MASTER EN SCIENCES INFORMATIQUES

Mémo-progrès
proposition d'adaptation au temps réel

Timmermans, Luc

Award date:
1979

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX
Institut d'informatique

NAMUR

MEMO-PROGES

PROPOSITION D'ADAPTATION AU TEMPS REEL

LUC TIMMERMANS
Promoteur: Mr. A. CLARINVAL

Mémoire presenté en vue de
l'obtention du titre de licencié
et maître en Informatique.

NAMUR 1979

FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX
Institut d'informatique

NAMUR

MEMO - PROGES

PROPOSITION D'ADAPTATION AU TEMPS REEL

LUC TIMMERMANS
Promoteur: Mr. A. CLARINVAL

Mémoire presenté en vue de
l'obtention du titre de licencié
et maître en Informatique.

NAMUR 1979

Je remercie Monsieur A. CLARINVAL pour ses innombrables conseils et encouragements, ainsi que mes parents qui m'ont toujours aidé lors de mes études.
Egalement un grand merci à Viviane.

TABLE DES MATIERES

.../...

INTRODUCTION : OBJET DE L'ETUDE

MEMO-PROGES [R 5] est une méthode de programmation modulaire se voulant adaptée à l'informatique administrative.

Elle est actuellement conçue et utilisée pour l'exploitation en temps différé. L'objet de la présente étude est de proposer une adaptation de la méthode à certaines modalités du temps réel.

Première partie :

La méthode MEMO-PROGES

Chapitre 1. : Introduction

1.A. Principes de MEMO-PROGES.

A. MEMO-PROGES est tout d'abord une méthode modulaire.
De ce point de vue, sa particularité réside dans la banalisation stricte des interfaces.
La méthode définit cet interface banalisé comme un fichier dit "virtuel"; chaque appel de module ou "fonction élémentaire" transmet en guise d'unique argument un article de fichier, le fichier virtuel est l'ensemble des articles transmis par les exécutions successives d'un tel appel.
La méthode implique l'existence de modules spécialisés d'accès aux fichiers réels (lecture, écriture), appelés fonctions terminales ; tous les autres modules sont des fonctions intermédiaires exclusivement définies sur des fichiers virtuels.

B. Le second principe de MEMO-PROGES est le traitement déclaratif de la dynamique répétitive des programmes.
A cette fin, la méthode distingue le "processus" (occurrence d'action) et la "session d'activité" (ensemble constitué de la répétition d'un processus.)
L'ordonnancement dynamique d'une session d'activité est assuré implicitement par des algorithmes préprogrammés contrôlant l'itération dans chaque module.
Quant aux processus, ils sont "quantifiés" par la déclaration de la quantité d'information d'entrée qu'ils doivent traiter pour produire un résultat.

C. Coordination modulaire et contrôle de l'itération reposent sur l'analyse de paramètres globaux banalisés, (appelés paramètres d'orientation) fournis par l'instruction d'appel normalisée. Cette dernière est de la forme CALL 'module' USING paramètres, argument.

.../...

1.B. Les outils de MEMO-PROGES.

A. Le système objet de la méthode.

Le système objet de la méthode est un système préprogrammé prenant en charge la concrétisation des concepts opératoires de la méthode, soit :

- le contrôle dynamique des modules et leur coordination.
- le contrôle dynamique global de l'unité de traitement.
- les fonctions terminales de gestion des entrées/sorties.

B. Les langages.

Pour l'expression de la structure logique des traitements, MEMO-PROGES utilise un langage de description des fonctions intermédiaires (c.à.d., les fonctions définies sur des fichiers virtuels).
Ce langage (LDF) permet principalement :

- la spécification des interfaces, c'est-à-dire la définition des fichiers virtuels et l'accès à ces fichiers.
- la quantification des processus de traitement.

MEMO-PROGES propose en outre un langage de manipulation des mécanismes de contrôle (LCM), permettant la coordination de toutes espèces de modules, fonctions intermédiaires et autres.

C. Objet de l'étude d'adaptation.

Les hypothèses de travail que nous avons retenues pour notre étude d'adaptation de la méthode au temps réel sont les suivantes :

"L'introduction du temps réel, ne doit pas modifier les lois de la modularité : la nécessité et les propriétés d'un interface banalisé, les critères d'homogénéité des fonctions élémentaires, les règles d'enchaînement des modules.
Elle est susceptible de modifier la gestion dynamique des programmes".

L'objet de cette étude est de proposer une extension des langages existants et d'adapter en conséquence le système objet qui les supporte.

.../...

2. LA DYNAMIQUE DES PROGRAMMES DANS MEMO-PROGES.

2.A. Principe.

1. Concepts.

Pour étudier la dynamique des programmes dans MEMO-PROGES, il nous faut expliciter les concepts suivants :

a. le "processus" : le processus est une occurrence d'action; action produite par la rencontre effective à un instant t d'un vecteur d'arguments et d'un opérateur actif.

b. La "session" : la session est l'ensemble constitué par la répétition d'un ou plusieurs processus.

L'utilité de distinguer la "session" s'explique par la nécessité d'initialiser et clôturer la session d'activité de certaines opérations (qualifiées d'itératives) par l'exécution de processus spécifiques. (ex. : OPEN/CLOSE).

2. Itérativité des fonctions.

Le souci d'uniformiser les propriétés dynamiques des fonctions mène à considerer que :

"Toute fonction est susceptible de contenir des opérations itératives."

Ainsi seront prévues dans toute fonction, trois sections distinctes (certaines pouvant être vides) pour la programmation des différents processus susceptibles d'être exécutés :

- initialisation de la session d'activité (INIT).
- processus normaux de l'itération (CORPUS).
- terminaison de la session d'activité (TERM).

En outre, tout module itératif peut contenir les sections suivantes :

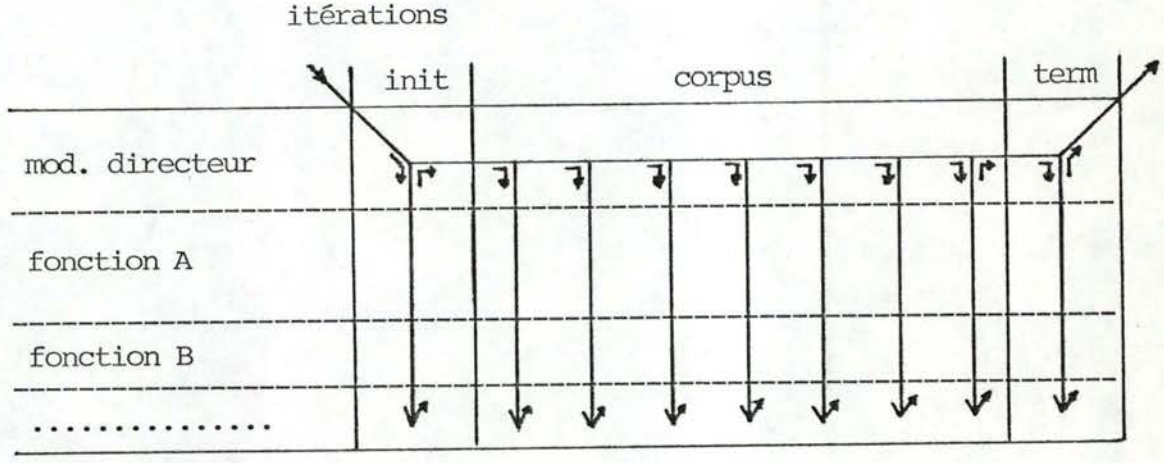
- CØNSTANT : traitement particulier à la première itération de l'unité de traitement.
- PARAM : réception des paramètres.

L'algorithme de choix peut être préprogrammé une fois pour toutes.

.../...

3. Contrôle dynamique de l'unité de traitement. Module directeur.

Tout appel d'une fonction B programmé dans une fonction A est répété en phases d'initialisation et de terminaison de la fonction A.
De la sorte, initialisation et terminaison se répercutent au même moment de fonction à fonction. On distingue donc globalement trois moments (INIT, CORPUS, TERM) dans une unité de traitement.
Pour traiter sur le mode déclaratif la dynamique répétitive des programmes, la méthode conçoit l'opérateur programmé de toute fonction sous la forme d'un module appelé; auquel sont communiquées, par le biais des indicateurs banalisés véhiculés par l'instruction d'appel, les décisions relatives à cette dynamique. Il faut donc dans toute unité de traitement un module racine, à l'origine de tous les appels, à qui est déléguée la seule tâche d'instaurer la dynamique en déterminant la valeur des indicateurs de contrôle.
A ce module, nous donnons le nom de module directeur de l'unité de traitement. La figure ci-dessous schématise l'organisation dynamique de l'unité de traitement.



2.B. Mécanismes du contrôle dynamique.

1. Les indicateurs globaux banalisés.

Les indicateurs banalisés permettant le contrôle dynamique de l'unité de traitement sont contenus dans l'article U-EXT transmis à tout module par l'instruction d'appel normalisée.

.../...

L'article U-EXT contient notamment les informations suivantes :

- U-SW1 : indicateur d'appel.
- U-SW2 : indicateur de retour.

Les valeurs principales de ces indicateurs sont :

- indicateur d'appel (SW1)

-1 = param
1 = init
2...6 = corpus + constant
9...99= term

- indicateur de retour (SW2)

au moment de l'appel (valeurs nulles) : comme SW1
au moment du retour : 7 = end-of-group;
fin du traitement d'un groupe de données.
8 = error; erreur :
fin du processus en cours.
9 = end;
fin normale de la session d'activité.
16 = fin anormale de la session.

Note : Ces paramètres globaux sont manipulés par les instructions des langages associés à la méthode.

2. Algorithme de choix du processus à exécuter dans un module itératif.

Tout module itératif comporte d'office un certain nombre de sections de traitements spécialisées :

- CORPUS SECTION : traitement normal d'itération;
- CONSTANT SECTION : traitement particulier à la première itération de l'unité de traitement.
- INIT SECTION : initialisation des opérations itératives internes.
- TERM SECTION : terminaison des opérations itératives internes.
- PARAM SECTION : réception des paramètres.

Le retour de chacune de ces sections au système de contrôle est implicite.
L'algorithme banalisé, pré-programmé, U-LDF ou U-LCM,du contrôle de l'itération dans un module sera incorporé par le compilateur en guise de première section exécutable dans le texte objet de ce module,

.../...

il enchaînera implicitement les processus programmés.
Cet algorithme se base sur les valeurs des indicateurs SW1, SW2 et les valeurs de certains indicateurs locaux. Les indicateurs locaux sont contenus dans l'article U-INT, incorporé par le compilateur en WORKING-STORAGE-SECTION des modules appelés.

Le schéma d'un module itératif traduit en COBOL se présente de la façon suivante :

(a) PROGRAM-ID. module.
    ......
    WORKING-STORAGE SECTION.
(b) COPY U-INT.
    ......
    LINKAGE SECTION.
(c) COPY U-EXT.
    ......
(d) PROCEDURE DIVISION USING U-EXT liste-d-arguments.
    DECLARATIVES.
    ......
    END DECLARATIVES.
(e) COPY U-LDF.
(f) U-CONSTANT SECTION. (......)
(g) U-LAST SECTION. (......)
(h) U-INIT SECTION. (......)
(i) U-TERM SECTION. (......)
(j) U-PARAM SECTION. (......)
(k) U-ICALL SECTION. (......)

} traitements rédigés dans le le programme source.

(a) nom du module.
(b) copie des indicateurs de contrôle internes.
(c) copie des indicateurs de contrôle globaux transmis par l'appel.
(d) définition du point d'entrée.
(e) copie de l'algorithme pré-programmé du contrôle de l'itération.
(k) initialisation et terminaison implicites des instructions d'appel itératives.

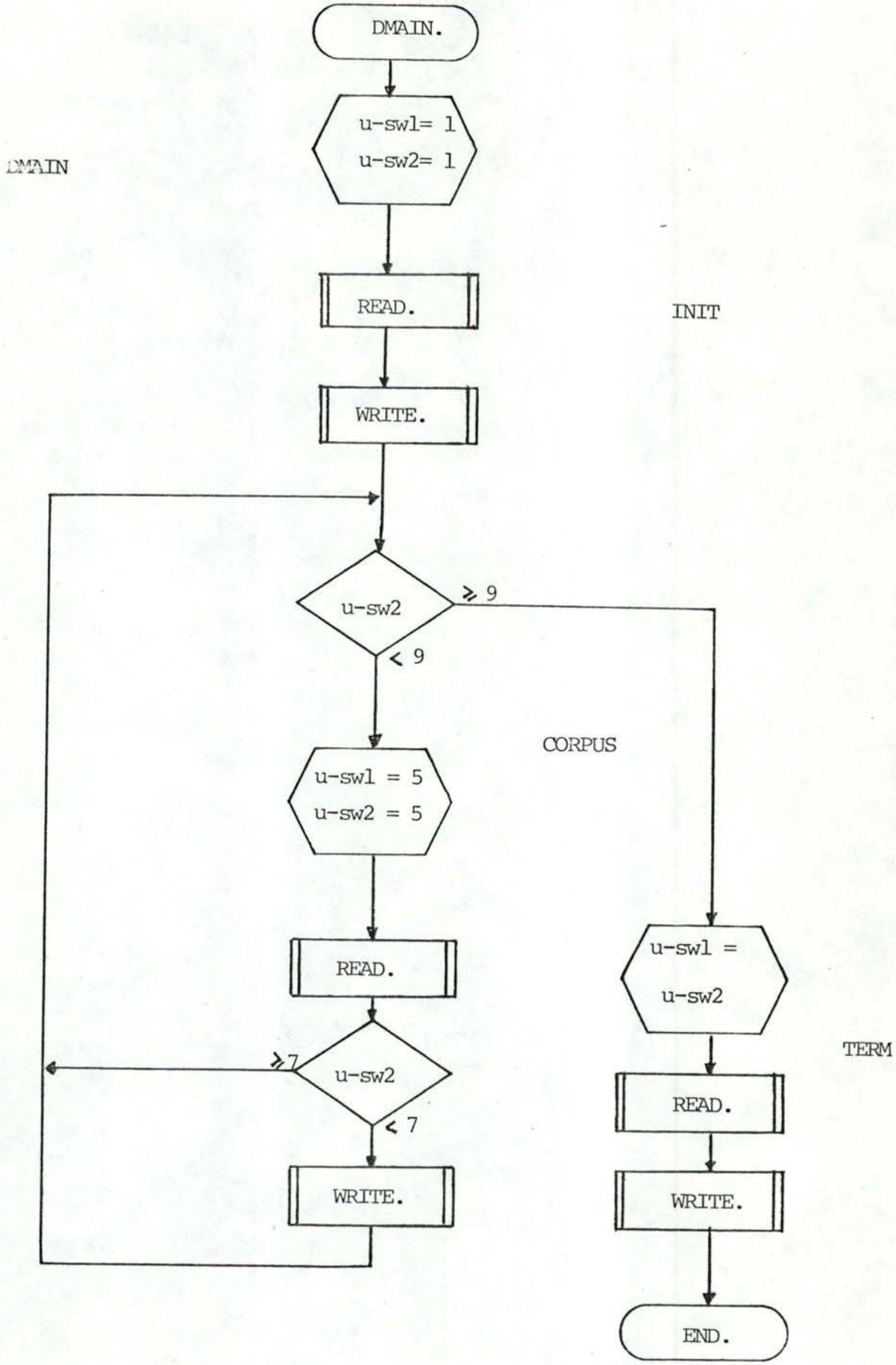
.../...

3. Module directeur.

Le module directeur ou module racine de l'unité de traitement s'occupe seulement du contrôle dynamique de cette dernière; c'est-à-dire qu'il n'effectue aucune transformation des données.
Pour pouvoir effectuer ce contrôle, le module directeur prend en compte les valeurs des indicateurs banalisés SW1, SW2.
Après chaque "ordre" donné par le module directeur à l'aide de l'indicateur d'appel (SW1), l'indicateur de retour (SW2) rend compte au module directeur de la manière dont cet "ordre" a été exécuté par le ou les modules de l'unité de traitement.
Les modules de l'unité de traitement constituent essentiellement deux ensembles; l'ensemble des modules d'acquisition des données d'entrée et l'ensemble des modules de production des résultats.
Le module directeur se superpose à ces deux ensembles et réalise la coordination entre eux. La position hiérarchique du module directeur explique le fait que ce dernier ne réalise que deux appels; d'une part un appel se propageant à travers l'ensemble des modules d'acquisition de données (CALL READ) et d'autre part un appel se propageant à travers l'ensemble des modules de production des résultats (CALL WRITE).

Voici le schéma de fonctionnement du module directeur.

.../...


DMAIN.
u-sw1= 1
u-sw2= 1
DMAIN
READ.
INIT
WRITE.
u-sw2
≥ 9
< 9
CORPUS
u-sw1 = 5
u-sw2 = 5
READ.
u-sw1 =
u-sw2
TERM
u-sw2
≥ 7
< 7
READ.
WRITE.
WRITE.
END.

Deuxième partie :

Adaptation au temps réel

Chap. 1. Modalités du temps réel

Chap. 2. Extension des langages

Chap. 3. Adaptation du système objet

Chapitre 1. : Modalités du temps réel

1.A. Introduction

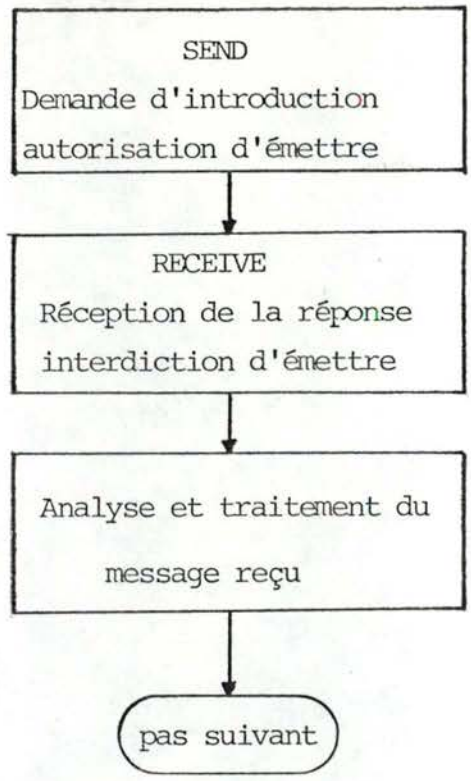
Lorsque nous parlons de "modalités du temps réel", nous considérons ces modalités dans le cadre de l'adaptation de la méthode MEMO-PROGES et non dans le cadre de développement d'un système temps réel complet. C'est la raison pourquoi nous n'envisagerons pas, parmi d'autres, les problèmes de sécurité (check-point/restart) ou d'urgence.
Nous allons plutôt étudier les problèmes de l'interaction et de la multiplicité des utilisateurs, car il nous semble que ces problèmes ont un impact plus important sur la conception des programmes d'application.
Une des caractéristiques de MEMO-PROGES est de produire des descriptions programmées indépendantes de la considération des ressources matérielles allouées à la réalisation des programmes. Notre point de vue respectera cette optique.

1.B. Aspect interactif

1. Segmentation du dialogue.

Nous étudierons l'interaction dans le cadre des programmes conversationnels.
L'initiative du dialogue -après le choix initial du programme par l'utilisateur- appartient à ce programme, qui enregistre et traite les réponses de l'utilisateur, réponses canalisées par les demandes émanant du programme lui-même.
Le dialogue, l'échange d'information entre la machine et un utilisateur sera structuré en "pas" successifs.
Un "pas" de dialogue peut être schématisé de la façon suivante :

.../...



a. le programme envoie d'abord une demande à l'utilisateur. Le terminal reçoit en plus du message, une autorisation d'émettre qui doit lui permettre de répondre.

b. la deuxième étape consiste à recevoir la réponse de l'utilisateur. Dès la réception de la réponse le programme interdit au terminal d'émettre, ceci pour éviter toute initiative de l'utilisateur.

c. la troisième étape comporte l'analyse et le traitement du message reçu : des opérations de contrôle, le traitement proprement dit en vue de l'élaboration du message de retour et le déclenchement d'opérations annexes.

