NO].USER_COUNT + 1;
        FILE_TB[INDX_FL_NO].USER_COUNT :=
        FILE_TB[INDX_FL_NO].USER_COUNT + 1;
```

```
      BLOCK_LGTH         := 3;
      BLOCK[I_RET_CODE]  := Ø;
      BLOCK[I_ERR_CODE]  := Ø;
      BLOCK[I_RET_CHAN]  := FREE_CH_NO;
    END
  ELSE
    BEGIN

      SEARCH_FILE (FILE_FOUND,DATA_FL_NO);

      SEARCH_FILE (FILE_FOUND,INDX_FL_NO);

      RESET_FILE (DATA_FL_NO,DATA_FL_TLE);

      IF IORESULTS <> Ø
      THEN
        ERROR (1ØØ)
      ELSE
        BEGIN

          RESET_FILE (INDX_FL_NO,INDX_FL_TLE);

          IF IORESULTS <> Ø
          THEN
            ERROR (1ØØ)
          ELSE
            BEGIN

              (* mise a jour de la table des canaux *)

              CHAN_TB [FREE_CH_NO].BUSY        := TRUE;
              CHAN_TB [FREE_CH_NO].FILE_NAME   := INDX_FL_TLE;
              CHAN_TB [FREE_CH_NO].CORR        := CORR    ;
              CHAN_TB [FREE_CH_NO].FST_FL_NO   := DATA_FL_NO;
              CHAN_TB [FREE_CH_NO].SEC_FL_NO   := INDX_FL_NO;

              (* mise a jour de la table des fichiers *)

              FILE_TB[DATA_FL_NO].BUSY         := TRUE;
              FILE_TB[DATA_FL_NO].USER_COUNT   := 1;
              FILE_TB[INDX_FL_NO].BUSY         := TRUE;

              (* mise a jour du bloc reponse *)

              BLOCK [I_BLOCK_LOW] := 3;
              BLOCK [I_BLOCK_HIGH]:= Ø;
              BLOCK [I_RET_CODE]  := Ø;
              BLOCK [I_ERR_CODE]  := Ø;
              BLOCK [I_RET_CHAN]    := FREE_CH_NO;

              (* envoi du bloc reponse *)

              UNITWRITE (18,BLOCK[4],1,Ø,Ø);

              (* reception quittance du bloc reponse *)

              UNITREAD (18,BLOCK[4],1,Ø,Ø);

              (* mise a jour du bloc rec_desc *)

              READBLOCK (INDX_FL_NO,BUF,1);
```

```
                BLOCK_LGTH := 200;

              END
            END
        END
    END; (*procedure openindx*)

    (*******************************************************)

BEGIN (*procedure openfile*)

    K := 0; L := I_FL_NM_BEG;

    REPEAT
      BEGIN
        EOF_STR := (BLOCK[L]= BLANK)
                   OR (BLOCK[L]=13)
                   OR (BLOCK[L]=0);

        IF NOT EOF_STR
        THEN
          BEGIN
            FILE_NAME.ARR[K]:= BLOCK[L];
            K := K+1; L := L+1;
          END
        ELSE
          FILE_NAME.ARR[-1] := K;
      END
    UNTIL (EOF_STR) OR (K>23);

    IF EOF_STR
    THEN
      BEGIN
        SUFFIX := FILE_NAME.STR;
        IF COPY (SUFFIX,POS('.',SUFFIX)+1,4) = 'INDX'
        THEN
          OPEN_INDX
        ELSE
          OPEN_SEQ
        END;
        (* else erreur *)
END; (*procedure openfile*)

(*******************************************************)
```

```
(**********************)
  PROCEDURE CLOSE_FILE;
(**********************)

VAR

  SUFFIX : STRING [24];

  (************************************)
    PROCEDURE CLFILE (FILE_NO:INTEGER);
  (************************************)

    {-------------------------------------------------}
    {Cette procedure a pour effet de fermer le fichier }
    {identifie par FILE_NO                             }
    {-------------------------------------------------}

  BEGIN
    CASE FILE_NO OF
      Ø: CLOSE (FILEØ,LOCK);
      1: CLOSE (FILE1,LOCK);
      2: CLOSE (FILE2,LOCK);
      3: CLOSE (FILE3,LOCK);
      4: CLOSE (FILE4,LOCK);
      5: CLOSE (FILE5,LOCK);
      6: CLOSE (FILE6,LOCK);
      7: CLOSE (FILE7,LOCK);
      8: CLOSE (FILE8,LOCK);
      9: CLOSE (FILE9,LOCK);
      1Ø:CLOSE (FILE1Ø,LOCK);
      11:CLOSE (FILE11,LOCK);
      12:CLOSE (FILE12,LOCK);
      13:CLOSE (FILE13,LOCK);
      14:CLOSE (FILE14,LOCK);
      15:CLOSE (FILE15,LOCK);
    END
END: (*procedure clfile*)

(*************************************************)

(***********************************)
  PROCEDURE LIB_CHAN (CHAN_NO:INTEGER);
(***********************************)

  {-------------------------------------------------}
  {Cette procedure a pour effet de liberer dans la table}
  {des canaux l'entree numero CHAN_NO                }
  {-------------------------------------------------}

VAR

  K: INTEGER;

BEGIN (*procedure libchan*)

  CHAN_TB[CHAN_NO].BUSY      := FALSE;
  CHAN_TB[CHAN_NO].FILE_NAME := '                    ';
  CHAN_TB[CHAN_NO].CORR      := -1;

END: (*procedure libchan*)
```

```
  PROCEDURE CLOSE_SEQ;
(**********************)

  {-----------------------------------------------------}
  {Cette procedure assure le traitement approprie dans}
  {le cadre de la fermeture d'un fichier de type      }
  {sequentiel                                         }
  {-----------------------------------------------------}

VAR

  CHAN_NO      : INTEGER;
  BUF          : T_BUFFER;
  SEQ_END_LOC  : WORD;
  SEQ_END_BLK  : WORD;

BEGIN (*procedure closeseq*)

  (* mise a jour de la fin du fichier *)

  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,Ø);

  SEQ_END_BLK.INT := CHAN_TB[BLOCK[I_CHAN]].SEQ_END_BLK;
  BUF.DATA[Ø]     := SEQ_END_BLK.ARR[Ø];
  BUF.DATA[1]     := SEQ_END_BLK.ARR[1];
  SEQ_END_LOC.INT := CHAN_TB[BLOCK[I_CHAN]].SEQ_END_LOC;
  BUF.DATA[2]     := SEQ_END_LOC.ARR[Ø];
  BUF.DATA[3]     := SEQ_END_LOC.ARR[1];

  WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,Ø);


  CLFILE (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO);

  IF IORESULTS <> Ø
  THEN
    ERROR (1Ø4)
  ELSE
    BEGIN

      CHAN_NO:= BLOCK[I_CHAN];
      LIB_CHAN (CHAN_NO);

      (* mise a jour de la table des fichiers *)

      FILE_TB[CHAN_NO].BUSY       := FALSE;
      FILE_TB[CHAN_NO].USER_COUNT := Ø;

      (* mise a jour du bloc reponse *)

      BLOCK_LGTH         := 2;
      BLOCK[I_RET_CODE]  := Ø;
      BLOCK[I_ERR_CODE]  := Ø;

    END

END; (*procedure closeseq*)

(*********************************************************)

(**********************)
```

```
  {-------------------------------------------------}
  {Cette procedure assure le traitement approprie dans}
  {le cadre de la fermeture d'un fichier de type    }
  {indexe pour l'utilisateur                        }
  {-------------------------------------------------}

VAR

  DATA_FL_NO : INTEGER;
  INDX_FL_NO : INTEGER;
  CHAN_NO    : INTEGER;

BEGIN (*procedure closeindx*)

  (* initialisation *)

  CHAN_NO    := BLOCK[I_CHAN];
  DATA_FL_NO := CHAN_TB[CHAN_NO].FST_FL_NO;
  INDX_FL_NO := CHAN_TB[CHAN_NO].SEC_FL_NO;

  IF FILE_TB[INDX_FL_NO].USER_COUNT > 1
  THEN
    BEGIN

      LIB_CHAN (CHAN_NO);

      (* mise a jour de la table des fichiers *)

      FILE_TB[INDX_FL_NO].USER_COUNT :=
      FILE_TB[INDX_FL_NO].USER_COUNT - 1;

      (* mise a jour du bloc reponse *)

      BLOCK_LGTH       := 2;
      BLOCK[I_RET_CODE]:= Ø;
      BLOCK[I_ERR_CODE]:= Ø;

    END

  ELSE

    BEGIN

      CLFILE (DATA_FL_NO);

      IF IORESULTS <> Ø
      THEN
        ERROR (1Ø6)
      ELSE
        BEGIN
          CLFILE (INDX_FL_NO);
          IF IORESULTS <> Ø
          THEN
            ERROR (1Ø7)
          ELSE
            BEGIN
              LIB_CHAN (CHAN_NO);

              (* mise a jour de la table des fichiers *)

              FILE_TB[INDX_FL_NO].BUSY       := FALSE;
              FILE_TB[INDX_FL_NO].USER_COUNT := Ø;
```

```
                BLOCK_LGTH          := 2;
                BLOCK[I_RET_CODE]   := 0;
                BLOCK[I_ERR_CODE]   := 0;

              END
            END
          END
    END; (*procedure closeindx*)

    (*********************************************************)

  BEGIN (*procedure closefile*)
    SUFFIX := CHAN_TB[BLOCK[I_CHAN]].FILE_NAME;
    IF COPY (SUFFIX,POS('.',SUFFIX)+1,4) = 'INDX'
    THEN
      CLOSE_INDX
    ELSE
      CLOSE_SEQ;
  END; (*procedure closefile*)

  (*********************************************************)
END. (*unit openclose *)
```

```
(*$S+*)
UNIT UNIT2;

INTERFACE

USES
        (*$U UNIT3.CODE*)
        UNIT3,

        (*$U UNIT4.CODE*)
        UNIT4;

PROCEDURE READSEQ;
PROCEDURE WRITESEQ;


IMPLEMENTATION


(*********************)
  PROCEDURE READSEQ;
(*********************)

VAR

  DATALGTH : INTEGER;
  SUFFIX   : STRING[24];
  SEQFLNB  : INTEGER;
  SEQCRTBLK: INTEGER;
  K        : INTEGER;
  SEQCRTLOC: INTEGER;
  ENDFILE  : BOOLEAN;

  (*********************************************)
    PROCEDURE GETDATA (FILENB     : INTEGER;
                       DATALGTH   : INTEGER;
                       VAR CRTBLK : INTEGER;
                       VAR CRTLOC : INTEGER;
                       VAR K      : INTEGER;
                       VAR ENDFILE: BOOLEAN);
  (*********************************************)

    {*************************************************}
    {Cette procedure a pour effet d'aller lire dans le  }
    {fichier, designe par FILENB, DATALGTH bytes.       }
    {Au terme de son execution,                         }
    {   CRTBLK et CRTLOC specifient la position courante  }
    {     dans le fichier des donnees.                  }
    {   ENDFILE = TRUE si la fin du fichier a ete atteinte}
    {     par l'operation de lecture                    }
    {   ENDFILE = FALSE dans le cas contraire           }
    {*************************************************}

  VAR

    BUF:BUFFER;
    L  :INTEGER;

  BEGIN
    IF CHANTIERLOCKEDHANDLE <>
```

```
    ENDFILE:=(CRTBLK=SEQENDBLK) AND (CRTLOC=SEQENDLOC);
    IF NOT ENDFILE THEN
      READBLOCK (FILENB,BUF,CRTBLK);
    WHILE NOT (ENDFILE OR
                (K=DATALGTH)) DO
      BEGIN
        BLOCK[L] := BUF.DATA[CRTLOC];
        L:= L+1;
        ENDFILE:=(CRTBLK=SEQENDBLK) AND (CRTLOC=SEQENDLOC);
        IF NOT ENDFILE THEN K:=K+1;
        CRTLOC := CRTLOC + 1;
        IF CRTLOC > 511
        THEN
          BEGIN
            CRTBLK := CRTBLK+1;
            CRTLOC := Ø;
            IF NOT ((CRTBLK=SEQENDBLK) AND
                (CRTLOC=SEQENDLOC))
                THEN READBLOCK (FILENB,BUF,CRTBLK);
          END;
      END;
  END;
END (*procedure getdata*);

(*********************************************************)

BEGIN (*procedure readseq*)

  (* initialisation *)

  SEQFLNB := CHANTB[BLOCK[CHAN]].FSTFILE;
  SEQCRTBLK := CHANTB[BLOCK[CHAN]].SEQCRTBLK;
  SEQCRTLOC := CHANTB[BLOCK[CHAN]].SEQCRTLOC;

  GETDATA (SEQFLNB,DATALGTH,SEQCRTBLK,SEQCRTLOC,K,ENDFILE);

  IF ENDFILE
  THEN
    BEGIN
      BLOCKLGTH := K+4;
      ERROR (1ØØ)
    END
  ELSE
    IF IORESULTS <> Ø
    THEN
      ERROR (IORESULTS)
    ELSE
      BEGIN

        (* mise a jour du bloc reponse *)

        BLOCKLGTH := K + 4;
        BLOCK[CODE] := Ø;BLOCK[CODE+1] := Ø;
        BLOCK[CODE+3]:=K;

      END;

  (* mise a jour de la table des canaux *)

  CHANTB[BLOCK[CHAN]].SEQCRTBLK := SEQCRTBLK;
```

```
(*********************************************************)

(**********************)
  PROCEDURE WRITESEQ;
(**********************)

VAR

  SEQFLNB :INTEGER;
  DATALGTH,SEQENDBLK,K:INTEGER;
  SEQENDLOC:INTEGER;
  IORES:INTEGER;

(*****************************************)
  PROCEDURE PUTDATA (FILENB    : INTEGER;
                     DATALGTH  : INTEGER;
                     VAR ENDBLK: INTEGER;
                     VAR ENDLOC: INTEGER;
                     VAR K     : INTEGER);
(*****************************************)

  {*********************************************************}
  {Cette procedure a pour effet d'aller ecrire dans le      }
  {fichier identifie par FILENB , DATALGTH bytes.           }
  {Au terme de son execution,                               }
  {   ENDBLK et ENDLOC specifient la position courante du   }
  {   fichier des donnees.                                  }
  {   K comprend le nombre de bytes effectivement ecrits    }
  {   dans le fichier.                                      }
  {*********************************************************}

VAR

  L   :INTEGER;
  BUF:BUFFER;

BEGIN

  READBLOCK (FILENB,BUF,ENDBLK);
  L:= DATABEG; K:= 1;
    WHILE K<=DATALGTH DO
      BEGIN
      BUF.DATA[ENDLOC] := BLOCK[L];
      ENDLOC := ENDLOC + 1; L := L+1; K := K+1;
      IF ENDLOC > 511
      THEN
        BEGIN
          WRITEBLOCK (FILENB,BUF,ENDBLK);
          IORES:=IORESULT;
          IF IORES<>Ø THEN K:=32767;
          ENDBLK := ENDBLK + 1;
          ENDLOC := Ø;
        END;
      END;
  WRITEBLOCK (FILENB,BUF,ENDBLK);

END (*procedure putdata*);

(*********************************************************)
```

```
  SEQFLNB := CHANTB[BLOCK[CHAN]].FSTFILE;
  DATALGTH := BLOCK [DATALOW];
  SEQENDBLK := CHANTB[BLOCK[CHAN]].SEQENDBLK;
  SEQENDLOC := CHANTB[BLOCK[CHAN]].SEQENDLOC;

  PUTDATA (SEQFLNB,DATALGTH,SEQENDBLK,SEQENDLOC,K);

  IF K=32767 THEN
    ERROR (IORES)
  ELSE
    BEGIN

      (* mise a jour du bloc reponse *)

      BLOCKLGTH := 4;
      BLOCK [8] := Ø;
      BLOCK [9] := Ø;

    END;

  (* mise a jour de la table des canaux *)


  CHANTB[BLOCK[CHAN]].SEQENDBLK := SEQENDBLK;
  CHANTB[BLOCK[CHAN]].SEQENDLOC := SEQENDLOC;

END (*procedure writenext*);

(*************************************************************)

END. (*unit4*)
```

```
(*$S+*)
UNIT UNIT3;

INTERFACE

CONST

        APPENDF   =  6;
        OPENF     =  5;
        CLOSEF    =  7;
        WRSEQ     = 11;
        RDSEQ     =  9;
        WRINDX    = 40;
        RDINDX    = 41;
        RDNEXT    = 42;
        RDMOD     = 43;
        DLREC     = 44;
        MDREC     = 45;


        I_CORR          = 4;
        I_LOCAL         = 5;
        I_BLOCK_LOW     = 6;
        I_BLOCK_HIGH    = 7;
        I_CODE          = 8;
        I_FL_NM_BEG     = 9;
        I_FL_NM_END     = 32;
        I_TERM          = 33;
        I_RET_CODE      = 8;
        I_ERR_CODE      = 9;
        I_CHAN          = 9;
        I_RET_CHAN      = 10;
        I_DATA_ID       = 10;
        I_DATA_HIGH     = 11;
        I_DATA_LOW      = 12;
        I_KEY_NO        = 10;
        I_KEY_VAL       = 11;
        I_KEY_OCC_BEG   = 13;
        I_REC_BEG       = 12;

        CHAN_TB_LGTH    = 7;
        FILE_TB_LGTH    = 15;
        END_BLK_TRT     = 20;
        KEY_MAX_LGTH    = 20;
        KEY_MAX_NB      = 10;
        MAX_LOCK_REC    = 5;

TYPE
        BYTE            = 0..255;

        WORD            = PACKED RECORD
                            CASE BOOLEAN OF
                            TRUE : (INT:INTEGER);
                            FALSE: (ARR:PACKED ARRAY [0..1] OF BYTE);
                          END;
```

```
                    FALSE:(ARR:PACKED ARRAY [Ø..24] OF BYTE);
                  END;

FILE_ID      = FILE;

FILE_TITLE   = STRING[24];

FILE_DESC    = PACKED RECORD
                  BUSY      :BOOLEAN;
                  USER_COUNT:INTEGER;
                  END;

LOCK_REC     = PACKED RECORD
                  BUSY          : BOOLEAN;
                  LOCK_REC_BLK  : INTEGER;
                  LOCK_REC_LOC  : BYTE;
                  END;

CHAN_DESC    = PACKED RECORD
                  BUSY      : BOOLEAN;
                  FILE_NAME : FILE_TITLE;
                  CORR      : INTEGER;
                  FST_FL_NO : INTEGER;
                  SEC_FL_NO : INTEGER;
                  CASE BOOLEAN OF
                  TRUE : ( SEQ_CRT_BLK : INTEGER;
                           SEQ_CRT_LOC : INTEGER;
                           SEQ_END_BLK : INTEGER;
                           SEQ_END_LOC : INTEGER);
                  FALSE: ( INDX_CRT_BLK: INTEGER;
                           INDX_CRT_LOC: INTEGER;
                           CRT_KEY_VAL : PACKED ARRAY
                             [Ø..KEY_MAX_LGTH] OF BYTE;
                           CRT_KEY_OCC : INTEGER;
                           LOCK_REC_TB : ARRAY
                             [1..MAX_LOCK_REC] OF LOCK_REC);
                  END;

KEY_DESC     = PACKED RECORD
                  STEP    : INTEGER;
                  KEY_PT  : INTEGER;
                  KEY_LGTH: INTEGER;
                  END;

K_TYPE       = (PRM,SEC);

CB_BLOCK     = PACKED ARRAY [Ø..263] OF BYTE;

BUFFER_TYPE = (INDX,HEAD,DATA);

INDX_BUFFER  = PACKED RECORD
                  ARR       : PACKED ARRAY [Ø..5Ø7] OF BYTE;
                  PREV_BLK  :INTEGER;
                  NEXT_BLK  :INTEGER;
                  END;

HEAD_BUFFER = PACKED RECORD
                  INDX_CRT_BLK:INTEGER;
                  DATA_CRT_BLK:INTEGER;
                  DATA_CRT_LOC:WORD;
                  SEC_KEY_NB  :INTEGER;
```

```
    DATA_BUFFER = PACKED ARRAY [0..511] OF BYTE;

    T_BUFFER    = PACKED RECORD
                    CASE BUFFER_TYPE OF
                    INDX: (INDX:INDX_BUFFER);
                    HEAD: (HEAD:HEAD_BUFFER);
                    DATA: (DATA:DATA_BUFFER);
                  END;

    T_REF_KEY    = PACKED RECORD
                     CASE BOOLEAN OF
                     TRUE : (STR:STRING[KEY_MAX_LGTH]);
                     FALSE: (ARR:PACKED ARRAY [0..20] OF BYTE);
                   END;

    KEY_KIND    = (INT,ALN,ARR);

    KEY         = RECORD
                    CASE KEY_KIND OF
                    INT:(INT:INTEGER);
                    ALN:(ALN:STRING[100]);
                    ARR:(ARR:PACKED ARRAY [0..99] OF BYTE);
                  END;

    T_ACCESS    = (A_WRINDX,A_RDINDX,A_RDNEXT,A_RDMOD,
                   A_DLREC,A_MDREC);

VAR
    BLOCK       : CB_BLOCK;

    BLOCK_NO    : WORD;

    FILE_TB     : PACKED ARRAY [0..FILE_TB_LGTH] OF FILE_DESC;

    CHAN_TB     : PACKED ARRAY [0..CHAN_TB_LGTH] OF CHAN_DESC;

    FILE0 ,FILE1 ,FILE2 ,FILE3 : FILE ;
    FILE4 ,FILE5 ,FILE6 ,FILE7 : FILE ;
    FILE8 ,FILE9 ,FILE10,FILE11: FILE ;
    FILE12,FILE13,FILE14,FILE15: FILE ;

    BLOCK_LGTH  : INTEGER;

    FILENAME    : T_FILENAME;

    CORR        : INTEGER;

    HD_BUF,BUF  : T_BUFFER;

    PT_LGTH     : INTEGER;

    KEY_LGTH    : INTEGER;

    KEY_LOC     : INTEGER;

    IMAX,J      : INTEGER;

    REF_KEY     : T_REF_KEY;

    KEY_TYPE    : KTYPE;
```

```
        FTL_ERR      : BOOLEAN;
        DATA_BLK     : WORD;
        DATA_CRT_BLK: WORD;
        DATA_LOC     : BYTE;
        DATA_CRT_LOC: BYTE;
        CRT_POS_BLK : INTEGER;
        CRT_POS_REC : INTEGER;
        REC_LOCK : BOOLEAN;
        FREE_LOCK_REC : BOOLEAN;
        LOCK_ID       : INTEGER;
        REC_ID : INTEGER;

PROCEDURE ERROR (ERR_NO:INTEGER);
PROCEDURE READBLOCK (FILE_NO:INTEGER;VAR BUF:T_BUFFER;
                     REL_BLOCK_NO:INTEGER);
PROCEDURE WRITEBLOCK (FILE_NO:INTEGER;VAR BUF:T_BUFFER;
                      REL_BLOCK_NO:INTEGER);


IMPLEMENTATION

(************************************)
PROCEDURE ERROR (*ERR_NO:INTEGER*);
(************************************)

  {************************************************************}
  {Cette procedure a pour effet de garnir un bloc reponse      }
  {a renvoyer a l'utilisateur lorsque sa requete a echoue      }
  {pour une raison quelconque. ERR-NO contient le numero       }
  {significatif del'erreur qui s'est produite.                 }
  {************************************************************}

BEGIN

  BLOCK_LGTH      := 2;
  BLOCK[I_RET_CODE] := 1;
  BLOCK[I_ERR_CODE] := ERR_NO;


END; (*procedure error*)

(************************************************************)

(************************************************)
PROCEDURE READBLOCK (*FILE_NO       : INTEGER  ;
                     VAR BUF        : T_BUFFER ;
                     REL_BLOCK_NO : INTEGER*);
(************************************************)
```

```
  {fichier identifie par FILE_NO le bloc de numero relatif}
  {REL_BLOCK_NO. Le resultat est stocke dans BUF.          }
  {________________________________________________________}

VAR

  N:INTEGER;  {contient le nombre de blocs de lu par
               la fonction BLOCKREAD }

BEGIN

  CASE FILE_NO OF
     Ø:N:= BLOCKREAD (FILEØ ,BUF,1,REL_BLOCK_NO);
     1:N:= BLOCKREAD (FILE1, BUF,1,RELBLOCKNØ);
     2:N:= BLOCKREAD (FILE2, BUF,1,REL_BLOCK_NO);
     3:N:= BLOCKREAD (FILE3, BUF,1,REL_BLOCK_NO);
     4:N:= BLOCKREAD (FILE4, BUF,1,REL_BLOCK_NO);
     5:N:= BLOCKREAD (FILE5, BUF,1,REL_BLOCK_NO);
     6:N:= BLOCKREAD (FILE6, BUF,1,REL_BLOCK_NO);
     7:N:= BLOCKREAD (FILE7, BUF,1,REL_BLOCK_NO);
     8:N:= BLOCKREAD (FILE8, BUF,1,REL_BLOCK_NO);
     9:N:= BLOCKREAD (FILE9, BUF,1,REL_BLOCK_NO);
    1Ø:N:= BLOCKREAD (FILE1Ø,BUF,1,REL_BLOCK_NO);
    11:N:= BLOCKREAD (FILE11,BUF,1,REL_BLOCK_NO);
    12:N:= BLOCKREAD (FILE12,BUF,1,REL_BLOCK_NO);
    13:N:= BLOCKREAD (FILE13,BUF,1,REL_BLOCK_NO);
    14:N:= BLOCKREAD (FILE14,BUF,1,REL_BLOCK_NO);
    15:N:= BLOCKREAD (FILE15,BUF,1,REL_BLOCK_NO);
  END;

END; (*procedure readblock*)

(*****************************************************)

(************************************************)
PROCEDURE WRITEBLOCK (*FILE_NO    :INTEGER    ;
                       VAR BUF    : T_BUFFER  ;
                       REL_BLOCK_NO:INTEGER*);
(************************************************)

  {-------------------------------------------------------}
  {Cette procedure a pour effet d'aller ecrire ans le     }
  {fichier identifie par FILE_NO le bloc dont le numero   }
  {relatif vaut REL_BLOCK_NO. Les informations a ecrire   }
  {dans le fichier sont contenues dans BUF                }
  {-------------------------------------------------------}

VAR

  N:INTEGER;  ( contient le nombre de blocs effectivement
                ecrits par la fonction BLOCKWRITE)

BEGIN

  CASE FILE_NO OF
          Ø:N:= BLOCKWRITE ( FILEØ,BUF,1,REL_BLOCK_NO);
          1:N:= BLOCKWRITE ( FILE1,BUF,1,REL_BLOCK_NO);
          2:N:= BLOCKWRITE ( FILE2,BUF,1,REL_BLOCK_NO);
```

