Facultés Universitaires
Notre-Dame de la Paix
Namur

Institut d'Informatique

P R O J E T N° 2

SYSTÈME DE GESTION DE BANQUE DE DONNÉES
SPHINX

QUATRIÈME PARTIE : LE SYSTEME SPHINX

UTILISATION, FONCTIONNEMENT ET DESCRIPTION INTERNE

JUIN 1978

SYSTÈME DE CONCEPTION ET D'EXPLOITATION DE BASES DE DONNEES

QUATRIÈME PARTIE : LE SYSTEME SPHINX

UTILISATION, FONCTIONNEMENT
ET DESCRIPTION INTERNE

PROJET DE RECHERCHE C.I.P.S. N° I.2/15.

AVERTISSEMENT

Cette brochure décrit l'organisation et le fonctionnement du système SPHINX qui permet de définir, construire et exploiter des bases de données accessibles à l'utilisateur au moyen des langages DDL, DML et NUL décrits dans les deux premières brochures de ce rapport.
Il n'est pas possible (ni probablement intéressant) de fournir dans un tel document une description technique complète du système. C'est la raison pour laquelle nous nous sommes limités à l'exposé des principes fonctionnels, de l'architecture et des choix algorithmiques qui sont à la base de cette réalisation; parfois, cependant, nous avons donné une description plus fine des informations internes et des modules. Le lecteur intéressé par des détails plus précis sera renvoyé soit à la documentation sur listing (pour certains composants), soit au texte même des programmes sources qui contiennent généralement leur propre documentation, soit encore, si cela s'avérait nécessaire, à la documentation du système SESAM.

o

o o

INTRODUCTION

1. Les composants du système SPHINX. I.2.
2. Schéma général d'une exploitation. I.3.

LE COMPILATEUR DDL

1. Les fonctions du compilateur DDL. DDL.2.
2. Les stratégies de compilation. DDL.5.
3. Les modules du compilateur. DDL.7.
4. Annexe : exemple de listing général d'une compilation. DDL.12.

LE COMPILATEUR DML

1. Les fonctions du compilateur DML DML.2.
 1.1. Description externe d'une compilation DML.2.
 1.2. Phases d'une compilation DML.3.
 1.3. Les programmes des phases DML.6.
2. Structure d'un programme objet COBOL DML.7.
 2.1. Programme source DML/COBOL DML.8.
 2.2. Programme objet COBOL DML.9.
 2.3. La génération correspondant aux ordres DML DML.10.
3. Stratégies de compilation DML.17.
4. Architecture du compilateur DML DML.19
 4.1. Le compilateur DML DML.19.
 4.2. Mise en forme des diagnostics DML et COBOL DML.20
5. Les tables du compilateur DML.21
6. Fichier des schémas externes DML.25
7. Les modules principaux du compilateur DML.31.
8. Les modules d'accès aux fichiers DML.32
9. Autres modules de service DML.36.

LA COMMUNICATION ENTRE TÂCHES DANS LE SYSTÈME SPHINX

1. Le système de communication entre tâches COM.1.
 1.1. OPCOM - Ouverture de communication COM.1.
 1.2. SEVNT - Envoi d'un buffer COM.1.
 1.3. REVNT - Réception d'un buffer COM.1.
 1.4. RELBF - Suppression d'un buffer de la file d'attente COM.2.
 1.5. CLCOM - Fermeture de communication COM.2.
2. Conventions de communication entre tâches dans le système SPHINX COM.3.
 2.1. Communication entre le noyau SESAM et le noyau SPHINX COM.3.
 2.2. Communication entre le noyau SPHINX et les programmes d'application COM.4.

L'INTERFACE DE COMMUNICATION ENTRE LES PROGRAMMES DML ET LE NOYAU SPHINX

1. Introduction INT.1.
2. Architecture globale de l'Interface INT.2.
3. Zones manipulées de l'Interface INT.4.
 3.1. Les arguments d'appel à SYSDML INT.4.
  3.1.1. La zone de communication de l'utilisateur (SYS-STACK) INT.4.
  3.1.2. Le vecteur diagnostic (SYS-ERAREA) INT.4.
  3.1.3. Les zones commandes (INST) INT.4.
 3.2. Les principales zones de l'Interface INT.6.
  3.2.1. Le stack de l'Interface ou stack descripteur (STACK) INT.6.
  3.2.2. La zone des variables de travail (VARAREA) INT.11.
  3.2.3. Le buffer de communication (CBUFFER) INT.13.
4. Les modules de l'Interface INT.15.
 4.1. SYSDML - Module principal de l'Interface INT.15.
 4.2. SYS01A,..., SYS20A - Modules spécialisés INT.15.
 4.3. SYSCOM - Module de gestion de la communication avec le noyau SPHINX INT.16.

LE NOYAU EXÉCUTIF SPHINX

1. Introduction SPH.1.
2. Les tables du noyau SPH.2.
 2.1. SCB - Bloc de contrôle du système SPH.2.
 2.2. Blocs associés à une tâche SPH.6.
  2.2.1. TCB - Bloc de contrôle de tâche SPH.6.
  2.2.2. STACK - Le stack des réalisations courantes de la tâche SPH.12.
  2.2.3. XENTRY - Bloc descripteur d'une réalisation d'objet complexe SPH.14.
  2.2.4. BUFFER - Buffer de communication SPH.16
  2.2.5. REQST - Bloc des demandes d'accès SPH.18.
 2.3. Blocs associés à une base de données SPH.20.
  2.3.1. DBH - Bloc d'en-tête d'une base de données SPH.20.
  2.3.2. OSCH - Schéma interne des objets complexes SPH.24.
  2.3.3. RSCH - Schéma interne des relations SPH.27.
  2.3.4. ACA - Zone de contrôle des accès SPH.30.
 2.4. Exemple graphique SPH.31.
3. Les modules du noyau SPH.33.
 3.1. Types de modules SPH.33.
 3.2. Modules de gestion des tables SPH.34.

3.2.1. SYSMEM - Module d'allocation dynamique de mémoire SPH.34.
3.2.2. SYSITC - Gestion des buffers de communication SPH.40.
3.2.3. SYSTCB-SYSCAN - Gestion des TCB SPH.41.
3.2.4. SYSDBH - Gestion des DBH SPH.44.
3.2.5. SYSREA - Création d'un bloc descripteur d'une réalisation SPH.47.
3.3. Modules du Superviseur SPH.48.
3.3.1. SYSPHX - Le superviseur proprement dit SPH.49.
3.3.2. SYS2OB,...,SYS2OG - Modules de contrôle des autorisations d'accès SPH.57.
3.3.3. SYSINT - Module de contrôle de l'état du noyau SPH.59.
3.3.4. SYSDWN - Arrêt du noyau SPH.60.
3.4. Modules spécifiques à chaque ordre DML SPH.61.

L'INTERPRÉTEUR NUL

1. Introduction NUL.1.
2. Les tables de l'interpréteur NUL NUL.2.
2.1. NULCB - Bloc de contrôle du système NUL NUL.2.
2.2. Blocs utilisés pour la description d'un contexte NUL.7.
2.2.1. CTXE - description d'une phrase du contexte NUL.8.
2.2.2. CTXINPUT - bloc descriptif de la phrase courante NUL.13.
2.2.3. EQE - bloc de description d'un critère élémentaire NUL.16.
2.2.4. QFE - bloc de description d'un critère complexe NUL.18.
2.2.5. Mémorisation du texte source NUL.20.
2.2.6. BUFCM - Buffer de communication avec le noyau SPHINX NUL.22.
2.2.7. QTFSW - QTFCNT - Indicateur et compteur associés à un quantificateur NUL.25.
2.3. Blocs de description d'une commande PRINT NUL.27.
2.3.1. EXCODE - Instructions en code intermédiaire NUL.28.
2.3.2. HEDT-EDTE - Table d'édition NUL.31.
2.4. Argument d'interrogation du schéma interne NUL.35.
3. Les modules de l'interpréteur NUL NUL.39.
3.1. Les différentes phases de l'exécution NUL.39.
3.2. NULMN1 - Enchaînement global des phases NUL.39.
3.3. Modules relatifs à la phase 2 - Représentation interne d'un contexte NUL.40.
3.3.1. NULPH2 - Module principal de construction d'un contexte NUL.40.
3.3.2. NULSTX - Analyse d'une phrase du contexte NUL.41.
3.3.3. NULCMD - Constitution d'un buffer de communication NUL.42.
3.3.4. NULCTX - Gestion des blocs CTXE NUL.43.

3.4. Modules relatifs à la phase 3 - Traduction et Interprétation d'une commande PRINT NUL.44.
3.4.1. NULP3S - Traduction d'une commande en code intermédiaire NUL.44.
3.4.2. NULP32 - Interprétation du code intermédiaire NUL.45.
3.4.3. NULEDT - Impression des valeurs d'objets élémentaires NUL.47.
3.5. NULINT - Contrôle du système NUL NUL.48.
3.6. Modules de service NUL.49.

LA SÉCURITÉ ET LES POINTS DE REPRISE

RÉORGANISATION D'UNE BASE DE DONNÉES

PROCÉDURES D'UTILISATION DU SYSTÈME SPHINX

1. Compilation d'un texte DDL. PRO.2.
2. Génération d'une base de données PRO.3.
3. Compilation et assemblage d'un programme d'application PRO.4.
4. Lancement d'une exploitation PRO.6.
5. Contrôle d'une exploitation PRO.8.

°

° °

I N T R O D U C T I O N

1. LES COMPOSANTS DU SYSTEME SPHINX

Chacun des chapitres décrit un composant du système complet.

Les compilateurs permettent de décrire, définir et générer une base de données (compilateur DDL) et de permettre l'accès à cette base à des programmes d'application (compilateur DML).
L'interface organise et contrôle les communications entre un programme d'application et le noyau SPHINX. Ce dernier est un moniteur qui centralise les demandes et les transforme en appel au système SESAM.
L'interpréteur NUL met à la portée d'un utilisateur non programmeur l'interrogation complexe de la base de données.
Nous dirons enfin quelques mots des utilitaires de restauration et de réorganisation d'une base de données.

Le dernier chapitre sera consacré à la présentation de procédures montrant la mise en oeuvre de certains de ces composants.

2. SCHEMA GENERAL D'UNE EXPLOITATION

L'exploitation d'un ensemble de bases de données se fait au moyen de programmes DML préalablement précompilés, compilés et édités avec la librairie de modules du système SPHINX. Chaque programme DML, s'exécutant, fait l'objet d'une tâche séparée du système BS 2000. De plus, chaque utilisateur NUL dialogue au terminal avec une occurrence séparée de l'interpréteur NUL (7).
Aucun de ces programmes ne réalise lui-même d'accès aux bases de données. Ces accès sont effectués par l'intermédiaire de deux autres tâches : le noyau SPHINX et le noyau SESAM. Ces tâches communiquent entre elles et avec les programmes d'application au moyen du système de communication entre tâches du BS 2000.
Ce système fournit des macros permettant d'échanger des informations entre différentes tâches normalement indépendantes.

Le noyau SESAM gère les accès physiques aux différentes bases de données (1). Le noyau SPHINX reçoit les requêtes des différentes tâches utilisateurs (2), fixe l'ordre de prise en charge de ces requêtes (concurrence) et les exécute, ce qui nécessite un certain nombre d'accès aux bases de données que le noyau SPHINX commande à SESAM par l'intermédiaire du module SESMOD (3).
Enfin, il renvoie les résultats aux tâches utilisateurs (4).

Chaque programme DML communique avec le noyau SPHINX (2) (4) via un interface (SYSDML) qui place dans un buffer de communication les requêtes (5) puis garnit les zones accessibles à l'utilisateur avec les résultats obtenus (6).

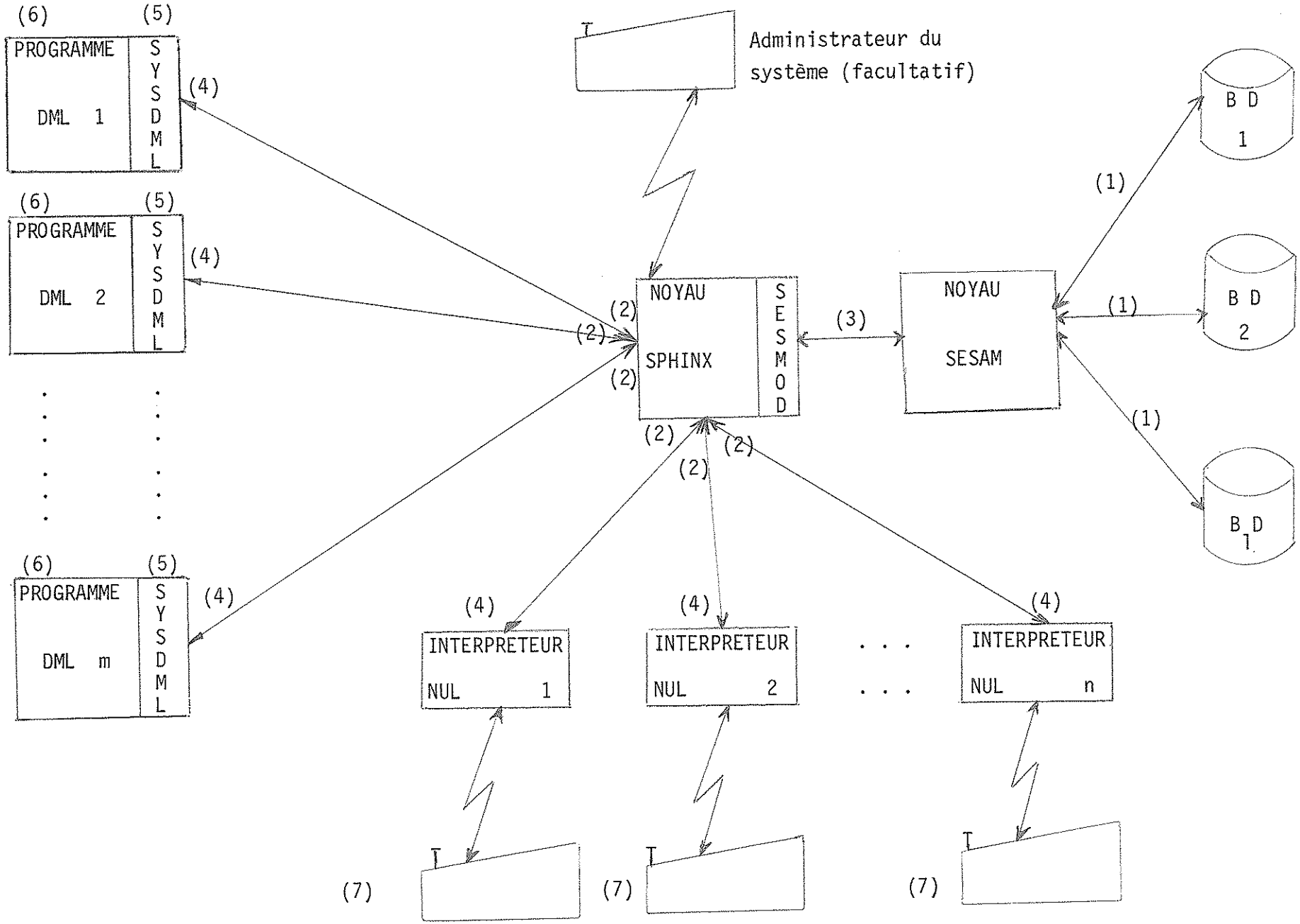


FIGURE 2. : Schéma général d'une exploitation

LE COMPILATEUR DDL

Remarque : ce chapitre constitue une description sommaire du compilateur DDL. Pour plus de détails, on consultera la documentation sur listing (fichier CDDL.DOCUMENTATION).

I. LES FONCTIONS DU COMPILATEUR DDL

La compilation d'un texte DDL correct a pour résultat la création et la mise à jour de fichiers propres à permettre la création et l'exploitation d'une base de données et à permettre la compilation de ses programmes d'application et le fonctionnement de l'interpréteur NUL; la compilation produit également un listing de ces fichiers et des diagnostics d'analyse.

On rappellera qu'une base de données gérée par SPHINX doit être décrite par un texte DDL. Lorsque cette base de données est opérationnelle, elle est aussi décrite par un schéma de compilation DML (membre d'un fichier de schémas) et un schéma destiné au noyau SPHINX (rangé dans la base de données elle-même et chargé en mémoire centrale lors de l'exécution de l'OPEN).
Une base de données étant réalisée sous forme de fichiers SESAM, il doit y correspondre une description dans ce système ainsi qu'un chargement initial correspondant entre autres choses au rangement du schéma nécessaire au noyau SPHINX.
Une base de données est identifiée par son nom, qui est celui de sa racine, par un numéro interne relatif à SPHINX (1 caractère) et par un numéro interne connu de SESAM (2 caractères).

La compilation réalise les fonctions suivantes :

- l'analyse d'un programme DDL décrivant une base de données;
- la création d'un fichier de diagnostics de compilation;
- la création du schéma destiné au compilateur DML et à l'interpréteur NUL et son insertion dans un fichier de schéma (ou son remplacement s'il y était déjà présent);
- la création (ou remplacement s'ils existaient déjà) des fichiers de paramètres de déclaration et de description de la base de données pour SESAM (fichiers suffixés par .SEDI12 et .SEFO);
- la création (ou remplacement) des fichiers permettant d'effectuer le chargement initial de la base de données (fichiers suffixés par .SEBE et .DATA);
- l'édition du listing source et de tous les fichiers créés par le compilateur (sauf le fichier préfixé par .DATA, qui n'est pas éditable directement, mais peut être visualisé par la commande /PRINT, BINARY = YES du BS2000).

Les entrées sont les suivantes :

- le fichier contenant le programme DDL à compiler; ce fichier doit être associé à l'entrée standard SYSDTA;
- le fichier des schémas DML et NUL dans lequel le schéma de la base de données sera inséré (ou remplacera l'ancienne version); ce fichier doit porter le nom CDDL.EXTERNAL.SCHEMA. Ce fichier peut ne pas exister, auquel cas il sera créé par le compilateur DDL. La structure précise de ce fichier est décrite dans la description du compilateur DML;

- le fichier des paramètres physiques qui seront imposés à la base de données SESAM. Ce fichier, qui porte le nom de CDDL.SEDI12, existe déjà, et contient des valeurs de paramètres par défaut; cependant d'autres valeurs peuvent être imposées par simple mise à jour de ce fichier. Complétés du nom et du numéro interne SESAM de la base de données ces paramètres constitueront le fichier d'entrée pour le programme PRO.SEDI12 de SESAM (fichier suffixé par .SEDI12).

Les sorties sont les suivantes :

- le fichier CDDL.EXTERNAL.SCHEMA augmenté, mis à jour ou créé.
  S'il existe déjà dans ce fichier un schéma DML correspondant au même nom de racine, ce schéma est remplacé par le nouveau mais le numéro interne SPHINX qui lui est associé reste le même.

- les fichiers de paramètres qui serviront à la déclaration et à la description de la base de données SESAM :
  - le fichier <nom de racine >.SEDI12, destiné au programme PRO.SEDI12, permettra de déclarer la base de données (nom et numéro SESAM) et certains paramètres physiques globaux : existence de fichiers inverses et numéros d'articles (data base key), taux de remplissage et seuil de réorganisation des différents composants, etc.(voir documentation SESAM).
  - le fichier <nom de racine >.SEFO, destiné au programme PRO.SEFO, permettra la constitution du catalogue des aspects SESAM : noms externes (longs) et internes (courts),longueur, cadrage, type de compactage, etc. (voir documentation SESAM).

- les fichiers qui permettront le chargement initial de la base de données; il s'agit d'un fichier de données et d'un fichier de description du format de ces données. Ces fichiers sont exploités par le programme PRO.SEBE de SESAM qui réalise le chargement (ou modification) d'une base de données SESAM.
  Il s'agit de :
  - <nom de racine >.DATA qui contient des enregistrements de
    - schéma des relations et schéma des objets complexes destinés au noyau SPHINX.
    - hiérarchies de suppression de chaque objet complexe; une hiérarchie décrit les accès dans le graphe des relations qui sont nécessaires pour réaliser la suppresssion d'une relation quelconque d'un objet complexe. Ces données sont exploitées par les modules SYS14 du noyau SPHINX relatifs à l'ordre DML SUPPRESS.
    - enregistrement (dit "bidon") constitué des valeurs "inconnu" de chaque objet élémentaire d'un objet complexe; ces enregistrements sont utilisés pour l'initialisation correspondant à l'exécution de l'ordre PREPARE.

- <nom de racine >.SEBE qui décrit les enregistrements du fichier précédent en établissant la correspondance nom d'aspect/localisation du champ.

- un listing constitué de l'édition du programme DDL, du fichier des diagnostics et des fichiers <nom de racine >.SEDI12, <nom de racine >.SEFO, <nom de racine > .SEBE et CDDL.EXTERNAL.SCHEMA.

On trouvera en fin de ce chapitre un listing correspondant à la compilation d'une base de données élémentaire , ainsi qu'un vidage du fichier <nom de racine >.DATA.

o

o o

II. STRATEGIES DE COMPILATION

Nous examinerons ici quelques techniques utilisées par le compilateur tant pour l'analyse du texte DDL que pour la création des fichiers.

A. La grammaire du DDL est représentée par un automate. Chaque élément de cet automate représente une transition possible; il est constitué comme suit :

- il est repéré par un numéro correspondant à l'état d'origine; ce numéro est l'adresse de l'élément dans la table qui représente l'automate (cf. module DDLTAB);
- un indicateur spécifiant que le symbole accepté pour la transition est soit un mot ou un caractère réservé du langage, soit une valeur (nom OC, nom OE, nom relation, nombre, ...);
- la valeur du symbole accepté (3 caractères) s'il s'agit d'un mot réservé ;
- le numéro de l'état d'arrivée de la transition si le symbole examiné est celui prévu pour l'élément;
- le numéro d'une autre transition acceptable à partir du même état d'origine; cette chaîne de transitions acceptables se termine quand ce numéro correspond à un état d'erreur;
- le numéro de l'action à réaliser lorsque cette transition se réalise.

Cet automate dirige l'analyse d'un programme DDL en indiquant à chaque pas la liste des symboles acceptables auxquels est comparé le symbole qui vient d'être extrait du programme, et en indiquant les opérations à réaliser lorsque ce symbole a été accepté.

B. Le compilateur construit en mémoire centrale une table décrivant les objets complexes et une table décrivant les relations. En raison de la taille possible de ces tables, on ne réservera que l'espace nécessaire. Cet espace est évalué par une lecture préalable du programmeDDL et comptage des descriptions d'objets complexes et de relations.
A cette occasion, le programme DDL est copié sur le fichier de travail CDDL.SOURCE à partir duquel s'effectuera l'analyse (module DDLRQM).

C. La création et l'insertion du schéma de compilation DML s'effectue dans le fichier de travail CDDL.SCHEMA. A cet effet, la version ancienne de CDDL.EXTERNAL.SCHEMA est copiée sur le fichier de travail soit jusqu'à la rencontre de l'ancienne version du schéma, soit jusqu'à la fin s'il n'y a pas d'ancienne version de ce schéma (module DDLCRT).
Le nouveau schéma est alors inséré à la suite de ce transfert.
Certains renseignements n'étant connus qu'à la fin de l'analyse du texte DDL

(longueur des SYS-C et SYS-PTR) ce fichier sera mis à jour à ce moment (module DDLUPD).
Si d'autres schémas suivaient l'ancienne version, ils seront copiés à la suite de la nouvelle version dans CDDL.SCHEMA.
Si la compilation se termine, ce dernier fichier remplacera CDDL.EXTERNAL.SCHEMA (suppression version ancienne puis catalogage).

D. L'utilitaire PRO.SEBE de chargement exige un fichier d'entrée <nom de racine >. DATA en longueur fixe. La longueur maximum n'étant connue qu'à la fin de l'analyse, on créera d'abord une version en longueur variable CDDL.DATA, que l'on associera par fusion au fichier HIEFIL contenant les hiérarchies de suppression pour constituer le fichier <nom de racine >. DATA en longueur fixe.

E. Pour les mêmes raisons que celles qui conduisent à la mise à jour différée de CDDL.SCHEMA, il est nécessaire d'insérer dans le fichier < nom de racine >.SEBE la description des champs correspondant à SYS-PTR et SYS-C. Ce fichier étant séquentiel, cette insertion s'effectue à l'occasion d'une copie sur CDDL.SEBE. Ce dernier fichier remplace alors <nom de racine >.SEBE (suppression ancienne version et catalogage). Cette opération est réalisée par le module DDLSBU.

F. Le fichier des diagnostics a pour nom <nom de racine >. DIAG. Cependant, lorsque le nom de la racine n'a pas été reconnu, il sera nommé CDDL.DIAG.

G. Lorsqu'une erreur est détectée dans une clause, l'analyse est reprise à partir du symbole qui suit le prochain ".".

o

o o

III. LES MODULES DU COMPILATEUR

Module DDLCMP

Il s'agit du module principal du compilateur. Outre son rôle de direction des différentes phases de la compilation, il prend directement en charge l'analyse et la génération relatives aux objets complexes et aux objets élémentaires et l'analyse des relations. La génération relative aux relations est laissée au module DDLREL.
Le module commande l'ouverture des différents fichiers et leur initialisation; il analyse la chaîne de symboles du programme DDL, consulte l'automate (DDLTAB) et en fonction de la réponse effectue l'action correspondante.
A la fin du fichier DDL analysé, il commande la fermeture des fichiers et l'édition du listing général.
L'acquisition des symboles s'effectue par un module DDLSYM (entry NXTSYM) analogue aux modules correspondants dans le compilateur DML et l'interpréteur NUL.

Module DDLRQM (entry REQMEM)

Ce module lit le fichier SYSDTA (fichier du programme DDL) et le copie dans CDDL.SOURCE. Il compte les clauses COMPLEX-OBJECT et RELATION, calcule l'espace nécessaire aux tables des objets complexes et des relations puis demande la mémoire correspondante au système d'exploitation.

Module DDLTAB (entry TABCMP)

Appelé avec un numéro d'état, ce module renvoie la description d'une transition.
L'automate y est représenté sous forme d'une table dont les éléments sont constitués de la manière décrite dans le paragraphe Stratégies de compilation.

Module DDLCRT (entry CRTFILE)

Ce module créé et ouvre un certain nombre de fichiers. Si le nom de racine a été détecté (dans le cas contraire le module se contente de créer CDDL.DIAG), il :

- crée et initialise le fichier <nom de racine >.DIAG.
- détermine si le schéma de compilation DML existe déjà;
  si oui : supprime les anciens fichiers .SEDI12,.SEFO, .SEBE;
  copie le début de CDDL.EXTERNAL.SCHEMA (qu'il a ouvert) dans CDDL.SCHEMA jusqu'à la description de la B.D.;

si non : affecte un numéro interne SESAM à la B.D.;
copie CDDL.EXTERNAL.SCHEMA dans CDDL.SCHEMA en y insérant une ligne de catalogue et la description de la racine;

- génère le fichier <nom de racine >.SEDI12 à partir de CDDL.SEDI12 en y insérant le nom et le n° SESAM de la base de données
- crée <nom de racine >.SEFO et l'initialise avec les deux premières lignes : carte AT10 de commande et description de l'aspect AAA.
- crée <nom de racine >.SEBE et l'initialise avec les trois premières cartes : ZD10, 2S1 et HDR1.

Module DDLREC (entry GENREC)

Il s'agit du module principal de génération d'enregistrements dans les fichiers de résultat. Les différentes fonctions sont les suivantes :

- génération de 4 lignes dans CDDL.SCHEMA :
  - ligne SYS-KREC ou identifiant DDL
  - ligne FILLER éventuelle
  - ligne SYS-PTR
  - ligne SYS-C
- génération ligne dans .SEBE pour l'aspect d'ordre
- génération ligne dans .SEBE
- génération ligne dans .SEFO pour un aspect autre que AAA
- génération ligne dans .SEFO pour aspect de 256 caractères (SYS-PTR et SYS-C)
- génération dans .SEBE et .SEFO pour les hiérarchies de suppression
- génération dans .SEFO pour les aspects ZZ7, ZZ8 et ZZ9
- génération dans .DATA des lignes de schéma pour le noyau SPHINX
- génération dans .SEBE et .SEFO d'une ligne de clôture.

Module DDLUPD (entry UPDREC)

Pour chaque objet complexe, le module détermine le nombre d'aspects nécessaire pour contenir SYS-PTR et SYS-C; il génère alors les lignes correspondantes dans .SEFO, puis il met à jour les lignes SYS-PTR et SYS-C temporaire de CDDL.SCHEMA. Le module indique également si trop d'aspects ne sont pas demandés par ces valeurs.

Module DDLSBU (entry SEBUPD)

Ce module effectue la mise à jour du fichier .SEBE en y insérant la description des valeurs initiales des compteurs (SYS-C) et des pointeurs (SYS-PTR).

Cette insertion se fait lors de la recopie de <nom de racine>.SEBE sur CDDL.SEBE. Le premier fichier est alors supprimé et le second catalogué sous son nom.

Module DDLASP (entry GENASP)

Fournit un nom d'aspect interne (3 caractères) dans l'ordre alphabétique, en commençant par BAA et en évitant certaines combinaisons réservées à SESAM.

Module TSTLEV (entry TLEVEL)

Vérifie la correction du numéro de niveau d'un objet élémentaire et calcule la longueur des valeurs d'un objet élémentaire de niveau 01.

Module DDLGDT (entry GENDTA)

Construit et écrit dans CDDL.DATA un enregistrement des valeurs "inconnu" pour un objet complexe.
Le module est constitué de plusieurs phases : initialisation, calcul de la valeur "inconnu" en fonction du type de l'OE, écriture lorsque toutes les valeurs ont été créées.

Module DDLVER

Test de la structure des objets élémentaires dépendant d'un même objet complexe. Le module vérifie qu'aucune qualification complète d'un objet élémentaire n'est une sous-liste de qualification d'un autre objet.
Le module est constitué de trois phases : initialisation, construction de l'arbre des objets élémentaires, vérification.

Module DDLHIE (entry SUPHIE)

Ce module construit les hiérarchies de suppression des objets complexes et les écrit dans le fichier HIEFIL. Il vérifie également l'absence de cycle de relations fortes. Il met à jour les paramètres de la table des objets complexes (par exemple : nombre et liste des aspects qui contiendront des hiérarchies).
Une hiérarchie est représentée sous une forme préordre (noeud suivi de ses sous-arbres descendants) chaque noeud est représenté par un numéro de niveau dans la hiérarchie et le numéro de la relation à emprunter (ces numéros valent 0 pour le noeud racine).

Module DDLREL (entry GENREL)

Ce module commande la vérification d'une relation, la création des aspects des blocs de pointeurs, la génération des lignes correspondantes dans CDDL.SCHEMA et <nom de racine >.SEFO et la mise à jour de la table des relations.

Module DDLTRL (entry VERREL)

Vérifie qu'une relation n'a pas déjà été définie.

Module DDLCLO (entry CLOSEFIL)

Ce module est le complémentaire de DDLCRT.
Il ferme DDL.SEDI12, <nom de racine >.SEDI12, <nom de racine >.SEFO et < nom de racine >.DIAG.
Il termine, si nécessaire, la copie de CDDL.EXTERNAL.SCHEMA sur CDDL.SCHEMA en évitant l'ancienne version du schéma; le premier fichier est alors supprimé puis le second est catalogué sous son nom.

Module DDLFCH (entry COPFCH)

Ce module complète le fichier SYSLST (le listing de sortie) en y ajoutant l'édition de <nom de racine >.DIAG, <nom de racine >.SEDI12, <nom de racine >.SEFO, <nom de racine >.SEBE et CDDL.EXTERNAL.SCHEMA
Cependant, en cas d'erreur grave, seul le fichier<nom de racine >.DIAG est copié.

Module DDLLST (entry WRLST)

Ecriture sur SYSLST de :
- en-tête des fichiers CDDL-SOURCE, .DIAG, .SEDI12, .SEFO, .SEBE et .CDDL.EXTERNAL.SCHEMA.
- ligne quelconque avec contrôle de saut de page et rappel d'en-tête.

Module DDLDTA (entry RDATA)

Lecture d'une ligne sur SYSDTA.

Module DDLERR (entry LISTERR)

Module de gestion du fichier CDDL.DIAG ou <nom de racine >.DIAG.

Module DDLMSG (entry DISPMSG)

Ce module transforme un numéro d'erreur en un message explicite et l'imprime sur le fichier .DIAG.
En outre, si la tâche est interactive, le message est aussi imprimé au terminal.

Module DDLSHI (entry REDSCH)

Module de gestion du fichier CDDL.EXTERNAL.SCHEMA.

Module DDLSHO (entry WRTSHO)

Module de gestion du fichier CDDL.SCHEMA.

Modules DDLDFO (entry WRTDFO) et DDLDFI (entry READDT)

Modules de gestion du fichier CDDL.DATA.

Module DDLDTF (entry WRTDTF)

Module de gestion du fichier <nom de racine >.DATA

Modules DDLSFO (entry WRTSFO) et DDLSFI (entry RDSFI)

Modules de gestion du fichier <nom de racine >.SEFO.

Module DDLS12 (entry RDS12)

Module de gestion du fichier CDDL.SEDI12

Modules DDLSBI (entry RWTSBE) et DDLSBO (entry WRTSBE)

Modules de gestion du fichier <nom de racine >.SEBE

Module DDLSEB (entry SEBEOUT)

Module de gestion du fichier CDDL.SEBE

Module DDLSCE (entry SOURCE)

Module de gestion du fichier CDDL.SOURCE

Module DDLHRD (entry READHIE)

Module de gestion du fichier HIEFIL.

o

o o

IV. ANNEXE : exemple de listing général d'une compilation

Le programme DDL décrit une base de données élémentaire contenant un objet complexe et trois relations.

Il est ajouté à un fichier CDDL.EXTERNAL.SCHEMA initialement vide.

```
0001    IDENTIFICATION NAME IS COLUSSI.
0002    ROOT NAME IS COL.
0003    C-O NAME IS COL.
0004    RELATION NAME IS RELATION FROM (0,1) COL TO (0,1) COL INVERTED BY RELATION.
0005    RELATION NAME IS ALLER FROM (0,1) COL TO (0,1) COL INVERTED BY RETOUR.
0006    END.
```

LINE NUMBER ERROR MESSAGES
☆☆☆☆☆☆☆☆☆☆☆ ☆☆☆☆☆☆☆☆☆☆☆☆☆☆

☆☆☆☆☆ D.D.L. COMPILATION COMPLETED WITHOUT ERRORS ☆☆☆☆☆

```
COL
JA
DB NR=0A
PARZGR=3
PWK=003
SNR=JA
AWL=JA
AWL SE=99999
AWL REORG=050
AWL DENSITY=080
ORG REORG=050
ORG DENSITY=080
ZDR REORG=050
ZDR DENSITY=080
2-BAND-VERSION=NEIN
ORG ARB=100
AK=020
/*
```

```
AT10 GEN,LIST=SAN,NAME='COL'
1DAAA12AAA                    03200L1S0JN        000ZN
1DBAA12BAA                    01200L1N0JN        000ZN
1DBAB12BAB                    01200L1N0JN        000ZN
1DZZ112ZZ1                    25600L1N0JN        000ZN
1DZZ712ZZ7                    06400L1N0JN        000ZN
1DZZ812ZZ8                    03600L1N0JN        000ZN
1DZZ912ZZ9                    00800L1N0JN        000ZN
/*
```

```
ZD10 NAME='COL',GEN
2S1 DT=BD,SF=S,BL=1862,SL=0266,SA=004,S2=0,S3=0,T1=010
HDR1COL.DATA
2T1001 00010003AAA00040008LU001
2T1001 00010003BAA00090020LU001
2T1001 00010003BAB00210032LU001
2T1001 00010003ZZ900330040LU001
2T1997 00010003AAA00040010LU001
2T1997 00010003ZZ700110074LU001
2T1998 00010003AAA00040010LU001
2T1998 00010003ZZ800110046LU001
2T1999 00010003AAA00040010LU001
2T1999 00010003ZZ100110266LU001
SK=ENDE
/*
```

```
$COL        A
A$$$$$      COL
A0001$      COL
A0001*M001  03 SYS-KREC        PIC  X(0008)                          .   0000008AAA
A0001*M002  03 FILLER          PIC  X(0019)                          .   0000019
A0001*M003  03 SYS-PTR         PIC  X(0004)                OCCURS 0003.   0000012BAA
A0001*M004  03 SYS-C           PIC  9(0005)         COMP   OCCURS 0003.   0000012BAB
AR001$      RELATION          0001$0001$R001$0000000001000000001      1001000000000001000000000001001
AR002$      ALLER             0001$0001$R003$0000000001000000001      1002000000000003000000000002003
AR003$      RETOUR            0001$0001$R002$0000000001000000001      1003000000000002000000000003002
```

```
FILENAME = COL.DATA


LOGICAL PAGE = 1

KEY -- (0001)  (000) 4445E8B0 01000001 00000100 07460000

(0001)  (000) F9F9F95B 5B5BC8F0 F0F10000 00000001  (010) 00010001
(0033)  (020) 40404040 40404040 40404040 40404040  (030) 40404040
          NEXT 6 LINES ARE IDENTICAL TO ABOVE LINE.
(0257)  (100) 40404040 40404040 4040F9F9 F75B5B5B  (110) D6F0F0F0
(0289)  (120) 40404040 40404040 40404040 40404040  (130) 40404040
          NEXT 6 LINES ARE IDENTICAL TO ABOVE LINE.
(0513)  (200) 40404040 40404040 40404040 40404040  (210) 40404040
(0545)  (220) 00030100 00020038 00304040 00000101  (230) 01C2C1C1
(0577)  (240) 40404040 40404040 40404040 40404040  (250) 40404040
          NEXT 5 LINES ARE IDENTICAL TO ABOVE LINE.
(0769)  (300) 40404040 40404040 40404040 40404040  (310) 40404040
(0801)  (320) F85B5B5B D9F0F0F0 00034040 40404040  (330) 40404040
(0833)  (340) 40404040 40404040 40404040 40404040  (350) 40404040
          NEXT 6 LINES ARE IDENTICAL TO ABOVE LINE.
(1057)  (420) 40404040 40404040 F9F9F85B 5B5BD9F0  (430) F0F10140
(1089)  (440) 00010000 00000000 00010001 00010000  (450) 00000000
(1121)  (460) 40404040 40404040 40404040 40404040  (470) 40404040
          NEXT 5 LINES ARE IDENTICAL TO ABOVE LINE.
(1313)  (520) 40404040 40404040 40404040 40404040  (530) 4040F9F9
(1345)  (540) 00010001 00000000 00000001 00000000  (550) 00000001
(1377)  (560) 40404040 40404040 40404040 40404040  (570) 40404040
          NEXT 5 LINES ARE IDENTICAL TO ABOVE LINE.
(1569)  (620) 40404040 40404040 40404040 40404040  (630) 40404040
(1601)  (640) 5B5BD9F0 F0F30140 40400001 00010000  (650) 00000000
(1633)  (660) 00030000 00000000 00024040 40404040  (670) 40404040
(1665)  (680) 40404040 40404040 40404040 40404040  (690) 40404040
          NEXT 5 LINES ARE IDENTICAL TO ABOVE LINE.
(1857)  (740) 40404040 40400000 00000000 00000000  (750) 00000000
(1889)  (760) 00000000 00000000 00000000 00000000  (770) 00000000
          NEXT 4 LINES ARE IDENTICAL TO ABOVE LINE.


LOGICAL PAGE = 2

KEY -- (0001)  (000) 4445E8B0 01000002 00000200 010A0000

(0001)  (000) F0F0F15B 5BF0F0F1 00000000 00000000  (010) 00000000
(0033)  (020) 00000000 00000000 40404040 40404040  (030) 40404040
(0065)  (040) 40404040 40404040 40404040 40404040  (050) 40404040
          NEXT 5 LINES ARE IDENTICAL TO ABOVE LINE.
(0257)  (100) 40404040 40404040 40400000 00000000  (110) 00000000
(0289)  (120) 00000000 00000000 00000000 00000000  (130) 00000000
          NEXT 54 LINES ARE IDENTICAL TO ABOVE LINE.
```

```
                                        ..Y.............

00020001 00030000 00004040   999$$$H001......................
40404040 40404040 40404040

00014040 40404040 40404040             997$$$0000..
40404040 40404040 40404040

F9F9F75B 5B5BD6F0 F0F10003                     997$$$0001..
C2C1C2E9 E9F14040 40404040   ..........  .....BAABABZZ1
40404040 40404040 40404040

40404040 40404040 4040F9F9                                99
40404040 40404040 40404040   8$$$R000..
40404040 40404040 40404040

40400001 00010000 00000000         998$$$R001.   ..........
00014040 40404040 40404040   ......................
40404040 40404040 40404040

F85B5B5B D9F0F0F2 01404040                   998$$$R002.
00020002 00000000 00000003   ..............................
40404040 40404040 40404040

40404040 40404040 F9F9F85B                             998$
00010000 00000000 00010003   $$R003.   ......................
40404040 40404040 40404040   ..........
40404040 40404040 40404040

00000000 00000000 00000000         ........................
00000000 00000000 00000000   ..............................


                                        ..Y.............

00000000 00000000 00000000   001$$001......................
40404040 40404040 40404040   ........
40404040 40404040 40404040

00000000 00000000 00000000             ....................
00000000 00000000 00000000   ..............................
```

LE COMPILATEUR DML

Remarque : ce chapitre constitue une description sommaire du compilateur DML. Pour plus de détails, on consultera la documentation sur listing (fichier DML.DOCUMENTATION) ainsi que les textes sources des programmes, abondamment commentés.

I. LES FONCTIONS DU COMPILATEUR DML

1.1. Description externe d'une compilation

La compilation complète d'un programme source DML correct a pour résultat la construction d'un programme objet (que l'on rangera généralement dans une bibliothèque de modules objets), ainsi que l'édition d'un listing du programme source accompagné des diagnostics résultant de son analyse.

La compilation a donc pour fonctions :

- l'analyse d'un programme source DML/COBOL;
- l'édition d'un listing du programme source et des diagnostics;
- la génération (et rangement éventuel en bibliothèque) d'un programme objet binaire si l'analyse reconnaît le programme correct.

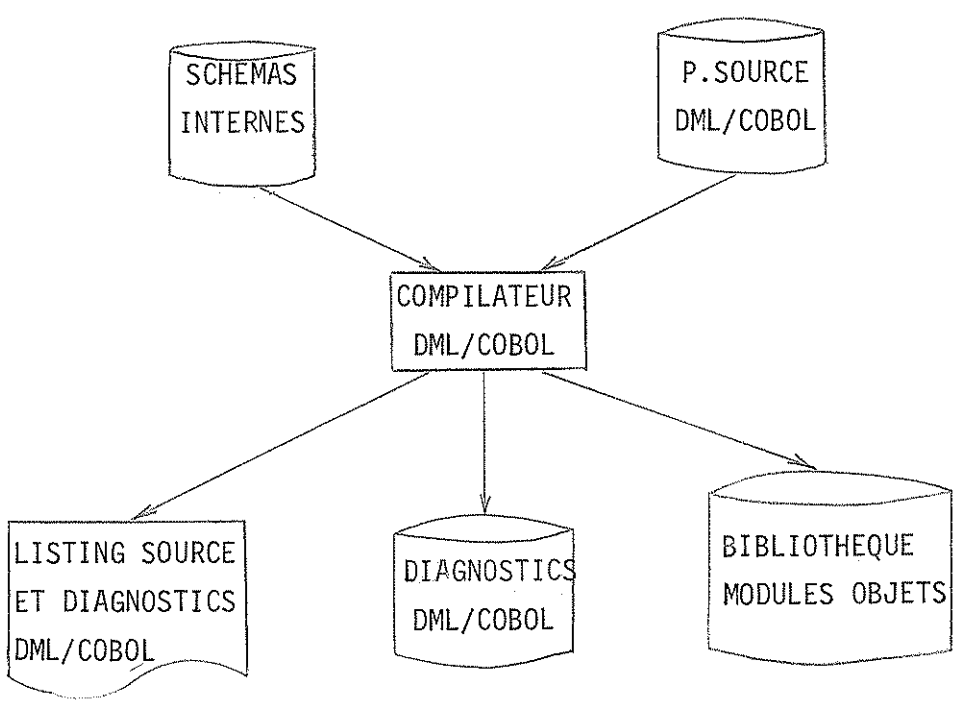
Les entrées sont :

- un fichier contenant le programme source DML/COBOL;
- un fichier contenant les schémas internes de compilation des bases de données citées dans le programme source.

Les sorties sont :

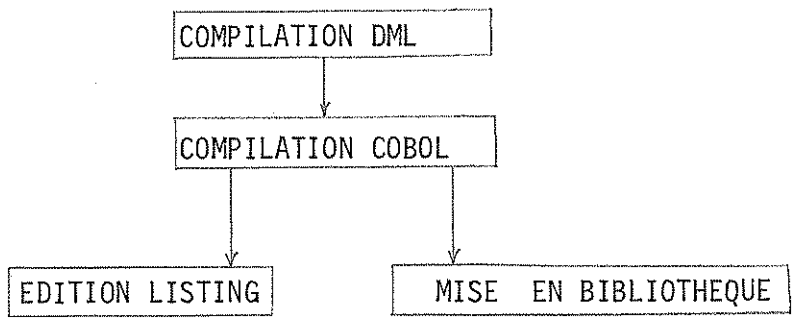
- un listing du programme source DML/COBOL et des diagnostics;
- un fichier résumé des diagnostics;
- le module objet binaire correspondant au programme source, et qui sera rangé en bibliothèque.

