MASTER EN SCIENCES INFORMATIQUES

Génération automatique d'interfaces d'accès à des bases de données cobol

Robert, Jean-Pol

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

GENERATION AUTOMATIQUE D'INTERFACES

D'ACCES A DES BASES DE DONNEES COBOL.

Mémoire présenté par

Jean - Pol Robert

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

Année académique 1980-1981

Remerciements

Je tiens à remercier, tout d'abord, Monsieur Jean - Luc Hainaut, promoteur de ce mémoire, qui, par ses directives et par sa disponibilité, m'a permis de mener à bien ce travail.

Ma reconnaissance s'adresse également à Monsieur Yves Delvaux dont les conseils ont toujours été bienvenus.

Je tiens aussi à témoigner ma gratitude à Monsieur Jean - Paul Adans pour l'assistance apportée dans l'utilisation du PDP-11 sur lequel a été réalisée l'édition de ce mémoire.

A mon épouse, pour sa patience et sa dévouée collaboration.

TABLE DES MATIERES

TABLE DES MATIERES.

INTRODUCTION

INTRODUCTION

La réalisation de nombreux projets informatiques nécessite, vu la structure et le volume des informations manipulées, un système évolué de gestion de données - SGBD ou SGF (1) -.

Ce système sera, sauf cas exceptionnels, choisi parmi les systèmes existants. Une conséquence non négligeable de ce choix est l'adoption implicite du modèle de données et des langages de description et de manipulation de données de ce système. Or, cette adoption implicite n'est pas sans inconvénients dont le moindre n'est pas d'hypothéquer toute indépendance entre programmes d'application et système de gestion de données.

Une analyse attentive de ces SGBD/SGF montre cependant,que les concepts décrits dans leurs modèles de données sont peu nombreux et se retrouvent presque tous dans chacun d'eux; dès lors un modèle de données unique pourrait suffire pour décrire n'importe lequel d'entre eux. Ce modèle de données, a été développé a l'institut d'informatique: il s'agit du modèle d'acc`s (2). Ce modèle, ou une vision restreinte de celui-ci, nous permet non seulement de décrire des SGBD (CODASYL ou autres) ou des SGF (COBOL par ex.), mais aussi de les manipuler grâce aux primitives qui lui sont associées.

Ce mémoire consiste en l'étude et en la réalisation des moyens qui doivent permettre l'exploitation d'un ensemble de fichiers "COBOL" considérés comme une réalisation - limitée aux possibilités de Cobol - d'un SGBD/SGF ayant pour modèle de données, le modèle d'accès.

C'est dans le cadre du projet I.D.M.L. (3) qu'est né l'intérêt d'une telle étude : I.D.M.L. vise à mettre à la disposition de ses utilisateurs un langage de manipulation de données de haut niveau; cela, dans un contexte multi SGBD/SGF. Pour atteindre ce but, I.D.M.L. devait ,entre autres, avoir, des données gérées par les différents

---

1. SGBD : Système de Gestion de Bases de Données.
   SGF : Système de Gestion de Fichiers.

2. [5] : J.L. HAINAUT " Le modèle d'accès "

3. I.D.M.L. : Interactive Data Manipulation Language
   [3] : J.L. HAINAUT "IDML : an interface for efficient use of Codasyl data bases".
   [8] : J.L. HAINAUT "IDML : système d'interaction avec des banques de données Codasyl".

SGBD/SGF un modèle unique de description : le modèle d'accès; il lui fallait aussi disposer de primitives uniques, applicables aux données gérées par chacun de ces systèmes : les primitives associées au modèle d'accès, regroupées en interfaces spécifiques d'abord à un SGBD/SGF , ensuite à une B.D. (1), ont été adoptées.

L'intérêt de ce mémoire ne se limite pas au projet I.D.M.L. L'adoption du modèle d'accès comme modèle unique de description de données introduit dans tous les cas un degré supplémentaire d'indépendance entre programmes d'application et SGBD/SGF, et permet donc de limiter les conséquences d'une évolution ou d'un remplacement de ce dernier.