```
        6:N:= BLOCKWRITE ( FILE6,BUF,1,REL_BLOCK_NO);
        7:N:= BLOCKWRITE ( FILE7,BUF,1,REL_BLOCK_NO);
        8:N:= BLOCKWRITE ( FILE8,BUF,1,REL_BLOCK_NO);
        9:N:= BLOCKWRITE ( FILE9,BUF,1,REL_BLOCK_NO);
       10:N:= BLOCKWRITE (FILE10,BUF,1,REL_BLOCK_NO);
       11:N:= BLOCKWRITE (FILE11,BUF,1,REL_BLOCK_NO);
       12:N:= BLOCKWRITE (FILE12,BUF,1,REL_BLOCK_NO);
       13:N:= BLOCKWRITE (FILE13,BUF,1,REL_BLOCK_NO);
       14:N:= BLOCKWRITE (FILE14,BUF,1,REL_BLOCK_NO);
       15:N:= BLOCKWRITE (FILE15,BUF,1,REL_BLOCK_NO);
    END;

  END; (*procedure writeblock*)

(*********************************************************)
END. (*unit utilities*)
```

```
(*$S+*)
UNIT UNIT4;

INTERFACE

USES
        (*$U UNIT3.CODE*)
        UNIT3;


TYPE
     COMP = (LT,GT,EQ);

PROCEDURE BLOCK_INIT ( KEY_TYPE:KTYPE; BUF:T_BUFFER;
                       POS:INTEGER );
PROCEDURE SEARCH_INIT (KEY_TYPE:KTYPE);
PROCEDURE SEARCH_TRT (OP_CODE:T_ACCESS;STEP,POINTER:INTEGER;
                      VAR KEY:T_REF_KEY;VAR BLOCK_NO:WORD;
                      VAR INSERT_FLAG,FTL_ERR:BOOLEAN);

IMPLEMENTATION

(*********************************************)
  PROCEDURE BLOCK_INIT (* KEY_TYPE : KTYPE;
                          BUF      : BUFFER;
                          POS      : INTEGER *);
(*********************************************)

  (*******************************************************)
  {Cette procedure a pour effet d'aller initialiser un    }
  {bloc du fichier .INDX (BUF), a partir de la position   }
  {indiquee par POS.                                      }
  {L'initialisation differe quelque peu selon KEY_TYPE    }
  {                                                       }
  {Si KEY_TYPE = PRM                                      }
  {                                                       }
  {Si KEY_TYPE = SEC                                      }
  {                                                       }
  (*******************************************************)

VAR

  NEG_NUMB_1 : WORD;
  K,M        : INTEGER;

BEGIN


 NEG_NUMB_1.INT := -1;
 IF KEY_TYPE = PRM
 THEN
  FOR K:= POS TO IMAX DO
      BEGIN

        BUF.INDX.ARR[(3+KEY_LGTH)*K] := NEG_NUMB_1.ARR[0];
        BUF.INDX.ARR[(3+KEY_LGTH)*K+1] := NEG_NUMB_1.ARR[1];
        FOR M := 3 TO KEY_LGTH+1 DO
            BUF.INDX.ARR[(3+KEY_LGTH)*K+M] := 32;
      END
 ELSE
```

```
            BUF.INDX.ARR[(6+KEY_LGTH)*K] :=
            NEG_NUMB_1.ARR[0];
            BUF.INDX.ARR[(6+KEY_LGTH)*K+3] :=
            NEG_NUMB_1.ARR[0];
            BUF.INDX.ARR[(6+KEY_LGTH)*K+1] :=
            NEG_NUMB_1.ARR[1];
            BUF.INDX.ARR[(6+KEY_LGTH)*K+4] :=
            NEG_NUMB_1.ARR[1];
            FOR M := 6 TO KEY_LGTH+1 DO
                BUF.INDX.ARR[(6+KEY_LGTH)*K+M] := 32;
        END;

END; (* procedure blockinit *)

(*************************************************************)

(************************************************)
  PROCEDURE SEARCH_INIT (* KEY_TYPE:(PRM,SEC) *);
(************************************************)

  {************************************************************}
  {Cette procedure a pour effet d'initialiser certaines       }
  {variables utiles dans le cadre de l'execution de la        }
  {procedure SEARCH_TRT.                                      }
  {L'initialisation differe quelque peu selon la valeur de    }
  {KEY_TYPE.                                                  }
  {************************************************************}

BEGIN

IF KEY_TYPE = PRM
THEN
PT_LGTH := 3
ELSE
PT_LGTH := 6;
KEY_FOUND := FALSE; FTL_ERR := FALSE;
IMAX := 508 DIV (PT_LGTH+KEY_LGTH) - 1;
J:= (IMAX+1) DIV 2;

END; (*procedure searchinit*)

(*************************************************************)

(*****************************************************)
  PROCEDURE SEARCH_TRT (*OP_CODE          : TACCESS;
                         STEP             : INTEGER;
                         POINTER          : INTEGER;
                         VAR KEY          : REF_KEY;
                         VAR BLOCK_NO      : WORD;
                         VAR INSERT_FLAG  : BOOLEAN ;
                         FTL_ERR          : BOOLEAN*);
(*****************************************************)

  {************************************************************}
  {Cette procedure est en fait le noeud du programme.         }
  {Elle est de type recursif.                                 }
  {On lui fournit une cle de reference (KEY) en fonction de   }
  {laquelle s'effectue la recherche dans l'arbre.             }
  {Arrivee au dernier niveau de l'arbre, elle declenche le    }
  {traitement approprie sur le fichier des donnees (la        }
  {nature du traitement demande par l'utilisateur a transmis) }
```

```
    {Au cours de cette ultime etape, il y a lieu de tester    }
    {s'il faut effectuer un traitement suite a l'adjonction   }
    {ou la suppression d'une valeur de cle dans l'arbre       }
    {*********************************************************}

VAR

  BUF1        : T_BUFFER;
  I           : INTEGER;
  M           : INTEGER;
  L           : INTEGER;
  XOFF        : BOOLEAN;
  PT_BLK      : WORD;
  I_DATA_BLK  : WORD;
  PT_FOUND    : WORD;
  PT_LOC      : BYTE;
  I_DATA_LOC  : BYTE;


  (***********************************************)
    FUNCTION COMPARE( REF_KEY      : T_REF_KEY;
                      REF_KEY_LGTH : INTEGER;
                      BUF          : T_BUFFER;
                      POS          : INTEGER):COMP;
  (***********************************************)

    {*********************************************************}
    {Cette procedure a pour objet de comparer byte a byte     }
    {en commencant par la gauche et sur une longeur de        }
    {REF_KEY_LGTH les deux series de bytes definies de la     }
    {facon suivante:                                          }
    {                                                         }
    {  - REF_KEY de type T_REF_KEY (cfr.     )                }
    {                                                         }
    {  - la serie de bytes commencant a la position POS       }
    {    dans BUF de type BUFFER (cfr.    )                   }
    {                                                         }
    {                                                         }
    {*********************************************************}

  VAR

    J       : INTEGER;
    M       : INTEGER;
    REF_KEY_ELT    : INTEGER;
    BUF_ELT : INTEGER;
    RESULT : COMP;

  BEGIN

  M:= (PT_LGTH+KEY_LGTH)*POS+PT_LGTH; J:= Ø;
  REPEAT
   BEGIN
     REF_KEY_ELT:= REF_KEY.ARR[J]; BUF_ELT := BUF.INDX.ARR[M];
     IF REF_KEY_ELT < BUF_ELT
     THEN
       RESULT:= LT
     ELSE
       IF REF_KEY_ELT > BUF_ELT
       THEN
```

```
      M:= M+1; J:= J+1;
    END
  UNTIL (RESULT <> EQ) OR (J=REF_KEY_LGTH);
  COMPARE := RESULT;

  END; (*function compare*)

  (*************************************************************)

(************************)
  PROCEDURE INSERTION ;
(************************)

  {*************************************************************}
  {Cette procedure a pour objet de mettre a jour, a chaque    }
  {niveau de l'arbre la structure d'indexe suite a l'         }
  {l'adjonction d'une valeur de cle a ce niveau               }
  {*************************************************************}


VAR

  MAX : INTEGER;
  K   : INTEGER;
  M   : INTEGER;

  (************************)
    PROCEDURE INDX_UPD;
  (************************)

    {*************************************************************}
    {Cette procedure a pour objet de repartir le contenu d'un   }
    {bloc plein (resultant de l'ajout d'une valeur de cle dans) }
    {la structure d'indexe) sur deux blocs.                     }
    {Les choses se passent de la facon suivante :               }
    {                                                           }




  VAR

    LST_PT_BLK   : WORD;
    K            : INTEGER;
    M            : INTEGER;
    PREVIOUS     : INTEGER;
    BUF2         : T_BUFFER;
    BUF3         : T_BUFFER;

  BEGIN

          LST_PT_BLK.ARR[0] := BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*IMAX];
          LST_PT_BLK.ARR[1] := BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*IMAX+1];

          IF LST_PT_BLK.INT <> 0
```

```
      WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,
                  BUF1,POINTER);
      INSERT_FLAG := FALSE;
    END
  ELSE
    BEGIN
      HD_BUF.HEAD.INDX_CRT_BLK := HD_BUF.HEAD.INDX_CRT_BLK+1;
      IF STEP = 0
      THEN
        BEGIN
          PREVIOUS := BUF1.INDX.PREV_BLK;
          IF PREVIOUS > 0
          THEN
            BEGIN
              READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,
                         BUF3,PREVIOUS);
              BUF3.INDX.NEXT_BLK :=
              HD_BUF.HEAD.INDX_CRT_BLK;
              WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,
                          BUF3,PREVIOUS);
            END
        END;
      FOR K := 0 TO J-1 DO
          FOR M := 0 TO KEY_LGTH+2 DO
                BUF2.INDX.ARR[(PT_LGTH+KEY_LGTH)*K+M]:=
                BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*K+M];
      BLOCK_INIT (KEY_TYPE,BUF2,J);
      IF STEP = 0
      THEN
        BEGIN
          IF PREVIOUS > 0
          THEN
            BUF2.INDX.PREV_BLK := PREVIOUS
          ELSE
            BEGIN
              BUF2.INDX.PREV_BLK := -1;
            END;
          BUF2.INDX.NEXT_BLK := POINTER;
        END;
      WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,
                  BUF2,HD_BUF.HEAD.INDX_CRT_BLK);
      FOR K := J TO IMAX DO
          FOR M := 0 TO KEY_LGTH+2 DO
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(K-J)+M]:=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*K+M];
      BLOCK_INIT (KEY_TYPE,BUF1,J);
      IF STEP = 0
      THEN
        BUF1.INDX.PREV_BLK := HD_BUF.HEAD.INDX_CRT_BLK;
      WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,
                  BUF1,POINTER);
      FOR M := 0 TO KEY_LGTH-1 DO
          KEY.ARR[M] := BUF2.INDX.ARR[(PT_LGTH+KEY_LGTH)*
                        (J-1)+PT_LGTH+M];
      FOR M := KEY_LGTH TO 99 DO
          KEY.ARR[M] := 32;
      KEY.ARR[-1] := 100;
      BLOCK_NO.INT := HD_BUF.HEAD.INDX_CRT_BLK;
      INSERT_FLAG := TRUE;
    END
```

```
BEGIN (* procedure insertion *)
  IF KEY_TYPE = PRM THEN MAX := 2
                    ELSE MAX := 5;
  FOR K := IMAX DOWNTO I+1 DO
      FOR M:= Ø TO KEY_LGTH + MAX DO
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*K+M] :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(K-1)+M];
  IF STEP <> Ø
  THEN
    BEGIN
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I] :=
      BLOCK_NO.ARR[Ø];
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+1] :=
      BLOCK_NO.ARR[1];
      FOR M := Ø TO KEY_LGTH-1 DO
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+3+M] :=
      KEY.ARR[M];
    END
  ELSE
    BEGIN

      IF KEY_TYPE = PRM THEN
      BEGIN

      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I] :=
      DATA_BLK.ARR[Ø];
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+1] :=
      DATA_BLK.ARR[1];
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+2] :=
      DATA_LOC;

      END

      ELSE
      BEGIN
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I]:=
          PT_BLK.ARR[Ø];
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+3] :=
          PT_BLK.ARR[Ø];
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+2] :=
          PT_BLK.ARR[1];
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+4] :=
          PT_BLK.ARR[1];
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+2] :=
          PT_LOC;
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+5] :=
          PT_LOC;

        END;

      FOR M := Ø TO KEY_LGTH-1 DO
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+PT_LGTH+M] :=
          REF_KEY.ARR[M];
    END;
    INDX_UPD;
END; (*procedure insertion*)
```

```
(*************************************************)
  PROCEDURE WRITE2BYTES( BYTE1,BYTE2 : BYTE;
                         VAR REL_NO   : INTEGER;
                         VAR POS      : WORD);
(*************************************************)

  (*************************************************************)
  (* Cette procedure a pour effet d'ecrire les deux bytes      *)
  (* BYTE1 et BYTE2 dans le fichier.                           *)
  (* Au terme de son execution, REL_NO et POS specifieront     *)
  (* la valeur position courante dans le fichier.              *)
  (*************************************************************)

BEGIN


BUF.DATA[2*POS.INT] := BYTE1;
BUF.DATA[2*POS.INT+1] := BYTE2;
POS.INT := POS.INT+1;
IF POS.INT > 255
THEN
 BEGIN
   WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,
               BUF,REL_NO);
   REL_NO := REL_NO + 1;
   POS.INT := Ø;
 END;
END; (*procedure write2bytes*)

(*****************************************************************)

(***************************************)
  PROCEDURE WRITE_REC( VAR CRT_BLK:WORD;
                       VAR CRT_LOC:BYTE);
(***************************************)

VAR

  L : INTEGER;
  DATA_REL_NO : INTEGER;
  N : INTEGER;
  DATA_POS : WORD;

BEGIN

DATA_REL_NO := HD_BUF.HEAD.DATA_CRT_BLK;
DATA_POS.ARR[Ø] := HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø];
DATA_POS.ARR[1] := Ø;


READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,DATA_REL_NO);

(* validation de l'article a ecrire *)

WRITE2BYTES (Ø,Ø,DATA_REL_NO,DATA_POS);

(* ecriture de l'article proprement dit *)
```

```
 BEGIN
   WRITE2BYTES (BLOCK[L],BLOCK[L+1],DATA_REL_NO,DATA_POS);
   L := L+2;
 END;
CRT_BLK.INT := DATA_REL_NO;
CRT_LOC := DATA_POS.ARR[Ø];

WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,CRT_BLK.INT);
END; (*procedure writerec*)

(*****************************************************)
PROCEDURE READ_REC (BLOCK_NO:INTEGER;LOCATION:BYTE);
(*****************************************************)

VAR K,L,M:INTEGER;
BEGIN

READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,
        BLOCK_NO);
M := LOCATION; L:= 12; K := 1;

(* test de la validite de l'article *)

IF BUF.DATA[2*M] = 1
THEN
  ERROR (1ØØ)
ELSE
  BEGIN
    M := M+1;
    IF M > 255
    THEN
      BEGIN
        BLOCK_NO := BLOCK_NO + 1;
        READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,
                   BUF,BLOCK_NO);
        M := Ø;
      END;
    (* lecture de l'article proprement dit *)


    REPEAT
     BLOCK[L] := BUF.DATA[2*M];
       L := L+1;

     BLOCK[L] := BUF.DATA[2*M+1] ;
       L := L+1;

     M := M+1;
     IF M > 255
     THEN
       BEGIN
         BLOCK_NO := BLOCK_NO + 1;
         READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,
                    BUF,BLOCK_NO);
         M := Ø;
       END;
     K := K+1;
    UNTIL (K>BLOCK[I_DATA_LOW]) OR (IORESULTS <> Ø);
    BLOCK_LGTH := K+3;
 END;
```

```
(*********************)
  PROCEDURE PT_UPD;
(*********************)

  (*************************************************************)
  (* Cette procedure a pour objet d'assurer la gestion des     *)
  (* pointeurs impliquee par l'adjonction, dans le fichier     *)
  (* des donnees, d'un enregistrement dont la valeur de cle    *)
  (* non identifiante existe deja dans la structure d'index    *)
  (* Les choses se passent de la facon suivante:               *)
  (*************************************************************)

VAR

R           : INTEGER;
WR_PT_LOC   : INTEGER;
LST_DATA_LOC : INTEGER;
P           : WORD;
NEG_NUMB_1  : WORD;
WR_PT_BLK   : WORD;
LST_DATA_BLK : WORD;

BEGIN

(* adjonction d'un nouveau pointeur dans la chaine*)

NEG_NUMB_1.INT := -1;
WR_PT_BLK.INT := HD_BUF.HEAD.DATA_CRT_BLK;
WR_PT_LOC := HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø];


READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,
        HD_BUF.HEAD.DATA_CRT_BLK);

R:= HD_BUF.HEAD.DATA_CRT_BLK;
P.INT := HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø];


WRITE2BYTES (DATA_BLK.ARR[Ø],DATA_BLK.ARR[1],R,P);
WRITE2BYTES (DATA_LOC,NEG_NUMB_1.ARR[Ø],R,P);
WRITE2BYTES (NEG_NUMB_1.ARR[1],32,R,P);
HD_BUF.HEAD.DATA_CRT_BLK := R;
HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø] := P.ARR[Ø];;

WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,R);

(* maj de l'ancien pointeur*)

LST_DATA_BLK.ARR[Ø] :=
BUFI.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+3];
LST_DATA_BLK.ARR[1] :=
BUFI.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+4];
LST_DATA_LOC :=
BUFI.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+5]+1;


IF LST_DATA_LOC > 255
THEN
BEGIN
```

```
        BUF.LST_DATA_BLK.INT);
BUF.DATA[2*LST_DATA_LOC+1] := WR_PT_BLK.ARR[Ø];
LST_DATA_LOC := LST_DATA_LOC+1;


IF LST_DATA_LOC > 255
THEN
BEGIN
 WRITEBLOCK(CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,
            BUF.LST_DATA_BLK.INT);
 LST_DATA_BLK.INT := LST_DATA_BLK.INT+1;
 LST_DATA_LOC := Ø;


 READBLOCK(CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,
           BUF.LST_DATA_BLK.INT);
END;
BUF.DATA[2*LST_DATA_LOC] := WR_PT_BLK.ARR[1];
BUF.DATA[2*LST_DATA_LOC+1] := WR_PT_LOC;
WRITEBLOCK(CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,
        BUF.LST_DATA_BLK.INT);

(* maj de la 2eme partie du pointeur dans le
fichier d'index*)

BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+3] := WR_PT_BLK.ARR[Ø];
BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+4] := WR_PT_BLK.ARR[1];
BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+5] := WR_PT_LOC;


WRITEBLOCK(CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,BUF1,
        BLOCK_NO.INT);

END; (*procedure ptupdate*)

(*****************************************************************)

(******************)
PROCEDURE TST_LOCK;
(******************)

VAR

  K : INTEGER;

BEGIN

  REPEAT
    REC_LOCK :=
    (CHAN_TB[BLOCK[I_CHAN]].LOCK_REC_TB[REC_ID].LOCK_REC_BLK
     = DATA_BLK.INT)
    AND
    (CHAN_TB[BLOCK[I_CHAN]].LOCK_REC_TB[REC_ID].LOCK_REC_LOC
     = DATA_LOC);

    K := K+1;
  UNTIL (REC_LOCK) OR (K > MAX_LOCK_REC);

END;

(*****************************************)
```

```
VAR

  K : INTEGER;

BEGIN

  K := 1;
  REPEAT
      FREE_LOCK_REC :=
      CHAN_TB [BLOCK[I_CHAN]].LOCK_REC_TB[K].BUSY = FALSE;
      IF FREE_LOCK_REC
      THEN

        WITH CHAN_TB [BLOCK[I_CHAN]].LOCK_REC_TB[K] DO
        BEGIN
          BUSY := TRUE;
          LOCK_REC_BLK := DATA_BLK.INT;
          LOCK_REC_LOC := DATA_LOC;
        END

      ELSE

        K := K+1;

  UNTIL (FREE_LOCK_REC) OR (K > MAX_LOCK_REC);

  LOCK_ID := K;

END;

(***************)
PROCEDURE UNLOCK;
(***************)

BEGIN

  WITH CHAN_TB [BLOCK[I_CHAN]].LOCK_REC_TB[REC_ID] DO
  BEGIN
    BUSY := FALSE;
    LOCK_REC_BLK := Ø;
    LOCK_REC_LOC := Ø;
  END;

END;


(*********************)
  PROCEDURE DATA_TRT ;
(*********************)

  (*************************************************************)
  (* Cette procedure est celle invoquee par la procedure       *)
  (* recursive au plus bas niveau de l'arbre en vue de         *)
  (* declencher le traitement approprie sur lr fichier des     *)
  (* donnees.                                                  *)
  (*************************************************************)


  (**********************)
```

```
(*Cette procedure a pour objet de declencher le traite- *)
(*ment approprie dans le fichier des donnees consecuti- *)
(*vement a l'adjonction d'un enregistrement dans le fi- *)
(*chier suite a la requete de l'utilisateur.            *)
(*Il est a noter que lors de l'appel a la procedure     *)
(*SEARCH_TRT en vue de gerer la structure d'index       *)
(*associee la cle primaire, on effectue egalement       *)
(*l'ecriture proprement dite de l'article dans le       *)
(*fichier                                               *)
(*******************************************************)

(***********************)
  PROCEDURE PRM_KEY_TRT;
(***********************)

  (*******************************************************)
  (* Cette procedure a pour objet                        *)
  (*    - de gerer la structure d'index relative a la cle *)
  (*      primaire lors de l'adjonction d'un nouvel en-   *)
  (*      registrement dans le fichier.                   *)
  (*                                                      *)
  (*    - de realiser l'ecriture proprement dite de l'en- *)
  (*      registrement dans le fichier                    *)
  (*******************************************************)

BEGIN


  IF KEY_FOUND
  THEN
    BEGIN
      FTL_ERR := TRUE;
      ERROR (100);
    END
  ELSE
    BEGIN
      DATA_BLK.INT := HD_BUF.HEAD.DATA_CRT_BLK;
      DATA_LOC := HD_BUF.HEAD.DATA_CRT_LOC.ARR[0];

      WRITE_REC (DATA_CRT_BLK,DATA_CRT_LOC);

      IF IORESULTS <> 0
      THEN
        BEGIN
          INSERT_FLAG := FALSE;
          ERROR (IORESULTS);
          FTL_ERR := TRUE;
        END
      ELSE
        BEGIN
          HD_BUF.HEAD.DATA_CRT_BLK        := DATA_CRT_BLK.INT;
          HD_BUF.HEAD.DATA_CRT_LOC.ARR[0] := DATA_CRT_LOC;
        END
    END

END; (* procedure prmkeytrt *)

(*******************************************************)

(***********************)
```

```
    {*************************************************************}
    {*Cette procedure a pour objet de gerer la structure      *}
    {* d'index associee a une valeur de cle secondaire        *}
    {*************************************************************}

  VAR

    NEG_NUMB_1  : WORD;
    P           : WORD;
    R           : INTEGER;

BEGIN

  IF KEY_FOUND
  THEN
    BEGIN
      INSERT_FLAG := FALSE;
      PT_UPD ;
    END
  ELSE
    BEGIN

      NEG_NUMB_1.INT := -1;
      READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,DATA_CRT_BLK.INT);
      PT_BLK.INT := DATA_CRT_BLK.INT;
      R := DATA_CRT_BLK.INT;
      PT_LOC := DATA_CRT_LOC;
      P.ARR[Ø] := DATA_CRT_LOC; P.ARR[1] := Ø;
      WRITE2BYTES (DATA_BLK.ARR[Ø],DATA_BLK.ARR[1],R,P);
      WRITE2BYTES (DATA_LOC,NEG_NUMB_1.ARR[Ø],R,P);
      WRITE2BYTES (NEG_NUMB_1.ARR[1],32,R,P);

      HD_BUF.HEAD.DATA_CRT_BLK := R;
      HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø] := P.ARR[Ø];
      WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,R);
    END

  END; (* procedure  seckeytrt *)

  {*************************************************************}


BEGIN


IF KEY_TYPE = PRM
  THEN
    PRM_KEY_TRT
  ELSE
    SEC_KEY_TRT;

END; (* procedure  wrindrec *)

{**********************}
  PROCEDURE  RD_IND_REC;
{**********************}

  {*************************************************************}
  (* Cette procedure a pour objet d'assurer le traitement *)
  (* suivie sur le fichier des données concernant un      *)
```

```
(*************************************************************)

(**********************)
  PROCEDURE PRM_KEY_TRT;
(**********************)

  (*************************************************************)
  (* Cette procedure a pour objet d'initialiser les para- *)
  (* metres de la procedure READ_REC si l'operation de     *)
  (* consultation demandee se refere a la cle primaire     *)
  (*************************************************************)

BEGIN


  IF KEY_FOUND
  THEN
    BEGIN
          DATA_BLK.ARR[Ø] :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)];
          DATA_BLK.ARR[1] :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+1];
          DATA_LOC :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+2];

          TST_LOCK;
          IF REC_LOCK THEN ERROR (1ØØ);

          READ_REC (DATA_BLK.INT,DATA_LOC);
        END
    ELSE
      ERROR (1ØØ);

  END: (* procedure prmkeytrt *)

  (*************************************************************)

  (**********************)
    PROCEDURE SEC_KEY_TRT;
  (**********************)

  BEGIN

    IF KEY_FOUND
    THEN
      BEGIN
        I_DATA_BLK.ARR[Ø] :=
        BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)];
        I_DATA_BLK.ARR[1] :=
        BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+1];
        I_DATA_LOC :=
        BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+2];


        READBLOCK (CHAN_TB[BLOCK[I_CHAN]].
                   FST_FL_NO,BUF,
                   I_DATA_BLK.INT);

        DATA_BLK.ARR[Ø] :=
        BUF.DATA [2*I_DATA_LOC];
        DATA_BLK.ARR[1] :=
```

```
        IF I_DATA_LOC > 255
        THEN
          BEGIN
            READBLOCK (CHAN_TB[BLOCK[I_CHAN]].
                       FST_FL_NO,BUF.
                       I_DATA_BLK.INT+1);
            I_DATA_LOC := 0;
          END;
        DATA_LOC := BUF.DATA[2*I_DATA_LOC];

        TST_LOCK;
        IF REC_LOCK THEN ERROR (100);

        READ_REC (DATA_BLK.INT,DATA_LOC);
      END
    ELSE
      ERROR (100);

END; (* procedure seckeytrt *)

(*********************************************************)



BEGIN

  IF KEY_TYPE = PRM
  THEN
    PRM_KEY_TRT
  ELSE
    SEC_KEY_TRT;

END; (* procedure rdindrec *)
```

```
(**********************)
  PROCEDURE  RD_MOD_REC;
(**********************)

  (*********************************************************)
  (* Cette procedure a pour objet d'assurer le traitement *)
  (* requis sur le fichier des donnees consecutivement a  *)
  (* une operation de consultation avec blocage demandee  *)
  (* par l'utilisateur.                                   *)
  (*********************************************************)

  (**********************)
    PROCEDURE PRM_KEY_TRT;
  (**********************)

    (*********************************************************)
    (* Cette procedure a pour objet d'initialiser les para- *)
    (* metres de la procedure READ_REC si l'operation de     *)
    (* consultation demandee se refere a la cle primaire    *)
    (*********************************************************)

  BEGIN


    IF KEY_FOUND
    THEN
      BEGIN
              DATA_BLK.ARR[Ø] :=
              BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)];
              DATA_BLK.ARR[1] :=
              BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+1];
              DATA_LOC :=
              BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+2];

              TST_LOCK;
              IF REC_LOCK
              THEN
                ERROR (1ØØ)
              ELSE
                BEGIN
                  SET_LOCK (LOCK_ID);