Du point de vue externe, une compilation se déroule donc comme suit :



1.2. Phases d'une compilation

Une compilation complète d'un programme DML/COBOL s'effectue en quatre phases qui s'enchaînent comme suit :



Compilation DML

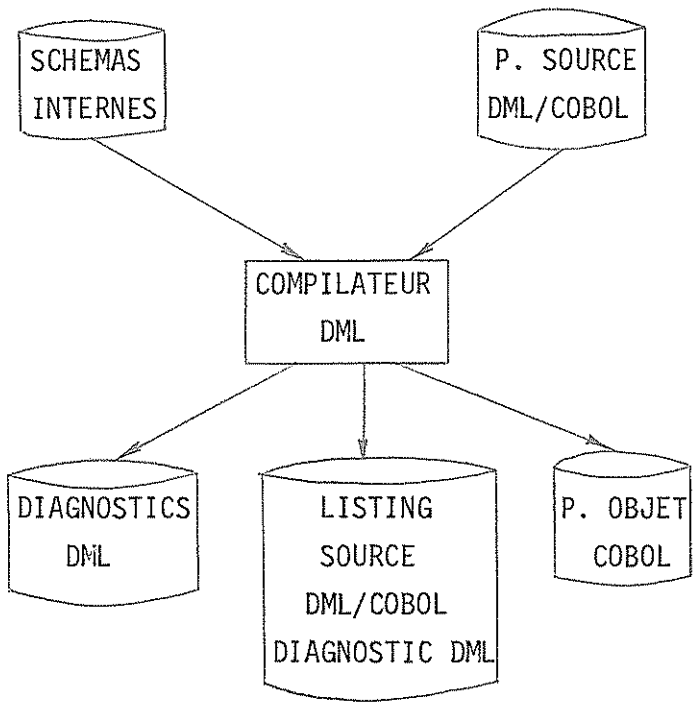
Fonctions :
- analyse DML d'un programme source DML/COBOL
- édition d'un listing source temporaire (fichier temporaire);
- édition des diagnostics DML (fichier temporaire)
- génération d'un programme objet COBOL (fichier temporaire);
  Ce programme est la traduction du programme source en COBOL pur; les ordres DML ont été transformés en instructions COBOL et en particulier en appels à l'interface SYSDML; ce programme objet contient également la zone de communication. Ce programme n'est créé que si l'analyse n'a décelé aucune erreur DML.

Entrées :
- fichier contenant le programme source DML/COBOL;
- fichier des schémas internes de compilation pour les bases de données concernées par le programme source.

Sortie :
- s'il n'a pas eu d'erreur, fichier contenant le programme objet en COBOL pur;
- un fichier contenant un listing temporaire du programme source DML/COBOL; ce listing contient le texte source augmenté :
  - d'une numérotation des lignes du programme source;
  - d'une numérotation des ordres DML présentée sous la forme d'une ligne de commentaire précédant chaque ordre DML et indiquant le numéro de séquence de cet ordre dans le programme;
  - liste des diagnostics d'erreur DML.
- un fichier temporaire des diagnostics DML; chaque détection d'une erreur lors de l'analyse donne lieu à la création d'une ligne de message constituée du numéro de la ligne du programme source où l'anomalie a été détectée, du code numérique de l'anomalie, du libellé

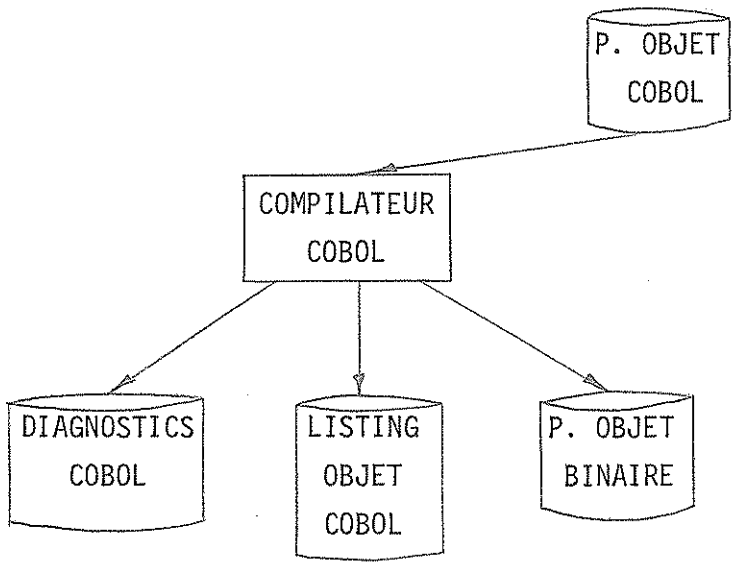
du message ainsi que d'un paramètre éventuel (la constitution du message est décrite dans la brochure MANUEL DE REFERENCE DES LANGAGES).

Schéma :



Compilation COBOL :

Schéma :



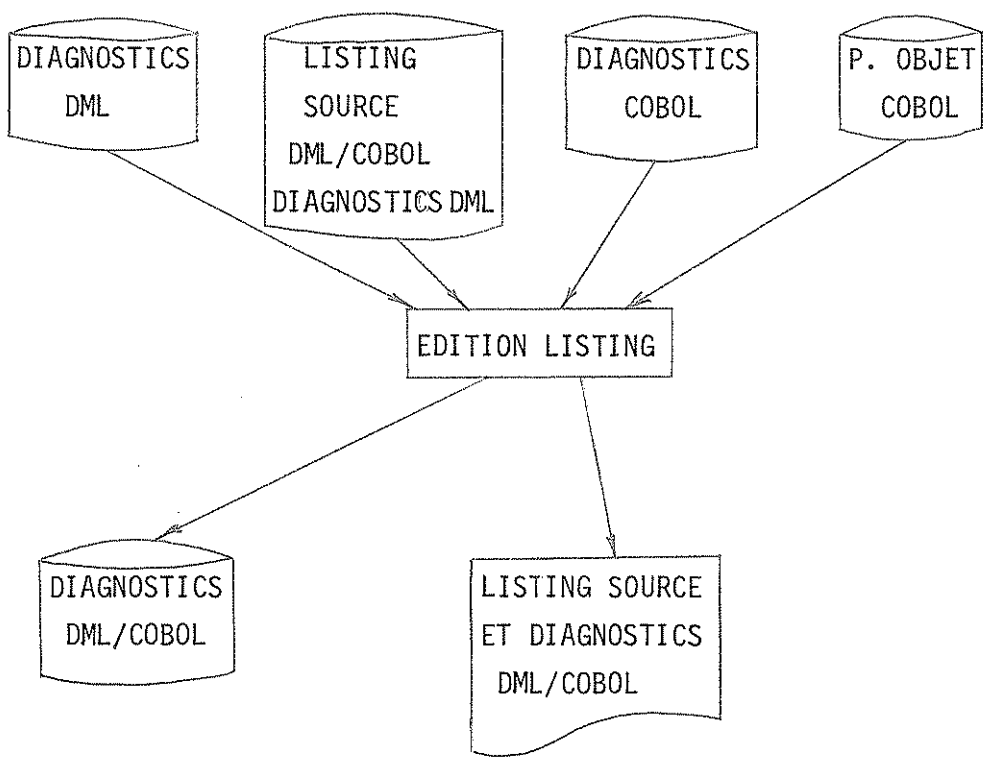
Edition du listing

Fonctions : - édition d'un listing unique du programme source DML/COBOL et des diagnostics DML et COBOL.

Entrées : - fichier du listing source DML/COBOL et diagnostics DML
- fichier des diagnostics DML
- fichier des diagnostics COBOL
- fichier du programme objet COBOL

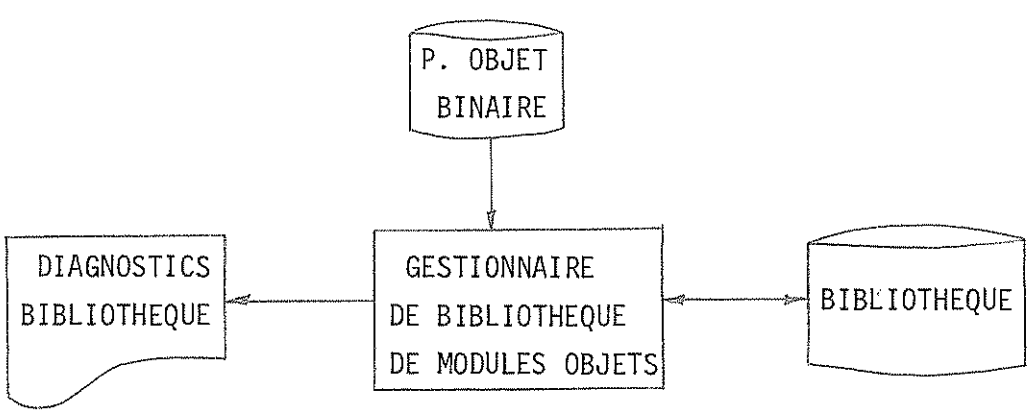
Sorties: - listing définitif du programme source DML/COBOL et des diagnostics DML et COBOL (il s'agit en fait d'un fichier éditable)
- fichier des diagnostics DML et COBOL

Schéma :



Mise en bibliothèque

Schéma :



Remarque

Le schéma général d'un compilation complète ignore en fait la création de certains fichiers de sortie tels que le listing objet COBOL et le listing de diagnostic du gestionnaire de bibliothèque; ces fichiers sont en fait secondaires du point de vue de l'utilisateur DML puisqu'ils correspondent à des opérations classiques de manipulation de programmes.

1.3. Les programmes des phases

A chaque phase correspond un programme spécifique.
Les phases de compilation COBOL et de mise en bibiothèque font appel à des programmes utilitaires offerts par le système d'exploitation, en l'occurrence le module ANSICOB et l'utilitaire LMR du système d'exploitation BS2000 de SIEMENS; nous n'en parlerons pas davantage.

La phase de compilation DML est réalisée par le programme DMLMN1 tandis que l'édition du listing final se fait par le programme DMLMN2.
C'est à l'étude de ces deux programmes et en particulier du premier, que seront consacrées les pages qui suivent; cependant nous examinerons avant tout la composition d'un programme objet DML et les stratégies de compilation adoptées avant d'aborder la description des tables et des modules.

II. STRUCTURE D'UN PROGRAMME OBJET COBOL

Par rapport au programme source DML/COBOL, le compilateur DML constitue le programme objet COBOL en :

- remplaçant les ordres DML par des instructions COBOL et en particulier par des appels à l'interface SYSDML;
- insérant la description de la zone de communication qui permet l'interaction entre le programme d'application lui-même et le système de gestion de bases de données (via l'interface).

Hormis la détection de "WORKING-STORAGE SECTION" il n'analyse pas les instructions COBOL.

D'une manière générale, les instructions COBOL relatives à un ordre DML se réduisent à :

- une initialisation éventuelle des zones de paramètres variables (REACH IF),
- un appel à l'interface avec citation des arguments,
- un contrôle du code de retour principal et les branchements relatifs à la conduite des boucles ou à la détection d'une anomalie grave.

Certains verbes cependant entraînent une génération plus simple encore : NEXT et EXIT ne provoquent pas d'appel à l'interface et COUNT est remplacé par la désignation d'un élément d'un vecteur.

La zone de communication contiendra trois types d'information ou de zones :

- des variables accessibles à l'utilisateur dans lesquelles le système rangera des renseignements sur l'exécution du dernier ordre DML; il s'agit de
  SYS-ERROR (code de retour général)
  SYS-STATUS (code du verbe DML)
  SYS-MAC-NUM (numéro séquentiel de l'ordre dans le programme)
  SYS-ERAREA (vecteur de codes de retour pour ordres multiples)
  SYS-REG (résultat d'un test ou d'une comparaison)

Ces variables sont décrites dans la brochure MANUEL DE REFERENCE DES LANGAGES.

- des paramètres d'appel à l'interface constituant une sorte de code intermédiaire; à chaque ordre DML qui entraîne un appel à l'interface correspond un paramètre qui décrit les opérations à effectuer.
- des zones dans lesquelles d'une part le système rangera les données obtenues lors des accès à la base de données et d'autre part, l'utilisateur rangera les données qu'il veut enregistrer dans la base de données; ces zones contiendront également les vecteurs de compteurs associés aux réalisations d'objets complexes; une partie de ces zones est accessible directement à l'utilisateur : il s'agit essentiellement de celles qui contiennent les valeurs d'objets élémentaires et qui peuvent être désignées par le nom de ces objets

élémentaires éventuellement qualifié par le nom de l'étiquette correspondante.

Les schémas suivants montrent les modifications que subit un programme source DML.COBOL pour être transformé en programme objet COBOL.

2.1. Programme source DML/COBOL

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION
      .
      .
      .
WORKING-STORAGE SECTION.
       (Zones de l'utilisateur)
PROCEDURE DIVISION .
      .
      .
      .
   @REACH CONSTRUCTEUR = CONSTR ...
      .
      .
      .
SYS-DUMP.
      .
      .
      .
```

2.2. Programme objet COBOL

```
IDENTIFICATION DIVISION

ENVIRONMENT DIVISION

DATA DIVISION
        :
WORKING-STORAGE SECTION

          (liste des arguments d'appel à SYSDML)

01 SYS-ERAREA

01 SYS-STACK.
  02 SYS-ERROR
  02 SYS-STATUS
  02 SYS-MAC-NUM
  02 SYS-REG
  02 FILLER PIC X(2140)
01 SYS-0002-STACK REDEFINES SYS-STACK.
  02 FILLER PIC X(16).
  02 CONSTR.
      :
              (Pile)

                  (Zones de l'utilisateur)
PROCEDURE DIVISION
        :

*                                        --DML  02--
     MOVE ...
     CALL SYSDML USING ...
     IF SYS-ERROR > "10" GO TO SYS-DUMP.

        :

SYS-DUMP.

        :
```

La partie de la zone de communication affectée aux informations relatives aux différentes réalisations est organisée d'une manière particulière qui permet de profiter de la structure de blocs du programme DML pour ne conserver en mémoire que les informations relatives aux réalisations actives à chaque endroit. Cette zone est, tout naturellement, organisée en pile.
La gestion de cette pile est statique (c'est-à-dire fixée au stade de la compilation) à la fois en raison de la rigidité de la structure des blocs et des possibilités du COBOL (pas de gestion dynamique de la mémoire).

L'accès aux informations de la pile peut s'effectuer en COBOL grâce à la technique de construction suivante :

- on réserve pour la pile une zone de longueur fixe correspondant à la longueur maximum que peut atteindre cette pile dans le programme; cette zone porte le nom de SYS-STACK et contient en son début quelques variables de longueur fixe (pour simplifier le passage d'argument vers l'interface)
- chaque fois que de l'espace doit être réservé dans cette pile, une nouvelle découpe de cette dernière est générée (ordre REDEFINES); cette découpe contient deux parties : un déplacement (FILLER) de positionnement au sommet de la pile puis la description des zones utiles : valeurs d'objets élémentaires, vecteur de compteurs, identifiant, etc.

2.3. La génération correspondant aux ordres DML

Le code intermédiaire, résultant de la compilation d'un ordre DML, et passé en argument à l'interface SYSDML, est constitué d'un fragment initial en format fixe identique pour tous les codes, et d'une chaîne de paramètres, en nombre et format variables selon le code.
Le nom du code est :

SYS-<n-séq> - C <suffixe>

ou <n-séq> est le numéro de séquence de l'ordre DML,
<suffixe> est un suffixe distinguant différents codes d'un même ordre.
Le fragment fixe est constitué de :

- numéro de code de l'ordre DML (voir brochure Manuel de Référence, p. DML.48)
- le niveau d'imbrication de l'ordre DML
- numéro de séquence de l'ordre DML
- numéro de la base de données concernée.

Génération de l'ordre OPEN (Code DML 01)

Code intermédiaire : - fragment fixe
- nom de la base de données.

Procédure : - appel à SYSDML argument : code intermédiaire, pile, vecteur SYS-ERLIST;
- test du code de retour : = 10 : BD inexistante, sortie du bloc
< 10 : BD ouverte
autre : erreur, brancher en SYS-DUMP.

Génération de CLOSE (code 02)

Code intermédiaire : - fragment fixe.
N.B. : son nom est suffixé par 1.

Procédure : - exécution éventuelle des sorties des blocs imbriqués,
- appel à SYSDML; argument : code intermédiaire; constitue un paragraphe de sortie
- test code de retour.
N.B. : il peut y avoir génération suite à un appel au module de protection.

Génération du REACH (code 03)

Code intermédiaire du premier accès (suffixe = A)

- fragment fixe
- type d'accès
- numéro interne de l'objet complexe
- niveau de l'ordre relatif au FROM
- numéro interne de la relation ou de la variable de travail
- niveau de l'ordre correspondant au IN (ou GET)
- adresse de la zone des valeurs de comparaison
- longueur de cette zone
- adresse de la zone correspondant au GET
- longueur de cette zone
- longueur de la zone d' "indication field" (code intermédiaire du type SESAM) relative aux conditions de sélection
- longueur de la zone d' "indication field" relative à la liste des valeurs à acquérir (GET)
- "indication field".

Code intermédiaire des accès suivants (suffixe=B)

- fragment fixe
- "9 *"

Génération dans la pile :

- déplacement dans la pile (FILLER)
- zone des valeurs de comparaison
- zone de réception des valeurs à acquérir

procédure :
- éventuellement génération d'appel à la protection (DML403)
- ordres de garnissage de la zone des valeurs de comparaison
- premier appel à SYSDML (1er accès)
- deuxième appel à SYSDML (accès suivants) constituant le paragraphe d'itération
- test code de retour : = 10 : sortie du bloc
  < 10 : réalisation trouvée
  autre : erreur, brancher en SYS-DUMP

Génération de l'ordre EXIT (code 04)

procédure :
- exécution (PERFORM) des paragraphes de sortie des blocs imbriqués
- branchement vers le paragraphe de sortie du bloc concerné (cf. END et CLOSE)

N.B. éventuellement,génération de l'argument d'appel à la protection (via DML403)

Génération de l'ordre NEXT (code 05)

procédure :
- exécution (PERFORM) des paragraphes de sortie des blocs imbriqués
- branchement vers le paragraphe d'itération (cfr. REACH)

N.B. éventuellement, génération de l'argument d'appel à la protection (via DML403)

Génération de l'ordre END (code 06)

code intermédiaire :
- fragment fixe

N.B. : nom suffixé par 1

procédure :
- exécution éventuelle des paragraphes de sortie des blocs imbriqués
- appel à SYSDML; argument : code intermédiaire; constitue un paragraphe de sortie;
- test code de retour

N.B. : éventuellement, génération de l'argument d'appel à la protection (via DML403)

Génération de l'ALLOCATE (code 07)

code intermédiaire :
- fragment fixe
- numéro interne de l'objet complexe
- niveau de l'ordre DML (inutile ici mais compatible avec PREPARE)

- adresse de la zone correspondant au GET
- longueur de cette zone

génération dans la pile : - déplacement dans la pile
- zone de réception des valeurs à acquérir

procédure : - génération d'appel à la protection si nécessaire (via DML403)
- appel à l'interface SYSDML
- test code de retour.

Génération de l'ordre PREPARE (code 08)

cf. génération de PREPARE; plus :

- argument de l'appel à SYSDML pour lecture des valeurs "inconnu" relatives aux objets élémentaires concernés
- en procédure : - appel à SYSDML pour lecture des valeurs "inconnu"
  - éventuellement appel à la protection (DML 403)

Génération de l'ordre CREATE (code 09)

Code intermédiaire :

- fragment fixe
- n° interne de l'objet concerné
- niveau d'imbrication de la réalisation à créer
- taille courante de la pile
- indicateur identifiant DDL ou technique
- déplacement dans l'enregistrement à créer dela zone des pointeurs
- idem pour les compteurs
- "indication field" SESAM d'écriture des blocs de pointeurs de l'enregistrement à créer
- "indication field" SESAM de réécriture de la zone de pointeurs de l'enregistrement à créer
- liste des indices des compteurs correspondant aux objets élémentaires obligatoires
- Pour chaque relation dont on crée des occurrences :
  - type de relation
  - numéro de l'inverse
  - déplacement du compteur dans la cible
  - indice dans la zone des compteurs de l'origine du compteur correspondant
  - différents indices des pointeurs concernés
  - liste des niveaux d'imbrication des réalisations cibles

procédure : - appel à SYSDML
- test du code de retour

Génération de l'ordre MODIFY (code 10)

Code intermédiaire : - fragment fixe
- Pour chaque réalisation à modifier :
  - numéro d'imbrication de la réalisation à modifier
  - nombre d'aspects à modifier
  - "indication field" SESAM
- "* *"

Procédure : - appel à SYSDML
- test du code de retour

Génération de l'ordre ATTACH (code 11)

Code intermédiaire : - fragment fixe
- niveau d'imbrication de l'origine
- Pour chaque cible : - numéro interne de la relation
  - numéro d'imbrication de la cible
- "$$$$$$"

Procédure : - appel à SYSDML"
- test du code de retour

Génération de l'ordre DETACH (code 12)

cf. ATTACH

Génération de l'ordre TRANSFER (code 13)

Code intermédiaire : - fragment fixe
- niveau d'imbrication de l'origine
- Pour chaque transfert partiel
  - numéro de la relation
  - taille de la liste FROM
  - taille de la liste TO
  - liste (FROM + TO) des niveaux d'imbrication des cibles FROM et TO
- "* *"

Procédure : - appel à SYSDML
- test du code de retour

Génération de l'ordre SUPPRESS (code 14)

Code intermédiaire : - fragment fixe
- niveau d'imbrication de la réalisation à supprimer

Procédure : - appel à SYSDML
- test du code de retour
  = 0 = correct
  sinon = aller à SYS-DUMP
- exécution éventuelle des sorties des blocs imbriqués
- branchement du paragraphe NEXT correspondant au bloc de la réalisation à supprimer.

Génération de l'ordre TEST (code 15)

Code intermédiaire : - fragment fixe
- type de test
- numéro d'imbrication (ou numéro de variable) du premier membre
- idem second membre

Procédure : - appel à SYSDML
- test du code de retour

Génération de l'ordre SAVE (code 16)

Code intermédiaire : - fragment fixe
- niveau de la réalisation
- numéro de la variable

Procédure : cf. TEST.

Génération de l'ordre MOVE (code 17)

Code intermédiaire : - fragment fixe
- numéro première variable
- numéro deuxième variable

procédure : cf. TEST.

Génération de l'ordre CLEAR (code 18)

Code intermédiaire : - fragment fixe
- liste des numéros des variables à vider
- "* *"

Procédure : cf. TEST.

Génération de COUNT (code 19)

L'expression COUNT (et ses paramètres) est remplacée par la désignation d'un élément du vecteur SYS-C :

SYS-C OF <étiquette> (<indice>)

°

° °

III. STRATEGIES DE COMPILATION

Un certain nombre d'ordres DML entraînent la génération de texte COBOL à trois endroits différents :
- dans la liste des arguments d'appel à l'interface (SYSDML),
- dans la description de la pile sous forme d'une découpe REDEFINES,
- dans la division PROCEDURE.

D'autre part, deux paramètres ne seront connus que lorsque l'on aura analysé le dernier ordre DML du programme; il s'agit du nombre d'éléments du vecteur de codes de retour SYS-ERAREA (ce nombre est le maximum réclamé par chaque ordre) et de la taille exacte de la pile.

Il n'est donc pas possible de construire directement le code objet par une simple analyse du code source.
Cependant, la structure rigide d'un programme DML, et le fait que le texte COBOL n'est pas analysé permettent d'envisager sans inconvénient une compilation en une seule passe.
Quant au problème de la dispersion de la génération, il sera réglé par un éclatement du code objet en trois fichiers dont la fusion fournira le programme objet complet.

Ces fichiers, qui ont pour noms internes WORK1, WORK2 et WORK3, vont recevoir les fragments suivants :

WORK1	WORK2	WORK3
IDENTIFICATION DIVISION ⋮ WORKING-STORAGE SECTION (liste arguments d'appel)	01 SYS-0002-STACK REDEFINES SYS-STACK. ⋮ (pile)	(zones de l'utilisateur) PROCEDURE DIVISION. ⋮ CALL SYSDML ⋮

Cette découpe du code objet permet une écriture purement séquentielle de ces fichiers.
Lorsque le programme source est complètement analysé, le compilateur dispose de ces trois fichiers, du nombre maximum d'éléments dans le vecteur de code de retour et de la taille maximum de la pile.

Supposons à titre d'exemple que ces deux derniers paramètres ont pour valeur 10 et 5096 (bytes).

Une phase de fusion permet alors de construire dans le fichier de nom interne GENFILE le programme objet COBOL :

GENFILE

Contenu de WORK1

```
01 SYS-ERAREA.
   02 SYS-ERNBR PIC 9(3) COMP.
   02 SYS-ERLIST PIC XX OCCURS 0010.