Remarques :

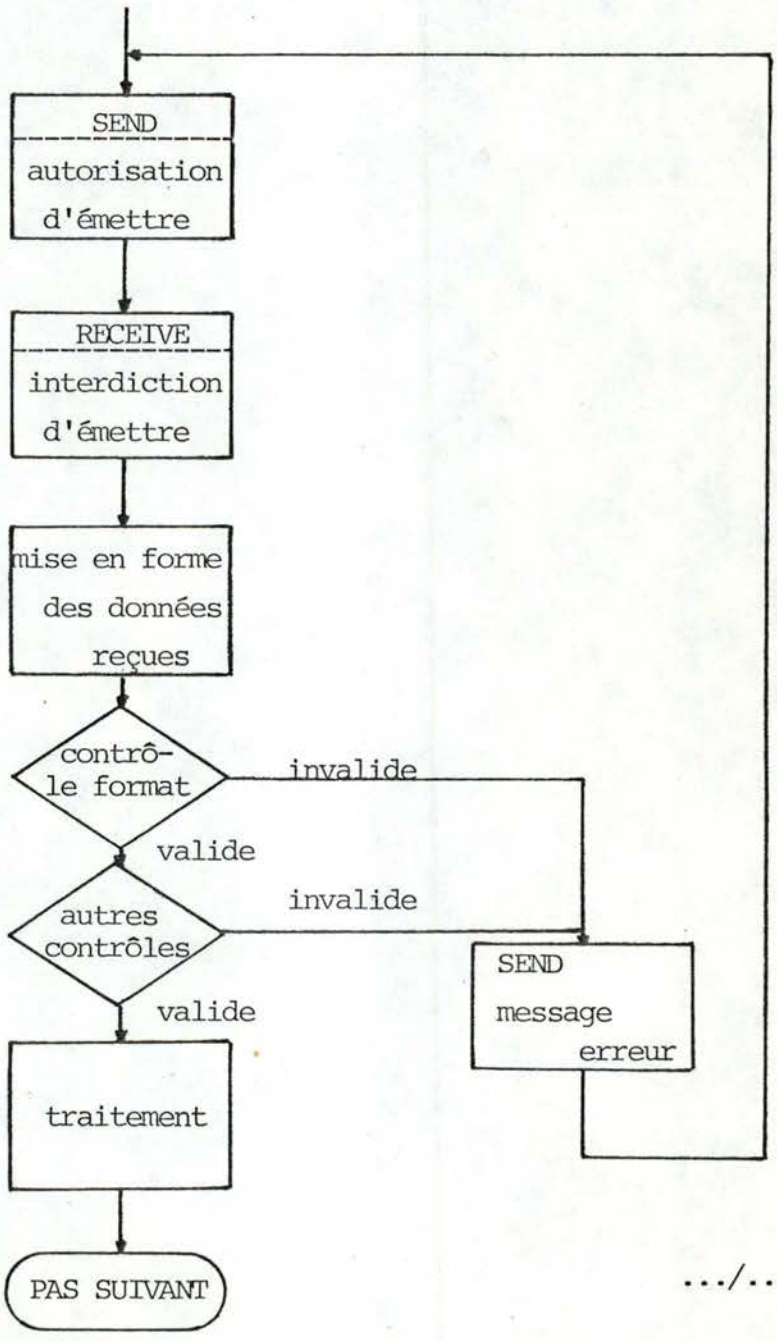
Le choix du "dialogue dirigé par la machine" nous permet d'introduire quelques simplifications non négligeables sur le plan pratique.
D'une part, l'utilisateur ne doit pas connaître de procédures, ni de mnémoniques pour se servir du terminal : chacune de ses interventions est la réponse à une question précise du programme.
D'autre part; en limitant le nombre d'actions valides que peut entreprendre l'utilisateur, on limite en même temps les contrôles à effectuer par le programme.

.../...

2. Correction immédiate des données.

La possibilité d'interaction, permet de corriger immédiatement toute donnée reçue. C'est là, nous semble-t-il, la différence peut-être la plus sensible entre une procédure en temps réel et une procédure en temps différé.

Remarquons la généralisation des formats variables dans l'introduction des données en temps réel; ceci nécessite des contrôles de format souvent inexistants en temps différé, mais qui présentent l'avantage d'être aisément standardisables (raison pour laquelle nous les distinguerons systématiquement).

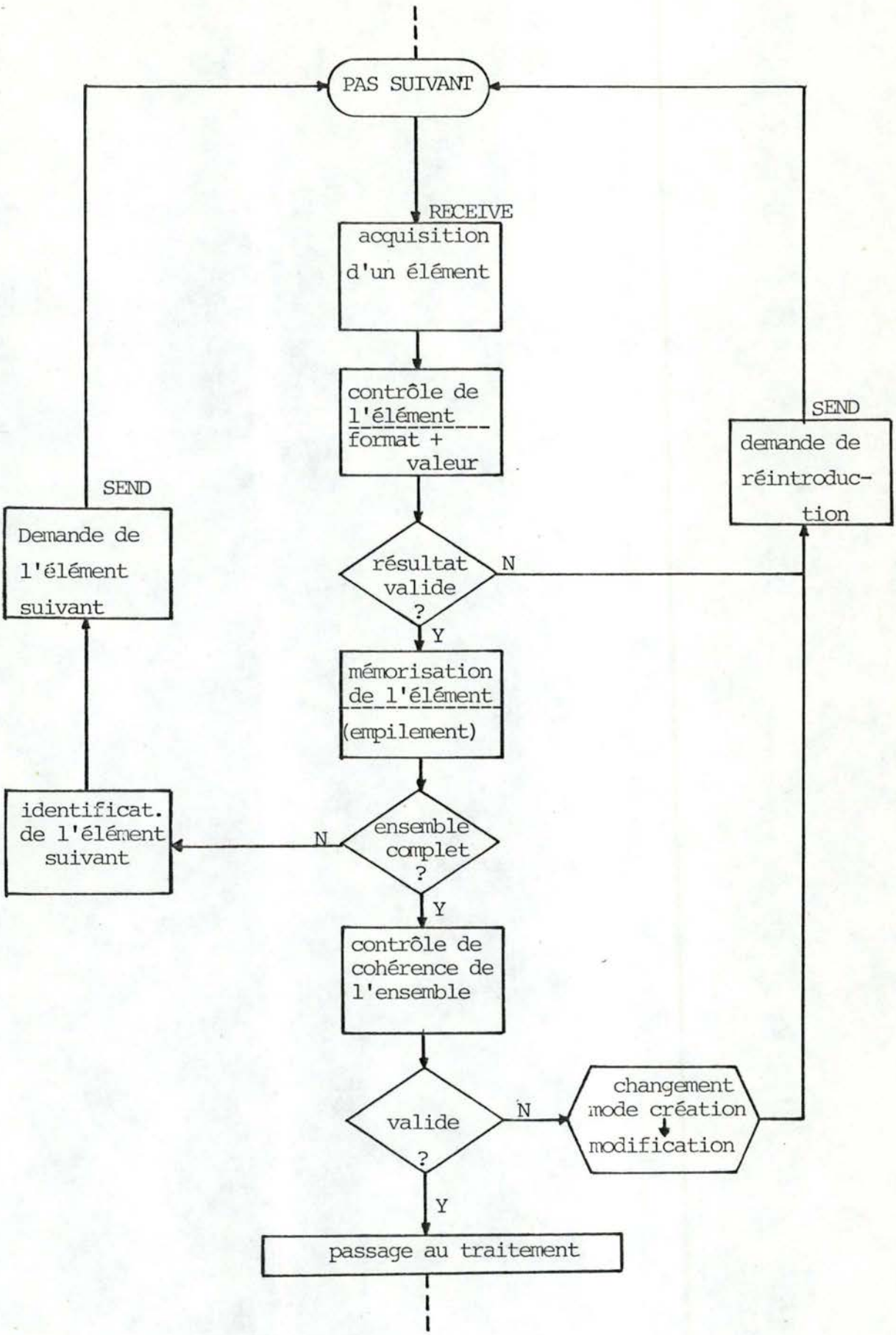


.../...

3. Progressivité d'acquisition des données.

La possibilité de la correction immédiate suggère de ne pas attendre l'introduction de l'ensemble complet des données sur lequel le traitement doit opérer, avant de procéder au contrôle et à la correction éventuelle des éléments.
Nous allons donc introduire l'ensemble de façon progressive.
Comme on pourra le voir sur l'ordinogramme qui suit, chaque élément est d'abord contrôlé individuellement.
Après constitution de l'ensemble de données en mémoire, cet ensemble est, s'il y a lieu, soumis à un contrôle de cohérence. Ce contrôle de cohérence peut remettre en question certaines valeurs d'éléments introduites. Lorsque l'utilisateur ré-introduit des valeurs d'éléments suite aux résultats du contrôle de cohérence, l'introduction n'est plus une création mais une correction (modification) d'un ensemble de données et le programme devra travailler sur l'image de l'ensemble de données qu'il vient de constituer en mémoire.

.../...


PAS SUIVANT
RECEIVE
acquisition d'un élément
contrôle de l'élément format + valeur
SEND
demande de réintroduction
SEND
Demande de l'élément suivant
résultat valide ?
N
Y
mémorisation de l'élément (empilement)
identificat. de l'élément suivant
N
ensemble complet ?
Y
contrôle de cohérence de l'ensemble
valide ?
N
changement mode création modification
Y
passage au traitement

1.C. La multiplicité des utilisateurs.

1. Concepts.

A l'exploitation en temps réel est généralement associée la multiplicité des utilisateurs : un même programme est actif simultanément pour plusieurs utilisateurs.
Dans un tel programme, chaque processus (occurrence d'action) est un processus "individuel" effectué pour compte d'un utilisateur. Dans un programme conversationnel, il convient en outre de faire l'hypothèse qu'il n'y a à aucun moment plus d'un processus actif pour un même utilisateur.
Un ensemble de processus successifs entrepris pour un même utilisateur constituera ce que nous appellerons une session-utilisateur ou un dialogue; à priori, une session-utilisateur, à l'instar de la session-programme globale (ensemble des processus successifs exécutés pendant l'exécution du programme), est susceptible de contenir des opérations itératives devant être initialisées et/ou clôturées par des processus spéciaux.

2. Interruption du dialogue.

En cas de concurrence de plusieurs utilisateurs, on ne peut admettre qu'un utilisateur monopolise le programme pendant toute la durée de son dialogue. La contrainte sur la durée du temps d'attente des divers utilisateurs oblige à segmenter le dialogue.
Plus précisement, il s'agit de segmenter chaque processus individuel, c'est-à-dire d'en permettre l'interruption et la poursuite.
Le problème qui se pose alors est de garder la trace des données mémorisées par un processus individuel et de l'état de ce processus.
A cet effet toute unité de traitement conduisant un dialogue devra gérer : - a. une table des utilisateurs, contenant un vecteur d'état par processus individuel en cours.
- b. une "pile" (⁂) des données mémorisées, pour compte de chaque utilisateur.

(⁂) Voir page suivante.

.../...

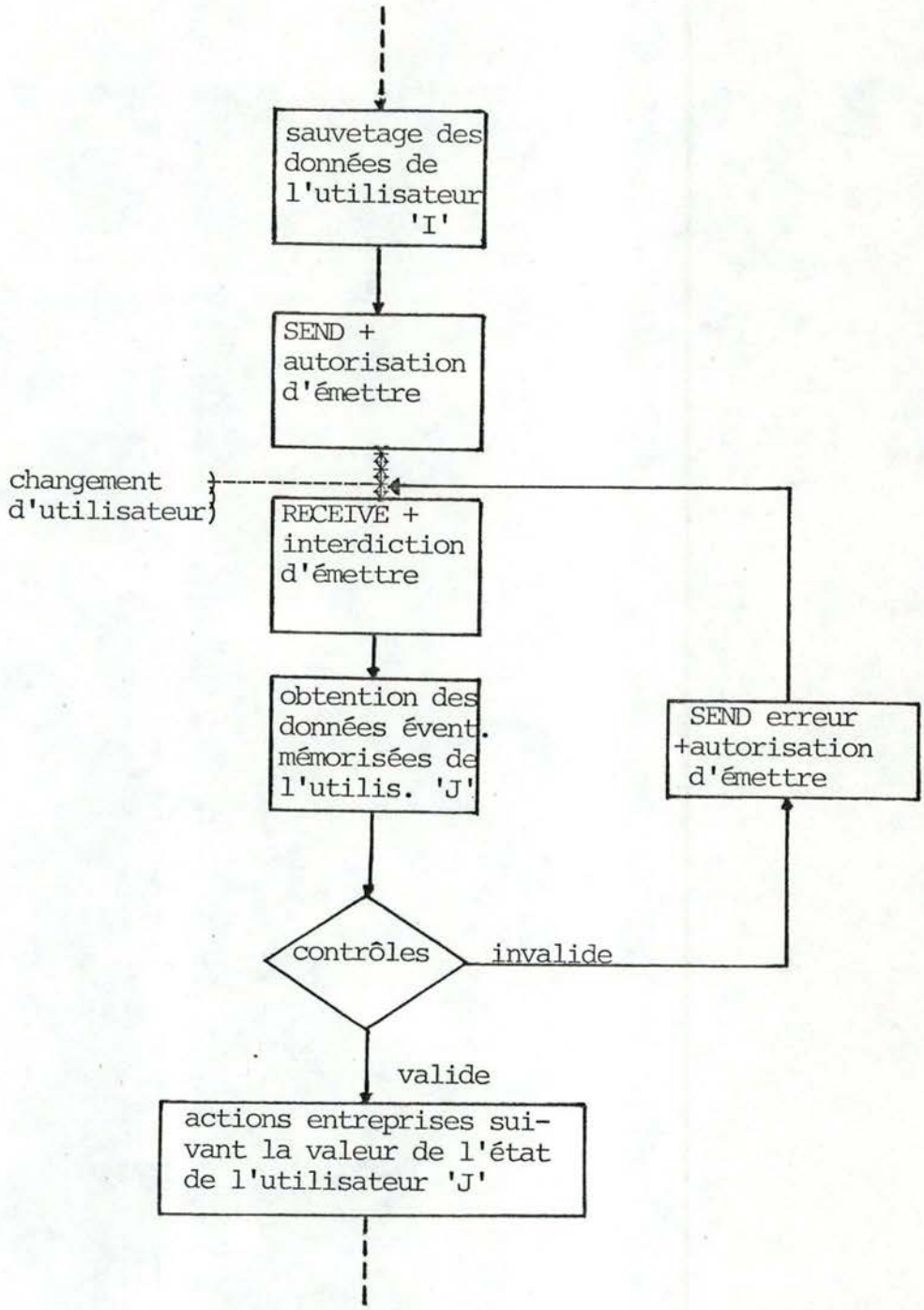
a. Un vecteur d'état de la table des utilisteurs comporte l'identification d'un utilisateur et un qualificatif d'état; un processus individuel doit être vu comme un ensemble ordonné de pas de dialogue, le qualificatif d'état peut être simplement un indice d'ordre dans cet ensemble.
Un utilisateur n'existe pour le module de dialogue que s'il est référencé dans la table.
Un traitement spécial est nécessaire à l'initialisation d'un dialogue (ou session-utilisateur) afin d'initialiser le vecteur d'état de cet utilisateur. Un utilisateur qui déclare son dialogue terminé devra ré-initialiser son vecteur d'état avant d'amorcer un nouveau dialogue.

b. En ce qui concerne la pile des données mémorisées, nous distinguerons plus loin différentes possibilités suivant la nature de l'ensemble de données à constituer.

Voir schéma page suivante.

(*)N.B. : En réalité, il ne s'agit pas proprement d'une pile, mais d'une file d'attente gérée selon le principe "first in, first out" (FIFO). Nous utiliserons néanmoins -improprement- le terme pile, dans le souci d'éviter toute confusion, toute assimilation avec les files d'attente d'entrée et de sortie gérées par le moniteur du télétraitement et non pas -comme ici- par le programme d'application.

Le schéma suivant montre l'enchaînement des opérations lors du passage d'un utilisateur 'I' à un utilisateur 'J' :



1.D. Synthèse.

1. Etapes d'un processus.

Tout processus entrepris pour un utilisateur se décompose en trois grandes étapes.

a) création de l'ensemble à traiter :
pour chaque élément de l'ensemble :
- acquisition proprement dite (opération RECEIVE),
- contrôle de validité individuel (format et valeur) et correction (réintroduction) éventuelle,
- mise en forme et insertion dans la pile des données mémorisées.

b) correction de l'ensemble constitué :
- extraction de l'ensemble constitué pour un utilisateur dans la pile des données mémorisées,
- vérification de cohérence,
- correction (réintroduction) éventuelle de certains éléments.

c) traitement proprement dit de l'ensemble constitué.

Remarques :

L'interaction n'intervient qu'au cours des deux premières étapes; l'utilisateur n'a plus aucune initiative pendant le traitement proprement dit des données qu'il a fournies.

Les deux premières étapes -création et correction- se différencient principalement par rapport à la mémorisation des données. A l'étape de création, les données fournies par l'utilisateur sont insérées dans la pile de mémorisation. A l'étape de correction, les opérations sur cette même pile sont des opérations de remplacement.

.../...

2. Structure du vecteur d'état.

Toute décision prise par le programme au cours d'un processus, repose sur l'analyse du vecteur d'état de l'utilisateur pour lequel le processus est exécuté.
Le vecteur d'état d'un utilisateur fournit au programme :

a. l'identification de l'utilisateur à l'aide de la zone USER, contenant le nom de l'utilisateur.

b. des indicateurs de l'état d'avancement du processus, soit
- l'indicateur STATE, identifiant l'étape du processus et pouvant prendre les valeurs : 0 inactif
  1 création, 2 correction } mode interactif
  3 traitement
- l'indicateur STEP, identifiant le "pas" du dialogue; les valeurs de l'indicateur STEP ne sont significatives que lorsque l'indicateur STATE possède les valeurs 1 ou 2. L'indicateur STEP est composé de : STEP1, indiquant le numéro de type de l'élément introduit.
  STEP2, indiquant le numéro d'occurrence de l'élément introduit parmi les éléments de même type.

Exemple d'utilisation des indicateurs STEP1 et STEP2.

Soit un mouvement de mise à jour d'un fichier signalétique dont les données composantes sont introduites une à une.

STEP1	STEP2	
1	1	matricule
2	1	nom
3	1	prénom
4	1	adresse
5	1	état civil
6	1	nombres d'enfants

.../...

Soit un bon de commande dont les lignes sont introduites une à une.

STEP1	STEP2	
1	1	ligne d'entête
2	1	ligne de corps de la commande
2	2	ligne de corps de la commande
2	3	ligne de corps de la commande
.	.	. . .
.	.	. . .
.	.	. . .

Rappel : Les vecteurs d'état des différents utilisateurs sont mémorisés dans la table des utilisateurs.

Chapitre 2. : Extension des langages

Pour permettre la réalisation de programmes en temps réel, nous proposons d'étendre les langages associés à la méthode MEMO-PROGES et d'adapter en conséquence le système objet qui en supporte la réalisation.

2.A. Paramètres globaux banalisés (LCM).

1. Adaptation du système objet.

Dans un programme en temps réel, tout processus est exécuté pour le compte d'un utilisateur et peut, par ailleurs, être interrompu. A tout processus commencé est associé un vecteur d'état comportant l'identification de l'utilisateur propriétaire et des indicateurs de l'état d'avancement du processus.

Ce vecteur d'état sera incorporé à l'article U-EXT des paramètres globaux transmis à tout module par l'instruction d'appel normalisée. De cette manière, ses éléments seront immédiatement accessibles à tout module, sans qu'il soit plus nécessaire de consulter la table des utilisateurs.