                  READ_REC (DATA_BLK.INT,DATA_LOC);

                  BLOCK [I_DATA_ID] := LOCK_ID;

                END
            END
      ELSE
        ERROR (1ØØ);

  END; (* procedure prmkeytrt *)

  (*********************************************************)

  (**********************)
    PROCEDURE SEC_KEY_TRT;
  (**********************)
```

```
      IF KEY_FOUND
      THEN
        BEGIN
          I_DATA_BLK.ARR[Ø] :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)];
          I_DATA_BLK.ARR[1] :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+1];
          I_DATA_LOC :=
          BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+2];

          READBLOCK (CHAN_TB[BLOCK[I_CHAN]].
                     FST_FL_NO,BUF,
                     I_DATA_BLK.INT);

          DATA_BLK.ARR[Ø] :=
          BUF.DATA [2*I_DATA_LOC];
          DATA_BLK.ARR[1] :=
          BUF.DATA [2*I_DATA_LOC+1];

          I_DATA_LOC := I_DATA_LOC+1;
          IF I_DATA_LOC > 255
          THEN
            BEGIN
              READBLOCK (CHAN_TB[BLOCK[I_CHAN]].
                         FST_FL_NO,BUF,
                         I_DATA_BLK.INT+1);
              I_DATA_LOC := Ø;
            END;
          DATA_LOC := BUF.DATA[2*I_DATA_LOC];

          TST_LOCK;
          IF REC_LOCK
          THEN
            ERROR (1ØØ)
          ELSE
            BEGIN
              SET_LOCK (LOCK_ID);

              READ_REC (DATA_BLK.INT,DATA_LOC);

              BLOCK [I_DATA_ID] := LOCK_ID;

            END;

        END
      ELSE
        ERROR (1ØØ);
  END; (* procedure seckeytrt *)

  (*********************************************************)


  BEGIN

    IF KEY_TYPE = PRM
```

```
  END; (* procedure rd_mod_rec *)


(****************)
PROCEDURE DEL_REC;
(****************)

VAR

  DL_REC_BLK : INTEGER;
  DL_REC_LOC : WORD;

BEGIN

  IF KEY_FOUND
  THEN
    BEGIN
      DATA_BLK.ARR[Ø] :=
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)];
      DATA_BLK.ARR[1] :=
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+1];
      DATA_LOC :=
      BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*(I-1)+2];

      TST_LOCK;

      IF REC_LOCK
      THEN
        ERROR (1ØØ)
      ELSE
        BEGIN
          READBLOCK (CHAN_TB [BLOCK[I_CHAN]].FST_FL_NO,BUF,
                     DATA_BLK.INT);
          DL_REC_BLK := DATA_BLK.INT;
          DL_REC_LOC.ARR[Ø] := DATA_LOC;
          DL_REC_LOC.ARR[1] := Ø;

          WRITE2BYTES (1,1,DL_REC_BLK,DL_REC_LOC);

          WRITEBLOCK (CHAN_TB [BLOCK[I_CHAN]].FST_FL_NO,BUF,
                     DATA_BLK.INT);
        END;

      END;

    END;

  (**********************)
    PROCEDURE  MOD_REC;
  (**********************)

    (*********************************************************)
    (*Cette procedure a pour objet de declencher le traite- *)
    (*ment approprie dans le fichier des donnees consecuti- *)
    (*vement a la mise a jourd'un enregistrement dans le    *)
    (*fichier suite a la requete de l'utilisateur.         *)
    (*Il est a noter que lors de l'appel a la procedure    *)
```

```
(*********************************************************)

(**********************)
  PROCEDURE PRM_KEY_TRT;
(**********************)

  (*********************************************************)
  (* Cette procedure a pour objet                          *)
  (*    - de gerer la structure d'index relative a la cle  *)
  (*      primaire lors de l'adjonction d'un nouvel en-    *)
  (*      registrement dans le fichier.                    *)
  (*                                                       *)
  (*    - de realiser l'ecriture proprement dite de l'en-  *)
  (*      registrement dans le fichier                     *)
  (*********************************************************)

BEGIN


  IF KEY_FOUND
  THEN
    BEGIN
      FTL_ERR := TRUE;
      ERROR (1ØØ);
    END
  ELSE
    BEGIN
      DATA_BLK.INT := HD_BUF.HEAD.DATA_CRT_BLK;
      DATA_LOC := HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø];

      WRITE_REC (DATA_CRT_BLK,DATA_CRT_LOC);

      IF IORESULTS <> Ø
      THEN
        BEGIN
          INSERT_FLAG := FALSE;
          ERROR (IORESULTS);
          FTL_ERR := TRUE;
        END
      ELSE
        BEGIN
          UNLOCK;
          HD_BUF.HEAD.DATA_CRT_BLK :=
          DATA_CRT_BLK.INT;
          HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø] :=
          DATA_CRT_LOC;
        END
      END

END; (* procedure prmkeytrt *)

  (*********************************************************)

(**********************)
  PROCEDURE SEC_KEY_TRT;
(**********************)

  (*********************************************************)
  (*Cette procedure a pour objet de gerer la structure     *)
```

```
    NEG_NUMB_1  : WORD;
    P           : WORD;
    R           :INTEGER;

BEGIN

  IF KEY_FOUND
  THEN
    BEGIN
      INSERT_FLAG := FALSE;
      PT_UPD ;
    END
  ELSE
    BEGIN


      NEG_NUMB_1.INT := -1;
      READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,DATA_CRT_BLK.INT);
      PT_BLK.INT := DATA_CRT_BLK.INT;
      R := DATA_CRT_BLK.INT;
      PT_LOC := DATA_CRT_LOC;
      P.ARR[Ø] := DATA_CRT_LOC; P.ARR[1] := Ø;
      WRITE2BYTES (DATA_BLK.ARR[Ø],DATA_BLK.ARR[1],R,P);
      WRITE2BYTES (DATA_LOC,NEG_NUMB_1.ARR[Ø],R,P);
      WRITE2BYTES (NEG_NUMB_1.ARR[1],32,R,P);

      HD_BUF.HEAD.DATA_CRT_BLK := R;
      HD_BUF.HEAD.DATA_CRT_LOC.ARR[Ø] := P.ARR[Ø];
      WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,R);
    END

  END; (* procedure  seckeytrt *)

  (*********************************************************)


BEGIN

IF KEY_TYPE = PRM
  THEN
    PRM_KEY_TRT
  ELSE
    SEC_KEY_TRT;

END; (* procedure  mod_rec *)




(************************)
```

```
DATA_BLK1,DATA_BLK2,DATA_BLK3 : WORD;
DATA_LOC1,DATA_LOC2,DATA_LOC3 : BYTE;
CHAN_NO: INTEGER;


PROCEDURE GET_NXT_KEY; FORWARD;

(************************)
  PROCEDURE GET_RECORD;
(************************)

VAR

  N: INTEGER;
  REC_FOUND : BOOLEAN;

(************************)
  PROCEDURE CHAN_TB_UPD;
(************************)

VAR

  L : INTEGER;
  XOFF: BOOLEAN;


    BEGIN

      XOFF:= FALSE; L := KEY_LGTH-1;
      WHILE (L>0) AND (NOT XOFF) DO
        BEGIN
          WITH CHAN_TB[BLOCK[I_CHAN]] DO
            BEGIN
              IF CRT_KEY_VAL[L] < 255
              THEN
                BEGIN
                  CRT_KEY_VAL[L] := CRT_KEY_VAL[L]+1;
                  XOFF := TRUE;
                END
              ELSE
                L := L-1;
            END;
          END;

    END;


  BEGIN


    WITH BUF1.INDX DO
      BEGIN
        DATA_BLK1.ARR[0] := ARR[(PT_LGTH+KEY_LGTH)*(I-1)];
        DATA_BLK1.ARR[1] := ARR[(PT_LGTH+KEY_LGTH)*(I-1)+1];
        DATA_LOC1        := ARR[(PT_LGTH+KEY_LGTH)*(I-1)+2];
      END;
```

```
    END
  ELSE
    BEGIN

      N := Ø;
      READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,DATA_BLK.INT);

      REPEAT

        N := N+1;
        DATA_BLK2.ARR[Ø]:= BUF.DATA [2*DATA_LOC1];
        DATA_BLK2.ARR[1]:= BUF.DATA [2*DATA_LOC1+1];
        DATA_LOC1 := DATA_LOC1+1;
        IF DATA_LOC1 > 255
        THEN
          BEGIN
            DATA_BLK1.INT := DATA_BLK1.INT + 1;
            READBLOCK (CHAN_TB[BLOCK[I_CHAN]].
                       FST_FL_NO,BUF,
                       DATA_BLK1.INT);
            DATA_LOC1 := Ø;
          END;
        DATA_LOC2 := BUF.DATA[2*DATA_LOC1];
        DATA_BLK3.ARR[Ø] := BUF.DATA [2*DATA_LOC1];
        DATA_LOC1 := DATA_LOC1+1;
        IF DATA_LOC1 > 255
        THEN
          BEGIN
            DATA_BLK1.INT := DATA_BLK1.INT + 1;
            READBLOCK (CHAN_TB[BLOCK[I_CHAN]].
                       FST_FL_NO,BUF,
                       DATA_BLK1.INT);
            DATA_LOC1 := Ø;
          END;
        DATA_BLK3.ARR[1] := BUF.DATA [2*DATA_LOC1];
        DATA_LOC3 := BUF.DATA [2*DATA_LOC1+1];
        DATA_BLK1.INT := DATA_BLK3.INT;
        DATA_LOC1 := DATA_LOC3;
        REC_FOUND := (DATA_BLK3.INT <> -1);
      UNTIL (N > CHAN_TB[CHAN_NO].CRT_KEY_OCC) OR (NOT REC_FOUND);

      IF REC_FOUND
      THEN
        BEGIN
          CHAN_TB[CHAN_NO].CRT_KEY_OCC := CHAN_TB[CHAN_NO].CRT_KEY_OCC+1;
          READ_REC (DATA_BLK2.INT,DATA_LOC1);
        END
      ELSE
        BEGIN
          GET_NXT_KEY;
        END;

    END;


END; (* procedure  getrecord *)


(************************)
  PROCEDURE GET_NXT_KEY;
```

```
    PROCEDURE  CHAN_TB_UPD;
  (************************)

  VAR

    K,L : INTEGER;

    BEGIN

      WITH CHAN_TB[BLOCK[I_CHAN]] DO
        BEGIN
          K := (PT_LGTH+KEY_LGTH)*I+PT_LGTH;
          FOR L :=  Ø TO KEY_LGTH-1  DO
            BEGIN
              CRT_KEY_VAL [L] := BUF1.INDX.ARR [K];
              K := K+1;
            END;
          FOR L := KEY_LGTH + 1 TO 99 DO
            CRT_KEY_VAL[L] := 32;
          CRT_KEY_VAL [-1] := KEY_LGTH;
        END;

    END; (* procedure chantb_upd *)

  BEGIN (* procedure getnxtkey *)

    I := I+1; (* I pointe alors sur la valeur de cle
                 suivant celle que l'on a deja examine *)
    IF I > IMAX
    THEN
      (* suivre le pointeur p1 *)
      IF BUF1.INDX.NEXT_BLK = -1
      THEN
        ERROR (1ØØ) (* EOF *)
      ELSE
        BEGIN
          CHAN_TB_UPD;
          GET_RECORD;
        END
    ELSE
      BEGIN
        CHAN_TB_UPD;
        GET_RECORD;
      END;

  END; (* procedure getnxtkey *)




BEGIN (* procedure rdnxtrec *)


  IF NOT KEY_FOUND
  THEN
    BEGIN
      IF I <> Ø
```

```
            GET_NXT_KEY
          END
        ELSE
          GET_RECORD;

    END;


  BEGIN

    CASE OP_CODE OF

      A_WRINDX : WR_IND_REC;
      A_RDINDX : RD_IND_REC;
      A_RDNEXT : RD_NXT_REC;
      A_RDMOD  : RD_MOD_REC;
      A_DLREC  : DEL_REC;
      A_MDREC  : MOD_REC;
    END;

  END; (* procedure datatrt *)


BEGIN (* procedure searchtrt *)

  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,BUF1,POINTER);

  I := Ø;

  REPEAT
    BEGIN
      XOFF:= COMPARE(REF_KEY,KEY_LGTH,BUF1,I) = LT;
      I := I+1;
    END
  UNTIL (XOFF) OR (I>IMAX);

  I := I-1;

  PT_FOUND.ARR[Ø] := BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I];
  PT_FOUND.ARR[1] := BUF1.INDX.ARR[(PT_LGTH+KEY_LGTH)*I+1];


  IF PT_FOUND.INT < Ø
  THEN
    FTL_ERR := TRUE
  ELSE
    BEGIN

      IF (STEP <> Ø) AND (NOT FTL_ERR)
      THEN
        SEARCH_TRT (OP_CODE,STEP-1,PT_FOUND.INT,KEY,BLOCK_NO,
                    INSERT_FLAG,FTL_ERR)
      ELSE
        BEGIN

          BLOCK_NO.INT := POINTER;
          FOR L := Ø TO KEY_LGTH-1 DO
            KEY.ARR [L] := BUF1.INDX.ARR
```

```
        DATA_TRT;
      END;
  END;
  IF INSERT_FLAG THEN INSERTION;

END; (*procedure searchtrt*)

      END. (*UNIT*)
```

```
(*$S+*)
UNIT UNIT5;

INTERFACE

USES
        (*$U UNIT3.CODE*)
        UNIT3,

        (*$U UNIT4.CODE*)
        UNIT4;

PROCEDURE WRITE_INDX;
PROCEDURE READ_INDX;
PROCEDURE READ_NEXT;

IMPLEMENTATION


(**********************)
  PROCEDURE WRITE_INDX;
(**********************)

  (*********************************************************)
  (* Cette procedure a pour objet d'aller ecrire dans le   *)
  (* fichier l'article envoye par l'utilisateur dans son   *)
  (* bloc requete. Elle assure la mise a jour des structu-*)
  (* res d'indexe relatives a chacune des valeurs de cle   *)
  (* presentes dans l'enregistrement a ecrire              *)
  (*********************************************************)


VAR

  OP_CODE      : T_ACCESS;
  STEP         : INTEGER;
  FST_BLK      : INTEGER;
  INSERT_FLAG  : BOOLEAN;
  FTL_ERR      : BOOLEAN;
  KEY          : T_REF_KEY;
  KEY_LOC      : INTEGER;

  (***************************************)
    PROCEDURE NEW_INDX_LVL (PT_LGTH:INTEGER);
  (***************************************)

    (*****************************************************)
    (* Cette procedure a pour effet d'ajouter un niveau  *)
    (* superieur dans la structure d'index si, au terme  *)
    (* de l'execution de la procedure SEARCHTRT cela     *)
    (* s'avere necessaire                                *)
    (*****************************************************)

  VAR
        PRM_KEY_PT   : WORD;
        K            : INTEGER;
        REL_BLK_NO   : INTEGER;
        BUFF         : T_BUFFER;
```

```
    BUF2.INDX.ARR[Ø] := BLOCK_NO.ARR[Ø];
    BUF2.INDX.ARR[1] := BLOCK_NO.ARR[1];
    FOR K:= Ø TO KEY_LGTH-1 DO
        BUF2.INDX.ARR[PT_LGTH+K] := KEY.ARR[K];
    BLOCKINIT (KEY_TYPE,BUF2,2);
    PRM_KEY_PT.INT := HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_PT;
    BUF2.INDX.ARR[PT_LGTH+KEY_LGTH] := PRM_KEY_PT.ARR[Ø];
    BUF2.INDX.ARR[PT_LGTH+KEY_LGTH+1] := PRM_KEY_PT.ARR[1];
    FOR K := PT_LGTH TO KEY_LGTH-1 DO
        BUF2.INDX.ARR[PT_LGTH+KEY_LGTH+K] := 255;
    HD_BUF.HEAD.INDX_CRT_BLK := HD_BUF.HEAD.INDX_CRT_BLK+1;
    REL_BLK_NO := HD_BUF.HEAD.INDX_CRT_BLK;

    WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,BUF2,REL_BLK_NO);

    HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_PT := REL_BLK_NO;
    HD_BUF.HEAD.KEY_DESC_TB[Ø].STEP :=
    HD_BUF.HEAD.KEY_DESC_TB[Ø].STEP+1;

END; (* procedure newindxlvl *)

(*************************************************************)

(**********************)
  PROCEDURE SEC_KEY_UPD;
(**********************)

  (*************************************************************)
  (* Cette procedure assure la gestion des  structures     *)
  (* d'index associees aux  cles secondaires               *)
  (*************************************************************)

VAR

  K,M,N       : INTEGER;


  POS_KEY_OCC : INTEGER;


BEGIN

    KEY_TYPE    := SEC;
    POS_KEY_OCC := I_KEY_OCC_BEG;


    FOR M := 1 TO HD_BUF.HEAD.SEC_KEY_NB DO (pour chaque cle secondaire)
      BEGIN

        (info relative a la cle m)
        KEY_LGTH := HD_BUF.HEAD.KEY_DESC_TB[M].KEY_LGTH;

        FOR N := 1 TO BLOCK[POS_KEY_OCC] DO (pour chaque occ de la cle m)

        BEGIN

        (* initialisation param.entree de SEARCH_TRT *)

        OP_CODE     := A_WRINDX;
```

```
          INSERT_FLAG := TRUE;
          FTL_ERR     := FALSE;
          (* initialisation info relative a l'occ n de la cle m*)

          FOR K := Ø TO KEY_LGTH - 1 DO
          BEGIN
            REF_KEY.ARR [K] := BLOCK [KEY_LOC];
            KEY_LOC         := KEY_LOC + 1;
          END;

          SEARCHINIT (KEY_TYPE);
          SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);
          IF INSERT_FLAG THEN NEW_INDX_LVL (6);
        END;
        (* on passe a la cle secondaire suivante *)
        POS_KEY_OCC := POS_KEY_OCC+1;
    END;
END;
(*************************************************************)

(**********************)
  PROCEDURE PRM_KEY_UPD;
(**********************)

  (*************************************************************)
  (* Cette procedure assure la gestion de la structure      *)
  (* d'index associee la cle primaire                       *)
  (* Elle realise en outre l'ecriture de l'enregistrement   *)
  (* dans le fichier                                        *)
  (*************************************************************)

VAR
  K           : INTEGER;  (indice)

BEGIN
  (* initialisation param. entree de SEARCH_TRT *)
  OP_CODE     := A_WRINDX;
  STEP        := HD_BUF.HEAD.KEY_DESC_TB[Ø].STEP;
  FST_BLK     := HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_PT;
  INSERT_FLAG := TRUE;
  FTL_ERR     := FALSE;
```

```
    KEY_TYPE := PRM;

    KEY_LGTH := HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_LGTH;

    FOR K := Ø TO KEY_LGTH - 1 DO
    BEGIN
      REF_KEY.ARR [K] := BLOCK [KEY_LOC];
      KEY_LOC         := KEY_LOC + 1;
    END;

    SEARCHINIT (KEY_TYPE);

    SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);

    IF INSERT_FLAG THEN NEW_INDX_LVL (3);

END;

(*************************************************************)

BEGIN (* procedure writeindx *)

  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);

  KEY_LOC := I_KEY_OCC_BEG + HD_BUF.HEAD.SEC_KEY_NB;

  PRM_KEY_UPD;

  IF NOT FTL_ERR
  THEN
    SEC_KEY_UPD;

  WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);

  BLOCK[I_RET_CODE] := Ø;
  BLOCK[I_ERR_CODE] := Ø;

END; (*procedure writeindx*)

(*************************************************************)

(**********************)
  PROCEDURE READ_INDX;
(**********************)

  (*********************************************************)
  (* Cette procedure assure le traitement consecutif a une*)
  (* demande de consultation sur base d'une cle d'acces   *)
  (*********************************************************)

VAR

  I,K,L        : INTEGER;

  OP_CODE      : T_ACCESS;
  STEP         : INTEGER;
  FST_BLK      : INTEGER;
  KEY          : T_REF_KEY;
  KEY_LOC      : INTEGER;
  INSERT_FLAG  : BOOLEAN;
```

```
  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);

  OP_CODE      := A_RDINDX;

  STEP         := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].STEP;

  FST_BLK    := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_PT;

  IF BLOCK [I_KEY_NO] = Ø
  THEN
    KEY_TYPE   := PRM
  ELSE
    KEY_TYPE   := SEC;


  FTL_ERR      := FALSE;

  INSERT_FLAG := FALSE;

  (* initialisation des info relative a la cle de reference *)

  KEY_LGTH     := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_LGTH;

  KEY_LOC      := I_KEY_VAL;

  FOR K := Ø TO KEY_LGTH - 1 DO
  BEGIN
    REF_KEY.ARR [K] := BLOCK [KEY_LOC];
    KEY_LOC         := KEY_LOC + 1;
  END;


  SEARCHINIT (KEY_TYPE);
  SEARCHTRT (OP_CODE ,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);

  BLOCK [I_RET_CODE] := Ø;
  BLOCK [I_ERR_CODE] := Ø;

END; (* procedure readindx *)

(******************)
PROCEDURE READ_NEXT;
(******************)

VAR

  K: INTEGER;  ( indice)

  OP_CODE      : T_ACCESS;
  STEP         : INTEGER;
  FST_BLK      : INTEGER;
  KEY          : T_REF_KEY;
  INSERT_FLAG : BOOLEAN;
  FTL_ERR      : BOOLEAN;

BEGIN

      READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);
```

```
      FST_BLK    := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_PT;

      IF BLOCK [I_KEY_NO] = Ø
      THEN
        KEY_TYPE  := PRM
      ELSE
        KEY_TYPE  := SEC;

      KEY_LGTH    := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_LGTH;

      FTL_ERR     := FALSE;

      INSERT_FLAG := FALSE;

      (* initialisation de la cle de reference *)

      FOR K := Ø TO KEY_LGTH-1 DO
          REFKEY.ARR [K] := CHAN_TB[BLOCK[I_CHAN]].CRTKEYVAL[K];

      SEARCHINIT (KEY_TYPE);
      SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);

      BLOCK [I_RET_CODE] := Ø;
      BLOCK [I_ERR_CODE] := Ø;

    END; (* procedure readnext *)

END. (* unit5 *)
```

```
(*$S+*)
UNIT UNIT6;

INTERFACE

USES
        (*$U UNIT3.CODE*)
        UNIT3,

        (*$U UNIT4.CODE*)
        UNIT4;

PROCEDURE DELETE;

PROCEDURE READ_MOD;

PROCEDURE MODIFY;


IMPLEMENTATION


(**********************)
  PROCEDURE DELETE :
(**********************)

  (*************************************************************)
  (* Cette procedure a pour objet d'aller detruire (marquer)*)
  (* dans le fichier .DATA l'article designe univoquement   *)
  (* par l'utilisateur au moyen de la cle primaire          *)
  (*************************************************************)

VAR

  I,K,L        : INTEGER;

  OP_CODE      : T_ACCESS;
  STEP         : INTEGER;
  FST_BLK      : INTEGER;
  KEY          : T_REF_KEY;
  KEY_LOC      : INTEGER;
  INSERT_FLAG  : BOOLEAN;
  FTL_ERR      : BOOLEAN;

BEGIN

  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);

  OP_CODE      := A_DLREC;

  STEP         := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].STEP;

  FST_BLK    := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_PT;

  KEY_TYPE     := PRM;

  KEY_LGTH     := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_LGTH;

  FTL_ERR      := FALSE;
```

```
    (* initialisation des info relative a la cle de reference *)

    KEY_LOC      := I_KEY_VAL;

    FOR K        := Ø TO KEY_LGTH - 1 DO
    BEGIN
      REF_KEY.ARR [K] := BLOCK [KEY_LOC];
      KEY_LOC         := KEY_LOC + 1;
    END;

    SEARCHINIT (KEY_TYPE);
    SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);

    BLOCK [I_RET_CODE] := Ø;
    BLOCK [I_ERR_CODE] := Ø;

  END; (* procedure delete *)




  (**********************)
    PROCEDURE READ_MOD ;
  (**********************)

    (*************************************************************)
    (* Cette procedure assure le traitement consecutif a une*)
    (* demande de consultation sur base d'une cle d'acces   *)
    (* Elle est censee etre appelee en vue d'effectuer une  *)
    (* modification eventuelle de l'article, a cet effet    *)
    (* on bloque l'article en question jusqu'a ce que l'uti-*)
    (* lisateur en demande une reecriture                   *)
    (*************************************************************)

  VAR

    I,K,L        : INTEGER;

    OP_CODE      : T_ACCESS;
    STEP         : INTEGER;
    FST_BLK      : INTEGER;
    KEY          : T_REF_KEY;
    KEY_LOC      : INTEGER;
    INSERT_FLAG  : BOOLEAN;
    FTL_ERR      : BOOLEAN;

  BEGIN

    READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);

    OP_CODE      := A_RDMOD;

    STEP         := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].STEP;

    FST_BLK    := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_PT;
```

```
    ELSE
      KEY_TYPE   := SEC;

    KEY_LGTH     := HD_BUF.HEAD.KEY_DESC_TB[BLOCK[I_KEY_NO]].KEY_LGTH;

    FTL_ERR      := FALSE;

    INSERT_FLAG := FALSE;

    (* initialisation des info relative a la cle de reference *)

    KEY_LOC := I_KEY_VAL;

    FOR K := Ø TO KEY_LGTH - 1 DO
    BEGIN
      REF_KEY.ARR [K] := BLOCK [KEY_LOC];
      KEY_LOC         := KEY_LOC + 1;
    END;


    SEARCHINIT (KEY_TYPE);
    SEARCHTRT (OP_CODE ,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);

    BLOCK [I_RET_CODE] := Ø;
    BLOCK [I_ERR_CODE] := Ø;

  END; (* procedure read_mod *)


  (**********************)
    PROCEDURE MODIFY    ;
  (**********************)

    (*************************************************************)
    (* Cette procedure a pour objet d'aller modifier dans        *)
    (* le fichier l'article designe par l'utilisateur au         *)
    (* moyen d'un identifiant obtenu lors d'un appel a la        *)
    (* procedure READ_MOD.                                       *)
    (*************************************************************)


  VAR

    OP_CODE      : T_ACCESS;
    STEP         : INTEGER;
    FST_BLK      : INTEGER;
    INSERT_FLAG  : BOOLEAN;
    FTL_ERR      : BOOLEAN;
    KEY          : T_REF_KEY;
    KEY_LOC      : INTEGER;
    DL_REC_BLK   : INTEGER;
    DL_REC_LOC   : WORD;

    (***************************************)
      PROCEDURE NEW_INDX_LVL (PTLGTH:INTEGER);
    (***************************************)

      (*****************************************************)
      (* Cette procedure a pour effet d'ajouter un niveau  *)
      (* supplementaire dans la structure d'index et, au terme  *)
```

```
VAR
      PRM_KEY_PT    : WORD;
      K             : INTEGER;
      REL_BLK_NO    : INTEGER;
      BUF2          : T_BUFFER;