01 SYS-STACK.
   02 SYS-ERROR    PIC XX.
   02 SYS-STATUS   PIC XX.
   02 SYS-MAC-NUM  PIC X(4).
   02 SYS-REG      PIC 9(3) COMP.
   02 FILLER       PIC X(4096).
   02 FILLER       PIC X(1000).
```

Contenu de WORK2

Contenu de WORK3

On remarquera que l'espace de la pile est réservé par blocs de 4K bytes maximum, ceci en raison des contraintes de compilateur COBOL utilisé.

o

o o

IV. ARCHITECTURE DU COMPILATEUR DML

Les programmes permettant de réaliser une compilation complète mettent en oeuvre deux programmes :

- le compilateur DML proprement dit
- le programme de mise en forme des diagnostics DML et COBOL.

4.1. Le compilateur DML

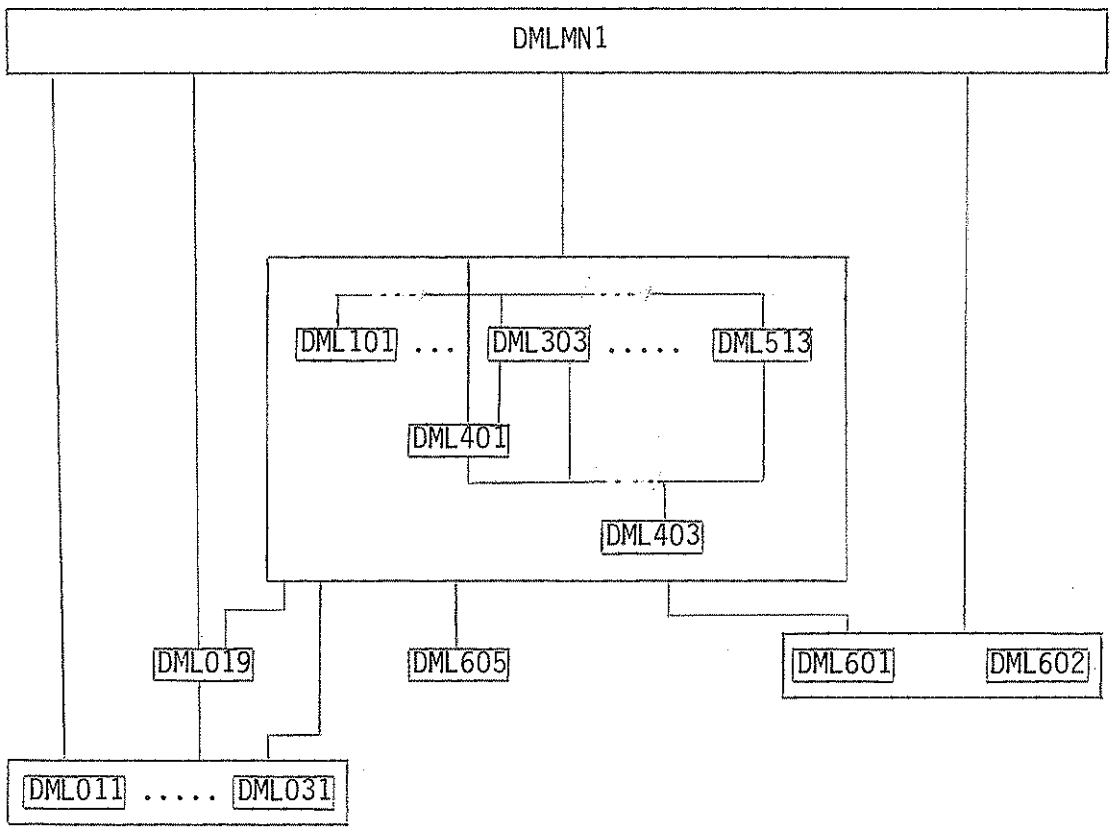
Le compilateur est constitué de trois classes de modules :

- le module directeur, qui contrôle et oriente la compilation des ordres DML qu'il détecte dans le programme source; son nom est DMLMN1;
- les modules spécialisés réalisant chacun la compilation d'un ordre DML, ou d'ordres DML similaires :

  DML101 : OPEN
  DML115 : TEST
  DML116 : SAVE
  DML117 : MOVE
  DML118 : CLEAR
  DML119 : COUNT
  DML201 : ATTACH et DETACH
  DML202 : SUPPRESS
  DML303 : REACH
  DML304 : EXIT et NEXT
  DML306 : CLOSE et END
  DML401 : PREPARE et ALLOCATE (ainsi que certaines fonctions du REACH)
  DML402 : CREATE
  DML510 : MODIFY
  DML513 : TRANSFER.

- les modules de service :
  - module relatif à la protection : DML403
  - modules d'accès aux fichiers standard : DML011 à DML031
  - module d'accès au fichier des schémas : DML605
  - modules de manipulation de chaînes de caractères DML601 et DML602.
  - module de formattage des messages d'erreurs DML019.

L'architecture de cet ensemble est (très schématiquement) la suivante :



4.2. Mise en forme des diagnostics DML et COBOL

Ce programme est constitué d'un programme principal (DMLMN2) et de modules d'accès standard à des fichiers.

o

o o

V. LES TABLES DU COMPILATEUR (programme DMLMN1)

Nous ne décrirons ici que les zones de données partagées et mises à jour par plusieurs modules, ou dont la structure doit être commune à tous les modules qui en manipulent un exemplaire.
Leur description est stockée en bibliothèque (DML.LIB-COPY) et peut être insérée par des ordres COPY.

On trouvera dans l'annexe 1 du listing de documentation le contenu de cette bibliothèque (dont l'ordre est quelque peu différent de celui de la présentation ci-dessous).

Nous décrirons successivement les zones élémentaires, la pile du compilateur, les paramètres de contrôle des entrées/sorties et d'autres zones diverses.

5.1. Les zones élémentaires

DEPL et LONG : paramètres de la fonction NXTSYM d'acquisition du symbole suivant dans la carte courante.

STK-TOP : taille active (en bytes) de la pile du programme généré à l'endroit courant de la procédure du programme source.

MAX-TOP : valeur maximum de STK-TOP depuis le début de l'analyse du programme source.

MAC-NBR : numéro de séquence de l'ordre DML en cours d'analyse.

TDIAG : nombre d'ordres DML contenant une erreur au moins depuis le début du programme source.

LAB-GEN : numéro constitutif du dernier identifiant COBOL généré.

ERR-TYPE : code d'erreur de compilation; est constitué de :
- ERR-PRE : code de l'ordre DML
- ERR-SU : numéro d'erreur pour cet ordre.

SW-COUNT : réservé à la compilation du COUNT (voir listing de documentation annexe 1).

5.2. La pile du compilateur

Les paramètres de gestion de la pile

IND-STK : indice du sommet actuel de la pile

MAX-ELEM-STACK : nombre maximum d'éléments dans la pile; est mis à 100 dans DMLMN1 dans la version actuelle.

Structure de la pile

MAC-STACK1 : nom de l'élément de la pile; cet élément contient les informations suivantes :

- MAC-TYPE : numéro de l'ordre DML
  - 01 OPEN
  - 03 REACH
  - 07 ALLOCATE
  - 08 PREPARE
  - 09 PREPARE ayant été référencé par un CREATE
- MAC-NAME : nom de l'étiquette déclarée par l'ordre DML.
- MAC-NUM : numéro séquentiel de l'ordre DML dans le programme.
- MAC-FROM : indice dans la pile de l'élément relatif à l'ordre DML spécifié dans la clause FROM (implicite ou explicite); dans le cas d'un OPEN, contient l'indice de l'élément lui-même.
- MAC-IN : indice dans la file de l'élément relatif à l'ordre DML spécifié dans la clause IN; pour les ordres OPEN et REACH, ALLOCATE et PREPARE sans IN, contient l'indice de l'élément lui-même.
- MAC-NASP : nombre d'aspects dont la valeur sera demandée au noyau.
- MAC-PTR1 : pointeur dans la table des aspects (ASP-STACK) vers le premier élément de la liste des aspects dont la valeur sera demandée au noyau.
- MAC-PTR2 : réservé à la compilation du CREATE; sert au calcul de la longueur de la commande SESAM (INDICATION FIELD).
- MAC-DELP-CNT : - idem -
- MAC-NOBJ : numéro interne de l'objet concerné par l'ordre DML; vaut "000" pour un OPEN.
- MAC-NREL : numéro de la relation empruntée pour accéder aux réalisations de l'objet; n'est valide que pour un ordre REACH VIA (explicite ou implicite); vaut 0 dans les autres cas.
- MAC-TREL : type d'accès : 1 si accès par relation entre objets complexes
  0 dans tous les autres cas.
- MAC-UI : identifiant d'utilisateur destiné au noyau SPHINX; est constitué de :
  - MAC-UI1 : numéro de la base de donnée concernée
  - MAC-UI2 : caractère complémentaire identifiant un utilisateur d'une base de données dans un programme (voir fonctionnement de SPHINX et SESAM); vaut "A" pour un OPEN.
- MAC-DBN : nom de la banque de données concernée.

- MAC-TOP : taille (en bytes) de la pile de la zone de communication dans le programme généré, avant la compilation de l'ordre DML relatif à l'élément.
- MAC-TREC : déplacement (en bytes) dans la pile du programme généré, de la zone de réception des valeurs d'objets élémentaires (vaut 0 pour un OPEN).
- MAC-NREC : longueur (en bytes) dans la file du programme généré, de la zone relative à la réalisation d'objet complexe concerné par l'ordre DML (vaut 0 pour un OPEN).
- MAC-STATUS : code d'état propre au module d'analyse de l'ordre DML concerné.

5.3. Paramètres de contrôle des entrées/sorties

A chaque fichier est associé un paramètre de code opération (suffixe -IO-STATE) et un paramètre de code de retour (suffixe -FILE-STATE).

Le nom de ces paramètres est préfixé par un identifiant du fichier concerné selon la convention suivante :

préfixe	fichier
WRK1	WORK1
WRK2	WORK2
WRK3	WORK3
DIAG	FDIAG
IPT	SYSIPT
LST	SYSLST
GEN	GENFILE

Une description complète de ces paramètres est données dans le paragraphe : Les modules d'accès aux fichiers.

5.4. Zones diverses

Table des aspects SESAM

Cette table, gérée comme une pile, contient les listes des noms des aspects dont la valeur sera demandée au noyau; elle est référencée par les zones MAC-NASP et MAC-PTR1 de la file du compilateur.

MAC-ASP-STACK : taille maximum de la table (initialisé à 333 dans le programme principal).

IND-ASP-STACK : indice dans la table.

ASP-STACK : élément de la table.

Zone de code de retour de DML605

Cf. paragraphe "Les modules d'accès aux fichiers".

STATUS : code de retour SESAM (2 caractères)

RCPT-UI : numéro d'utilisateur du dernier appel.

RCPT-KEY : "data-base-key" concernant le dernier appel; inutilisé dans le compilateur et ignoré du simulateur DML605.

Dessin de carte (IPT-CARTE)

Permet l'accès aisé et uniforme aux différentes zones d'une ligne de programme DML.

Dessin d'une ligne de programme généré (INSTRUCT)

Offre plusieurs découpes correspondant aux différents types de lignes à générer : commentaires, étiquettes, instructions.

Zone de réception d'un symbole (SYMBOLE)

Permet la désignation directe des n premiers caractères du symbole lu, avec n = 1, 2, 8, 15, 17 et 52.

o

o o

VI. FICHIER DES SCHEMAS INTERNES

Ce fichier, de nom interne SCHEMA-S, contient la description d'une ou plusieurs bases de données qui est nécessaire à la compilation de programmes DML. Cette description permet la vérification du programme et la génération des zones de communication.
Il s'agit d'un fichier séquentiel indexé à plusieurs types d'articles.

Le compilateur DML est conçu pour consulter ce fichier comme s'il était géré directement par SESAM; cependant, dans la version actuelle, les ordres d'interrogation sont interceptés par un interface qui permet de travailler sur un simple fichier ISAM.
Cet interface est constitué par le programme DML605 qui simule le comportement de SESAM.

Le dictionnaire

En début du fichier ISAM se trouve un catalogue qui indique le nom des bases de données décrites ainsi que leur caractère d'identification interne ("numéro" de la base de données).
Chaque base de données y est décrite par l'article suivant :

1. Clé (11 caractères)
1.1. Préfixe "$";
1.2. Nom de la base de données (9 caractères);
1.3. Suffixe "1 espace".

2. Numéro de la base de données (1 caractère).

L'accès à ce dictionnaire est réalisé par un positionnement sur l'article de clé égale à $ <nom racine>.

Les schémas internes

Une base de données est décrite par la séquence d'articles suivante :

- un article correspondant à l'objet racine;
- une sous-séquence pour chaque objet complexe constitué de :
  - un article correspondant à l'objet complexe
  - un article par objet élémentaire associé à cet objet complexe; les articles décrivant les composants ou attributs d'un objet élémentaire composé suivent directement l'article décrivant ce dernier.
- un article décrivant chaque relation d'accès implémentée; rappelons qu'à chaque relation déclarée dans un schéma correspondent deux relations d'accès inverses implémentées.

Tous les articles ont une clé de 11 caractères contenant un préfixe de deux caractères :

- numéro de la base de données (spécifié dans le dictionnaire)
- type d'élément décrit : $: racine
  O : objet
  R : relation

Ces préfixes permettent de conserver la séquence des articles décrite plus haut.

Article décrivant la racine

1. Clé
1.1. Préfixe
1.1.1. Numéro de la base de données
1.1.2. "$"

1.2. Reste : "$$$$ "

2. Nom de la racine (9 caractères)

Article décrivant un objet complexe

1. Clé
1.1. Préfixe
1.1.1. Numéro de la base de données
1.1.2. "O"

1.2. Numéro interne de l'objet complexe (3 caractères numériques)
1.3. Code objet complexe : "$"
1.4. Reste : 5 espaces

2. Nom de l'objet complexe (18 caractères)

Article décrivant un objet élémentaire

1. Clé
1.1. Préfixe
1.1.1. Numéro de la base de données
1.1.2. "O"

1.2. Numéro interne de l'objet complexe associé (3 caractères numériques)
1.3. Code objet élémentaire : "*"
1.4. Code de présence : "M" correspondant à OCCURS(1,J)
"O" correspondant à OCCURS(Ø,J)
1.5. Numéro interne de l'objet élémentaire pour l'objet complexe (3 caractères numériques)
1.6. Reste : 1 espace.

ont une liste de noms d'aspects égale à "AAA"; à la zone FILLER ne sont associés ni indice de compteur ni liste d'aspect.

- Liste des pointeurs

Cette liste est décrite comme un vecteur dont l'élément, de nom SYS-PTR et de description "PIC X(4)" est répété ("OCCURS") autant de fois qu'il y a de pointeurs; s'il n'y a pas de pointeurs associés aux réalisations de l'objet complexe, ce vecteur n'est pas déclaré.
A cette description sont associées : la longueur totale de la liste et la liste des noms d'aspects (de 1 à 5).
Cet article ne fait pas l'objet d'une génération dans la zone de communication puisque les pointeurs ne sont pas transmis aux programmes de l'utilisateur.

- Liste des compteurs

Cette liste est décrite comme un vecteur dont l'élément, de nom SYS-C et de description "PIC 9(5) COMP" est répété ("OCCURS") autant de fois qu'il y a de compteur : s'il n'y a pas de compteurs associés aux réalisations de l'objet complexe, ce vecteur n'est pas déclaré.
A cette description sont associées : la longueur totale de la liste et la liste des noms d'aspects (1 à 5).
Cet article, s'il existe, entraînera la génération de la description du vecteur dans la zone de communication.

Les articles correspondant à l'identifiant DDL ou SYS-KREC, suivis de l'article éventuel FILLER, sont placés directement à la suite de l'article décrivant l'objet complexe. Ils sont suivis des articles des pointeurs et des compteurs. Du point de vue de leur clé, ces articles sont considérés comme décrivant des objets élémentaires de nom SYS-KREC, FILLER, SYS-PTR, SYS-C associés à l'objet complexe. Cependant, le nom SYS-KREC est transformé en FILLER lors de la génération de la zone de communication.

Article décrivant une relation

1. Clé
1.1. Préfixe
1.1.1. Numéro de la base de données
1.1.2. "R"

1.2. Numéro interne de la relation (3 caractères numériques)
1.3. Reste : "$ "

2. Nom de la relation (18 caractères)

Si la relation porte un nom déclaré dans le texte DDL, c'est lui qui est renseigné. Si la relation est déclarée WITHOUT NAME, le nom interne "$" lui est affecté.

2. Description COBOL de l'objet élémentaire

2.1. Numéro de niveau (2 caractères numériques)
Un objet élémentaire directement relié à l'objet complexe est au niveau 03.

2.2. 1 espace

2.3. Nom de l'objet élémentaire (15 caractères)

2.4. 1 espace

2.5. Description COBOL : PICTURE, USAGE, OCCURS suivie de "." (43 caractères).

3. Code de composition (1 caractère) : espace : objet élémentaire simple
"C" : objet élémentaire composé.
Ce code n'est valide que dans l'article d'un objet élémentaire directement associé à l'objet complexe; il est remplacé par un espace dans les articles des objets élémentaires attributs.

4. Indice du compteur correspondant dans le vecteur des compteurs SYS-C (voir ci-dessous). Seuls les objets élémentaires non attributs possèdent un indice valide; il est mis à "ØØØ" pour les objets attributs (3 caractères numériques).

5. Longueur totale maximum des valeurs de l'objet élémentaire pour une réalisation de l'objet complexe; elle vaut "ØØØØ" pour les autres (4 caractères numériques).

6. Liste des noms internes d'aspects SESAM représentant les valeurs de l'objet élémentaires; elle n'est valide que pour les objets élémentaires non attributs (5 fois 3 caractères).

Outre ces articles descriptifs d'objets élémentaires, on trouvera pour chaque objet complexe de 0 à 4 articles de format analogue décrivant des zones liées à l'implémentation des structures de données; ces zones sont relatives à la structure de l'aspect d'ordre (clé du séquentiel indexé de SESAM), à la liste des pointeurs et à la liste des compteurs qui composent les enregistrements SESAM associés aux réalisations de l'objet complexe :

- Structure de l'aspect d'ordre

Si un objet élémentaire est défini comme identifiant, ses valeurs seront contenues dans un sous-champ de l'aspect d'ordre; si la longueur de ces valeurs est inférieure à la longueur de l'aspect d'ordre, la différence sera comblée par une zone FILLER.
Si aucun objet identifiant n'est déclaré dans le texte DDL, un identifiant technique est défini; il est constitué de 8 octets, et porte le nom de SYS-KREC; il est suivi d'une zone FILLER de 19 caractères complétant l'aspect d'ordre.
Les zones identifiant DDL, SYS-KREC et FILLER sont décrites de niveau 03; seul l'identifiant DDL possède un indice de compteur; l'identifiant DDL et SYS-KREC

Une relation non déclarée dans le texte DDL portera le nom interne "#"; une telle relation est l'inverse d'une relation déclarée.

3. Identification de l'objet complexe origine sous la forme d'un fragment de sa clé :

3.1. "O"

3.2. numéro interne (3 chiffres)

3.3. "ß"

4. Identification de l'objet complexe cible (cf. origine)

5. Identification de la relation inverse sous la forme d'un fragment de sa clé :

5.1. "R"

5.2. numéro interne

5.3. "ß"

6. Valeurs des caractéristiques I-J, K-L : 4 zones de 5 caractères numériques affectées, dans l'ordre, à I,J,K,L. La valeur "99999" est mise pour "∞".

7. Nom de l'aspect SESAM contenant les blocs de pointeurs éventuels (3 caractères). Cette zone est à blanc s'il n'existe pas de blocs de pointeurs pour cette relation, vue comme chemin d'accès unidirectionnel.

8. Nom de l'aspect SESAM contenant les blocs de pointeurs éventuels relatifs à la relation inverse (3 caractères). Cette zone est à blanc s'il n'existe pas de blocs de pointeurs pour cette relation inverse, vue comme chemin d'accès unidirectionnel.

9. Type d'implémentation (1 caractère)
   Valeurs permises :
   1 = 1 pointeur dans SYSPTR de l'origine et 1 pointeur dans SYSPTR de la cible (relations 1-1)
   2 = blocs de pointeurs attachés à l'origine et 1 pointeur dans SYSPTR de la cible (relations 1-n).
   3 = 1 pointeur dans SYSPTR de l'origine et blocs de pointeurs attachés à la cible (relations n-1)
   4 = blocs de pointeurs attachés à l'origine et blocs de pointeurs attachés à la cible (relations m-n).

10. Indice d'un élément SYS-PTR relatif à l'enregistrement de l'origine (3 caractères numériques).
    Cet élément de SYS-PTR représente :
    - le pointeur vers la cible dans le type d'implémentation 1 et 3;
    - le pointeur vers le premier bloc de pointeurs dans les implémentations 2 et 4.

11. Indice d'un élément dans SYS-PTR relatif à l'enregistrement de l'origine (3 caractères numériques).
    Cet élément est un pointeur vers le dernier bloc de pointeurs dans les implémentations du type 2 et 4.
    L'indice est "ØØØ" dans les implémentations 1 et 3.

12. Indice d'un élément dans SYS-PTR relatif à l'enregistrement de l'origine (3 caractères numériques).
    Cet élément est un pointeur vers le bloc de pointeurs qui est probablement le moins plein (implémentations 2 et 4).
    L'indice est à "ØØØ" dans les implémentations 1 et 3.

13. Indice d'un élément dans SYS-PTR relatif à l'enregistrement de l'origine (3 caractères numériques).
    Cet élément contient le nombre de pointeurs dans le bloc probablement le moins plein (implémentations 2 et 4).
    L'indice est à "ØØØ" dans les implémentations 1 et 3.

14.
15. Copie des champs 10,11, 12 et 13 de l'article relatif à la relation inverse.
16.
17.

18. Indice dans SYS-C de l'enregistrement de l'origine du compteur attaché à la relation (3 caractères numériques).

19. Indice dans SYS-C de l'enregistrement de la cible du compteur attaché à la relation inverse (3 caractères numériques).

On trouvera à la fin du chapitre sur le compilateur DDL, le listing d'un schéma élémentaire.

°

° °

VII. LES MODULES PRINCIPAUX DU COMPILATEUR.

Par opposition aux modules de service ou utilitaires, nous désignons par ce titre le module directeur (DMLMN1) et les modules de compilation.

Le rôle et le fonctionnement de ces modules font l'objet d'une documentation sur listing; par ailleurs chaque programme contient une documentation plus complète sous forme de commentaires (description fonctionnelle et technique, forme des zones générées, codes d'erreur, etc).

Néanmoins, nous décrirons très brièvement les fonctions de ces modules.

7.1. Le module directeur

- Il initialise différentes zones de travail et paramètres;
- il ouvre les fichiers nécessaires (via les modules d'accès);
- il parcourt le texte source en réalisant :
    - l'éclatement sur les trois fichiers de travail WORK1, WORK2 et WORK3
    - l'édition du listing source
    - le repérage des ordres DML et l'appel du module de compilation correspondant,
- il contrôle les erreurs (niveau de gravité : récupérables ou arrêt immédiat)
- il clôture la compilation en fusionnant les fichiers de travail en un fichier du programme généré (s'il n'y a pas eu d'erreur fatale) et en fermant les fichiers.

7.2. Les modules de compilation

Chacun de ces modules est relativement automone.
Il contrôle la phase DML du point de vue de la syntaxe et de la sémantique; il construit et écrit le texte généré dans les fichiers de travail, il édite les diagnostics nécessaires et met à pour les tables du système pour y consigner des renseignements utiles pour la compilation des ordres suivants ou pour y supprimer (END ou CLOSE) des renseignements qui deviennent périmés.

o

o o

VIII. LES MODULES D'ACCES AUX FICHIERS

A chaque fichier utilisé par le compilateur DML, est associé un module d'interface réalisant les différentes opérations nécessaires : ouvertures, lecture, écriture, fermetures, et contrôlant l'ordre et la validité de ces opérations.

Pour la plupart des fichiers, le module d'accès est standard tant en ce qui concerne son mode d'appel que sa structure interne, ce qui assure au compilateur une grande indépendance vis à vis des fichiers eux-mêmes.
Le fichier des schémas de compilation, eu égard à ses modes d'accès très spécifiques, est doté d'un module d'accès spécialisé.

A. Les modules d'accès standard

L'appel à un module de ce type s'accompagne du passage de trois paramètres :

- une zone de données à émettre ou à recevoir; cette zone a une structure propre à chaque fichier.
- un code d'opération de nom <id-fichier>- IO-STATE (en un caractère numérique) choisi parmi les valeurs suivantes :
  1 ouvrir en mode OUTPUT
  2 écrire le contenu de la zone
  3 fermer en mode OUTPUT
  4 ouvrir en mode INPUT
  5 lire l'enregistrement suivant et le ranger dans la zone
  6 fermer en mode INPUT
- un code de retour de nom <id-fichier>-FILE-STATE (un caractère numérique) choisi parmi les valeurs suivantes :
  0 opération exécutée normalement
  1 fin de fichier en mode INPUT
  2 erreur

Au sein du module, l'état du fichier est représenté par une variable rémanente prenant ses valeurs parmi :

0 fichier fermé
1 fichier ouvert en mode OUTPUT
2 fichier ouvert en mode INPUT

A.1. Fichier de travail WORK1 - module d'accès DML011

Ce fichier séquentiel contient le début du programme source et les arguments d'appel à SYSDML.

L'appel au module d'accès s'effectue comme suit :

CALL WRK1 USING ZONE, WRK1-IO-STATE, WRK1-FILE-STATE.

A.2. Fichier de travail WORK2 - module d'accès DML012

Ce fichier séquentiel contient les lignes de description de la pile du programme généré.

L'appel au module d'accès s'effectue comme suit :
CALL WRK2 USING ZONE, WRK2-IO-STATE, WRK2-FILE-STATE.

A.3. Fichier de travail WORK3 - module d'accès DML013

Fichier séquentiel contenant la section WORKING-STORAGE du programme source et la division PROCEDURE.

L'appel au module d'accès s'effectue comme suit :
CALL WRK3 USING ZONE, WRK3-IO-STATE, WRK3-FILE-STATE.

A.4. Fichier des messages d'erreurs FDIAG - module d'accès DML014

Fichier séquentiel contenant les messages d'erreurs créés par le compilateur DML.
Appel au module :
CALL DIAG USING MSG-ERROR, DIAG-IO-STATE, DIAG-FILE-STATE.

A.5. Fichier des messages d'erreurs FDIAG - module d'accès DML015

Ce module permet d'ajouter au fichier FDIAG créé par le compilateur DML (voir DML014) la liste des messages d'erreurs créés par le compilateur COBOL; cette phase d'extension est réalisée par le programme DMLMN2 de constitution des listings de diagnostic. Le fichier FDIAG ainsi étendu pourra être consulté au terminal au moyen d'un éditeur.
Appel du module d'accès :
CALL CDIAG USING PARAM-ERR, IO-STATE, FILE-STATE.

A.6. Fichier du programme généré GENFILE- module d'accès DML016

Ce fichier séquentiel est créé par fusion des fichiers WORK1, WORK2 et WORK3 à la fin de la compilation DML.
Appel du module d'accès :
CALL GENERE USING CARTE, GEN-IO-STATE, GEN-FILE-STATE.

A.7. Fichier du programme généré ISOURCE - module d'accès DML017

Ce module permet de relire dans le programme DMLMN2 le fichier GENFILE créé par le module DML016 et qui porte le nom interne ISOURCE.
Appel du module d'accès :
CALL GEN USING PARAM-GEN, IO-STATE, FILE-STATE.

A.8. Fichiers d'erreurs COBOL ERRFIL - module d'accès DML018

Ce fichier est créé par le compilateur COBOL. Il est lu dans le programme DMLMN2 grâce à l'appel au module d'accès :
CALL SYSERR USING PARAM-ERR, IO-STATE, FILE-STATE.

A.9. Listing source et diagnostic - module DML021

Le module écrit sur la sortie standard imprimante SYSLST, une ligne du listing du programme source mis en forme et garni des diagnostics DML.
Il est appelé comme suit :
CALL LIST USING IPT-CARTE, LST-IO-STATE, LST-IO-STATE.

A.10. Listing source et diagnostic - module DML022

Le module complète le listing source créé par DML021 en y ajoutant les messages d'erreurs du compilateur COBOL. Cette extension est réalisée par le programme DMLMN2 qui appelle le module d'accès comme suit :
CALL DIAGNT USING PARAM-WRER, IO-STATE, FILE-STATE.

A.11. Fichier du programme source - module d'accès DML031

Le module lit les lignes du programme source (format carte) sur le fichier d'entrée standard SYSIPT.
Appel du module :
CALL SOURCE USING IPT-CARTE, IPT-IO-STATE, IPT-FILE-STATE.

B. Module d'accès au fichier des schémas de compilation

Le fichier porte le nom interne de SCHEMA-S; il est d'organisation séquentielle indexée; la description du format et du contenu de ses articles a été donnée préalablement.

Le module d'accès, de nom DML605, et de point d'entrée SYSSCH est en réalité un simulateur du système SESAM en ce qui concerne les opérations d'accès séquentiel, séquentiel indexé et par fichier inverse.
L'appel du module s'effectue comme suit :
CALL SYSSCH USING I-FIELD, RCPT-FIELD, RESP-FIELD, QUES-FIELD.
où - I-FIELD est le code de commande de l'opération demandée
- RCPT-FIELD est un code (complexe) de retour (voir Tables Internes)
- RESP-FIELD est la zone de réception des valeurs demandées dans I-FIELD
- QUES-FIELD est la zone contenant les valeurs de comparaison des critères spécifiés éventuellement dans I-FIELD.

Remarque : le module ne reconnaît que les codes de commande strictement nécessaires à l'exécution du compilateur DML. Sa structure permet cependant de l'étendre aisément à la simulation de fonctions plus nombreuses.

De plus amples renseignements pouvent être trouvés à la fois dans les brochures d'utilisation de SESAM et dans la documentation interne du programme DML605.

o

o o

IX. AUTRES MODULES DE SERVICE

9.1. Module de mise en forme des messages d'erreur DML019

Ce module construit les messages d'erreur à partir du code d'erreur qui lui est transmis par le compilateur DML. Le message ainsi construit est écrit sur le fichier FDIAG grâce au module DML014.

L'appel se fait comme suit :

CALL UNDIAG USING MESSAGE-IN, DIAG-IO-STATE, DIAG-FILE-STATE.

où - MESSAGE-IN est structuré en

- SEQNBR : numéro de séquence de l'instruction erronée
- ERRNBR : code d'erreur
- PARAM : paramètre éventuel

- DIAG-IO-STATE et DIAG-FILE-STATE : cf. DML014; ces paramètres sont simplement transmis de DML014 au programme appelant.

9.2. Modules de manipulation de caractères

Module de transfert de chaîne de caractères (DML601)

Rôle : copier une chaîne de LONG caractères de la position (ZONE1, POS1) vers la position (ZONE2, POS2) où LONG est le nombre de caractères à copier
ZONE1 et ZONE2 sont des identificateurs de zones COBOL
POS1 et POS2 sont des positions à partir du début des zones ZONE1 et ZONE2.

Appel :

CALL STRING USING ZONE1, POS1 , ZONE2, POS2, LONG

Module d'acquisition d'un symbole dans une chaîne (DML602)

Rôle : recherche le symbole suivant dans une zone de format carte à partir d'une position de départ.
La notion de symbole, qui est définie précisément dans le listing de documentation, inclut l'identificateur COBOL, les signes spéciaux, les caractères de ponctuation et toute chaîne entre guillemets.

Appel :

CALL NXTSYM USING ZONE, DEPL, SYMBOLE, LONG

où ZONE est la zone source
DEPL est la position du premier caractère suivant à analyser
SYMBOLE doit recevoir le symbole trouvé
LONG est la longueur du symbole trouvé.

N.B. DEPL est mis à jour par NXTSYL de manière à toujours vérifier sa définition.

9.3. Module de gestion de la protection (DML403)

Ce module réalise quatre fonctions auxquelles correspondent quatre points d'entrée.

1. Déclarer que l'objet complexe spécifié peut faire l'objet d'un accès (entrée MACCESS); son numéro est ajouté à la liste des demandes d'accès s'il n'en fait pas déjà partie.
2. Déclarer que l'objet complexe spécifié peut faire l'objet d'une modification (entrée MUPDATE); son numéro est ajouté à la liste des demandes de modification s'il n'en fait pas déjà partie.
3. Générer l'ordre de demande de protection; cette génération a lieu avant le traitement d'un ordre DML REACH,ALLOCATE ou PREPARE dont l'élément n'est précédé dans la pile du compilateur que par des éléments relatifs à des OPEN. L'argument de cet appel généré est une zone spécifiant en particulier les listes de demandes d'accès et de demandes de modification (entrée MINIT).
4. Génération de l'argument de la demandede protection.
   Cette génération a lieu lorsque l'on rencontre l'ordre DML de fermeture du bloc avant lequel la demande de protection a été générée; on dispose en effet à cet endroit des listes complètes des objets susceptibles d'être demandés ou modifiés (entrée MGEN).

Les entrées MACCESS et MUPDATE demandent en argument le numéro de la base de données et celui de l'objet complexe.
L'entrée MINIT demande en argument l'adresse des tables internes.
L'entrée MGEN ne demande pas d'argument.

9.4. Programme d'édition du listing final - DMLMN2

Ce programme résoud essentiellement le problème de la non concordance entre les numéros de séquence des lignes du programme source et les numéros de ligne repris dans les diagnostics COBOL; ces derniers sont en effet les numéros de séquence des lignes du programme généré.

La rectification des numéros de séquence s'effectue par une lecture parallèle du fichier des diagnostics du compilateur COBOL, et du fichier du programme objet COBOL dont on compte les lignes; chaque ligne de ce dernier fichier contient le numéro de séquence du programme source.
L'identité du numéro de ligne spécifié dans un message COBOL et du compteur de ligne permet d'associer à tout message COBOL le numéro de séquence du programme source.

Les messages COBOL ainsi corrigés sont écrits à la suite des fichiers listing source et diagnostics déjà constitués par le compilateur DML.

o
o o

LA COMMUNICATION ENTRE TACHES

DANS LE SYSTÈME SPHINX

Développons quelque peu, maintenant, le fonctionnement du système de communication entre les différentes tâches présentes simultanément en mémoire virtuelle, lors d'une exploitation d'un ensemble de bases de données par le système SPHINX.

Cet exposé se divise en deux parties :

a. description des macros BS 2000 de communication entre tâches;
b. description des conventions particulières choisies pour le système SPHINX.

1. LE SYSTEME DE COMMUNICATION ENTRE TACHES (I.P.K. - BS 2000)

Une zone est réservée dans la mémoire virtuelle du système d'exploitation pour la communication entre tâches. A chaque tâche est associée, dans cette zone, une file d'attente de buffers de communication, pourvu que la tâche se soit fait connaître du système de communication (macro OPCOM).

Toute tâche connue est identifiée par un nom d'au plus huit caractères EBCDIC, elle peut envoyer vers d'autres tâches des buffers de communication et extraire de sa file d'attente ceux qui lui sont destinés.

Voici une description sommaire des macros de communication entre tâches.

1.1. OPCOM - OUVERTURE DE COMMUNICATION

Cette macro permet à une tâche de se faire connaître du système de communication par un nom de huit caractères, au plus. Ce nom devra être spécifié lors de l'envoi d'un buffer à cette tâche. La macro OPCOM ne peut être exécutée une seconde fois avant la fermeture de communication.

1.2. SEVNT - ENVOI D'UN BUFFER

La macro SEVNT (Send Event) permet à une tâche d'envoyer un buffer de longueur variable à une autre tâche, connue du système de communication, en la désignant par son nom. Si la tâche réceptrice est suspendue en attente d'un buffer, elle sera réactivée. Si elle est active, le buffer sera placé en queue de la file d'attente associée à la tâche.

1.3. REVNT - RECEPTION D'UN BUFFER

La macro REVNT (Receive Event) permet d'extraire de la file d'attente de la tâche, le buffer le plus ancien ou le plus ancien envoyé par une tâche

particulière. (Sélection sur le nom de cette tâche). Si la file est vide, l'exécution de la macro a pour effet de suspendre la tâche pour un temps, spécifié en paramètre, compris entre 1 et 600 secondes. Toutefois, la tâche sera réactivée dès qu'un buffer du type désiré sera placé dans la file d'attente.

Le buffer extrait est placé dans une zone spécifiée du programme. Si celle-ci est trop courte, le buffer ne sera pas extrait mais sa longueur sera fournie à la tâche qui pourra alors exécuter une seconde fois la macro en spécifiant une zone ayant la longueur nécessaire.
L'extraction d'un buffer de la file est réalisé par la recopie de celui-ci d'une zone de mémoire (système), dans une autre (utilisateur) et n'implique donc pas forcément sa suppression de la file. Un paramètre de la macro permet de commander ou non cette suppression.

1.4. RELBF - SUPPRESSION D'UN BUFFER DE LA FILE D'ATTENTE

Cette macro permet de supprimer le premier buffer de la file. Elle peut être utilisée en conjonction avec la macro REVNT, particulièrement lorsque les buffers sont de longueur indéterminée ou qu'une erreur se produit.

1.5 CLCOM - FERMETURE DE COMMUNICATION

La macro CLCOM supprime la tâche de l'ensemble de celles qui utilisent le système de communication. Une fois exécutée, aucun message ne pourra plus être envoyé à la tâche et celle-ci ne pourra plus en envoyer.
Malgré tout, il est autorisé que la file associée à la tâche ne soit pas vide au moment de l'exécution de CLCOM. Elle sera, alors, détruite sauf si le paramètre "KEEP" a été spécifié dans la macro. Dans ce dernier cas, la tâche pourra encore exécuter des macros REVNT et RELBF pour traiter des buffers restants. Une seconde exécution de CLCOM sera, en outre, nécessaire pour clôturer définitivement la communication.

2. CONVENTIONS DE COMMUNICATION ENTRE TACHES DANS LE SYSTEME SPHINX

2.1. COMMUNICATION ENTRE LE NOYAU SESAM ET LE NOYAU SPHINX

Le noyau SPHINX apparaît au noyau SESAM comme un programme d'application ordinaire (du système SESAM). C'est pourquoi la communication entre les noyaux SPHINX et SESAM, se fait via le module d'interface SESMOD, intégré au noyau SPHINX et standard pour tous les programmes d'application dans le système SESAM.

Le module SESMOD, utilisé normalement, se charge des fonctions suivantes :

- ouverture de communication sous le nom SES<numéro de tâche>. (F0)
- Analyse préliminaire des requêtes du programme d'application;
  constitution d'un buffer de communication sous un format reconnu par le noyau SESAM;
  envoi du buffer au noyau SESAM par une macro SEVNT. (F1)
- Réception du buffer contenant le réponse du noyau SESAM (macro REVNT);
  analyse du buffer ;
  placement des résultats dans les zones de travail du programme d'application. (F2)
- Fermeture de communication.

Il est nécessaire d'utiliser SESMOD comme interface entre le noyau SPHINX et le noyau SESAM, car l'anatomie des buffers de communication SESAM n'est pas connue. Malheureusement, une des fonctions de SESMOD n'est pas adaptée à son utilisation par le noyau SPHINX : c'est l'ouverture de communication (F0). En effet, lorsque le noyau SPHINX fait appel à SESMOD pour communiquer avec le noyau SESAM une première fois, cela veut dire qu'il a déjà reçu un buffer d'un programme d'application (DML ou NUL) et, donc, qu'il a déjà ouvert une communication.
Or, le premier appel à SESMOD va entraîner une seconde tentative d'ouverture de communication, ce qui n'est pas permis. De plus, l'ouverture de communication par SESMOD identifierait le noyau sous le nom SES<numéro de tâche>qui ne permettrait pas aux programmes d'application de communiquer avec le noyau SPHINX puisque le numéro de tâche de celui-ci leur est inconnu.
Cette partie du module SESMOD a donc été modifiée et c'est un autre module du noyau SPHINX qui exécutera l'ouverture de communication sous le nom SES0001.

REMARQUES

- Cette solution implique qu'aucun programme d'application SESAM, autre que le noyau SPHINX ne pourra travailler sous le numéro de tâche 0001.

- Le modèle SESMOD possède trois points d'entrée.
  1. SESAM : réalise les fonctions (F1) et (F2), consécutivement, sans rendre la main, entretemps, au programme utilisateur.

2. SESPUT : réalise la fonction (F1) puis rend la main.

3. SESGETW : réalise la fonction (F2).

Si, pour un programme utilisateur SESAM normal, l'utilisation du point d'entrée SESAM est, en général, suffisante, le noyau SPHINX, par contre, a intérêt à occuper le temps compris entre l'envoi d'une requête et la réponse, pour préparer l'envoi d'autres requêtes issues d'autres programmes DML ou NUL. C'est donc les points d'entrée SESPUT et SESGETW qui seront utilisés dans le noyau SPHINX.

2.2. COMMUNICATION ENTRE LE NOYAU SPHINX ET LES PROGRAMMES D'APPLICATION (DML ou NUL)

Le noyau SPHINX doit être chargé avant tout programme d'application. Chaque programme DML s'identifie au système de communication sous le nom TSN <numéro de tâche> et s'adresse au noyau par son nom SES0001.

Chaque fois que le noyau SPHINX reçoit un buffer de communication, celui-ci peut provenir soit du noyau SESAM (nom d'expéditeur = SESAM), soit d'un programme d'application. Dans le second cas, le buffer sera extrait et supprimé de la file d'attente du noyau SPHINX, puis directement traité. Par contre, si le buffer provient du noyau SESAM, il sera maintenu dans la file d'attente et la main sera donnée à SESMOD qui analysera le buffer et fournira la réponse mise en forme (Point d'entrée SESGETW).

L'INTERFACE DE COMMUNICATION

ENTRE LES PROGRAMMES DML ET LE NOYAU SPHINX

1. INTRODUCTION

Lors de l'édition des liens d'un programme DML, un certain nombre de modules du système SPHINX sont liés à ce programme, constituant l'interface entre celui-ci et le noyau SPHINX. On donne dans ce chapitre une description du fonctionnement général de l'interface et des principales zones de mémoires et modules qui le constituent.

2. ARCHITECTURE GLOBALE DE L'INTERFACE (Fig. 2)

Chaque programme d'application DML est traduit par le précompilateur DML en un programme COBOL ne contenant plus que des appels au module SYSDML, module principal de l'interface, en lieu et place des ordres DML (*).
La zone passée en argument (INST) contient, sous forme codée, les renseignements nécessaires à l'exécution de l'ordre par l'interface ou le noyau SPHINX. On trouve globalement, parmi ces renseignements, le code de l'ordre DML, son niveau d'imbrication, son numéro séquentiel dans le programme, le numéro de la base de données concernée, une ou plusieurs adresses relatives dans la zone de communication de l'utilisateur (SYS-STACK), des commandes pour le noyau SESAM, etc.
Le module SYSDML effectue une première vérification du contenu de INST puis donne la main au module spécialisé de traitement de ce type d'ordre (SYSo1A, SYSo2A, ..., SYS2oA).
On peut répartir les modules spécialisés selon deux classes : ceux qui communiquent avec le noyau et ceux qui ne le font pas. Les premiers sont les plus nombreux : leur travail consiste à vérifier, de manière plus fouillée, le contenu de INST; ensuite, ils constituent un buffer de communication (CBUFFER),destiné au noyau SPHINX, contenant les mêmes renseignements que la zone INST et, de plus, des valeurs extraites de la zone de l'utilisateur : SYS-STACK (valeurs d'objets élémentaires, compteurs d'occurrences, critères d'interrogation ...)
Le buffer de communication est alors envoyé au noyau SPHINX, via le module SYSCOM, également chargé de réceptionner la réponse. Il reste alors au module spécialisé à vérifier la validité de la réponse et à placer les résultats obtenus dans la zone de communication de l'utilisateur. Enfin, la main est rendue à SYSDML, puis au programme utilisateur.

Les modules spécialisés qui ne communiquent pas avec le noyau sont essentiellement ceux qui manipulent les variables de travail (ordres @CLEAR, @SAVE, @MOVE, @TEST). Ceux-ci travaillent sur deux zones de mémoire propres à l'interface : le stack de l'interface (STACK), contenant une description de chaque niveau actif dans le programme DML (type de la macro, référence à la réalisation courante, ...),

(*) Il existe des exceptions : @COUNT, @NEXT, @EXIT, qui ne génèrent pas d'appel à l'interface.

et la zone des variables de travail (VARAREA) contenant une description de l'état courant de chaque variable de travail$01,...,$99.

Qu'une erreur, grave ou non, ait été détectée ou pas, chaque appel à SYSDML entraîne une mise à jour de zones diagnostic que pourra consulter l'utilisateur (code de retour : SYS-ERROR,vecteur d'erreur : SYS-ERAREA, etc.)

En outre, si une erreur grave est détectée dans l'interface avant l'envoi du buffer de communication au noyau, il sera malgré tout nécessaire d'envoyer au noyau un buffer signalant cette erreur.

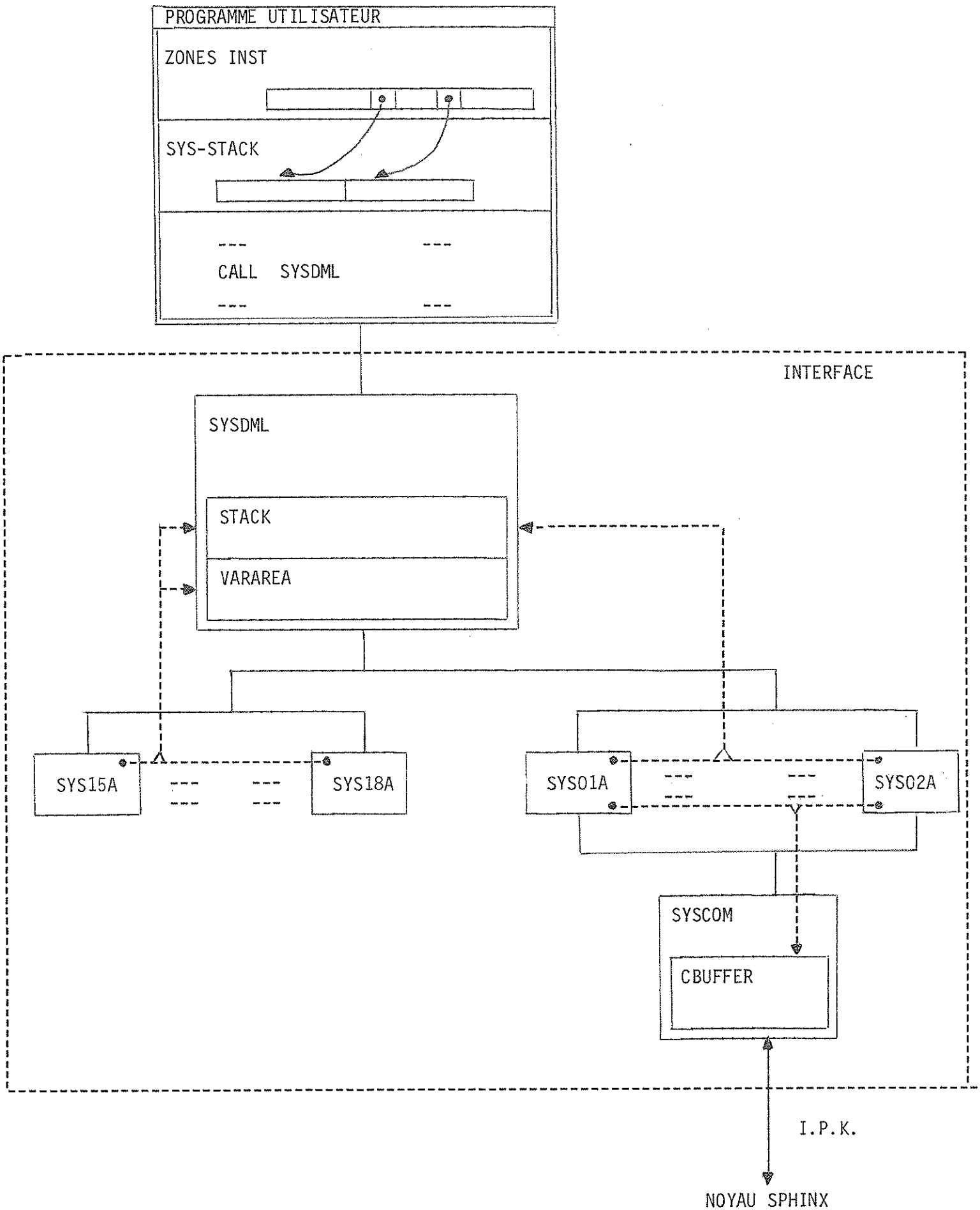


Figure 2.: Structure de l'interface

3. ZONES MANIPULEES PAR L'INTERFACE

3.1. LES ARGUMENTS D'APPEL DE SYSDML

Ces zones contiennent les renseignements nécessaires à l'interface pour effectuer les opérations exigées par les différents ordres DML.

3.1.1. LA ZONE DE COMMUNICATION DE L'UTILISATEUR (OU STACK DE L'UTILISATEUR)

Cette zone générée par le compilateur DML dans la WORKING-STORAGE SECTION du programme DML contient, à un moment donné, toutes les valeurs associées aux ordres DML en cours : valeurs d'objets élémentaires et compteurs d'occurrences issues des réalisations courantes, valeurs servant de critères d'accès. Dans le programme DML, elle porte le nom SYS-STACK et elle est passée en argument à l'interface lors de l'exécution d'un ordre OPEN; ensuite son adresse est mémorisée par celui-ci dans le premier mot du stack de l'interface (USTKA).
On trouvera une description de la structure de SYS-STACK dans la documentation du compilateur DML.

3.1.2. LE VECTEUR DIAGNOSTIC (OU VECTEUR D'ERREURS)

Cette zone est générée comme la précédente, par le précompilateur DML dans le programme utilisateur sous le nom SYS-ERAREA.
Après chaque retour de l'interface, dans le programme DML, elle contient un certain nombre de diagnostics partiels précédés du nombre de ces diagnostics.
Ce nombre n'est pas nul seulement après cinq DML, @ATTACH, @CREATE, @DETACH, @ MODIFY @TRANSFER.
L'adresse de cette zone est également passée à l'interface lors de l'exécution d'un ordre OPEN puis mémorisée dans le sixième mot du stack de l'interface (ERLST).

3.1.3. LES ZONES COMMANDES (INST)

Pour chaque ordre DML, une zone "commande" est générée dans le programme DML (deux pour l'ordre REACH), sous le nom SYS-<num-seq.> -<suffixe>.
Cette zone servira de premier argument à l'appel de SYSDML généré à la place de l'ordre DML, elle contient la description complète, sous forme codée, de l'ordre à exécuter. Le format de cette description varie d'un type d'ordre à un autre : on le trouvera dans les listings des modules spécialisés de l'interface et du compilateur DML. Toutefois, les 10 premiers bytes de chaque zone commande ont une signification standard dans tous les types d'ordres. La figure 3.1.3 donne la description assembleur du début de la zone commande :

- ICODE : code de l'ordre DML (2 caractères numériques)
- ILEVEL : niveau imbrication de l'ordre (3 caractères numériques)
- ISEQNBR : numéro séquentiel de l'ordre dans le programme DML (4 caractères numériques)
- IDBN : numéro de la base de données (1 caractère $\in$ {A,B,...,Z,1,...,6}).

L'adresse de la zone commande est placée par SYSDML dans le deuxième mot du stack de l'interface (INSTA).

```
STMNT M SOURCE

                #INST
1 *-------------------------------------------------------+
1 *                INSTRUCTION FORMAT.                    I
1 *-------------------------------------------------------+
1 *
1 *
1 *  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+-
1 *  I     I        I           I  I//////////
1 *  I  1  I   2    I     3     I4 I///  5  //
1 *  I     I        I           I  I//////////
1 *  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+-
1 *
1 *  (1) INST-CODE (DECIMAL).
1 *  (2) INST-LEVEL (DECIMAL).
1 *  (3) INST-SEQNBR.
1 *  (4) INST-DATA-BASE-NUMBER.
1 *  (5) SPECIFIC PARAMETERS.
1 *
1 INST      DSECT                        |INSTRUCTION FORMAT.
1 ICODE     DS     CL2                   |INST-CODE.
1 ILEVEL    DS     CL3                   |INST-LEVEL.
1 ISEQNBR   DS     CL4                   |INST-SEQNBR.
1 IDBN      DS     CL1                   |INST-DB-NUMBER.
```

Figure 3.1.3.

3.2. LES PRINCIPALES ZONES DE L'INTERFACE

3.2.1. LE STACK DE L'INTERFACE OU STACK DESCRIPTEUR (STACK)

Cette zone est divisée en 101 entrées de huit mots, numérotées de 0 à 100. La 0ème entrée (CONTROL) joue un rôle particulier : elle contient la liste des adresses des principales zones intéressant l'interface (sauf celle de STACK) et le nombre de niveau actif dans le programme DML. Une description assembleur en est donnée à la figure 3.2.1.1.
Seuls les six premiers mots de l'entrée sont significatifs; se sont, dans l'ordre :

- USTKA : adresse de la zone de communication de l'utilisateur (SYS-STACK)
- INSTA : adresse de la zone INST courante
- STKTOP : nombre de niveaux actifs dans le programme DML (et donc aussi dans STACK)
- VARWA : adresse de la zone des variables de travail (VARAERA)
- BUFA : adresse du buffer de communication (CBUFFER)
- ERLST : adresse du vecteur diagnostic dans le programme DML (SYS-ERAREA).

Si n est la valeur contenue dans STKTOP (codée en binaire), la ième entrée du stack ($1 \leqslant i \leqslant n$) donne une description du bloc courant de niveau i du programme DML.
La figure 3.2.1.2. donne la structure ASSEMBLEUR d'une de ces entrées (STKENTRY).
Précisons la signification de ses composants (ième entrée).

Bytes	Nom	Signification
1-2	CODE	Code de deux caractères numériques identifiant le type de l'ordre DML, constituant le premier ordre du bloc courant de niveau i. Les quatre codes possibles sont 01 (@OPEN), 03 (@REACH), 07 (@ALLOCATE), 08 (@PREPARE). A cet ordre correspond 0 ou 1 réalisation courante dont certaines caractéristiques sont décrites dans les zones suivantes.
3-6	SEQNBR	Numéro séquentiel du premier ordre DML du bloc courant de niveau i (4 caractères numériques)
7-7	DBN	Numéro de la base de données sur laquelle opère l'ordre DML dont il est question ci-dessus, (un caractère parmi A,B,C,...,Z,1,...,6)
9-10	OBJN	Numéro d'objet de la réalisation courante : nombre entier, compris entre 1 et 999, codé en binaire (seulement si CODE ≠ 0 1)
11-12	RELN	Si le premier ordre du bloc courant est un @REACH (CODE = 0 3), - si c'est un REACH via une relation entre objets complexes, RELN contient le numéro de cette relation, compris entre 1 et 500, codé en binaire, - si c'est un REACH sur variable de travail, RELN contient le numéro de cette variable, compris entre 1 et 99 codé en binaire.

13-14	FLEVEL	Si le premier ordre du bloc courant est un @REACH (CODE = 0 3), FLEVEL contient le niveau j du bloc dont la réalisation courante est l'origine de l'accès réalisé par le REACH. ($1 \leqslant j < i$) Si CODE(j) = 0 1, la réalisation en question est celle de l'objet racine de la base de données.
15-16	XLEVEL	Si le premier ordre du bloc courant contient une clause "IN <étiquette >", XLEVEL contient le niveau j du bloc dont le premier ordre a pour étiquette <étiquette> ($1 \leqslant j < i$). Sinon XLEVEL = i.
17-20	QFLDA	Si le premier ordre du bloc courant est un @REACH, QFLDA contient l'adresse absolue d'une zone du programme DML contenant des valeurs servant de critères d'accès pour cet ordre. La signification exacte et la structure de cette zone est exposé dans la documentation du compilateur DML (*).
21-24	RFLDA	Si le premier ordre du bloc courant n'est pas un @OPEN,RFLDA contient l'adresse absolue d'une zone du programme DML contenant les valeurs associées à la réalisation courante correspondant à cet ordre (si elle existe). Ces valeurs sont essentiellement des valeurs d'objets élémentaires et des compteurs d'occurrences de relations (*).
25-28	RECNBR	Si le premier ordre du bloc courant n'est pas un @OPEN, RECNBR = 0 (binaire) si la réalisation courante n'existe pas, sinon RECNBR est égal au numéro d'enregistrement SESAM correspondant à la réalisation.
29-30	QLENGTH	Longueur de la zone d'adresse QFLDA, si elle existe (demi-mot binaire).
31-32	RLENGTH	Longueur de la zone d'adresse RFLDA, si elle existe (demi-mot binaire).
9-26		Si le premier ordre du bloc est @OPEN, le nom de l'objet racine, suivi de 8 à 16 blancs.

(*) Ces zones font partie de la zone de communication de l'utilisateur (SYS-STACK).

```
)DR2  STMNT M SOURCE
                    #CNTR
4
5  1  *-----------------------------------------------------------+
6  1  *    DESCRIPTION OF THE FIRST 8 WORDS OF STACK.             I
7  1  *                  (INTERFACE)                              I
8  1  *-----------------------------------------------------------+
9  1  *                                                           I
10 1  *                                                           I
11 1  *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12 1  *  I           I           I           I           I           I
13 1  *  I     1     I     2     I     3     I     4     I
14 1  *  I           I           I           I           I
15 1  *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
16 1  *                                                           I
17 1  *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
18 1  *  I           I           I//////////////////////////////I
19 1  *  I     5     I     6     I//////////  7  //////////////I
20 1  *  I           I           I//////////////////////////////+
21 1  *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
22 1  *
23 1  *  (1) ADDR. OF USER-STACK.
24 1  *  (2) ADDR. OF CURRENT INSTRUCTION
25 1  *  (3) NUMBER OF ENTRIES IN STACK.
26 1  *  (4) ADDR. OF WORKING VARIABLES AREA.
27 1  *  (5) ADDR. OF COMMUNICATION BUFFER.
28 1  *  (6) ADDR. OF ERROR-LIST.
29 1  *  (7) RESERVED.
30 1  *
31 1  CONTROL  DSECT                          |CONTROL-FIELD FORMAT.
32 1  USTKA    DS     1F                      |ADDR. OF USER-STACK.
33 1  INSTA    DS     1F                      |ADDR. OF INSTRUCTION.
34 1  STKTOP   DS     1F                      |NBR OF ENTRIES IN STACK.
35 1  VARWA    DS     1F                      |ADDR. OF WORK-VAR-AREA.
36 1  BUFA     DS     1F                      |ADDR. OF BUFFER.
37 1  ERLST    DS     2F                      |ADDR. OF ERROR-LIST.
38 1                                          |RESERVED.
```

Figure 3.2.1.1.

```
DDR2  STMNT M SOURCE

                #STK
1 *-------------------------------------------------------------+
1 *                STACK ENTRY (INTERFACE).                     I
1 *-------------------------------------------------------------+
1 *
1 *
1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 *  I   I   I   I   I   I   I   I   I   I   I   I   I   I   I
1 *  I 1 I   I 2 I   I 3 I   I 4 I   I 5 I   I 6 I   I 7 I   I
1 *  I   I   I   I   I   I   I   I   I   I   I   I   I   I   I
1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 *
1 *
1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 *  I   I   I   I   I   I   I   I   I   I   I   I   I   I   I
1 *  I 8 I   I 9 I   I 10 I   I 11 I   I 12 I   I
1 *  I   I   I   I   I   I   I   I   I   I   I   I   I   I   I
1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 *
1 *  (1) 01 OPEN,
1 *      03 REACH,
1 *      07 ALLOCATE,
1 *      08 PREPARE.
1 *
1 *  (2) INSTUCTION SEQUENCE NUMBER.
1 *  (3) DATA BASE NUMBER.
1 *  (4) OBJECT-NUMBER (BINARY).
1 *  (5) RELATION-NUMBER (BINARY) OR WORK-VAR-NUMBER
1 *      (BINARY)  -  (FOR REACH ONLY).
1 *  (6) LEVEL OF THE ENTRY CORRESPONDING TO THE "FROM"
1 *      PARAMETER (REACH ONLY).
1 *  (7) LEVEL OF THE ENTRY CORRESPONDING TO THE
1 *      "IN"/"GET" PARAMETER.
1 *  (8) ADDR. OF THE QUESTION-FIELD IN USER-STACK
1 *      (REACH ONLY).
1 *  (9) ADDR. OF THE RESPONSE-FIELD IN USER-STACK.
1 * (10) RECORD-NUMBER OF THE CORRESPONDING OCCURRENCE.
1 * (11) LENGTH OF QUESTION-FIELD (HALFWORD).
1 * (12) LENGTH OF RESPONSE-FIELD (HALFWORD).
1 *
1 *
1 *  REM. FOR AN OPEN INSTRUCTION THE DATA-BASE NAME
1 *       IS PUT IN FIELD 4,5,... (17 CHARACHERS).
1 *
1 *                    * * *
```

Figure 3.2.1.2.

```
DDR2  STMNT M SOURCE

        83 1 STKENTRY DSECT                  |STACK-ENTRY FORMAT
        84 1 CODE     DS     CL2             |ENTRY-CODE.
        85 1 SEQNBR   DS     CL4             |ENTRY-SEQNBR.
        86 1 DBN      DS     CL1             |ENTRY-DB-NUMBER.
        87 1          DS     CL1             |RESERVED.
        88 1 OBJN     DS     1H              |ENTRY-OBJ-NBR.
        89 1 RELN     DS     1H              |ENTRY-REL-NBR.
        90 1 FLEVEL   DS     1H              |FROM-LEVEL.
        91 1 XLEVEL   DS     1H              |IN/GET-LEVEL.
        92 1 QFLDA    DS     1F              |ADDR. OF QUES-FLD.
        93 1 RFLDA    DS     1F              |ADDR. OF RESP-FLD.
        94 1 RECNBR   DS     1F              |CORRESP. REC-NBR.
        95 1 QLENGTH  DS     1H              |QUES-FIELD-LENGTH.
        96 1 RLENGTH  DS     1H              |RESP-FIELD-LENGTH.
```

Figure 3.2.1.2. (suite)

3.2.2. LA ZONE DES VARIABLES DE TRAVAIL (VARAREA)

Cette zone est formée de 100 entrées de 2 mots. La 0ème entrée est inutilisée. La ième entrée ($01 \leqslant i \leqslant 99$) constitue l'implémentation de la variable de travail $\$<i>$. La fig. 3.2.2. fournit la description ASSEMBLEUR d'une entrée VARENTRY. Une telle entrée se divise en trois champs élémentaires :

- VDBN : 1 caractère valant 0 (binaire) si la variable de travail est vide, et égal au numéro de la base de données contenant la réalisation référencée, sinon.

- VOBJN : le numéro d'objet de la réalisation référencée par la variable ou 0 si la variable est vide (un demi-mot binaire).

- VRECNBR : le numéro d'enregistrement SESAM de la réalisation ou 0 si la variable est vide (1 mot binaire).

Remarque : Les champs VDBN et VRECNBR sont suffisants pour identifier une réalisation car il existe une relation fonctionnelle de (VDBN,VRECNBR) vers VOBJN

Cette dernière zone sert donc essentiellement à faire des vérifications de cohérence lors des manipulations de variables de travail.

DDR2 STMNT M SOURCE

```
                #VAR
*-----------------------------------------------------------------+
*   ENTRY IN TABLE OF WORKING VARIABLES (INTERFACE) I
*-----------------------------------------------------------------+
*
*
*  +--+--+--+--+--+--+--+--+
*  I  I//I     I           I
*  I1 I//I  2  I     3     I
*  I  I//I     I           I
*  +--+--+--+--+--+--+--+--+
*
*  (1) DATA BASE NUMBER.
*  (2) OBJECT NUMBER.
*  (3) RECORD NUMBER (DATA BASE KEY).
*
VARENTRY DSECT                          |VAR-DB-NUMBER.
VDBN     DS     CL1                     |RESERVED.
         DS     CL1                     |VAR-OBJ-NBR.
VOBJN    DS     CL2                     |VAR-REC-NBR.
VRECNBR  DS     1F
```

Figure 3.2.2.

3.2.3. LE BUFFER DE COMMUNICATION (CBUFFER)

Le buffer de communication est une zone fixe de 6K bytes dans laquelle les modules spécialisés construisent les requêtes à envoyer au noyau et reçoivent en retour les réponses de celui-ci.
Chaque module spécialisé impose une structure différente du buffer de communication, aussi bien à l'envoi qu'à la réception. La partie commune à toutes ces structures est représentée à la figure 3.2.3. sous forme ASSEMBLEUR.Elle occupe les 16 premiers bytes du buffer. On trouvera dans les listings spécialisés, la description complète du buffer de communication pour chaque ordre DML.

La zone standard contient 6 champs principaux dont la signification est donnée ci-dessous.

- BLG : longueur du buffer (1/2 mot binaire)
- BCODE : type de l'ordre DML (1/2 mot binaire)
- BLEVEL : niveau d'imbrication de l'ordre DML (")
- BSEQNBR : numéro séquentiel de l'ordre DML dans le programme utilisateur (4 caractères numériques)
- BDBN : numéro de la base de données sur laquelle opère l'ordre DML (1 caractère)
- BSTATUS : code de retour du buffer réponse.