La fonction terminale d'acquisition des messages d'entrée (fonction RECEIVE) doit naturellement être dotée des moyens d'identifier l'utilisateur dont émane chaque message qu'elle reçoit. A la réception de cette identification, le module directeur extraira dans la table des utilisateurs le vecteur d'état concernant l'utilisateur identifié et en transfèrera le contenu dans l'article commun U-EXT.

A la fin de chaque itération du traitement, lors du retour au module directeur de l'unité de traitement, ce dernier réinsèrera dans la table des utilisateurs le vecteur d'état éventuellement modifié par le programme.

.../...

2. Langage et programmation.

Les éléments du vecteur d'état incorporés à l'article U-EXT sont des variables globales implicitement declarées, accessibles à tout module sous les noms suivants :

- identification de l'utilisateur :

  @USER — nom de l'utilisateur (12 caractères alphanumériques) (°)

- état d'avancement du processus :

  @STATE — état du processus, mode d'opération du programme
  valeurs : OFF = utilisateur inactif
  CREATE = mode "création" } mode
  MODIFY = mode "correction" } "interactif"
  USE = mode "traitement"

  @STEP — identification du "pas" de dialogue, identification de la donnée reçue et à traiter (n'est significatif que lorsque @STATE = CREATE ou MODIFY).
  @STEP est composé des éléments suivants :

  @STEP1 — numéro de type de la donnée reçue (valeurs possibles : de 00 à 99).

  @STEP2 — numéro d'occurrence de la donnée reçue (valeurs possibles : de 00 à 99).

Traduction (COBOL).

```
01  U-EXT
    ..... (paramètres définis antérieurement).
    02  U-USER     PICTURE X(12).
    02  U-STATE    PICTURE 9.
    02  U-STEP.
      03  U-STEP1  PICTURE 99.
      03  U-STEP2  PICTURE 99.
```

(°) Le format retenu pour cette donnée est celui que définit la norme COBOL pour les opérations du télétraitement. Cf. annexe.

.../...

Les variables @USER, @STEP, @STEP1, @STEP2 sont accessibles à toute instruction de tout module.
Les instructions @SET STATE et @RESET STATE permettent de manipuler l'indicateur @STATE :

<instr set-state> ::= @SET STATE = <val set-state>
<val set-state> := CREATE/MODIFY/USE

<instr reset state> ::= @RESET STATE (place la valeur OFF dans @STATE)

Le test @STATE permet de connaître la position actuelle de cet indicateur :

<test state> ::= @STATE = <val state>
<val state> ::= OFF/CREATE/MODIFY/INTERACTIVE/USE

@STATE = INTERACTIVE est synonyme de (@STATE = CREATE OR
@STATE = MODIFY).

2.B. Gestion des piles (LDF).

1. Table des utilisateurs.

a) adaptation du système objet.

La table des utilisateurs est un fichier contenant un enregistrement (vecteur d'état) pour chaque utilisateur. Les accès à ce fichier, comme à tout autre, seront assurés par une fonction terminale.

@USER (identification de l'utilisateur) est la clé primaire de ce fichier.

Deux possibilités de réalisation peuvent être envisagées, ou bien la table contient un vecteur d'état pour chaque utilisateur potentiel du programme, même inactif; ou bien elle contient les vecteurs d'état des seuls utilisateurs actifs.

.../...

b) langage et programmation.

Dans toute fonction devant communiquer avec la table des utilisateurs, cette dernière est traitée comme un fichier virtuel qui sera déclaré de la manière suivante :

@FILE <id-fichier> TYPE USER-TABLE

USER-TABLE est un type de fichier prédéfini; la structure de ce fichier est la suivante :

@USER : identification de l'utilisateur.
@STATE : état du dialogue.
@STEP1 : valeur de l'indicateur STEP1.
@STEP2 : valeur de l'indicateur STEP2.

2. Pile des données mémorisées. (LDF)

a) adaptation du système objet.

L'ensemble des données progressivement acquises pour un utilisateur en mode "interactif" et devant être traité ensuite est mémorisé dans un fichier. Les accès à ce fichier, comme à tout autre, seront assurés par une fonction terminale.

L'ensemble mémorisé pour un utilisateur peut être un article (ex. : un mouvement de mise à jour d'une fiche signalétique). Dans ce cas, la clé primaire de chaque enregistrement est @USER (identification de l'utilisateur).

L'ensemble mémorisé pour un utilisateur peut être un groupe d'articles (ex. : l'ensemble des lignes constituant un bon de commande).
@USER (identification de l'utilisateur) est alors la clé du groupe d'articles; la clé primaire de chaque enregistrement est composée de @USER (identification de l'utilisateur) et @STEP (identification de l'élément dans l'ensemble).

.../...

b) langage et programmation

Dans toute fonction devant communiquer avec la pile des données mémorisées, cette dernière est traitée comme un fichier virtuel, déclaré d'une des manières suivantes en fonction du format adopté pour les enregistrements mémorisés : (*)

@ FILE <id-fichier> <syst-enreg> TYPE <type-fichier>
FOR REAL-TIME.

<syst-enreg> ::= SHORT/LONG

Le système d'enregistrement signifie que chaque article est (LONG) ou n'est pas (SHORT) précédé d'un préfixe d'article contenant le nom de type (@ TYPE) de cet article.
L'indication FOR REAL-TIME associe à chaque article les indicateurs permettant de gérer la pile, à savoir @ USER, @ STEP.
(Remarque : Si la déclaration @ FILE ne mentionne pas de système d'enregistrement, le système retenu sera celui qui est indiqué par défaut dans la description cataloguée sous le nom <type-fichier> .)

(*) SHORT : sans préfixe d'articles.
LONG : avec préfixe standardisé; le contenu du préfixe permet au programme de gérer l'article du fichier.

Les opérations possibles sur la pile des données sont toutes à accès sélectif.
En ce qui concerne l'article nous distinguons les opérations suivantes :

OBTAIN RECORD (KEY =@ USER)
EXTRACT RECORD (KEY =@ USER)
INSERT (KEY =@ USER)
REPLACE (KEY =@ USER)
DELETE (KEY =@ USER)

Dans le cas de mémorisation d'un groupe d'articles la gestion de la pile est possible grâce aux opérations suivantes :

Voir page suivante.

OBTAIN RECORD	(KEY =@USER, @STEP)
OBTAIN GROUP	(KEY =@USER)
EXTRACT GROUP	(KEY =@USER)
INSERT	(KEY =@USER, @STEP)
REPLACE	(KEY =@USER, @STEP)
DELETE	(KEY =@USER, @STEP)

Remarque :

L'opération EXTRACT est une opération OBTAIN (consultation) doublée d'une opération DELETE. Elle sera utilisée lorsque, la mémorisation des données acquises pour un utilisateur étant achevée, cet ensemble de données doit être transmis aux fonctions de traitement proprement dit.

.../...

2.C. Déclaration des interfaces. (LDF)

1. Adaptation du système objet.

L'acquisition et la production des messages sont réalisés par les fonctions terminales SEND et RECEIVE. Ces fonctions opèrent la conversion de support entre un fichier réel et un fichier virtuel; ce dernier servira, comme tout autre fichier virtuel, d'interface aux fonctions programmées.
Les messages transmis et constituant le fichier sont des chaînes de caractères non structurées. Comme la longueur de la chaîne est souvent variable, le fait d'indiquer cette longueur pourra faciliter la gestion de la chaîne. En plus, nous associerons au message un indicateur de fin; cet indicateur est nécessité par le fait qu'un ensemble de données peut-être constitué de plusieurs messages et il informera le programme du fait que l'ensemble de données est complet ou non.

ex. : soit un bon de commande

chaîne de caract.	indicateur
ligne d'entête	fin de ligne
ligne de corps	fin de ligne
ligne de corps	fin de ligne
. . .	. . .
ligne de corps	fin de commande

la valeur de "fin de commande" signale que l'ensemble de données 'bon de commande' est complet.

2. Langage et programmation.

La déclaration d'un fichier virtuel (et celle d'une zone de mémorisation en mémoire interne) doit faire apparaître la structure particulière du message.

.../...

A cette fin, on déclarera un système d'enregistrement REAL-TIME; cette déclaration aura pour effet de générer un préfixe standardisé de message. Ce préfixe est composé du nom du propriétaire du message (OWNER) de l'indication de longueur (LENGTH) et de l'indicateur de fin (END-INDIC).

Le seul attribut variable, et donc à déclarer, de la chaîne de caractères est sa longueur maximum.

La déclaration aura alors la forme suivante :

{@FILE / @RECORD} <id> REAL-TIME LENGTH = <longueur maximum>

Toute instruction de tout module peut faire référence aux éléments du message, ces éléments sont accessibles sous les noms :

@OWNER OF <id>
@LENGHT OF <id>
@END-INDIC OF <id>
@DATA OF <id>

Traduction (COBOL).

```
01  <id>
 11   F-OWNER      PIC X(12).
 11   F-LENGHT     PIC 999 COMP.
 11   F-END-INDIC  PIC X.
 11   F-DATA
  12   F-CHAR      PIC X OCCURS '&LENGTH'.
 11   F-X          PIC 999 COMP.
```

2.D. Opérations d'accès (LDF).

1. Adaptation du système objet.

Rappelons que les fichiers d'entrée et de sortie d'un programme peuvent avoir différents statuts ; à chaque statut de fichier correspondent des opérations d'accès spécifiques.

.../...

Le tableau suivant reprend les différents statuts de fichiers :

mode d'accès	statut d'entrée	statue de sortie
exhaustif	M (master file) READ	N (new file) WRITE
sélectif	A (alternate input) OBTAIN	U (updated file) INSERT REPLACE DELETE

Les fichiers de messages ne sont utilisés qu'en accès exhaustif et ne peuvent donc posséder que les statuts M et N; les opérations d'accès à ces fichiers étant READ et WRITE.

2. Langage et programmation.

Les instructions permettant l'exécution de ces opérations d'accès devront désigner le fichier auquel elles accèdent et localiser la chaîne de caractères faisant l'objet du transfert.
Compte tenu de la multiplicité des utilisateurs, une instruction d'accès en temps réel devra également identifier l'émetteur ou le destinataire du message; cette identification n'est prise en compte qu'au niveau des fonctions terminales SEND et RECEIVE.
Rappelons le schéma de base des instructions d'accès sur toute espèce de fichier :

opération	désignation du fichier	désignation de l'argument	options propres au système d'enregistrement du fichier
{READ / WRITE}	<id. fichier>	[USING <id argument>]	.....

.../...

Par défaut, l'argument de l'opération (donnée tranférée) est le contenu de la zone de stockage implicitement associée au fichier et qui est elle-même désignée par <id.fichier>.
L'option USING permet de spécifier en guise d'argument le contenu d'une autre zone de mémoire; <id.argument> doit désigner soit la zone de stockage associée à un autre fichier (zone définie par une déclaration FILE), soit un article constitué en mémoire interne du module (zone définie par une déclaration RECORD). Cet argument doit avoir le même système d'enregistrement que le fichier concerné par l'opération d'accès.

Opérations READ (REAL-TIME).

- identification du message (option USING)
  L'argument cité éventuellement par l'option USING doit posséder l'attribut "REAL-TIME".

- description du contenu (préfixe d'enregistrement)
  Les indicateurs de longueur et de fin sont garnis au cours de l'exécution de l'instruction READ, c'est-à-dire soit par le module terminal RECEIVE (réception du message par le programme), soit par l'opération WRITE programmée dans la fonction couplée par l'interface en cause.

- identification de l'émetteur
  Le terminal émetteur du message est identifié par le contenu de la zone USER incorporée à l'article U-EXT.

  Le schéma complet d'une instruction READ aura alors la forme suivante :

  @ READ <id.fichier> [USING <id.argument>]

.../...

Opération WRITE (REAL-TIME).

- identification du message (option USING)
  L'option USING peut être utilisée comme pour l'opération READ.
  Nous proposons en outre une facilité syntaxique permettant l'envoi de messages constants : à cette fin, l'option USING pourra être remplacée par l'option ='littéral'

- description du contenu (préfixe d'enregistrement, option ENABLE).
  Sauf si l'argument désigné est le contenu de la zone associée au fichier maître de la fonction -auquel cas l'indicateur de longueur est déjà garni-, l'indicateur de longueur de message est calculé par une routine appelée par l'opération WRITE.
  L'option ENABLE positionne l'indicateur de fin à la valeur "fin de message". Si cette option n'est pas retenue, l'indicateur est positionné à la valeur "fin de segment". Cet indicateur n'est décodé que par la fonction terminale SEND.
  La valeur "fin de segment" provoque l'envoi du texte au terminal destinataire; la valeur "fin de message" provoque l'envoi du texte et d'une autorisation d'émettre.

- identification du destinataire (option USER =)
  Par défaut, le destinataire du message est celui qu'identifie l'indicateur USER de l'article U-EXT.
  L'option USER de l'instruction WRITE permet d'envoyer le message à un autre utilisateur qu'elle désigne.

Le schéma complet de l'instruction WRITE est le suivant :

@ WRITE <id. fichier>
[{USING <id. argument> | = ' littéral '}]
[,USER = <iden>]
[, ENABLE]

.../...

2.E. Structure des modules de traitement (LDF).

1. Rappel

En LDF, la structure des traitements est définie par la déclaration de boucles quantifiées. Le corps d'une boucle est encadré par les titres @FØR ... @ENDFØR et est exécuté sur chaque occurrence de la quantité d'information du fichier maître de la fonction (fichier lu par l'instruction @READ) désignée par le titre ou quantificateur @FØR.

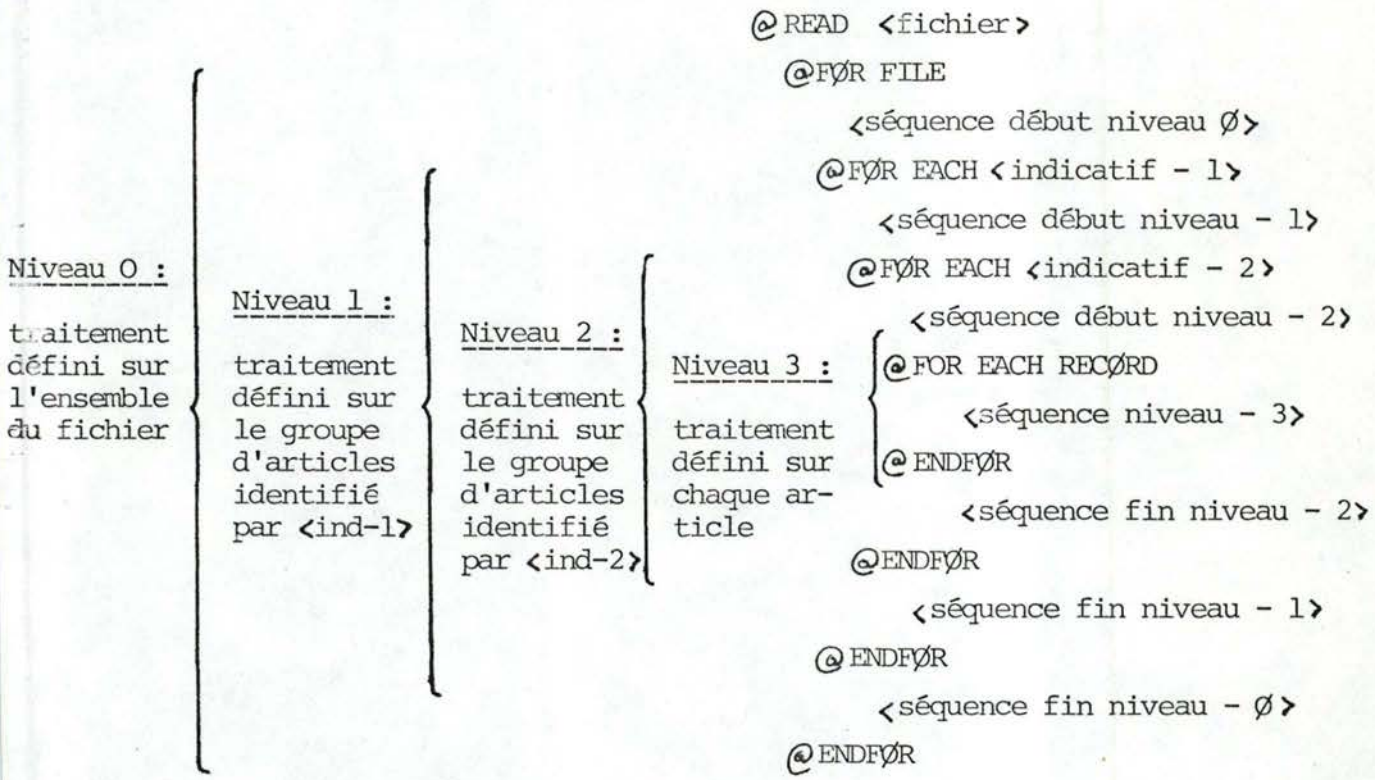
Les quantificateurs prévus pour une exploitation en temps différé sont :

- @FØR FILE pour un traitement défini sur l'ensemble du fichier d'entrée.
- @FØR EACH <indicatif> pour un traitement défini sur chaque groupe d'articles identifié par chaque valeur d'un indicatif.
- @FØR EACH RECORD pour un traitement défini sur chaque article.

Ainsi le module de traitement apparaîtra comme composé d'un certain nombre de blocs de traitement quantifiés, inclus les uns dans les autres, lui donnant une structure hiérarchisée.

Voir exemple page suivante.

Exemple :



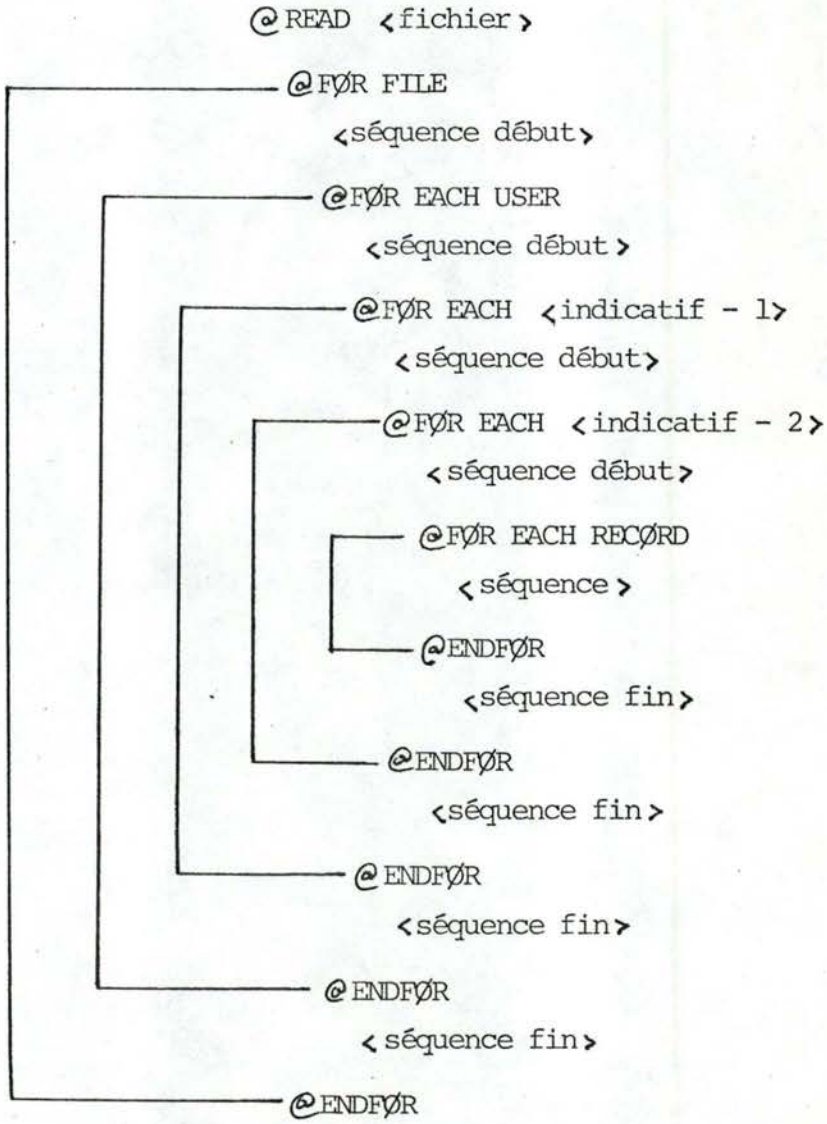
Le passage d'une occurrence à l'occurrence suivante d'un groupe d'articles désigné par le titre @FØR EACH <indicatif> s'effectue à chaque rupture sur l'indicatif.
Une rupture se produit lorsque la valeur actuelle de l'indicatif est différente de sa valeur à l'itération précédente.