BEGIN


  BUF2.INDX.ARR[Ø] := BLOCK_NO.ARR[Ø];
  BUF2.INDX.ARR[1] := BLOCK_NO.ARR[1];
  FOR K:= Ø TO KEY_LGTH-1 DO
      BUF2.INDX.ARR[PT_LGTH+K] := KEY.ARR[K];
  BLOCKINIT (KEY_TYPE,BUF2,2);
  PRM_KEY_PT.INT := HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_PT;
  BUF2.INDX.ARR[PT_LGTH+KEY_LGTH] := PRM_KEY_PT.ARR[Ø];
  BUF2.INDX.ARR[PT_LGTH+KEY_LGTH+1] := PRM_KEY_PT.ARR[1];
  FOR K := PT_LGTH TO KEY_LGTH-1 DO
      BUF2.INDX.ARR[PT_LGTH+KEY_LGTH+K] := 255;
  HD_BUF.HEAD.INDX_CRT_BLK := HD_BUF.HEAD.INDX_CRT_BLK+1;
  REL_BLK_NO := HD_BUF.HEAD.INDX_CRT_BLK;

  WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,BUF2,REL_BLK_NO);

  HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_PT := REL_BLK_NO;
  HD_BUF.HEAD.KEY_DESC_TB[Ø].STEP :=
  HD_BUF.HEAD.KEY_DESC_TB[Ø].STEP+1;

END; (* procedure newindxlvl *)

(*****************************************************************)

(**********************)
  PROCEDURE SEC_KEY_UPD;
(**********************)

  (*********************************************************)
  (* Cette procedure assure la gestion des  structures     *)
  (* d'index associees aux  cles secondaires               *)
  (*********************************************************)

VAR

  K,M,N        : INTEGER;

  POS_KEY_OCC : INTEGER;


BEGIN

    KEY_TYPE     := SEC;
    POS_KEY_OCC := I_KEY_OCC_BEG;


    FOR M := 1 TO HD_BUF.HEAD.SEC_KEY_NB DO (pour chaque cle secondaire)
      BEGIN

        (info relative a la cle m)
        KEY_LGTH := HD_BUF.HEAD.KEY_DESC_TB[M].KEY_LGTH;
```

```
        (* initialisation param.entree de SEARCH_TRT *)
        OP_CODE      := A_WRINDX;
        STEP         := HD_BUF.HEAD.KEY_DESC_TB[M].STEP;
        FST_BLK   := HD_BUF.HEAD.KEY_DESC_TB[M].KEY_PT;
        INSERT_FLAG := TRUE;
        FTL_ERR      := FALSE;
        (* initialisation info relative a l'occ n de la cle m*)

        FOR K := 0 TO KEY_LGTH - 1 DO
        BEGIN
          REF_KEY.ARR [K] := BLOCK [KEY_LOC];
          KEY_LOC         := KEY_LOC + 1;
        END;

        SEARCHINIT (KEY_TYPE);
        SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);
        IF INSERT_FLAG THEN NEW_INDX_LVL (6);
      END;
      (* on passe a la cle secondaire suivante *)
      POS_KEY_OCC := POS_KEY_OCC+1;
  END;
END;
(*************************************************************)

(**********************)
  PROCEDURE PRM_KEY_UPD;
(**********************)

  (*************************************************************)
  (* Cette procedure assure la gestion de la structure     *)
  (* d'index associee la cle primaire                      *)
  (* Elle realise en outre l'ecriture de l'enregistrement  *)
  (* dans le fichier                                       *)
  (*************************************************************)

VAR
  K          : INTEGER;
BEGIN

  (* initialisation param. entree de SEARCH_TRT *)
```

```
        (* initialisation param.entree de SEARCH_TRT *)
        OP_CODE      := A_WRINDX;
        STEP         := HD_BUF.HEAD.KEY_DESC_TB[M].STEP;
        FST_BLK    := HD_BUF.HEAD.KEY_DESC_TB[M].KEY_PT;
        INSERT_FLAG := TRUE;
        FTL_ERR      := FALSE;
        (* initialisation info relative a l'occ n de la cle m*)

        FOR K := 0 TO KEY_LGTH - 1 DO
        BEGIN
          REF_KEY.ARR [K] := BLOCK [KEY_LOC];
          KEY_LOC         := KEY_LOC + 1;
        END;

        SEARCHINIT (KEY_TYPE);
        SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);
        IF INSERT_FLAG THEN NEW_INDX_LVL (6);
      END;
      (* on passe a la cle secondaire suivante *)
      POS_KEY_OCC := POS_KEY_OCC+1;
  END;
END;
(*************************************************************)

(***********************)
  PROCEDURE PRM_KEY_UPD;
(***********************)

  (*************************************************************)
  (* Cette procedure assure la gestion de la structure      *)
  (* d'index associee la cle primaire                       *)
  (* Elle realise en outre l'ecriture de l'enregistrement   *)
  (* dans le fichier                                        *)
  (*************************************************************)

VAR
  K           : INTEGER;
BEGIN

  (* initialisation param. entree de SEARCH_TRT *)
```

```
    STEP        := HD_BUF.HEAD.KEY_DESC_TB[Ø].STEP;
    FST_BLK     := HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_PT;
    INSERT_FLAG := TRUE;
    FTL_ERR     := FALSE;
    (* initialisation des info relatives a la cle de reference *)
    KEY_TYPE := PRM;
    KEY_LGTH := HD_BUF.HEAD.KEY_DESC_TB[Ø].KEY_LGTH;
    FOR K := Ø TO KEY_LGTH - 1 DO
    BEGIN
      REF_KEY.ARR [K] := BLOCK [KEY_LOC];
      KEY_LOC         := KEY_LOC + 1;
    END;
    SEARCHINIT (KEY_TYPE);
    SEARCHTRT (OP_CODE,STEP,FST_BLK,KEY,BLOCK_NO,INSERT_FLAG,FTL_ERR);
    IF INSERT_FLAG THEN NEW_INDX_LVL (3);

END;
(*************************************************************)
BEGIN (* procedure modify *)
  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].SEC_FL_NO,HD_BUF,Ø);
  (* marquage de l'article a detruire *)
  REC_ID  := BLOCK [I_DATA_ID];
  DL_REC_BLK     := CHAN_TB [BLOCK[I_CHAN]].LOCK_REC_TB
                    [REC_ID].LOCK_REC_BLK;
  DL_REC_LOC.INT := CHAN_TB [BLOCK[I_CHAN]].LOCK_REC_TB
                    [REC_ID].LOCK_REC_LOC;
  READBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,DL_REC_BLK);

  BUF.DATA [2*DL_REC_LOC.INT]     := 1;
  BUF.DATA [2*DL_REC_LOC.INT + 1] := 1;
  WRITEBLOCK (CHAN_TB[BLOCK[I_CHAN]].FST_FL_NO,BUF,DL_REC_BLK);

  KEY_LOC := I_KEY_OCC_BEG + HD_BUF.HEAD.SEC_KEY_NB;
  PRM_KEY_UPD;
  IF NOT FTL_ERR
  THEN
    SEC_KEY_UPD;
```

```
END; (*procedure modify*)
(*************************************************************)
END. (* UNIT *)
```

Routines de communication

```
PROCEDURE READ_MESS (VAR MESS:MESSBUFFER; MESSLGTH:INTEGER;
                     CORRESP,LOCAL:INTEGER);

CONST
        CORR     = 4;
        LOC      = 5;
        MESSBEG  = 8;

VAR
        I        : INTEGER;


  BEGIN

    BLOCK [CORR] := CORRESP;
    BLOCK [LOC] := LOCAL;

    UNITREAD (18,BLOCK[4],1,Ø,Ø);

    FOR I := MESSBEG TO MESSBEG+MESSLGTH-1 DO
      MESS[I] := BLOCK [I];

  END;
```

```
PROCEDURE SENDMESS (MESS:MESSBUFFER; MESSLGTH:INTEGER;
                    CORRESP,LOCAL:INTEGER);

CONST
        CORR     = 4;
        LOC      = 5;
        BLKLOW   = 6;
        BLKHIGH  = 7;
        MESSBEG  = 8;

VAR
        I        : INTEGER;

  BEGIN

    BLOCK [CORR] := CORRESP;
    BLOCK [LOC] := LOCAL;
    BLOCK [BLKLOW] := (MESSBEG+MESSLGTH-1) MOD 256;
    BLOCK [BLKHIGH] :=(MESSBEG+MESSLGTH-1) DIV 256;

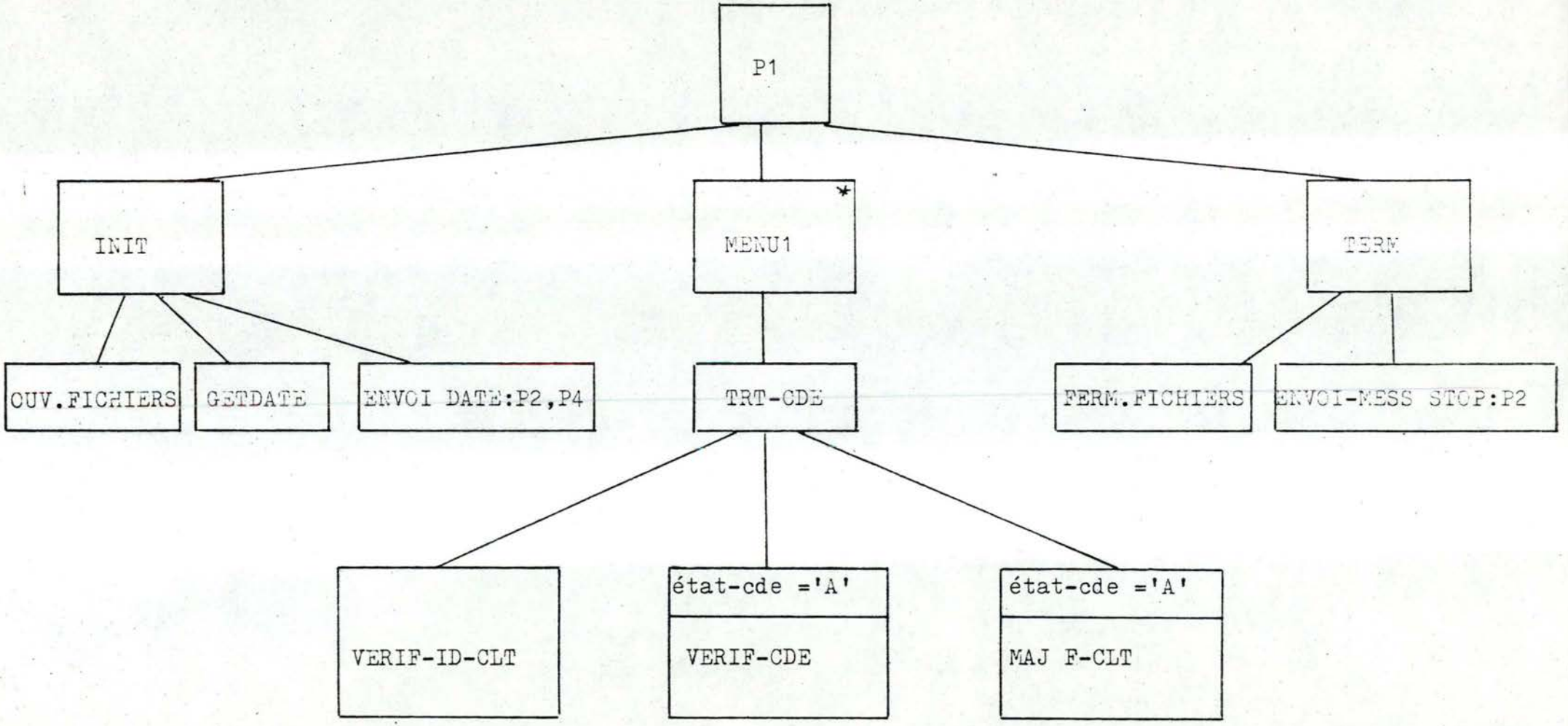
    FOR I := MESSBEG TO MESSBEG+MESSLGTH-1 DO
      BEGIN
        BLOCK[I] := MESS[I];
        WRITELN ('BLOCK[',I,']: ',BLOCK[I]); READLN;
      END;

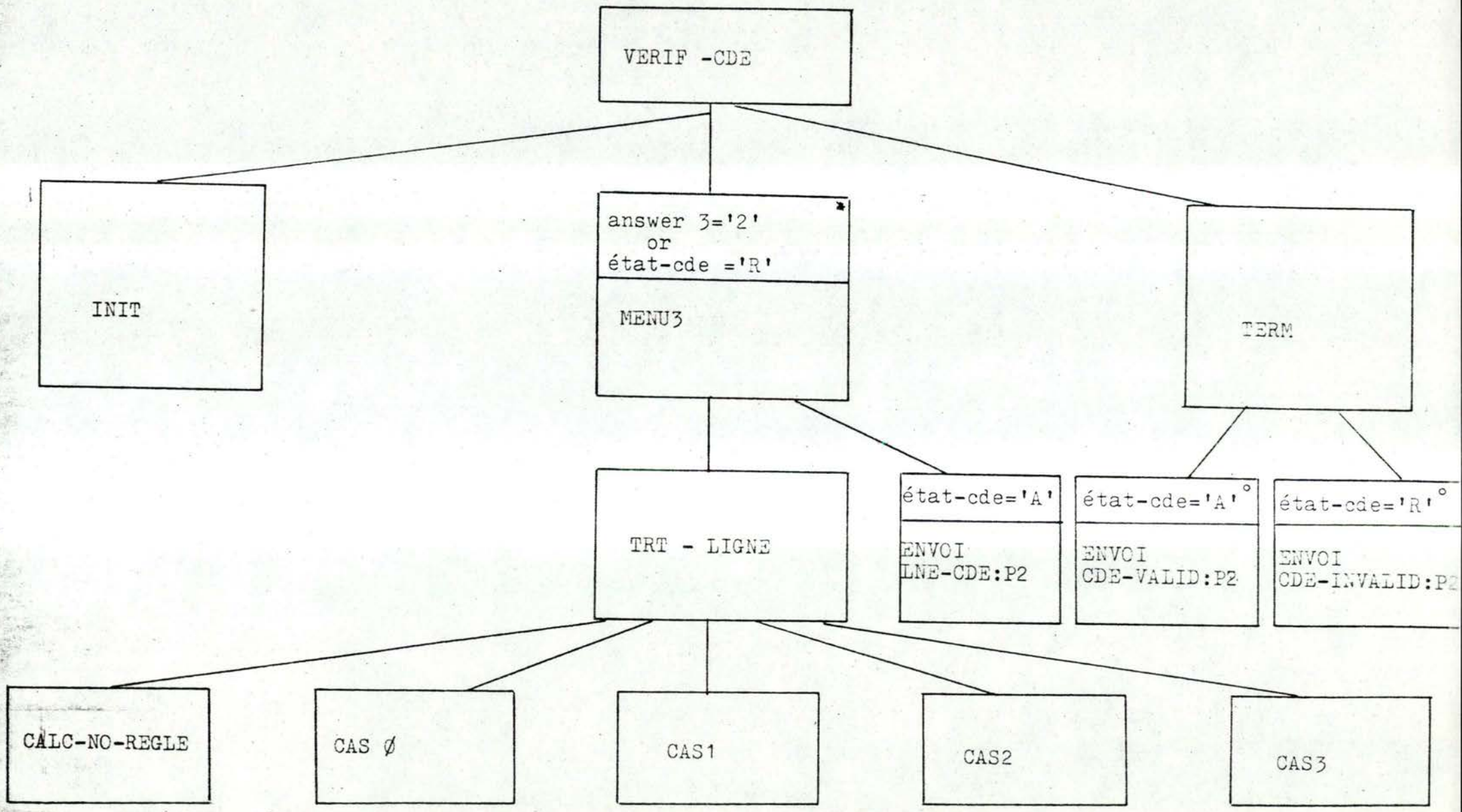
    UNITWRITE (18,BLOCK[4],1,0,0);