Des interfaces d'accès, spécifiques à une B.D., constituent le moyen qui a été retenu pour concrétiser l'adoption du modèle d'accès pour décrire des données gérées par le SGF Cobol. L'objet de ces interfaces sera de traduire les requêtes d'exécution de primitives "modèle d'accès", émises par un utilisateur (I.D.M.L. ou programme d'application), en ordres Cobol.

Les différences entre deux interfaces ne reflétant que celles existant dans le schéma des B.D. qu'ils gèrent, un moyen facile d'obtenir ces interfaces serait de réaliser un système automatique chargé d'adapter les interfaces aux caractéristiques des B.D. qu'ils sont appelés à gérer : il s'agira du système de génération automatique d'interfaces.

Afin de permettre cette génération, il nous a fallu prévoir un moyen pour décrire de façon formelle une B.D. Cobol : un langage de description de données (D.D.L.) adapté aux exigences du Cobol a donc été conçu.

Un second objectif, lié à l'utilisation de l'interface par le système I.D.M.L. a été de fournir à celui-ci une description de la B.D. adaptée à ses exigences.

La structure de ce mémoire reflète la démarche qui a été suivie pour atteindre les objectifs définis ci-dessus.

Nous proposons tout d'abord une définition du modèle de données Cobol en termes de modèle d'accès (chapitre 1) ; basées sur cette définition,les spécifications des interfaces seront précisées dans le chapitre 2, tandis que le langage de description de données sera défini dans le chapitre 3. Le chapitre 4 montrera comment les primitives du modèle d'accès peuvent être exprimées en Cobol. Le schéma des bases de données Cobol se verra adapté au schéma de la base de données des schémas I.d.M.L. dans le chapitre 5. Nous introduirons ensuite le système de génération et son architecture (chapitre 6). Enfin la synthèse et la conclusion du travail font l'objet du dernier chapitre.

---

1. B.D. : Base de Données.

CHAPITRE 1

CHAPITRE 1 LES MODELES DE DONNEES

Dans ce premier chapitre, nous décrivons les modèles de données concernés dans ce mémoire. Après un bref rappel des concepts du modèle d'accès, nous effectuons la description du modèle de données Cobol. Une adaptation de quelques définitions, nous permettra d'aboutir à une définition du modèle d'accès restreint aux possibilités de Cobol.

1.1 Le modèle d'accès (1).

Ce modèle élaboré et mis au point à l'institut d'informatique se veut être, de par la généralité de ses concepts, un modèle de description de données apte à décrire tout ensemble de données, quel que soit le SGBD/SGF qui les gère.

Nous décrivons brièvement ci-dessous les concepts du modèle d'accès, répartis en trois ensembles :

- les structures de données,
- les mécanismes d'accès,
- les primitives.

1.1.1 Les structures de données.

a. La base de données (B.D.).

C'est la collection des articles d'un ensemble de fichiers; elle contient toujours un article particulier (virtuel ou non) qui est son point d'entrée et qui porte le nom de cette base de données.

---

1. Le lecteur se référera pour une description plus détaillée à [5] : J.L. HAINAUT - Le modèle d'accès. Le résumé qui est fait ici s'inspire largement de celui écrit dans [13] : Y. DELVAUX et J.L. HAINAUT - Système portable de manipulation de bases de données hétérogènes.

b. Le fichier.

Un fichier est une collection dynamique d'articles; un article appartient toujours à un fichier.

c. Le type d'article.

Il décrit les propriétés générales des articles qui appartiennent à ce type.

d. L'item.

L'item est un type d'information défini par un ensemble de valeurs. Un item est associé à au moins un type d'article. Il porte un nom qui l'identifie parmi les items d'une B.D.

Un item peut être élémentaire ou décomposable : la valeur d'un item élémentaire est atomique du point de vue de sa signification; la valeur d'un item décomposable est une liste de valeurs significatives; chacune de ces dernières appartient à un item qui est dit composant de l'item décomposable. Un composant peut être lui-même décomposable.

Un item peut être simple ou répétitif : il est dit simple si à chaque article n'est jamais associée plus d'une valeur de cet item : il est répétitif dans le cas contraire.