Un groupe d'articles identifié par une valeur particulière d'un indicatif "majeur" (indicatif - 1) peut être lui-même partitionné en différents sous-groupes correspondant aux différentes valeurs d'un indicatif "mineur" (indicatif - 2), et ainsi de suite. Une rupture sur un indicatif majeur entraîne une rupture simultanée sur tous les indicatifs mineurs.

.../...

2. En temps réel, tout sous-ensemble significatif dans le fichier de messages lu par une fonction appartient à un utilisateur particulier. Au premier niveau de décomposition, ce fichier est donc partitionné en "groupes d'articles" correspondant aux différents noms d'utilisateurs.
Pour définir un traitement sur un tel groupe d'articles, on utilisera un quantificateur @FØR EACH USER.

La structure du module de traitement aura alors la forme suivante :



.../...

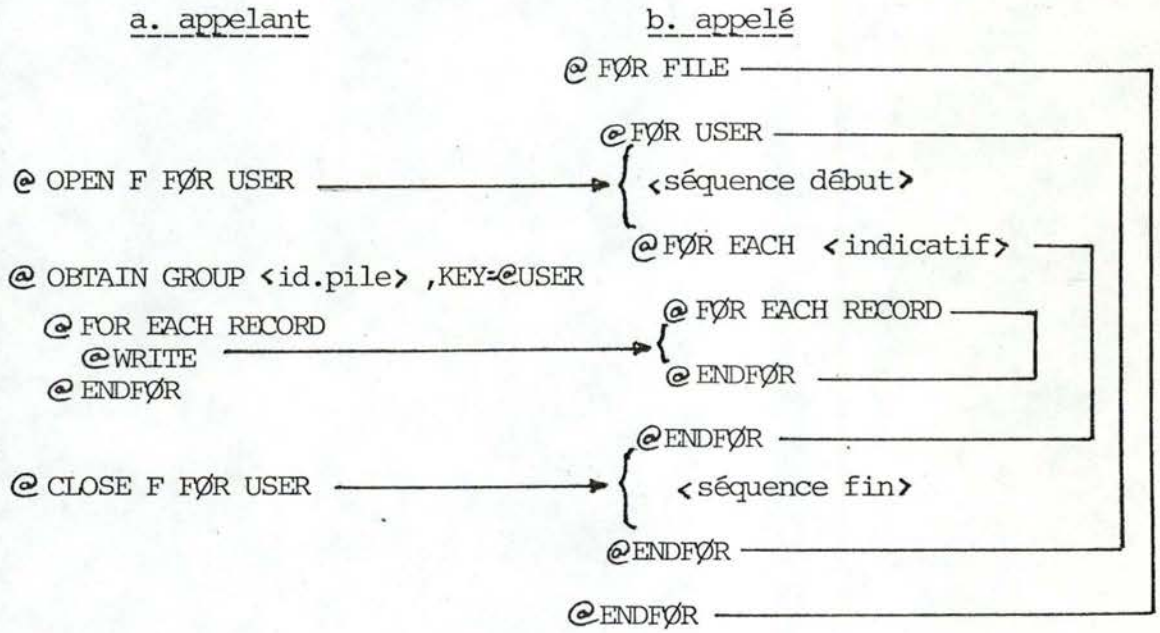
3. Pour l'initialisation/la clôture (<séquence début> / <séquence fin>) d'un bloc FØR EACH USER, le programme ne peut attendre le moment aléatoire où un nouvel utilisateur se manifestera en provoquant une rupture sur le nom d'utilisateur.
   Nous devrons donc initialiser/clôturer explicitement ce bloc FØR EACH USER; à cette fin le module appelant le module de traitement contiendra deux instructions, c'est-à-dire :

   l'instruction @OPEN <fichier>* FØR USER
   qui provoquera l'initialisation du bloc FØR EACH USER

   et l'instruction @CLOSE <fichier>* FØR USER
   qui provoquera la clôture du bloc FØR EACH USER

   * <fichier> désigne la fonction de traitement.

   Le schéma suivant met en évidence de quelle façon le module appelant déclenche les traitements programmés dans le module de traitement :



.../...

Exemple : traitement des bons de commande

appelant :

```
@OPEN commandes FØR USER
  @OBTAIN GROUP <id.pile> , KEY = @USER
    @FØR EACH RECØRD
      @WRITE commandes USING <id.pile>
    @ENDFØR
@CLOSE commandes FØR USER
```

appelé :

```
@READ commandes
  @FØR FILE
    @OBTAIN dernier - n° - facture
  @FØR EACH USER
      total - facture : = 0
      dernier - n° - facture : = dernier - n° - facture + 1
  @FØR EACH RECØRD   IF TYPE = en-tête
      n° - facture : = dernier - n° - facture
    @OBTAIN RECØRD client,
        KEY = n° - client FROM commandes
    @WRITE factures, TYPE = adresse
  @ENDFØR
  @FØR EACH RECØRD  IF TYPE = corps
    @OBTAIN RECØRD produit,
        KEY = n° - produit FROM commandes
        prix : = prix-unitaire * quantité
    @WRITE factures,  TYPE = ligne
        total - facture : = total - facture + prix
  @ENDFØR
  @WRITE  factures USING total - facture
  @ENDFØR
    @REPLACE dernier - n° - facture
@ENDFØR
```

.../...

Chapitre 3. : Adaptation du système objet (*)

3.A. Adaptation du module directeur.

Du point de vue de l'adaptation du module directeur nous avons relevé deux problèmes.
Les problèmes concernent :

- l'identification du propriétaire des données reçues à chaque itération.
- l'initialisation et la terminaison de la session d'activité de l'unité de traitement.

a. Identification de l'utilisateur.

A chaque itération le module directeur appelle le module RECEIVE qui, au retour, lui transmet le message reçu de la file d'attente; message préfixé par le nom (@OWNER) du terminal émetteur.
Le module directeur doit, avant tout, identifier l'utilisateur propriétaire de ces données pour pouvoir individualiser l'itération.

A cette fin, le module directeur accède à la table des utilisateurs et extrait de cette table le vecteur d'état de l'utilisateur en question, possédant ainsi toutes les informations pour la mise à jour de l'article U-EXT des paramètres globaux transmis à tout module.

A la fin de l'itération pour un utilisateur, le module directeur accède de nouveau à la table des utilisateurs et met à jour le vecteur d'état de l'utilisateur avec les valeurs renvoyées dans l'article U-EXT par le module appelé.

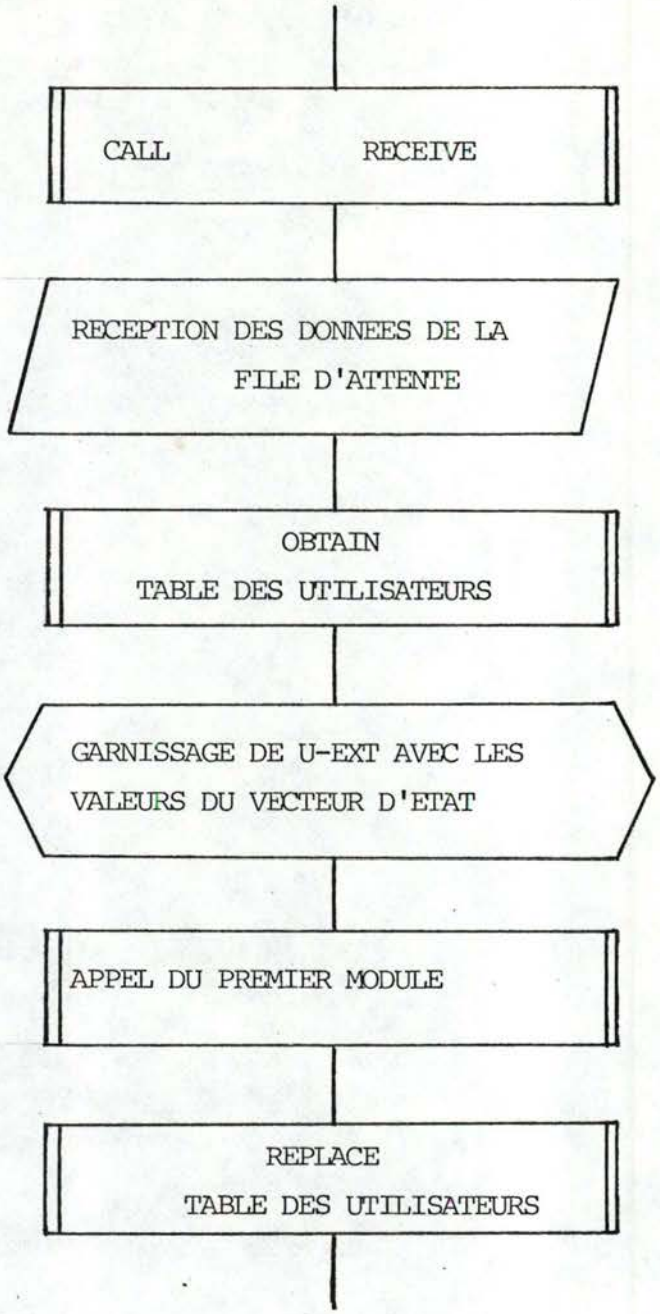
La fonction d'identification du module directeur est illustrée par le schéma suivant.

Ce schéma illustre les opérations effectuées lors d'une itération.

Voir page suivante pour schéma.

(*) : Remarque : afin que le lecteur puisse mieux comprendre les programmes, nous lui conseillons de lire l'annexe traitant de la gestion du télétraitement en COBOL.

.../...


CALL RECEIVE
RECEPTION DES DONNEES DE LA FILE D'ATTENTE
OBTAIN TABLE DES UTILISATEURS
GARNISSAGE DE U-EXT AVEC LES VALEURS DU VECTEUR D'ETAT
APPEL DU PREMIER MODULE
REPLACE TABLE DES UTILISATEURS

b. Initialisation et terminaison de la session d'activité.

Il existe deux possibilités pour initialiser et terminer la session d'activité.

1) L'initiative de chargement peut venir d'un utilisateur; l'initialisation de l'unité de traitement s'effectuera alors du chargement de celle-ci.
Le module directeur devra détecter lui-même l'instant où il peut terminer la session; cet instant se présente lorsque tous les dialogues sont terminés et que le module RECEIVE signale au module directeur que la file d'attente d'entrée est vide, en renvoyant @STATUS = END.

Le module directeur peut connaître l'état de tous les dialogues en consultant la table des utilisateurs qui contient les vecteurs d'état de tous les utilisateurs.

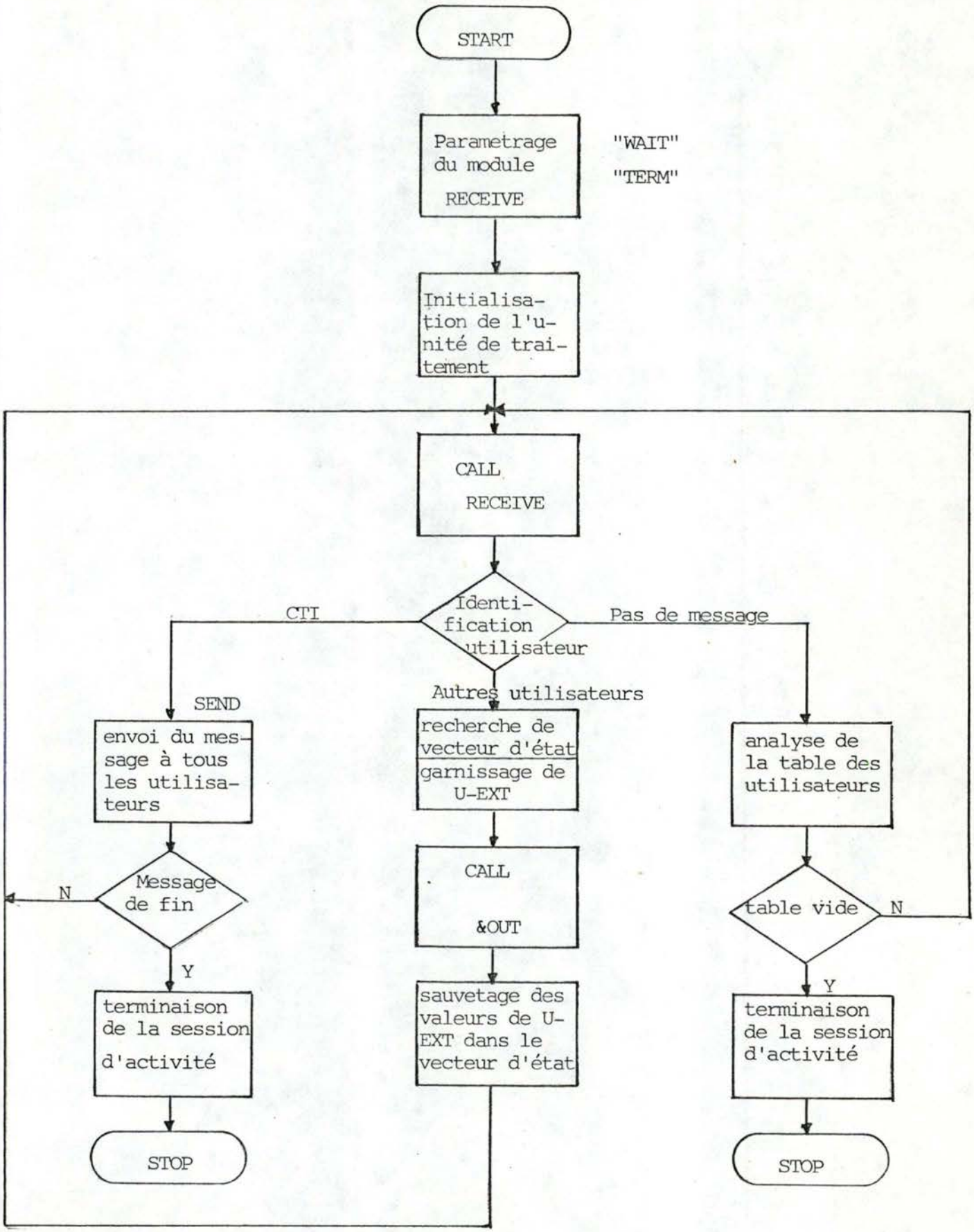
2) Dans le cas où l'initiative du chargement émane du centre de traitement, la terminaison de l'unité de traitement sera normalement également décidée par les responsables du centre de traitement. Dans le cadre du langage COBOL, le seul moyen satisfaisant pour détecter et traiter correctement cette demande est que ces responsables envoient un message ad hoc au programme; il faut pour cela effectuer un terminal au centre de traitement et relier ce terminal aux files d'attente d'entrée du programme.
Dans ce cas-ci le module RECEIVE ne peut lui-même signaler la fin de la session d'activité.
Ceci nous oblige à spécifier deux modes de travail pour le module RECEIVE, soit le mode "WAIT" et le mode "TERM".

WAIT : lorsque la file d'attente des messages d'entrée est vide, RECEIVE provoque une mise en attente du programme ("WAIT STATUS"); ce mode sera utilisé lorsque l'unité de traitement a été chargée à partir du CTI.

TERM : lorsque la file d'attente des messages d'entrée est vide, RECEIVE renvoie au module directeur un signal de fin pour clotûrer immédiatement l'exécution (à condition que plus aucun utilisateur ne soit actif); ce mode sera utilisé lorsque l'unité de traitement a été chargée à la demande d'un utilisateur.

.../...

Schéma de fonctionnement du module directeur :



Programmation du module directeur (MAIN)

Lors de la programmation du module directeur nous avons utilisé les langages "LDF" et "LCM" afin de faciliter la compréhension du programme. Normalement l'utilisation de ces langages n'est pas permise dans les modules qui, comme c'est le cas ici, ne sont pas des modules appelés.

.../...

PROGRAMMATION DU MODULE DIRECTEUR (MAIN).

```
 @PARAMS &LENGTH,&OUT,&SYS,&END-MESS
*  &LENGTH: longueur maximum de l'article du fichier virtuel
*  &OUT: nom du module appele par MAIN pour le traitement
*        des messages recues.
*  &SYS: nom du terminal de controle (CTI).
*  &END-MESS: texte du message indiquant la fin de la session
             d'activite.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. MAIN.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 @INTERFACE SECTION.
*  definition de la table des utilisateurs.
     @FILE STATE-VECTOR TYPE USER-TABLE.
*  definition du fichier contenant les messages de controle
*  envoyes vers tous les utilisateurs actifs.
     @FILE REPLY REAL-TIME LENGTH = &LENGTH.
 WORKING-STORAGE SECTION.
*  definition de l'article contenant les indicateurs globaux de
*  gestion de l'unite de traitement.
   COPY U-EXT.
*  EXEC-MODE: zone permettant de specifier le mode de
*             travail (WAIT/TERM) du module RECEIVE.
 01  EXEC-MODE PIC X(4).
*  definition du fichier contenant les messages recus/envoyes
*  de/vers les utilisateurs.
     @FILE ART-MESS REAL-TIME LENGTH = &LENGTH.
 COMMUNICATION SECTION.
 CD  ENTREE FOR INITIAL INPUT
     SOURCE IS EMETTEUR.

 PROCEDURE DIVISION.

 U-ENTER SECTION.
 G-ENTER.
     MOVE SPACES TO U-EXT.
*  specification du mode de travail du module RECEIVE.
     IF EMETTEUR = SPACES OR "&SYS" MOVE "WAIT" TO EXEC-MODE
        ELSE MOVE "TERM" TO EXEC-MODE.
     @PREPARE RECIVE (EXEC-MODE).

 U-INIT SECTION.
 G-INIT.
     @OPEN RECIVE @OPEN &OUT.
     @OPEN STATE-VECTOR @OPEN REPLY.
```

```
 U-CORPUS SECTION.
 G-CORPUS.
     @ICALL RECIVE (ART-MESS).
*  detection de la fin de la session d'activite.
     IF @STATUS = END @RESET STATUS GO TO U-TERM.
*  envoi d'un message de controle ?
     IF @OWNER OF ART-MESS = "&SYS"
        PERFORM U-MESS GO TO G-CORPUS.
*  recherche des indicateurs propres a l'utilisateur
*  duquel on a recu un message.
     MOVE @OWNER OF ART-MESS TO @USER OF STATE-VECTOR
     @OBTAIN RECORD STATE-VECTOR KEY = @USER
     MOVE CORRESPONDING STATE-VECTOR TO U-EXT
*  appel du premier module de traitement du message.
     @ICALL &OUT (ART-MESS)
*  sauvetage des indicateurs propres a l'utilisateur
*  dans la table des utilisateurs.
     MOVE CORRESPONDING U-EXT TO STATE-VECTOR
     @REPLACE STATE-VECTOR KEY = @USER.
     GO TO G-CORPUS.


 U-TERM SECTION.
 G-TERM.
     ALTER SUITE TO U-STOP.
*  y a-t-il encore des utilisateurs actifs ?
     @OBTAIN ALL STATE-VECTOR.
 @FOR EACH RECORD.
     IF @STATE OF STATE-VECTOR NOT = 0 ALTER SUITE TO G-CORPUS.
 @ENDFOR.

 SUITE.
     GO TO.
 U-MESS SECTION.
 G-MESS.
*  envoi d'un message de controle vers les utilisateurs actifs.
     @OBTAIN ALL STATE-VECTOR.
 @FOR EACH RECORD.
     IF @STATE OF STATE-VECTOR NOT = 0
        @WRITE REPLY USING ART-MESS USER = @USER OF STATE-VECTOR.
 @ENDFOR.
*  lorsque le message de controle est le message de fin
*  on termine la session d'activite.
     IF @DATA OF ART-MESS = "&END-MESS" GO TO U-STOP.


 U-STOP SECTION.
 G-STOP.
     @CLOSE RECIVE  @CLOSE &OUT.
     @CLOSE STATE-VECTOR  @CLOSE REPLY.
     STOP RUN.
```

3.B. Fonctions terminales.

a. Introduction.