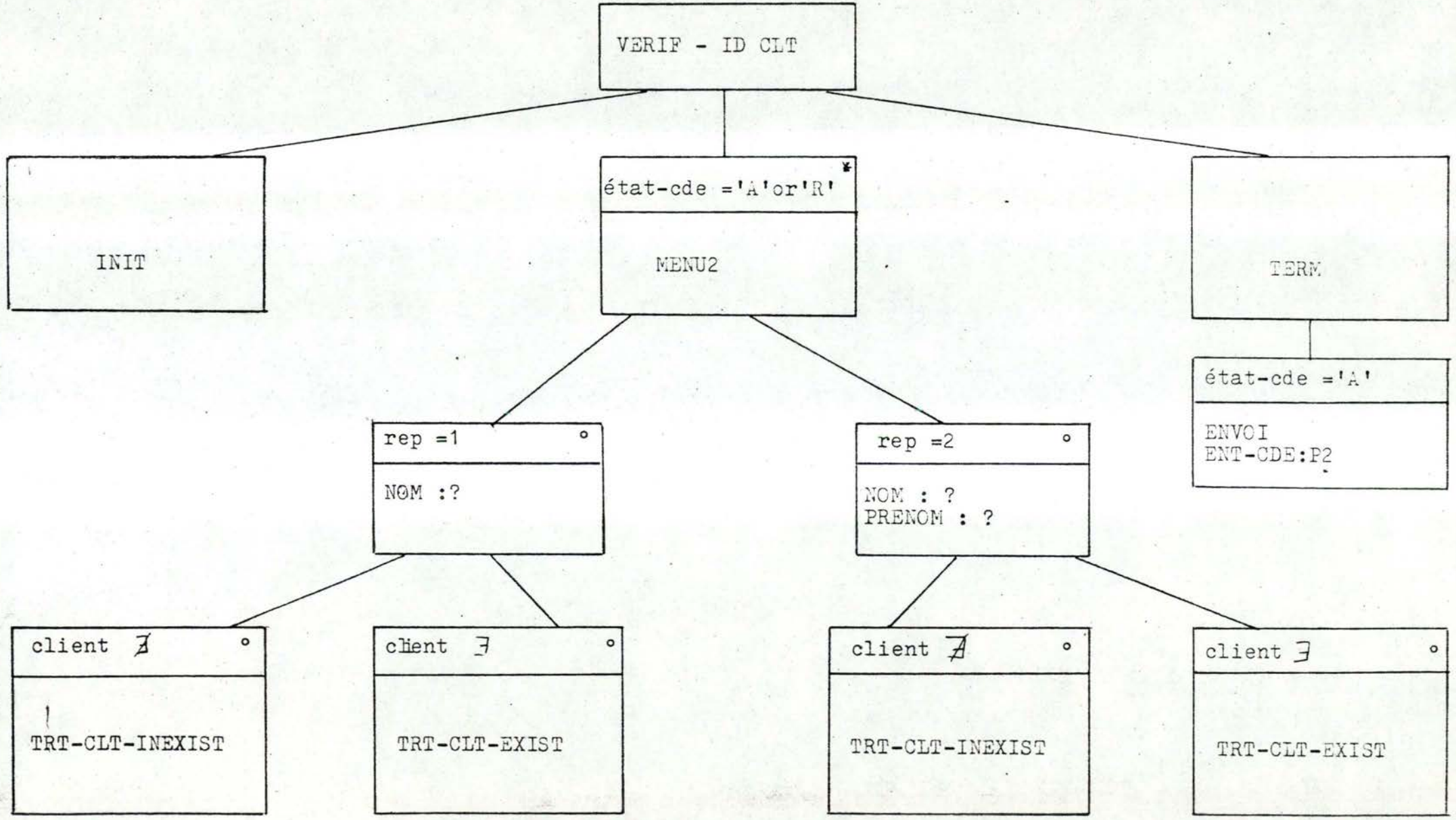
  END;
```

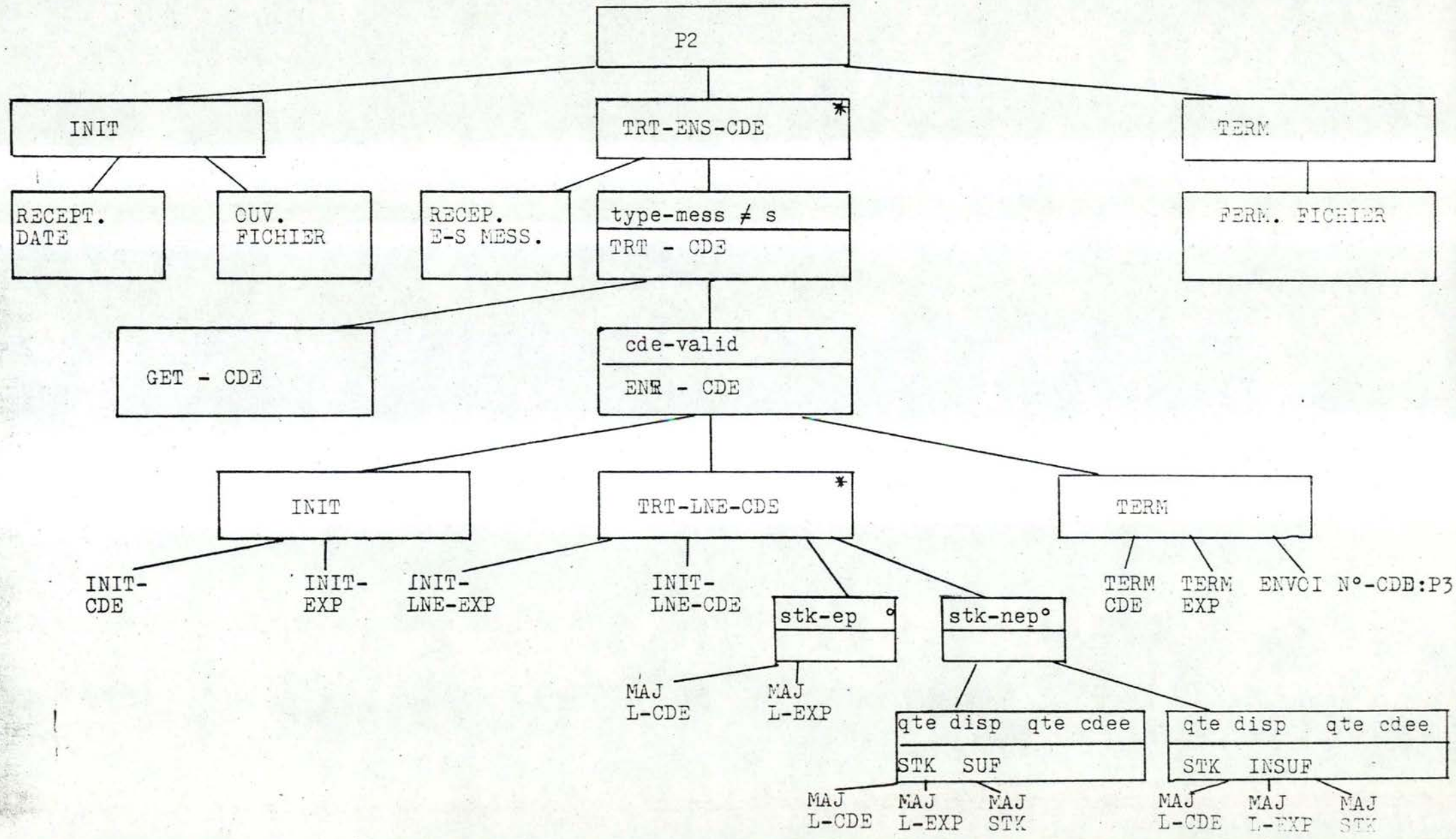
ANNEXE E

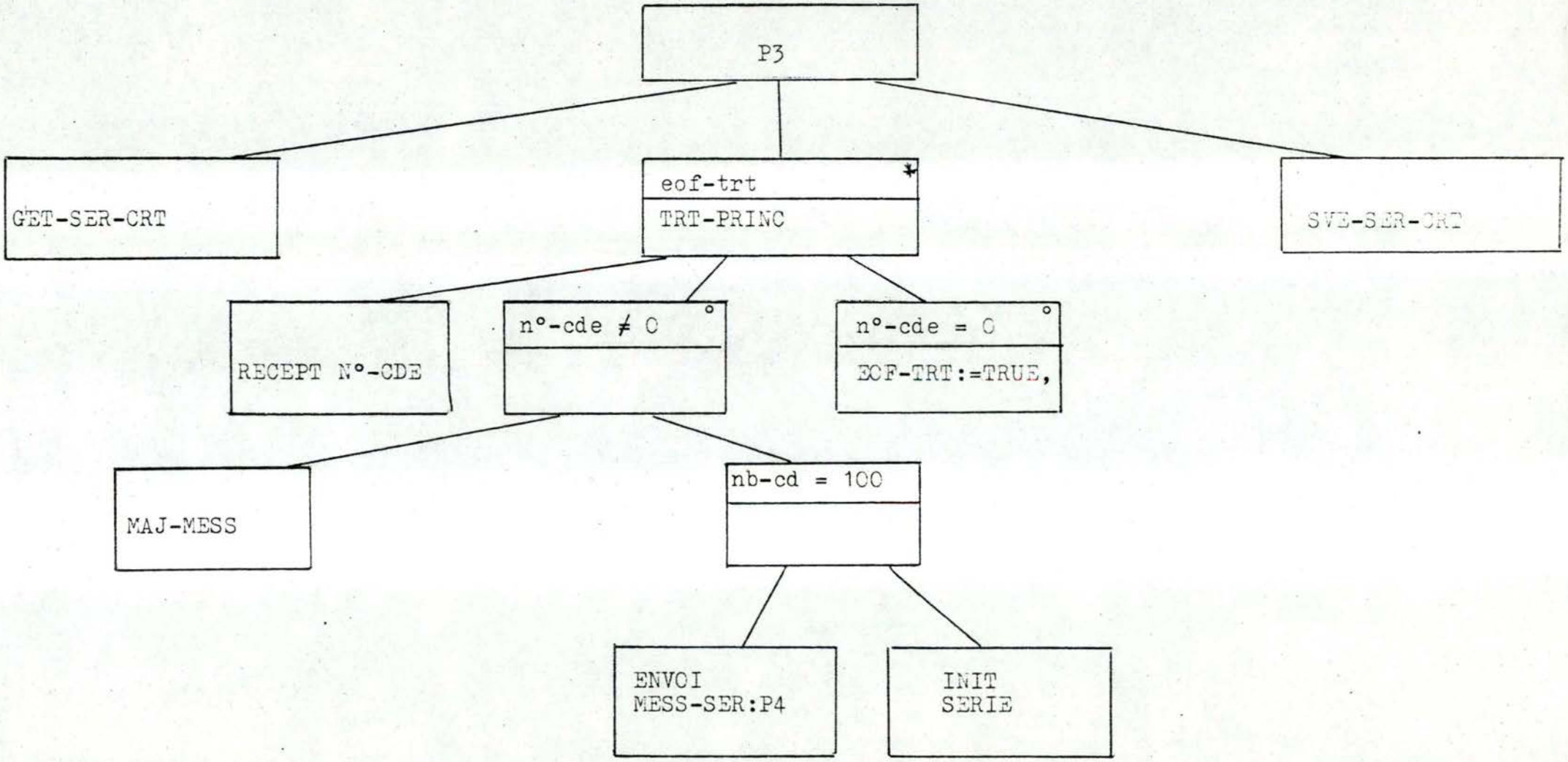
Programmes de l'application "PETIT PAS"

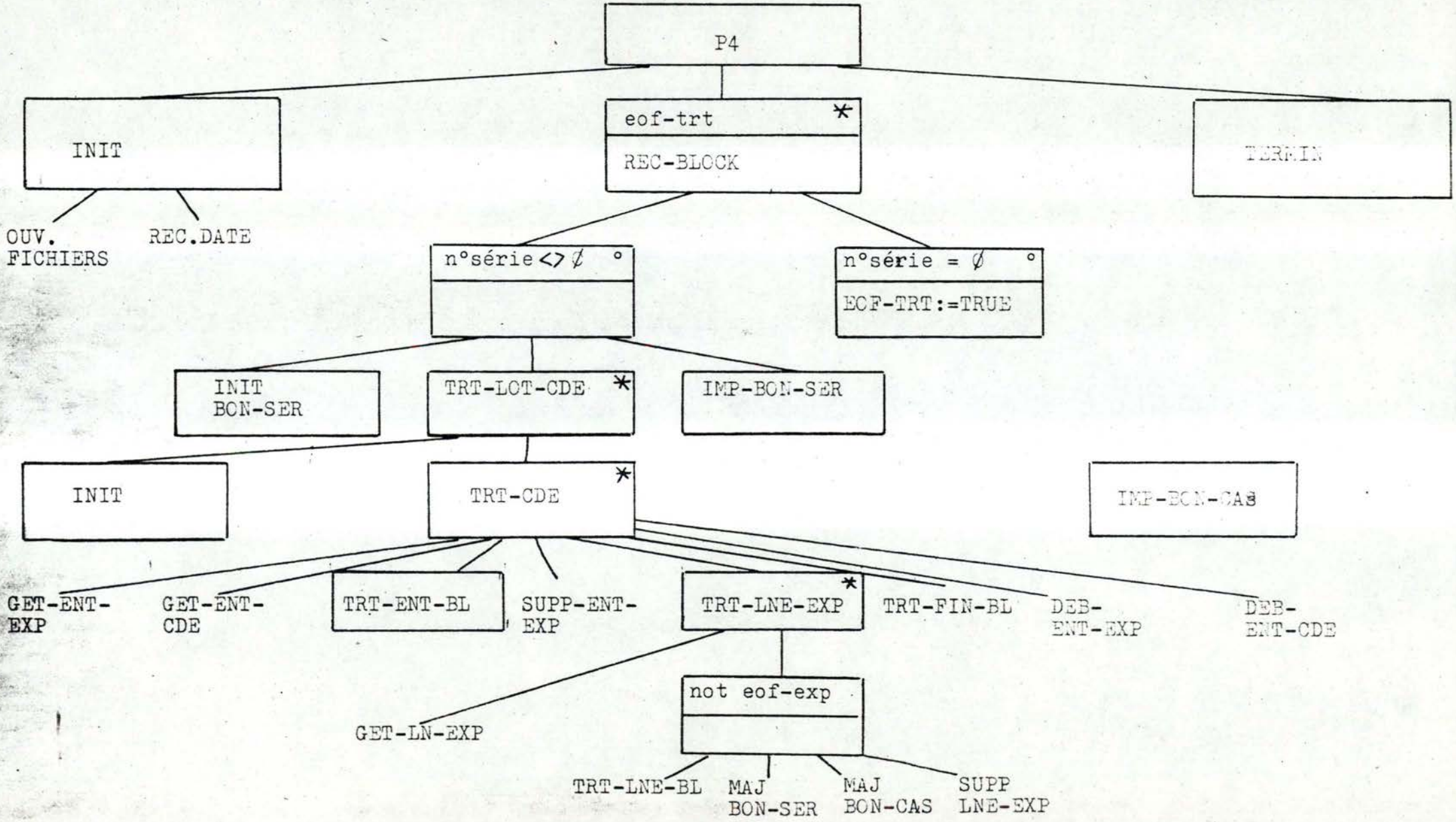

P1
INIT
MENU1
*
TERM
OUV.FICHIERS
GETDATE
ENVOI DATE:P2,P4
TRT-CDE
FERM.FICHIERS
ENVOI-MESS STOP:P2
VERIF-ID-CLT
état-cde ='A'
VERIF-CDE
état-cde ='A'
MAJ F-CLT

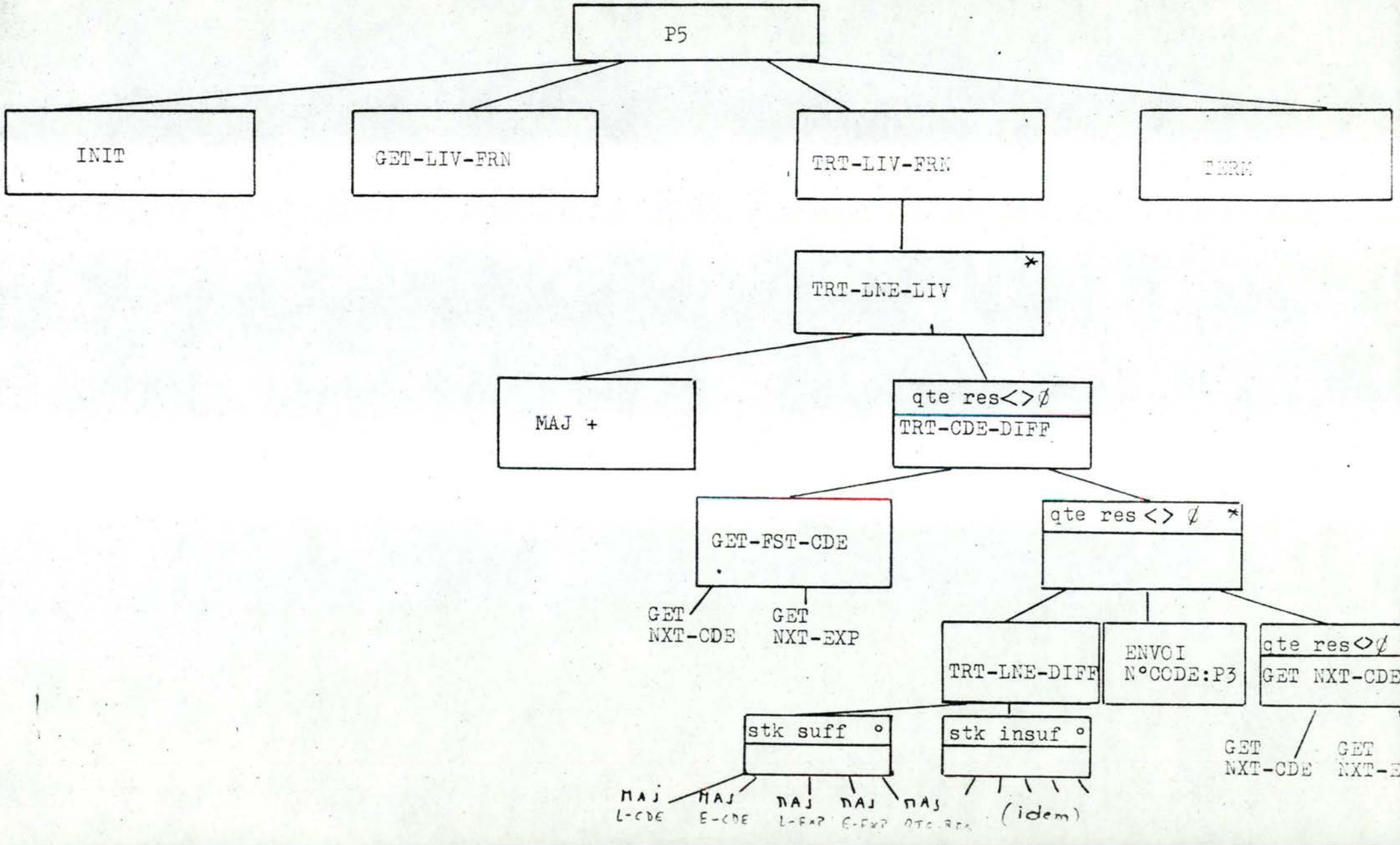

VERIF -CDE
INIT
answer 3='2'
or
état-cde ='R'
MENU3
TERM
TRT - LIGNE
état-cde='A'
ENVOI
LNE-CDE:P2
état-cde='A'°
ENVOI
CDE-VALID:P2
état-cde='R'°
ENVOI
CDE-INVALID:P2
CALC-NO-REGLE
CAS Ø
CAS1
CAS2
CAS3

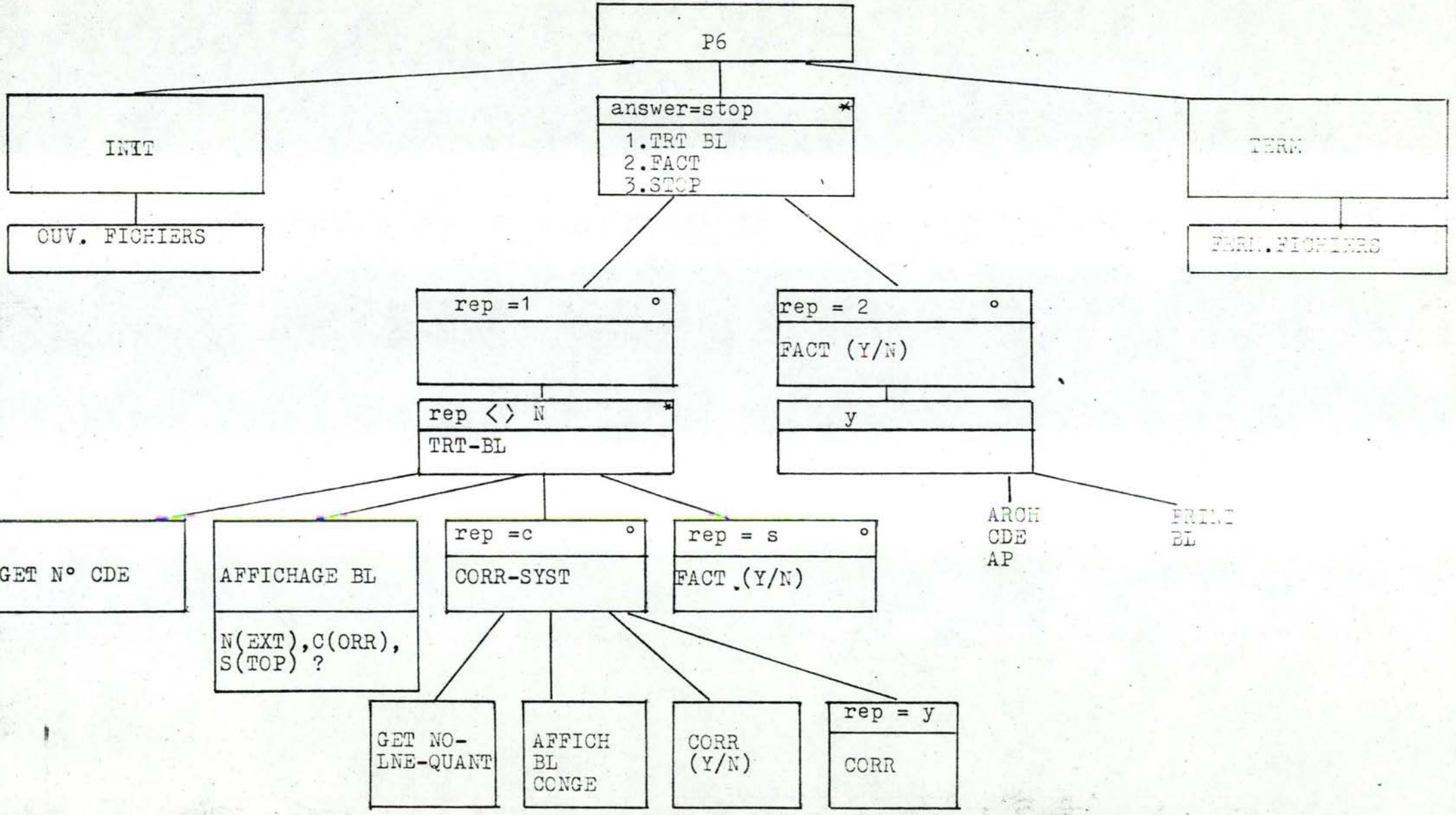

VERIF - ID CLT
INIT
état-cde ='A'or'R'
MENU2
TERM
état-cde ='A'
ENVOI
ENT-CDE:P2
rep =1
NOM :?
rep =2
NOM : ?
PRENOM : ?
client ∄
TRT-CLT-INEXIST
client ∃
TRT-CLT-EXIST
client ∄
TRT-CLT-INEXIST
client ∃
TRT-CLT-EXIST


P2
INIT
TRT-ENS-CDE
TERM
RECEPT. DATE
OUV. FICHIER
RECEP. E-S MESS.
type-mess ≠ s
TRT - CDE
FERM. FICHIER
GET - CDE
cde-valid
ENR - CDE
INIT
TRT-LNE-CDE
TERM
INIT-CDE
INIT-EXP
INIT-LNE-EXP
INIT-LNE-CDE
stk-ep °
stk-nep°
TERM CDE
TERM EXP
ENVOI N°-CDE:P3
MAJ L-CDE
MAJ L-EXP
qte disp qte cdee
STK SUF
qte disp qte cdee
STK INSUF
MAJ L-CDE
MAJ L-EXP
MAJ STK
MAJ L-CDE
MAJ L-EXP
MAJ STK


P3
GET-SER-CRT
eof-trt
TRT-PRINC
SVE-SER-CRT
RECEPT N°-CDE
n°-cde ≠ 0
n°-cde = 0
ECF-TRT:=TRUE,
MAJ-MESS
nb-cd = 100
ENVOI
MESS-SER:P4
INIT
SERIE


P4
INIT
eof-trt *
REC-BLOCK
TERMIN
OUV. FICHIERS
REC.DATE
n°série <> ∅ °
n°série = ∅ °
ECF-TRT:=TRUE
INIT BON-SER
TRT-LOT-CDE *
IMP-BON-SER
INIT
TRT-CDE *
IMP-BON-CAS
GET-ENT-EXP
GET-ENT-CDE
TRT-ENT-BL
SUPP-ENT-EXP
TRT-LNE-EXP *
TRT-FIN-BL
DEB-ENT-EXP
DEB-ENT-CDE
not eof-exp
GET-LN-EXP
TRT-LNE-BL
MAJ BON-SER
MAJ BON-CAS
SUPP LNE-EXP


P5
INIT
GET-LIV-FRN
TRT-LIV-FRN
TERM
TRT-LNE-LIV
MAJ +
qte res<>∅
TRT-CDE-DIFF
GET-FST-CDE
GET NXT-CDE
GET NXT-EXP
qte res <> ∅
TRT-LNE-DIFF
ENVOI N°CODE:P3
qte res<>∅
GET NXT-CDE
stk suff °
stk insuf °
(idem)


P6
INIT
OUV. FICHIERS
answer=stop *
1.TRT BL
2.FACT
3.STOP
TERM.
FERM. FICHIERS
rep =1 °
rep = 2 °
FACT (Y/N)
rep <> N *
TRT-BL
y
ARCH
CDE
AP
PRINT
BL
GET N° CDE
AFFICHAGE BL
N(EXT),C(ORR),
S(TOP) ?
rep =c °
CORR-SYST
rep = s °
FACT (Y/N)
GET NO-
LNE-QUANT
AFFICH
BL
CONGE
CORR
(Y/N)
rep = y
CORR

```
(*$S+*)
PROGRAM cont_cde;              (*cont1*)

USES
       (*$U FILESYS.CODE*)
       FILESYS,

       (*$U COMSYS.CODE*)
       COMSYS;

CONST

       debx                = 20;
       deby                = 2;
       tab1                = 4;

       lg_no_client        = 6;
       lg_nom_client       = 31;
       lg_prenom_client    = 15;
       lg_rue              = 25;
       lg_no_rue           = 5;
       lg_localite         = 25;
       lg_code_postal      = 5;

       nb_champ            = 9;

       client_no           = 0;
       clt_nom_prenom      = 1;
       prod_no             = 0;
       prod_lib            = 1;

       del                 = 127;
       right_arrow         = 6;
       left_arrow          = 4;
       up_arrow            = 18;
       down_arrow          = 3;
       esc                 = 27;
       def                 = 31;

       cle_no_clt          = 0;       (* cle fichier client *)
       cle_nompr_ct        = 1;

       cle_no_prod         = 0;       (* cle fichier produit *)
       cle_lib_prod        = 1;


TYPE
       rec_client          = RECORD
                               CASE BOOLEAN OF
                               TRUE : (arr:datarray);
                               FALSE: (REC: RECORD
                                 no           : INTEGER;
                                 nom          : STRING [31];
                                 prenom       : STRING [15];
                                 adresse      : RECORD
                                    rue           : STRING [25];
                                    no_rue        : STRING [5];
                                    localite      : STRING [25];
                                    code_postal   : STRING [5];
                                              END
                                           END)
                             END;
```

```
rec_produit      = RECORD
                    CASE BOOLEAN OF
                    TRUE : (arr:dat_array);
                    FALSE: (REC:RECORD
                                 no           : INTEGER;
                                 libelle      : STRING [51];
                                 unite_mes    : STRING [3];
                                 poids_unit   : INTEGER;
                                 loc          : STRING [5];
                                 prix_unit    : INTEGER;
                                 point_cde    : INTEGER;
                                 qte_eco_cde  : INTEGER;
                                 ind_epuist   : STRING [1];
                                 qte_a_cder   : INTEGER;
                               END);
                   END;

tm_date           = RECORD
                     CASE BOOLEAN OF
                     TRUE :(arr:mess_array);
                     FALSE:(mes:RECORD
                                 jour         : 1..31;
                                 mois         : 1..12;
                                 annee        : 80..99;
                               END)
                    END;

tm_cde            = RECORD
                       CASE BOOLEAN OF
                        TRUE :(arr:mess_array);
                        FALSE:(mes:RECORD
                                    ty_lne : t_lne_mes;
                                    CASE t_lne_mes OF
                                    e: (ent:RECORD
                             no_client : INTEGER;
                             nom       : STRING [31];
                             prenom    : STRING [15];
                             adresse   : RECORD
                               rue          : STRING [25];
                               no_rue       : STRING [5];
                               localite     : STRING [25];
                               code_postal  : STRING [5];
                                         END;
                                            END);
                                   l: (lne:RECORD
                              no_prod     : INTEGER;
                              libelle     : STRING [51];
                              qte_cdee    : INTEGER;
                              prix_unit   : INTEGER;
                              poids_unit  : INTEGER;
                                            END);
                                    f: (fin:RECORD
                                                valid : BOOLEAN;
                                             END) )
                                 END);
                       END;

tm_ack            = RECORD
                     CASE BOOLEAN OF
                     TRUE :(arr:mess_array);
```

```
                    FALSE:(mes:BOOLEAN);
                  END;


VAR

      f_client,f_produit : INTEGER;

      name_f_client      : t_filename;
      name_f_produit     : t_filename;

      ce_f_client        : INTEGER;
      ce_f_produit       : INTEGER;
      c_cle_nompr        : RECORD
                             CASE BOOLEAN OF
                             TRUE :(concat:key);
                             FALSE:(rec:RECORD
                                          nom:STRING[31];
                                          prenom:STRING[15];
                                        END)
                             END;

      v_cle_no_clt       : key;
      v_cle_no_prod      : key;
      v_cle_lib_prod     : key;

      occ_cle            : key_array;

      order              : STRING[1];
      char_order         : CHAR;
      leg_order          : BOOLEAN;
      valid_order        : STRING;
      answer1            : CHAR;
      type_maj           : (ECR,RECR);
      etat_cde           : CHAR;

      client             : rec_client;
      id_client          : INTEGER;
      no_client          : STRING;
      client_valid       : BOOLEAN;
      produit            : rec_produit;
      id_produit         : INTEGER;

      m_cde              : tm_cde;
      crt_date           : tm_date;
      ack                : tm_ack;

{*****************************************}
PROCEDURE get_order (valid_order: string);
{*****************************************}

BEGIN
  REPEAT
    BEGIN
      GOTOXY (20,23); READLN (order);
      IF POS (order,valid_order) = 0
      THEN
        BEGIN
          GOTOXY (20,23);
```

```
        WI    LN      EG.. OMM....');
          leg_order := FALSE;
        END
      ELSE
        leg_order:= TRUE;
    END
  UNTIL leg_order;

END;
(*************************************************************)

(****************************************)
PROCEDURE display_menu (valid_order:string);
(****************************************)

BEGIN

  IF POS ('E',valid_order) <> Ø
  THEN WRITELN ('E: RETOUR AU MENU COURANT');
  IF POS ('A',valid_order) <> Ø
  THEN WRITELN ('A: ACCEPTATION D"UNE COMMANDE');
  IF POS ('R',valid_order) <> Ø
  THEN WRITELN ('R: REFUS D"UNE COMMANDE');
  IF POS ('C',valid_order) <> Ø
  THEN WRITELN ('C: CREATION D"UN NOUVEAU CLIENT');
  IF POS ('U',valid_order) <> Ø
  THEN WRITELN ('U: MAJ DE L"ENREGISTREMENT CLIENT');
  IF POS ('?',valid_order) <> Ø
  THEN WRITELN ('?: LISTE DES COMMANDES ACCEPTEES A CE NIVEAU');

END;
(*****************************************************************)

(***********************)
PROCEDURE initialisation;
(***********************)

  VAR   i : INTEGER;

  BEGIN

    FOR i := 1 TO 1Ø DO occ_cle[i] := 1;

    name_f_client.str     := 'CLIENT.INDX ';
    name_f_produit.str    := 'PRODUIT.INDX ';

    OPENFILE (f_client,name_f_client.arr,ce_f_client);
    OPENFILE (f_produit,name_f_produit.arr,ce_f_produit);

    WRITELN ('DATE:?');
    WRITE ('  JOUR:'); READLN (crt_date.mes.jour);
    WRITE ('  MOIS:'); READLN (crt_date.mes.mois);
    WRITE ('  ANNEE:');READLN (crt_date.mes.annee);
    PAGE (OUTPUT);

    SENDMESS (crt_date.arr,lg,corr,local); { P1 => P2 }
    RECVMESS (ack.arr,lg,corr,local)       { P2 <= P1 }
    SENDMESS (crt_date.arr,lg,corr,local); { P1 => P4 }
    RECVMESS (ack.arr,lg,corr,local)       { P4 <= P2 }

  END;
(*****************************************************)
```

```
(**)    (**)    *)
PROCEDURE menu1;
(***************)

VAR
        leg_answer1: boolean;

  (***************)
  PROCEDURE trt_cde;
  (***************)

  VAR
        etat_cde        : CHAR;

    (*********************)
    PROCEDURE verif_id_clt;
    (*********************)



(*****************************************)
PROCEDURE vide_client (VAR client: rec_client);
(*****************************************)

  BEGIN

    no_client := ' ';
    WITH client.rec DO
      BEGIN
        nom     := ' ';
        prenom  := ' ';
        WITH adresse DO
          BEGIN
            rue         := ' ';
            no_rue      := ' ';
            localite    := ' ';
            code_postal  := ' ';
          END;
      END;
  END;

  (*****************************************)
  PROCEDURE affiche_client ( client: rec_client);
  (*****************************************)

  BEGIN

    WITH client.rec DO
      BEGIN
        GOTOXY (Ø    , deby+Ø)          ; WRITE (' CLIENT NO: ');
        GOTOXY (debx , deby+Ø)          ; WRITE ( no );
        GOTOXY (Ø    , deby+1)          ; WRITE (' NOM: ');
        GOTOXY (debx , deby+1)          ; WRITE ( nom );
        GOTOXY (Ø    , deby+2)          ; WRITE (' PRENOM: ');
        GOTOXY (debx , deby+2)          ; WRITE ( prenom );
        GOTOXY (Ø    , deby+3)          ; WRITE (' ADRESSE: ');
        WITH adresse DO
          BEGIN
            GOTOXY (tab1 , deby+5)          ; WRITE (' RUE: ');
            GOTOXY (debx+tab1 , deby+5)     ; WRITE ( rue );
            GOTOXY (tab1 , deby+6)          ; WRITE (' NO RUE: ');
            GOTOXY (debx+tab1 , deby+6)     ; WRITE ( no_rue );
            GOTOXY (tab1 , deby+7)          ; WRITE (' LOCALITE: ');
```

```
                XY      x+t     , d     7)          ;       E (     ali     ;
            GOTOXY (tabl , deby+8)                  ; WRITE (' CODE POSTAL: ');
            GOTOXY (debx+tabl , deby+8)             ; WRITE ( code_postal );
          END
      END;

  END;



(*************************************************************************)
PROCEDURE modif_client ( VAR client: rec_client; VAR valid: BOOLEAN);
(*************************************************************************)
VAR
        s                   : string;
        no_client           : STRING [6];
        one_char            : STRING [1];
        l,i,champ,x,maxx    : INTEGER;
        ch                  : CHAR;

   (*********************************)
   PROCEDURE adjust ( VAR name: STRING);
   (*********************************)

   VAR
         fini : BOOLEAN;

     BEGIN

       fini := FALSE; i := LENGTH (name);
       WHILE ( i>0 ) AND ( NOT fini) DO
         IF name [1] = ' '
         THEN
           BEGIN
             DELETE (name,1,1);
             INSERT (' ',name,LENGTH (name)+1);
             i := i-1;
           END
         ELSE
           fini := TRUE;

     END;

   (***************)
   PROCEDURE rdline;
   (***************)

     BEGIN

       REPEAT

         BEGIN
           READ (ch);
           CASE ORD (ch) OF
             right_arrow : IF x < maxx THEN x := x+1;
             left_arrow  : IF x > debx THEN x := x-1;
             del         : BEGIN
                             IF x > debx THEN x := x-1;
                             DELETE (s,(x-debx)+1,1);
                             INSERT (' ',s,1);
                           END;
         END;
```

```
      IF     IN      ..'     A'.     .'0     9'.     '-'     .'
        AND ( NOT EOLN (KEYBOARD))
      THEN
        BEGIN
          one_char := ' '; one_char[1] := ch;
          INSERT (one_char,s,(x-debx)+1);
          DELETE (s,l+1,1);
          IF x < maxx THEN x := x+1;
        END;
      GOTOXY (debx,deby+champ);
      adjust (s);
      WRITE (s);
      GOTOXY (x,deby+champ);
    END
  UNTIL (ORD (ch) IN [up_arrow,down_arrow,esc,def]) OR (EOLN(KEYBOARD));

END;

(*************************************************)
PROCEDURE conv_str_int (str:string; VAR int:INTEGER);
(*************************************************)

VAR
    l,i: INTEGER;

  BEGIN

    l := LENGTH (str); int := Ø;
    FOR i := 1 to l DO
      int := int*1Ø + (ord(str[i])-ord('Ø'));
  END;
(*************************************************************)


BEGIN (* procedure modif_client *)

  affiche_client (client);
  champ := Ø;
  REPEAT
    BEGIN
    CASE champ OF
      Ø: BEGIN l := lg_no_client ; s := no_client END;
      1: BEGIN l := lg_nom_client ; s := client.rec.nom END;
      2: BEGIN l := lg_prenom_client ; s := client.rec.prenom END;
    3,4: BEGIN s := '                              '; END;
      5: BEGIN l := lg_rue ; s := client.rec.adresse.rue END;
      6: BEGIN l := lg_no_rue ; s := client.rec.adresse.no_rue END;
      7: BEGIN l := lg_localite ; s := client.rec.adresse.localite END;
      8: BEGIN l := lg_code_postal ; s := client.rec.adresse.code_postal END
  IF LENGTH (s) < l THEN
    BEGIN
      FOR I := LENGTH (s) TO l-1 DO
        INSERT (' ',s,i+1);
    END;
  x := debx; maxx := (debx+l)-1;
  GOTOXY (x,deby+champ);
  rdline;
  adjust (s);
  GOTOXY (debx,deby+champ);
  WRITE (s);
  CASE champ OF
      Ø: no_client                  := s;
      1: client.rec.nom                 := s;
```