```
DDR2  STMNT M SOURCE
                #BUF                                                   
      1 *---------------------------------------------------------------+
      1 *        COMMUNICATION BUFFER (INTERFACE).                      I
      1 *---------------------------------------------------------------+
      1 *
      1 *   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-
      1 *   I      I//////I     I     I           I  I//I     I////////
      1 *   I   1  I//////I  2  I  3  I     4     I5 I//I  6  I// 7 //
      1 *   I      I//////I     I     I           I  I//I     I////////
      1 *   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-
      1 *
      1 *   (1) BUFFER-LENGTH.
      1 *   (2) BUFFER-CODE (BINARY).
      1 *   (3) BUFFER-LEVEL (BINARY).
      1 *   (4) BUFFER-SEQNBR.
      1 *   (5) BUFFER-DATA-BASE-NUMBER.
      1 *   (6) COMPLETION STATUS.
      1 *   (7) SPECIFIC FIELDS.
      1 *
      1 BUFFER   DSECT                          |BUFFER FORMAT.
      1 BLG      DS      1H                     |BUFFER-LENGTH.
      1          DS      1H                     |RESERVED.
      1 BCODE    DS      1H                     |BUFFER-CODE.
      1 BLEVEL   DS      1H                     |BUFFER-LEVEL.
      1 BSEQNBR  DS      CL4                    |BUFFER-SEQNBR.
      1 BDBN     DS      CL1                    |BUFFER-DB-NUMBER.
      1 BFILLER  DS      CL1                    |NOT USED.
      1 BSTATUS  DS      CL2                    |RETURN STATUS.
        SPHINX   CSECT
```

Figure 3.2.3.

4. LES MODULES DE L'INTERFACE

4.1. SYSDML - module principal de l'interface

Ce module effectue les actions suivantes :

- Mémorisation, dans la première entrée du stack de l'interface (CONTROL), des adresses des zones commandes et données fournies par le programme DML lors de l'appel : INST et, lors d'un OPEN, SYS-STACK et SYS-ERAERA.
- Placement dans les zones SYS-STATUS et SYS-MACNUM du programme DML, du type et du numéro séquentiel de l'ordre DML.
- Vérification de validité du type et du niveau d'imbrication de la commande (ICODE, ILEVEL).
- Si la commande correspond à l'un des ordres @ALLOCATE, @PREPARE, @OPEN, @REACH, vérification plus poussée de la commande de manière à initialiser une nouvelle entrée STKENTRY du stack de l'interface, augmenter STKTOP d'une unité.
- Passer la main au module spécialisé, après avoir placé dans le registre 1 l'adresse du stack de l'interface.
- Rendre la main au programme utilisateur, au retour du module spécialisé.

4.2. SYS01A,..., SYS20A - MODULES SPECIALISES

On peut distinguer 2 classes parmi ces modules. Ceux qui réalisent eux-mêmes la fonction correspondant à l'ordre DML et ceux qui se contentent de transmettre au noyau SPHINX les informations nécessaires. Seuls, les seconds utilisent la communication entre tâches.

La première classe ne comporte, en fait, que quatre modules (SYS15A,..., SYS18A) correspondant aux ordres DML, @SAVE, @MOVE, @TEST, @CLEAR dont l'importance, dans le DML, est relativement limitée.
L'implémentation de ces ordres nécessite seulement la connaissance des <u>références</u> aux réalisations d'objets complexes c'est-à-dire des couples (n° de base de données, n° d'enregistrement SESAM).

Ces renseignements sont consignés dans le stack de l'interface pour les réalisations courantes (DBN, RECNBR) et dans la zone VARAREA pour les variables de travail (VARENTRY).

Les modules de la seconde classe effectuent en général les actions suivantes :

- vérification du contenu de la commande INST.
- Constitution du buffer de communication à envoyer au noyau SPHINX.

Celui-ci contiendra l'essentiel des informations contenues dans INST et, aussi, diverses données recopiées de la zone de communication de l'utilisateur (valeurs d'objets élémentaires, compteurs d'occurrences, critères d'accès) ou même de la zone des variables de travail (numéros d'enregistrement SESAM).

- Passer la main au module SYSCOM de communication avec le noyau SPHINX.
- Analyser le buffer de communication renvoyé par le noyau et garnir en conséquence les zones de communication du programme DML (SYS-STACK, SYS-ERAREA), c'est-à-dire effectuer certaines actions parmi les suivantes :
  - recopier des valeurs d'objets élémentaires dans SYS-STACK;
  - positionner le code de retour SYS-ERROR, dans SYS-STACK;
  - garnir le vecteur diagnostic SYS-ERAREA;
  - recopier des valeurs de compteurs d'occurrences de relations entre objets complexes, dans SYS-STACK.

Lors de l'accès ou de la création d'une nouvelle réalisation, le buffer de retour contient le numéro d'enregistrement SESAM de la réalisation; le le placer dans le mot RECNBR du dernier niveau du stack de l'interface.

On trouvera dans les listings des modules spécialisés un exposé complet de ces opérations pour chaque cas précis.

4.3. SYSCOM - module de gestion de la communication avec le noyau SPHINX

Le rôle de ce module est de gérer la communication entre le programme d'application DML et le noyau SPHINX.
Il effectue les fonctions suivantes :

- ouverture de la communication
- envoi du buffer de communication au noyau.
- attente et réception du buffer-réponse en provenance du noyau.
  Si aucune réponse n'est reçue après un certain temps limite (par exemple, 10 minutes), la main est rendue au module spécialisée, après avoir placé le code d'erreur '83' dans la zone BSTATUS de CBUFFER.

La clôture de la communication est prise en charge par le système BS2000 lors de la terminaison (normale ou anormale) du programme d'application.

LE NOYAU EXECUTIF SPHINX

1. Introduction

Le noyau SPHINX est un programme qui s'exécute dans une tâche distincte de celle des programmes d'application (DML - NUL). Il reçoit de ceux-ci des buffers de communication correspondant aux ordres DML contenus dans ces programmes. Ces buffers sont, alors, analysés par le noyau qui doit entreprendre les actions nécessaires à la réalisation de ces ordres. Ces actions seront notamment des appels au noyau SESAM (*), mais aussi la mémorisation de renseignements relatifs au déroulement de chaque programme utilisateur : nombre de niveaux d'imbrication, base de données ouvertes, objets complexes en cours de traitement, type des manipulations effectuées (accès, modification)... L'existence de ces renseignements permet au noyau de déterminer l'ordre selon lequel les accès aux diverses bases de données seront réalisés, via SESAM, en fonction de la concurrence des différets programmes d'application, en s'efforçant de maintenir la cohérence de chaque base de données et de servir chaque programme le plus vite possible et sans discrimination.

Le module du noyau qui prend ces décisions est appelé le superviseur. C'est le module central du noyau. Il est décrit au paragraphe 3.3. L'action du superviseur s'appuie sur un ensemble dynamique de tables qui font l'objet du paragraphe 2 et de modules de gestion de ces tables décrits au paragraphe 3.2. Les autres modules du noyau réalisent les fonctions particulières de chaque ordre DML, ils sont appelés par le superviseur à des moments déterminés par lui en fonction de la concurrence et de la disponibilité de SESAM. Ils rendent la main au superviseur pour réaliser les accès aux bases de données. Pour une description détaillée de ces modules, nous renvoyons le lecteur à la documentation contenue dans les listings.

(*) Dans la suite de ce chapitre, on désignera le noyau SPHINX par le terme "noyau" et par le terme "SESAM", le noyau SESAM.

2. Les tables du noyau

2.1. SCB - Bloc de contrôle du système

La figure 2.1. donne la découpe du bloc de contrôle du système (SCB). Ce bloc de mémoire fixe, situé physiquement au début du noyau, contient les principaux renseignements relatifs à l'état du superviseur et permet d'accéder à toutes les zones de mémoire dynamiques allouées pour la gestion des différentes tâches utilisateur connues du noyau et pour la description des bases de données auxquelles ces tâches accèdent.

Accessible à tout module du noyau, le SCB sert de point de départ à toute recherche de renseignements concernant les tâches utilisateur ou les bases de données.

Précisons la signification des différents champs du SCB.

FTCBA,LTCBA pointent, respectivement vers le premier et le dernier bloc de contrôle de tâche (TCB) ou sur le SCB lui-même si aucun TCB n'existe.

CTCBA vaut zéro ou pointe vers le bloc de contrôle de la tâche pour laquelle le noyau est en train de travailler. Ce TCB est appelé TCB courant. Il est évident, en effet, que le noyau, comme n'importe quel programme gérant plusieurs processus concurrents, partage son temps entre les différentes tâches, mais ne peut en servir qu'une à la fois.

NTCB a pour valeur le nombre de TCB contenus dans le système, c'est-à-dire le nombre de tâches d'application connues du noyau (1/2 mot binaire).

FFAREA,LFAREA sont des pointeurs vers le premier et le dernier bloc libre de la "mémoire dynamique", manipulée par le module SYSMEM (voir § 3.2.1.). Ils sont encadrés de zones constantes dont la présence est exigée par l'algorithme de gestion dynamique de la mémoire utilisé dans SYSMEM.

FSAREA,LSAREA pointent vers le premier et le dernier bloc de mémoire dynamique réservé par le superviseur. Ces blocs seront des TCB et des blocs associés aux bases de données (voir § 2.3.).

SCW est le "mot de contrôle du système" (System Control Word).
Les 3 premiers bits du SCW ont une signification particulière.
Nous les désignerons par SCW(i) (i=1,2,3). On a :

SCW(1) = 1 si un accès à une base de données a été commandé à SESAM par le noyau (appel de SESPUT) et n'a pas encore reçu de réponse (réception d'un buffer de communication par SESGETW);

= 0 sinon.

SCW(2) = 1 si une modification des pointeurs de la chaîne des TCB est en cours;
= 0 sinon.
La raison d'être de cette information sera établie au § 3.3.3. consacré au module SYSINT d'interrogation de l'état du noyau.

SCW(3) = 1 si le système est dans sa phase de terminaison (SHUTDOWN), il n'accepte plus de traiter de nouvelles tâches, mais achève normalement l'exécution des tâches en cours.
= 0 le système n'est pas en phase de terminaison.

SBOX Les deux derniers bytes du SCW (SBOX) sont utilisés comme code de retour par tous les modules de gestion des tables du noyau ne travaillant pas pour une tâche spécifique.

SINDA,SRCPTA,SRESPA,SQUESA constituent la zone paramètre passée à SESPUT lorsque le superviseur veut réaliser un accès à une base de données et que cet accès n'a pas pour but de servir une tâche particulière. Ces quatre zones contiennent, respectivement, l'adresse de l'indication-field, du receipt-field, du response-field et du question-field SESAM (voir le fascicule "Une implémentation du Modèle d'Accès").

STIMER1 contient l'heure à laquelle à été chargé le noyau, en six caractères numériques, sous la forme Heure/Minutes/Secondes.

La SAVE AREA de 18 mots qui suit est utilisée pour sauver les registres lorsque le module superviseur fait appel à un module de gestion des tables.

$DBHA_i$ Les 32 mots suivants, numérotés A,B,C,...,Z,1,2,3,4,5,6, contiennent soit 0, soit l'adresse d'un bloc d'en-tête de base de données (voir § 2.3.).

```
DDR2  STMNT M SOURCE

171                 #SCB
172 1 SCB           DSECT
173 1               #SCT
174 2 *-----------------------------------------------------------+
175 2 *                 SYSTEM CONTROL BLOCK.                     I
176 2 *-----------------------------------------------------------+
177 2 *
178 2 *
179 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
180 2 *  I              I              I              I      I/////I
181 2 *  I   FTCBA      I   LTCBA      I   CTCBA      I INTCB I/////I
182 2 *  I              I              I              I      I/////I
183 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
184 2 *  I              I              I              I              I
185 2 *  I01 00 00 02I  I  FFAREA      I  LFAREA      I01 00 00 02I
186 2 *  I              I              I              I              I
187 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
188 2 *  I              I              I              I              I
189 2 *  I01 00 00 00I  I  FSRAREA     I  LSRAREA     I  S  C  W     I
190 2 *  I              I              I              I              I
191 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
192 2 *  I              I              I              I              I
193 2 *  I   SINDA      I  SRCPTA      I  SRESPA      I  SQUESA      I
194 2 *  I              I              I              I              I
195 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
196 2 *  I//////////I                 I//////////////////////////I
197 2 *  I//////////I  S T I M E R 1   I//////////////////////////I
198 2 *  I//////////I                 I//////////////////////////I
199 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
200 2 *  I                                                        I
201 2 *  I                                                        I
202 2 *        S A V E   A R E A   (18 FULLWORDS)                 I
203 2 *  I                                                        I
204 2 *  I                                                        I
205 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
206 2 *  I              I                          I              I
207 2 *  I   DBHA1      I          . . .           I   DBHA32     I
208 2 *  I              I                          I              I
209 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.1.

```
DDR2  STMNT M SOURCE

      210 2 * FTCBA    ADDRESS OF FIRST TCB (OR ADDR. OF SCB).
      211 2 * LTCBA    ADDRESS OF LAST  TCB (OR ADDR. OF SCB).
      212 2 * CTCBA    ADDRESS OF CURRENT TCB (OR X'00000000').
      213 2 * NTCB     NUMBER OF TCB IN TCB CHAIN.
      214 2 * FFAREA   ADDRESS OF FIRST FREE AREA.
      215 2 * LFAREA   ADDRESS OF LAST  FREE AREA.
      216 2 * HSRAREA  HEADER OF RESERVED AREA CHAIN.
      217 2 * FSRAREA  ADDRESS OF FIRST SYSTEM RESERVED AREA;
      218 2 *          IF NO AREA : ADDRESS OF HSRAREA IN SCB.
      219 2 * LSRAREA  ADDRESS OF LAST  SYSTEM RESERVED AREA;
      220 2 *          IF NO AREA : ADDRESS OF HSRAREA IN SCB.
      221 2 * SCW      SYSTEM CONTROL WORD. THE FIRST 2 BYTES (SSTATUS)
      222 2 *          ARE  RESERVED FOR SYSTEM MANAGEMENT; THE LAST 2
      223 2 *          BYTES (SBOX) ARE USED AS RETURN CODE BY UTILITY
      224 2 *          ROUTINES.
      225 2 * SINDA   ADDR. OF INDICATION-FIELD FOR SESAM.
      226 2 * SRCPTA  ADDR. OF RECEIPT-FIELD   FOR SESAM.
      227 2 * SRESPA  ADDR. OF RESPONSE-FIELD  FOR SESAM.
      228 2 * SQUESA  ADDR. OF QUESTION-FIELD  FOR SESAM.
      229 2 * STIMER1 TIME AT WHICH SYSTEM HAS BEEN LOADED.
      230 2 * DBHA(I) ADDRESS OF DATA BASE HEADER (I).


      231 2 FTCBA     DS     F
      232 2 LTCBA     DS     F
      233 2 CTCBA     DS     F
      234 2 NTCB      DS     H
      235 2           DS     H
      236 2 MCA       DS     F
      237 2 FFAREA    DS     F
      238 2 LFAREA    DS     F
      239 2           DS     F
      240 2 HSRAREA   DS     F
      241 2 FSRAREA   DS     F
      242 2 LSRAREA   DS     F
      243 2 SCW       DS     OF
      244 2 SSTATUS   DS     CL2
      245 2 SBOX      DS     CL2
      246 2 SINDA     DS     F
      247 2 SRCPTA    DS     F
      248 2 SRESPA    DS     F
      249 2 SQUESA    DS     F
      250 2           DS     F
      251 2 STIMER1   DS     CL6
      252 2           DS     CL6
      253 2 SYSSAVE   DS     18F
      254 2 DBHA      DS     F
      255 2           DS     31F
      256 1 SPHINX    CSECT
```

Figure 2.1. (suite)

2.2. Blocs associés à une tâche

2.2.1. TCB - Bloc de contrôle de tâche

Lorsqu'une tâche d'application (programme DML ou interpréteur NUL) envoie un premier buffer de communication au noyau (open de niveau 1), celui-ci alloue, pour cette nouvelle tâche, un bloc de mémoire qui contiendra les principaux renseignements sur l'évolution et l'état de cette tâche et servira de point de départ pour la recherche de toutes les autres informations la concernant.

La figure 2.2.1. représente la découpe de ce bloc de contrôle de tâche ou TCB (Task Control Block). Nous précisons, ci-dessous, la signification des différentes zones qui les composent.

NTCBA, PTCBA pointent vers les TCB suivants et précédents (ou vers le SCB si le suivant ou le précédent n'existent pas), dans la chaîne des TCB. Cette chaîne double, issue du SCB, permet d'atteindre tous les TCB et est trié par ordre chronologique d'arrivée du dernier buffer. Cela veut dire, par exemple, que le premier TCB de la chaîne est celui dont le dernier buffer est arrivé il y a le plus longtemps.

TSN contient quatre caractères numériques formant le numéro de la tâche. Ce numéro est extrait du premier buffer de communication (champ SNAME : voir § 2.2.4).

BUFA Lorsqu'il reçoit un buffer de communication d'un programme d'application, le superviseur place son adresse dans BUFA, avant de passer la main au module spécialisé qui en assurera le traitement. Celui-ci y placera, ensuite, l'adresse du buffer "réponse", puis rendra la main au superviseur qui l'enverra au programme DML.

STKA pointe vers un autre bloc de mémoire : le stack des réalisations courantes de la tâche (voir § 2.2.2.).

SLEVEL contient, en binaire, le nombre de niveaux actifs dans la tâche, c'est-à-dire le nombre de niveaux du stack des réalisations.

REQSTA vaut zéro si la tâche n'est pas suspendue en raison de la concurrence, sinon REQSTA pointe vers le bloc des demandes d'accès (voir § 2.2.5).

FRAREA,LRAREA Les TCB participent à la chaîne des blocs réservés par le superviseur (voir 2.1. : FSAREA,LSAREA). Par contre, les blocs de mémoire dynamique que nous allons décrire dans la suite de ce paragraphe 2.2 sont associés à la tâche seule. Ils seront inclus, lors de leur allocation, à la chaîne des blocs réservés pour la tâche.
Si une erreur se produit lors du déroulement de la tâche, ces zones

dynamiques ne pourront être libérées normalement.
La chaîne des blocs réservés par la tâche fournit le moyen de les libérer globalement (Garbage Collection). FRAREA et LRAREA pointent, respectivement, vers le premier et le dernier de ces blocs.

TCW Le mot de contrôle de tâche (TCW) contient divers renseignements sur l'état de la tâche.
Les huit premiers bits de ce mot ont une fonction particulière.
Désignons-les par TCW(i) (i = 1,2,...,8). On a, alors :

TCW(1) = 1 si le TCB est actif. C'est-à-dire si un buffer de communication a été reçu de la tâche utilisateur et qu'un buffer réponse n'a pas encore été renvoyé par le noyau.

= 0 si le TCB est inactif. Le noyau est, alors, en attente d'un buffer originaire de cette tâche.

TCW(2) = 1 si le TCB est "prêt à faire appel à SESPUT".
Lorsque le traitement du TCB nécessite l'intervention du noyau SESAM et que ce dernier est actif (SCW(1) = 1), le bit TCW(2) est mis à 1 et le traitement du TCB est interrompu jusqu'à ce que SESAM soit disponible (au retour d'un appel de SESGETW).

= 0 dans tous les autres cas.

TCW(3) = 1 si le TCB est "prêt à faire appel à SESGETW"
Lorsque le traitement du TCB donne lieu à une communication avec SESAM, via SESPUT, le bit TCW(2) est éventuellement repositionné à 0, puis le bit TCW(3) est positionné à 1 et le traitement du TCB est interrompu.
Il sera repris, par un appel à SESGETW, soit lorsque le superviseur extraira, de la file d'attente du noyau, un buffer envoyé par SESAM, soit lorsqu'un autre TCB sera "prêt à faire appel à SESPUT".

= 0 dans tous les autres cas.

TCW(4) = 1 si le TCB est "en cours de traitement" (ACTIVE-RUNNING).
Cela veut dire qu'un buffer a été reçu de la tâche correspondante, qu'un module spécialisé est en train de le traiter et qu'aucun accès, via SESAM, n'est actuellement en cours pour cette tâche.

= 0 dans tous les autres cas.

TCW(5) = 1 dans le cas, très particulier, où la tâche effectue un ordre @OPEN sur une base de données dont le schéma interne est en cours de chargement. Ce schéma, décrit aux paragraphes 2.3.2. et 2.3.3. doit, en effet, être lu, de la base de données elle-même, lors d'un premier OPEN, effectué par une première tâche.

Tant que ce chargement n'est pas achevé, la main ne peut être rendue à aucune tâche travaillant sur cette base. Tous les TCB correspondant à ces tâches seront donc suspendus après positionnement à 1 de TCW(5).

= 0 Dès que le chargement des schémas internes est terminé, TCW(5) est positionné à 0 pour toutes les tâches en train de faire un OPEN sur cette base.

TCW(6) = 1 si le TCB est suspendu parce qu'il désire faire des accès incompatibles avec ceux déjà autorisés à d'autres TCB (concurrence). A ce moment, la tâche correspondance ne peut avoir exécuté que des ordres OPEN, sur une ou plusieurs bases de données (voir § 2.2.5., 2.3.4., 3.3.2.).

= 0 dans les autres cas.

TCW(7) = 1 lorsque la tâche est en attente d'être éliminée du système, soit parce qu'une erreur a été détectée dans son déroulement, soit parce que l'administrateur du système a introduit la commande "CAN <TSN>" au pupitre de contrôle (voir § 3.3.3.). Le superviseur déclenchera la suppression du TCB et la récupération de tous les blocs réservés par lui, lorsqu'aucune action plus importante ne sera en attente.

= 0 dans les autres cas.

TCW(8) = 1 <u>si le processus de suppression du TCB est en cours</u> (voir § 3.3.2.1 : organigramme de SYSPURGE). Ce processus contient, en effet, un appel au module FREEUI qui effectue la libération des "numéros d'utilisateur" SESAM alloués à la tâche (voir § 3.2.3. : CANTCB). Cette libération se traduit par des communications avec SESAM qui ne peuvent s'effectuer qu'une fois. Le positionnement de TCW(8) à 1 assure qu'on ne lancera pas plusieurs fois le processus de suppression du TCB.

= 0 tant que le processus de suppression du TCB n'est pas initialisé.

TBOX Les deux derniers bytes du TCW(TBOX) sont utilisés comme code de retour par les différents modules travaillant spécifiquement pour cette tâche.

INDA, RESTA, RESPA, QUESA sont garnis avant de rendre la main au superviseur pour faire appel à SESPUT. Ils constituent les arguments d'appel de ce module.

TIMER1 contient, en 6 caractères numériques (heure/minutes/secondes), l'heure de création du TCB.

TIMER2 contient, sous la même forme, l'heure de prise en charge, par le noyau, du dernier buffer issu de la tâche utilisateur. La chaîne des TCB est triée par valeurs croissantes de TIMER2.

Conventions

Dans la suite, nous utiliserons les conventions suivantes :
si α est un des noms de champ définis plus haut et si β désigne l'adresse d'un TCB,

$\alpha.\beta$ désignera le champ α du TCB d'adresse β

α désignera le champ α du TCB courant.

Exemples

TCW(2) . TCBAD désigne le bit TCW(2) du TCB dont l'adresse est contenue dans la zone TCBAD

TIMER1 (ou TIMER1 . CTCBA) désigne l'heure de création du TCB courant.

```
DDR2  STMNT M SOURCE
    257              #TCB
    258 1 *-----------------------------------------------------------------+
    259 1 *                   TASK CONTROL BLOCK.                          I
    260 1 *-----------------------------------------------------------------+
    261 1 *
    262 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    263 1 *   I              I              I              I              I
    264 1 *   I    NTCBA     I    PTCBA     I     TSN      I     BUFA     I
    265 1 *   I              I              I              I              I
    266 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    267 1 *   I              I              I              I//////////////I
    268 1 *   I    STKA      I    SLEVEL    I    REQSTA    I//////////////I
    269 1 *   I              I              I              I//////////////I
    270 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    271 1 *   I              I              I              I              I
    272 1 *   I01 00 00 00   I    FRAREA    I    LRAREA    I    T C W     I
    273 1 *   I              I              I              I              I
    274 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    275 1 *   I              I              I              I              I
    276 1 *   I    INDA      I    RCPTA     I    RESPA     I    QUESA     I
    277 1 *   I              I              I              I              I
    278 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    279 1 *   I//////////////I                             I              I
    280 1 *   I//////////////I      T I M E R 1            I  T I M E R 2 I
    281 1 *   I//////////////I                             I              I
    282 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    283 1 *   I                                                              I
    284 1 *   I                                                              I
    285 1 *   I          S A V E   A R E A     (18 FULLWORDS)                I
    286 1 *
    287 1 *   I                                                              I
    288 1 *   I                                                              I
    289 1 *   +--------------------------------------------------------------+
```

Figure 2.2.1.

```
DDR2  STMNT M SOURCE

290 1 * NTCBA    ADDRESS OF NEXT  TCB.
291 1 * PTCBA    ADDRESS OF PRIOR TCB.
292 1 * TSN      TASK SEQUENCE NUMBER.
293 1 * BUFA     ADDRESS OF COMMUNICATION BUFFER.
294 1 * STKA     ADDRESS OF STACK (ENTRY 0 IS RESERVED).
295 1 * SLEVEL   NUMBER OF ENTRIES IN STACK.
296 1 * REQSTA   ADDRESS OF REQUEST AREA.
297 1 * HRAREA   HEADER OF RESERVED AREA CHAIN.
298 1 * FRAREA   ADDRESS OF FIRST RESERVED AREA;
299 1 *          IF NO AREA : ADDRESS OF HRAREA IN TCB.
300 1 * LRAREA   ADDRESS OF LAST  RESERVED AREA;
301 1 *          IF NO AREA : ADDRESS OF HRAREA IN TCB.
302 1 * TCW      THE  FIRST 2 BYTES  (STATUS)  CONTAIN THE STATUS OF
303 1 *          THE TCB; THE LAST 2 BYTES (TBOX) ARE USED AS RETURN
304 1 *          CODE BY UTILITY ROUTINES.
305 1 * INDA     ADDR. OF INDICATION-FIELD  FOR SESAM.
306 1 * RCPTA    ADDR. OF RECEIPT-FIELD     FOR SESAM.
307 1 * RESPA    ADDR. OF RESPONSE-FIELD    FOR SESAM.
308 1 * QUESA    ADDR. OF QUESTION-FIELD    FOR SESAM.
309 1 * TIMER1   FIRST ACTIVATION TIME.
310 1 * TIMER2   LAST  ACTIVATION TIME.


311 1 TCB        DSECT
312 1 NTCBA      DS     F
313 1 PTCBA      DS     F
314 1 TSN        DS     CL4
315 1 BUFA       DS     F
316 1 STKA       DS     F
317 1 SLEVEL     DS     F
318 1 REQSTA     DS     F
319 1            DS     F
320 1 HRAREA     DS     F
321 1 FRAREA     DS     F
322 1 LRAREA     DS     F
323 1 TCW        DS     OF
324 1 STATUS     DS     CL2
325 1 TBOX       DS     CL2
326 1 INDA       DS     F
327 1 RCPTA      DS     F
328 1 RESPA      DS     F
329 1 QUESA      DS     F
330 1            DS     F
331 1 TIMER1     DS     CL6
332 1 TIMER2     DS     CL6
333 1 TCBSAVE    DS     18F
334 1 SPHINX     CSECT
```

Figure 2.2.1. (suite)

2.2.2. Le stack des réalisations courantes de la tâche

Lorsqu'un TCB est créé, un autre bloc, de 202 mots de mémoire, est réservé et son adresse est placée dans le champ STKA du TCB.
Ce bloc, appelé stack des réalisations, se décompose en 101 entrées numérotées 0,1,...,100. La ième entrée est associée au ième niveau de la structure de blocs du programme DML. La 0ème entrée n'est pas utilisée.

Le nombre de niveaux actifs est contenu dans SLEVEL et chaque entrée (STACK) se compose de deux champs COBJA et WORKA (voir figure 2.2.2.). Pour les niveaux actifs, la signification de ces champs est donnée ci-dessous.

COBJA - vaut zéro, si l'ordre correspondant à ce niveau, dans le programme utilisateur est @OPEN.
- pointe vers un bloc de mémoire (XENTRY) décrivant la réalisation courante de ce niveau, si l'ordre correspondant, dans le programme DML, est @ALLOCATE, @CREATE, @PREPARE ou @REACH (cfr. § 2.2.3.).

WORKA - pointe vers le bloc d'en-tête de base de données (DBH), si l'ordre correspondant est @OPEN.
- pointe, dans d'autres cas, vers des zones de mémoire dynamique réservées par les modules spécialisés.

DDR2 STMNT M SOURCE

```
373                 #STACK
374 1 *-----------------------------------------------------------+
                                                                  I
375 1 *                STACK ENTRY (NUCLEUS).                     +
376 1 *-----------------------------------------------------------+
377 1 *
378 1 *
379 1 *              +--+--+--+--+--+--+--+--+
380 1 *              I           I           I
381 1 *              I   COBJA   I   WORKA   I
382 1 *              I           I           I
383 1 *              +--+--+--+--+--+--+--+--+
384 1 *
385 1 *
386 1 * COBJA    ADDRESS OF CURRENT OBJECT IN AREA OF OCCURRENCES.
387 1 * WORKA    ADDRESS OF WORKING AREA.
388 1 *
389 1 *
390 1 STACK     DSECT
391 1 COBJA     DS     F
392 1 WORKA     DS     F
393 1 SPHINX    CSECT
```

Figure 2.2.2.

2.2.3. XENTRY - Bloc descripteur d'une réalisation d'objet complexe

A chaque niveau actif d'un programme DML peut correspondre zéro ou une réalisation d'objet complexe. Les caractéristiques de cette réalisation, nécessaires au noyau, sont rassemblées dans un bloc de mémoire, nommé XENTRY, accessible à partir du stack des réalisations (COBJA).

La figure 2.2.3. présente la découpe d'un bloc XENTRY. Donnons la signification des différents champs qui le composent.

XUI est le numéro d'utilisateur SESAM associé par le noyau à la réalisation. Le premier caractère de XUI est le numéro de la base de données (A,B,..., Z,1,2,...,6), le second est affecté par le noyau.

XOBJN est un demi mot binaire contenant le numéro de l'objet complexe fixé par le DDL. Ce numéro sert d'indice pour la table des objets complexes définie en 2.3.2.

XRECNBR contient le numéro d'enregistrement SESAM de la réalisation ou zéro si celle-ci est "vide" (@ALLOCATE).

XNPTR contient le nombre N_p de pointeurs associés à la réalisation.

XNCNT contient le nombre N_c de compteurs associés à la réalisation.

XPTRI, XCNTI les zones définies ci-dessus sont suivies de N_p mots contenant les pointeurs et de N_c mots contenant les compteurs associés à la réalisation. Si la réalisation est vide, le contenu de ces N_p+N_c mots n'est pas significatif.

```
DDR2  STMNT M SOURCE
                    #XENTRY
394                                                                  I
395 1 *-------------------------------------------------------------+
396 1 *          ENTRY IN AREA OF OCCURRENCES.
397 1 *-------------------------------------------------------------+
398 1 *
399 1 *                                                              I
400 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
401 1 *  I     I     I                 I     I     I                 I
402 1 *  I XUI IXOBJNI  XRECNBR  IXNPTRIXNCNTI   XPTR1               I
403 1 *  I     I     I                 I     I     I                 I
404 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
405 1 *                 --+---+---+---+---+---+---+---+---+--
406 1 *                   I           I           I
407 1 *                   I   XPTRN   I   XCNT1   I   . . .
408 1 *      . . .        I           I           I
409 1 *                 --+---+---+---+---+---+---+---+---+--
410 1 *
411 1 *
412 1 *
413 1 * XUI        USER-ID (DATA BASE NUMBER + SUFFIX).
414 1 * XOBJN      OBJECT NUMBER.
415 1 * XRECNBR    RECORD NUMBER (DATA BASE KEY).
416 1 * XNPTR      NUMBER OF POINTERS.
417 1 * XNCNT      NUMBER OF COUNTERS.
418 1 * XPTRI      POINTER.
419 1 * XCNTI      COUNTER.
420 1 *
421 1 *
422 1 XENTRY    DSECT
423 1 XUI       DS     CL2
424 1 XOBJN     DS     H
425 1 XRECNBR   DS     F
426 1 XNPTR     DS     H
427 1 XNCNT     DS     H
428 1 XPTRI     DS     F
429 1 XCNTI     DS     F
430 1 SPHINX    CSECT
```

Figure 2.2.3.

2.2.4. BUFFER - Buffer de communication

La figure 2.2.4. décrit la partie fixe du buffer de communication associé au TCB. Ce buffer n'existe que lorsque la tâche est active, son adresse est alors placée dans le mot BUFA du TCB.
On trouvera la description de la partie variable dans les listings des modules spécialisés.

La description ci-dessous est valable aussi bien pour les buffers reçus par le noyau que pour ceux qu'il renvoie aux différentes tâches utilisateur.

SNAME contient le nom de la tâche pour le système de communication entre tâches : TSN <numéro de tâche> (voir chapitre : La communication entre tâches dans le système SPHINX).

BLNGTH est un demi mot binaire, contenant la longueur totale du buffer, en bytes, SNAME non compris.

BCODE est un demi mot binaire contenant le code de l'ordre DML. (On trouvera une liste de ces codes dans la brochure "Manuels de Référence" page DML6).

BLEVEL est un demi mot binaire contenant le niveau d'imbrication de la macro. Il doit être égal à SLEVEL.

BSEQNBR contient le numéro séquentiel de l'ordre DML, dans le programme utilisateur (4 caractères numériques).

BDBN contient le numéro de la base de données à laquelle l'ordre DML est relatif.

BSTATUS contient le code de retour du buffer réponse (2 caractères numériques).

```
DDR2  STMNT M SOURCE
                    #CBUF
335                                                          +
336 1 *-----------------------------------------------------I
337 1 *           COMMUNICATION BUFFER (NUCLEUS).            +
338 1 *-----------------------------------------------------
339 1 *
340 1 *
341 1 *  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
342 1 *  I                          I     I/////I     I     I
343 1 *  I     S  N  A  M  E        I  1  I/////I  2  I  3  I
344 1 *  I                          I     I/////I     I     I
345 1 *  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
346 1 *  I                 I  I//I  I
347 1 *  I     4           I5 I//I  6  I . . .  7  . . .
348 1 *  I                 I  I//I  I
349 1 *  +--+--+--+--+--+--+--+--+--+--+--
350 1 *
351 1 *
352 1 *        SNAME      SENDER NAME  (TASK SEQUENCE NUMBER).
353 1 * (1)   BLNGTH      BUFFER LENGTH.
354 1 * (2)   BCODE       BUFFER CODE (BINARY).
355 1 * (3)   BLEVEL      BUFFER LEVEL (BINARY).
356 1 * (4)   BSEQNBR     BUFFER SEQUENCE NUMBER (SEQUENCE NUMBER
357 1 *                   OF CORRESPONDING DML CALL).
358 1 * (5)   BDBN        DATA BASE NUMBER.
359 1 * (6)   BSTATUS     RETURN CODE.
360 1 * (7)               SPECIFIC FIELDS.
361 1 *
362 1 BUFFER    DSECT
363 1 SNAME     DS     CL8
364 1 BLNGTH    DS     H
365 1           DS     H
366 1 BCODE     DS     H
367 1 BLEVEL    DS     H
368 1 BSEQNBR   DS     CL4
369 1 BDBN      DS     CL1
370 1           DS     CL1
371 1 BSTATUS   DS     CL2
372   SPHINX    CSECT
```

Figure 2.2.4.

2.2.5. Bloc des demandes d'accès (REQST)

Ce bloc, dont l'adresse est placée dans le champ REQSTA du TCB, contient la liste des autorisations d'accès demandées par la tâche, qu'elles lui aient été délivrées, ou non. (Dans ce dernier cas, la tâche est suspendue).
Si REQSTA vaut 0, cela signifie que la tâche ne possède aucune autorisation d'accès (et n'en a pas encore demandé). Les seuls blocs actifs, dans le programme utilisateur, correspondent donc à des ordres @OPEN.

La structure du bloc des demandes d'accès est représentée à la figure 2.2.5.

NBD est un demi mot binaire contenant le nombre l de bases de données auxquelles la tâche peut ou veut accéder.

BDN_i ($i = 1, \ldots, l$) contient le numéro de la base de données appelée, ci-dessous, la ième base de données ($BDN_i \in \{A, \ldots, Z, 1, \ldots, 6\}$)
On a : $(1 \leqslant i_1 < i_2 \leqslant l) \Rightarrow (BDN_{i_1} < BDN_{i_2}$ (ordre EBCDIC)).

$NACC_i$ ($i = 1, \ldots, l$) est un demi mot binaire contenant le nombre n_i d'objets complexes susceptibles d'être accédés, en lecture, pour la ième base de données.

$NMAJ_i$ ($i = 1, \ldots, l$) est un demi mot binaire contenant la nombre m_i d'objets complexes susceptibles d'être accédés et mis à jour, pour la ième base de données.

$NOBJ_i^j$ ($i = 1, \ldots, l$; $j = 1, \ldots, n_i$) est un demi mot binaire contenant le numéro du jème objet complexe, susceptible d'être accédé, en lecture, pour la ième base de données.

$NOBJ_i^j$ ($i = 1, \ldots, l$; $n_i < j \leqslant n_i + m_i$) est un demi mot binaire contenant le numéro du $(j-n_i)$ ème objet complexe susceptible d'être accédé, en mise à jour, pour la ième base de données.

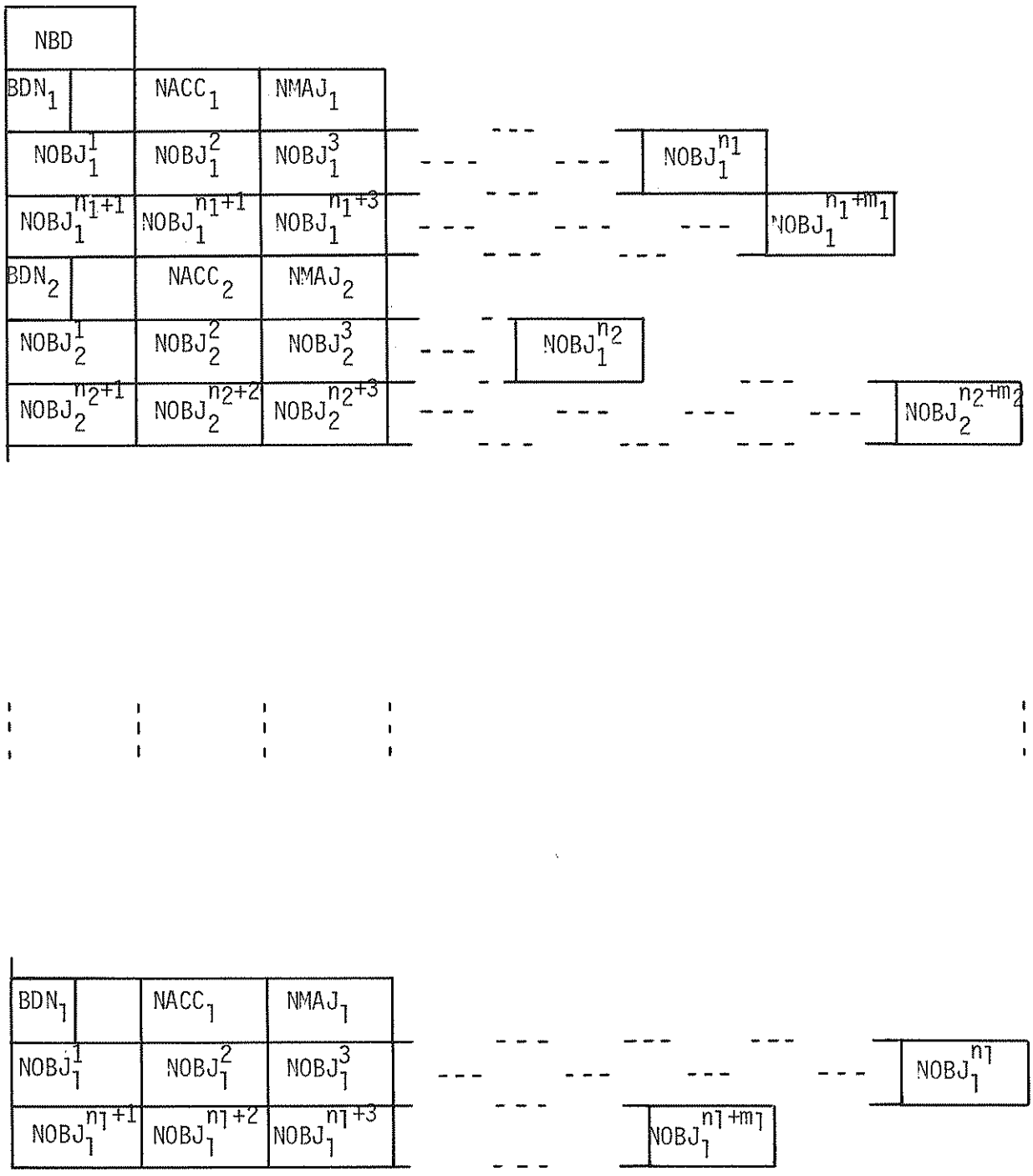

NBD
BDN1
NACC1
NMAJ1
NOBJ1 1
NOBJ1 2
NOBJ1 3
NOBJ1 n1
NOBJ1 n1+1
NOBJ1 n1+1
NOBJ1 n1+3
NOBJ1 n1+m1
BDN2
NACC2
NMAJ2
NOBJ2 1
NOBJ2 2
NOBJ2 3
NOBJ1 n2
NOBJ2 n2+1
NOBJ2 n2+2
NOBJ2 n2+3
NOBJ2 n2+m2
BDNl
NACCl
NMAJl
NOBJl 1
NOBJl 2
NOBJl 3
NOBJl nl
NOBJl nl+1
NOBJl nl+2
NOBJl nl+3
NOBJl nl+ml

Figure 2.2.5.

2.3. Blocs associés à une base de données

2.3.1. Bloc d'en-tête d'une base de données

Lors de la compilation DDL, toute base de données se voit affecter un numéro de base de données qui est soit une lettre, soit un chiffre de 1 à 6. Aux 32 numéros de base de données possibles correspondent les 32 mots $DBHA_i$ du SCB (i = A,B,...,Z,1,...,6). Cela implique que toutes les bases de données actives lors d'une exécution du noyau doivent avoir des numéros de base de données distincts.

Lors du premier OPEN d'une base de données, quatre blocs de mémoire sont alloués dynamiquement pour la décrire. Le premier d'entre eux, nommé bloc d'en-tête de base de données ou DBH (Data Base Header) permet d'accéder aux trois autres. Son adresse est placée dans le champ $DBHA_i$ fixé par le numéro de base de données.

La figure 2.3.1. représente la découpe d'un bloc d'en-tête de base de données. Voici la signification des différents champs qui le composent :

- NEOSCH demi mot binaire contenant le nombre d'objets complexes de la base de données.
- NERSCH demi mot binaire contenant le nombre de relations entre objets complexes de la base de données.
- OSCHA adresse (-64) du schéma interme des objets complexes (cfr. 2.3.2.).
- RSCHA adresse (-32) du schéma interne des relations entre objets complexes ou 0 si NERSCH = 0. (cfr. § 2.3.3.).
- ACA adresse de la zone de contrôle des accès (cfr. § 2.3.4.).
- DBNAME nom de la base de données (17 caractères dont, au moins, les 8 derniers sont à blanc).
- UIPFX numéro de la base de données (1 caractère).
- DBSTATUS état de la base de données (1 caractère contenant 1 code binaire) :

  DBSTATUS = 0 signifie que la base de données est active, c'est-à-dire qu'au moins un OPEN a été effectué.

  DBSTATUS = 1 signifie que la base de données est inactive, c'est-à-dire qu'aucune tâche ne l'exploite plus.

  DBSTATUS = 2 signifie qu'une erreur a été détectée dans le contenu de la base de données, soit par SESAM, soit par le noyau et qu'en conséquence, plus aucune tâche ne peut exploiter cette base de données.

- NOPEN nombre de tâches (de TCB) ayant effectué sur la base de données un open non clôturé (1/2 mot binaire) (DBSTATUS = 1 ⇒ NOPEN = 0).

- La fin du bloc d'en-tête de base de données se compose de 32 groupes de 2 caractères nommés $UISFX_i$ et $UIFLAG_i$ (i = A,...,Z,1,...,6)
  où $UISFX_i$ = i
  $UISFLAG_i$ = 0 (binaire) si le numéro d'utilisateur n_i formé de la concaténation de UIPFX et de $UISFX_i$ n'est pas actif pour SESAM;
  $UISFLAG_i$ = 1 si la base de données a été ouverte sous le numéro d'utilisateur n_i, mais qu'aucune tâche n'utilise n_i pour interroger SESAM;
  $UISFLAG_i$ = 2 si une tâche utilise n_i pour interroger SESAM.

```
DDR2  STMNT M SOURCE

487                 #DBH                                                  +
488 1 *-------------------------------------------------------------------I
489 1 *                    DATA BASE HEADER.                              +
490 1 *-------------------------------------------------------------------+
491 1 *
492 1 *    +-------- NEOSCH
493 1 *    I
494 1 *    I      +-- NERSCH
495 1 *    I      I
496 1 *    V      V
497 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
498 1 *   I      I      I              I      I      I      I      I      I
499 1 *   I      I      I    OSCHA     I    RSCHA    I    A C A    I      I
500 1 *   I      I      I              I              I             I      I
501 1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
502 1 *
503 1 *   +---+- - - - - -+---+---+---+---+---+---+---+---+---+--  - - -
504 1 *   I              I   I   I//I      I   I   I
505 1 *   I    DBNAME    I   I   I//INOPENI   I   I  (32 ENTRIES)
506 1 *   I (17.CHARACT.)I   I   I//I      I   I   I
507 1 *   +---+- - - - - -+---+---+---+---+---+---+---+---+---+--  - - -
508 1 *                    ^   ^                 ^   ^
509 1 *                    I   I                 I   I
510 1 *          UIPFX -----+   I                 I   +-- UIFLAG
511 1 *                         I                 I
512 1 *          DBSTATUS ------+                 +----- UISFX
513 1 *
514 1 *
515 1 * NEOSCH     NUMBER OF ENTRIES IN  OBJECT  SCHEMA.
516 1 * NERSCH     NUMBER OF ENTRIES IN RELATION SCHEMA.
517 1 * OSCHA      (ADDRESS OF  OBJECT  SCHEMA)-64 .
518 1 * RSCHA      (ADDRESS OF RELATION SCHEMA)-36 .
519 1 * ACA        ADDRESS OF ACCESS CONTROL AREA.
520 1 * DBNAME     NAME OF THE DATABASE.
521 1 * UIPFX      DATABASE NUMBER USED AS FIRST CHARACTER OF USER-ID.
522 1 * DBSTATUS   DATA BASE STATUS;
523 1 *            X'00' DATA BASE ACTIVE,
524 1 *            X'01' DATA BASE INACTIVE,
525 1 *            X'02' DATA BASE LOCKED.
526 1 * NOPEN      NUMBER OF TASKS USING THIS DATABASE.
527 1 * UISFX      VALUE OF USER-ID-SUFFIX ('A',....,'Z','1',...,'6').
528 1 * UIFLAG     X'00' CORRESPONDING USER-ID IS NOT OPENED.
529 1 *            X'01' CORRESPONDING USER-ID IS OPENED.
530 1 *            X'02' CORRESPONDING USER-ID IS CURRENTLY USED.
```

Figure 2.3.1.

```
DR2  STMNT M SOURCE

     531 1 DBH          DSECT
     532 1 NEOSCH       DS     H
     533 1 NERSCH       DS     H
     534 1 OSCHA        DS     F
     535 1 RSCHA        DS     F
     536 1 ACA          DS     F
     537 1 DBNAME       DS     CL17
     538 1 UIPFX        DS     CL1
     539 1 DBSTATUS     DS     CL1
     540 1              DS     CL1
     541 1 NOPEN        DS     H
     542 1 UISFX        DS     CL1
     543 1 UIFLAG       DS     CL1
     544 1 SPHINX       CSECT
```

Figure 2.3.1. (suite)

2.3.2. Schéma interne des objets complexes

Ce bloc de mémoire est alloué lors d'un premier OPEN de la base de données. Il contient autant d'entrées OSCH qu'il y a d'objets complexes dans la base de données. La figure 2.3.2. exhibe la découpe d'une entrée OSCH. La ième de ces entrées fournit la description nécessaire au noyau de l'objet complexe de numéro i. Le schéma interne est garni lors du premier OPEN par une lecture séquentielle à partir de la base de données.

Voici maintenant la signification des différentes zones composant une entrée OSCH.