Les règles d'homogénéïté des fonctions programmées amènent à distinguer une classe de fonctions terminales.
La seule opération de ces fonctions terminales consiste à exécuter des transferts de données entre support périphérique et mémoire centrale.
Vu que ces fonctions ne traitent en aucune façon le contenu des données elles sont aisément pré-programmables.
En temps réel nous distinguerons les fonctions terminales SEND et RECEIVE.

b. Fonction SEND

Lorsqu'elle est appelée, la fonction SEND reçoit en guise d'argument un article de format REAL-TIME.
La fonction SEND exécute deux opérations. La première consiste dans le transfert des données du programme vers les files d'attente de sortie; les données transférées contiennent les informations objet du traitement et des paramètres d'identification du destinataire.
Cette opération est exécutée grâce à l'instruction COBOL "SEND".
La seconde opération exécutée par la fonction SEND est accomplie par l'instruction "ENABLE INPUT TERMINAL" et elle consiste à autoriser le terminal à émettre une réponse.
Dans l'ordre on rencontre d'abord l'instruction SEND, qui envoie des segments de message. Un segment correspond à une ligne d'impression obéissant à certains critères de mise en page spécifiés dans l'instruction.
Le message complet constitue l'unité logique de transfert et peut être composé d'un ou plusieurs segments.
Quant à l'instruction ENABLE, elle est seulement exécutée après l'envoi du message complet, c'est-à-dire lorsqu'on rencontre l'indicateur de fin de message. (EMI)
Les paramètres et arguments nécessaires aux instructions sont repris dans le tableau page suivante :

.../...

	APPEL	RETOUR
SEND	. Texte du message à transmettre (@ DATA) (a). . Indicateur de quantité (@ END-INDIC 'option ENABLE') (a) . Critère de mise en page (b) . Destination (@ OWNER option USER) (a) . Nombre de destinations . Longueur du texte (@ LENGTH) (a)	. Error key : = 'OO' si réussi = "lO" si non
ENABLE	. Identification du terminal source(a). . Mot de passe (c).	

Remarques :

- (a) : contenu dans le préfixe de l'article reçu à l'appel.
- (b) : n'est considéré que pour l'édition.
- (c) : obtenu par le module par consultation d'une table ou par calcul.
- l'indicateur de quantité peut prendre la valeur '1' quand il s'agit d'un segment et '2' quand il s'agit d'un message.
- dans le mode conversationnel le nombre de destinations est toujours égal à '1'.

.../...

PROGRAMMATION DU MODULE SEND.

```
 @PARAMS &PGM,&LENGTH
*  &PGM: nom du module.
*  &LENGTH: longueur maximum de l'article du fichier virtuel.


 IDENTIFICATION DIVISION.
 @IENTRY &PGM (ART-MESS).
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*  mot de passe.
 01  PASSWD PIC X(10).

 LINKAGE SECTION.
*  definition du fichier virtuel contenant les messages
*  envoyes vers les utilisateurs.
     @FILE ART-MESS REAL-TIME LENGTH = &LENGTH.

 COMMUNICATION SECTION.
 CD  SORTIE FOR OUTPUT
     DESTINATION COUNT IS NB-DESTIN
     TEXT-LENGTH IS LONG-TXT.
     DESTINATION IS DESTINATAIRE.

 CD  ENTREE FOR INPUT
     SOURCE IS EMETTEUR.


 PROCEDURE DIVISION.

 @CONSTANT SECTION.
*  le nombre de destinataires est toujours egal a 1.
     MOVE 1 TO NB-DESTIN.

 @CORPUS SECTION.
*  garnissage du bloc de controle de sortie.
     MOVE @OWNER TO DESTINATAIRE
     MOVE @LENGTH TO LONG-TXT
*  envoi du message vers l'utilisateur.
     SEND SORTIE FROM @DATA OF ART-MESS WITH @END-INDIC.
*  detection de la fin du message (END OF MESSAGE).
     IF @END-INDIC = 2
*  appel d'un module pour l'obtention du mot de passe
*  de l'utilisateur.
     @ICALL OBTPWD (ART-MESS PASSWD)
*  garnissage du bloc de controle d'entree.
     MOVE @OWNER TO EMETTEUR
*  autorisation d'emettre.
     ENABLE INPUT TERMINAL ENTREE WITH KEY PASSWD.
```

c. Fonction RECEIVE.

La fonction RECEIVE exécute deux opérations. La première opération consiste dans l'acquisition de l'information émise par un terminal. L'autre opération a pour but d'interdire toute émission de la part du terminal.
L'opération d'acquisition est assurée par l'exécution de l'instruction "RECEIVE", qui effectue le transfert d'information de la file d'attente d'entrée vers le programme d'application.
L'opération de déconnexion logique du terminal est garantie par l'exécution de l'instruction "DISABLE INPUT TERMINAL".

Au terme de son intervention, la fonction transmet au module appelant le premier message disponible de la file d'attente désignée.
Le tableau suivant indique les paramètres nécessaires à l'appel et les arguments disponibles au retour, s'il n'y a pas de signal d'erreur.

	APPEL	
RECEIVE	. File d'attente à explorer (a)	. Texte du message (@DATA) . Date du jour (d) . L'heure d'envoi (d) . Identification du terminal source (@OWNER) (e) . Longueur du texte transmis (@LENGTH) (e) . Nombre de messages dans la file (d) . Indicateur de quantité (f) . Status key (@END-INDIC) cas possible : '00' (réuss
DISABLE	. Identification du terminal source (b) . Mot de passe (c)	

.../...

Remarques :

- (a) : paramètre constant du programme en cours, ce paramètre est passé lors de la préparation du programme d'application.
- (b) : obtenu au retour de l'instruction RECEIVE.
- (c) : obtenu par le module par consultation d'une table ou par calcul.
- (d) : sans intérêt pour les programmes envisagés ici.
- (e) : à retourner au programme appelant.
- (f) : égal à '2' (message) par hypothèse.
  ou à '3' (fin de groupe).
  lorsque le message est le dernier d'un groupe de messages reçus.

.../...

PROGRAMMATION DU MODULE RECIVE.

```
 @PARAMS &PGM,&LENGTH,&QUEUE
*  &PGM: nom du module.
*  &LENGTH: longueur maximum de l'article du fichier virtuel.
*  &QUEUE: specification du nom des files d'attente
*          d'entree.


 IDENTIFICATION DIVISION.
 @IENTRY &PGM (ART-MESS).
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*  EXEC-MODE: zone permettant de specifier le mode de
*             travail (WAIT/TERM) du module RECEIVE.
 01  EXEC-MODE PIC X(4).
*  mot de passe de l'utilisateur.
 01  PASSWD PIC X(10).

 LINKAGE SECTION.
*  definition du fichier virtuel contenant les messages
*  recus des utilisateurs.
     @FILE ART-MESS REAL-TIME LENGTH = &LENGTH.

 COMMUNICATION SECTION.
 CD  ENTREE FOR INPUT
     SOURCE IS EMETTEUR
     TEXT LENGHT IS LONG-TXT
     END-KEY IS FIN-TXT.

*  redefinition du bloc de controle d'entre.
 01  BLOC-CONTROLE.
   02  QUEUE-ID PIC X(48).
   02  FILLER PIC X(39).


 PROCEDURE DIVISION.

 @CORPUS SECTION.
*  appel d'un module pour l'obtention du mot de passe.
     @ICALL OBTPWD (ART-MESS PASSWD).
*  garnissage du bloc de controle avec la structure des
*  files d'attente d'entree.
     MOVE "&QUEUE" TO QUEUE-ID
*  determination du mode de travail (WAIT/TERM).
     IF EXEC-MODE = "WAIT"
        RECEIVE ENTREE MESSAGE INTO @DATA OF ART-MESS
     ELSE RECEIVE ENTREE MESSAGE INTO @DATA OF ART-MESS
          NO DATA @SET STATUS = END  @EXIT.
*  interdiction d'emettre.
     DISABLE INPUT TERMINAL ENTREE WITH KEY PASSWD.
*  garnissage du prefixe de l'article du fichier virtuel.
     MOVE EMETTEUR TO @OWNER
     MOVE LONG-TXT TO @LENGTH
     MOVE FIN-TXT TO @END-INDIC.


*  reception du parametre specifiant le mode de travail.
 @PARAM SECTION USING EXEC-MODE.
```

Troisième partie :
Exemple : Traitement d'un bon de commande

1. Introduction
2. Description des modules spécifiques

1. Introduction

a. le problème

Plusieurs utilisateurs peuvent introduire un ou plusieurs bons de commande. Ces bons de commande seront contrôlés par le programme et éventuellement corrigés et/ou modifiés par les utilisateurs, après quoi le programme créera les factures correspondant aux bons de commande.

Le bon de commande introduit par les utilisateurs est de la forme suivante :

entête : - NOM
- ADRESSE
- NUMERO-POSTAL LOCALITE

corps : - NUMERO DE PRODUIT1 QUANTITE-1
- NUMERO DE PRODUIT2 QUANTITE-2

.
.
.

Les lignes d'entête et du corps de la commande seront introduites une à une par l'utilisateur. Lorsqu'une ligne contient plus d'un élément,
ex. : dans l'entête NUMERO-POSTAL LOCALITE
dans le corps NUMERO DE PRODUIT-n QUANTITE-n
les différents éléments de cette ligne seront introduits séparés par une virgule.
En ce qui concerne les corrections des données introduites nous donnerons à l'utilisateur deux niveaux de corrections possibles :

- à l'introduction l'utilisateur aura la possibilité de corriger les erreurs détectées par le programme. Il s'agit ici soit d'erreurs syntaxiques (longueur de l'élément introduit), soit d'incompatibilités (quantité en stock insuffisante pour satisfaire la demande en produit de l'utilisateur ou produit inexistant).

- après l'introduction complète du bon de commande l'utilisateur aura la possibilité de modifier son bon de commande; c'est-à-dire qu'il pourra ajouter, modifier ou supprimer des lignes au/du bon de commande.
Afin que l'utilisateur puisse sélectionner une ligne de la commande les différentes lignes du bon de commande seront précédées par des numéros attribués par le programme.
Lorsque l'utilisateur désire modifier une ligne il devra faire précéder la modification du numéro de la ligne et d'un code indiquant l'opération (ajoute, modification, suppression) à effectuer.

.../...

Exemple :

AAAA:B:CCCCCCC ,CCCCC...

AAAA : numéro de ligne à modifier

B : opération à effectuer

soit ajoute : "A"

modification : "M"

suppression : "S"

CCC... : élément de la nouvelle ligne

b. le programme

Le programme contient trois niveaux d'appel de modules. Au premier niveau se situe la fonction ORDONN qui est appelée par le module directeur (MAIN).

La fonction ORDONN teste la valeur de l'état du dialogue et transmet le message reçu du module directeur (MAIN) à la fonction CREATION ou MODIFICATION, suivant la valeur de l'état du dialogue.

Au deuxième niveau se situent les fonctions CREATION, MODIFICATION et TRAITEMENT.

La fonction CREATION est appelée par la fonction ORDONN lors de l'introduction du bon de commande et s'occupe de la gestion des indicateurs @STEP et @STATE. Elle appelle la fonction CONTROLE. La fonction MODIFICATION est appelée par la fonction ORDONN lors de la modification (ajoute, modification, suppression) d'une ligne du bon de commande et s'occupe de la gestion des indicateurs @STEP et @STATE. Elle appelle la fonction CONTROLE. Enfin la fonction TRAITEMENT est appelée par la fonction ORDONN lorsque le bon de commande est complètement corrigé et modifié. Elle s'occupe de la gestion de l'indicateur global @STATE et transmet le bon de commande complet à la fonction EDIFACT.

Au troisième niveau se trouvent les fonctions CONTROLE et EDIFACT.

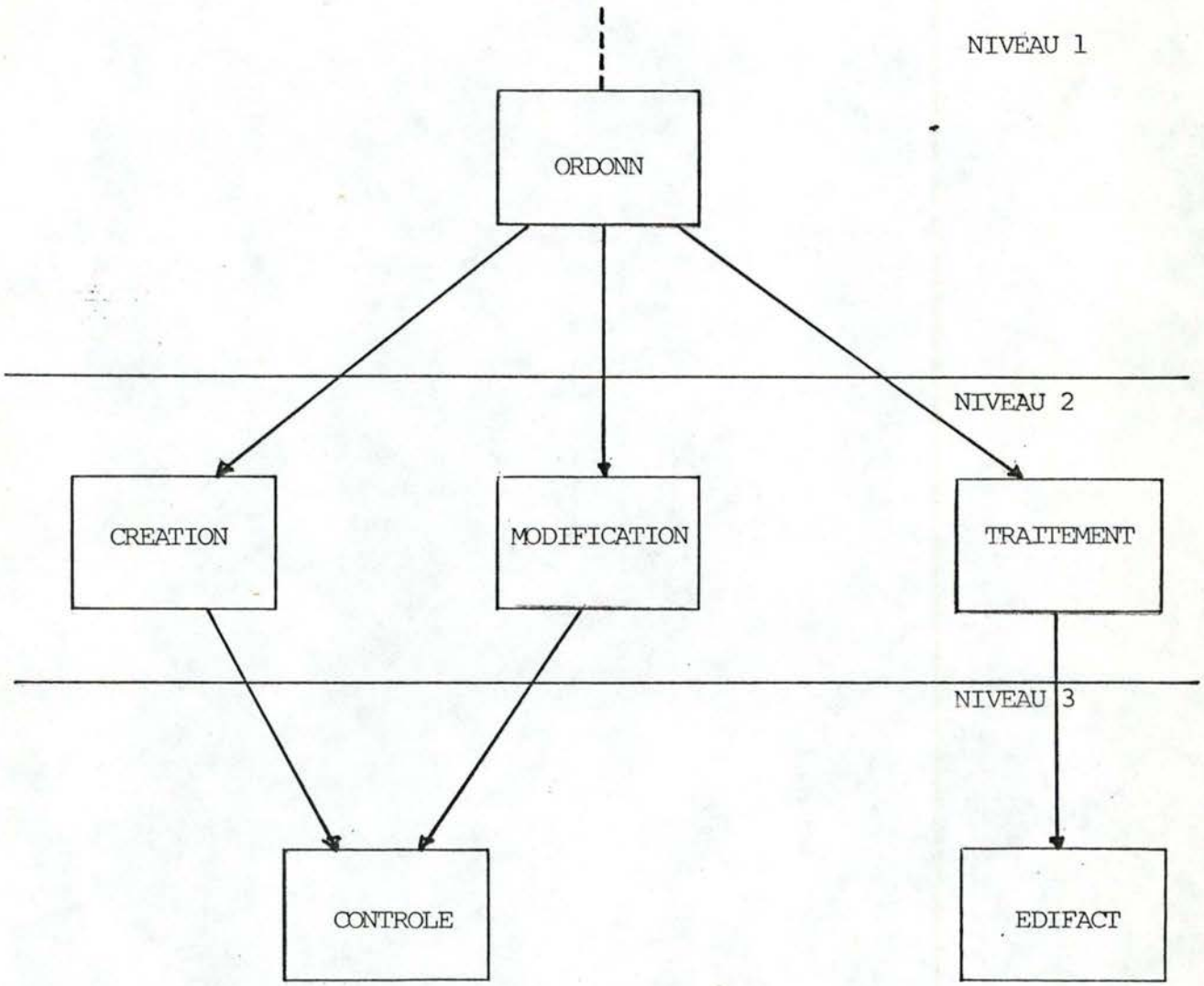
La fonction CONTROLE est appelée soit par la fonction CREATION soit par la fonction MODIFICATION. Elle effectue divers contrôles, dont nous parlerons plus loin, sur le contenu du message reçu et gère la pile des données mémorisées.

La fonction EDIFACT est appelée par la fonction TRAITEMENT et crée la facture correspondant au bon decommande transmis par la fonction TRAITEMENT.

Voir schéma page suivante.

.../...

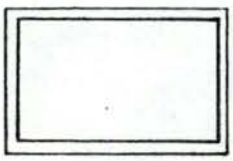
Les niveaux d'appel du programme :



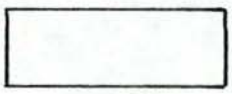
c. schéma d'enchaînement des modules

La page suivante montre le schéma d'enchaînement des modules dans l'unité de traitement.

Signification des symboles utilisés :

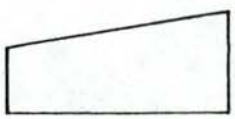


module standardisé



module spécifique au traitement

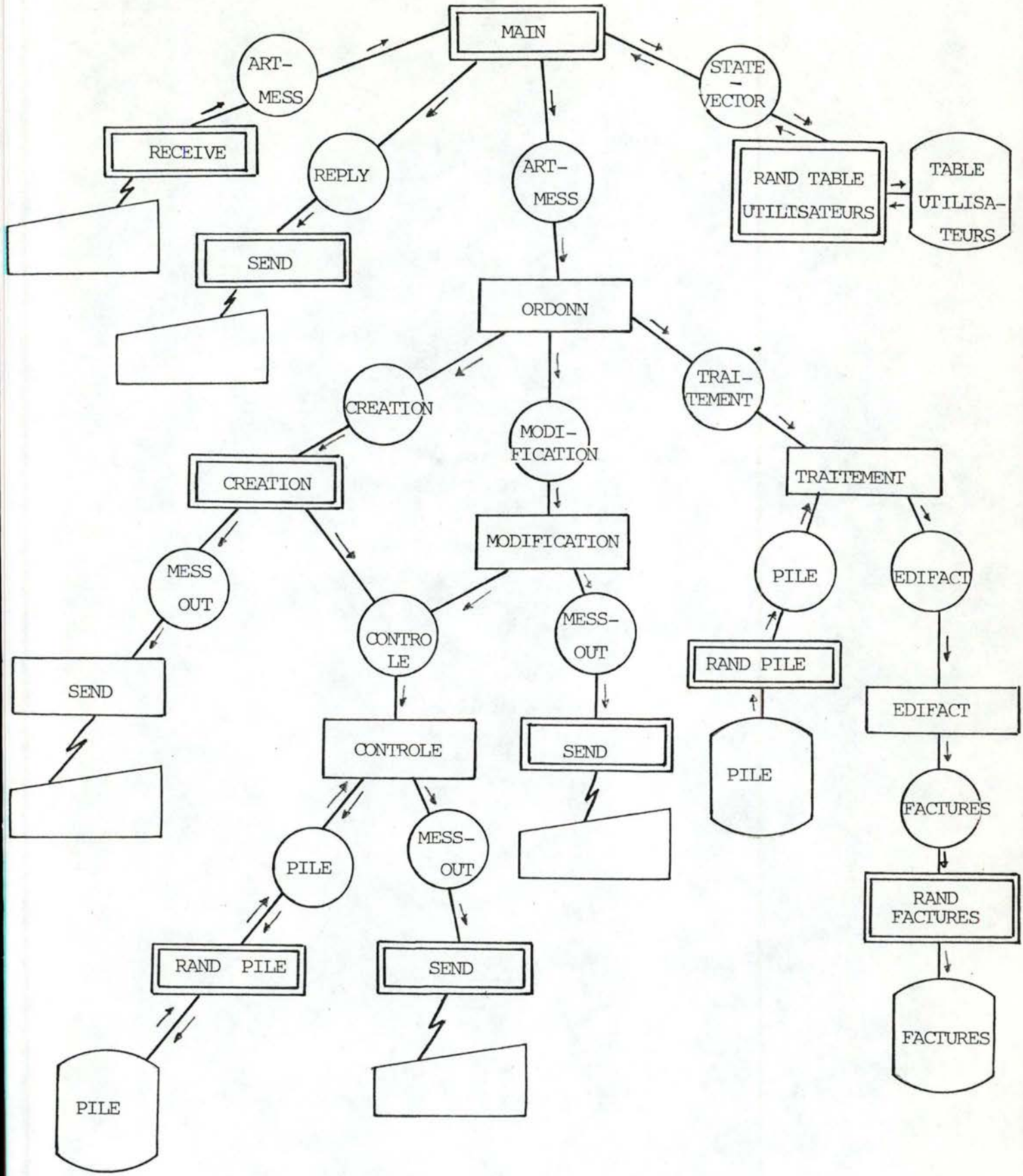
fichier virtuel



terminal

fichier réel

indicateur du flux des informations



NOTE :

Le module CONTROLE appelle les modules :

- CTRL-01
- CTRL-02

Ces modules ne sont pas repris dans le schéma pour la clarté de celui-ci.