```
          2: client.rec.prenom                := s;
          5: client.rec.adresse.rue           := s;
          6: client.rec.adresse.no_rue        := s;
          7: client.rec.adresse.localite      := s;
          8: client.rec.adresse.code_postal   := s;
        END;

        CASE ORD (ch) OF
          down_arrow: IF champ > 0 THEN champ := champ-1;
          up_arrow  : IF champ < nb_champ - 1 THEN champ := champ+1;
        END;

        IF EOLN (KEYBOARD) AND (champ < nb_champ-1) THEN
          champ := champ+1;

      END

      UNTIL ORD (ch) IN [esc,def];

      CASE ORD (ch) OF
        esc: valid := FALSE;
        def: valid := TRUE;
      END;

      IF valid THEN conv_str_int (no_client,client.rec.no);

      PAGE (OUTPUT);

      GOTOXY (0,2);

    END; (* procedure modif_client*)
    (*********************************************************)

    (********************)
    PROCEDURE cree_client;
    (********************)
      BEGIN

        vide_client (client);
        modif_client(client,client_valid);

      END;

  (*$I CONT2.TEXT*)
  (*$I CONT3.TEXT*)

  (*********************************************************)
```

```
(**************)
PROCEDURE menu2;                    (*cont2*)
(**************)

VAR

      answer2: CHAR;
      leg_answer2: BOOLEAN;

(******************)
  PROCEDURE trt_num;
(******************)

    (********************)
    PROCEDURE client_exist;
    (********************)

    (********************)
      PROCEDURE trt_order;
    (********************)

      BEGIN

        char_order  := order[1];
        CASE char_order OF
          'E': EXIT (trt_num);
          'A': BEGIN
                 m_cde.mes.ent.no_client := client.rec.no;
                 etat_cde := 'A';
               END;
          'R': etat_cde := 'R';
          'C': BEGIN
                 cree_client;
                 IF client_valid THEN type_maj := ECR;
                 get_order ('EARCU?');
                 trt_order;
               END;
          'U': BEGIN
                 modif_client (client,client_valid);
                 IF client_valid THEN type_maj := RECR;
                 get_order ('EARC?');
                 trt_order;
               END;
          '?': BEGIN
                 display_menu ('EARCU?');
                 get_order ('EARCU?');
                 PAGE (OUTPUT);
                 trt_order;
               END;

        END;
      END; (* procedure trt_order ************************)


    BEGIN  (* procedure client_exist ********************)

      affiche-client (client);
      GOTOXY (20,23);
      get_order ('EARCU?');
      PAGE (OUTPUT);
      trt_order;
```

```
    END;  (* procedure client_exist ************************)

  (************************)
  PROCEDURE client_inexist;
  (************************)

  (********************)
    PROCEDURE trt_order;
  (********************)

      BEGIN  (* procedure trt_order ************************)

        char_order  := order[1];
        CASE char_order OF
          'e','E': EXIT (trt_num);
          'r','R': etat_cde := 'R';
          'c','C': BEGIN
                     cree_client;
                     IF client_valid THEN type_maj := ECR;
                     get_order ('EARC?');
                     PAGE (OUTPUT);
                     trt_order;
                   END;
          '?': BEGIN
                     display_menu ('ERC?');
                     get_order ('ERC?');
                     PAGE (OUTPUT);
                     trt_order;
                   END;

        END;

      END;  (* procedure trt_order ************************)


      BEGIN (* procedure client_inexist ******************)

        GOTOXY (20,6);
        WRITE ('LE CLIENT NO: ',cle_no_clt.int,'N"EXISTE PAS');
        WRITELN;
        get_order ('ERC?');
        PAGE (OUTPUT);
        trt_order;

      END; (* procedure client_inexist ******************)


BEGIN  (* procedure trt_num ******************************)

  GOTOXY (20,10); WRITE ('NUM CLIENT:?');
  READLN (v_cle_no_clt.int);

  READMOD (f_client,cle_no_clt,v_cle_no_clt.arr,id_client,
           client.arr,ce_f_client);

  PAGE (OUTPUT);

  IF ce_f_client = 0 THEN client_exist
                     ELSE client_inexist;
END;  (* procedure trt_num ******************************)


(********************)
```

```
PROCEDURE trt_nom_prenom;
(**********************)

VAR
    key_nom_pren     :STRING;
    nom              :STRING[31];
    prenom           :STRING[15];
    code_ret         :INTEGER;

  (********************)
  PROCEDURE client_exist;
  (********************)

  VAR
      tab_ord         :STRING;

    (*****************)
    PROCEDURE trt_order;
    (*****************)

      BEGIN  (* procedure trt_order ************************)

        char_order  := order[1];
        CASE char_order OF
          'E','e': EXIT (trt_nom_prenom);
          'R','r': etat_cde := 'R';
          'A','a': BEGIN
                     m_cde.mes.ent.no_client :=
                      client.rec.no;
                     etat_cde := 'A';
                   END;
          'C','c': BEGIN
                     cree_client;
                     IF client_valid THEN type_maj := ECR;
                     get_order ('EARCU?');
                     PAGE(OUTPUT);
                     trt_order;
                   END;
          'U','u': BEGIN
                     modif_client (client, client_valid);
                     IF client_valid THEN type_maj := RECR;
                     get_order ('EARC?');
                     trt_order;
                   END;
            '?': BEGIN
                     display_menu ('EARCU?');
                     get_order ('EARCU?');
                     PAGE (OUTPUT);
                     trt_order;
                   END;

        END; (* case *)

      END; (* procedure trt_order ************************)


  BEGIN (* procedure client_exist ***********************)

    affiche-client (client);
    GOTOXY (20,23);
    get_order ('EARCU?');
    PAGE (OUTPUT);
    trt_order;
```

```
END. . procedure client_exist      ********************)

(***********************)
PROCEDURE client;
(***********************)

VAR
      tab_ord: STRING;
  (******************)
  PROCEDURE trt_order;
  (******************)

    BEGIN  (* procedure trt_order *********************)

    char_order  := order[1];
    CASE char_order OF
        'E','e' : EXIT (trt_nom_prenom);
        'A','a' : BEGIN
                    m_cde.mes.ent.no_client :=
                    client.rec.no;
                    etat_cde := 'A';
                  END;
        'R','r' : etat_cde := 'R';
        'C','c' : BEGIN
                  cree_client;
                  IF client_valid THEN type_maj := ECR;
                  get_order ('EARCU?');
                  PAGE (OUTPUT);
                  trt_order;
                END;
          '?' : BEGIN
                  display_menu ('EARC?');
                  get_order ('EARCU?');
                  PAGE (OUTPUT);
                  trt_order;
                END;
      END; (* case *)

    END; (* procedure trt_order **********************)


  BEGIN (* procedure client ***************************)

    GOTOXY (20,5);
    WRITE ('LE CLIENT DE '); WRITELN; WRITELN;
    WRITELN ('     NOM: ', nom);
    WRITELN ('     PRENOM: ',prenom); WRITELN;
    WRITELN ('N"EXISTE PAS');
    get_order ('EARC?');
    PAGE (OUTPUT);
    trt_order;

  END; (* procedure client ***************************)

BEGIN (* procedure trt_nom_prenom ********************)

    GOTOXY (20,6);
    WRITE ('NOM: ?'); READLN (c_cle_nompr.cle.nom);
    WRITE ('PRENOM: ?'); READLN (c_cle_nompr.cle.prenom);

    READMOD (f_client,cle_nompr_clt,c_cle_nompr.concat.arr,
             id_client,client.arr,ce_f_client);
```

```
    PAGE (OUTPUT);

    GOTOXY (20,10); WRITE ('CODE RET:? ');
    READLN (code_ret);

    CASE code_ret OF
      0 : client_exist;
      1 : client_inexist;
    END;

  END; (* procedure trt_nom_prenom **********************)

  BEGIN  (* procedure menu2 ****************************)

    GOTOXY (20,5);
    WRITE ('CONSULTATION DU FICHIER CLIENT SUR BASE:');
    WRITELN; WRITELN; WRITELN;
    WRITE ('   1. DU NUMERO CLIENT?');
    WRITELN; WRITELN;
    WRITE ('   2. DES NOM ET PRENOM?');
    WRITELN; WRITELN;

    REPEAT
      BEGIN
        GOTOXY (20,10); READLN (answer2);
        IF (answer2 <> '1') AND (answer2 <> '2')
        THEN
          BEGIN
            GOTOXY (20,23); WRITE ('ILLEGAL COMMAND!');
            leg_answer2 := FALSE;
          END
        ELSE
          leg_answer2 := TRUE;
      END
    UNTIL leg_answer2;

    PAGE (OUTPUT);

    CASE answer2 OF
      '1': trt_num;
      '2': trt_nom_prenom;
    END;

  END; (* procedure menu2 ****************************)


BEGIN (************************************************)

  (* initialisation *)
  GOTOXY (20,10); WRITE ('PHASE D"IDENTIFICATION DU CLIENT');

  REPEAT menu2 UNTIL ( etat_cde = 'A') OR (etat_cde = 'R');

  (* termination *)
  IF etat_cde = 'R'
  THEN
    BEGIN
      GOTOXY (20,23);
      WRITE ('COMMANDE REFUSEE');
    END
  ELSE
    BEGIN
      WITH m_cde.mes.ent DO
```

```
        BEGIN
          no_client         := client.rec.no;
          prenom            := client.rec.nom;
          adresse.rue       := client.rec.adresse.rue;
          adresse.no_rue    := client.rec.adresse.no_rue;
          adresse.localite  := client.rec.adresse.localite;
          adresse.code_postal := client.rec.adresse.code_postal;

          m_cde.mess.type_lne := E;

          SENDMESS (m_cde.arr,lg,corr,local);

          RECVMESS (ack.arr,lg,corr,local);

        END;
    END;
END;  (*************************************************)
```

```
(*****************)
PROCEDURE verif_cde;          (*cont3*)
(*****************)

CONST

     frais_exp    = 40;


VAR
     line_no      : INTEGER;
     answer3      : CHAR;
     leg_answer3  : BOOLEAN;
     tot_ligne,tot_cde : INTEGER;
     prod         : rec_produit;

  (*************)
  PROCEDURE init;
  (*************)

    BEGIN  (* procedure init ***************************)

      WRITELN ('PHASE 2: VERIFICATION DU CORPS DE LA COMMANDE');

      line_no := 1;
      tot_cde := frais_exp;

    END;  (* procedure init ***************************)

  (**************)
  PROCEDURE menu3;
  (**************)

    (*****************)
    PROCEDURE trt_ligne;
    (*****************)

    VAR
        no_regle : INTEGER;
        code_ret : INTEGER;

      (*******************)
      PROCEDURE calc_no_regle;
      (*******************)

        BEGIN  (* procedure calc_no_regle **************)

          GOTOXY (20,6);
          WRITE ('NO PRODUIT: ?'); READLN (cle_no_prod.aln);
          WRITELN;
          WRITE ('LIBELLE DU PRODUIT: ?'); READLN (cle_lib_prod.aln);
          no_regle := 0;
          IF cle_no_prod.aln <> ' ' THEN no_regle := no_regle + 2;
          IF cle_lib_prod.aln <> ' ' THEN no_regle := no_regle + 2;

        END;  (* procedure calc_no_regle **************)

        (*****************)
        PROCEDURE trt_valid;
        (*****************)
```

```
    IN  .  procedure trt_valid ****************)
    (* maj ligne message *)

    m_cde.mess.lne.no_prod := prod.rec.no;
    m_cde.mess.lne.libelle:= prod.rec.libelle;
    WRITE ('QUANTITE COMMANDEE: ?');
    READLN (m_cde.mess.lne.qte_cdee);
    m_cde.mess.lne.prix_unit :=
    prod.rec.prix_unit;
    m_cde.mess.lne.poids_unit:=
    prod.rec.poids_unit;
    tot_ligne := prod.rec.prix_unit *
                 m_cde.mess.lne.qte_cdee;

    (* affichage de la ligne du bon de commande *)

    PAGE (OUTPUT); GOTOXY (20,26);
    WRITELN ('NO PRODUIT:          ', prod.rec.no);
    WRITELN ('LIBELLE:             ', prod.rec.libelle);
    WRITELN ('QUANTITE COMMANDEE: ',
    m_cde.mess.lne.qte_cdee);

    WRITELN ('PRIX UNITAIRE:       ',prod.rec.prix_unit);
    WRITELN ('TOTAL LIGNE:         ',tot_ligne);

    GOTOXY (20,23); WRITE ('TYPE <RET> TO CONTINUE');
    READLN;

  END;  (* procedure trt_valid ****************)

  (*******************)
  PROCEDURE trt_invalid;
  (*******************)

    BEGIN  (* procedure trt_invalid ***********)

      etat_cde := 'R';

    END;  (* procedure trt_invalid ************)


  BEGIN (* procedure trt_ligne ***************)

    calc_no_regle;
    CASE no_regle OF
      0 : trt_invalid;
      1 : BEGIN


            READINDX (f_produit,cle_lib_prod,
            v_cle_lib_prod.arr,produit.arr,ce_f_produit);


            CASE ce_f_produit OF
              0 : trt_valid;
              1 : trt_invalid;
            END;
          END;
      2 : BEGIN


            READINDX (f_produit,cle_no_prod,
            v_cle_no_prod.arr,produit.arr,ce_f_produit);
```

```
            CASE ce_f_produit OF
              Ø : trt_valid;
              1 : trt_invalid;
            END;
          END;
      3 : BEGIN


            READINDX (f_produit,cle_lib_prod,
            v_cle_lib_prod.arr,produit.arr,ce_f_produit);


            CASE ce_f_produit OF
              Ø : trt_valid;
              1 : trt_invalid;
            END;
          END;
    END; (* case *)

    line_no := line_no + 1;

  END; (* procedure trt_ligne ****************)

(*************)
PROCEDURE term;
(*************)

VAR
    answer       : CHAR;
    leg_answer   : BOOLEAN;

  BEGIN  (* proceudre term *********************)

    IF line_no = 1 THEN etat_cde := 'R';

    CASE etat_cde OF
      'R' : BEGIN
              GOTOXY (2Ø,23);
              WRITE ('COMMANDE CLIENT REFUSEE!');
            END;
      'A' : BEGIN
              GOTOXY (2Ø,6);
              WRITELN ('MONTANT TOTAL: ',tot_cde);
              REPEAT
                BEGIN
                  GOTOXY (2Ø,8);
                  WRITE ('Y/N ?'); READLN (answer);
                  IF (answer <> 'Y') AND (answer <> 'N')
                  THEN
                    BEGIN
                      GOTOXY (2Ø,23); WRITE ('ILLEGAL ANSWER');
                      leg_answer := FALSE;
                    END
                  ELSE
                    leg_answer := TRUE;
                END
              UNTIL leg_answer;

              CASE answer OF
                'Y','y' : etat_cde := 'A';
                'N','n' : etat_cde := 'R';
              END;
            END;
```

```
            END;
        END; (* procedure term **********************)

        BEGIN (* procedure menu3 *********************)

          GOTOXY (20,6); WRITE ('VOTRE CHOIX ?');
          WRITELN; WRITELN;WRITELN;
          WRITE ('    1.ENREGISTRER UNE LIGNE');
          WRITELN ('    2.STOP');

          REPEAT
            BEGIN
              GOTOXY (34,6); READLN (answer3);
              IF (answer3 <> '1') AND (answer3 <> '2')
              THEN
                BEGIN
                  GOTOXY (20,23); WRITE ('ILLEGAL ANSWER');
                  leg_answer3 := FALSE
                END
              ELSE
                leg_answer3 := TRUE;
            END
          UNTIL leg_answer3;

          CASE answer3 OF
            '1' : trt_ligne;
          END;

          IF etat_cde = 'A'
          THEN
            BEGIN

              m_cde.mes.type_lne := L;

              SENDMESS (m_cde.arr,lg,corr,local); (P1 => P2)
              RECVMESS (ack.arr,lg,corr,local);   (P2 <= P1)

            END;

        END; (* procedure menu3 *********************)

        BEGIN (* procedure verif_cde *)

            REPEAT menu3 UNTIL (answer3 = '2') OR (etat_cde ='R');

            IF etat_cde = 'A'
            THEN
              m_cde.mess.fin.valid := TRUE
            ELSE
              m_cde.mess.fin.valid := FALSE;

            m_cde.mes.type_lne := F;

            SENDMESS (m_cde.arr,lg,corr,local); (P1 => P2)
            RECVMESS (ack.arr,lg,corr,local);   (P2 <= P1)

        END; (* procedure menu3 *********************)


BEGIN (*****************************************************
```

```
        _cde     'I .
    verif_id_clt;
    IF etat_cde = 'A'
    THEN
      verif_cde;
    IF etat_cde = 'A'
    THEN
      BEGIN
        (* maj du fichier clients *)

        CASE type_maj OF

          ECR:  WRITEINDX (f_client,occ_cle,client.arr,ce_f_client);

          RECR: MODIFY (f_client,id_client,occ_cle,client.arr,ce_f_client);

        END;


      END;
  END;  (*******************************************************)

BEGIN  (*******************************************************)

 GOTOXY (20,6); WRITE ('VOTRE CHOIX:? ');
 GOTOXY (26,10);
 WRITELN ('1. ENREGISTREMENT D"UNE COMMANDE');
 WRITELN ('2. STOP');

 REPEAT
   BEGIN
     GOTOXY (20,14); READLN (answer1);
     IF (answer1 <> '1') AND (answer1 <> '2')
     THEN
       BEGIN
         GOTOXY (20,23); WRITE ('ILLEGAL COMMAND!');
         leg_answer1 := FALSE;
       END
     ELSE
       leg_answer1 := TRUE;
   END
 UNTIL leg_answer1;

 CASE answer1 OF
   '1': trt_cde;
 END;

 END;  (*******************************************************)

 (*********************)
 PROCEDURE termination;
 (*********************)

   BEGIN  (* procedure termination ****************************)

     CLOSEFILE (f_client,ce_f_client);
     CLOSEFILE (f_produit,ce_f_produit);

     m_cde.mess.type_lne := 0;

     SENDMESS (m_cde.arr,lg,corr,local); ( P1 => P2)
     RECVMESS (ack.arr,lg,corr,local);   ( P2 <= P1)
```

```
END;  (* procedure terminale ********** (**) (**)

BEGIN (* program principal ************************************)
   initialisation;
   REPEAT menu1 UNTIL answer1 = '2';
   termination;

END.  (* programme principal ************************************)
```

```
PROGRAM  enr_maj_neg;

CONST    cle_ncde_nlne  = Ø; (* cles fichier cde *)
         cle_elne_npro  = 1;
         cle_nexp       = 2;

         cle_npro       = Ø; (* cle fichier stk *)

         cle_nexp_nlne  = Ø; (* cles fichier exp *)
         cle_ncde       = 1;


TYPE     mess_array     = PACKED ARRAY [8..25Ø] OF byte;

         lne_cde_acq    = RECORD
                            no_prod     : INTEGER;
                            libelle     : STRING[51];
                            qte_cdee    : INTEGER;
                            prix_unit   : INTEGER;
                            poids_unit  : INTEGER;
                          END;

         t_date         = RECORD
                            jour  :  1..31;
                            mois  :  1..12;
                            annee : 8Ø..99;
                          END;

         tm_no_cde      = RECORD
                           CASE BOOLEAN OF
                           TRUE :(arr:mess_array);
                           FALSE:(mes:INTEGER);
                          END;

         tm_ack         = RECORD
                           CASE BOOLEAN OF
                           TRUE :(arr:mess_array);
                           FALSE:(mes:BOOLEAN);
                          END;

         tm_cde         = RECORD
                           CASE BOOLEAN OF
                           TRUE :(arr:mess_array);
                           FALSE:(mes:
                                                          RECORD
                              ty_lne : t_lne_mes;
                              CASE t_lne_mes OF
                              e: (ent:RECORD
                                    no_client : INTEGER;
                                    nom       : STRING[31];
                                    prenom    : STRING[15];
                                    adresse   : RECORD
                                      rue         : STRING[25];
                                      no_rue      : STRING[5];
                                      localite    : STRING[25];
                                      code_postal : STRING[5];
                                                END
                                  END);
                               l: (lne:RECORD
```

```
                    no_prod    : INTEGER;
                    libelle    : STRING[51];
                    qte_cdee   : INTEGER;
                    prix_unit  : INTEGER;
                    poids_unit : INTEGER;
                         END);
                f: (fin:RECORD
                    valid : BOOLEAN
                         END)
                    END)
              END;

tm_date       = RECORD
                  CASE BOOLEAN OF
                  TRUE :(arr:mess_array);
                  FALSE:(mes:RECORD
                               jour   :  1..31;
                               mois   :  1..12;
                               annee  : 80..99;
                             END)
                  END;

rec_ent_cde   = RECORD
                  CASE BOOLEAN OF
                  TRUE:(arr:data_array);
                  FALSE:(rec:RECORD
                               no_cde     :INTEGER;
                               no_lne     :INTEGER;
                               etat_lne   :INTEGER;
                               nb_lne_diff:INTEGER;
                               date       :t_date;
                               no_clt     :INTEGER;
                               nom        :STRING[31];
                               prenom     :STRING[15];
                               adresse    :RECORD
                                rue         :STRING[25];
                                no_rue      :STRING[5];
                                localite    :STRING[25];
                                code_postal:STRING[25];
                                            END;
                               ind_l_liv  :BOOLEAN;
                               no_exp     :INTEGER;
                             END);
                  END;

rec_lne_cde   = RECORD
                  CASE BOOLEAN OF
                  TRUE:(arr:data_array);
                  FALSE:(rec:RECORD
                               no_cde  :INTEGER;
                               no_lne  :INTEGER;
                               etat_lne:INTEGER;
                               no_prod :INTEGER;
                               qte_cdee:INTEGER;
                               qte_liv :INTEGER;
                               qte_due :INTEGER;
                             END);
                  END;

rec_ent_exp   = RECORD
                  CASE BOOLEAN OF
                  TRUE:(arr:data_array);
                  FALSE:(rec:RECORD
                               no_exp     :INTEGER;
```

```
                                    no_lne      :INTEGER;
                                    no_cde      :INTEGER;
                                    date        :t_date;
                                    poids_colis:INTEGER;
                                    montant     :INTEGER;
                                   END);
                        END;

    rec_lne_exp     = RECORD
                        CASE BOOLEAN OF
                        TRUE:(arr:data_array);
                        FALSE:(rec:RECORD
                                    no_exp       :INTEGER;
                                    no_lne       :INTEGER;
                                    no_prod      :INTEGER;
                                    libelle      :STRING[51];
                                    prix_unit    :INTEGER;
                                    poids_unit   :INTEGER;
                                    qte_a_liv    :INTEGER;
                                    motif_refus  :STRING[31];
                                   END)
                        END;

     rec_prod_stk   = RECORD
                         CASE BOOLEAN OF
                         TRUE:(arr:data_array);
                         FALSE:(rec:RECORD
                                     no             :INTEGER;
                                     indic_epuist   :BOOLEAN;
                                     qte_disp       :INTEGER;
                                     qte_diff       :INTEGER;
                                    END)
                         END;



VAR      f_cde              :INTEGER;
         f_exp              :INTEGER;
         f_stk              :INTEGER;
         name_f_cde         :t_filename;
         name_f_exp         :t_filename;
         name_f_stk         :t_filename;
         ce_f_cde           :INTEGER;
         ce_f_exp           :INTEGER;
         ce_f_stk           :INTEGER;

         v_cle_no_prod      :key;

         occ_cle            :key_array;

         no_cde_crt         :INTEGER;
         no_lne_trt         :INTEGER;
         no_exp_crt         :INTEGER;
         eof_trt            :BOOLEAN;

         ent_cde            :rec_ent_cde;
         id_ent_cde         :INTEGER;
         lne_cde            :rec_lne_cde;
         id_lne_cde         :INTEGER;
         ent_exp            :rec_ent_exp;
         id_ent_exp         :INTEGER;
         lne_exp            :rec_lne_exp;
```

```
      id_lne_exp      :INTEGER;
      prod_stk        :rec_prod_stk;
      id_prod_stk     :INTEGER;

      cde_acq         : RECORD
                          ent : RECORD
                                  no_client : INTEGER;
                                  nom       : STRING[31];
                                  prenom    : STRING[15];
                                  adresse   : RECORD
                                    rue         : STRING[25];
                                    no_rue      : STRING[5];
                                    localite    : STRING[25];
                                    code_postal : STRING[5];
                                                END;
                                END;
                           bdy : ARRAY [1..20] of lne_cde_acq;
                          END;

      crt_date        :tm_date;
      m_cde           :tm_cde;


(**********************)
PROCEDURE initialisation;
(**********************)

  VAR   i :INTEGER;

  BEGIN  (* procedure initialisation ************************)

    (* ouverture des fichiers *)
       name_f_cde.str := 'cde.idx';
       OPENFILE (f_cde,name_f_cde.arr,ce_f_cde);
       name_f_exp.str := 'exp.idx';
       OPENFILE (f_exp,name_f_exp.arr,ce_f_exp);
       name_f_stk.str := 'stk.idx';
       OPENFILE (f_stk,name_f_stk.arr,ce_f_stk);

    (* reception de la date *)
       RECVMESS (crt_date.arr,,,);
       SENDMESS (ack.arr,,,);

    FOR i := 1 TO 10 DO occ_cle[i] := 1;

  END;  (* procedure initialisation ************************)


(********************)
PROCEDURE trt_ens_cde;
(********************)


(*****************)
PROCEDURE trt_cde;
(*****************)

   VAR  nb_lne_cde : INTEGER;
        nb_lne_trt : INTEGER;
        nb_lne_diff : INTEGER;
```

```
[redacted]***     **)
PROCEDURE get_cde;
(*****************)

   VAR  no_lne : INTEGER;

   BEGIN  (* procedure get_cde ********************************)

     cde_acq.ent := m_cde.mes.ent;
     no_lne := 1;
     RECVMESS (m_cde.arr,,,);
     SENDMESS (ack.arr,,,);
     WHILE m_cde.mes.type_lne <> f do
           BEGIN
             cde_acq.bdy[no_lne] := m_cde.mes.lne;
             RECVMESS (m_cde.arr,,,);
             SENDMESS (ack.arr,,,);
             no_lne := no_lne + 1;
           END;
     nb_lne_cde := no_lne - 1;

   END;  (* procedure get_cde ********************************)
    ;

(****************)
PROCEDURE enr_cde;
(****************)



(*************)
PROCEDURE init;
(*************)

   BEGIN  (* procedure init **********************************)

     (* initialisation de la commande *)
          GETNUMBER (no_cde_crt);
          nb_lne_diff := Ø;

     (* initialisation de l'expedition *)
          GETNUMBER (no_exp_crt);
          ent_exp.rec.poids_colis := Ø;
          ent_exp.rec.montant := Ø;

   END;  (* procedure init **********************************)


(*******************)
PROCEDURE trt_lne_cde;
(*******************)


(***************)
PROCEDURE stk_ep;
(***************)

   BEGIN  (* procedure stk_ep ********************************)