- SNPTR : nombre de pointeurs dans une réalisation (1/2 mot binaire)
- SNCNT : nombre de compteurs dans une réalisation (1/2 mot binaire)
- SHLG, SHNL, SPLG, SCLG, SBLG, SNAHCH sont des zones utilisées uniquement par les modules de traitement de l'ordre SUPPRESS. On trouvera leur signification dans les listings de ces modules.
- SNAPTR : nombre i d'aspects pour la zone des pointeurs ($0 \leqslant i \leqslant 5$) (1 byte binaire)
- SNACNT : nombre j d'aspects pour la zone des compteurs ($0 \leqslant j \leqslant 5$) (1 byte binaire)
- SASPI : zone de 45 caractères contenant les noms des i aspects réservés pour la zone des pointeurs suivis des j aspects réservés pour la zone des pointeurs suivis de 1 à 5 aspects utiles à l'ordre SUPPRESS.

```
DDR2  STMNT M SOURCE
                     #OSCH                                                    I
        1 *-----------------OBJECT SCHEMA ENTRY.------------------------------+
        1 *
        1 *-------------------------------------------------------------------
        1 *
        1 *                                                    +---+---+---+---+---+---+---+---+
        1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+    I       I       I       I
        1 *   I       I       I       I       I       I       I       I       I
        1 *   ISNPTRISNCNTISHLG ISHNL ISTLG ISPLG ISCLG ISBLG I
        1 *   I       I       I       I       I       I       I       I       I
        1 *   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
        1 *                                                    -+---+---+---+
        1 *   +---+---+---+---+---+---+---+---+---+---+---+-     I       I
        1 *   I   I   I   I SASP1   I SASP2   I   . . .  I SASP15 I
        1 *   I   I   I   I         I         I          I       I
        1 *   I   I   I   I         I         I         -+---+---+---+
        1 *   +---+---+---+---+---+---+---+---+---+---+---+-
        1 *    ^   ^   ^
        1 *    I   I   I
        1 *    I   I   +-- SNAHCH
        1 *    I   I
        1 *    I   +------ SNACNT
        1 *    I
        1 *    +--------- SNAPTR
        1 *
        1 *
        1 * SNPTR      NUMBER OF POINTERS.
        1 * SNCNT      NUMBER OF COUNTERS.
        1 * SHLG       LENGTH OF SEARCH HIERARCHY.
        1 * SHNL       MAX. NUMBER OF LEVELS IN SEARCH-HIERARCHY.
        1 * STLG       TOTAL LENGTH TO BE RESERVED.
        1 * SPLG       TOTAL LENGTH OF POINTER AREA.
        1 * SCLG       TOTAL LENGTH OF COUNTER AREA.
        1 * SBLG       TOTAL LENGTH OF POINTER ARRAYS.
        1 * SNAPTR     NUMBER OF ASPECTS FOR POINTERS.
        1 * SNACNT     NUMBER OF ASPECTS FOR COUNTERS.
        1 * SNAHCH     NUMBER OF ASPECTS FOR SEARCH HIERARCHY.
        1 * SASPI      LIST OF ASPECT NAMES.
```

Figure 2.3.2.

DDR2 STMNT M SOURCE

```
583 1 OSCH      DSECT
584 1 SNPTR     DS     H
585 1 SNCNT     DS     H
586 1 SHLG      DS     H
587 1 SHNL      DS     H
588 1 STLG      DS     H
589 1 SPLG      DS     H
590 1 SCLG      DS     H
591 1 SBLG      DS     H
592 1 SNAPTR    DS     CL1
593 1 SNACNT    DS     CL1
594 1 SNAHCH    DS     CL1
595 1 SASPI     DS     CL3
596 1           DS     14CL3
597 1 SPHINX    CSECT
```

Figure 2.3.2. (suite)

2.3.3. Schéma interne des relations

Si la base de données contient au moins une relation entre objets complexes, un bloc de mémoire est alloué, lors du premier OPEN pour contenir la description, nécessaire au noyau, de chacune de ces relations.
La ième entrée RSCH décrit la relation de numéro i. Comme pour les objets complexes le schéma interne des relations est chargé la première fois qu'un programme utilisateurs effectue un ordre OPEN relatif à cette base de données.

La figure 2.3.3. représente une entrée RSCH dont les différentes zones sont décrites ci-dessous :

- SRTYP : 1 caractère décrivant le type d'implémentation de la relation.
  SRTYP = 1,2,3,4 selon que la relation est de type 1-1, 1-n, m-1, m-n.
- SAPTRA : si la relation est de type 1-n ou m-n, le nom de l'aspect utilisé pour ses blocs de pointeurs (3 caractères).
- SORGN : numéro de l'objet complexe origine (1/2 mot binaire).
- STRGN : numéro de l'objet complexe cible (1/2 mot binaire).
- SI, SJ, SK, SL : mots binaires contenant les valeurs des paramètres i,j,k,l de la relation.
- SICNT : indice du compteur d'occurrences de la relation, dans la zone des compteurs de la réalisation origine (1/2 mot binaire).
- SIFPA : si la relation est de type 1-1 ou m-1, indice du pointeur vers la réalisation cible, dans la zone des pointeurs de la réalisation origine; si la relation est de type 1-n ou m-n, indice du pointeur vers le premier "bloc de pointeurs" associé à l'origine (1/2 mot binaire).
- SILPA : si la relation est de type 1-n ou m-n, indice du pointeur vers le dernier bloc de pointeurs.
- SIBPA : si la relation est de type 1-n ou m-n, indice du pointeur vers le "meilleur" bloc de pointeurs.
- SIFSC : si la relation est de type 1-n ou m-n, indice, dans la zone des pointeurs, du mot contenant le nombre de pointeurs du "meilleur" bloc.
- SIRLN : numéro de la relation inverse (1/2 mot binaire).

```
DDR2  STMNT M SOURCE

                #RSCH
1 *-------------------------------------------------------------+
1 *                RELATION SCHEMA ENTRY.                       I
1 *-------------------------------------------------------------+
1 *
1 *
1 *    +-- SRTYP
1 *    I
1 *    V
1 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1 *    I I         I     I     I           I           I
1 *    I I SAPTRA ISORGNISTRGNI    S I    I    S J    I
1 *    I I         I     I     I           I           I
1 *    I I         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1 *    I           I                I     I     I     I     I     I     I     I
1 *    I    S K    I    S L         ISICNTISIFPAISILPAISIBPAI
1 *    I           I                I     I     I     I     I
1 *    I           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1 *    I     I     I
1 *    ISIFSCISIRLNI
1 *    I     I     I
1 *    +-+-+-+-+-+-+
1 *
1 *
1 *
1 * SRTYP    IMPLEMENTATION TYPE.
1 * SAPTRA   ASPECT NAME OF POINTER ARRAY.
1 * SORGN    ORIGIN OBJECT NUMBER.
1 * STRGN    TARGET OBJECT NUMBER.
1 * SI       MIN. NUMBER OF TARGET OCCURRENCES.
1 * SJ       MAX. NUMBER OF TARGET OCCURRENCES.
1 * SK       MIN. NUMBER OF ORIGIN OCCURRENCES.
1 * SL       MAX. NUMBER OF ORIGIN OCCURRENCES.
1 * SICNT    INDEX OF CORRESPONDING COUNTER.
1 * SIFPA    INDEX OF RECORD NUMBER CORRESPONDING TO THE
1 *          FIRST POINTER ARRAY OR RECORD NUMBER OF
1 *          TARGET IF SJ = 1.
1 *
1 * SILPA    INDEX OF RECORD NUMBER CORRESPONDING TO THE
1 *          LAST POINTER ARRAY.
1 *
1 * SIBPA    INDEX OF RECORD NUMBER CORRESPONDING TO THE
1 *          BEST POINTER ARRAY.
1 *
1 * SIFSC    INDEX OF THE BEST POINTER ARRAY CAPACITY.
1 * SIRLN    INVERSE RELATION NUMBER.
```

Figure 2.3.3.

```
>DR2  STMNT M SOURCE

      640 1 RSCN       DSECT
      641 1 SRTYP      DS     CL1
      642 1 SAPTRA     DS     CL3
      643 1 SORGN      DS     H
      644 1 STRGN      DS     H
      645 1 SI         DS     F
      646 1 SJ         DS     F
      647 1 SK         DS     F
      648 1 SL         DS     F
      649 1 SICNT      DS     H
      650 1 SIFPA      DS     H
      651 1 SILPA      DS     H
      652 1 SIBPA      DS     H
      653 1 SIFSC      DS     H
      654 1 SIRLN      DS     H
      655 1 SPHINX     CSECT
```

Figure 2.3.3. (suite)

2.3.4. Zone de contrôle des accès (ACA)

Ce bloc de mémoire de n+2 bytes mémorise la nature des accès effectuées sur chacun des n objets complexes de la base de données, il mémorise aussi le nombre de tâches accédant à chaque objet.

Le premier demi-mot de cette zone contient, en binaire, le nombre d'objets complexes dont la zone contrôle des accès. Ce nombre, en principe égal au nombre d'objets de la base de données peut lui être momentanément inférieur (cas d'un OPEN inclus dans un bloc REACH, par exemple).

Le ième byte suivant est relatif à l'objet complexe numéro i de la base de données. Désignons le par ACA_i.

ACA_i = 0 (binaire) si aucune tâche n'est susceptible d'accéder aux réalisations de l'objet complexe i, ni pour les lire, ni pour les modifier.

ACA_i = l (codé binaire, $1 \leqslant l < 255$) si l tâches ont reçu l'autorisation de lire les réalisations de l'objet complexe i.

ACA_i = 255 (binaire) si une tâche a reçu l'autorisation de modifier les réalisations de l'objet complexe i.

On voit qu'un objet complexe ne peut être modifié par plus d'un programme DML à la fois et ne peut être consulté par plus de 254 programmes simultanés (nombre nettement supérieur à la charge normale du BS 2000).

2.4. Exemple graphique

La figure 2.4. présente schématiquement l'ensemble des tables du noyau et la façon d'y accéder.
Dans cet exemple, on a représenté 3 TCB.

Le premier (8760) effectue un premier open de la base de données n° B (chargement des schémas internes). Le second (8754) a reçu l'autorisation d'accès à la base de données n° 6 et consulte l'objet n° 2. Le troisième (8757) désire mettre à jour les objets n° 1 et 2 de cette même base de données et est donc suspendu puisque cette demande d'accès est incompatible avec la précédente.

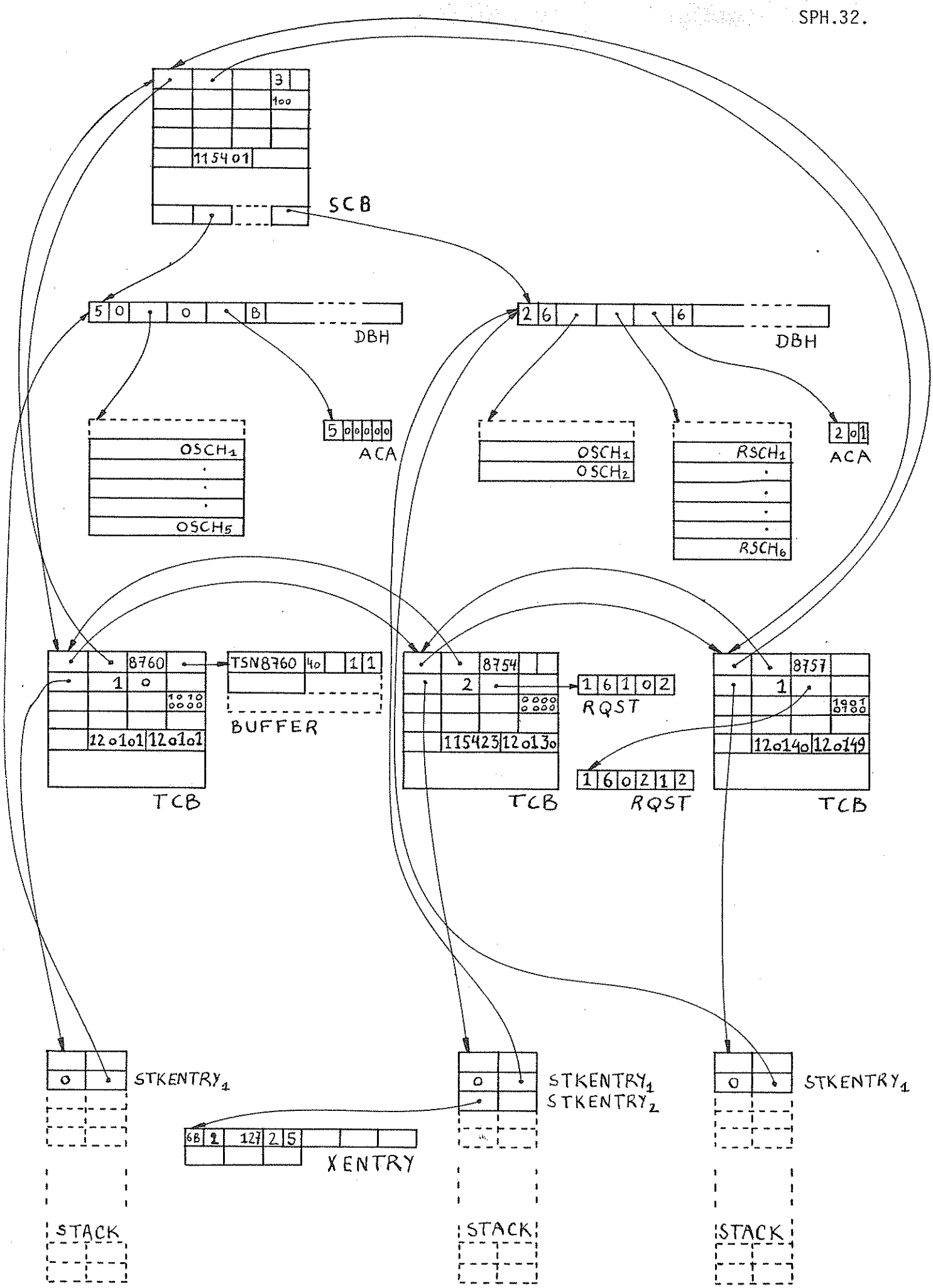

SCB
DBH
DBH
OSCH1
OSCH5
ACA
OSCH1
OSCH2
RSCH1
RSCH6
ACA
TSN8760
BUFFER
RQST
RQST
TCB
TCB
TCB
STKENTRY1
STKENTRY1
STKENTRY2
STKENTRY1
XENTRY
STACK
STACK
STACK

Figure 2.4.

3. Les modules du noyau

3.1. Types de modules

Nous distinguerons trois types principaux de modules intervenant dans la constitution du noyau.

- Les modules de gestion des tables assurent la création, l'initialisation, la recherche et la suppression des différentes tables décrites au § 2. Le plus fondamental de ces modules est SYSMEM qui gère l'allocation dynamique de la mémoire.
  Ces modules sont constamment utilisés par ceux des deux autres types.
- Les modules du superviseur gèrent l'ordre dans lequel seront exécutées les actions demandées par les programmes d'application et se chargent, en particulier, des communications avec SESAM et de la régulation des accès concurrents.
- Les modules spécifiques à chaque ordre DML se chargent de l'analyse détaillée des buffers de communication et en réalisent le traitement proprement dit. Toutefois, lorsque ce traitement nécessitera l'intervention de SESAM, ils rendront momentanément, la main au superviseur. Pour la description détaillée de ces modules nous renvoyons le lecteur aux listings.

3.2. Modules de gestion des tables

3.2.1. SYSMEM - Module d'allocation dynamique de mémoire

Ce module gère une zone de mémoire centrale de 256 K bytes (mémoire virtuelle), découpée en blocs possédant la structure de la figure 3.2.1.1. Parmi les champs décrits, dans cette figure, TAG1, TAG2, ID1, ID2 occupent 1 byte SIZE1, SIZE2 sont des demi-mots binaires, FLINK et BLINK occupent un mot.
Un bloc de mémoire est dit libre si TAG2 = TAG2 = 0 (binaire),
il est dit réservé si TAG1 = TAG2 = 1.
Les champs SIZE1, SIZE2 contiennent la taille du bloc en doubles mots (8 bytes).
Les champs ID1, ID2 contiennent 2 chiffres hexadécimaux précisant l'utilisation du bloc (F0 : TCB, F1 : STACK, F2 : RQST, F3 : XENTRY, F4 : WORK, E0 : DBH, E1 : OSCH, E2 : RSCH, E3 : ACA, C1 : BUFFER, etc.).
Leur rôle est uniquement documentaire et utile surtout pour la mise au point du noyau (lecture des DUMPS). Les blocs libres participent à une chaîne double issue du SCB. Dans cette chaîne, FLINK pointe vers le bloc suivant et BLINK le précédent (voir § 2.1. : FFAREA, LFAREA).
Les blocs réservés participent à une parmi n + 1 classes où n est le nombre de TCB existants. Une de ces classes est celle des blocs réservés par le noyau pour le fonctionnement global du système (TCB, DBH, OSCH, RSCH, ACA). Les n autres classes sont celles des blocs spécifiquement réservés pour les n tâches connectées au noyau.
Les blocs réservés pour le système sont doublement chaînés entre eux par FLINK et BLINK et au SCB (voir § 2.1. : FSAREA, LSAREA).
Les blocs réservés pour une tâche particulière sont doublement chaînés entre eux et au TCB de la tâche (voir § 2.2.1. : FRAREA, LRAREA).
La figure 3.2.1.2. présente une configuration de mémoire particulière.
Lorsqu'une tâche disparaît du système, tous les blocs encore réservés pour cette tâche peuvent être atteints par le double chaînage au TCB et réintégrés à la chaîne des blocs libres (Garbage Collection).

Nous décrivons maintenant brièvement les algorithmes de réservation et de libération des blocs. Pour plus de détails, on consultera la référence [*] aux pages 435-442.

GETAREA - Réservation d'un bloc (First Fit Method)

On parcourt la chaîne des blocs libres jusqu'à en trouver un de taille supérieure ou égale à celle du bloc désiré. Si le bloc libre rencontré est "raisonnablement" plus grand que le bloc demandé, il est scindé en 2 parties : la première, ayant la taille désirée, est insérée dans la chaîne des blocs réservés, la seconde, réintégrée dans celle des blocs libres. Sinon, le bloc tout entier est extrait de la chaîne des blocs libres et inséré dans celle des blocs réservés.

---

[*] D.E., KNUTH. The art of Computer programming. Volume 1.

GETAREA est le nom du point d'entrée, permettant la réservation d'un bloc. Avant d'y faire appel, il faut charger dans le régistre 1 l'adresse d'une zone contenant, dans son premier mot, le nombre de bytes désirés, dans le deuxième l'adresse du SCB ou du TCB auquel rattacher le bloc et dans le dernier byte, la valeur de ID1 et ID2 pour le bloc.
Au retour, le registre 15 contiendra l'adresse + 12 du bloc, c'est-à-dire l'adresse du début de la zone utile.
La figure 3.1.2.3. représente l'argument d'appel et le point d'entrée GETAREA.

FREEAREA - Libération d'un bloc

Lors de la libération d'un bloc réservé, on examine l'état des blocs contigus, au moyen de leurs champs TAG1 ou TAG2. S'ils sont libres, on les fusionne avec le bloc à libérer et on insère le bloc ainsi obtenu dans la chaîne des blocs libres.

FREEAREA est le nom du point d'entrée permettant de libérer un bloc. Avant d'y faire appel, on chargera, dans le registre 1, l'adresse d'un mot contenant l'adresse + 12 du bloc à libérer, c'est-à-dire l'adresse de la zone de mémoire "utile"; celle qui avait été fournie par GETAREA.

La figure 3.1.2.4. représente l'argument d'appel et le point d'entrée FREEAREA.

INITAREA - Initialisation de la mémoire dynamique

Le module de gestion de la mémoire dynamique contient un troisième point d'entrée INITMEM, qui doit être appelé une seule fois et le premier.
Il réserve la zone de 256 K de mémoire,la formate comme un seul bloc libre et l'insère dans la chaîne des blocs libres.

Remarque

Dans toutes les descriptions de zones dynamiques données précédemment, on a toujours ignoré les champs TAG1, TAG2, FLINK, BLINK, ID1, ID2, SIZE1, SIZE2 de gestion et on a seulement décrit le contenu de la zone "utile", ce qui est légitime puisqu'on peut ignorer complètement le mécanisme d'allocation de mémoire, à condition de veiller à ne pas écrire en dehors des bornes de la zone utile, pour ne pas détruire les champs de gestion. Dans la suite, on ne parlera plus de ces champs.

```
DDR2  STMNT M SOURCE

      * FORMAT OF A MEMORY AREA.
      * ------------------------
      *
      *          +------ TAG1
      *          I
      *          I   +-- ID1
      *          I   I
      *          V   V
      *          +---+---+---+---+---+---+---+---+---+---+---+---+
      *          I   I   I       I                   I           I
      *          I   I   ISIZE1I     FLINK           I   BLINK   I
      *          I   I   I       I                   I           I
      *          +---+---+---+---+---+---+---+---+---+---+---+---+
      *          I                                               I
      *          I                                               I
      *          I                                               I
      *
      *                                                          I
      *                                                          I
      *                                                          I
      *                                          +---+---+---+---+
      *                                          I   I   I       I
      *                                          I   I   ISIZE2I
      *                                          I   I   I       I
      *          +-----------------------------+---+---+---+---+
      *                                          ^   ^
      *                                          I   I
      *                                          I   +-- ID2
      *                                          I
      *                                          +------ TAG2
      *
      *
      *
      *
      *  TAG1      X'00' AREA IS FREE.
      *            X'01' AREA IS RESERVED.
      *
      *  ID1       IDENTIFICATION OF CORRESPONDING USAGE.
      *
      *  SIZE1     SIZE OF AREA IN DOUBLE-WORDS.
      *
      *  FLINK     IF TAG1 = X'00' ADDRESS OF NEXT FREE AREA.
      *            IF TAG1 = X'01' ADDRESS OF NEXT RESERVED
      *                            AREA FOR THE SAME TCB.
      *
      *  BLINK     IF TAG1 = X'00' ADDRESS OF PRIOR FREE AREA.
      *            IF TAG1 = X'01' ADDRESS OF PRIOR RESERVED
      *                            AREA FOR THE SAME TCB.
      *
      *  TAG2      SAME AS TAG1  (FOR COLLAPSING PROCESS).
      *
      *  ID2       SAME AS ID1.
      *
      *  SIZE2     SAME AS SIZE1 (FOR COLLAPSING PROCESS).
```

Figure 3.2.1.1.

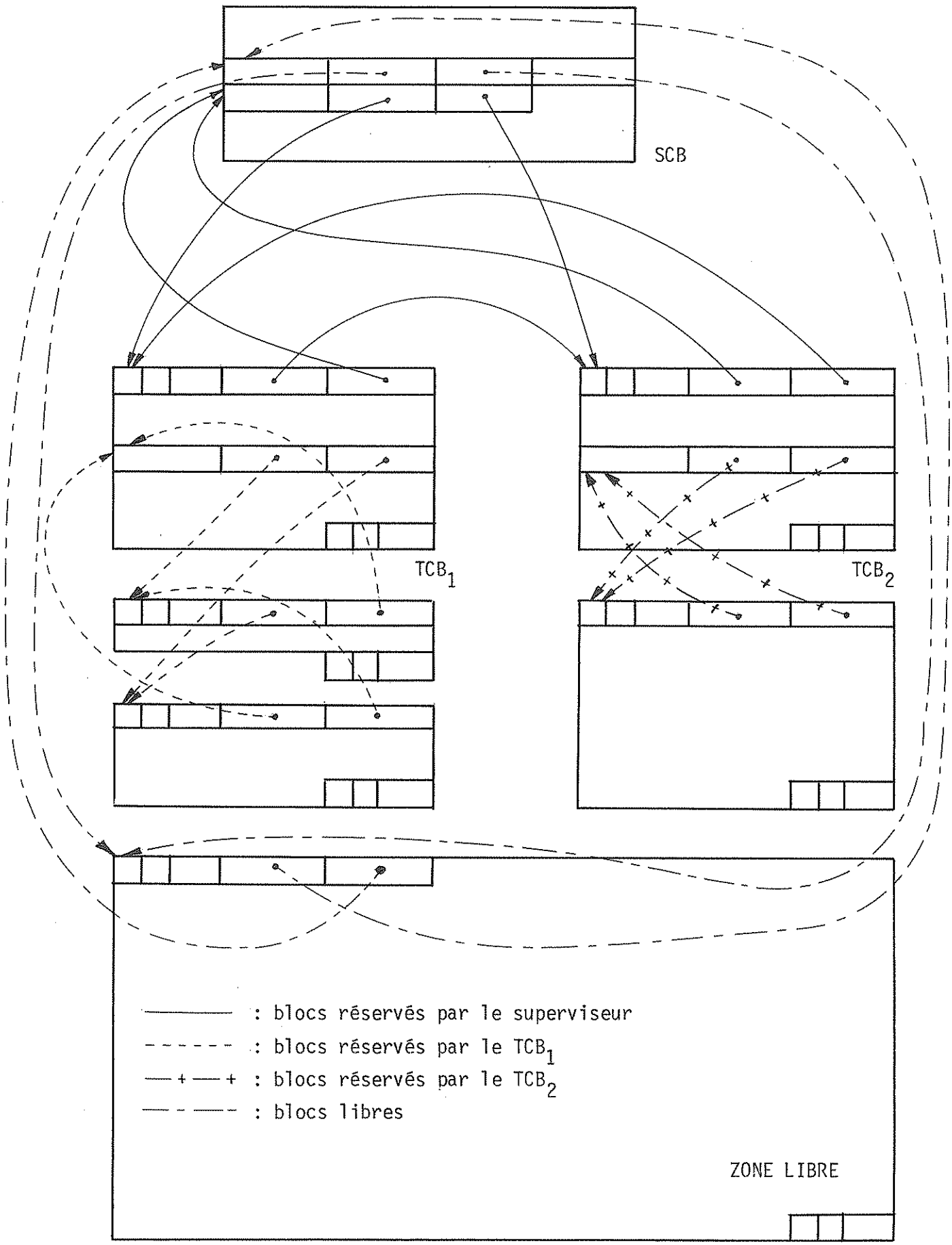

SCB
TCB1
TCB2
: blocs réservés par le superviseur
: blocs réservés par le TCB1
: blocs réservés par le TCB2
: blocs libres
ZONE LIBRE

Figure 3.2.1.2.

```
DDR2  STMNT M SOURCE
                                                                    +
      *=================================================================+
      *          ENTRY POINT FOR MEMORY RESERVATION.                  I
      *=================================================================+
      *
      * STRUCTURE OF THE ARGUMENT.
      * --------------------------
      *                  +---+---+---+---+---+---+---+---+
      *                  I                               I   I
      *                  I                I               I3  I
      *                  I       1        I       2       I   I
      *                  I                I               I   I
      *                  +---+---+---+---+---+---+---+---+
      *                  ^
      *                  I
      *            (R1) --+
      *
      *  (1) SIZE OF THE AREA TO BE RESERVED (IN BYTES).
      *  (2) ADDRESS OF CORRESPONDING TCB.
      *  (3) IDENTIFICATION OF CORRESPONDING USAGE.
      *
      *
      *   AFTER THE ALLOCATION PROCESS, GENERAL REGISTER 15
      * CONTAINS THE ADDRESS OF THE FIRST AVAILABLE BYTE IN
      * RESERVED AREA. IF CORE MEMORY IS FULL, A MESSAGE IS
      * WRITTEN OUT AND A SYSTEM DUMP OCCURS.


                ENTRY GETAREA
      GETAREA   EQU   *
                USING *,15
                CNOP  0,4
                STM   13,12,SAVAREA        |SAVE REGISTERS.
                L     12,*+8               |LOAD BASE REGISTER.
                B     *+8
                DC    A(BEGIN)
                DROP  15                   |DEFINE BASE REGISTER.
                USING BEGIN,12
```

Figure 3.2.1.3.

```
ADR2  STMNT M SOURCE
                                                                    +
                                                                    I
  395   *=================================================================+
  396   *          ENTRY POINT FOR MEMORY LIBERATION.
  397   *=================================================================+
  398   *
  399   * STPUCTURE OF THE ARGUMENT.
  400   * ---------------------------
  401   *                 +--+--+--+--+
  402   *                 I           I
  403   *                 I    (1)    I
  404   *                 I           I
  405   *                 +--+--+--+--+
  406   *                 ^
  407   *                 I
  408   *        (R1)  --+
  409   *
  410   *
  411   * (1)    ADDRESS OF RELEASED AREA.


                         ENTRY FREEAREA
  412
  413   FREEAREA EQU     *
  414            USING   *,15
  415            CNOP    0,4                  |SAVE REGISTERS.
  416            STM     13,12,SAVAREA        |LOAD BASE REGISTER.
  417            L       12,*+8
  418            B       *+8
  419            DC      A(BEGIN)
  420            DROP    15                   |DEFINE BASE REGISTER.
  421            USING BEGIN,12
```

Figure 3.2.1.4.

3.2.2. SYSITC - Gestion des buffers de communication

Ce module gère les communications entre le noyau et les programmes DML. Il ne traite pas la communication entre le noyau et SESAM laquelle est gérée par le module SESMOD (du système SESAM). Il possède 4 points d'entrée.

Point d'entrée INITCOM - Ouverture de communication

Ce point d'entrée, appelé au début de l'exécution du noyau, identifie le noyau au système de communications entre tâches, sous le nom SES0001 (macro OPCOM). Si le nom SES001 est déjà connu du système de communication, cela signifie que le noyau est déjà chargé, dans une autre tâche et la présente exécution du noyau d'arrête.

Point d'entrée GETBUF - Prise en charge d'un buffer de communication

On examine le premier buffer de la file associée au noyau; s'il n'y en a pas, on attend durant quinze secondes l'arrivée d'un buffer (macro revnt).
Si le buffer provient de SESAM ou si aucun buffer n'a été reçu après quinze secondes, le contrôle est rendu au programme appelant avec la valeur o dans le registre 1.
Si le buffer provient d'un programme DML, un bloc de mémoire est alloué, dans lequel il est copié, puis l'adresse de ce bloc est placée dans le registre 1 et le contrôle rendu au programme appelant.

Point d'entrée PUTBUF - Envoi d'un buffer à un programme DML

Avant d'appeler ce point d'entrée, il faut mettre dans le registre 1 l'adresse d'un buffer "réponse" destiné à une tâche DML.

Le buffer est envoyé, par une macro SEVNT, à la tâche puis le bloc mémoire du buffer est désalloué.

Point d'entrée STOPCOM - Fermeture de communication

Ce point d'entrée est appelé lors de la phase d'arrêt du système, après fermeture de toutes les bases de données et achèvement de toutes les tâches DML, pour déconnecter le noyau du système de communication (macro CLCOM).

3.2.3. SYSTCB, SYSCAN - Gestion des TCB

Le module SYSTCB assure la gestion normale des TCB : création, recherche, désactivation. Le module SYSCAN clôture le traitement de la tâche associée à un TCB lorsqu'une erreur grave a été détectée dans son exécution.

SYSTCB possède 3 points d'entrée GETTCB, RETRTCB, FREETCB et SYSCAN 1 point d'entrée CANTCB que nous décrivons ci-dessous.

Point d'entrée GETTCB - Recherche d'un TCB inactif - Création d'un TCB

Lors de l'appel à GETTCB, le registre 1 doit contenir l'adresse d'un buffer de communication issu d'une tâche d'application.

On balaie la chaîne des TCB jusqu'à en trouver un qui ait même numéro de tâche que le buffer (SNAME = "TSN <TSN >").

Si un tel TCB est trouvé,

- on vérifie qu'il est inactif (TCW(1) = 0);
  sinon, le bit TCW(7) est mis à 1 pour provoquer la destruction de ce TCB par SYSCAN, lorsque le superviseur activera le processus d'élimination des TCB invalides. La valeur 85 est placée dans le champ SBOX du SCB et le contrôle rendu au superviseur.
- Si le TCB est inactif, mais "marqué pour être détruit" (TCW(7)=1), on rend la main au superviseur avec SBOX = 87.
- Si aucune erreur n'est détectée, l'adresse du TCB est placée dans CTCBA (le TCB est dit "courant"), l'adresse du buffer placée dans BUFA, l'heure d'arrivée du buffer placée dans TIMER2. Puis le TCB est déplacé à la fin de la chaîne des TCB de façon à conserver le tri de cette chaîne par ordre croissant des valeurs de TIMER2.
- Ensuite, le contrôle est rendu au superviseur avec SBOX = 00.

S'il n'existe aucun TCB ayant même numéro de tâche que le buffer,

- si le système est dans sa phase terminale, le contrôle est rendu au superviseur avec SBOX = 88;
- sinon, on vérifie que le buffer correspond bien à un OPEN de niveau 1 (BCODE = BLEVEL = 1) faute de quoi le contrôle est rendu avec SBOX = 87;
- si aucune erreur n'est détectée, un bloc de mémoire est alloué pour un nouveau TCB et placé à la fin de la chaîne des TCB.
  Le nombre de TCB connus du système est augmenté de 1 (NTCB), le nouveau TCB est rendu courant, les valeurs de BUFA, TSN, TIMER1, TIMER2 sont mises à jour.
  Puis un bloc de mémoire est alloué pour le stack des réalisations. Son adresse est placée dans STKA.

- Enfin, le contrôle est rendu au superviseur avec SBOX = 00.

Remarque : Pendant les mises à jour de la chaîne des TCB, le bit SCW(2) est mis à 1, pour éviter un balayage de cette chaîne par le module SYSINT qui peut à tout moment interrompre le noyau pour interroger l'état du système. Un tel balayage ne peut, en effet, avoir lieu que si la chaîne des TCB est dans un état cohérent.

Point d'entrée RETRTCB - Recherche d'un TCB sur la valeur du TCW

Le byte de droite du régistre 1 est utilisé comme masque de recherche, par ce point d'entrée. La chaîne des TCB est parcourue, à partir du TCB suivant le TCB courant, jusqu'à trouver la fin de la chaîne ou un TCB dont le premier byte du mot d'état a tous les bits spécifiés par le masque, positionnés à 1.
Si un tel TCB est trouvé, son adresse est placée dans la zone CTCBA du SCB, sinon la valeur o y est placée. Le contrôle est ensuite rendu au programme appelant.

Remarques

1. Si l'on veut parcourir la chaîne à partir du premier TCB, il faut mettre dans CTCBA l'adresse du SCB.
2. Avant d'appeler RETRTCB, il faut s'assurer que CTCBA contient bien l'adresse d'un TCB ou du SCB (et non o par exemple).

Point d'entrée FREETCB - Libération des blocs alloués à une tâche

Le point d'entrée FREETCB est activé lorsqu'une tâche d'application effectue un CLOSE de niveau 1. A ce moment, le programme utilisateur se déconnecte (peut-être momentanément) du noyau. Aussi, tous les blocs qui lui étaient encore alloués sont libérés. Il est prévu, dans le noyau, qu'un bloc, alloué par un ensemble de modules réalisant une certaine fonction, soit désalloué par ce même ensemble de modules. Aussi, lors de l'appel de FREETCB, les seuls blocs restant alloués à la tâche sont théoriquement le TCB et le stack des réalisations.
Toutefois, pour plus de sécurité, on parcourra la chaîne des blocs réservés par le TCB (pointeurs FFAREA et LFAREA) et on libérera tous ces blocs avant de faire de même pour le TCB. Si la chaîne n'est pas vide, un message sera affiché au terminal du système pour signaler cette exécution anormale.

L'appel de FREETCB est fait après avoir mis dans CTCBA l'adresse du TCB à libérer.

Remarque : Ce point d'entrée est aussi appelé, en cas de terminaison prématurée d'une tâche, juste après CANTCB.

Point d'entrée CANTCB - Libération des ressources allouées à une tâche

Le point d'entrée CANTCB est appelé, lors de la détection d'une erreur dans l'exécution d'une tâche ou si l'on a introduit une commande CAN <n° de tâche> au terminal du système (module SYSINT). Sa fonction est de libérer les ressources allouées à la tâche, par le noyau, pour permettre à celui-ci de les attribuer à d'autres tâches.

L'appel de CANTCB nécessite la mise dans CTCBA de l'adresse du TCB. Les autorisations d'accès données à la tâche lui sont retirées par un appel au point d'entrée DML2OC du module SYS2OB. Ensuite, tous les numéros d'utilisateurs réservés à la tâche sont libérés $\begin{pmatrix} \text{NOPEN} := \text{NOPEN}-1 \\ \text{UIFLAG}_i := 1 \end{pmatrix}$ par le module SYSO2Y (point d'entrée : FREEUI).

De plus, si la tâche est seule à travailler sur une ou plusieurs bases de données, les blocs d'en-tête de ces bases de données sont libérés. (Point d'entrée FREEDBH du module SYSDBH).
Enfin, si la tâche est active, c'est-à-dire si elle attend une réponse du noyau, un buffer standard d'erreur lui est renvoyé.

3.2.4. SYSDBH - Gestion des DBH

Ce module assure la gestion des blocs d'en-tête de bases de données. Il comporte quatre points d'entrée : GETDBH, RETRDBH, FREEDBH, CANDBH, admettant les mêmes arguments d'appel : le byte de droite du régistre 1 contient un numéro de bases de données, CTCBA contient l'adresse du TCB courant.

Point d'entrée FREEDBH - Désactivation d'un DBH

Lorsque le dernier programme travaillant sur une base de données de n°i ($i \in \{A,\ldots,Z,1,\ldots,6\}$) effectue un CLOSE de cette base de données, ce point d'entrée est appelé, pour "désactiver" le DBH.

A première vue, on pourrait penser qu'une telle désactivation doive comporter la libération des blocs DBH, OSCH, RSCH, ACA. Mais une telle façon de faire risquerait d'être, à la fois, dangereuse et coûteuse. Tout d'abord, si l'on supprime le DBH, on permet à d'autres programmes de travailler, par la suite, sur une autre base de données ayant même numéro de base de données, ce qui est dangereux car le CLOSE effectué par le premier programme ne signifie pas nécessairement, que celui-ci ait définitivement abandonné le traitement de la première base de données. Or, il lui sera maintenant impossible d'y accéder. De plus, la place mémoire réservée par un DBH n'est pas si considérable qu'on ne puisse le conserver, même si il n'est plus actif. Par contre, la place mémoire allouée aux 3 autres blocs peut être considérable, surtout pour OSCH et RSCH. On ne les libérera, pourtant, pas tout de suite, car ces blocs sont garnis par des accès à la base de données dont le coût en temps d'exécution n'est pas négligeable et il se peut que le prochain OPEN effectué le soit pour la même base de données, nécessitant ainsi une réallocation des mêmes zones et leur garnissage. Aussi, on se contentera de mémoriser la désactivation du DBH et, lors du prochain OPEN, relatif à une base de données encore inconnue ou désactivée et sans schémas internes du noyau, on libérera la place utilisée par les blocs OSCH, RSCH, ACA des bases de données désactivées.

Finalement, le traitement réalisé par FREEDBH est le suivant :

- Branchement à RETRDBH pour obtenir l'adresse du DBH. Si celui-ci n'existe pas ou si la base de données est bloquée, le contrôle est rendu au programme appelant après mise du code d'erreur (99 ou 77) dans le champ TBOX du TCB courant.
- Mettre 1 dans DBSTATUS pour désactiver le DBH.
- Effectuer un CLOSE Sesam pour tous les numéros d'utilisateurs ouverts relativement à cette base de données (CLOSEDB).
- Retour au programme appelant avec TBOX = 00.

Point d'entrée GETDBH - Activation d'un DBH - Libération des schémas internes non actifs

Ce point d'entrée est appelé chaque fois qu'un OPEN est effectué par un programme d'application et aussi par le module du noyau qui traite le buffer de demande d'autorisations d'accès. Dans ce dernier cas, il se peut qu'un DBH soit créé pour une base de données sans que son nom soit connu (Mais seulement son numéro de base de données).

Le traitement effectué est le suivant :

- Rechercher l'adresse du DBH dans la zone $DBHA_i$ du SCB.
  Si le DBH existe et est bloqué, le contrôle est rendu au programme appelant avec TBOX = 77. Si le numéro de base de données n'est pas valide, le contrôle est rendu avec TBOX = 99.
- Si le DBH existe et est actif (DBSTATUS = 0), son adresse est placée dans le registre 1. Le contrôle est rendu avec TBOX = 00.
- Si le DBH existe et est désactivé (DBSTATUS = 1), il est réactivé par DBSTATUS := 0. Ensuite, si les schémas internes n'ont pas été libérés (NEOSCH > 0), le contrôle est rendu avec TBOX = 00, sinon on libérera au préalable les blocs OSCH, RSCH et ACA de tous les DBH inactifs.
- Si le DBH n'existe pas, il est créé et initialisé. Les blocs OSCH, RSCH et ACA de tous les DBH inactifs sont libérés, puis le contrôle est rendu au programme appelant avec TBOX = 00.

Point d'entrée RETRDBH - Recherche d'un DBH existant

Ce point d'entrée est utilisé par tout module désirant obtenir des renseignements sur une base de données supposée active.

Après vérification de la validité du numéro de base de données i, on examine le contenu de $DBHA_i$ ($i \in \{A,\ldots,Z,1,\ldots,6\}$).
Si i n'est pas valide ou si $DBHA_i = 0$, le contrôle est rendu au programme appelant avec TBOX = 99 (DBH inconnu) ou TBOX = 77 (BD bloquée). Sinon, le contenu de $DBHA_i$ est placé dans le registre 1. C'est l'adresse du DBH recherché.
Le contrôle est rendu au programme appelant avec TBOX = 00.

Point d'entrée CANDBH - Blocage d'une base de données

Ce point d'entrée est appelé lorsqu'une erreur est détectée dans le contenu de la base de données (Que ce soit par SESAM ou par un module du noyau). Dès ce moment, toutes les opérations en cours sur cette base de données doivent être suspendues.

Les actions effectuées par CANDBH sont les suivantes :

- Branchement à RETRDBH pour obtenir l'adresse du DBH. Si celui-ci n'existe pas ou si la base de données est déjà bloquée, le contrôle est rendu au programme appelant avec TBOX = 99 ou 77.
- Mettre 2 dans DBSTATUS pour bloquer la base de données. Dès ce moment, toute demande de renseignements sur cette base de données via RETRDBH sera refusée.
- Tenter d'effectuer les CLOSE SESAM nécessaire à la fermeture de la base de données. Ceci ne sera pas toujours possible si l'erreur a été détectée par SESAM.
- Libérer les blocs OSCH, RSCH, ACA de ce DBH.
- Retour au programme appelant avec TBOX = 00.

3.2.5. SYSREA - Création d'un bloc descripteur d'une réalisation

Ce module crée et initialise un bloc XENTRY ou procède au "nettoyage" d'un tel bloc. Son point d'entrée GETAREA est appelé par les modules du noyau réalisant le traitement des ordres REACH, ALLOCATE, PREPARE.

Lors de l'appel du module, le SCB fournit l'adresse du TCB courant (CTCBA), qui lui-même contient un pointeur (BUFA) vers un buffer issu d'un programme DML. Ce buffer contient tous les renseignements nécessaires à la création du bloc XENTRY : numéro de base de données, numéro d'objet complexe, niveaux d'imbrication et type de l'ordre DML.

- Un nouveau niveau est initialisé dans le stack des réalisations (STKENTRY).
- Si l'ordre DML est associé à un ALLOCATE de moindre niveau d'imbrication par une clause IN, le bloc XENTRY correspondant est réinitialisé et son adresse est placée dans la zone COBJA du dernier niveau STKENTRY.
- Si l'ordre DML n'est pas associé à un tel ALLOCATE, un nouveau bloc XENTRY est créé et initialisé dont la taille et les valeurs de XNPTR et XNCNT sont tirées du schéma interne des objets (RETRDBH). L'adresse du nouveau bloc est placée dans la zone COBJA du dernier niveau STKENTRY.
- Si aucune erreur n'est détectée dans le traitement ci-dessus, le contrôle est rendu au programme appelant avec TBOX = 00. Si une erreur est détectée dans le buffer, le contrôle est rendu avec TBOX = 99, il est rendu avec TBOX = 77 si le DBH est bloqué. Dans ces deux cas d'erreur, le stack des réalisations demeure dans l'état antérieur à l'appel.

3.3. Modules du superviseur

Le rôle de ces modules est de régler l'enchaînement des modules réalisant le traitement proprement dit des ordres DML contenus dans chaque tâche d'application, en essayant de réaliser une bonne utilisation globale des ressources. La ressource qu'il faut absolument chercher à optimiser est la communication avec SESAM. En effet, le noyau SPHINX, étant considéré par SESAM comme un utilisateur standard, ne peut envoyer à SESAM qu'une requête à la fois. Puisque toute simultanéité est impossible à ce niveau, il importe de ne jamais laisser SESAM inactif et, lorsque SESAM renvoie sa réponse, d'interrompre immédiatement tout traitement en cours, pour relancer, sans attendre, une nouvelle requête.
Les autres ressources qu'il faut essayer de distribuer au mieux entre les tâches utilisateurs sont les autorisations d'accès aux différents objets des différentes bases données. Rappelons que ces autorisations d'accès sont de deux types : l'accès en consultation, partageable, et l'accès en modification qui est exclusif.

3.3.1. SYSPHX - Le superviseur proprement dit

Le module SYSPHX est le module directeur du superviseur et donc aussi du noyau, il possède deux autres points d'entrée : CALLDB et SYSDB utilisés lorsqu'un module rend la main au superviseur pour communiquer avec SESAM.
SYSPHX effectue une première analyse des buffers extraits de la file par SYSITC, en fonction de quoi il confie le traitement de chaque buffer, soit à un module spécifique du noyau, soit à SESMOD pour les buffers envoyés par SESAM.
Ces derniers sont traités, en priorité, puisque SYSPHX cherche, avant tout, à rentabiliser l'utilisation de SESAM.

Nous allons décrire la logique du module SYSPHX à l'aide de deux organigrammes. Le premier, simplifié à l'extrême, donne déjà une idée de la stratégie utilisée sans faire appel aux noms de zones et de modules.
Le second, sans être totalement complet, rentre déjà dans le détail de l'utilisation des tables et des modules de gestion. Ces deux organigrammes ont pour but de préparer à la lecture du listing de SYSPHX qui contient évidemment tous les renseignements supplémentaires.