2. Description des fonctions spécifiques

a. fonction ORDONN

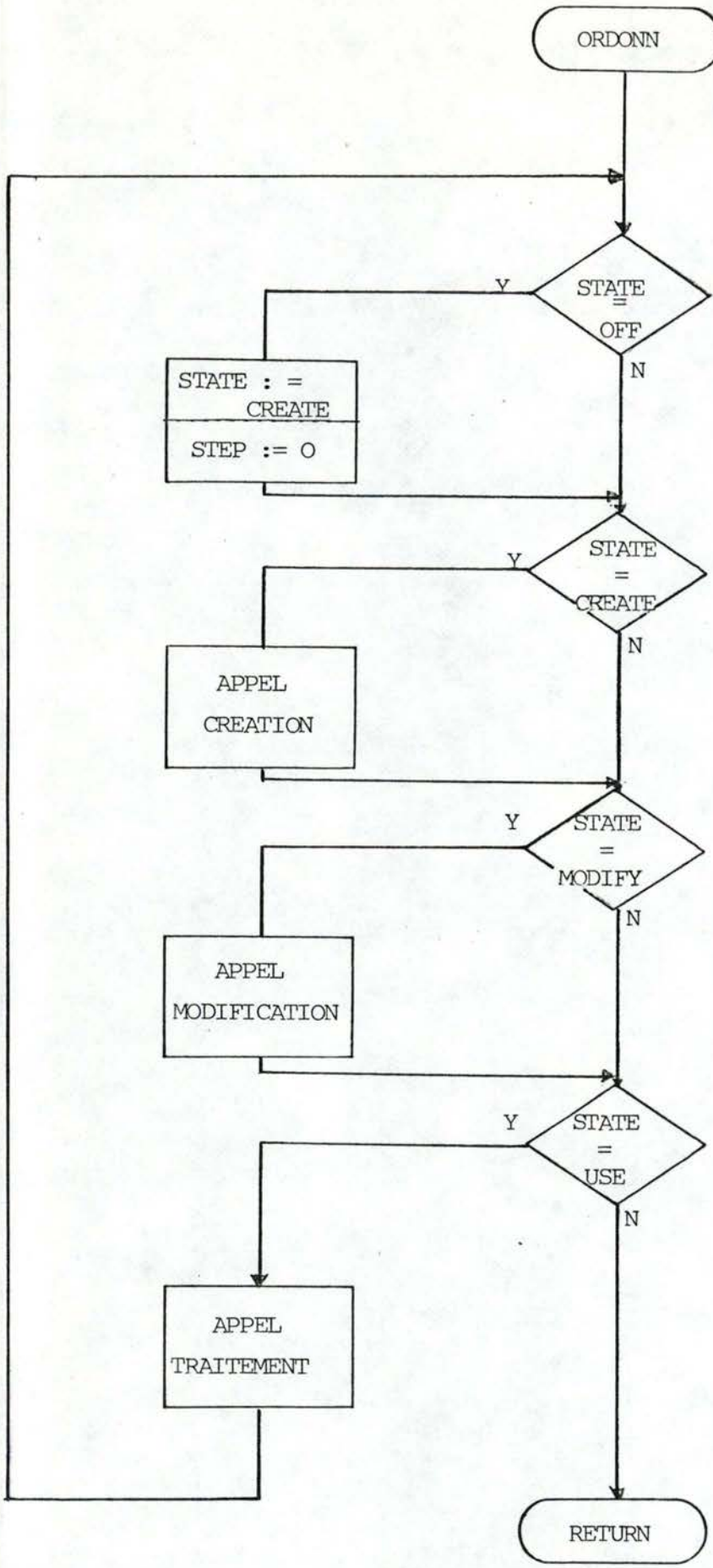
La fonction d'ordonnancement (ORDONN) teste la valeur de l'état du dialogue (@ STATE) et, suivant la valeur trouvée, appelle soit la fonction MODIFICATION soit la fonction TRAITEMENT.

La fonction d'ordonnancement ne manipule pas la valeur de l'indicateur @ STEP, sauf dans le cas où la valeur de l'état du dialogue est à "OFF" pour l'initialiser à 0.

fichier virtuel d'entrée : MESS-IN (⌘)
fichiers virtuels de sortie : CREATION (⌘)
MODIFICATION (⌘)
TRAITEMENT (⌘)

(⌘) : ces fichiers contiennent le message tel qu'il a été envoyé par l'utilisateur

TRAITEMENT est un fichier déclaré uniquement pour satisfaire aux règles de syntaxe de l'opération @WRITE.



La transition d'un état à un autre peut être provoquée par la fonction appelée, ainsi nous devons tester après l'appel si l'état du dialogue a été modifié.

PROGRAMMATION DU MODULE ORDONN.

```
 IDENTIFICATION DIVISION.
 @FUNCTION ORDONN.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 @INTERFACE SECTION.
     @FILE MESS-IN REAL-TIME LENGTH = 47.
     @FILE CREATION REAL-TIME LENGTH = 47.
     @FILE MODIFICATION REAL-TIME LENGTH = 47.
     @FILE TRAITEMENT REAL-TIME LENGTH = 1.

 PROCEDURE DIVISION.
 @CORPUS SECTION.
     @READ MESS-IN.
 @FOR EACH RECORD.

 BOUCLE.
*  utilisateur non-actif ?
     IF @STATE = OFF @SET STATE = CREATE
                     MOVE 0 TO @STEP1 @STEP2.

*  stade de creation du bon de commande ?
     IF @STATE = CREATE
        @WRITE CREATION USING MESS-IN.

*  stade de modification du bon de commande ?
     IF @STATE = MODIFY
        @WRITE MODIFICATION USING MESS-IN.

*  stade de traitement du bon de commande ?
     IF @STATE = USE
        @WRITE TRAITEMENT
        GO TO BOUCLE.

 @ENDFOR.
```

b. fonction CREATION

La fonction CREATION est appelée par la fonction ORDONN lorsque l'utilisateur est au stade d'introduction du bon de commande (@ STATE = CREATE). Elle manipule les indicateurs @STATE et @STEP.

La fonction CREATION appelle à son tour la fonction CONTROLE. Si la fonction CONTROLE détecte une erreur , la fonction CREATION demandera la réintroduction de la ligne sinon elle demandera l'introduction de la ligne suivante; dans les deux cas elle envoie d'abord au terminal le numéro identifiant la ligne, en prévision des corrections ultérieures.

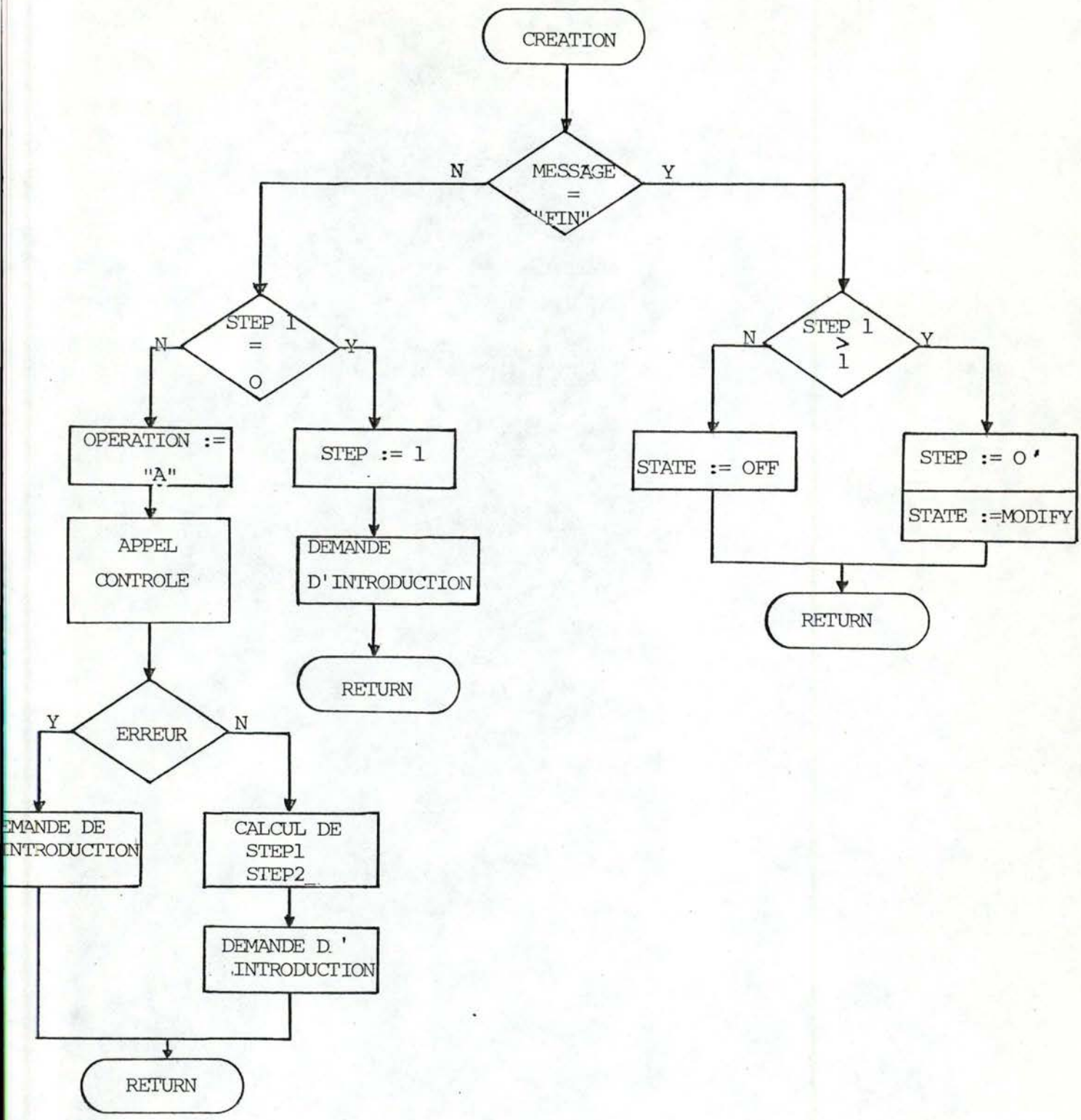
A la réception du message "FIN", la fonction provoque une transition d'état (@STATE) : passage au mode MODIFY (fin de commande) ou OFF (fin de session-utilisateur).

fichier virtuel d'entrée : -MESS-IN contenant le message de l'utilisateur.

fichiers virtuels de sortie : -MESS-OUT contenant le message pour la fonction SEND.

-CONTROLE contenant le message de l'utilisateur.

.../...


CREATION
MESSAGE = "FIN"
N
Y
STEP 1 = 0
N
Y
STEP 1 > 1
N
Y
OPERATION := "A"
STEP := 1
STATE := OFF
STEP := 0'
STATE :=MODIFY
APPEL CONTROLE
DEMANDE D'INTRODUCTION
RETURN
ERREUR
Y
N
EMANDE DE INTRODUCTION
CALCUL DE STEP1 STEP2
DEMANDE D.' INTRODUCTION
RETURN

PROGRAMMATION DU MODULE CREATION.

```
 IDENTIFICATION DIVISION.
 @FUNCTION CREATION.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 @INTERFACE SECTION.
     @FILE MESS-IN REAL-TIME LENGTH = 47.
     @FILE MESS-OUT REAL-TIME LENGTH = 80.
     @FILE CONTROLE REAL-TIME LENGTH = 41.
 WORKING-STORAGE SECTION.
*  definition des valeurs maximales des indicateurs
*  slobaux @STEP1 et @STEP2.
 01  DESCR-CDE VALUE "0299".
   02  MAX-STEP PICTURE 99 OCCURS 2.


 PROCEDURE DIVISION.
 @CORPUS SECTION.
     @READ MESS-IN.
 @FOR EACH RECORD.
*  detection de la fin de creation du bon de commande.
     IF @DATA OF MESS-IN = "FIN"
        IF @STEP1 NOT > 1 @SET STATE = OFF @EXIT
           ELSE @SET STATE = MODIFY
                MOVE 0 TO @STEP1 MOVE 0 TO @STEP2 @EXIT.

*  debut d'introduction d'un bon de commande.
     IF @STEP1 = 0 MOVE 1 TO @STEP1 MOVE 1 TO @STEP2
        @WRITE MESS-OUT = "INTRODUISEZ UNE COMMANDE OU FIN"
        GO TO DEMANDE.

*  garnissage du code operation.
     STRING "A" @DATA OF MESS-IN DELIMITED BY SIZE
        INTO @DATA OF CONTROLE.

*  appel du module CONTROLE.
     @WRITE CONTROLE.
     IF @STATUS = ERROR @RESET STATUS
        @WRITE MESS-OUT = "REINTRODUISEZ LA LIGNE"
        GO TO DEMANDE.

*  calcul de la valeur de l'indicateur slobal @STEP.
     IF @STEP2 < MAX-STEP (@STEP1) ADD 1 TO @STEP2
        ELSE ADD 1 TO @STEP1 MOVE 1 TO @STEP2.

 DEMANDE.
     @WRITE MESS-OUT USING @STEP ENABLE.
 @ENDFOR.
```

c. fonction MODIFICATION

La fonction MODIFICATION est appelée par la fonction ORDONN lorsque l'utilisateur est au stade de modification du bon de commande introduit (@ STATE = MODIFY).

L'utilisateur introduit le numéro de la ligne, suivi du code d'opération (Ajoute, Modification ou Suppression) et de valeurs de données (dans les cas "A" et "M"). La fonction devra ainsi obtenir la valeur de l'indicateur @ STEP à partir du numéro de ligne introduit ainsi que le code d'opération sur lesquels elle effectuera un contrôle de validité.

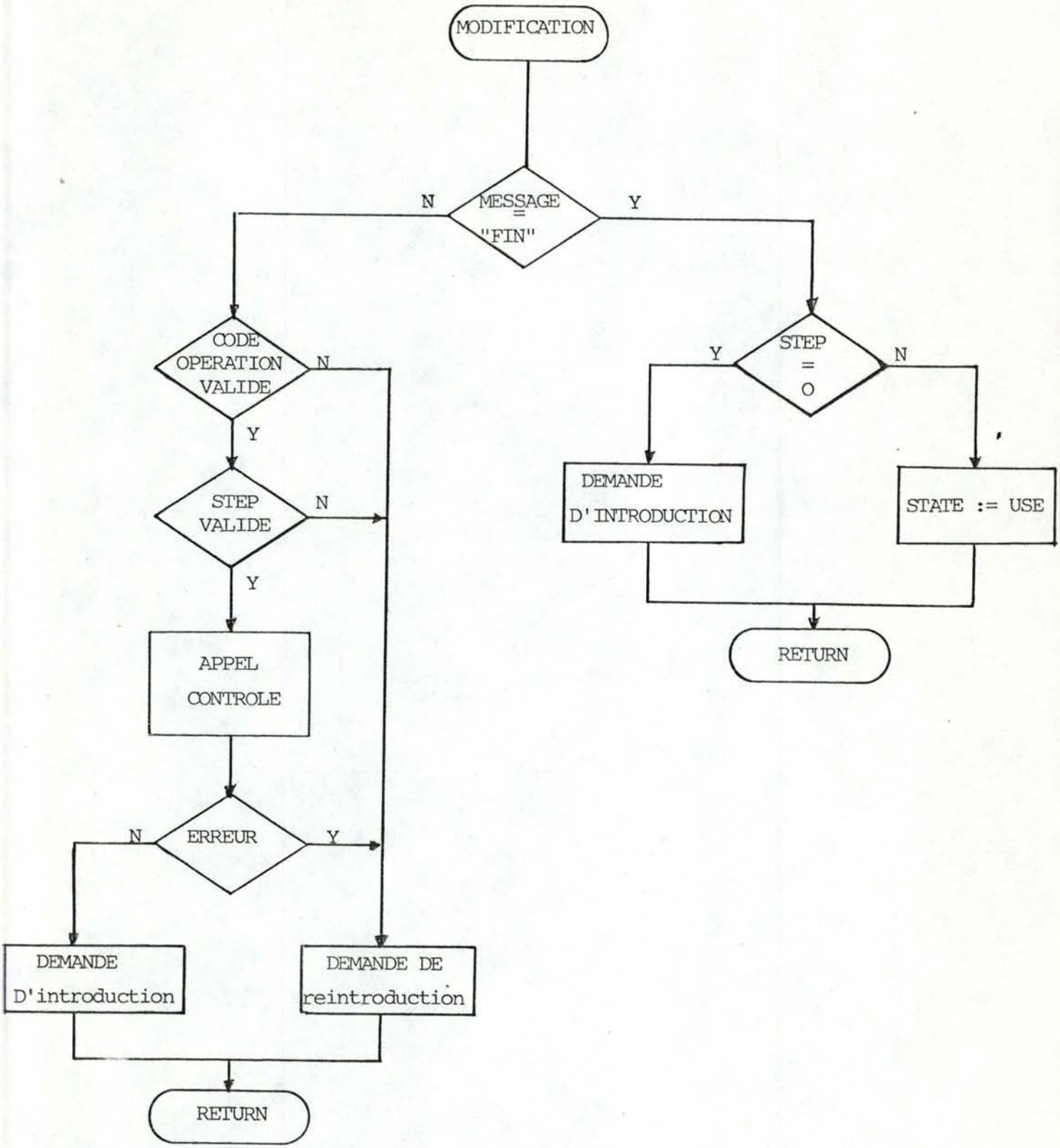
La fonction appelle ensuite la fonction CONTROLE; lorsque la fonction CONTROLE détecte une erreur, la fonction MODIFICATION demande la réintroduction de la ligne (dans les cas "A" ou "M") sinon elle demande l'introduction de la modification/ajoute/suppression suivante. Lors de la détection du message "FIN" envoyé par l'utilisateur, la fonction MODIFICATION met l'état du dialogue (@ STATE) à la valeur USE.

fichier virtuel d'entrée : -MESS-IN contenant le message envoyé par l'utilisateur.

fichiers virtuels de sortie :-MESS-OUT contenant le message pour la fonction SEND

-CONTROLE contenant le message envoyé par l'utilisateur.

.../...