     (* maj ligne cde et enregistrement *)
         WITH lne_cde.rec DO
              BEGIN
                   etat_lne := e;
                   qte_liv  := Ø;
```

```
                    _due      Ø;
            END;
        WRITEINDX (true,f_cde,occ_cle,lne_cde.arr,ce_f_cde);

    (* maj ligne exp et enregistrement *)
        WITH lne_exp.rec DO
            BEGIN
                qte_liv     := Ø;
                motif_refus := 'stock epuise';
            END;
        WRITEINDX (true,f_exp,occ_cle,lne_exp.arr,ce_f_exp);

END;  (* procedure stk_ep ********************************)


(****************)
PROCEDURE stk_nep;
(****************)


(****************)
PROCEDURE stk_suf;
(****************)

BEGIN  (* procedure stk_suf ******************************)

    (* maj de la ligne de commande et enregistrement *)
        WITH lne_cde.rec DO
            BEGIN
                etat_lne := a;
                qte_liv  := qte_cdee;
                qte_due  := Ø;
            END;
        WRITEINDX (true,f_cde,occ_cle.lne_cde.arr,ce_f_cde);

    (* maj de la ligne expedition et enregistrement *)
        WITH lne_exp.rec DO
            BEGIN
                qte_a_liv := lne_cde.rec.qte_cdee;
                motif_refus := '';
            END;
        WRITEINDX (true,f_exp,occ_cle,lne_exp.arr,ce_f_exp);

    (* maj entete cde *)
        IF ent_cde.rec.etat_cde = e
           THEN ent_cde.rec.etat_cde := a;

    (* maj entete exp *)
        WITH ent_exp.rec DO
            BEGIN
                poids_colis := poids_colis +
                   (lne_cde.rec.qte_liv * lne_cde.rec.poids_unit);
                montant := montant +
                   (lne_cde.rec.qte_liv * lne_cde.rec.prix_unit);
            END;

    (* maj etat du stock et enregistrement *)
        WITH prod_stk.rec DO
            BEGIN
                qte_disp := qte_disp - lne_cde.rec.qte_liv;
            END;
        REWRITE ();

END;  (* procedure stk_suf ******************************)
```

```
(******************)
PROCEDURE stk_insuf;
(******************)

  BEGIN  (* procedure stk_insuf ******************************)

    (* maj ligne cde et enregistrement *)
        WITH lne_cde.rec DO
             BEGIN
                  etat_lne := d;
                  qte_liv := prod_stk.rec.qte_disp;
                  qte_due := qte_cdee - prod_stk.rec.qte_disp;
             END;
        WRITEINDX (true,f_cde,occ_cle,lne_cde.arr,ce_f_cde);

    (* maj ligne exp et enregistremnet *)
        WITH lne_exp.rec DO
             BEGIN
                  qte_a_liv := prod_stk.rec.qte_disp;
                  motif_refus := 'reste plus tard';
             END;
        WRITEINDX (true,f_exp,occ_cle,lne_exp.arr,ce_f_exp);

    (* maj entete cde *)
        WITH ent_cde.rec DO
             BEGIN
                  etat_cde := Ø;
                  nb_lne_diff := nb_lne_diff + 1;
             END;

    (* maj entete exp *)
        WITH ent_exp.rec DO
             BEGIN
                  poids_colis := poids_colis +
                       (lne_cde.rec.qte_liv * lne_cde.rec.poids_unit);
                  montant := montant +
                       (lne_cde.rec.qte_liv * lne_cde.rec.prec.prix_unit);
             END;

     (* maj stock et enregistremnet *)
          WITH prod_stk.rec DO
               BEGIN
                    qte_disp := Ø;
                    qte_diff := qte_diff + lne_cde.rec.qte_due;
               END;
          REWRITE ();

  END;  (* procedure stk_insuf ******************************)


(* instructions de la procedure stk_nep *)

  BEGIN  (* procedure stk_nep ******************************)

    IF prod_stk.rec.qte_disp >= lne_cde.rec.qte_cdee
      THEN stk_suf
      ELSE stk_insuf;

  END;  (* procedure stk_nep ******************************)
```

```
*)    uct     de    pro    re    _lne_    *)
  BEGIN  (* procedure trt_lne_cde ***************************)

    (* initialisation de la ligne d'expedition *)
       WITH lne_exp.rec DO
            BEGIN
              no_exp      := no_exp_crt;
              no_lne      := no_lne_trt;
              no_prod     := cde_acq.bdy[no_lne_trt].no_prod;
              libelle     := cde_acq.bdy[no_lne_trt].libelle;
              prix_unit   := cde_acq.bdy[no_lne_trt].prix_unit;
              poids_unit  := cde_acq.bdy[no_lne_trt].poids_unit;
            END;

    (* initialisation de la ligne de commande *)
       WITH lne_cde.rec DO
            BEGIN
              no_cde    := no_cde_crt;
              no_lne    := no_lne_trt;
              no_prod   := cde_acq.bdy[no_lne_trt].no_prod;
              qte_cdee  := cde_acq.bdy[no_lne_trt].qte_cdee;
            END;

    (* consultation du stock *)
       v_key_no_prod.int := cde_acq.bdy[no_lne_trt].no_prod;
       READINX (true,f_stk.key_no_prod,v_key_no_prod,prod_stk.arr,
                ce_f_stk);
       IF prod_stk.rec.indic_epuist
          THEN stk_ep
          ELSE stk_nep;

  END:  (* procedure trt_lne_cde ***************************)


(*************)
PROCEDURE term;
(*************)

  BEGIN  (* procedure term ***********************************)

    (* term de ent_cde et enregistrement *)
       WITH ent_cde.rec DO
            BEGIN
              no_cde     := no_cde_crt;
              no_lne     := Ø;
              date       := crt_date.mes;
              no_client  := m_cde.ent.no_client;
              nom        := m_cde.ent.nom;
              prenom     := m_cde.ent.prenom;
              adresse    := m_cde.ent.adresse;
              IF (ent_cde.rec.etat_cde = d) AND
                        (ent_exp.rec.poids_colis = Ø)
                 THEN ind_i_liv := true
                 ELSE ind_i_liv := false;
              no_exp     := no_exp_crt;
            END;
       WRITEINDX (true,f_cde,occ_cle,ent_cde.arr,ce_f_cde);

    (* term de ent_exp et enregistrement *)
       WITH ent_exp.rec DO
            BEGIN
              no_exp := no_exp.crt;
```

```
                n__  e :
                date   := crt_date.mes;
                no_cde := no_cde_crt;
              END;
          WRITEINDX (true,f_exp,occ_cle,ent_exp.arr,ce_f_exp);

    END;  (* procedure term ***********************************)


(* instructions de la procedure enr_cde *)

    BEGIN  (* procedure enr_cde ********************************)

      init;
      FOR no_lne_trt := 1 TO nb_lne_cde DO trt_lne_cde;
      term;

    END;  (* procedure enr_cde ********************************)


(* instructions de la procedure trt_cde *)

    BEGIN  (* procedure trt_cde ********************************)

      get_cde:
      IF m_cde.mes.fin.valid THEN enr_cde;

    END;  (* procedure trt_cde ********************************)

(* instructions de la procedure trt_enr_cde *)

    BEGIN  (* procedure trt_enr_cde ****************************)

      (* reception ligne message *)
           RECVMESS (m_cde.arr,,,);
           SENDMESS (ack.arr,,,);
           IF m_cde.mes.type_lne <> s
              THEN trt_cde
              ELSE eof_trt := true;

    END;  (* procedure trt_enr_cde ****************************)


(********************)
PROCEDURE termination;
(********************)

    BEGIN  (* procedure termination ****************************)

      (* fermeture des fichiers *)
           CLOSEFILE (f_cde,ce_f_cde);
           CLOSEFILE (f_exp,ce_f_exp);
           CLOSEFILE (f_stk,ce_f_stk);

    END;  (* procedure termination ****************************)


(* instructions du programme principal *)

    BEGIN  (* programme principal ******************************)
```

```
    initialisation;
    REPEAT trt_enr_cde UNTIL eof_trt;
    termination;

END.  (* programme principal ******************************)
```

```
    CLOSEFILE(f_cde,ce_f_cde);
    CLOSEFILE(f_exp,ce_f_exp);

END. (* programme principal ********************************)
```

```
PROGRAM  const_ser;


TYPE   rec_ser    = RECORD
                      CASE BOOLEAN OF
                        TRUE  : (arr   : mess_array);
                        FALSE : (mess : RECORD
                                          no_serie : INTEGER;
                                          arr_cde  : ARRAY[1..100] OF INTEGER);
                                        END);
                      END;

       tm_no_cde = RECORD
                      CASE BOOLEAN OF
                        TRUE  : (arr: mess_array);
                        FALSE : (mes: INTEGER);
                      END;


VAR    f_ser          : FILE OF rec_ser;
       m_ser          : rec_ser;

       no_cde_found : BOOLEAN;
       end_ser        : BOOLEAN;
       i              : INTEGER;

       m_no_cde       : tm_no_cde;


(******************)
PROCEDURE trt_princ;
(******************)

  BEGIN  (* procedure trt_princ ******************************)

    RECVMESS (m_no_cde.arr,,,);
    SENDMESS (ack.arr,,,);

    IF m_no_cde.mess <> 0
       THEN BEGIN
              i := 1;
              WHILE NOT (end_ser) AND NOT (no_cde_found) DO
                    BEGIN
                     no_cde_found := m_ser.mess.arr_cde[i] =
                                          m_no_cde.mess;
                     end_ser := m_ser.mess.arr_cde[i] = 0;
                     i := i + 1;
                    END;
              IF NOT no_cde_found
                 THEN m_ser.mess.arr_cde[i - 1] := m_no_cde.mess;
              IF i > 100
                 THEN BEGIN
                       SENDMESS (m_ser.arr,,,);
                       RECVMESS (ack.arr,,,);
                       m_ser.mess.no_serie := m_ser.mess.no_serie + 1;
                       FOR i := 1 TO 100 DO m_ser.mess.arr_cde[i] := 0;
                      END;
            END
       ELSE eof_trt := true;
```

```
PROGRAM enr_maj_neg;

CONST

        cle_ncde_nlne = 0;  (* cles fichier cde *)
        cle_elne_npro = 1;
        cle_nexp      = 2;
        cle_nexp_nlne = 0;  (* cles fichier exp *)
        cle_ncde      = 1;

        cle_nser_nref = 0;  (* cle fichier bon_liv *)


TYPE

        t_bon_ser     = RECORD
                          no_ser    :INTEGER;
                          cas  :ARRAY [1..10] OF RECORD
                                    no_cas:INTEGER;
                                    nb_prod  :INTEGER;
                                    prod :ARRAY [1..5]
                                          OF RECORD
                                              no_prod:INTEGER;
                                              qte    :INTEGER;
                                             END;
                                                  END;
                        END;

        t_bon_cas     = RECORD
                          no_cas :INTEGER;
                          cde        :ARRAY [1..10] OF RECORD
                                      no_cde :INTEGER;
                                      no_exp :INTEGER;
                                                  END;
                        END;

        rec_bon_liv   = RECORD
                          CASE BOOLEAN OF
                          TRUE:(arr:data_array);
                          FALSE:(rec:RECORD
                                    no_ser :INTEGER;
                                    no_ref :INTEGER;
                                    no_bl  :INTEGER;
                                    date   :t_date;
                                    no_clt :INTEGER;
                                    nom    :INTEGER;
                                    prenom :STRING[15];
                                    adresse:RECORD
                                      rue         :STRING[25];
                                      no_rue      :STRING[5];
                                      localite    :STRING[25];
                                      code_postal:STRING[5];
                                          END;
                                    lne_bl:ARRAY [1..10] OF RECORD
                                      no_prod     :INTEGER;
                                      libelle     :STRING[51];
                                      qte_liv     :INTEGER;
                                      prix_unit   :INTEGER;
                                      montant     :INTEGER;
                                      motif_refus:STRING[31];
                                                  END;
```

```
                                frais_exp:INTEGER;
                                total    :INTEGER;
                               END);
                    END;

tm_ser          = RECORD
                      CASE BOOLEAN OF
                      TRUE :(arr:mess_array);
                      FALSE:(mes:RECORD
                                  no_serie : INTEGER;
                                  co_cde   : ARRAY[1..100]
                                             OF INTEGER;
                                 END)
                    END;

tm_date         = RECORD
                      CASE BOOLEAN OF
                      TRUE :(arr:mess_array);
                      FALSE:(mes:RECORD
                                  jour    :  1..31;
                                  mois    :  1..12;
                                  annee   : 80..99;
                                 END)
                    END;

t_date          = RECORD
                      jour   :  1..31;
                      mois   :  1..12;
                      annee  : 80..99;
                     END;

rec_ent_cde     = RECORD
                      CASE BOOLEAN OF
                      TRUE:(arr:data_array);
                      FALSE:(rec:RECORD
                                  no_cde     :INTEGER;
                                  no_lne     :INTEGER;
                                  etat_lne   :INTEGER;
                                  nb_lne_dif:INTEGER;
                                  date       :tm_date;
                                  no_clt     :INTEGER;
                                  nom        :STRING[31];
                                  prenom     :STRING[15];
                                  adresse    :RECORD
                                   rue          :STRING[25];
                                   no_rue       :STRING[5];
                                   localite     :STRING[25];
                                   code_postal:STRING[25];
                                                END;
                                  ind_l_liv  :BOOLEAN;
                                  no_exp     :INTEGER;
                                 END);
                    END;

rec_lne_cde     = RECORD
                      CASE BOOLEAN OF
                      TRUE:(arr:data_array);
                      FALSE:(rec:RECORD
                                  no_cde   :INTEGER;
                                  no_lne   :INTEGER;
                                  etat_lne:INTEGER;
                                  no_prod  :INTEGER;
                                  qte_cdee:INTEGER;
                                  qte_liv  :INTEGER;
```

```
                                qte_due :INTEGER;
                               END);
                    END;

    rec_ent_exp  = RECORD
                    CASE BOOLEAN OF
                    TRUE:(arr:data_array);
                    FALSE:(rec:RECORD
                                no_exp      :INTEGER;
                                no_lne      :INTEGER;
                                no_cde      :INTEGER;
                                date        :t_date;
                                poids_colis:INTEGER;
                                montant     :INTEGER;
                               END);
                    END;

    rec_lne_exp  = RECORD
                     CASE BOOLEAN OF
                     TRUE:(arr:data_array);
                     FALSE:(rec:RECORD
                                 no_exp       :INTEGER;
                                 no_lne       :INTEGER;
                                 no_prod      :INTEGER;
                                 libelle      :STRING[51];
                                 prix_unit    :INTEGER;
                                 poids_unit   :INTEGER;
                                 qte_a_liv    :INTEGER;
                                 motif_refus :STRING[31];
                                END)
                     END;


VAR
    f_cde            :INTEGER;
    f_exp            :INTEGER;
    f_bon_liv        :INTEGER;
    name_f_cde       :t_filename;
    name_f_exp       :t_filename;
    name_f_bon_liv   :t_filename;
    ce_f_cde         :INTEGER;
    ce_f_exp         :INTEGER;
    ce_f_bon_liv     :INTEGER;

    c_cle_nser_nref : RECORD
                       CASE BOOLEAN OF
                       TRUE:(concat:key);
                       FALSE:(rec:RECORD
                                   no_serie:INTEGER;
                                   no_ref  :INTEGER;
                                  END);
                      END;
    c_cle_nexp_nlne : RECORD
                       CASE BOOLEAN OF
                       TRUE:(concat:key);
                       FALSE:(rec:RECORD
                                   no_exp:INTEGER;
                                   no_lne:INTEGER;
                                  END)
                      END;
    c_cle_ncde_nlne : RECORD
                       CASE BOOLEAN OF
                       TRUE:(concat:key);
```

```
                                    no_cde:INTEGER;
                                    no_lne:INTEGER;
                                   END)
                    END;
      c_cle_elne_npro : RECORD
                         CASE BOOLEAN OF
                         TRUE:(concat:key);
                         FALSE:(rec:RECORD
                                    etat_lne:INTEGER;
                                    no_prod :INTEGER;
                                   END)
                         END;

      v_cle_nexp       : key;
      v_cle_ncde       : key;

      occ_cle          : key_array;

      bon_ser          : t_bon_ser;
      bon_cas          : t_bon_cas;
      eof_trt          : BOOLEAN;

      ent_cde          : rec_ent_cde;
      id_ent_cde       : INTEGER;
      lne_cde          : rec_lne_cde;
      id_lne_cde       : INTEGER;
      ent_exp          : rec_ent_exp;
      id_ent_exp       : INTEGER;
      lne_exp          : rec_lne_exp;
      id_lne_exp       : INTEGER;
      bon_liv          : rec_bon_liv;

      m_ser            : tm_ser;
      crt_date         : tm_date;

(******************)
PROCEDURE recv_blk;
(******************)

   VAR i:INTEGER;

(********************)
PROCEDURE trt_lot_cde;
(********************)

   VAR j:INTEGER;

(****************)
PROCEDURE trt_cde;
(****************)

   VAR eof_exp    :BOOLEAN;
       no_lne_bl :INTEGER;

(********************)
PROCEDURE trt_ent_bl;
(********************)

   BEGIN  (* procedure trt_ent_bl ****************************)

       WITH bon_liv.rec DO
            BEGIN
```

```
                    _ser       :  m_ser.mes.no_serie;
                   no_ref      := m_ser.mes.no_cde[(i*10)+j];
                   get_number(no_bl);
                   date        := ent_exp.rec.date;
                   no_clt      := ent_cde.rec.no_clt;
                   nom         := ent_cde.rec.nom;
                   prenom      := ent_cde.rec.prenom;
                   rue         := ent_cde.rec.rue;
                   no_rue      := ent_cde.rec.no_rue;
                   localite    := ent_cde.rec.localite;
                   code_postal := ent_cde.rec.code_postal;
                END;
        no_lne_bl := 0;

   END;  (* procedure trt_ent_bl ******************************)


(*********************)
PROCEDURE trt_lne_exp;
(*********************)

   VAR prod_found :BOOLEAN;
       l          :INTEGER;

   BEGIN  (* procedure trt_lne_exp ****************************)

     (* get_ligne_expedition *)
        REPEAT
             READNEXT(f_exp,cle_nexp_nlne,lne_exp.arr,ce_f_exp);
           UNTIL NOT (ent_exp.rec.poids_colis=0) AND
                     (lne_exp.rec.motif_refus='');
        IF lne_exp.rec.no_cde <> m_ser.mes.no_cde[(i*10)+j]
           THEN eof_exp := TRUE
           ELSE no_lne_bl := no_lne_bl+1;
        IF NOT eof_exp
           THEN BEGIN
                (* trt_ligne_bon_livr *)
                   WITH bon_liv.rec.lne_bl[no_lne_bl] DO
                        BEGIN
                         no_prod := lne_exp.rec.no_prod;
                         libelle := lne_exp.rec.libelle;
                         qte_liv := lne_exp.rec.qte_a_liv;
                         prix_unit := lne_exp.rec.prix_unit;
                         montant := lne_exp.rec.qte_a_liv *
                                       lne_exp.rec.prix_unit;
                         motif_refus := lne_exp.rec.motif_refus;
                        END;
                    WRITEINDX(f_bon_liv,occ_cle,bon_liv.arr,ce_f_exp);

                (* maj_bon_ser *)
                    prod_found := false;
                    l := 1;
                    WHILE ( NOT prod_found) AND
                          (l <= bon_ser.cas[i] - nb_prod) DO
                      BEGIN
                       prod_found := lne_exp.rec.no_prod
                                   = bon_ser.cas[i].prod[l].no_prod;
                       l := l+1;
                      END;
                    IF prod_found
                       THEN bon_ser.cas[i].prod[l-1].qte
                            :=bon_ser.cas[i].prod[l-1].qte
                              + lne_exp.rec.qte_a_liv
```

```
                    ELSE WITH bon_ser.cas[i] DO
                         BEGIN
                          nb_prod := nb_prod + 1;
                          prod[i].no_prod := lne_exp.rec.no_prod;
                          prod[i].qt := lne_exp.rec.qte_a_liv;
                         END;

                (* maj_bon_par_cas *)
                    WITH bon_cas.cde[j] DO
                         BEGIN
                          no_cde := m_serie.mes.no_cde[(i*10)+j];
                          no_exp := ent_exp.rec.no_exp;
                         END;

                (* suppression_ligne_expedition *)
                    WITH lne_exp.rec DO
                         BEGIN
                          qte_a_liv := 0;
                          motif_refus := '';
                         END;
                  REWRITE ........

   END; (* procedure trt_lne_exp ****************************)


(* instructions de la procedure trt_cde *)

   BEGIN  (* procedure trt_cde ******************************)

     (* get entete expedition* )
        c_cle_nexp_nlne.rec.no_exp := m_ser.mes.no_cde[(i*10)+j];
        c_cle_nexp_nlne.rec.no_lne := 0;
        READINDX(f_exp,cle_nexp_nlne,c_cle_nexp_nlne.concat,
                 ent_exp.arr,ce_f_exp);

     (* get entete commande *)
        c_cle_ncde_nlne.rec.no_cde := m_serie.mes.no_cde[(i*10)+j];
        c_cle_ncde_nlne.rec.no_lne := 0;
        READINDX(f_cde,cle_ncde_nlne,c_cle_ncde_nlne.concat,
                 ent_cde.arr,ce_f_cde);
        trt_ent_bl;

     (* suppression entete expedition *)
        WITH ent_exp.rec DO
             BEGIN
               poids_colis := 0;
               montant := 0;
             END;
        REPEAT trt_lne_exp UNTIL eof_exp;

     (* trt fin bon livr *)
        IF ent_cde.rec.ind_l_liv
           THEN IF ent_exp.rec.poids_colis = 0
                   THEN bon_liv.rec.frais_exp := 0
                   ELSE BEGIN
                         bon_liv.rec.frais_exp := frais_exp;
                         ent_cde.rec.ind_l_liv := false;
                        END
           ELSE bon_liv.rec.frais_exp := 0;
        bon_liv.rec.total := ent_exp.rec.montant +
                             bon_liv.rec.frais_exp;
```

```
    de    uer    _cd     en __p *.
  END;  (* procedure trt_cde ********************************)


(* instructions de la procedure trt_lot_cde *)

  BEGIN  (* procedure trt_lot_cde ***************************)

    (* initialisation *)
       bon_ser.cas.no_cas := i;
       bon_ser.cas.nb_prod := Ø;
       bon_cas.no_cas := i;
       FOR j := 1 TO 1Ø DO trt_cde;

    (* terminaison *)

    (* impression du bon par cas *)

  END:  (* procedure trt_lot_cde ***************************)

  ;

(* instructions de la procedure recv_blk *)

  BEGIN  (* procedure recv_blk ******************************)

    IF m_ser.mes.no_serie <> Ø
       THEN BEGIN
              (* initialisation du bon par serie *)
                 bon_ser.no_serie := m_ser.mes.no_serie;
                 FOR i := 1 TO 1Ø DO trt_lot_cde;

              (* terminaison du bon par serie *)

              (* impression du bon_ser avec bonne mise en page*)
            END
       ELSE eof_trt := TRUE;

  END:  (* procedure recv_blk ******************************)


(* instructions du programme principal *)

  BEGIN  (* programme principal ******************************)

    (* ouverture des fichiers *)
       name_f_cde.str := 'cde';
       OPENFILE(f_cde,name_f_cde.arr,ce_f_cde);
       name_f_exp.str := 'exp';
       OPENFILE(f_exp,name_f_exp.arr,ce_f_exp);
       name_f_bon_liv := 'bon_liv';
       OPENFILE(f_bon_liv,name_f_bon_liv.arr,ce_f_bon_liv);

    (* reception date *)
       RECVMESS(crt_date.arr,.....
       SENDMESS(ack.arr,.....
       REPEAT recv_blk UNTIL eof_trt;

    (* fermeture des fichiers *)
       CLOSEFILE(f_bon_liv,ce_f_bon_liv);
```

```
PROGRAM enr_maj_pos;

CONST   cle_ncde_nlne = Ø;  (* cles fichier cde *)
        cle_elne_npro = 1;
        cle_nexp      = 2;

        cle_nexp_nlne = Ø;  (* cles fichier exp *)
        cle_ncde      = 1;

        cle_npro      = Ø;  (* cle fichier stk *)


TYPE

        t_date        = RECORD
                          jour  :  1..31;
                          mois  :  1..12;
                          annee : 8Ø..99;
                        END;

        tm_no_cde     = RECORD
                          CASE BOOLEAN OF
                          TRUE  :(arr:mess_array);
                          FALSE:(mes:INTEGER);
                        END;

        rec_ent_cde   = RECORD
                            CASE BOOLEAN OF
                            TRUE:(arr:data_array);
                            FALSE:(rec:RECORD
                                         no_cde     :INTEGER;
                                         no_lne     :INTEGER;
                                         etat_lne   :INTEGER;
                                         nb_lne_dif:INTEGER;
                                         date       :t_date;
                                         no_clt     :INTEGER;
                                         nom        :STRING[31];
                                         prenom     :STRING[15];
                                         adresse    :RECORD
                                          rue          :STRING[25];
                                          no_rue       :STRING[5];
                                          localite     :STRING[25];
                                          code_postal:STRING[25];
                                                       END;
                                         ind_1_liv  :BOOLEAN;
                                         no_exp     :INTEGER;
                                        END);
                        END;

        rec_lne_cde   = RECORD
                            CASE BOOLEAN OF
                            TRUE:(arr:data_array);
                            FALSE:(rec:RECORD
                                         no_cde   :INTEGER;
                                         no_lne   :INTEGER;
                                         etat_lne:INTEGER;
                                         no_prod  :INTEGER;
                                         qte_cdee:INTEGER;
                                         qte_liv  :INTEGER;
                                         qte_due  :INTEGER;
                                        END);
```

```
                END;

    rec_ent_exp   = RECORD
                      CASE BOOLEAN OF
                      TRUE:(arr:data_array);
                      FALSE:(rec:RECORD
                                 no_exp      :INTEGER;
                                 no_lne      :INTEGER;
                                 no_cde      :INTEGER;
                                 date        :t_date;
                                 poids_colis:INTEGER;
                                 montant     :INTEGER;
                                END);
                END;

    rec_lne_exp   = RECORD
                      CASE BOOLEAN OF
                      TRUE:(arr:data_array);
                      FALSE:(rec:RECORD
                                 no_exp       :INTEGER;
                                 no_lne       :INTEGER;
                                 no_prod      :INTEGER;
                                 libelle      :STRING[51];
                                 prix_unit    :INTEGER;
                                 poids_unit   :INTEGER;
                                 qte_a_liv    :INTEGER;
                                 motif_refus :STRING[31];
                                END)
                      END;

    rec_prod_stk  = RECORD
                      CASE BOOLEAN OF
                      TRUE:(arr:mess_array);
                      FALSE:(rec:RECORD
                                  no             :INTEGER;
                                  indic_epuist :BOOLEAN;
                                  qte_disp       :INTEGER;
                                  qte_diff       :INTEGER;
                                 END)
                      END;

    lne_liv_frn   = RECORD
                       no_prod:INTEGER;
                       qte_app:INTEGER;
                      END;