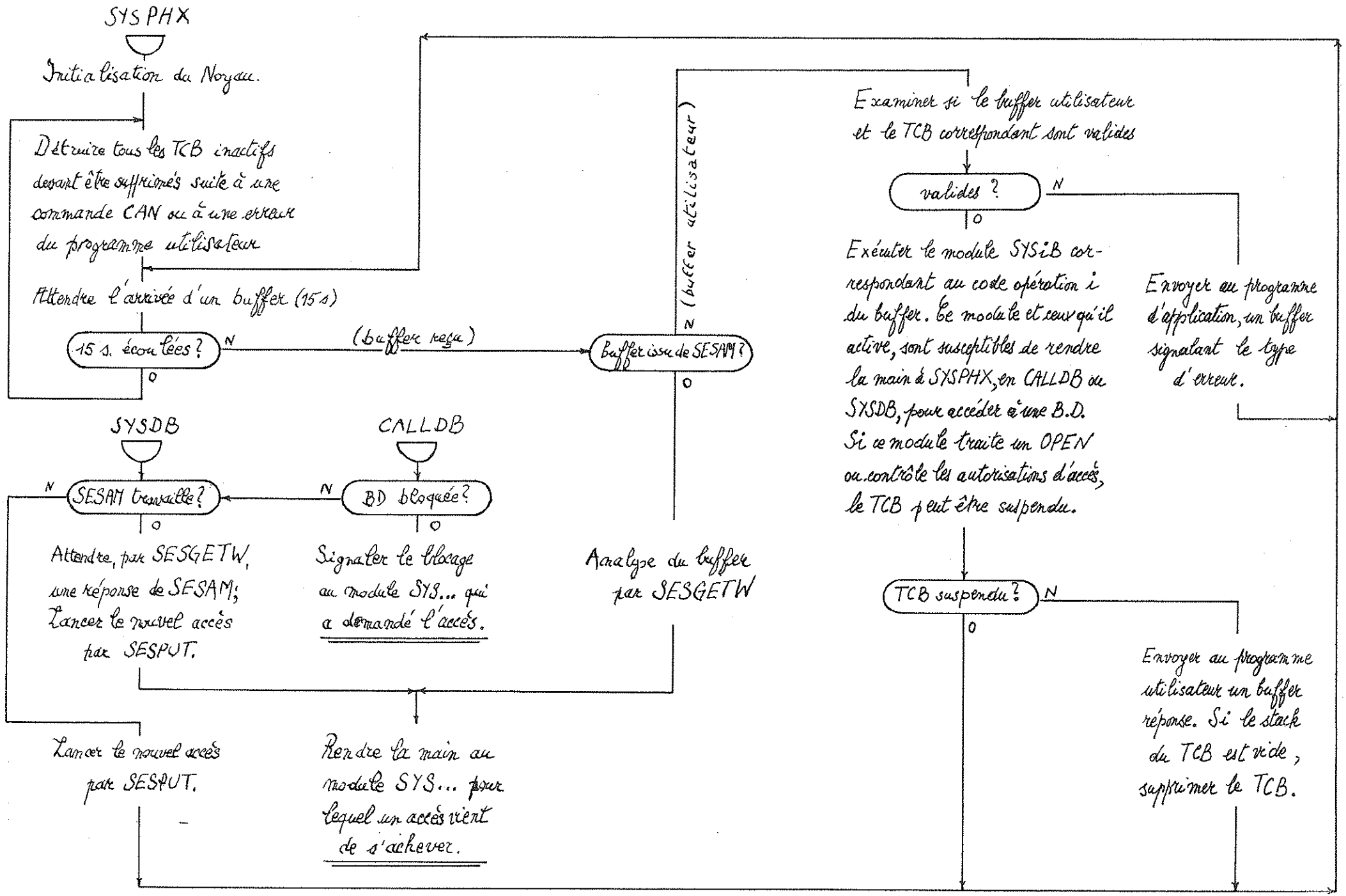


Figure 3.3.1.1. - Organigramme très simplifié de SYSPHX

Remarques

Malgré son caractère approximatif, l'organigramme précédent dégage bien la stratégie adoptée dans l'utilisation de SESAM : lorsque SESAM ne travaille pas (pour le noyau), celui-ci attend l'arrivée d'un buffer utilisateur susceptible de déclencher un accès SESAM. Cet accès lancé, le noyau profite du temps d'attente pour traiter un autre buffer, jusqu'à ce que ce traitement nécessite lui aussi un accès SESAM. A ce moment, on attend la fin de l'accès en cours et on relance immédiatement le nouvel accès avant de reprendre le traitement du premier utilisateur.
Cette stratégie implique donc que le noyau ne traite en parallèle que 2 utilisateurs au plus, les autres utilisateurs actifs restant en attente dans la file du noyau. Pourtant, le traitement d'un buffer par le noyau étant généralement plus court qu'un accès SESAM, cette méthode risque bien de réaliser une utilisation de SESAM proche de l'optimum. Ce ne serait plus vrai si le noyau pouvait lancer plusieurs accès SESAM en parallèle auquel cas une stratégie prévoyant le traitement simultané d'un nombre quelconque de TCB devrait être établie.
Mais celle-ci ne demanderait vraisemblablement pas de modifier les tables du noyau.

Les points d'entrée CALLDB et SYSDB ne sont pas à considérer comme des points d'entrée de sous-programmes au sens habituel du terme, ils définissent les endroits du superviseur où un module rend la main au superviseur.
Celui-ci interrompt alors l'exécution du module et déclenche de nouvelles actions qui ne sont pas forcément celles demandées par le module interrompu.
Celles-ci seront bien sûr réalisées mais lorsque le superviseur le jugera bon.
Puis, la main sera rendue au module appelant. CALLDB est le point d'entrée en cas d'exécution normale. SYSDB permet d'accéder quand même à une base de données si elle est bloquée (du moins, si SESAM le permet).

Figure 3.3.1.2. - Organigramme simplifié de SYSPHX

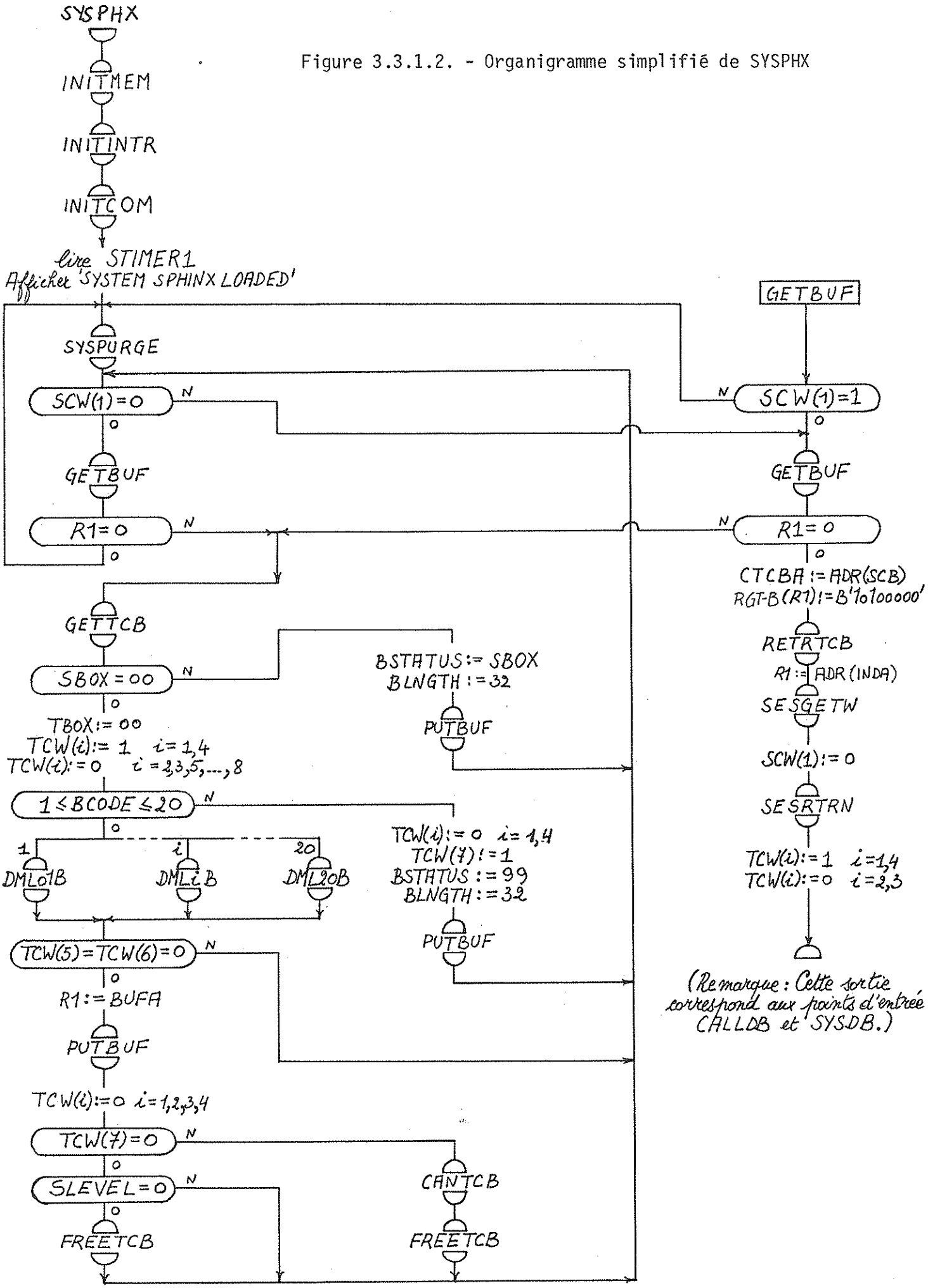
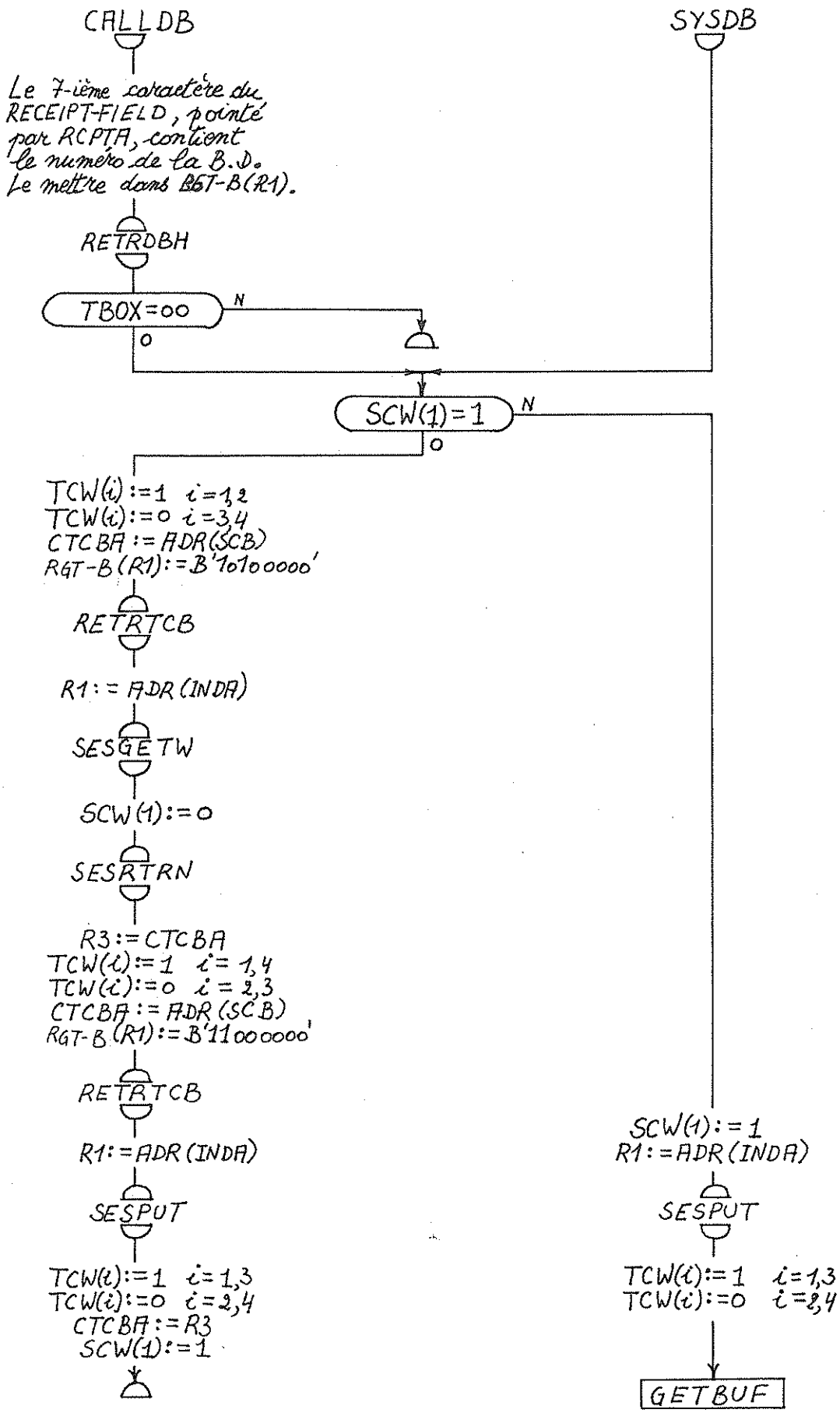
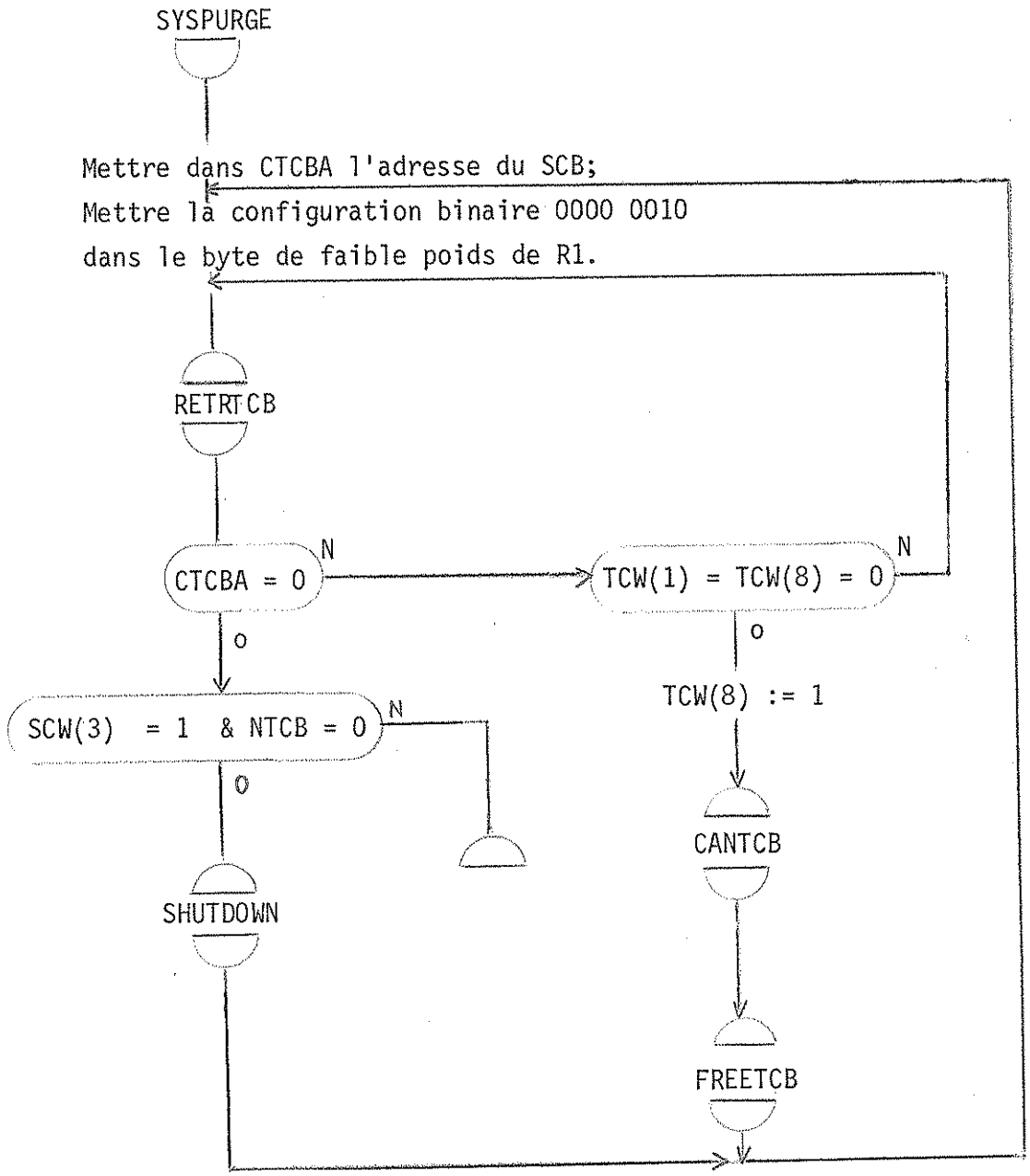


Figure 3.3.1.3. - Organigramme simplifié de SYSPHX (suite)



Les deux sous-programmes SYSPURGE et SESRTRN mentionnés dans l'organigramme sont, en fait, deux sections internes à SYSPHX. Nous décrirons SYSPURGE par un organigramme.



SESRTRN a pour effet de transformer le code retour SESAM en un code retour significatif pour les modules du noyau. Ce code est contenu dans le receipt-field. On trouvera la table de traduction à la page SPH.56.

Remarques

- Rappelons que des expressions telles que TCW(i), TBOX, INDA, SLEVEL ... désignent les zones correspondantes du TCB courant, c'est-à-dire celui dont l'adresse est contenue dans CTCBA; de même BCODE, BSTATUS, BLNGTH sont des zones du buffer associé au TCB courant.
  (Du moins si celui-ci existe. Dans le cas contraire, il s'agit du dernier buffer extrait de la file).

- R1, R3 désignent les registres 1 et 3.
- La notation GETBUF ⟶ permet de désigner par le nom GETBUF le point de l'organigramme situé à l'extrémité de la flèche. Il ne doit pas être confondu avec le point d'entrée GETBUF du module SYSITC.
- La notation GETBUF représente un branchement en ce point et connecte donc entre eux les deux morceaux d'organigramme.
- L'expression ADR(α) où α est un nom de zone désigne l'adresse de cette zone.
- L'expression RGT-B(R1) désigne le byte de plus faible poids du registre R1 (byte de droite).
- La lettre B suivie de huit chiffres binaires entre quotes désigne cette configuration binaire (ex. : B'10100000').

```
SOURCE                                                                  +
                                                                        I
*-----------------------------------------------------------------------+
*              TABLE OF RETURN-CODES                                    +
*-----------------------------------------------------------------------+
SIZETAB  DC    F'156'                  |SIZE OF TABLE.

TABCODE  DS    0F
         DC    C'0000'     OK.
         DC    C'101C'     NOT FOUND.
         DC    C'2099'     INDICATION-FIELD DESTROYED.
         DC    C'6099'     INDICATION-FIELD DESTROYED.
         DC    C'6171'     RESPONSE TRUNCATED.
         DC    C'6278'     DATA BASE MUST BE LOCKED.
         DC    C'6375'     ASPECT-NAME UNKNOWED.
         DC    C'6474'     TOO MANY ASPECTS.
         DC    C'6573'     SORT FIELD TOO LONG.
         DC    C'6672'     TOO MANY ASPECTS.
         DC    C'6772'     QUESTION-FIELD TOO LONG.
         DC    C'6W10'     NO RESPONSE.
         DC    C'6269'     PASSWORD ERROR.
         DC    C'7099'     INDICATION-FIELD DESTROYED.
         DC    C'7269'     PASSWORD ERROR.
         DC    C'9199'     INDICATION-FIELD DESTROYED.
         DC    C'9278'     DATA BASE MUST BE LOCKED.
         DC    C'9375'     ASPECT-NAME UNKNOWN.
         DC    C'9474'     TOO MANY ASPECTS.
         DC    C'9510'     RECORD DOES NOT EXIST.
         DC    C'9649'     RECORD ALREADY EXIST.
         DC    C'9772'     QUESTION-FIELD TOO LONG.
         DC    C'9269'     PASSWORD ERROR.
         DC    C'8099'     INDICATION-FIELD DESTROYED.
         DC    C'8269'     PASSWORD ERROR.
         DC    C'1999'     INDICATION-FIELD DESTROYED.
         DC    C'1269'     PASSWORD ERROR.
         DC    C'81$1'     SESPUT AFTER SESPUT.
         DC    C'82$2'     SESGET WITHOUT SESPUT.
         DC    C'83$3'     SESGET WITHOUT SESAM-FLAG.
         DC    C'84$4'     SESGET MUST BE ISSULD AGAIN.
         DC    C'85$5'     NO SPACE FOR THIS COMMUNICATION.
         DC    C'8V$V'     IND-FLD OR QUES-FLD NOT A V-TYPE RECORD.
         DC    C'9099'     INVALID OP-CODE.
         DC    C'98#8'     DATA BASE WITHOUT DB-KEY.
         DC    C'99#9'     ERROR IN DATA BASE HANDLER.
         DC    C'9U#U'     USER-ID NOT DECLARED.
         DC    C'9S#S'     RECORD IS LOCKED.
         DC    C'9T#T'     RECORD IS NOT IN PROTECTION TABLE.
```

3.3.2. SYS2oB,..., SYS2oG - Modules de contrôle des autorisations d'accès

Lorsqu'un programme DML entre dans un bloc REACH, ALLOCATE ou PREPARE de plus bas niveau, c'est-à-dire non inclus dans un bloc du même type, il envoie, tout d'abord, au noyau un buffer de demande d'autorisation d'accès. Celui-ci a pour données spécifiques l'équivalent du contenu d'un bloc de demandes d'accès (cfr. § 2.2.5. page). Le code de ce buffer est 20.
Il est reçu et traité par SYSPHX comme un buffer normal relatif à un ordre DML, c'est pourquoi les modules qui le traitent ont des noms construits sur le même canevas que les modules spécifiques à chaque ordre DML. Mais en réalité, le rôle de ces modules est tout autre : distribuer au mieux entre les différentes tâches DML les ressources que constituent les autorisations d'accès aux différents objets des différentes bases de données.

Le buffer de demande d'autorisations contient donc la liste des objets susceptibles d'être accédés en consultation (REACH, ALLOCATE) ou en modification (ATTACH, CREATE, DETACH, MODIFY, SUPPRESS, TRANSFER) à l'intérieur du bloc de plus bas niveau dans lequel le programme DML va pénétrer. Son analyse est confiée au point d'entrée DML20B du module SYS20B. Celui-ci recopie le contenu du buffer dans bloc de demande d'accès dont l'adresse est placée dans la zone REQSTA du TCB. Ensuite, le bloc REQST est comparé avec les blocs ACA de chaque base de données concernée. S'il ne demande à modifier aucun objet accessible par une autre tâche et s'il ne demande à lire aucun objet modifiable par une autre tâche, les autorisations d'accès sont accordées à la tâche, c'est-à-dire que :

- tous les blocs ACA concernés sont mis à jour comme suit : (cf. § 2.3.4)
  $ACA_i := ACA_i+1$ si l'objet i est accédé en consultation,
  $ACA_i := 255$ si l'objet i est accédé en modification.
- un buffer "OK" est renvoyé à la tâche DML

Si par contre, le programme demande de pouvoir accéder en consultation à un objet auquel une autre tâche a été autorisée d'accéder en modification, ou en modification à un objet auquel une autre tâche a été autorisée d'accéder en consultation ou en modification, aucune autorisation d' accès n'est accordée, le buffer réponse n'est pas renvoyé à la tâche utilisateur qui est suspendue (TCW(6) := 1).

Lorsqu'un programme sort d'un bloc ALLOCATE, PREPARE ou REACH de plus bas niveau (ordre END), le module du noyau chargé du traitement du END active le point d'entrée DML20C de SYS20B pour abandonner les autorisations d'accès qu'il possède : tous les blocs ACA concernés sont mis à jour.
- $ACA_i := ACA_i-1$ si la tâche était autorisée à consulter l'objet i
- ACA := 0 si la tâche était autorisée à modifier l'objet i.

Ensuite, le module SYS20B va examiner l'ensemble des blocs REQST des TCB suspendus (TCW(6) = 1) et en réactiver un certain nombre, si c'est possible.
La réactivation d'un TCB comporte les opérations suivantes :

- Attribution des autorisations d'accès avec mise à jour des ACA.
- Envoi d'un buffer à la tâche.
- TCW(6) := TCW(1) := 0.

La méthode de choix des tâches à réactiver comporte deux phases :

1. Un premier algorithme parcourt la chaîne des TCB suspendus, triée par ordre chronologique d'arrivée. Tant que les TCB rencontrés sont suspendus depuis un temps supérieur à τ (paramètre du système), ils sont réactivés si leur REQST est compatible avec les ACA.

2. Le second algorithme envisage toutes les combinaisons de TCB compatibles entre eux et avec les ACA, suspendus depuis un temps inférieur ou égal à τ.
   Il choisit parmi ces combinaisons celles qui comportent le plus grand nombre de TCB et parmi ces dernières celle qui est la plus petite pour la relation d'ordre définie ci-dessous.
   Désignons par $TCB_1^i \ldots TCB_n^i$ chaque combinaison, triée comme la chaîne des TCB et posons :
   $TCB_1^i \ldots TCB_n^i < TCB_1^j \ldots TCB_n^j$ ssi $\exists m < n$ tel que $TCB_1^i = TCB_1^j, \ldots, TCB_m^i = TCB_m^j$
   et TCB_{m+1}^i est avant TCB_{m+1}^j dans la chaîne des TCB.
   On voit que l'idée de la méthode est de réactiver le plus grand nombre possible de TCB tout en favorisant les plus anciens. De plus, les TCB d'un âge supérieur à τ reçoivent une priorité absolue.

3.3.3. SYSINT - Module de contrôle de l'état du noyau

Ce module permet d'obtenir divers renseignements concernant l'éxécution du noyau et des tâches qui y sont connectées. Son fonctionnement repose sur un outil fourni par le BS2000 : la macro STXIT. Sans décrire toutes les possibilités de celle-ci, disons qu'elle permet de définir dans un programme un point d'entrée auquel le BS2000 donnera le contrôle lorsque :

- on aura interrompu l'exécution du noyau en poussant sur la touche 'escape' (T200) ou en envoyant un caractère 'ETX' (vidéo);
- on aura introduit la commande /INTR.

Le retour à l'exécution normale sera assuré par la macro EXIT CR.

Le module SYSINT possède le point d'entrée INITINTR, appelé une seule fois lors de l'initialisation du noyau pour définir le point d'entrée INTR par la macro STXIT. Par la suite, il pourra être activé par la méthode décrite ci-dessus et invitera l'utilisateur à entrer une commande.
La description complète de ces commandes est donnée ailleurs (cfr. PRO, §5).
Nous donnons ici quelques indications sur leur traitement.
La commande DBNAME fournit la liste des bases de données connues du noyau et leur état : ouverte, fermée, bloquée (DBSTATUS = 0,1,2) par un parcours de la table des adresses des DBH contenue dans le SCB ($DBHA_i$).

Les commandes TSN ACT, TSN SUS, TSN ALL fournissent des renseignements sur les tâches connues du noyau, par un parcours de la chaîne des TCB.
Mais ce parcours ne peut se faire sans danger que si la chaîne des TCB est dans un état cohérent, c'est-à-dire si SCW(2) = 0. Dans le cas contraire, l'exécution de la commande sera refusée.

La commande CAN <n° de tâche> permet d'éliminer une tâche du système en positionnant à 1 le bit TCW(7) de son TCB. De plus, la remarque faite pour les commandes TSN s'applique ici également.

La commande SHUTDOWN met le noyau dans l'état 'arrêt du système' en positionnant SCW(3) à 1. L'arrêt du noyau n'interviendra toutefois que lorsque plus aucune tâche ne sera connectée au système.

La commande KILL génère un DUMP et l'arrêt brutal du noyau dans le cas d'une exécution jugée anormale.

La commande USER fournit le nombre d'utilisateurs connectés (NTCB).

3.3.4. SYSDWN - Arrêt du noyau SPHINX

Le module SYSDWN possède le point d'entrée SHUTDOWN activé lorsque le noyau est dans l'état 'arrêt du système' et qu'aucun TCB ne demeure plus dans le système (NTCB = 0). Il imprime l'heure de terminaison, la liste et l'état des bases de données ayant été exploitées, clôture la communication (STOPCOM) et s'arrête.

3.4. Modules spécifiques à chaque ordre DML

Nous ne parlerons pas des modules spécialisés de traitement de chaque ordre DML. Les listings de ces différents modules contiennent une documentation complète.

A titre d'information, voici le nom et la fonction de chacun de ces modules.

SYS01B module principal de traitement de l'ordre OPEN.

SYS01X allocation d'un numéro d'utilisateur.

SYS01Y lecture des schémas internes d'une base de données.

SYS02Y libération d'un numéro d'utilisateur.

SYS03B module de traitement de l'ordre REACH.

SYS06B module de traitement de l'ordre END.

SYS07B module de traitement des ordres ALLOCATE et PREPARE.

SYS09B module principal de traitement de l'ordre CREATE.

SYS10B module de traitement de l'ordre MODIFY.

SYS11B module principal de traitement des ordres ATTACH et DETACH.

SYS11C création ou suppression d'une occurrence de relation entre objets complexes.

SYS13B module principal de traitement du TRANSFER.

SYS13C traitement des erreurs pour l'ordre TRANSFER.

SYS13D tri d'une liste de cibles sur le numéro d'enregistrement (TRANSFER).

SYS14B module principal de traitement de l'ordre SUPPRESS.

SYS14C suppression d'une réalisation d'objet complexe.

SYS14X suppression d'un pointeur dans un bloc de pointeurs.

SYS14Y recherche d'un pointeur dans un bloc de pointeurs.

L'INTERPRÉTEUR NUL

1. INTRODUCTION

L'implémentation du langage NUL (Navigational User Language) est réalisée par un interpréteur. Pour chaque utilisateur NUL travaillant au terminal, il existe en mémoire virtuelle une copie de l'interpréteur constituant une tâche séparée, communiquant avec le noyau SPHINX.
Ce dernier traite de la même façon les buffers reçu de l'interpréteur NUL et ceux reçus des programmes DML via l'interface SYSDML.

L'interpréteur NUL dirige le dialogue avec l'utilisateur et vérifie la cohérence des requêtes introduites par celui-ci. Ses principales fonctions sont :

- analyse du contexte introduit par l'utilisateur et représentation de ce contexte sous forme interne.

- analyse des commandes PRINT et traduction de celles-ci sous forme d'un code intermédiaire, contenant des références à la représentation interne du contexte.
- interprétation de ce code intermédiaire, ce qui provoque l'échange de buffers de communication avec le noyau SPHINX et l'impression des résultats obtenus de celui-ci.

Nous décrivons, dans les pages suivantes, les tables et les modules réalisant ces fonctions. La compréhension de ces pages nécessite une connaissance approfondie du langage NUL et de sa terminologie tels qu'ils ont été exposés dans les brochures : "Modèles et Langages" et "Manuels de référence des langages".
En particulier, précisons que nous utiliserons le terme <u>définition</u> pour désigner une partie du contexte commençant par $S_i \leftarrow$ et se terminant par un point vigule, le terme <u>qualificatif</u> pour désigner une partie commençant par $X_i \leftarrow$ et se terminant par un point virgule et le terme <u>phrase</u> pour désigner l'une ou l'autre de ces deux constructions.

2. LES TABLES DE L'INTERPRETEUR NUL

2.1. NULCB - Le bloc de contrôle du système NUL

La figure 2.1 présente la découpe du bloc NULCB de contrôle de l'interpréteur NUL. Ce bloc contient un ensemble de renseignements fondamentaux sur l'état de l'exécution à un moment déterminé.
Nous détaillons ci-dessous la nature de ces informations.

TSN contient le numéro de la tâche BS2000 dans laquelle s'exécute l'interpréteur (4 caractères numériques).

LNT est un byte dont le contenu est 0 (binaire) si la tâche est non conversationnelle et égal à un code définissant le type du terminal si la tâche est conversationnelle. Ce renseignement est exploité pour déterminer le milieu de sortie des résultats :
LNT = 0 : imprimante
LNT ≠ 0 : imprimante et terminal (option standard)

ITIME contient l'heure de chargement de l'interpréteur NUL en six caractères numériques sous la forme Heure/Minutes/Secondes.

NULSTW Le "mot d'état" de l'interpréteur NUL se décompose en quatre bytes NULST1, NULST2, NULST3, NULST4 dont certains bits possèdent une signification précise.

Désignons par NULSTi(j) (i = 1, ..., 4; j = 1, ..., 8) le jème bit de NULSTi (compté à partir des poids les + forts), on a :

NULST1 (1) = 1 si une base de donnée est ouverte,
= 2 sinon.

NULST1 (2) = 1 entre le moment ou un contexte a été clôturé par une commande END et la prochaine entrée en mode contexte.
= 0 durant l'introduction d'un contexte.

NULST1 (3) = 1 si le schéma interne est ouvert.
= 0 sinon.

NULST4 (1) = 1 si le milieu de sortie des résultats est l'imprimante <u>et</u> le terminal.
= 0 si le milieu de sortie est l'imprimante.

NULST4 (8) = 1 si l'utilisateur a interrompu le système NUL et introduit la commande spéciale 'CANCEL = PRINT' pour arrêter une impression de résultats en cours.
= 0 sinon. Le bit NULST4 (8) est repositionné à 0 dès que l'impression en cours a été stoppée.

La signification des bits constitutifs de NULST3 varie selon le module où ils sont utilisés.

Lors de l'analyse syntaxique d'un contexte (module NULSTX), ils sont utilisés pour déterminer l'endroit où il faut reprendre l'exécution, en cas d'erreur (après modification de la phrase par l'utilisateur) :

NULST3(1) = 1 : reprendre l'exécution au traitement de la clause "FOR S_i".

NULST3(2) = 1 : reprendre la traduction d'un qualificatif "X_i ←..." au début.

NULST3(3) = 1 : reprendre l'exécution au traitement de l'opérateur "SUCH THAT".

NULST3(4) = 1 : reprendre l'exécution au traitement des séparateurs (";" ou "&" ou "AND").

NULST3(8) = 1 : reprendre l'exécution au début du traitement d'une phrase.

Un seul de ces bits peut valoir 1, au même moment.
Dans les modules de construction d'un buffer (NULCMD) et lors de la construction d'une table d'édition (NULP3S), le bit NULST3(8) joue un rôle très particulier exposé dans la documentation contenue dans les listings de ces modules.

FFAREA, LFAREA pointent, respectivement, vers le premier et le dernier bloc de mémoire dynamique, libres(voir § 3.6. : NULMEM).

FAREA, LAREA pointent, respectivement, vers le premier et le dernier bloc de mémoire dynamique, réservés pour la constitution de buffers de communication.

RCODE contient le code de retour renvoyé par tout module de l'interpréteur NUL.

DBNAME contient, cadré à gauche et suivi de blancs, le nom de la base de données courante, si elle existe (NULST1(1) = NULST1(3) = 1).

DBN contient, sous les mêmes conditions, le numéro i de la base de données courante (i ∈ {A,B,...,Z,1,...,6 }).

LINE est un demi mot binaire contenant la longueur d'une ligne d'impression (à l'imprimante ou au terminal).

SMAX, XMAX sont des constantes, égales aux valeurs maximales permises pour STOP et XTOP (resp.).

STOP contient l'indice, dans la zone CONTROL, de la dernière entrée CTXE construite pour décrire une définition de contexte (S_i ← ...).
(On a : 0 ⩽ STOP ⩽ SMAX).

XTOP contient l'indice, dans la zone CONTROL,de la dernière entrée CTXE construite pour décrire un qualificatif (X_i ← ...).
(On a : SMAX ⩽ STOP ⩽ XMAX)

La signification des bits constitutifs de NULST3 varie selon le module où ils sont utilisés.

Lors de l'analyse syntaxique d'un contexte (module NULSTX), ils sont utilisés pour déterminer l'endroit où il faut reprendre l'exécution, en cas d'erreur (après modification de la phrase par l'utilisateur) :

NULST3(1) = 1 : reprendre l'exécution au traitement de la clause "FOR S_i".

NULST3(2) = 1 : reprendre la traduction d'un qualificatif "X_i ←..." au début.

NULST3(3) = 1 : reprendre l'exécution au traitement de l'opérateur "SUCH THAT".

NULST3(4) = 1 : reprendre l'exécution au traitement des séparateurs (";" ou "&" ou "AND").

NULST3(8) = 1 : reprendre l'exécution au début du traitement d'une phrase.

Un seul de ces bits peut valoir 1, au même moment.
Dans les modules de construction d'un buffer (NULCMD) et lors de la construction d'une table d'édition (NULP3S), le bit NULST3(8) joue un rôle très particulier exposé dans la documentation contenue dans les listings de ces modules.

FFAREA, LFAREA pointent, respectivement, vers le premier et le dernier bloc de mémoire dynamique, libres(voir § 3.6. : NULMEM).

FAREA, LAREA pointent, respectivement, vers le premier et le dernier bloc de mémoire dynamique, réservés pour la constitution de buffers de communication.

RCODE contient le code de retour renvoyé par tout module de l'interpréteur NUL.

DBNAME contient, cadré à gauche et suivi de blancs, le nom de la base de données courante, si elle existe (NULST1(1) = NULST1(3) = 1).

DBN contient, sous les mêmes conditions, le numéro i de la base de données courante (i ∈ {A,B,...,Z,1,...,6 }).

LINE est un demi mot binaire contenant la longueur d'une ligne d'impression (à l'imprimante ou au terminal).

SMAX, XMAX sont des constantes, égales aux valeurs maximales permises pour STOP et XTOP (resp.).

STOP contient l'indice, dans la zone CONTROL, de la dernière entrée CTXE construite pour décrire une définition de contexte (S_i ← ...).
(On a : 0 ⩽ STOP ⩽ SMAX).

XTOP contient l'indice, dans la zone CONTROL,de la dernière entrée CTXE construite pour décrire un qualificatif (X_i ← ...).
(On a : SMAX ⩽ STOP ⩽ XMAX)

CRNT,XCRNT sont utilisés lors du remplissage des entrées CTXE de description des phrases de contexte.

CRNT pointe sur l'entrée courante en cours de remplissage.

XCRNT pointe sur la dernière entrée traitée correspondant à un qualificatif.

TMNT M SOURCE

```
1292                #NULCB
1293 1 NULCB        DSECT
1294 1              #NULCT
1295 2 *-------------------------------------------------------------+
1296 2 *                N.U.L. CONTROL BLOCK.                        I
1297 2 *-------------------------------------------------------------+
1298 2 *
1299 2 *
1300 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1301 2 *  |                 |L ///|                 |               |
1302 2 *  |     T S N       |N ///|     I T I M E   |    NULSTW     |
1303 2 *  |                 |T ///|                 |               |
1304 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1305 2 *  |                 |                 |                 |               |
1306 2 *  |01 00 00 02|  FFAREA   |   LFAREA  |01 00 00 02|
1307 2 *  |                 |                 |                 |               |
1308 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1309 2 *  |                 |                 |                 |       |///////|
1310 2 *  |01 00 00 00|   FAREA   |   LAREA   |RCODE|///////|
1311 2 *  |                 |                 |                 |       |///////|
1312 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1313 2 *  |                                                             |
1314 2 *  |                   D B N A M E                               |
1315 2 *  |                                                             |
1316 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1317 2 *  |D I///|       |///////////////////////////////////////////|
1318 2 *  |B I///|LINE   |///////////////////////////////////////////|
1319 2 *  |N I///|       |///////////////////////////////////////////|
1320 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1321 2 *  |       |       |       |       |       |       |       |       |///////|
1322 2 *  |SMAX   |XMAX   |STOP   |XTOP   |SROOT  |CRNT   |XCRNT  |///////|
1323 2 *  |       |       |       |       |       |       |       |       |///////|
1324 2 *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1325 2 *
1326 2 *
1327 2 *  (1)  TSN      TASK SEQUENCE NUMBER.
1328 2 *  (2)  LNT      LINE TYPE (TERMINAL).
1329 2 *  (3)  ITIME    INIT TIME.
1330 2 *  (4)  NULSTW   N.U.L. STATUS WORD.
1331 2 *  (5)  MCA      MEMORY CONTROL AREA.
1332 2 *  (6)  FFAREA   ADDRESS OF FIRST FREE AREA.
1333 2 *  (7)  LFAREA   ADDRESS OF LAST  FREE AREA.
1334 2 *  (8)  HAREA    HEADER OF RESERVED AREA CHAIN.
1335 2 *  (9)  FAREA    ADDRESS OF FIRST RESERVED AREA.
1336 2 *  (10) LAREA    ADDRESS OF LAST  RESERVED AREA.
1337 2 *  (11) RCODE    GENERAL RETURN CODE.
1338 2 *  (12) DBNAME   CURRENT DATA BASE NAME.
1339 2 *  (13) DBN      CURRENT DATA BASE NUMBER.
1340 2 *  (14) LINE     LENGTH OF OUTPUT LINE.
```

Figure 2.1.

```
TMNT M SOURCE
                                CONTEXT CONTROL AREA.
                                - - - - - - - - - - -
1341 2 *
1342 2 *
1343 2 *
1344 2 * (15) SMAX      MAX. INDEX  OF S-LABEL.
1345 2 * (16) XMAX      MAX. INDEX  OF X-LABEL.
1346 2 * (17) STOP      TOP  INDEX  OF S-LABEL.
1347 2 * (18) XTOP      TOP  INDEX  OF X-LABEL.
1348 2 * (19) SROOT     INDEX OF CURRENT ROOT.
1349 2 * (20) CRNT      INDEX OF CURRENT S-LABEL OR X-LABEL.
1350 2 * (21) XCRNT     INDEX OF CURRENT X-LABEL.
1351 2 *
1352 2 TSN        DS    CL4                    |TASK SEQ. NUMBER.
1353 2 LNT        DS    CL1                    |LINE TYPE.
1354 2            DS    CL1                    |RESERVED
1355 2 ITIME      DS    CL6                    |INIT TIME.
1356 2 NULSTW     DS    OF                     |N.U.L. STATUS WORD.
1357 2 NULST1     DS    CL1                    |N.U.L. STATUS 1.
1358 2 NULST2     DS    CL1                    |N.U.L. STATUS 2.
1359 2 NULST3     DS    CL1                    |N.U.L. STATUS 3.
1360 2 NULST4     DS    CL1                    |N.U.L. STATUS 4.
1361 2 MCA        DS    F                      |MEMORY CONTROL AREA.
1362 2 FFAREA     DS    F                      |ADDR. OF FIRST FREE AREA.
1363 2 LFAREA     DS    F                      |ADDR. OF LAST  FREE AREA.
1364 2            DS    F                      |RESERVED.
1365 2 HAREA      DS    F                      |HEADER OF RESERVED AREA CHAIN.
1366 2 FAREA      DS    F                      |ADDR. OF FIRST RESERVED AREA.
1367 2 LAREA      DS    F                      |ADDR. OF LAST  RESERVED AREA.
1368 2 RCODE      DS    CL2                    |GENERAL RETURN CODE.
1369 2            DS    CL2                    |RESERVED.
1370 2 DBNAME     DS    CL16                   |CURRENT DATA BASE NAME.
1371 2 DBN        DS    CL1                    |CURRENT DATA BASE NUMBER.
1372 2            DS    CL1                    |RESERVED.
1373 2 LINE       DS    H                      |LENGTH OF DISPLAY.
1374 2            DS    CL12                   |RESERVED.
1375 2 SMAX       DS    H                      |MAX. INDEX OF S-LABEL.
1376 2 XMAX       DS    H                      |MAX. INDEX OF X-LABEL.
1377 2 STOP       DS    H                      |TOP INDEX OF S-LABEL.
1378 2 XTOP       DS    H                      |TOP INDEX OF X-LABEL.
1379 2 SROOT      DS    H                      |INDEX OF CURRENT ROOT.
1380 2 CRNT       DS    H                      |INDEX OF CURRENT LABEL.
1381 2 XCRNT      DS    H                      |INDEX OF CURRENT X-LABEL.
1382 2            DS    H                      |RESERVED.
1383 1 NULDOC     CSECT
```

Figure 2.1. (suite)

2.2. Blocs utilisés pour la description d'un contexte

Les différents blocs utilisés pour décrire un contexte sont répartis dans des zones de mémoire d'adresses et de longueurs fixes.
La longueur, limitée a priori, de ces zones, impose des contraintes sur le nombre de définitions, de qualificatifs, de valeurs présents dansun contexte.
L'optique utilisée dans l'implémentation de l'interpréteur NUL est donc fort différente de celle choisie pour le noyau SPHINX où l'allocation de mémoire est entièrement dynamique.

Donnons les noms des zones statiques réservées pour les blocs de description du contexte.

CONTROL : zone contenant au maximum XMAX entrées CTXE de description d'une définition ou d'un qualificatif. Les entrées 1 à SMAX sont réservées pour les définitions, les entrées SMAX+1 à XMAX, pour les qualificatifs.

ICODEO : zone contenant le bloc CTXINPUT de description de la dernière phrase du contexte entrée au terminal.

ICODE1 : zone contenant les blocs EQE de description des critères élémentaires de la dernière phrase du contexte.

ICODE2 : zone contenant les blocs QFE de description des critères complexes de la dernière phrase du contexte.

VALAREA : zone contenant les valeurs d'objets élémentaires figurant dans les critères élémentaires de la dernière phrase du contexte.
Les 32 premiers caractères de cette zone contiendront la valeur de l'aspect d'ordre SESAM.

Nous pouvons maintenant donner une description des différents blocs précités.

2.2.1. CTXE - description d'une phrase du contexte

Chacun de ces blocs contient les principaux renseignements relatifs à une phrase du contexte. Ils sont contenus dans la zone CONTROL.

La figure 2.2.1. présente la découpe d'un bloc CTXE.

SXPRNT contient l'indice, dans la zone CONTROL, du bloc CTXE dont dépend directement le bloc courant (PARENT). Si le contexte contient les phrases $S_i \leftarrow \ldots = X_j \ldots$;
$X_j \leftarrow \ldots$;
$S_k \leftarrow$ FOR $S_i \ldots$;

S_i est parent de X_j et S_k, X_j est X-enfant de S_i.
S_k est S-enfant de S_i.

SCHILD vaut 0 si le bloc ne possède pas (ou pas encore) de bloc CTXE S-enfant et contient, sinon, l'indice dans CONTROL du premier de ces blocs (dans l'ordre d'introduction des définitions au terminal).
En particulier, SCHILD = 0 lorsque le bloc courant décrit un qualificatif (X-entry).

XCHILD vaut 0 si le bloc ne possède pas de critère complexe et donc pas de X-enfant, sinon il contient l'indice, dans CONTROL, du premier bloc CTXE décrivant le qualificatif associé au premier critère complexe.