MODIFICATION
MESSAGE = "FIN"
N
Y
CODE OPERATION VALIDE
N
Y
STEP VALIDE
N
Y
APPEL CONTROLE
ERREUR
N
Y
DEMANDE D'introduction
DEMANDE DE reintroduction
RETURN
STEP = O
Y
N
DEMANDE D'INTRODUCTION
STATE := USE
RETURN

PROGRAMMATION DU MODULE MODIFICATION.

```
 IDENTIFICATION DIVISION.
 @FUNCTION MODIFICATION.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 @INTERFACE SECTION.
     @FILE MESS-IN REAL-TIME LENGTH = 47.
     @FILE MESS-OUT REAL-TIME LENGTH = 80.
     @FILE CONTROLE REAL-TIME LENGTH = 41.
 WORKING-STORAGE SECTION.
*  delimiteur pour l'instruction UNSTRING.
 77  COLON PICTURE X VALUE ":".
*  valeur maximale de l'indicateur global @STEP.
 77  VAL-STEP PICTURE 9999 VALUE "0299".
*  code operation (AJoute,Modification,Suppression).
 77  OPERATION PICTURE X.
*  definition d'une zone de travail REAL-TIME pour
*  les instructions STRING/UNSTRING.
 @RECORD TEXTE REAL-TIME LENGTH = 40.

 PROCEDURE DIVISION.
 @CORPUS SECTION.
     @READ MESS-IN.
 @FOR EACH RECORD.
*  detection de la fin de modification d'un bon de commande.
     IF @DATA OF MESS-IN = "FIN"
        IF @STEP > 0 @SET STATE = USE @EXIT
           ELSE GO TO DEMANDE.

     UNSTRING @DATA OF MESS-IN DELIMITED BY COLON
              INTO @STEP
                   OPERATION
                   @DATA OF TEXTE COUNT IN
                                @LENGTH OF CONTROLE.

*  controle du code operation.
     IF OPERATION NOT = "A" AND "M" AND "S"
        @WRITE MESS-OUT = "OPERATION INVALIDE" ENABLE
        @EXIT.

*  controle du numero de ligne.
     IF @STEP > VAL-STEP
        @WRITE MESS-OUT = "NUMERO DE LIGNE INVALIDE"
        ENABLE @EXIT.

*  appel du module CONTROLE.
     STRING OPERATION @DATA OF TEXTE DELIMITED BY SIZE
            INTO @DATA OF CONTROLE.
     @WRITE CONTROLE
     IF @STATUS = ERROR @RESET STATUS
     @WRITE MESS-OUT = "MODIFICATION ERRONEE" ENABLE @EXIT.

 DEMANDE.
     @WRITE MESS-OUT = "INTRODUISEZ UNE MODIFICATION OU FIN"
     ENABLE @EXIT.
 @ENDFOR.
```

d. fonction CONTROLE

La fonction CONTROLE est appelée par la fonction CREATION et la fonction MODIFICATION.

La fonction CONTROLE examine d'abord le code d'opération contenu dans le fichier CTRL-IN; si le code est égal à "S" (suppression) la fonction extrait la ligne correspondant à la valeur de la clé (@USER@STEP) de la pile de mémorisation des données (PILE), sinon elle procède au contrôle des données reçues.
Le tableau suivant donne une vue d'ensemble sur les contrôles effectués pour chaque type d'article du fichier CTRL-IN.

ARTICLE		CONTROLES	
TYPE	CONTENU	SYNTAXIQUE	AUTRES
NOM	nom	longueur	
ADRESSE	adresse	longueur	
LOCALITE	nr-postal	longueur	
	localité		
LIGNE	nr-produit	longueur	existence du numéro de produit
	quantité	longueur	quantité suffisante pour satisfaire
		numérique	la demande
		>0	

A chaque commande de produit acceptée la fonction met à jour le fichier STOCKS contenant les quantités disponibles. Lorsqu'il s'agit d'une Ajoute la fonction insère la donnée concernée dans la pile, s'il s'agit d'une Modification la fonction recherche la ligne concernée dans la pile et la remplace avec les nouvelles valeurs de données.

fichier virtuel d'entrée : -CTRL-IN contenant le message envoyé par l'utilisateur.

fichier virtuel d'entrée-sortie :

- PILE contenant les articles
  -TETE-CDE : NOM, ADRESSE, LOCALITE
  -CORPS-CDE : NR-PRODUIT, QUANTITE
- STOCKS contenant l'article
  -F-STOCK : NR-PRODUIT, QUANTITE
- MESS-OUT contenant les mesaages pour la fonction SEND.

.../...

PROGRAMMATION DU MODULE CONTROLE.

```
IDENTIFICATION DIVISION.
@FUNCTION CONTROLE.
ENVIRONMENT DIVISION.
DATA DIVISION.
@INTERFACE SECTION.
    @FILE CTRL-IN REAL-TIME LENGTH = 41.
    @FILE CTRL-01 REAL-TIME LENGTH = 41.
    @FILE CTRL-02 REAL-TIME LENGTH = 41.

PROCEDURE DIVISION.
@CORPUS SECTION.
    @READ CTRL-IN.

@FOR EACH RECORD.
    IF @STEP1 = 1 @WRITE CTRL-01 USING CTRL-IN.
    IF @STEP1 = 2 @WRITE CTRL-02 USING CTRL-IN.
@ENDFOR.
```

NOTE:

Le module CONTROLE appelle les modules :

- CTRL-01 pour controler une ligne d'entete d'un bon de commande.
- CTRL-02 pour controler une ligne de corps d'un bon de commande.

PROGRAMMATION DU MODULE CTRL-01.

```
 IDENTIFICATION DIVISION.
 @FUNCTION CTRL-01.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 @INTERFACE SECTION.
     @FILE CTL1-IN REAL-TIME LENGTH = 41.
     @FILE PILE TYPE COMMANDES FOR REAL-TIME.
     @FILE MESS-OUT REAL-TIME LENGTH = 80.
 WORKING-STORAGE SECTION.
*  zone de travail pour l'instruction UNSTRING.
 77  LONG PICTURE 99.
*  zone de sauvetage pour l'indicateur global @STEP2.
 77  S-STEP2 PICTURE 99.
*  zone de travail contenant le code operation (Ajoute,
*  Modification,Suppression).
 77  OPERATION PICTURE X.
*  definition d'une zone REAL-TIME utilisee par l'instruction
*  UNSTRING.
 @RECORD TEXTE REAL-TIME LENGTH = 40.
*  definition des longueurs maximales des lignes d'entete
*  du bon de commande.
 01  V-MAX-L VALUE "204031".
   02  MAX-L PICTURE 99 OCCURS 3.

 PROCEDURE DIVISION.
 @CORPUS SECTION.
     @READ CTL1-IN.
 @FOR EACH RECORD.
     UNSTRING @DATA OF CTL1-IN
              INTO OPERATION
                   @DATA OF TEXTE COUNT IN LONG.

*  controle de la longueur de la ligne d'entete introduite.
     IF LONG > MAX-L (@STEP2)
        @WRITE MESS-OUT = "DONNEE TROP LONGUE"
        @SET STATUS = ERROR @EXIT.

*  sauvetage de la valeur de l'indicateur global @STEP2.
     MOVE @STEP2 TO S-STEP2 MOVE 0 TO @STEP2.

*  recherche de la ligne dans la pile des donnees
*  memorisees.
     @OBTAIN RECORD PILE KEY = @USER @STEP.

     IF S-STEP2 = 1 MOVE @DATA OF TEXTE TO NOM.
     IF S-STEP2 = 2 MOVE @DATA OF TEXTE TO ADRESSE.
     IF S-STEP2 = 3 MOVE @DATA OF TEXTE TO LOCALITE.

*  lorsque la ligne n'existe pas encore dans la pile (ERROR)
*       ---> insertion de la ligne
*  sinon ---> remplacement de la ligne.
     IF @STATUS = ERROR @RESET STATUS
        @INSERT PILE KEY = @USER @STEP
        ELSE @REPLACE PILE KEY = @USER @STEP.

*  restauration de la valeur de l'indicateur global @STEP2.
     MOVE S-STEP2 TO @STEP2.
 @ENDFOR.
```

PROGRAMMATION DU MODULE CTRL-02.

```
 IDENTIFICATION DIVISION.
 @FUNCTION CTRL-02.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 @INTERFACE SECTION.
     @FILE CTL2-IN REAL-TIME LENGTH = 41.
     @FILE MESS-OUT REAL-TIME LENGTH = 80.
     @FILE PILE TYPE COMMANDES FOR REAL-TIME.
     @FILE STOCKS TYPE STOCK.
 WORKING-STORAGE SECTION.
*  zone de travail contenant le code operation (Ajoute,
*  Modification,Suppression).
  77  OPERATION PICTURE 99.
*  definition d'une zone de travail pour
*  l'instruction UNSTRING.
 @RECORD LIGNE-PROPOSEE TYPE CORPS-CDE.

 PROCEDURE DIVISION.
 @CORPUS SECTION.
     @READ CTL2-IN.

 @FOR EACH RECORD.


*  controle de la longueur.
     IF @LENGTH OF CTL2-IN > 15
        @WRITE MESS-OUT = "LIGNE TROP LONGUE"
        @SET STATUS = ERROR @EXIT.

     UNSTRING @DATA OF CTL2-IN
              INTO OPERATION
                   LIGNE-PROPOSEE.

*  Suppression de la ligne ?
     IF OPERATION = "S" MOVE 0 TO LIGNE-PROPOSEE
        ELSE @OBTAIN RECORD STOCKS KEY = NR-PRODUIT FROM
                                              LIGNE-PROPOSEE
             IF @STATUS = ERROR
                @WRITE MESS-OUT = "PRODUIT INEXISTANT" @EXIT.

*  obtention de la ligne dans la pile des donnees memorisees,
*  si ERROR  ---> operation d'Ajoute.
     @OBTAIN RECORD PILE KEY = @USER @STEP
     IF @STATUS = ERROR @RESET STATUS
        MOVE 0 TO QUANTITE OF PILE
        ELSE IF NR-PRODUIT OF PILE NOT = NR-PRODUIT OF LIGNE-
                PROPOSEE PERFORM ADD-STOCK.

*  calcul de la quantite residuelle en stock.
     COMPUTE QUANTITE OF STOCKS = QUANTITE OF STOCKS +
         QUANTITE OF PILE - QUANTITE OF LIGNE-PROPOSEE.
```

```
*  controle de la quantite residuelle en stock.
     IF QUANTITE OF STOCKS < 0
        @WRITE MESS-OUT = "STOCK INSUFFISANT"
        @SET STATUS = ERROR @EXIT.

*  mise a jour du stock.
     @REPLACE STOCKS KEY = NR-PRODUIT FROM LIGNE-PROPOSEE.

*  mise a jour de la pile des donnees memorisees (A ou M).
     IF OPERATION = "A"
        @INSERT PILE USING LIGNE-PROPOSEE KEY = @USER @STEP
        ELSE @REPLACE PILE USING LIGNE-PROPOSEE
                                 KEY = @USER @STEP.
 @ENDFOR.


 @SUBPARTS SECTION.
*  retablissement du niveau de stock avant toute autre
*  modification.
 ADD-STOCK.
     @OBTAIN RECORD STOCKS KEY = NR-PRODUIT OF PILE.
     COMPUTE QUANTITE OF STOCKS = QUANTITE OF STOCKS
                                + QUANTITE OF PILE.
     @REPLACE RECORD STOCKS KEY = NR-PRODUIT OF PILE.
     IF OPERATION = "M"
        MOVE 0 TO QUANTITE OF PILE
        @OBTAIN RECORD STOCKS KEY = NR-PRODUIT
                                    FROM LIGNE-PROPOSEE
        ELSE @DELETE PILE KEY = @USER @STEP
             @EXIT.
```

e. fonction TRAITEMENT

La fonction TRAITEMENT est appelée par la fonction ORDONN lorsque la valeur de l'état du dialogue (@ STATE) est égale à USE.

Cette fonction extrait toutes les lignes du bon de commande de la pile de mémorisation des données et les passe à la fonction EDIFACT qui éditera la facture correspondant au bon de commande.

fichiers virtuels d'entrée :

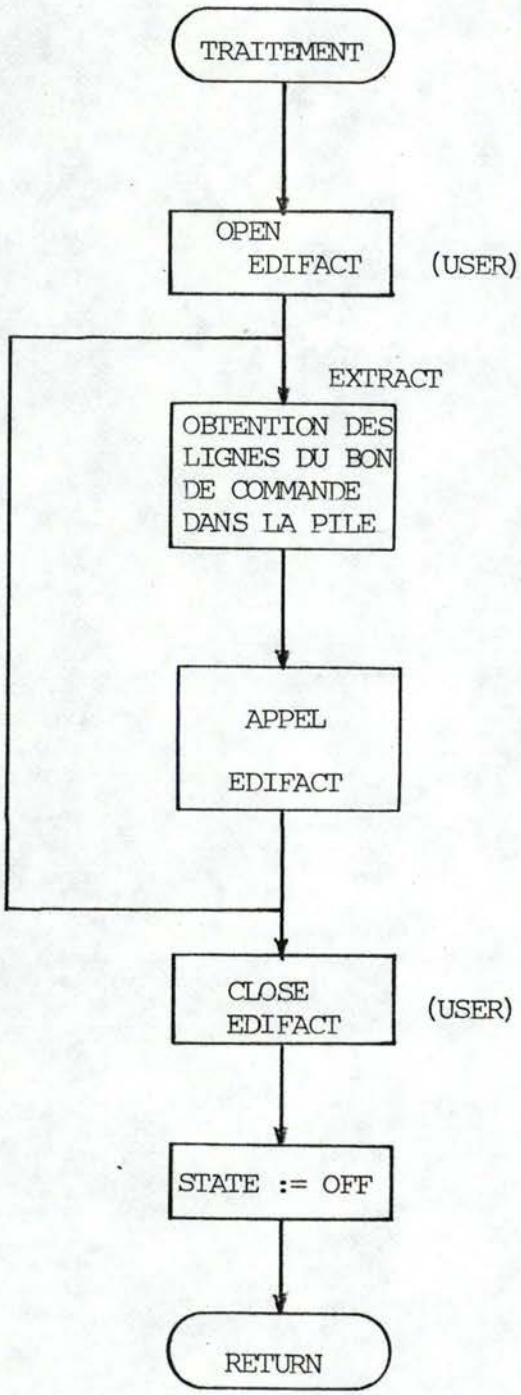
- ENTREE (*)
- PILE contenant les articles
  - TETE-CDE : NOM, ADRESSE, LOCALITE
  - CORPS-CDE : NR-PRODUIT, QUANTITE

fichier virtuel de sortie :

- EDIFACT contenant les articles
  - TETE-CDE : NOM, ADRESSE, LOCALITE
  - CORPS-CDE : NR-PRODUIT, QUANTITE

(*) : fichier déclaré uniquement pour satisfaire aux règles de syntaxe de l'opération @READ.

.../...


TRAITEMENT
OPEN
EDIFACT
(USER)
EXTRACT
OBTENTION DES
LIGNES DU BON
DE COMMANDE
DANS LA PILE
APPEL
EDIFACT
CLOSE
EDIFACT
(USER)
STATE := OFF
RETURN

PROGRAMMATION DU MODULE TRAITEMENT.

```
IDENTIFICATION DIVISION.
@FUNCTION TRAITEMENT.
ENVIRONMENT DIVISION.
DATA DIVISION.
@INTERFACE SECTION.
    @FILE ENTREE REAL-TIME LENGTH = 1.
    @FILE PILE TYPE COMMANDES FOR REAL-TIME.
    @FILE EDIFACT TYPE COMMANDES FOR REAL-TIME.

PROCEDURE DIVISION.
@CORPUS SECTION.
    @READ ENTREE.

@FOR EACH RECORD.
    @OPEN EDIFACT FOR USER.
    @EXTRACT ALL PILE.

@FOR EACH RECORD.
    @WRITE EDIFACT USING PILE.
@ENDFOR.

    @CLOSE EDIFACT FOR USER.
    @RESET STATUS.
@ENDFOR.
```

f. fonction EDIFACT

La fonction EDIFACT est appelée par la fonction TRAITEMENT.

La fonction écrit l'entête de la commande dans le fichier FACTURES et calcule ensuite le prix, à l'aide du fichier TARIF, pour chaque ligne du corps de la commande. Après le calcul du prix la ligne du corps de la commande est écrite dans le fichier FACTURES.
A la fin du traitement de la commande la fonction ajoute une ligne contenant le total de la facture dans le fichier FACTURES.

fichiers virtuels d'entrée :

- CDES contenant les articles
  -TETE-CDE : NOM, ADRESSE, LOCALITE
  -CORPS-CDE : NR-PRODUIT, QUANTITE

- TARIF contenant un prix pour chaque numéro de produit :
  NR-PRODUIT, PRIX-UN

- DERN-NO-FACT contenant un numéro de facture

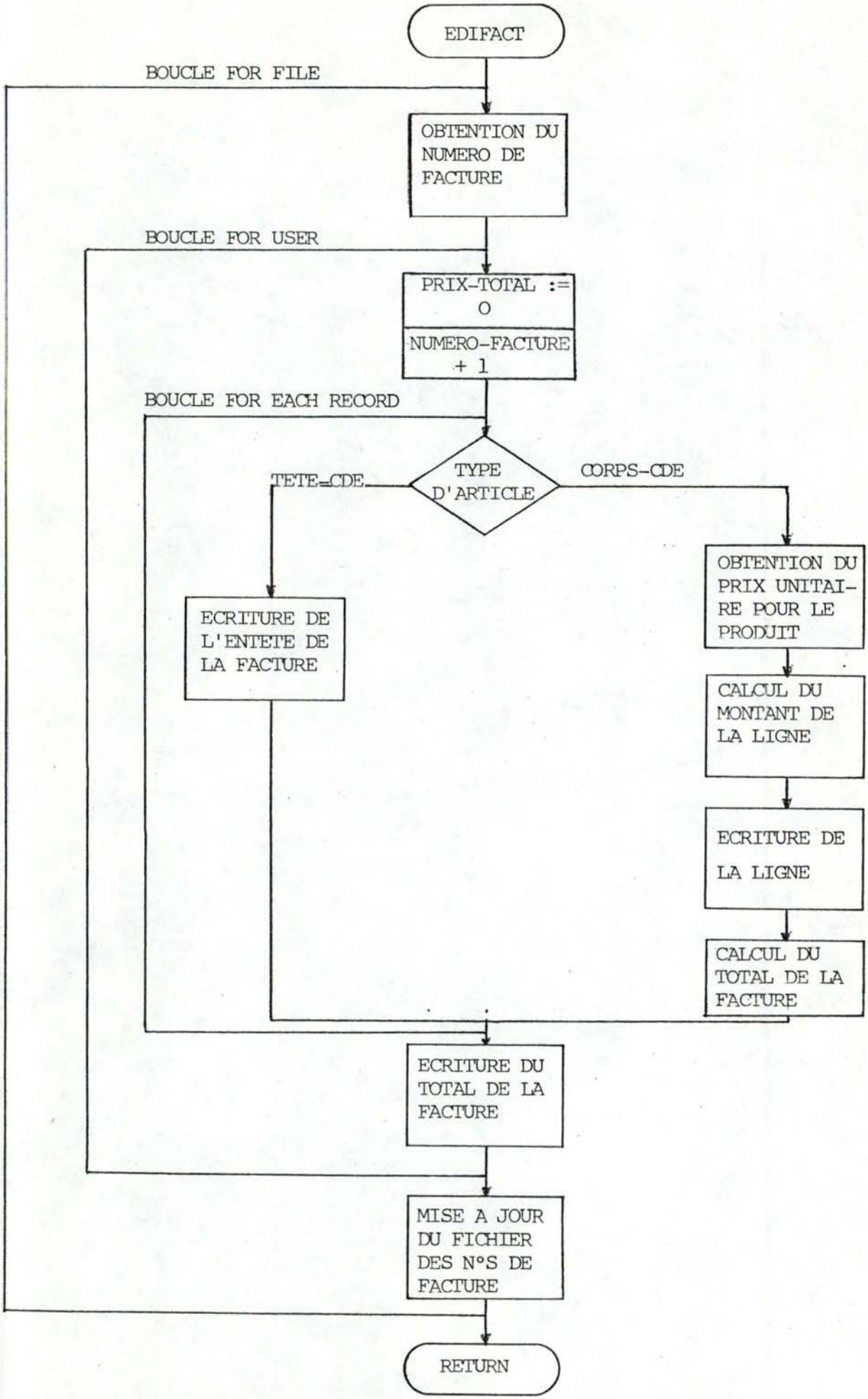
fichier virtuel de sortie :

- FACT contenant les articles

  -TETE-FACT : NR-FTR, NOM, ADRESSE, LOCALITE
  -CORPS-FACT : NR-PRODUIT, QUANTITE, PRIX-FACT
  -FINALE : TOTAL-FACT

-DERN-NO-FACT contenant un numéro de facture

.../...