VAR
    f_cde           :INTEGER;
    f_exp           :INTEGER;
    f_stk           :INTEGER;
    name_f_cde      :t_filename;
    name_f_exp      :t_filename;
    name_f_stk      :t_filename;
    ce_f_cde        :INTEGER;
    ce_f_exp        :INTEGER;
    ce_f_stk        :INTEGER;

    c_cle_elne_npro : RECORD
                       CASE BOOLEAN OF
                       TRUE:(concat:key);
                       FALSE:(rec:RECORD
                                  etat_lne:INTEGER;
```

```
                                no_pro  :INTEGER;
                               END);
                    END;
      c_cle_nexp_nlne : RECORD
                     CASE BOOLEAN OF
                     TRUE:(concat:key);
                     FALSE:(rec:RECORD
                                no_exp:INTEGER;
                                no_lne:INTEGER;
                               END)
                    END;
      c_cle_ncde_nlne : RECORD
                     CASE BOOLEAN OF
                     TRUE:(concat:key);
                     FALSE:(rec:RECORD
                                no_cde:INTEGER;
                                no_lne:INTEGER;
                               END)
                    END;

      v_cle_npro      : key;
      v_cle_nexp      : key;
      v_cle_ncde      : key;

      answer          : CHAR;
      etat_cde        : CHAR;
      nb_lne_dif      : INTEGER;
      no_lne_liv      : INTEGER;
      nb_lne_liv      : INTEGER;
      qte_res         : INTEGER;

      ent_cde         : rec_ent_cde;
      id_ent_cde      : INTEGER;
      lne_cde         : rec_lne_cde;
      id_lne_cde      : INTEGER;
      ent_exp         : rec_ent_exp;
      id_ent_exp      : INTEGER;
      lne_exp         : rec_lne_exp;
      id_lne_exp      : INTEGER;
      prod_stk        : rec_prod_stk;
      id_prod_stk     : INTEGER;

      m_no_cde        : tm_no_cde;

{**********************}
PROCEDURE initialisation;
{**********************}

  BEGIN  (* procedure initialisation ************************)

    name_f_cde.str := 'cde.indx';
    OPENFILE(f_cde,name_f_cde.arr,ce_f_cde);
    name_f_exp.str := 'exp.indx';
    OPENFILE(f_exp,name_f_exp.arr,ce_f_exp);
    name_f_stk.str := 'stk.indx';
    OPENFILE(f_stk,name_f_stk.arr,ce_f_stk);

  END; (* procedure initialisation ************************)


{********************}
PROCEDURE get_liv_frn;
```

```
***     ***     ***

  BEGIN  (* procedure get_liv_frn ****************************)

     WRITELN; WRITELN; WRITELN;
     no_lne_liv := 1;
     REPEAT
       WRITE('PRODUIT NØ:?');
       READLN(liv_frn[no_lne_liv].no_prod);
       WRITE('QUANTITE:?');
       READLN(liv_frn[no_lne_liv].qte_app);
       IF liv_frn[no_lne_liv].no_prod <> Ø
          THEN BEGIN
                 WRITELN('NØ PROD: ',liv_frn[no_lne_liv].no_prod,' ',
                         'QUANTITE: ',liv_frn[no_lne_liv].qte_app);
                 WRITELN('OK:? (Y/N)');
                 READLN(answer);
                 IF (answer = 'y') OR (answer = 'Y')
                    THEN no_lne_liv := no_lne_liv + 1;
               END;
      UNTIL liv_frn[no_lne_liv].no_prod = Ø;
     nb_lne_liv := no_lne_liv - 1;
   END;  (* procedure get_liv_frn ****************************)


(*********************)
PROCEDURE trt_liv_frn;
(*********************)

   VAR   no_lne_trt : INTEGER;

(*********************)
PROCEDURE trt_lne_liv;
(*********************)


(*********************)
PROCEDURE maj_stk_pos;
(*********************)

   BEGIN  (* procedure maj_stk_pos ****************************)

     v_cle_npro.int := liv_frn[no_lne_trt].no_prod;
     READINDX(f_stk,cle_npro,v_cle_npro.arr,prod_stk.arr,ce_f_stk);
     WITH prod_stk.rec DO
          BEGIN
            IF qte_diff > liv_frn[no_lne_trt].qte_app
               THEN BEGIN
                      qte.res := liv_frn[no_lne_-liv].qte_app;
                      qte_diff:= qte_diff - liv_frn[no_lne_trt].qte_app;
                    END
               ELSE BEGIN
                      qte_res := qte_diff;
                      qte_disp := qte_disp + (liv_frn[no_lne_trt].qte_app
                                  - qte_diff);
                      qte_diff := Ø;
                    END;
          END;
      REWRITE .........

   END;  (* procedure maj_stk_pos ****************************)
```

```
(**********************)
PROCEDURE trt_cde_diff;
(**********************)


(*********************)
PROCEDURE get_fst_cde;
(*********************)

   BEGIN  (* procedure get_fst_cde ****************************)

     (* acquisition de la ligne ed *)
        c_cle_elne_npro.rec.etat_lne := D;
        c_cle_elne_npro.rec.no_prod := liv_frn[no_lne_trt].no_prod;
        READINDX(f_cde,cle_elne_npro,c_cle_elne_npro.concat,
                 lne_cde.arr,ce_f_cde);
        WHILE lne_cde.rec.no_lne = Ø DO
              READNEXT(f_cde,cle_elne_npro,lne_cde.arr,ce_f_cde);

     (* acquisition de l'entete ed *)
        c_cle_ncde_nlne.rec.no_cde := lne_cde.rec.no_cde;
        c_cle_ncde_nlne.rec.no_lne := Ø;
        READINDX(f_cde,cle_ncde_nlne,c_cle_ncde_npro.concat,
                 ent_cde.arr,ce_f_cde);

     (* acquisition de l'entete exp *)
        c_cle_nexp_nlne.rec.no_exp := ent_cde.rec.no_exp;
        c_cle_nexp_nlne.rec.no_lne := Ø;
        READINDX(f_exp,cle_nexp_nlne,c_cle_nexp_nlne.concat,
                 ent_exp.arr,ce_f_exp);

     (* acquisition de la ligne d'expedition *)
        c_cle_nexp_nlne.rec.no_lne := lne_cde.rec.no_lne;
        READINDX(f_exp,cle_nexp_nlne,c_cle_nexp_nlne.concat,
                 lne_exp.arr,ce_f_exp);

   END;  (* procedure get_fst_cde ******************************)


(**********************)
PROCEDURE trt_lne_diff;
(**********************)

   VAR   nvelle_qte_liv : INTEGER;


(******************)
PROCEDURE stk_suff;
(******************)

   BEGIN  (* procedure stk_suff ********************************)

     (* maj de la ligne commande et enregist *)
        WITH lne_cde.rec DO
             BEGIN
                etat_lne := A;
                nvelle_qte_liv := qte_due;
                qte_liv := qte_liv + nvelle_qte_liv;
                qte_due := Ø;
             END;
        REWRITE(.....)
```

```
    (* maj ent_cde et enregistrement *)
       WITH ent_cde.rec DO
            BEGIN
               nb_lne_dif := nb_lne_dif - 1;
               IF nb_lne_dif = Ø
                  THEN etat_cde := A;
             END;
       REWRITE(......)

    (* maj de la ligne expedition et enregistrement *)
       WITH lne_exp.rec DO
            BEGIN
               qte_a_liv := qte_a_liv + nvelle_qte_liv;
               motif_refus := '';
            END;
       REWRITE(.........)

    (* maj ent_exp et enregistrement *)
       WITH ent_exp.rec DO
            BEGIN
               poids_colis := poids_colis + (nvelle_qte_liv *
                              lne_cde.rec.poids_unit);
               montant := montant + (nvelle_qte_liv *
                          lne_cde.rec.prix_unit);
            END;
       REWRITE(...........)

    (* maj qte_res *)
       qte_res := qte_res - nvelle_qte_liv ;

  END;  (* procedure stk_suff ******************************)


(******************)
PROCEDURE stk_insuff;
(******************)

  BEGIN  (* procedure stk_insuff*****************************)

     (* maj ligne commande et enregistrement *)
       WITH lne_cde.rec DO
            BEGIN
                nvelle_qte_liv := qte_res;
                qte_liv := qte_liv + nvelle_qte_liv;
                qte_due := qte_due - nvelle_qte_liv;
            END;
       REWRITE(...........)

     (* l'etat de la comande reste inchange *)
       REWRITE(..............)

     (* maj ligne expedition et enregistrement *)
       WITH lne_exp.rec DO
            BEGIN
                qte_a_liv := qte_a_liv + nvelle_qte_liv;
            END;
       REWRITE(.............)

     (* maj et enregistrement de ent_exp *)
       WITH ent_exp.rec DO
            BEGIN
                poids_colis := poids_colis + (nvelle_qte_liv *
```

```
                                lne_cde.rec.poids_unit);
                        montant := montant + (nvelle_qte_liv *
                                lne_cde.rec.prix_unit);
                END;
            REWRITE(.....................)

        (* maj qte_res *)
            qte_res := Ø;

    END;  (* procedure stk_insuff ****************************)



(* instructions de la procedure trt_cde_diff *)

    BEGIN  (* procedure trt_lne_diff ***************************)

      IF qte_res >= lne_cde.rec.qte_due
         THEN stk_suff
         ELSE stk_-insuff;

    END;  (* procedure trt_lne_diff ***************************)


(********************)
PROCEDURE trt_nxt_cde;
(********************)

     BEGIN  (* procedure trt_nxt_cde ***************************)

       (* get next cde_diff *)

       (* acquisition de la ligne *)
          READNEXT(f_cde,cle_elne_npro,lne_cde.arr,ce_f_cde);
          WHILE lne_cde.rec.no_lne = Ø DO
                READNEXT(f_cde,cle_elne_npro.concat,lne_cde.arr,
                         ce_f_cde);

        (* acquisition de l'entete *)
           c_cle_ncde_nlne.rec.no_cde := lne_cde.rec.no_cde;
           c_cle_ncde_nlne.rec.no_lne := Ø;
           READINDX(f_cde,cle_ncde_nlne,c_cle_ncde_nlne.concat,
                    ent_exp.arr,ce_f_cde);

        (* acquisition de l'expedition *)

        (* acquisition de l'entete expedition *)
           c_cle_nexp_nlne.rec.no_exp := lne_cde.rec.no_exp;
           c_cle_nexp_nlne.rec.no_lne := Ø;
           READINDX(f_exp,cle_nexp_nlne,c_cle_nexp_nlne.arr,
                    ent_exp.arr,ce_f_exp);

        (* acquisition de la ligne d'expedition concernee *)
           c_cle_nexp_nlne.rec.no_lne := lne_cde.rec.no_lne;
           READINDX(f_exp,cle_nexp_nlne,c_cle_nexp_nlne.concat,
                    lne_exp.arr,ce_f_exp);

     END;  (* procedure trt_nxt_cde ***************************)


(* instructions de la procedure trt_cde_diff *)
```

```
    BEGIN  (* procedure trt_cde_diff ***************************)

      get_fst_cde;
      WHILE qte_res <> Ø DO
            BEGIN
               trt_cde_diff;
               SENDMESS(.....);
               RECVMESS(ack,....);
               IF qte_res <> Ø
                  THEN trt_nxt_cde
            END;

    END;  (* procedure trt_cde_diff ***************************)



(* instructions de la procedure trt_lne_liv *)

    BEGIN  (* procedure trt_lne_liv ***************************)
      {
        maj_stk_pos;
        IF prod_stk.qte_res <> Ø
           THEN trt_cde_diff;

     END;  (* procedure trt_lne_liv ***************************)


(* instructions de la procedure trt_liv_frn *)

    BEGIN  (* procedure trt_liv_frn ***************************)

       FOR no_lne_trt = 1 TO nb_lne_liv DO
           trt_lne_liv;

    END;  (* procedure trt_liv_frn ***************************)



(********************)
PROCEDURE terminaison;
(********************)

    BEGIN  (* procedure terminaison ***************************)

       CLOSEFILE(f_stk,ce_f_stk);
       CLOSEFILE(f_cde,ce_f_cde);
       CLOSEFILE(f_exp,ce_f_exp);

    END;  (* procedure terminaison ***************************)



(* instructions du programme principal *)

    BEGIN  (* programme principal ***************************)

       initialisation;
       get_liv_frn;
```

```
    trt_liv_frn;
    terminaison;

END.  (* programme principal ********************************)
```

```
PROGRAM  trt_bl;

CONST    cle_ncde_nlne = Ø;  (* cles fichier cde *)
         cle_elne_npro = 1;
         cle_nexp      = 2;

         cle_npro      = Ø;  (* cle fichier stk *)

         cle_nser_nref = Ø;  (* cle fichier bon_-liv *)


TYPE

         t_date          = RECORD
                             jour  :  1..31;
                             mois  :  1..12;
                             annee : 8Ø..99;
                           END;

         rec_bon_liv     = RECORD
                            CASE BOOLEAN OF
                            TRUE:(arr:data_array);
                            FALSE:(rec:RECORD
                                         no_ser :INTEGER;
                                         no_ref :INTEGER;
                                         no_bl  :INTEGER;
                                         date   :t_date;
                                         no_clt :INTEGER;
                                         nom    :INTEGER;
                                         prenom :STRING[15];
                                         adresse:RECORD
                                           rue         :STRING[25];
                                           no_rue      :STRING[5];
                                           localite    :STRING[25];
                                           code_postal:STRING[5];
                                                END;
                                         lne_bl:ARRAY [1..1Ø] OF RECORD
                                           no_prod     :INTEGER;
                                           libelle     :STRING[51];
                                           qte_liv     :INTEGER;
                                           prix_unit   :INTEGER;
                                           montant     :INTEGER;
                                           motif_refus:STRING[31];
                                                                 END;
                                         frais_exp:INTEGER;
                                         total    :INTEGER;
                                        END);
                           END;


         tm_date         = RECORD
                            CASE BOOLEAN OF
                            TRUE :(arr:mess_array);
                            FALSE:(mes:RECORD
                                         jour   :  1..31;
                                         mois   :  1..12;
                                         annee  : 8Ø..99;
                                        END)
                           END;
```

```
    rec_ent_cde     = RECORD
                       CASE BOOLEAN OF
                       TRUE:(arr:data_array);
                       FALSE:(rec:RECORD
                                  no_cde     :INTEGER;
                                  no_lne     :INTEGER;
                                  etat_lne   :INTEGER;
                                  nb_lne_dif:INTEGER;
                                  date       :t_date;
                                  no_clt     :INTEGER;
                                  nom        :STRING[31];
                                  prenom     :STRING[15];
                                  adresse    :RECORD
                                   rue          :STRING[25];
                                   no_rue       :STRING[5];
                                   localite     :STRING[25];
                                   code_postal:STRING[25];
                                                END;
                                  ind_1_liv  :BOOLEAN;
                                  no_exp     :INTEGER;
                                 END);
                      END;

    rec_lne_cde     = RECORD
                       CASE BOOLEAN OF
                       TRUE:(arr:data_array);
                       FALSE:(rec:RECORD
                                  no_cde   :INTEGER;
                                  no_lne   :INTEGER;
                                  etat_lne:INTEGER;
                                  no_prod  :INTEGER;
                                  qte_cdee:INTEGER;
                                  qte_liv  :INTEGER;
                                  qte_due  :INTEGER;
                                 END);
                      END;

    rec_prod_stk    = RECORD
                        CASE BOOLEAN OF
                        TRUE:(arr:data_array);
                        FALSE:(rec:RECORD
                                    no           :INTEGER;
                                    indic_epuist:INTEGER;
                                    qte_disp     :INTEGER;
                                    qte_diff     :STRING[51];
                                   END)
                       END;


VAR
    f_cde           :INTEGER;
    f_stk           :INTEGER;
    f_bon_liv       :INTEGER;
    name_f_cde      :t_filename;
    name_f_stk      :t_filename;
    name_f_bon_liv  :t_filename;
    ce_f_cde        :INTEGER;
    ce_f_stk        :INTEGER;
    ce_f_bon_liv    :INTEGER;

    c_cle_nser_nref : RECORD
                       CASE BOOLEAN OF
```

```
                        TRUE:(concat:key);
                        FALSE:(rec:RECORD
                                    no_serie:INTEGER;
                                    no_ref  :INTEGER;
                                   END);
                       END;
      c_cle_ncde_nlne : RECORD
                        CASE BOOLEAN OF
                        TRUE:(concat:key);
                        FALSE:(rec:RECORD
                                    no_cde:INTEGER;
                                    no_lne:INTEGER;
                                   END)
                       END;

      v_cle_npro       : key;

      answer1          : CHAR;
      leg_answer1      : BOOLEAN;
      order            : STRING[1];
      leg_order        : BOOLEAN;
      etat_cde         : CHAR;

      ent_cde          :rec_ent_cde;
      id_ent_cde       :INTEGER;
      lne_cde          :rec_lne_cde;
      id_lne_cde       :INTEGER;
      prod_stk         :rec_prod_stk;
      id_prod_STK      :INTEGER;
      bon_liv          :rec_bon_liv;
      id_bon_liv       :INTEGER;
      bl_lu, bl_corr   : rec_bon_liv;

(**************)
PROCEDURE menu1;
(**************)


(******************)
PROCEDURE get_order;
(******************)

  BEGIN  (* procedure get_order ******************************)

    REPEAT BEGIN
             GOTOXY (0,0);
             READLN (order);
             IF pos(order,valid_order) = 0
               THEN BEGIN
                      GOTOXY (20,23);
                      WRITELN ('ILLEGAL COMMAND');
                      leg_order := false;
                    END
               ELSE leg_order := true;
           END
      UNTIL leg_order;

  END;  (* procedure get_order ******************************)


(***************)
PROCEDURE trt_bl;
(***************)
```

```
VAR  yes_no : CHAR;

(*****************************************)
PROCEDURE aff_bon_liv ( bon_liv : rec_bon_liv);
(*****************************************)

VAR  ind_y, h, min : INTEGER;

BEGIN  (* procedure aff_bon_liv ***************************)

  WITH bon_liv.rec DO
       BEGIN
        GOTOXY( 2, 1); WRITE('******************************');
        GOTOXY(31, 1); WRITE('******************************');
        FOR ind_y := 2 TO 22 DO
            BEGIN
                 GOTOXY( Ø,ind_y); WRITE('*');
                 GOTOXY(69,ind_y); WRITE('*');
            END;
        GOTOXY( 2,22); WRITE('******************************');
        GOTOXY(31,22); WRITE('******************************');
        GOTOXY(4Ø, 2): WRITE('VOTRE COMMANDE DU:   /  /  ');
        GOTOXY(59, 2); WRITE(date.jour);
        GOTOXY(62, 2); WRITE(date.mois);
        GOTOXY(65, 2); WRITE(date.annee);
        GOTOXY(22, 5); WRITE('M,Mme,Melle: ',nom,'  ',prenom);
        GOTOXY( 4, 9); WRITE('NO PRO*LIBELLE  *QTE-LIV*');
        GOTOXY(4Ø, 9); WRITE('PR-UNIT*MONTANT*JUSTIF-N-LIV);
        GOTOXY( 4,1Ø); WRITE('**************************');
        GOTOXY(31,1Ø); WRITE('**************************');
        no_lne := 1;
        FOR ind_y := 11 TO 15 DO
            BEGIN
                 GOTOXY( 4,ind_y);
                 WRITE(lne_bl[no_ligne].no_prod:6,'*');
                 GOTOXY(21,ind_y);
                 FOR h := 1 TO 1Ø DO
                     WRITE(lne_bl[no_ligne].libelle[h]);
                 WRITE('*');
                 GOTOXY(32,ind_y);
                 WRITE(lne_bl[no_ligne].qte_liv:6,' * ');
                 GOTOXY(4Ø,ind_y);
                 WRITE(lne_bl[no_ligne].prix_unit:6,' *');
                 GOTOXY(48,ind_y);
                 WRITE(lne_bl[no_ligne].montant:6,' *');
                 GOTOXY(56,ind_y);
                 IF LENGTH(lne_bl[no_ligne].motif_refus) < 1Ø
                    THEN min := .....
                    ELSE min := 12;
                 FOR h := 1 TO min DO
                     WRITE(lne_bl[no_ligne].motif_refus[h]);
                 FOR h := min TO 12 DO
                     WRITE('');
                 no_ligne := no_ligne + 1;
            END;
        GOTOXY( 4,16); WRITE('**************************');
        GOTOXY(31,16); WRITE('**************************');
        FOR ind_y := 17 TO 21 DO
            BEGIN
                 GOTOXY(47,ind_y);
                 WRITE('*         *');
            END;
```

```
        GOTOXY(26,18); WRITE('FRAIS D''EXPEDITION    ');
        GOTOXY(48,18); WRITE(frais_exp);
        GOTOXY(26,20); WRITE('TOTAL A PAYER          ');
        GOTOXY(48,20); WRITE(total);
        GOTOXY( 2,22); WRITE('*****************************');
        GOTOXY(31,22); WRITE('*****************************');

END;  (* procedure aff_bon_liv ****************************)


(*********************)
PROCEDURE correc_syst;
(*********************)

VAR  no_ligne, qte_livrable, dfce_qte : INTEGER;
     iere_liv, bl_valid               : BOOLEAN;

(*********************)
PROCEDURE enr_correct;
(*********************)

  BEGIN  (* procedure enr_correct ***************************)

    (* maj bon de livraison *)
        IF bl_valid
           THEN REWRITE .......
           ELSE REWRITE .......;

    (* maj commande *)
        c_cle_ncde_nlne.rec.no_cde := bl_corr.rec.no_cde;
        c_cle_ncde_nlne.rec.no_lne := 0;
        READINDX (f_cde,cle_ncde_nlne,c_cle_ncde_nlne.concat,
                  ent_cde.arr,ce_f_cde);
        c_cle_ncde_nlne.rec.no_cde := bl_corr.rec.no_ref;
        c_cle_ncde_nlne.rec.no_lne := no_ligne;
        READINDX (f_cde,cle_ncde_nlne,c_cde_ncde_nlne.concat,
                  lne_cde.arr,ce_f_cde);
        WITH lne_cde.rec DO
             BEGIN
               qte_liv := qte_liv - dfce_qte;
               qte_due := qte_cdee - qte_liv;
               IF etat_lne = a THEN
                  ent_cde.nb_lne_dif := ent_cde.nb_lne_dif + 1;
             END;
        etat_cde := d;
        REWRITE .........;
        REWRITE .........;

    (* maj stock *)

        v_cle_npro.int := bl_corr.rec.lne_bl[no_ligne].no_prod;
        READINDX (f_stk,cle_npro,v_cle_npro,prod_stock.arr,
                  ce_f_stk);
        WITH prod_stk.rec DO
             BEGIN
               qte_disp := qte_disp - dfce_qte;
               qte_dif  := qte_dif  + dfce_qte;
             END;
        REWRITE .........;

  END;  (* procedure enr_correct ***************************)
```

```
(* instructions de la procedure correc_syst *)

BEGIN  (* procedure correc_syst ***************************)

  GOTOXY(   ,   );
  WRITE('NUMERO DE LA LIGNE A MODIFIER ?');
  READLN(no_ligne);
  WRITE('QUANTITE LIVRABLE');
  READLN(qte_livrable);
  PAGE(output);
  bl_corr.rec := bl_lu.rec;

  (* maj ligne bl *)
       dfce_qte := bl_lu.rec.lne_bl[no_ligne].qte_liv
                   - qte_livrable;
       WITH bl_corr.rec.lne_bl[no_ligne] DO
            BEGIN
              qte_liv      := qte_livrable;
              montant      := prix_unit * qte_liv;
              motif_refus := 'reste plus tard';
            END;

  (* maj fin bl *)
       WITH bl_corr.rec DO
            BEGIN
              total := total -
                       (bl_corr.rec.lne_bl[no_ligne].prix_unit
                        * dfce_qte);
              IF total - frais_exp <> Ø
                 THEN bl_valid := true
                 ELSE IF frais_exp = Ø
                           THEN bl_valid := false
                           ELSE BEGIN
                                  frais_exp := Ø;
                                  iere_liv  := true;
                                  bl_valid  := true;
                                END;
            END;
       aff_bl (bl_corr);
       GOTOXY(   ,   );
       WRITE('OK ? ( Y/N )');
       READLN(yes_no);
       IF (yes_no = 'Y') OR (yes_no = 'y')
          THEN enr_correct;

END;  (* procedure correc_syst ***************************)


(* instructions de la procedure trt_bl *)

BEGIN  (* procedure trt_bl ********************************)

  GOTOXY(   ,   );
  WRITE('NUMERO DE COMMANDE: ?');
  READLN(c_cle_nser_nref.rec.no_cde);
  READINX(f_bon_liv,cle_nser_nref,c_cle_nser_nref.arr,ce_f_bon_liv);
  IF ce_f_bon_liv <> Ø
     THEN BEGIN
            GOTOXY(   ,   );
            WRITE('NUMERO DE COMMANDE ERRONE');
            get_order ('ns');
            PAGE (output);
          END
```

```
            ELSE BEGIN
                   aff_bon_liv (bl_lu);
                   get_order ('ncs');
                   PAGE (output);
                 END;
      IF order = 'c'
         THEN correc_syst
         ELSE BEGIN
                GOTOXY(  ,  );
                WRITE('FACTURATION: ? ( Y/N )');
                READLN(yes_no);
                IF (yes_no = 'Y') OR (yes_no = 'y')
                   THEN facturation
                   ELSE EXIT menu1;
              END;

  END;  (* procedure trt_bl ********************************)


(* instructions de la procedure menu1 *)

  BEGIN  (* procedure menu1 ********************************)

    GOTOXY(  ,  );
    WRITE('VOTRE CHOIX: ?');
    GOTOXY(  ,  );
    WRITELN('1. TRAITEMENT BONS LIVRAISONS');
    WRITELN('2. FACTURATION');
    WRITELN('3. STOP');
    REPEAT
      GOTOXY(  ,  );
      READLN(answer1);
      PAGE (output);
      IF (answer1 <> '1') AND (answer1 <> '2') AND (answer3 <> '3')
         THEN BEGIN
                GOTOXY(  ,  );
                WRITE('ILLEGAL COMMAND !');
                leg_answer1 := false;
              END
         ELSE leg_answer1 := true;
     UNTIL leg_answer1;
    CASE leg_answer1 OF
      '1': REPEAT trt_bl UNTIL order <> 'N';
      '2': facturation;
     END;

  END;  (* procedure menu1 ********************************)


(* instructions du programme principal *)

  BEGIN  (* programme principal ********************************)

    (* initialisations *)
      name_f_bon_liv.str := 'bonliv.indx';
      OPENFILE(f_bon_liv,name_f_bon_liv.arr,ce_f_bon_liv);
      name_f_cde.str := 'cde.indx';
      OPENFILE(f_cde,name_f_cde.arr,ce_f_cde);
      name_f_stk.str := 'stk.indx';
      OPENFILE(f_stk,name_f_stk.arr,ce_f_stk);
      GOTOXY(  ,  );
      WRITE('NUMERO DE SERIE: ?');
      READLN(c_cde_nser_nref.rec.no_ser);
```

```
  (* fermeture des fichiers *)
     CLOSEFILE(f_bon_liv,ce_f_bon_liv);
     CLOSEFILE(f_cde,ce_f_cde);
     CLOSEFILE(f_stk,ce_f_stk);

END.  (* programme principal ********************************)
```

BUMP
0 0 3 4 4 2 2 8 9
*FM B16/1981/14/2