SXNEXT vaut 0 si le bloc représente la racine de la dernière hiérarchie du contexte ou le dernier enfant d'un certain parent. Sinon, il contient l'index de la racine suivante ou de l'enfant suivant (du même type).

LABEL contient les trois caractères de l'étiquette de la phrase du contexte décrite par le bloc.
(LABEL = Xi ou Si $i \in \{00,\ldots,16\}$).

TYPE ce byte donne divers renseignements sur le type de la phrase décrite par le bloc CTXE.
Désignons par TYPE(i) (i = 1,...,8) les différents bits de TYPE, numérotés de gauche à droite.

TYPE(1) = 1 ssi la phrase est une définition de contexte.
TYPE(2) = 1 ssi la phrase est un qualificatif.
TYPE(3) = 1 ssi le bloc décrit un qualificatif vide (DUMMY) c'est-à-dire un critère complexe dans lequel ne figure pas "X_i". Dans ce cas, LABEL = X00.

D'autres bits de TYPE fournissent des renseignements relatifs à une commande PRINT introduite pour le contexte courant.
Ces renseignements sont exploités lors de l'exécution de la commande PRINT (interprétation du code intermédiaire).

TYPE(6) = 0 lorsque, à partir d'une réalisation de l'objet complexe du bloc parent, on veut effectuer un accès à une première réalisation de de l'objet complexe du bloc courant (OBJN) via la relation du bloc courant (RELN)

= 1 lorsqu'on veut effectuer un accès à une réalisation "suivante".

TYPE(8) = 1 ssi des valeurs d'objets élémentaires associés à la réalisation ainsi atteinte seront imprimées.
C'est le cas d'une définition de contexte dont l'étiquette Si a été spécifiée, au moins une fois, dans la commande PRINT.

TYPE(7) = 1 ssi la réalisation atteinte ne peut faire l'objet d'aucune impression. C'est le cas des qualificatifs et des définitions de contexte dont l'étiquette ne figure pas dans la commande PRINT.

OBJN contient sous forme binaire le numéro de l'objet complexe principal de la phrase NUL. Dans le cas d'une définition de contexte, c'est l'objet complexe dont le nom figure immédiatement après la clause FOR S_i (éventuellement vide). Dans le cas d'un qualificatif, c'est l'objet du critère complexe correspondant.

RELN contient sous forme binaire le numéro de la relation d'accès principale de la phrase NUL. Dans le cas d'une définition de contexte, c'est la relation dont le nom figure dans la clause FOR S_i [BY <relation>].
Dans le cas d'un qualificatif, c'est la relation spécifiée dans le critère complexe. Si le bloc courant est racine d'une hiérarchie du contexte, RELN = 0.

LEVEL contient, en binaire, le niveau du bloc dans la hiérarchie du contexte. Si le bloc correspond à la racine, LEVEL = 2, s'il est enfant d'un bloc de niveau i, LEVEL = i + 1.

CNTLG contient, en bytes, la longueur de la zone des compteurs dans une réalisation de l'objet complexe OBJN.

CCNT contient, lors de l'exécution d'une commande PRINT, associée à la hiérarchie d'impression, le nombre de réalisations de OBJN atteintes et retenues, à un moment donné, via la relation RELN, à partir de la réalisation courante du bloc parent. Ce compteur binaire est exploité, en conjonction avec les quantificateurs, dans les critères complexes.

ICNT contient l'indice du compteur d'occurrences de la relation RELN dans la zone des compteurs de l'objet complexe parent. Ce compteur permet dans

certains cas, de résoudre des critères complexes sans accéder à la base de données. (Notamment dans un critère sans qualificatif).

QT contient le code du quantificateur contenu dans le critère complexe associé au bloc courant (QT n'a donc un contenu significatif que lorsque le bloc CTXE décrit un qualificatif).

QVAL contient la valeur associée au quantificateur.
On a :

Quantificateur	QT	QVAL
AT LEAST n	1	n
AT MOST n	2	n
NO	0	0
ALL	3	0
SOME	1	1
(omission)	1	1

QT et QVAL sont codés en binaire.

TXTA pointe vers le premier bloc dynamique de texte source correspondant au bloc CTXE (si ce bloc existe, sinon TXTA = 0).

CMDA contient l'adresse du buffer de communication pour le noyau SPHINX, associé au bloc CTXE. Ce buffer aura pour effet de commander l'accès aux réalisations de OBJN via la relation RELN à partir de la réalisation courante du bloc parent.

RESPA contient l'adresse du bloc de mémoire dans lequel sera placée chaque réponse du noyau.

EDTA contient l'adresse du bloc d'en-tête (HEDT) de la table d'édition associée au bloc CTXE. Cette table n'existe que si le bloc décrit une définition de contexte dont l'étiquette figure dans la dernière commande PRINT. (TYPE(8) = 1). Si la table n'existe pas, EDTA = 0.

NAME contient le nom de l'objet complexe OBJN.

```
'MNT M SOURCE
                    #CTXE
1384                                                                  I
1385 1 *-------------------------------------------------------------+
1386 1 *                    CONTEXT ENTRY.                           |
1387 1 *-------------------------------------------------------------+
1388 1 *
1389 1 *
1390 1 *  +---+---+---+---+---+---+---+---+
1391 1 *  |   |   |   |   |   |   |   |   |
1392 1 *  | 1 | 2 | 3 | 4 | LABEL |6 |OBJN |RELN |
1393 1 *  |   |   |   |   |   |   |   |   |
1394 1 *  +---+---+---+---+---+---+---+---+
1395 1 *  |   |   |   |   |   |Q |///|   |
1396 1 *  |LEVEL| 10 |  CCNT  |ICNT |T |///|  QVAL  |
1397 1 *  |   |   |   |   |   |   |///|   |
1398 1 *  +---+---+---+---+---+---+---+---+
1399 1 *  |   |   |   |   |   |   |   |   |
1400 1 *  |  TXTA |  CMDA |  RESPA |  EDTA |
1401 1 *  |   |   |   |   |   |   |   |   |
1402 1 *  +---+---+---+---+---+---+---+---+
1403 1 *  |                                |
1404 1 *  |          N A M E               |
1405 1 *  |                                |
1406 1 *  +---+---+---+---+---+---+---+---+
1407 1 *
1408 1 *
1409 1 * (1)   SXPRNT     INDEX OF PARENT (S- OR X-LABEL).
1410 1 * (2)   SCHILD     INDEX OF FIRST S-CHILD (OR X'0000').
1411 1 * (3)   XCHILD     INDEX OF FIRST X-CHILD (OR X'0000').
1412 1 * (4)   SXNEXT     INDEX OF BROTHER (OR X'0000').
1413 1 * (5)   LABEL      LABEL OF ENTRY.
1414 1 * (6)   TYPE       TYPE  OF ENTRY.
1415 1 *                    2**7    S-LABEL.
1416 1 *                    2**6    X-LABEL.
1417 1 *                    2**5    DUMMY X-LABEL.
1418 1 *                    2**2    FIRST OR NEXT ACCESS.
1419 1 *                    2**1    ENTRY TO BE ACCESSED.
1420 1 *                    2**0    ENTRY TO BE DISPLAYED.
1421 1 * (7)   OBJN       OBJECT NUMBER.
1422 1 * (8)   RELN       RELATION NUMBER.
1423 1 * (9)   LEVEL      LEVEL OF ENTRY IN CONTEXT-HIERARCHY.
1424 1 * (10)  CNTLNG     LENGTH OF COUNTER AREA.
1425 1 * (11)  CCNT       CURRENT NUMBER OF OCCURRENCES.
1426 1 * (12)  ICNT       INDEX OF COUNTER IN ORIGIN.
1427 1 *                  (ONLY IF ACCESS VIA A RELATION).
1428 1 * (13)  QT         CODE  OF QUANTIFIER (X-ENTRY).
1429 1 * (14)  QVAL       VALUE OF QUANTIFIER (X-ENTRY).
1430 1 * (15)  TXTA       ADDRESS OF SOURCE TEXT.
1431 1 * (16)  CMDA       ADDRESS OF COMMAND.
1432 1 * (17)  RESPA      ADDRESS OF RESPONSE.
1433 1 * (18)  EDTA       ADDRESS OF EDIT TABLE.
1434 1 * (19)  NAME       NAME OF OBJECT.
1435 1 *
```

Figure 2.2.1.

```
TMNT M SOURCE

1436 1 *
1437 1 CTXE      DSECT
1438 1 SXPRNT    DS     H        |INDEX OF PARENT.
1439 1 SCHILD    DS     H        |INDEX OF S-CHILD.
1440 1 XCHILD    DS     H        |INDEX OF X-CHILD.
1441 1 SXNEXT    DS     H        |INDEX OF BROTHER.
1442 1 LABEL     DS     CL3      |LABEL OF ENTRY.
1443 1 TYPE      DS     CL1      |TYPE  OF ENTRY.
1444 1 OBJN      DS     H        |OBJECT NUMBER.
1445 1 RELN      DS     H        |RELATION NUMBER.
1446 1 LEVEL     DS     H        |LEVEL OF ENTRY.
1447 1 CNTLNG    DS     H        |LENGTH OF COUNTER AEA.
1448 1 CCNT      DS     F        |CURRENT NUMBER OF OCCURENCES.
1449 1 ICNT      DS     H        |INDEX OF COUNTER.
1450 1 QT        DS     CL1      |CODE OF  QUANTIFIER.
1451 1           DS     CL1      |RESERVED.
1452 1 QVAL      DS     F        |VALUE OF QUANTIFIER.
1453 1 TXTA      DS     F        |ADDR. OF SOURCE TEXT.
1454 1 CMDA      DS     F        |ADDR. OF COMMAND.
1455 1 RESPA     DS     F        |ADDR. OF RESPONSE.
1456 1 EDTA      DS     F        |ADDR. OF EDIT TABLE.
1457 1 NAME      DS     CL16     |NAME OF OBJECT.
1458 1 NULDOC    CSECT
```

Figure 2.2.1. (suite)

2.2.2. CTXINPUT - Bloc descriptif de la phrase courante du contexte

Ce bloc fixe est utilisé lors de l'analyse consécutive à l'introduction d'une nouvelle phrase du contexte. Il sera ensuite utilisé pour garnir une nouvelle entrée CTXE dans la table CONTROL de description du contexte. La figure 2.2.2. représente la découpe de CTXINPUT dont le contenu est détaillé ci-dessous.

MNEQ,MNQF contiennent respectivement le nombre maximum de critères élémentaires et complexes permis dans une même phrase. Ce sont des paramètres du système.

CNEQ,CNQF contiennent le nombre de critères élémentaires et complexes déjà analysés à un moment donné de l'analyse de la phrase courante. ($0 \leqslant CNEQ \leqslant MNEQ$, $0 \leqslant CNQF \leqslant MNQF$).

La plupart des zones qui suivent seront recopiées dans une entrée CTXE et ont donc la même signification que les zones correspondantes de CTXE.

CPRNT voir SXPRNT
CLABEL voir LABEL
CTYPE voir TYPE (i = 1,2,3)
COBJN voir OBJN
CRELN voir RELN
CLEVEL voir LEVEL
CCNTLNG voir CNTLNG
CCMDA voir CMDA

CFTXTA,CLTXTA pointent respectivement vers le premier et vers le dernier bloc de texte source reçu pour la phrase courante. Il est permis en effet d'introduire une telle phrase en plusieurs fois.
Plus tard, CFTXTA sera copié dans TXTA.

```
DR2  STMNT M SOURCE

1905                ENTRY ICODEO
1906                ENTRY ICODE1
1907                ENTRY ICODE2
1908                #ICODEO
1909 1 *-----------------------------------------------------------+
1910 1 *                CURRENT CONTEXT INPUT.                     I
1911 1 *-----------------------------------------------------------+
1912 1 *
1913 1 *
1914 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1915 1 *  |   |   |   |   |///////////////////////////////////////|
1916 1 *  |MNEQ |MNQF |CNEQ |CNQF |///////////////////////////////////////|
1917 1 *  |   |   |   |   |///////////////////////////////////////|
1918 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1919 1 *  |   |///////////////////////////|       |   |   |   |
1920 1 *  |CPRNT|///////////////////////////| CLABEL |7 |COBJN|CRELN|
1921 1 *  |   |///////////////////////////|       |   |   |   |
1922 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1923 1 *  |   |   |///////////////////////////////////////////////|
1924 1 *  | 10  | 11  |///////////////////////////////////////////////|
1925 1 *  |   |   |///////////////////////////////////////////////|
1926 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1927 1 *  |           |           |///////////////|           |
1928 1 *  |  CFTXTA   |  CCMDA    |///////////////|  CLTXTA   |
1929 1 *  |           |           |///////////////|           |
1930 1 *  |---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1931 1 *  |                                                       |
1932 1 *  |                C    N    A    M    E                   |
1933 1 *  |                                                       |
1934 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1935 1 *
1936 1 *
1937 1 * (1)  MNEQ     MAXIMUM NBR. OF ELEMENTARY QUALIFICATIONS.
1938 1 * (2)  MNQF     MAXIMUM NBR. OF COMPLEX    QUALIFICATIONS.
1939 1 * (3)  CNEQ     NBR. OF ELEMENTARY QUALIFICATIONS.
1940 1 * (4)  CNQF     NBR. OF COMPLEX    QUALIFICATIONS.
1941 1 * (5)  CPRNT    INDEX OF PARENT.
1942 1 * (6)  CLABEL   CURRENT LABEL.
1943 1 * (7)  CTYPE    LABEL TYPE (SEE #CTXE).
1944 1 * (8)  COBJN    OBJECT NUMBER.
1945 1 * (9)  CRELN    RELATION NUMBER.
1946 1 * (10) CLEVEL   LEVEL OF LABEL.
1947 1 * (11) CCNTLNG  LENGTH OF COUNTER AREA.
1948 1 * (12) CFTXTA   ADDRESS OF FIRST SOURCE LINE.
1949 1 * (13) CCMDA    ADDRESS OF COMMAND.
1950 1 * (14) CLTXTA   ADDRESS OF LAST  SOURCE LINE.
1951 1 * (15) CNAME    NAME OF OBJECT.
1952 1 *
1953 1 ICODEO    DS    OF
1954 1 MNEQ      DS    H                    |MAX. NBR. OF ELEM QUAL.
1955 1 MNQF      DS    H                    |MAX. NBR. OF COMPLEX QUAL.
1956 1 CNEQ      DS    H                    |NBR. OF ELEM. QUAL.
```

Figure 2.2.2.

```
ADDR2  STMNT M SOURCE

       1957 1 CNQF       DS    H        |NBR. OF COMPLEX QUAL.
       1958 1            DS    CL8      |NOT USED.
       1959 1 CPRNT      DS    H        |INDEX OF PARENT
       1960 1            DS    CL5      |NOT USED.
       1961 1 CLABEL     DS    CL3      |CURRENT LABEL.
       1962 1 CTYPE      DS    CL1      |LABEL TYPE.
       1963 1 COBJN      DS    H        |OBJECT-NUMBER.
       1964 1 CRELN      DS    H        |RELATION NUMBER.
       1965 1 CLEVEL     DS    H        |LEVEL OF CURRENT LABEL.
       1966 1 CCNTLNG    DS    H        |LENGTH OF COUNTER ARE
       1967 1            DS    CL12     |NOT USED.
       1968 1 CFTXTA     DS    F        |ADDR. OF FIRST SOURCE LINE.
       1969 1 CCMDA      DS    F        |ADDR. OF COMMAND
       1970 1            DS    F        |NOT USED.
       1971 1 CLTXTA     DS    F        |ADDR. OF LAST  SOURCE LINE.
       1972 1 CNAME      DS    CL16     |NAME OF OBJECT.
       1973   ICODE1     DS    CL288
       1974   ICODE2     DS    CL320
```

Figure 2.2.2.(suite)

2.2.3. EQE - Bloc de description d'un critère élémentaire

Un bloc de ce type est garni dans la zone ICODE1 lors de l'analyse d'un critère élémentaire. Celle-ci sera, ensuite, exploitée pour la construction du buffer de communication destiné au noyau SPHINX, associé à la phrase courante.

La figure 2.2.3. représente la découpe d'un bloc EQE. Le caractère bizarre de cette découpe est dû à l'utilisation de SESAM : à partir du quatrième caractère, EQE peut être considéré comme un morceau d'une commande SESAM (indication-Field).

Voici la signification de ces différentes zones :

NBRC contient, en binaire, le nombre de valeurs de comparaison contenues dans le critère élémentaire.

ATTLNG a pour valeur la longueur de l'objet élémentaire mentionné dans le critère (longueur interne).

BOOL contient le caractère 'C' si NBRC = 1
'D' si NBRC > 1

ASP contient le nom de l'aspect SESAM, constituant la représentation SESAM de l'objet élémentaire mentionné dans le critère.

COMPO contient le caractère '5'.

COMPI les 2 * NBRC caractères suivants contiennent un code de deux caractères répété NBRC fois. Ce code dépend de l'opérateur contenu dans le critère élémentaire :

Opérateur	COMPI	
=	0 1	
>	0 4	(1 seule valeur)
<	0 2	(1 seule valeur)
NOT =	0 6	
NOT >	0 5	(1 seule valeur)
NOT <	0 3	(1 seule valeur)

```
#

>DR2  STMNT M SOURCE

1975                 #EQE
1976 1 *-------------------------------------------------------------+
1977 1 *              ELEMENTARY QUALIFICATION ENTRY.                I
1978 1 *-------------------------------------------------------------+
1979 1 *
1980 1 *                                                   +-----+
1981 1 *  +---+---+---+---+---+---+---+---+---+            |     |
1982 1 *  |   |   |   |   |   |   |   |   |   |            |     |
1983 1 *  |NBRC |2  |3  |   ASP   |5  |COMP1|   ...      |COMPI|
1984 1 *  |   |   |   |   |   |   |   |   |   |            |     |
1985 1 *  +---+---+---+---+---+---+---+---+---+            +-----+
1986 1 *
1987 1 *
1988 1 * (1) NBRC     NUMBER OF COMPARISON VALUES.
1989 1 * (2) ATTLNG   LENGTH OF ATTRIBUTE.
1990 1 * (3) BOOL     BOOLEAN OPERATOR CODE.
1991 1 * (4) ASP      ASPECT NAME.
1992 1 * (5) COMPO    COMPARISON OPERATOR CODE (PREFIX).
1993 1 * (6) COMPI    COMPARISON OPERATOR CODE (SUFFIX).
1994 1 *
1995 1 EQE       DSECT                          |NUMBER OF COMP. VAL.
1996 1 NBRC      DS     H                       |LENGTH OF ATTRIBUTE.
1997 1 ATTLNG    DS     CL1                     |BOOLEAN OPERATOR.
1998 1 BOOL      DS     CL1                     |ASPECT NAME.
1999 1 ASP       DS     CL3                     |COMP. OPERATOR (PREFIX).
2000 1 COMPO     DS     CL1                     |COMP. OPERATOR (SUFFIX).
2001 1 COMPI     DS     CL2
2002 1 NULSTX    CSECT
```

Figure 2.2.3.

2.2.4. Bloc de description d'un critère complexe

Un bloc de ce type est garni, dans la zone ICODE2, lors de l'analyse d'un critère complexe. Il sera, ensuite, utilisé pour initialiser une nouvelle entrée CTXE, correspondant à un qualificatif, dans la zone CONTROL. Cette entrée sera, finalement, complétée par l'analyse du qualificatif correspondant.

La figure 2.2.4. représente la découpe d'un bloc QFE. Toutes les zones d'un tel bloc ont un correspondant de même signification, dans le bloc CTXE, il suffit de supprimer le premier X du nom de chaque champ de QFE, pour obtenir le nom de son équivalent dans CTXE.

```
ADDR2  STMNT M SOURCE

2003                 #QFE
2004  1  *-----------------------------------------------------------------+
2005  1  *                 QUALIFICATION FLAG ENTRY.                       I
2006  1  *-----------------------------------------------------------------+
2007  1  *
2008  1  *
2009  1  *  +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2010  1  *  |     |     |     |     |     |     |/////////////////////|
2011  1  *  | XLABEL |2 |XOBJN|XRELN|  5  |XPRNT|/////////////////////|
2012  1  *  |     |     |     |     |     |     |/////////////////////|
2013  1  *  +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2014  1  *  |     |     |///|                 |                         |
2015  1  *  |XICNT|8    |///|   XQVAL         |   XNAME (16 CHAR.)      |
2016  1  *  |     |     |///|                 |                         |
2017  1  *  +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2018  1  *
2019  1  *
2020  1  *  (1)  XLABEL   LABEL OF QUALIFICATION FLAG.
2021  1  *  (2)  XTYPE    TYPE OF QUALIFICATION FLAG.
2022  1  *  (3)  XOBJN    OBJECT NUMBER.
2023  1  *  (4)  XRELN    RELATION NUMBER.
2024  1  *  (5)  XLEVEL   LEVEL OF QUALIFICATION FLAG.
2025  1  *  (6)  XPRNT    INDEX OF PARENT.
2026  1  *  (7)  XICNT    INDEX OF COUNTER IN ORIGIN.
2027  1  *  (8)  XQT      QUANTIFIER CODE.
2028  1  *  (9)  XQVAL    QUANTIFIER VALUE.
2029  1  *  (10) XNAME    OBJECT NAME.
2030  1  *
2031  1  QFE       DSECT
2032  1  XLABEL    DS     CL3                    |LABEL NAME.
2033  1  XTYPE     DS     CL1                    |LABEL TYPE.
2034  1  XOBJN     DS     H                      |OBJECT NUMBER.
2035  1  XRELN     DS     H                      |RELATION NUMBER.
2036  1  XLEVEL    DS     H                      |LEVEL OF LABEL.
2037  1  XPRNT     DS     H                      |INDEX OF PARENT.
2038  1           DS     F                      |RESERVED.
2039  1  XICNT     DS     H                      |INDEX OF COUNTER.
2040  1  XQT       DS     CL1                    |QUANTIFIER CODE.
2041  1           DS     CL1                    |RESERVED.
2042  1  XQVAL     DS     F                      |QUANTIFIER VALUE.
2043  1  XNAME     DS     CL16                   |OBJECT NAME.
2044  1  NULSTX    CSECT
```

Figure 2.2.4.

2.2.5. Mémorisation du texte source

Chaque ligne, introduite par l'utilisateur pour constituer une phrase dans un contexte ou une commande PRINT, est mémorisée dans un bloc de mémoire réservé dynamiquement. Tous les blocs mémorisant une même phrase sont chaînés entre eux. L'adresse du premier bloc sera finalement mémorisée dans la zone TXTA du bloc CTXE correspondant.

La figure 2.2.5. présente la découpe d'un bloc de texte source. Pour le dernier bloc, on a TXTCHAIN = 0.

```
ADDR2  STMNT N SOURCE

       *-----------------------------------------------------------------*
       *                        SOURCE TEXT                              *
       *-----------------------------------------------------------------*
       *
       *
       * +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
       * |                 |                 |                         |
       * |  TXTCHAIN       |     VTYPE       |        T  E  X  T       |
       * |                 |                 |                         |
       * +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
       *
       * TEXTCHAIN: POINTER TO NEXT AREA
       * VTYPE    : LENGTH OF TEXT (V-TYPE)
       * TEXT     : SOURCE TEXT (76 CHARACTERS)
```

Figure 2.2.5.

2.2.6. BUFCM - Buffer de communication avec le noyau SPHINX

Les buffers de communication envoyés par l'interpréteur NUL au noyau SPHINX ont exactement la même forme que ceux qui lui sont envoyés par l'interface des programmes DML. La seule restriction porte sur les codes possibles : ils se réduisent à cinq.

01 : OPEN
02 : CLOSE
03 : REACH
06 : END
20 : Demande d'autorisation d'accès.

Pour chaque phrase du contexte, un buffer de communication de code 03 est créé dans un bloc dynamique de mémoire. Son adresse est placée dans la zone CMDA du bloc CTXE. Ce buffer sera envoyé au noyau lors de l'évaluation d'une commande PRINT portant sur la hiérarchie du contexte contenant cette phrase. Pour plus de détails sur le contenu de ces buffers de communication, leur création et leur exploitation, on se reportera aux listings des modules de l'interpréteur NUL et du noyau SPHINX.

La figure 2.2.5. représente la structure d'un buffer NUL associé à une phrase du contexte.

```
TMNT M SOURCE
706   *=================================================+
707   *                                                 I
708   *              B U I L D   C O M M A N D          I
709   *                                                 I
710   *=================================================+
711   *
712   * THIS MODULE  PREPARES  THE COMMAND  ASSOCIATED WITH
713   * THE CURRENT CONTEXT ENTRY.
714   *
715   * THE FORMAT OF THE  COMMAND  IS COMPATIBLE  WITH THE
716   * THE FORMAT REQUIRED BY SPHINX V.
717   *
718   * THE ADDRESS OF THE COMMAND IS STORED IN CCMDA.
719   *
720   *
721   * GENERAL FORMAT OF AN ACCESS COMMAND.
722   * ------------------------------------
723   *
724   *  +---+---+---+---+---+---+---+---+---+---+---+---+
725   *  %       %/////%       %       %       %  %//%      %
726   *  %   1   %/////%   2   %   3   %   4   % 5%//%   6  %
727   *  %       %/////%       %       %       %  %//%      %
728   *  +---+---+---+---+---+---+---+---+---+---+---+---+
729   *  %  %/////////%           %     %     %     %     %
730   *  %7 %/////////%     8     %  9  %  10 %  11 %  12 %
731   *  %  %/////////%           %     %     %     %     %
732   *  +---+---+---+---+---+---+---+---+---+---+---+---+
733   *  %     %     %     %     %                       %
734   *  %  13 %  14 %  15 %  16 %  17 / 18     ...      %
735   *  %     %     %     %     %                       %
736   *  +---+---+---+---+---+---+---+---+---+---+---+---+
737   *
738   *
739   * (1)  BLNGTH      BUFFER LENGTH.
740   * (2)  BCODE       BUFFER CODE (BINARY).
741   * (3)  BLEVEL      BUFFER LEVEL (BINARY).
742   * (4)  BSEQNBR     BUFFER SEQUENCE NUMBER ("@NUL").
743   * (5)  BDBN        DATA BASE NUMBER.
744   * (6)  BSTATUS     SPHINX V RETURN CODE.
745   * (7)  BREACHC     REACH CODE.
746   *        - 1  ACCESS WITH CRITERIA ON ATTRIBUTES VALUES.
747   *        - 2  ACCESS VIA A RELATION WITH CRITERIA ON ATTRIBUTES VALUES.
748   *        - 3  ACCESS VIA A RELATION WITHOUT CRITERIA.
749   *        - 9  NEXT RESPONSE.
```

Figure 2.2.6.

```
TMNT M SOURCE
                                                                    077
 750    * (8)   BRECNBR      RECORD NUMBER.                          077
 751    * (9)   BOBJN        OBJECT NUMBER.                          077
 752    * (10)  BRELN        RELATION NUMBER.                        077
 753    * (11)  BFLEVEL      FROM LEVEL.                             077
 754    * (12)  BXLEVEL      GET LEVEL.                              077
 755    * (13)  BQLENGTH     LENGTH OF QUESTION-FIELD.               078
 756    * (14)  BRLENGTH     LENGTH OF RESPONSE-FIELD.               078
 757    * (15)  BINDLG1      LENGTH OF CRITERIA PART IN INDICATION-FIELD.  078
 758    * (16)  BINDLG2      LENGTH OF   'GET'  PART IN INDICATION-FIELD.  078
 759    * (17)  BINDFLDV     INDICATION-FIELD (V-TYPE).              078
 760    * (18)  BINDFLD      INDICATION-FIELD.
```

Figure 2.2.6. (suite)

2.2.7. QTFSW - QTFCNT - Indicateur et compteur associés à un quantificateur

A tout qualificatif représenté par un bloc CTXE d'indice i dans la table CONTROL, est associé un indicateur QTFSW (1 byte) et un compteur QTFCNT (1 demi mot) dans la zone QTFAREA.

L'indice de l'indicateur et du compteur dans leurs sous-zones respectives est j = i - SMAX.

Ce compteur et cet indicateur seront utilisés lors de l'interprétation du code intermédiaire associé à une commande PRINT.

QTFCNT contiendra le nombre de réalisations de l'objet complexe principal du qualificatif X_k (représenté par $CTXE_i$) associées par la relation principale du qualificatif à la réalisation courante du parent.

QTFSW vaudra vrai (01 binaire) si le quantificateur du qualificatif X_k est vérifié pour la réalisation courante du parent. Il vaudra faux (0 binaire) sinon.

```
T M SOURCE                                                                  113301
                                                                            113401
 9   *  QUANTIFIER AREA (QTFAREA).                                          113501
10   *  --------------------------                                          113601
11   *                                                                      113701
12   *  A QUANTIFIER  SWITCH  AND A  QUANTIFIER COUNTER IS                  113801
13   *  IN THE QUANTIFIER AREA (QTFAREA) IS  ASSOCIATED TO                  113901
04   *  EACH X-ENTRY.                                                       114001
05   *                                                                      114101
06   *  THE QUANTIFIER  SWITCH  IS SET ON BY THE OPERATION                  114201
07   *  TSTCNT IF QUALIFICATION IS TRUE                                     114301
08   *  THE QUANTIFIER COUNTER IS UPDATED BY THE OPERATION                  114401
09   *  SETCNT AND CLEARED BY TSTQTF.                                       114501
10   *                                                                      114601
11   *                                                                      114701
12   *  QTFSW   %S % 1% 2% 3% 4% 5% 6%  ...  %14%15%16%                     114801
13   *                                                                      114901
14   *                                                                      115001
15   *                                                                      115101
16   *                                                                      115201
17   *  QTFCNT  % QTFC   %     1     %  ...  %     16     %
18   *
```

Figure 2.2.7.

2.3. Blocs de description d'une commande PRINT

L'introduction, par l'utilisateur NUL, d'une commande PRINT produit, tout d'abord, la transformation de cette commande en un code intermédiaire. Ce code intermédiaire est ensuite interprété.

Trois blocs de mémoire fixes sont réservés pour contenir le code intermédiaire :

ICODE3 : contient des instructions en code intermédiaire, associées aux définitions de contexte.

ICODE4 : contient des instructions en code intermédiaire, associées aux qualificatifs.

EDTAB : contient les tables d'édition associées à chaque étiquette S_i figurant dans la commande PRINT.

2.3.1. EXCODE - Instructions en code intermédiaire

La figure 2.3.1.représente les différentes configurations possibles d'une instruction en code intermédiaire. La traduction d'une commande PRINT par l'interpréteur NUL provoque le remplissage des blocs de mémoire ICODE3 et ICODE4 par une série d'instructions EXCODE, rangées séquentiellement à partir du début de ces zones. L'indice d'une instruction dans un des blocs joue le rôle d'adresse de l'instruction.

Une instruction EXCODE se décompose en plusieurs champs n'ayant pas tous une signification pour tous les codes instructions. Ces champs sont les suivants :

OPCODE	contient le 'code instruction' (1 byte)
BRCODE	contient un indicateur précisant si l'instruction suivante devra être recherchée dans ICODE3 (tous les bits à 1) ou dans ICODE4 (tous les bits à 0) (1 byte).
OPER1	contient l'indice, dans la zone CONTROL, du bloc CTXE décrivant la phrase du contexte à laquelle l'instruction est associée. (1 demi mot binaire)
OPER2F,OPER2H	contiennent les indices des instructions suivantes à exécuter dans ICODE3 ou ICODE4 lorsque ces instructions ne sont pas en séquence. (2 demi mots binaires)

On trouvera dans la description du module NULP32 (paragraphe 3.4.2.) la signification des différents codes possibles.

```
IT M SOURCE
)1    * INIT (04)
)2    * ---------
)3    *
)4    *          +---+---+---+---+---+---+---+---+
)5    *          %1 %//%  3  %//////////%
)6    *          +---+---+---+---+---+---+---+---+
)7    *
)8    * (1) OPERATION CODE
)9    * (3) INDEX OF CONTEXT ENTRY
)10   *
)11   * REACH (08)
)12   * ----------
)13   *
)14   *          +---+---+---+---+---+---+---+---+
)15   *          % 1% 2%  3  %  4  %  5  %
)16   *          +---+---+---+---+---+---+---+---+
)17   *
)18   * (1) OPERATION CODE
)19   * (2) BRANCHING CODE
)20   *      (2) = X'FF' --> PROCESSING CONTINUE IN ICODE3
)21   *      (2) = X'00' --> PROCESSING CONTINUE IN ICODE4
)22   * (3) INDEX OF CONTEXT ENTRY.
)23   * (4) BRANCHING INDEX IN CODING TABLE ICODE4 ((2)=00)
)24   * (5) BRANCHING INDEX IN CODING TABLE ICODE3 (STATUS 10)
)25   *
)26   * END (0C).
)27   * --------
)28   *
)29   *          +---+---+---+---+---+---+---+---+
)30   *          % 1%//%  3  %//////////%
)31   *          +---+---+---+---+---+---+---+---+
)32   *
)33   * (1) OPERATION CODE
)34   * (3) INDEX OF CONTEXT ENTRY
)35   *
)36   * PRINT (10).
)37   * ----------
)38   *
)39   *          +---+---+---+---+---+---+---+---+
)40   *          % 1%//%  3  %/////%  5  %
)41   *          +---+---+---+---+---+---+---+---+
)42   *
)43   * (1) OPERATION CODE
)44   * (3) INDEX OF CONTEXT ENTRY
)45   * (5) ADDRESS OF EDIT TABLE.
)46   *
)47   * NOTE: FIRST BIT OF NULST4 INDICATES OUTPUT OPTION.
```

Figure 2.3.1.

```
T M SOURCE

8    * GOTO (14)
.9   * ---------
0    *
1    *          +---+---+---+---+---+---+---+
2    *          % 1%//%  3  %//////%  5  %
3    *          +---+---+---+---+---+---+---+
4    *
5    *
5    * (1) OPERATION CODE
6    * (3) INDEX OF CONTEXT ENTRY
7    * (5) BRANCHING INDEX IN ICODE3
8    *
9    * TSTCNT (18)
0    * -----------
1    *
2    *          +---+---+---+---+---+---+---+
3    *          % 1%//%  3  %//////%  5  %
4    *          +---+---+---+---+---+---+---+
5    *
5    *
6    * (1) OPERATION CODE
7    * (3) INDEX OF CONTEXT ENTRY
8    * (5) BRANCHING INDEX IN CODING TABLE ICODE4
9    *
0    * TSTQTF (1C)
1    * -----------
2    *
3    *
4    *          +---+---+---+---+---+---+---+
5    *          % 1% 2%  3  %  4  %  5  %
6    *          +---+---+---+---+---+---+---+
7    *
7    *
8    * (1) OPERATION CODE
9    * (2) BRANCHING CODE
0    *     (2) = X'FF' --> BRANCHING IN ICODE3
1    *     (2) = X'00' --> BRANCHING IN ICODE4
2    * (3) INDEX OF CONTEXT ENTRY
3    * (4) BRANCHING INDEX IN ICODE3 IF TEST POSITIVE
4    * (5) BRANCHING ADDRESS IN CODING TABLE ICODE3/ICODE4
5    *     DEPENDING ON (2) IF TEST NEGATIVE
6    *
7    *
8    * SETCNT (20)
9    * -----------
0    *
1    *
2    *          +---+---+---+---+---+---+---+
3    *          % 1% 2%  3  %//////////%
4    *          +---+---+---+---+---+---+---+
5    *
6    * (1) OPERATION CODE
7    * (2) BRANCHING CODE
8    * (3) CONTEXT ENTRY INDEX
```

Figure 2.3.1. (suite)

2.3.2. HEDT, EDTE - Table d'édition

Pour chaque étiquette S_i citée dans la commande PRINT, l'interpréteur construit une table d'édition dans la zone EDTAB.
Cette table sera utilisée lors de l'interprétation du code intermédiaire pour éditer chaque réponse reçue de SPHINX pour l'étiquette S_i.

Chaque table d'édition comprend 1 bloc d'en-tête (HEDT) et zéro, un ou plusieurs blocs de description des valeurs d'un objet élémentaire (de niveau 1) dont l'impression a été demandée dans la commande PRINT, implicitement ou explicitement (EDTE).

La figure 2.3.2.1. présente la découpe du bloc d'en-tête HEDT.

ENBR contient, en binaire, le nombre de blocs EDTE de la table d'édition.

ELEVEL contient deux caractères numériques représentant le niveau de S_i dans la hiérarchie d'impression.

ELABEL a pour valeur S_i où i est représenté en 2 caractères numériques.

ELEVB contient, en binaire, le niveau de S_i dans la hiérarchie d'impression.

La figure 2.3.2.2. présente la découpe d'un bloc EDTE.
Ce bloc contient les renseignements suivants :

ENAME nom d'un objet élémentaire dont l'impression a été demandée implicitement ou explicitement dans la commande PRINT.

EOFFSET déplacement de la première valeur de l'objet élémentaire par rapport au début de la zone réponse renvoyée par le noyau SPHINX.

ELNG longueur totale occupée par les valeurs de l'objet élémentaire.

```
T M SOURCE                                                                  134100
                #HEDT
1 1 *-------------------------------------------------------------+
1 1 *         HEADER OF EDIT TABLE (FIRST ENTRY)                  I
1 1 *-------------------------------------------------------------+
1 1 *
1 1 *
1 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 1 *  |   |                                |   |///////////////|
1 1 *  |ENBR|        '-LL-XXX.'             | 3 |///////////////|
1 1 *  |   |                                |   |///////////////|
1 1 *  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 1 *
1 1 *
1 1 *   ENBR    NUMBER OF ATTRIBUTES.
1 1 *   LL      LEVEL IN OUTPUT HIERARCHY (ELEVEL).
1 1 *   XXX     LABEL OF CORRESPONDING ENTRY IN CONTEXT (ELABEL).
1 1 *   ELEVB   OUTPUT LEVEL (BINARY).
1 1 *
1 1 HEDT      DSECT
1 1 ENBR      DS    H                    |NUMBER OF ATTRIBUTES.
1 1 EHEADER   DS    0CL8                 |
1 1           DS    CL1                  |
1 1 ELEVEL    DS    CL2                  |LEVEL IN OUTPUT HIERARCHY.
1 1           DS    CL1                  |
1 1 ELABEL    DS    CL3                  |LABEL OF CORRESPONDING ENTRY.
1 1           DS    CL1                  |
1 1 ELEVB     DS    H                    |OUTPUT LEVEL (BINARY)
1 1 NULDOC    CSECT
```

Figure 2.3.2.1.

```
IT M SOURCE

57                #EDTE
58 1 *-----------------------------------------------------------+
59 1 *                    EDIT TABLE ENTRY.                      I
60 1 *-----------------------------------------------------------+
61 1 *
62 1 *
63 1 *  +--+--+-    - - - -    -+--+--+--+--+--+--+--+
64 1 *  |                                |     |     |
65 1 *  |              E N A M E         |  2  |  3  |
66 1 *  |                                |     |     |
67 1 *  +--+--+-    - - - -    -+--+--+--+--+--+--+--+
68 1 *
69 1 *
70 1 * (1)  ENAME   NAME OF ATTRIBUTE.
71 1 * (2)  EOFFSET OFFSET OF VALUE IN RESPONSE.
72 1 * (3)  ELNG    LENGTH OF VALUE.
73 1 *
74 1 EDTE      DSECT
75 1 ENAME     DS     CL16                  |NAME OF ATTRIBUTE.
76 1 EOFFSET   DS     H                     |OFFSET OF VALUE.
77 1 ELNG      DS     H                     |LENGTH OF VALUE.
78 1 NULDOC    CSECT
```

Figure 2.3.2.2.

2.4. Argument d'interrogation du schéma interne

Le schéma interne utilisé par l'interpréteur NUL est le même que celui qui est nécessaire au précompilateur DML.
Ce schéma est contenu dans un fichier séquentiel indexé dont on trouvera une description complète, par exemple, dans le listing du modèle NULSCH chargé de l'interrogation du schéma interne au sein de l'interpréteur NUL.

La figure 2.4. fournit la description de l'argument d'appel de ce module. Les composants de cet argument n'ont pas tous une signification pour toutes les valeurs du code opération.

Une composante peut être garnie lors de l'appel, elle sert alors d'argument de recherche , ou lors du retour, elle est alors un résultat de l'interrogation. Certaines composantes peuvent jouer l'un ou l'autre rôle selon le code opération.

OPCODE est un byte décrivant le type d'opération demandée. On trouvera une description des codes possibles dans le listing du module NULSCH.

DBNBR contient le numéro de la base de donnée concernée par l'opération (DBNBR $\in$ {A,B,...,Z,1,...,6}).

OBJN1,OBJN2 sont deux demi-mots binaires contenant des numéros d'objets complexes (par exemple, l'origine et la cible d'une relation).

RELN est un demi-mot binaire contenant le numéro d'une relation entre objets complexes.

NAME est une zone de 16 caractères contenant le nom d'un type d'information du schéma : objet ou relation entre objets complexes. Les relations sans nom entre objets complexes ont pour nom '$' dans le schéma interne.

ICNT est un demi-mot binaire contenant l'indice, dans la zone des compteurs de la réalisation origine, du compteur d'occurrences issues de la réalisation, pour une certaine relation.

TYPE est un byte dont chaque bit (TYPE(i) i = 1,...,8) fournit une caractéristique des valeurs d'un objet élémentaire.

TYPE(1) = 1 ssi chaque valeur d'objet est une chaîne simple de caractères.
TYPE(2) = 1 ssi chaque valeur d'objet est un nombre décimal étendu.
TYPE(3) = 1 ssi chaque valeur d'objet est un nombre décimal condensé.
TYPE(4) = 1 ssi chaque valeur d'objet est un nombre binaire.
TYPE(7) = 1 ssi l'objet élémentaire est répétitif (J > 1).
TYPE(8) = 1 ssi l'objet élémentaire est composé.

OCCRS est un demi mot binaire égal à zéro si l'objet élémentaire est cible d'une relation de caractéristique J = 1 et égal à J si J > 1.

LNG est un demi mot binaire contenant le nombre de bytes nécessaires pour stocker l'ensemble des valeurs de l'objet élémentaire associées à une même réalisation d'objet complexe.

NASP est un demi mot binaire égal au nombre d'aspects SESAM réservés pour stocker les valeurs de l'objet élémentaire.

ASPLST contient la liste des noms de ces aspects (3 caractères par nom d'aspect).

```
IT M SOURCE
37   *=================================================+
                                                       I
38   *                                                 I
39   *            S C H E M A   H A N D L E R          I
40   *                                                 I
41   *=================================================+
42   *
43   *
44   *(GR1)
45   *  %
46   *  %
47   *  V
48   *  +---+---+---+---+---+---+---+---+---+-                  -+---+---+
49   *  %   %   %       %       %       %                                  %
50   *  %1  %2  %OBJN1%OBJN2%RELN %    NAME (16 CHARACT.)                  %
51   *  %   %   %       %       %       %                                  %
52   *  +---+---+---+---+---+---+---+---+---+-                  -+---+---+
53   *  %7  %//%        %       %       %                                  %
54   *  %   %   %OCCRS% LNG %NASP %   ASPLST (15 CHARACT.)                 %
55   *  %ICNT %         %       %       %                                  %
56   *  +---+---+---+---+---+---+---+---+---+-                  -+---+---+
57   *
58   *  (1) OP CODE OPERATION CODE
59   *  (2) DBNBR  DATA BASE NUMBER.
60   *  (3) OBJN1  OBJECT NUMBER.
61   *  (4) OBJN2  OBJECT NUMBER.
62   *  (5) RELN   RELATION NUMBER.
63   *  (6) NAME   NAME  OF  DATA BASE / OBJECT / RELATION
64   *                       / ATTRIBUTE.
65   *  (7)  ICNT   INDEX OF COUNTER INTO ORIGIN.
66   *  (7') TYPE   TYPE OF ATTRIBUTE.
67   *              2**7  STRING OF CHARACTERS.
68   *              2**6  ZONED DICIMAL NUMBER.
69   *              2**5  PACKED NUMBER.
70   *              2**4  BINARY NUMBER.
71   *              2**3  RESERVED.
72   *              2**2  RESERVED.
73   *              2**1  ARRAY (OCCURS CLAUSE).
74   *              2**0  STRUCTURE.
75   *  (8) OCCRS  OCCURS VALUE - IF NO ARRAY X'0000'.
76   *  (9) LNG    TOTAL LENGTH OF ATTRIBUTE.
77   * (10) NASP   NUMBER OF ASPECTS.
78   * (11) ASPLST ASPECT NAMES.
```

Figure 2.4.

```
T M SOURCE                                                                  03930(
                                                                            03940(
9    * (1) OPCODE   X'00'  OPEN DATA BASE SCHEMA;                           03950(
                           USING :                                          03960(
10   *                     - NAME   = DATA BASE NAME.                       03970(
11   *                     GIVING :                                         03980(
32   *                     + DBNBR = DATA BASE NUMBER.                      03990(
33   *                X'04' SEARCH OBJECT DESCRIPTION;                      04000(
84   *                     USING :                                          04010(
85   *                     - DBNBR = DATA BASE NUMBER.                      04020(
86   *                     - NAME   = OBJECT NAME.                          04030
87   *                     GIVING :                                         04040
88   *                     + OBJN1 = OBJECT NUMBER.                         04050
89   *                X'08' SEARCH RELATION DESCRIPTION;                    04060
90   *                     USING :                                          04070
91   *                     - DBNBR = DATA BASE NUMBER.                      04080
92   *                     - OBJN1 = ORIGIN NUMBER.                         04090
93   *                     - OBJN2 = TARGET NUMBER.                         04100
94   *                     - NAME   = RELATION NAME.                        04110
95   *                     GIVING :                                         0412
96   *                     + RELN   = RELATION NUMBER.                      0413
97   *                     + ICNT   = INDEX OF COUNTER.                     0414
98   *                X'0C' SEARCH ATTRIBUTE DESCRIPTION;                   0415
99   *                     USING :                                          0416
00   *                     - DBNBR = DATA BASE NUMBER.                      0417
01   *                     - OBJN1 = OBJECT NUMBER.                         0418
02   *                     - NAME   = ATTRIBUTE NAME.                       041
03   *                     GIVING :                                         042
04   *                     + TYPE   = TYPE OF ATTRIBUTE.                    042
05   *                     + OCCRS = OCCURS VALUE.                          042
06   *                     + LNG    = TOTAL LENGTH.                         042
07   *                     + NASP   = NUMBER OF ASPECTS.                    042
08   *                     + ASPLST= ASPECT NAMES.                          042
09   *                X'10' SEARCH OBJECT DESCRIPTION;                      042
10   *                     USING :                                          04
11   *                     - DBNBR = DATA BASE NUMBER.                      04
12   *                     - OBJN1 = OBJECT NUMBER.                         04
13   *                     GIVING :                                         04
14   *                     + NAME   = OBJECT NAME.                          04
15   *                X'14' SEARCH FIRST OBJECT DESCRIPTION;                04
16   *                     USING :                                          04
17   *                     - DBNBR = DATA BASE NUMBER.                      04
18   *                     GIVING :                                         04
19   *                     + OBJN1 = OBJECT NUMBER.                         04
20   *                     + NAME   = OBJECT NAME.
21   *
```

Figure 2.4. (suite)

```
T M SOURCE
               X'18'  SEARCH FIRST ATTRIBUTE DESCRIPTION;
                      USING :
                      - DBNBR = DATA BASE NUMBER,
                      - OBJN1 = OBJECT NUMBER.
                      GIVING :
                      + NAME   = ATTRIBUTE NAME,
                      + TYPE   = ATTRIBUTE TYPE,
                      + OCCRS  = OCCURS VALUE,
                      + LNG    = TOTAL LENGTH,
                      + NASP   = NUMBER OF ASPECT,
                      + ASPLST= ASPECT NAMES.
               X'1C'  SEARCH FIRST RELATION DESCRIPTION;
                      USING :
                      - DBNBR = DATA BASE NUMBER.
                      GIVING :
                      + RELN   = RELATION NUMBER,
                      + OBJN1  = ORIGIN NUMER,
                      + OBJN2  = TARGET NUMBER,
                      + NAME   = RELATION NAME,
                      + ICNT   = INDEX OF COUNTER.
               X'20'  SEARCH FIRST RELATION DESCRIPTION;
                      USING :
                      - DBNBR = DATA BASE NUMBER,
                      - OBJN1 = ORIGIN NUMBER.
                      GIVING :
                      + RELN   = RELATION NUMBER,
                      + OBJN2  = TARGET NUMBER,
                      + NAME   = RELATION NAME,
                      + ICNT   = INDEX OF COUNTER.
               X'24'  SEARCH FIRST RELATION DESCRIPTION;
                      USING :
                      - DBNBR = BATA BASE NUMBER,
                      - OBJN1 = ORIGIN NUMBER,
                      - OBJN2 = TARGET NUMBER.
                      GIVING :
                      + RELN   = RELATION NUMBER,
                      + NAME   = RELATION NAME,
                      + ICNT   = INDEX OF COUNTER.
               X'28'  SEARCH NEXT DESCRIPTION DEPENDING
                      ON THE PRECEDING CALL.
               X'2C'  CLOSE SCHEMA.

      THE RETURN CODES ARE :
      X'0000'  NORMAL RETURN.
      X'0004'  DATA BASE / OBJECT / RELATION / ATTRIBUTE
               NOT FOUND.
```

Figure 2.4. (suite)

3. LES MODULES DE L'INTERPRETEUR NUL

3.1. Les différentes phases de l'exécution

On distingue diverses phases dans l'exécution de l'interpréteur NUL. Ces phases correspondent à une découpe modulaire. De plus, il existe une correspondance entre les phases et les modes d'utilisation (ouverture, contexte, commande).