EDIFACT
BOUCLE FOR FILE
OBTENTION DU NUMERO DE FACTURE
BOUCLE FOR USER
PRIX-TOTAL := 0
NUMERO-FACTURE + 1
BOUCLE FOR EACH RECORD
TYPE D'ARTICLE
TETE-CDE
CORPS-CDE
ECRITURE DE L'ENTETE DE LA FACTURE
OBTENTION DU PRIX UNITAIRE POUR LE PRODUIT
CALCUL DU MONTANT DE LA LIGNE
ECRITURE DE LA LIGNE
CALCUL DU TOTAL DE LA FACTURE
ECRITURE DU TOTAL DE LA FACTURE
MISE A JOUR DU FICHIER DES N°S DE FACTURE
RETURN

PROGRAMMATION DU MODULE EDIFACT.

```
IDENTIFICATION DIVISION.
@FUNCTION EDIFACT.
ENVIRONMENT DIVISION.
DATA DIVISION.
@INTERFACE SECTION.
    @FILE CDES TYPE COMMANDES FOR REAL-TIME.
    @FILE FACT TYPE FACTURES.
    @FILE TARIF TYPE TARIFS.
    @FILE DERN-NO-FACT TYPE NR-FACT.

PROCEDURE DIVISION.
    @READ CDES.

@FOR FILE.
    @OBTAIN DERN-NO-FACT.

@FOR EACH USER.
    MOVE 0 TO PRIX-TOTAL.
    ADD 1 TO DERN-NO-FACT.

@FOR EACH RECORD IF TYPE = TETE-CDE.
    @WRITE FACT TYPE = TETE-FACT.
@ENDFOR.

@FOR EACH RECORD IF TYPE = CORPS-CDE.
    @OBTAIN RECORD TARIF KEY = NO-PROD FROM CDES.
    COMPUTE PRIX-FACT = PRIX-UN * QTE-CDE.
    @WRITE FACT TYPE = CORPS-FACT.
    ADD PRIX-FACT TO TOTAL-FACT.
@ENDFOR.

    @WRITE FACT TYPE = FINALE.
@ENDFOR.

    @REPLACE DERN-NO-FACT.
@ENDFOR.
```

ANNEXE.

LA GESTION DU TELETRAITEMENT EN COBOL.

A.1. Principes de la réalisation COBOL.

Le module "télétraitement" du langage COBOL fournit plusieurs instructions nécessaires au programme d'application pour pouvoir travailler en temps réel.
L'exécution de ces instructions suppose l'existence d'un moniteur de télétraitement qui s'occupe de la gestion du réseau de télétraitement.
Ce moniteur doit fournir au programme d'application :

- une image logique du réseau de télétraitement, quelle que soit l'organisation réelle de ce réseau.
- une structuration des données transitant dans le réseau.

Le moniteur et les programmes d'application communiquent par le biais de copies de blocs de contrôle définies dans le programme d'application.

A.2. Les concepts du télétraitement en COBOL.

a. la structure logique du réseau.

La structure logique du réseau est composée de deux sous-structures indépendantes : la sous-structure des files d'attente d'entrée et la sous-structure des files d'attente de sortie. Un message se trouve soit dans la structure d'entrée, soit en mémoire interne du programme d'application, soit dans la structure de sortie.
Les files d'attente d'entrée sont hiérarchisées en "queues" et "subqueues". A une file d'attente d'entrée, définie par "queue" et "subqueue", sont reliés un ou plusieurs terminaux. Le programme d'application doit au moins identifier la "queue" qui l'intéresse, il peut en outre préciser jusqu'à trois niveaux de "subqueues".
Le programme d'application ne désigne pas explicitement les files d'attente de sortie mais bien les terminaux destinataires connus par leur nom symbolique. C'est au moniteur d'organiser ces files d'attente en fonction des destinations visées.

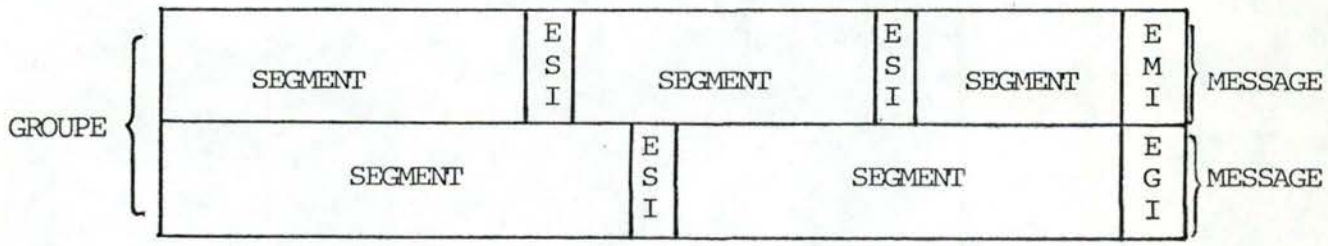
.../...

b. la structure des données.

L'ensemble des données transitant dans le réseau de télétraitement est hiérarchisé en trois niveaux, à savoir : les niveaux de groupe, de message et de segment.
Pour que le moniteur de télétraitement puisse identifier à quel niveau un texte (*) appartient, des "des indicateurs de fin" accompagnent le texte transmis.
Ainsi des caractères de contrôle identifiant le niveau hiérarchique sont compris dans les données transmises entre les terminaux et les files d'attente. Par contre ces caractères n'existent pas dans les textes transitant entre le moniteur et le programme d'application mais leur équivalent logique est repris dans la copie du bloc de contrôle sous forme d'un indicateur prenant les valeurs suivantes :

0 = pas d'indication (portion de message non clôturée).
1 = fin de segment (End of Segment Indicator).
2 = fin de message (End of Message Indicator).
3 = fin de groupe (End of Group Indicator).

Exemple de structuration des données :



Remarque :

Un indicateur de fin de message sert également de fin du dernier segment dans le message; un indicateur de fin de groupe sert également d'indicateur de fin du dernier message et du dernier segment dans le groupe.

(*) texte : sous-ensemble de l'ensemble des données transitant dans le réseau de télétraitement.

.../...

Enfin voici la signification de chaque niveau :

- SEGMENT : le segment est l'unité physique de transfert entre une file d'attente et un terminal (ex. : dans le cas d'un terminal à écran un segment correspond à une ligne de cet écran).
- MESSAGE : un message est la quantité d'information dont le transfert est synchronisé; aucune donnée n'est extraite d'une file d'attente, que ce soit une file d'attente d'entrée ou de sortie, tant que le message n'est point complet, le message est le seul niveau obligatoire.
- GROUPE : le groupe est un ensemble de messages défini en vue de l'ordonnancement des traitements.

L'exemple suivant fera mieux comprendre la signification de chaque niveau :

Enregistrement de bons de commande :

A)	1° ligne d'en-tête	-ESI-
	2° ligne d'en-tête	-EMI-
	ligne de commande	-EMI-
	ligne de commande	-EMI-
	.	
	.	
	ligne de commande	-EGI-

B)	1° ligne d'en-tête	-ESI-

La structure adoptée ci-dessus pour les bons de commande permettra de les traiter de la façon suivante. L'en-tête d'une commande étant complète (signal EMI), le programme peut en vérifier la validité. Chaque ligne de la commande peut-être successivement vérifiée, ce qui permettra de signaler les erreurs et d'en faire donner la correction aussitôt, sans que l'opérateur du terminal ne soit obligé de rédiger complètement sa commande avant de pouvoir corriger une erreur à la première ligne.

.../...

Lorsque la fin du groupe (fin de la commande) est atteinte, le programme pourra renvoyer au terminal un accusé de réception, une facture, ...; ces derniers seront, eux, contitués d'un seul message.

c. les blocs de contrôle.

Pour la gestion du réseau, le moniteur de télétraitement a besoin d'une série d'indicateurs. Certains de ces indicateurs doivent être accessibles au programme d'application qui dispose à cet effet de la copie d'une partie des blocs de contrôle du moniteur.
Les zones de mémoire nécessaires aux copies des blocs de contrôle sont déclarées dans la DATA DIVISION et plus précisement sous la rubrique CD en COMMUNICATION SECTION.

Voici un aperçu des zones des blocs de contrôle :

- bloc de contrôle d'entrée.

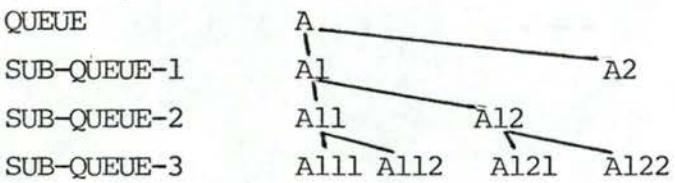
*	ZONE	CONTENU
pm	SYMBOLIC QUEUE	Zone destinée à recevoir les noms symboliques identifiant un sous-ensemble des files d'attente d'entrée.
pm	SYMBOLIC SUB-QUEUE-1	
pm	SYMBOLIC SUB-QUEUE-2	
pm	SYMBOLIC SUB-QUEUE-3	
m	MESSAGE DATE	Date de réception du message par le moniteur.
m	MESSAGE TIME	Heure de réception du message par le moniteur.
pm	SYMBOLIC SOURCE	Nom symbolique des terminaux d'entrée.
m	TEXT LENGTH	Nombre de caractères constituant le texte transmis au programme d'application.
m	END KEY	Borne du texte transmis au programme d'application : 0, 1, 2 ou 3
m	STATUS KEY	Code d'erreur si l'opération sur la structure d'entrée n'a pas réussi.
m	QUEUE DEPTH	Nombre de messages complets présents dans le sous-ensemble des files d'attente désigné.

.../...

Les zones marquées m sont garnies par le moniteur au terme de ses interventions. Les zones marquées pm, qui servent à identifier la structure d'entrée concernée, sont garnies par le programme d'application préalablement à l'exécution des instructions adressées au moniteur et ce dernier en précise s'il y a lieu le contenu au terme de ses interventions.

Exemples de désignation d'un sous-ensemble de la structure d'entrée :

Soit la structure d'entrée suivante :



Si l'on veut s'adresser au sous-ensemble A12, on devra garnir les zones du bloc de contrôle de la façon suivante :

QUEUE = 'A', SUB-QUEUE-1 = 'A1', SUB-QUEUE-2 = 'A12',
SUB-QUEUE-3 = SPACES.

- bloc de contrôle de sortie.

*	ZONE	CONTENU
p	DESTINATION COUNT	Nombre de destinations visées par l'instruction référençant le bloc de contrôle.
p	TEXT LENGTH	Nombre de caractères constituant le texte transmis au (x) terminal(-aux).
m	STATUS KEY	Code d'erreur si l'opération sur la structure de sortie n'a pas réussi.
	DESTINATION TABLE OCCURS n TIMES	Table reprenant n couples (ERROR KEY, SYMBOLIC DESTINATION).
m	ERROR KEY	Si = '0' : opération réussie. Si = '1' : erreur; la destination symbolique associée est inconnue du moniteur.
p	SYMBOLIC DESTINATION	nom symbolique des terminaux destinataires.

.../...

Les zones marquées m sont garnies par le moniteur au terme de ses interventions; les zones marquées p sont garnies par le programme utilisateur préalablement à l'exécution des instructions données au moniteur.

Remarque :

Le programme d'application devra contenir un bloc de contrôle d'entrée (unique) déclaré FØR INITIAL INPUT, qui sera utilisé pour initialiser le programme d'application.
Lorsque le programme d'application est initialisé par un ordre du "Job Control Language", ce bloc des garni de valeurs nulles.
Lorsque le programme est appelé par un terminal (ayant à cette fin envoyé un message au superviseur), le contenu du bloc identifie l'origine du message.

A.3. Description des instructions de télétraitement.

a. instruction RECEIVE

format :

RECEIVE nom-de-CD {MESSAGE / SEGMENT} INTØ identificateur

[NØ DATA instruction impérative]

Cette instruction ôte le premier message ou segment disponible des files d'attente de la sous-structure d'entrée désignée et elle le transfère dans la zone interne au programme désignée par 'identificateur'.
La structure d'entrée est définie dans le bloc de contrôle référencé sous nom-de-CD.
Les zones SYMBOLIC QUEUE et SYMBOLIC SUB-QUEUE doivent être garnies avant l'exécution de l'instruction RECEIVE. Les zones SYMBOLIC SUB-QUEUE peuvent éventuellement être garnies de blancs.
S'il n'y a pas de message complet disponible dans la file d'attente et si l'incise NØ DATA n'est pas programmée le programme est mis en attente jusqu'à ce que le moniteur puisse satisfaire la demande de transfert.
Par contre, si l'incise NØ DATA est programmée, l'instruction impérative est exécutée.

.../...

b. instruction SEND

format :

SEND nom-de-CD [FROM identificateur-1]

WITH { identificateur-2 / ESI / EMI / EGI }

[{ BEFORE / AFTER } ADVANCING { { identificateur-3 / entier } { LINE / LINES } / nom-mnémonique / PAGE }]

L'instruction SEND déclenche le transfert des données contenues dans la zone identificateur-1, qui est interne au programme d'application, vers les files d'attente de sortie. Un identificateur de fin de segment (ESI), de message (EMI), ou de groupe (EGI) désigné par l'incise WITH est transféré en même temps que les données. La valeur correspondante de ces indicateurs, c'est-à-dire dans l'ordre les valeurs '1', '2', ou '3', peut également être mémorisée dans la zone identificateur-2 du programme.

Dans le cas où l'on désire une mise en page on utilisera l'incise BEFORE/AFTER ADVANCING. Cette incise est sans effet si le terminal visé par l'instruction SEND ignore le concept de "page".

Les zones du bloc de contrôle de sortie, référencé sous nom-de-CD, qui doivent être garnies préalablement à l'exécution de l'instruction sont : DESTINATION COUNT, TEXT LENGTH et SYMBOLIC DESTINATION.

c. instruction ENABLE.

format :

ENABLE { INPUT / OUTPUT } [TERMINAL] nom-de-CD

L'instruction ENABLE autorise la communication avec les terminaux spécifiés dans le bloc de contrôle (nom-de-CD). L'instruction donne l'autorisation aux terminaux, soit d'émettre un message (INPUT), soit de recevoir un message (OUTPUT).

.../...

Les zones de bloc de contrôle à garnir sont :

- dans le cas d'un bloc de contrôle d'entrée; pour l'option INPUT TERMINAL, la zone SYMBOLIC SOURCE et pour l'option INPUT simple; au moins la zone SYMBOLIC QUEUE et si nécessaire les zones SUB-QUEUE (dans ce cas tous les terminaux reliés aux files d'attente désignés sont concernés par l'instruction).
- dans le cas d'un bloc de contrôle de sortie (option OUTPUT) : les zones DESTINATION COUNT et SYMBOLIC DESTINATION.

d. instruction DISABLE.

format :

DISABLE { INPUT [TERMINAL] / OUTPUT } nom-de-CD

L'instruction DISABLE interdit la communication avec les terminaux spécifiés dans le bloc de contrôle (nom-de-CD).
L'instruction donne l'interdiction aux terminaux, soit d'émettre un message (INPUT), soit de recevoir un message (OUTPUT). Les zones de blocs de contrôle à garnir sont les mêmes que celles décrites pour l'instruction ENABLE.

e. instruction ACCEPT MESSAGE COUNT

format :

ACCEPT MESSAGE COUNT FOR nom-de-CD

Cette instruction a pour but d'analyser les files d'attente d'entrée. Le nom-de-CD doit désigner un bloc de contrôle d'entrée, dont les zones SYMBOLIC QUEUE et SUB-QUEUE sont garnies. L'analyse des files d'attente aura pour effet de mettre à jour la zone QUEUE DEPTH du bloc de contrôle en y indiquant le nombre de messages complets que contiennent les files d'attente désignées.

.../...

CONCLUSION

1. Etendue de l'étude

L'étude présentée ici est une proposition limitée à expérimenter.
Nous avons pris des restrictions, d'une part en ne prenant en compte qu'une partie des modalités de l'exploitation en temps réel, d'autre part en ne considérant leur aspect qu'au niveau des programmes d'application.
En ce qui concerne les modalités d'exploitation, nous n'avons pas, par exemple, envisagé les problèmes d'édition ni du "multitasking". Nous avons étudié le problème strictement du point de vue des programmes d'application sans considérer les problèmes, telle la sécurité (checkpoint/restart), qui nous apparaissent plutôt devoir être résolus au niveau du moniteur de télétraitement.

2. Idées maîtresses de la réalisation

Premièrement, la réalisation proposée respecte le concept d'interface banalisé voulu par la méthode MEMO-PROGES : les messages manipulés par l'unité de traitement constituent des fichiers virtuels non structurés (chaînes de caractères).

Deuxièmement, nous avons voulu garantir, au niveau des modules programmés, la "transparence" de la multiplicité des utilisateurs en incluant le vecteur d'état de l'utilisateur pris en compte dans l'ensemble des paramètres banalisés contrôlant l'itération dans le module. Ces modules sont programmés comme s'il n'y avait qu'un seul utilisateur actif.

Troisièmement, nous avons banalisé le vecteur d'état comme tous les paramètres de contrôle dans le système MEMO-PROGES. Ce vecteur permet le contrôle de l'itération pour un utilisateur (USER), de l'interaction (STATE) et de la progressivité d'acquisition des données (STEP).

Quatrièmement, pour la manipulation de ces paramètres comme pour celle des entrées-sorties en temps réel, nous avons suggéré quelques extensions aux langages de programmation définis dans le système MEMO-PROGES

NOTE :

Dans la présente étude nous avons choisi le langage COBOL comme langage de programmation, pour deux raisons :

Le langage COBOL comporte un module de télétraitement assurant notamment l'indépendance physique par rapport aux terminaux.
D'autre part, le système actuel est déjà réalisé en COBOL.

.../...

BIBLIOGRAPHIE

1 ANSI — Proposed Revision of American National Standard COBOL
(ref. X3.23.1974)
National Technical Information Service, Springfield VA; 1974

2 CENTI — Gestion en temps réel
vol2 : Organisation du Système
CENTI, Paris; 1969

3 A. CLARINVAL — MEMO-PROGES: méthode modulaire de programmation pour l'informatique de gestion
thèse de doctorat, NAMUR 24 octobre 1977

4 A. CLARINVAL — Cours de COBOL
Univ. de Namur 1974-1975

5 A. CLARINVAL — MEMO-PROGES: définition des langages "LDF" et "LCM"
(à paraître)

6 S. DE HEPCEE — Systèmes en temps réel
Cours de l'institut d'informatique, Namur
Sd

7 A. FREEDMAN & R. LEES — Real-Time Computers Systems
Edward Arnold, London; 1977

8 J. MARTIN — Design of Real-Time Computer Systems
Prentice-Hall, Englewood Cliffs NJ; 1967

9 J. MARTIN — Utilisation et programmation des ordinateurs en temps réel
(Programming Real-Time Computer Systems)
éd. d'Organisation, Paris; 1969

10 J. MARTIN — Systems Analysis for Data Transmission
Prentice-Hall, Englewood Cliffs NJ; 1972

11 E. YOURDON Design of On-Line Computer Systems
Prentice-Hall, Englewood Cliffs NJ; 1972

12 Manuels de référence des compilateurs COBOL des firme I.B.M., PHILIPS, UNIVAC (Sperry Rand)