Nous donnons ci-dessous une description sommaire des modules directeurs correspondant à cette découpe.

3.2. NULMN1 - Enchaînement global des phases

Le module principal de NUL a pour seul rôle d'enchaîner les principales phases de fonctionnement :

- phase 0 : initialisation du système.
- phase 1 : ouverture de la base de donnée et du schéma interne (mode ouverture).
- phase 2 : construction d'un contexte (mode contexte).
- phase 3 : analyse et exécution des commandes d'impression (mode commande).
- phase 8 : fermeture de la base de donnée et du schéma interne (retour au mode ouverture).
- phase 9 : arrêt du système NUL (commande Halt).

Le module NULMN1 réalisé lui-même la phase 0, les phases 1,2,3,8,9, sont réalisées respectivement par les modules NULPH1, NULPH2, NULPH3, NULPH8, NULPH9. Ces modules sont enchaînés à l'intérieur de NULMN1. Le lecteur se reportera aux listings des modules correspondants pour ce qui concerne les phases 0,1,8,9 qui sont très simples. Dans les paragraphes suivants, on trouvera une description un peu plus détaillée des modules relatifs aux phases 2 et 3.

3.3. Modules relatifs à la phase 2 - Représentation interne d'un contexte

3.3.1. NULPH2 - Module principal de construction d'un contexte

Ce module enchaîne les différentes sous-phases nécessaires à la traduction d'un contexte sous forme interne :

- initialisation des zones de représentation du contexte (CONTROL, XTOP, STOP,...) par un appel au point d'entrée CLRCTX du module NULCTX.
- itération sur l'appel du module NULSTX d'analyse syntaxique d'une phase du contexte. Soit i l'indice de la dernière définition de contexte traité (après CLRCTX, i = 0), l'itération se fait comme suit :
  - i := i+1;
  - appeler NULSTX pour traiter la définition de contexte S_i;
  - cette définition contient n qualificatifs $X_{j1},\ldots,X_{jn}$;
    appeler NULSTX n fois pour traiter les qualificatifs $X_{j1},\ldots,X_{jn}$.

Ces trois pas d'itération seront répétés tant que l'utilisateur n'aura pas rentré 'END','HALT' et que la capacité des tables internes de NUL ne sera pas dépassée.

Remarque : l'introduction d'un contexte vide provoquera le passage en mode ouverture : phase 8 suivie de phase 1.

3.3.2. NULSTX - Analyse d'une phrase du contexte

Ce module effectue les fonctions suivantes :

- analyse syntaxique d'une définition de contexte ou d'un qualificatif. Vérification de l'existence des objets et relations mentionnés dans la phrase NUL. La détection d'une erreur durant cette étape provoque l'impression d'un message et la modification de la phrase par l'utilisateur.
- constitution d'une représentation interne, intermédiaire, de la phrase dans les zones :
  ICODEO(CTXINPUT) : description générale de la phrase,
  ICODE1(EQE) : description des critères élémentaires,
  ICODE2(QFE) : description des critères complexes,
  VALAREA : valeurs d'objets élémentaires figurant dans les critères élémentaires.
- appel du module NULCMD qui construira à partir des renseignements contenus dans les tables ci-dessus, un buffer de communication pour SPHINX.
- appel du point d'entrée PUTCTX du module NULCTX, pour placer dans une nouvelle entrée CTXE, la représentation interne définitive de la phrase, tirée de ICODEO et ICODE2 (qualificatifs).

3.3.3. NULCMD - Constitution d'un buffer de communication associé à une phrase du contexte.

Le point d'entrée BUILDCMD de ce module a pour fonction de construire dans un bloc de mémoire dynamique, un buffer de communication pour le noyau SPHINX. Ce buffer est construit à partir des renseignements placés par NULSTX en ICODEO, ICODE1 et VALAREA et possède le code opération 03 (REACH). Il permettra de commander des accès à la base de données nécessaires pour réaliser le traitement défini par la phrase courante du contexte (accès ou vérification de conditions complexes).
On trouvera dans le listing de NULCMD la description complète de ce buffer. Essentiellement, il contiendra, outre des renseignements propres à SPHINX, une zone de commande SESAM (indication-field) décrivant l'équivalent dans ce système des critères élémentaires, contenus dans la phrase courante, et une zone de données selon un format SESAM (question-field) contenant les valeurs figurant dans les critères élémentaires.
Ces deux zones sont construites facilement à partir des blocs EQE et VALAREA, dont la structure a été définie dans ce but.

Dans le cas des définitions de contexte, le buffer de communication pourra être modifié, lors du traitement d'une commande PRINT, de manière à obtenir les valeurs d'objets élémentaires demandées dans la commande.

3.3.4. NULCTX - Gestion de la table des blocs CTXE de description du contexte

Ce module possède deux point d'entrée :

- CLRCTX est activé à chaque entrée dans le mode contexte pour initialiser les zones de description de contexte. C'est-à-dire pour rétablir la description d'un contexte vide. Ce qui correspond à positionner à leur valeur initiale les indices STOP, XTOP, CRNT,XCRNT (voir § 2.1.).
- PUTCTX est activé à la fin de l'analyse d'une phrase du contexte.
  Les fonctions suivantes sont réalisées :
  - Si la phrase courante est une définition de contexte, garnir le bloc CTXE d'indice STOP avec le contenu de CTXINPUT.
    Si c'est un qualificatif, garnir le bloc CTXE d'indice XCRNT avec les renseignements contenus dans CTXINPUT, non encore présents dans ce bloc : TXTA, CMDA. Les autres renseignements ayant été placés dans le bloc lors de l'activation de PUTCTX pour la phrase dont la phrase courante est un qualificatif (voir ci-dessous).
  - Si la phrase courante contient n critères complexes nantis de n qualificatifs $X_{j_1},\ldots,X_{j_n}$ (si pas de qualificatif, j_k = 00, on dit que le qualificatif est vide), initialiser n nouvelles entrées CTXE d'indices XTOP + 1,...,XTOP + n avec les renseignements contenus dans les n blocs QFE produits par NULSTX; faire XCRNT := XTOP + 1; XTOP := XTOP + n.

3.4. Modules relatifs à la phase 3 - Traduction et Interprétation d'une commande PRINT

3.4.1. NULP3S - Traduction d'une commande en code intermédiaire

Ce module effectue les fonctions suivantes :

- Détection du type de la commande introduite par l'utilisateur.
  Si c'est une commande END, HALT, CLOSE DB, l'exécution du module se termine avec un code de retour (RCODE) définissant un changement de phase.

- Si c'est une commande PRINT, analyse syntaxique de la commande et vérification de sa concordance avec le contexte.
  Les règles de concordance sont exposées à la page NUL.15 de la brochure "Manuels de référence".
  La structure de blocs de la commande (hiérarchie) nécessite, lors de cette analyse, l'utilisation de deux stacks STKOUT, STKSUB.

- Toute détection, dans la commande 'PRINT', d'une étiquette S_i provoque la construction d'une table d'édition des objets élémentaires et la modification du buffer de communication avec SPHINX. Si l'étiquette S_i figure seule dans la commande PRINT, la table d'édition décrira tous les objets élémentaires de niveau 1 associés à l'objet complexe de la définition S_i et le buffer de communication sera augmenté des aspects SESAM réservés pour ces objets élémentaires. Si l'étiquette est concaténée une ou plusieurs fois à un nom d'objet élémentaire, seuls ces objets seront concernés.

- De plus, lors de l'analyse de la commande PRINT, le code intermédiaire correspondant à cette commande est générée dans la zone ICODE3. On trouvera dans la description du module NULP32, chargé d'interpréter le code intermédiaire, la liste des instructions de code intermédiaire et leur signification.

- Lorsque les fonctions précédentes sont réalisées, l'exécution de NULP3S se termine après avoir appelé le module NULP3X, chargé de générer des instructions en code interpédiaire dans ICODE4. Ces instructions représentent les opérations à effectuer pour vérifier les critères complexes contenus dans la hiérarchie du contexte à laquelle la commande PRINT est relative. On trouvera également dans la description de NULP32 la signification de ces instructions.

3.4.2. NULP32 - Interprétation du code intermédiaire

Ce module résout la commande PRINT en exécutant le code intermédiaire placé par les modules NULP3S et NULP3X dans ICODE3 et ICODE4. L'exécution commence à la première instruction de ICODE3 et se déroule en séquence sauf branchement explicite, spécifié dans l'instruction.

Nous classons les instructions en trois catégories (voir § 2.3.1).

1. Les instructions pouvant figurer dans ICODE3 et ICODE4

Ce sont principalement les instructions d'accès, de bouclage et de branchement inconditionnel.

INIT (OPCODE = 4(binaire)) : OPER1 contient l'indice d'un bloc CTXE. L'exécution de l'instruction a pour effet de mettre à zéro, dans ce bloc, CCNT (compteur de réalisations) et TYPE(6) (indicateur : premier accès), de préparer l'envoi du buffer de communication d'adresse CMDA et, si c'est un accès de plus bas niveau, d'envoyer un bloc de demande d'autorisation d'accès à SPHINX.

REACH (OPCODE = 8) : OPER1 contient l'indice d'un bloc CTXE. Pour ce bloc, si TYPE(6) = 0, on envoie à SPHINX le buffer d'adresse CMDA (accès à la première réalisation). Si TYPE(6) = 1, on envoie un buffer de demande de réalisation suivante. Si une réponse est obtenue de SPHINX, son adresse est rangée en RESPA, on fait TYPE(6) := 1, ensuite, si BRCODE = x'ff' (tous bits à 1),l'exécution continue en séquence ; si BRCODE = 0, l'exécution continue dans la table ICODE4 à l'instruction d'indice OPER2F. Si SPHINX renvoie le message 'pas de réalisation suivante', l'exécution se continue dans la table ICODE3 (BRCODE = x'ff') ou ICODE4 (BRCODE = 0) à l'instruction d'indice OPER2H.

END (OPCODE = 12) : on envoie au noyau SPHINX un buffer signalant la clôture de l'itération sur l'ensemble des réalisations d'objet complexe défini par le bloc CTXE d'indice OPER1. (Equivalent à l'ordre DML @EXIT).

GOTO (OPCODE = 20) : branchement inconditionnel à l'instruction dont l'indice se trouve dans OPER2H, dans la table ICODE3 si BRCODE = x'ff' ou dans la table ICODE4 si BRCODE = 0.

2. Instructions propres aux définitions de contexte (ICODE3)

PRINT (OPCODE = 16). On imprime les valeurs d'objets élémentaires associés à la réalisation courante du bloc CTXE d'indice OPER1. Ces valeurs, placées dans la zone d'adresse RESPA, seront imprimées grâce à la table d'édition d'adresse EDTA (Appel du module NULEDT).

HALT (OPCODE = 255). Fin de l'interprétation du code intermédiaire. Impression du rapport cumulatif. Sortie du sous-programme NULP32.

3. Instructions propres aux qualificatifs (résolution des critères complexes - ICODE4)

SETCNT (OPCODE = 32). A chaque qualificatif X_i (i = 0,...,16) est associé un indicateur QTFSW et un compteur QTFCNT. OPER1 contient l'indice d'un bloc CTXE décrivant un qualificatif X_i. On met à faux (o binaire) l'indicateur QTFSW correspondant et, dans QTFCNT, le nombre de réalisations de l'objet complexe du qualificatif accessibles à partir de la réalisation courante du parent. (nombre d'occurrences de la relation d'accès)

TSTCNT (OPCODE = 24). OPER1 contient l'indice d'un bloc CTXE décrivant le qualificatif X_i. Deux cas sont possibles.

Le qualificatif est vide (i=o)

Alors si la valeur contenue dans le compteur QTFCNT satisfait le quantificateur, QTFSW est positionné à vrai (QTFSW :=1) et l'exécution continue à l'instruction d'indice OPER2H dans ICODE4.

Si la valeur contenue dans QTFCNT ne satisfait pas le quantificateur, l'indicateur QTFSW est positionné à faux (QTFSW := 0).

On remarque que dans ce cas, le critère complexe est résolu sans faire d'accès à la base de données.

Le qualificatif n'est pas vide ($i \in \{1,...,16\}$)

Alors, si la valeur contenue dans le compteur CCNT (nombre de réalisations accédées à partir du dernier parent) satisfait le quantificateur, l'indicateur QTSW est positionné à vrai, sinon il est positionné à faux.

Si, selon la nature du quantificateur, on peut déduire que la valeur de QTSW ne sera plus modifiée, l'exécution se continue à l'instruction d'indice OPER2H.

Exemple : 'AT LEAST 20' CCNT = 20.

TSTQTF (OPCODE = 28). OPER1 contient l'indice d'un bloc CTXE décrivant le qualificatif X_i. Si l'indicateur QTFSW associé a pour valeur vrai, l'exécution se continuera à l'instruction d'indice OPER2H, dans ICODE3 si BRCODE = x'ff' et dans ICODE4 si BRCODE = 0.

3.4.3. NULEDT - Impression des valeurs d'objets élémentaires associées à l réalisation

Ce module reçoit comme argument l'adresse d'une zone réponse reçue du noyau SPHINX contenant les valeurs d'objets élémentaires associées à une réalisation d'objet complexe, comme deuxième argument l'adresse d'une table d'édition et comme troisième argument le format d'édition 01, 02 ou 03. Il imprime les valeurs d'objets élémentaires conformément au format choisi. Cette impression se déroule soit à l'imprimante (LNT = 0), soit à l'imprimante et au terminal (LNT ≠ 0). Enfin, l'impression peut être arrêtée en cours de route par l'introduction de la commande 'CANCEL = PRINT'. (NULST4(8) := 1). On sortira alors du module NULEDT et l'interprétation du code intermédiaire sera également arrêtée.

3.5. NULINT - Contrôle du système NUL

Ce module est l'équivalent dans l'interpréteur NUL du module SYSINT du noyau SPHINX. La base de son fonctionnement est l'existence dans ce module d'une macro STXIT (BS2000), activée au début de l'exécution de l'interpréteur (point d'entrée INITMEM). Par la suite, lorsque l'utilisateur aura interrompu l'exécution normale en appuyant sur la touche 'ESCAPE' (T200) ou en envoyant un caractère 'EXT' (vidéo), il pourra provoquer l'activation du sous-programme NULINT en introduisant la commande '/INTR'.

Il pourra ensuite introduire n'importe quelle commande décrite dans le manuel NUL sous la rubrique 'Fonctions de contrôle du système NUL'. Le module NULINT exécutera la commande en consultant soit les tables internes de l'interpréteur, soit le schéma interne (appel du module NULSCH). Certaines commandes permettent aussi de modifier certains paramètres de l'interpréteur. Enfin, l'exécution normale reprendra là où elle avait été interrompue (macro EXIT CR).

3.6. Modules de service

L'interpréteur contient encore un certain nombre de modules dont le rôle est plus utilitaire et dont nous énonçons seulement les fonctions de base. Certains d'entre eux possèdent d'ailleurs un équivalent dans le noyau SPHINX.

NULSCH est le module d'interrogation du schéma interne. La figure 2.4. présente la liste des types d'interrogations possibles.

NULSYM est le module qui découpe le texte introduit par l'utilisateur en entités significatives pour NUL. C'est l'équivalent du module NXTSYM du compilateur DML.

NULDLG est un module qui imprime une ligne de résultats et accepte une ligne de réponse de l'utilisateur (ou une seule de ces fonctions à la fois). C'est le module standard d'entrée/sortie.

NULMSG est le module chargé de l'édition des messages d'erreur.

NULMEM est le module de gestion de la mémoire dynamique. C'est l'équivalent du module SYSMEM du noyau SPHINX.

NULREL libère les blocs de mémoire dynamique alloués pour les buffers de communication et le texte source associé à un contexte.

NULSPH gère l'échange des buffers de communication entre l'interpréteur NUL et le noyau SPHINX.

NULITC réalise les fonctions de communication entre tâches : ouverture, envoi d'un buffer, réception d'un buffer, fermeture de la communication. C'est l'équivalent du module SYSITC du noyau SPHINX.

LA SÉCURITÉ ET LES POINTS DE REPRISE

La sécurité d'une base de données gérée par le système SPHINX est assurée par la journalisation des mises à jour et les possibilités de restauration du système SESAM.

Les procédures de restauration de ce dernier système, si elles sont puissantes, sont cependant relativement complexes. C'est la raison pour laquelle deux utilitaires (UTY02 et UTY03) ont été construits, permettant de consulter les différentes marques (checkpoints) des journaux SESAM et de restaurer une base de données. En fait, l'utilitaire de restauration est un programme conversationnel capable de générer une procédure de restauration SESAM.

Le mode d'emploi et la description interne de ces programmes font l'objet d'une documentation sur listing. Celle-ci explique également les procédures de restauration SESAM (documentation PAU.DOC.CKPT).

°

° °

RÉORGANISATION D'UNE BASE DE DONNÉES

Une base de données en activité est soumise à des mises à jour (création, suppression, modification) susceptibles non seulement de modifier le volume des données, mais aussi de dégrader ses performances.
Cette dégradation s'observe à la fois dans une diminution des performances en accès et mises à jour, et dans une moins bonne utilisation de l'espace sur mémoire secondaire.

Une base de données gérée par SPHINX peut être soumise à deux types de dégradation; le premier type concerne l'organisation de la base telle qu'elle est vue et gérée par SESAM, et le second type concerne les caractéristiques propres au modèle d'accès et ignorées par SESAM.
Le premier type de dégradation est relatif aux différents index, à la séquence des n° d'enregistrement, à l'occupation des pages physiques (débordements, etc.).
Le second type de dégradation concerne la structure des blocs de pointeurs implémentant les relations d'accès. En effet, s'il y a extension d'une chaîne de blocs en cas d'accroissement, il n'y a pas compactage en cas de suppression d'occurrences d'une relation (*). Certaines situations peuvent arriver où plusieurs blocs d'une chaîne sont peu remplis, entraînant des pertes de place et une diminution des performances des accès.

Il y a lieu dans ce cas de réorganiser la base de données.

Réorganisation SESAM

Cette réorganisation est assez simple à réaliser; elle consiste à réorganiser les fichiers de base (par PRO.SEBE), puis à demander la reconstruction des fichiers inverses si nécessaire.
On consultera la documentation SESAM.

Réorganisation des blocs de pointeurs

L'utilitaire UTY01A a pour but de réorganiser les blocs de pointeurs d'une base de données soit tous, soit ceux qui concernent les relations issue d'un objet complexe déterminé.

Ce programme conversationnel demande le nom de la base de données, puis soit "ALL" (réorganisation complète), soit un numéro d'objet complexe; il ouvre alors une communication avec SESAM, consulte le schéma de la base de données

---

(*) Il y a une certaine optimisation dans la mesure où un bloc terminal vide est supprimé, et où les blocs non pleins sont remplis en priorité avant la création de nouveaux blocs. Cette stratégie n'est cependant pas suffisante pour éviter la déterioration progressive des accès.

puis réorganise les blocs attachés aux réalisations de l'objet complexe précisé (ou des objets complexes).

Cet utilitaire ne peut travailler que lorsque SPHINX n'a pas ouvert la base de données que l'on veut réorganiser (en effet SPHINX ne peut gérer cette utilisation partagée de la base de données); d'autre part, il faut veiller à ce que la tâche de réorganisation ne porte pas le numéro 1 qui est le numéro standard sous lequel SPHINX se fait connaître à SESAM.

La description interne de ce module est incluse dans le listing source.

°

° °

PROCÉDURES D'UTILISATION DU SYSTEME SPHINX

1. COMPILATION D'UN TEXTE DDL

Outre la déclaration des fichiers de travail, le compilateur ne demande que l'affectation à SYSDTA du fichier contenant le texte à compiler.

La procédure suivante permet la compilation d'un texte DDL avec rangement du schéma DML non pas dans le fichier de schémas standard CDDL.EXTERNAL.SCHEMA mais dans un fichier propre à l'utilisateur, en l'occurrence META.EXTERNAL.SCHEMA.

```
/PROC N,(&DBN)
/SYSFILE SYSDTA=&DBN
/FILE CDDL.SEDI12,LINK=S12READ
/FILE CDDL.SCHEMA,LINK=SCHWRIT
/CAT CDDL.EXTERNAL.SCHEMA,META.A,STATE=UPDATE
/STEP
/CAT META.EXTERNAL.SCHEMA,CDDL.EXTERNAL.SCHEMA,STATE=UPDATE
/STEP
/FILE CDDL.EXTERNAL.SCHEMA,LINK=SCHREAD
/FILE CDDL.SOURCE,LINK=INPUT
/FILE HIEFIL,LINK=HIEFIL
/FILE CDDL.DATA,LINK=DTFORM
/FILE CDDL.SEBE,LINK=SEBEOUT
/EXEC L.DDLEXEC
/SYSFILE SYSDTA=(PRIMARY)
/CAT CDDL.EXTERNAL.SCHEMA,META.EXTERNAL.SCHEMA,STATE=UPDATE
/CAT META.A,CDDL.EXTERNAL.SCHEMA,STATE=UPDATE
/ENDP
```

2. GENERATION D'UNE BASE DE DONNEES

On désigne par ce terme l'ensemble des opérations qui à partir des fichiers de paramètres construits par le compilateur DDL, conduisent à la création d'une base de données SPHINX initialisée. Cette base de données est une base SESAM contenant les enregistrements de description du schéma (schémas du noyau SPHINX) et des enregistrements nécessaires à l'exécution de certains ordres DML : enregistrements (dits "bidons") des valeurs "inconnu" des objets élémentaires et enregistrements des hiérarchies de suppression.

Ces différentes opérations, qui ne font appel qu'à des utilitaires SESAM, sont au minimum les suivantes :

- déclaration des fichiers ORG et ZDR au système d'exploitation
- déclaration de la base de données SESAM et fixation des paramètres physiques (utilitaire PRO.SEDI12 travaillant sur le fichier <nom de racine >.SEDI12)
- chargement du catalogue des aspects (utilitaire PRO.SEFO travaillant sur le fichier <nom de racine >.SEFO)
- chargement des enregistrements de description de SPHINX (utilitaire PRO.SEBE avec option GEN, travaillant sur les fichiers <nom de racine >.SEBE et <nom de racine >.DATA).

Pour plus de précisions, on se reportera à la documentation SESAM.

Exemple de procédure :

```
/PROC ,(&RAC)
/FILE &RAC..O
/FILE &RAC..Z
/SYSFILE SYSDTA=&RAC..SEDI12
/EXEC PRO.SEDI12
/SYSFILE SYSDTA=&RAC..SEFO
/EXEC  PRO.SEFO
/SYSFILE SYSLST=DDL.LST
/SYSFILE SYSDTA=&RAC..SEBE
/EXEC PRO.SEBE
/SYSFILE SYSLST=(PRIMARY)
/ER DDL.LST
/STEP
/SYSFILE SYSDTA=(PRIMARY)
/EXEC PRO.SEBE
/ENDP
```

3. COMPILATION ET ASSEMBLAGE D'UN PROGRAMME D'APPLICATION

3.1. Compilation d'un programme DML

La compilation d'un programme DML s'effectue par l'exécution d'une procédure cataloguée telle que celle qui est donnée ci-dessous.
Elle utilise un paramètre (&SOURCE) qui est le nom d'un fichier SAM ou ISAM (article à longueur variable) constitué du programme à compiler; d'autres paramètres pourraient être définis, tels que le nom du fichier des schémas (ici DEMO.EXTERNAL.SCHEMA) et le nom de la bibliothèque des modules objets (ici SIM.LIB.OBJECT).

Les différentes phases sont les suivantes :

- effacer tous les fichiers résultats ou de travail résultant d'une compilation antérieure éventuelle du même fichier source.
- définir les fichiers d'entrée, (&SOURCE, fichier des schémas), de travail (WORK2, WORK2, WORK3) et résultats (GENFILE, FDIAG et listing source).
- appeler le compilateur DML, dont le nom est L.SPHINX.DML1.
- appeler le compilateur COBOL qui doit travailler sur le fichier GENFILE et fournir un listing d'erreur (ERRFIL) et un généré binaire (fichier temporaire standard "*").
- mettre en bibliothèque le fichier temporaire "*".
- effacer les fichiers de travail, GENFILE et "*".

```
/PROC N,(&SOURCE)
/ERASE &SOURCE..
/STEP
/FILE &SOURCE..WORK.1,LINK=WORK1
/FILE &SOURCE..WORK.2,LINK=WORK2
/FILE &SOURCE..WORK.3,LINK=WORK3
/FILE &SOURCE..GEN,LINK=GENFILE
/FILE &SOURCE..DIAG,LINK=FDIAG
/FILE DEMO.EXTERNAL.SCHEMA,LINK=SCHEMA-S
/SYSFILE SYSLST=&SOURCE.-LIST
/SYSFILE SYSIPT=&SOURCE
/EXEC L.SPHINX.DML1
/PARAM ERRFIL=YES,LIST=YES,SYMDIC=YES
/FILE &SOURCE..ERRFIL,LINK=ERRFIL
/SYSFILE SYSDTA=&SOURCE..GEN
/EXEC L.ANSICOB(07)
/STEP
/SYSFILE SYSDTA=(SYSCMD)
/FILE &SOURCE..ERRFIL,LINK=ERRFIL
/FILE &SOURCE..GEN,LINK=ISOURCE
/FILE &SOURCE..DIAG,LINK=FDIAG,OPEN=EXTEND
/EXEC L.SPHINX.DML2
/STEP
```

```
/SYSFILE SYSLST=(PRIMARY)
/PRINT &SOURCE.-LIST,SPACE=E,ERASE
/EXEC LMR
CONTROL OUTFILE=(SIM.LIB.OBJECT,OLDLIB),LISTING=(BOTH,SYSOUT)
COPYALL SOURCE=*
END
/STEP
/ERASE &SOURCE..WORK.
/ERASE &SOURCE..GEN
/ERASE *
/ENDP
```

3.2. Assemblage d'un programme DML

Le module généré binaire résultant de la phase précédente doit être associé aux modules de tous les sous-programmes qu'il appelle (ou au programme qui l'appelle), et en particulier au module SYSDML qui constitue l'interface avec le noyau SPHINX.
Ce module (ainsi que les modules qu'il appelle) se trouvent dans la bibliothèque de nom DML.LIB.OBJECT.

L'exemple suivant réalise l'assemblage d'un programme à partir du module principal PP (nom du PROGRAM-ID COBOL) qui se trouve dans SIM.LIB.OBJECT et qui appelle des modules se trouvant dans cette même bibliothèque.
Le résultat constitue le fichier PROGPP.

```
/EXEC  TSOSLNK
*PROG XXX, FILENAM = PROGPP
*INCLUDE PP, SIM.LIB.OBJECT
*RESOLVE , SIM.LIB.OBJECT
*RESOLVE , DML.LIB.OBJECT
*END
```

L'exécution du programme complet sera alors commandée par :

```
/EXEC PROGPP
```

ou

```
/LOAD PROGPP
/R
```

4. LANCEMENT DE L'EXPLOITATION D'UN ENSEMBLE DE BASES DE DONNEES

On a vu précédemment (schéma général d'une exploitation page I.3.) que l'exploitation d'un ensemble de bases de données nécessitait la présence simultanée, en mémoire virtuelle,

- du noyau SESAM,
- du noyau SPHINX,
- d'un certain nombre de programmes DML et d'un certain nombre d'occurrences de l'interpréteur NUL.

Nous décrivons maintenant les procédures de lancement de ces différents programmes.

4.1. Lancement du noyau SESAM

Ce programme doit être lancé le premier. Il peut l'être soit en BATCH, soit en conversationnel, à partir d'un terminal.
Supposons qu'on ait rangé, dans le fichier PRO.SESAM, le noyau SESAM sous forme de programme objet. Il suffira alors d'entrer la commande

/EXEC PRO.SESAM

Le noyau SESAM sera chargé en mémoire virtuelle et demandera l'introduction de paramètres d'exécution, soit au terminal, soit au pupitre opérateur (BATCH). Le lecteur trouvera la liste et la signification de ces différents paramètres dans les brochures du système SESAM. Deux d'entre eux sont importants, en liaison avec le système SPHINX : le nombre maximum d'utilisateurs et le nombre maximum de bases de données ouvertes, autorisés lors de la session.

On trouvera également dans les brochures du système SESAM la liste des commandes de contrôle permettant d'interroger le noyau SESAM sur l'état de l'exploitation. Ces commandes sont peu utiles, en liaison avec le système SPHINX, puisqu'il est possible d'obtenir des renseignements semblables, selon une terminologie 'modèle d'accès', à partir du noyau SPHINX.

4.2. Lancement du noyau SPHINX

Ce programme doit être lancé après le noyau SESAM et avant tout programme d'application (DML ou NUL). Il peut l'être en BATCH, à la rigueur, mais seul un lancement en conversationnel, à partir d'un terminal, permettra d'exercer un contrôle sur le déroulement de l'exploitation et éventuellement de modifier celui-ci.
Prenant la convention de placer le noyau SPHINX dans le fichier L.SPHINX, sous forme de programme objet, il suffira alors d'entrer la commande

/EXEC L.SPHINX

Le noyau SPHINX sera chargé et affichera le message

SYSTEM SPHINX V LOADED -<HEURE>:<MINUTES>:<SECONDES>
PLEASE FASTEN YOUR SEAT BELT AND RELAX ...

Son exécution se continuera jusqu'à l'introduction de la commande HALT (ou éventuellement KILL).
Entre ces deux instants, le noyau SPHINX échangera avec l'opérateur du système, messages et commandes sur le déroulement de l'exploitation. Ceux-ci sont décrits au paragraphe 5.
Si le noyau SPHINX a été chargé en BATCH sous le numéro de tâche xxxx, l'arrêt du système doit se faire par une commande (BS2000)

/CAN xxxx

Et on se sera assuré au préalable du déchargement de tous les programmes d'application et du noyau SESAM .
Sans cette précaution, il est possible que les bases de données soient rendues incohérentes.

4.3. Lancement d'un programme DML

Après chargement des noyaux SESAM et SPHINX, un programme DML, compilé et édité comme il a été dit au paragraphe 3, se charge comme un programme BS2000 normal.

4.4. Lancement de l'interpréteur NUL

Le système NUL peut être chargé par un utilisateur conversationnel dès que les noyaux SESAM et SPHINX ont été chargés.
Nous supposerons que les conventions suivantes ont été prises :

- le fichier CDDL.EXTERNAL.SCHEMA contient la description des schémas des différentes bases de données accessibles, telle qu'elle est générée par le compilateur DDL à usage du compilateur DML et de l'interpréteur NUL.
- le fichier L.NUL contient l'interpréteur NUL sous forme objet.

Le lancement du système NUL se fait alors au moyen des 2 commandes :
/FILE CDDL.EXTERNAL.SCHEMA, LINK = SCHREAD
/EXEC L.NUL

A partir de ce moment, le dialogue entre le système NUL et l'utilisateur se déroulera comme indiqué dans la brochure "Manuel de référence des langages".

Remarque : Dans l'implémentation actuelle, il n'est pas possible d'exécuter simultanément une session NUL et une compilation DDL, car le fichier CDDL.EXTERNAL.SCHEMA mis à jour par le compilateur DDL n'est pas partageable.

5. CONTROLE DE L'EXPLOITATION

Lorsque le noyau SPHINX est chargé en conversationnel, il affiche au terminal des messages concernant les principaux événements émaillant son exécution. Il est également permis à l'opérateur du système d'introduire des commandes permettant d'obtenir des renseignements sur l'état de l'exploitation et de modifier dans une certaine mesure son déroulement.

5.1. Messages envoyés à la console système par le noyau SPHINX

5.1.1. Les messages de contrôle (CONTROL-)

Ils renseignent sur certains événements passés ou en cours dans le système, de manière à permettre la surveillance du déroulement de l'exploitation

Format : CONTROL - <message>

<message>: 'DATA BASE <DB name>CLOSED FOR TSN xxxx'
'DATA BASE <DB name>OPEN FOR TSN xxxx'

Ces messages signalent l'ouverture (@OPEN) ou la fermeture (@CLOSE) de la base de données <DB name>par le programme utilisateur travaillant sous le numéro de tâche xxxx.

5.1.2. Les messages correspondant à des incidents mineurs (WARNING-)

Format : WARNING - <message>

<message>: 'TSN xxx IS CANCELLED'
'GARBAGE COLLECTION FOR TSNxxxx'
'SYSTEM SPHINX V ALREADY LOADED'
'I.T.C. NOT CLOSED'
'DATA BASE ALREADY CLOSED'

Le premier de ces messages signale la destruction du TCB correspondant à la tâche de numéro xxxx, suite à une erreur grave détectée dans le programme utilisateur ou à l'introduction d'une commande CAN xxxx par l'opérateur du système. Le troisième message est affiché lorsqu'on tente de charger une deuxième fois le noyau SPHINX. L'exécution de cette deuxième occurrence du noyau se termine, alors, aussitôt. Les autres messages, dus à une défaillance du système, ne doivent pas apparaître en utilisation normale.
Il en est de même des messages des deux catégories suivantes.

5.3.1. Les messages correspondant à des erreur dans la B.D. (DATA BASE-)

Format : DATA BASE - ERROR xxyy IN zzzzzz

zzzzz : nom du module (CSECT) ayant découvert l'erreur

xxyy : numéro de l'erreur construit comme suit :

xx : classe de l'erreur : ØØ : générale
01 : erreur de pointeur vers un bloc
02 : erreur de pointeur vers une réalisation
03 : erreur de cohérence pour une relation
04 :
05 : erreur acquisition schéma
06 : base de données à réorganiser
SE : code SESAM non interprété.

yy : n° de l'erreur dans la classe

Classe ØØ : ØØØ1 : un pointeur ne correspond à aucun enregistrement (lecture) = Ø1Ø1 + Ø2Ø1

Classe Ø1 : Ø1Ø1 : un pointeur ne correspond pas à un bloc (lecture)
Ø1Ø2 : idem (écriture)

Classe Ø2 : Ø2Ø1 : un pointeur ne correspond pas à une réalisation (lecture)
Ø2Ø2 : idem (écriture)

Classe Ø3 : Ø3Ø1 : absence de pointeur inverse dans un bloc pour une occurrence de relation.

5.4.1. Les messages d'erreur dans le système (SYSTEM DUMP-)

Format : SYSTEM DUMP - <message>

<message >:

"ERROR IN DATA BASE HANDLER"
"ERROR IN TCB SCHEDULER
"ERROR IN <ITC macro name >
"ERROR IN REQM"
"NO INITIAL MEMORY SPACE"
"ERROR xx IN zzzzzz"

zzzzzz : nom du module

xx : - de 01 à 09 : erreur propre au module
- autres : reprise erreurs SESAM

5.2. Commandes de contrôle introduites à la console système par l'opérateur (mode conversationnel)

L'opérateur du noyau SPHINX peut interrompre le déroulement normal de l'exécution en poussant sur la touche ESC (T200) ou EXT (vidéo). Il peut alors introduire la commande BS2000 /INTR et, enfin, une commande de contrôle du noyau SPHINX.

Les diverses commandes sont les suivantes :

- USER — Nombre de tâches connectées
- DBNAME — Liste des banques de données et leur état (OPENED, CLOSED, LOCKED).
- TSN ACT — Liste des tâches actives.
- TSN SUS — Liste des tâches suspendues pour accès concurrents
- TSN ALL — Liste de toutes les tâches et de leur état (TCW)
- CAN xxxx — Demande de CANCEL de la tâche numéro xxxx.
- SHUTDOWN — Demande de clôture du système. Toutes les tâches connectées s'achèvent normalement; par contre, tout nouvel utilisateur est refusé.
- KILL — Permet de générer un DUMP du système dans le cas d'une exécution jugée anormale.

On trouve ci-après un exemple d'exécution du système.

```
exec l.sphinxv(045),idm=yes
% C P500 LOADING.

SYSTEM  SPHINX V  LOADED - 23:03:10

PLEASE FASTEN YOUR SEAT BELT AND RELAX ...
VERS 10     SESMOD - SPHINXV

CONTROL - DATA BASE DBTEST3                    OPENED FOR TSN 4211
/intr
ENTER COMMAND - help
ACTUAL TIME IS 23:07:09
USER / DBNAME / TSN ACT / TSN SUS / TSN ALL
CAN XXXX / SHUTDOWN / KILL
/intr
ENTER COMMAND - tsn act
ACTUAL TIME IS 23:09:47
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
4182     230856     230856     ACTIVE
4211     230417     230903     ACTIVE
--- COMMAND TERMINATED ---
/intr
ENTER COMMAND - dbname
ACTUAL TIME IS 23:10:09
DBNAME = FNDP                  OPENED
DBNAME = DBTEST3               OPENED
--- COMMAND TERMINATED ---

CONTROL - DATA BASE FNDP                       OPENED FOR TSN 4182

CONTROL - DATA BASE DBTEST3                    CLOSED FOR TSN 4211

CONTROL - DATA BASE DBTEST3                    OPENED FOR TSN 4214

CONTROL - DATA BASE DBTEST3                    CLOSED FOR TSN 4214

CONTROL - DATA BASE FNDP                       CLOSED FOR TSN 4182

CONTROL - DATA BASE FNDP                       OPENED FOR TSN 4220

CONTROL - DATA BASE DBTEST3                    OPENED FOR TSN 4219

CONTROL - DATA BASE FNDP                       OPENED FOR TSN 4182
/intr
ENTER COMMAND - tsn all
ACTUAL TIME IS 23:33:14
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
4219     232648     232803     00000000
4182     233301     233301     00000000
4220     232358     233306     10010000
--- COMMAND TERMINATED ---

CONTROL - DATA BASE FNDP                       CLOSED FOR TSN 4182
```

```
/intr
ENTER COMMAND - tsn all
ACTUAL TIME IS 23:49:26
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
4219     232648     232803     00000000
4220     232358     234905     10100000
--- COMMAND  TERMINATED ---
/intr
ENTER COMMAND - can 4219
ACTUAL TIME IS 23:49:48
TSN 4219 FLAGGED FOR CANCEL

WARNING - TSN 4219 IS CANCELLED

WARNING - GARBAGE COLLECTION FOR TSN 4219
/intr
ENTER COMMAND - tsn all
ACTUAL TIME IS 23:59:04
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
4220     232358     235830     10100000
--- COMMAND  TERMINATED ---
/intr
ENTER COMMAND - dbname
ACTUAL TIME IS 23:59:38
DBNAME = FNDP                  OPENED
DBNAME = DBTEST3               CLOSED
--- COMMAND  TERMINATED ---
/intr
ENTER COMMAND - tsn act
ACTUAL TIME IS 23:59:54
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
4220     232358     235946     ACTIVE
--- COMMAND  TERMINATED ---
/intr
ENTER COMMAND - tsn act
ACTUAL TIME IS 00:00:57
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
4220     232358     000336     ACTIVE
--- COMMAND  TERMINATED ---

CONTROL - DATA BASE FNDP                CLOSED FOR TSN 4220

CONTROL - DATA BASE FNDP                OPENED FOR TSN 4132

CONTROL - DATA BASE FNDP                CLOSED FOR TSN 4132

CONTROL - DATA BASE FNDP                OPENED FOR TSN 4132

CONTROL - DATA BASE FNDP                CLOSED FOR TSN 4132

CONTROL - DATA BASE FNDP                OPENED FOR TSN 4132

CONTROL - DATA BASE FNDP                CLOSED FOR TSN 4132

CONTROL - DATA BASE FNDP                OPENED FOR TSN 4132
```

```
/intr
 ENTER COMMAND - tsn all
 ACTUAL TIME IS 00:30:27
 TSN      TIMER1     TIMER2     STATUS
 ---      ------     ------     ------
 4182     002652     002306     00000000
 --- COMMAND  TERMINATED ---
/intr
 ENTER COMMAND - can 4182
 ACTUAL TIME IS 00:30:49
 TSN 4182 FLAGGED FOR CANCEL

 CONTROL - DATA BASE DBTEST3            OPENED FOR TSN 4238

 WARNING - TSN 4182 IS CANCELLED

 WARNING - GARBAGE COLLECTION FOR TSN 4182

 CONTROL - DATA BASE FNDP               OPENED FOR TSN 4239

 CONTROL - DATA BASE FNDP               CLOSED FOR TSN 4239

 CONTROL - DATA BASE FNDP               OPENED FOR TSN 4239

 CONTROL - DATA BASE DBTEST3            OPENED FOR TSN 4240
/intr
 ENTER COMMAND - tsn all
 ACTUAL TIME IS 00:39:57
 TSN      TIMER1     TIMER2     STATUS
 ---      ------     ------     ------
 4238     003038     003125     00000000
 4240     003530     003601     10010100
 4239     003332     003235     00000000
 --- COMMAND  TERMINATED ---
/intr
 ENTER COMMAND - can 4240
 ACTUAL TIME IS 00:41:33
 TSN 4240 FLAGGED FOR CANCEL

 WARNING - TSN 4240 IS CANCELLED
/intr
 ENTER COMMAND - can 4238
 ACTUAL TIME IS 00:42:35
 TSN 4238 FLAGGED FOR CANCEL

 CONTROL - DATA BASE FNDP               CLOSED FOR TSN 4239

 WARNING - TSN 4238 IS CANCELLED

 WARNING - GARBAGE COLLECTION FOR TSN 4238

 CONTROL - DATA BASE FNDP               OPENED FOR TSN 4239

 CONTROL - DATA BASE DBTEST3            OPENED FOR TSN 4240

 CONTROL - DATA BASE FNDP               CLOSED FOR TSN 4239

 CONTROL - DATA BASE FNDP               OPENED FOR TSN 4239

 CONTROL - DATA BASE FNDP               CLOSED FOR TSN 4239

 CONTROL - DATA BASE FNDP               OPENED FOR TSN 4239
```

```
SYSTEM  SPHINX V  LOADED - 10:13:22

PLEASE FASTEN YOUR SEAT BELT AND RELAX ...
/intr
ENTER COMMAND - help
ACTUAL TIME IS 10:13:36
USER / DBNAME / TSN ACT / TSN SUS / TSN ALL
CAN XXXX / SHUTDOWN / KILL
/intr
ENTER COMMAND - user
ACTUAL TIME IS 10:13:50
THERE ARE 0000 USERS CONNECTED
/intr
ENTER COMMAND - dbname
ACTUAL TIME IS 10:14:06
---  NO   BATA   BASE   ---
/intr
ENTER COMMAND - tsn act
ACTUAL TIME IS 10:14:22
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
---  NO  ACTIVE  TASK   ---
/intr
ENTER COMMAND - tsn sus
ACTUAL TIME IS 10:14:39
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
---  NO SUSPENDED TASK  ---
/intr
ENTER COMMAND - tsn all
ACTUAL TIME IS 10:14:57
TSN      TIMER1     TIMER2     STATUS
---      ------     ------     ------
---     N O   T A S K    ---
/intr
ENTER COMMAND - can 4045
ACTUAL TIME IS 10:15:20
TSN 4045 NOT FOUND
/intr
ENTER COMMAND - shutdown
ACTUAL TIME IS 10:15:38
S Y S T E M   S H U T D O W N

SYSTEM  SPHINX V  TERMINATED - 10:15:42
THANK YOU - I HOPE TO SEE YOU AGAIN ...
/
```