Un item peut être obligatoire ou facultatif : il est obligatoire pour un type d'article si à chaque article de ce type est associée au moins une valeur de cet item; il est facultatif si il est permis de n'associer aucune valeur de cet item à un article de ce type.

e. L'article.

Unité d'information enregistrée, il peut faire l'objet d'une demande d'accès, de création, de modification ou de suppression. Tous les articles sont distincts et chacun appartient à un seul type d'article.

f. La valeur d'item.

Donnée manipulable par un programme, ses propriétés sont décrites par l'item auquel elle est associée.

g. Notion d'identifiant.

Un identifiant est un item (ou une liste d'items) d'un type d'article tel qu'il n'existe pas, dans le référentiel précisé (ex : B.D. ou fichier), plus d'un article qui soit associé à une même

valeur de cet item (ou des items de cette liste).

h. Clé d'accès.

Une clé d'accès est un item (ou une liste d'items) d'un type d'article tel qu'il existe un mécanisme qui permette d'accéder successivement aux articles auxquels est associée une valeur déterminée de cette clé.

Une clé d'accès peut être identifiante ou non.

Une clé d'accès est caractérisée par le référentiel dans lequel elle porte ses effets (ex : fichier, B.D.).

i. Le chemin d'accès.

Le chemin d'accès est un mécanisme qui associe un article dit origine à 0 ,1 ou plusieurs articles dits cibles, d'une manière telle qu'il soit possible, à partir de son article origine, d'accéder successivement aux différents articles cibles ainsi associés.

j. Type de chemin d'accès.

Tout chemin appartient à un et un seul type qui en définit les propriétés générales. Un type de chemin est caractérisé par les types d'article auxquels doivent appartenir d'une part, les articles origines et d'autres part, les articles cibles de ses chemins.

k. L'article système.

Toute base de données contient un et un seul article d'un type particulier qui porte le nom SYSTEME.

Cet article constitue un point d'entrée privilégié dans la base de données; il peut être origine de chemin d'accès.

l. Chemin d'accès implicite.

Certains systèmes offrent des primitives d'accès séquentiel à des articles, alors qu'aucun chemin d'accès n'a été déclaré pour ces derniers; tel serait le cas de l'accès :

- à tous les articles de la B.D. (1),
- à tous les articles d'un type dans la B.D.,
- à tous les articles d'un fichier,

---

1. B.D. : base de données

- à tous les articles d'un type dans un fichier.

On peut décrire ces possibilités sous forme d'un chemin d'accès (implicite) ayant pour origine l'article SYSTEME.

m. Ordre des articles cibles d'un chemin d'accès.

L'accès aux articles cibles offert par le mécanisme du chemin d'accès étant de nature essentiellement séquentielle (accès aux cibles successives), il convient, dans certains cas, de pouvoir spécifier une règle qui définit l'ordre des articles cibles dans la séquence d'un chemin.

Quatre classes d'ordre peuvent être envisagées :

- quelconque,
- lié au moment de l'insertion : chronologique ou antichronologique,
- trié selon les valeurs d'une clé de tri (croissantes ou décroissantes),
- dynamique : laissé à l'initiative de l'utilisateur.

1.1.2 Les mécanismes d'accès.

L'accès est pour un programme la mise à disposition d'un objet de la base de données. Cette opération n'est possible qu'à travers les mécanismes d'accès que voici :

- l'accès à une base de données
- l'accès à un fichier
- l'accès aux articles d'une base de données, d'un fichier ou d'un type déterminé
- l'accès aux valeurs d'item d'un article
- l'accès aux articles auxquels est associée une valeur déterminée d'un item (clé d'accès)
- l'accès à des articles à partir d'un article (chemin d'accès inter-article).

1.1.3 Les primitives.

Elles correspondent aux opérations élémentaires qu'il est possible d'effectuer sur les objets d'une base de données. Trois classes de primitives peuvent être distinguées :

- les primitives d'accès qui mettent en oeuvre les mécanismes d'accès
- les primitives de modification qui font évoluer l'état de la base de données
- les primitives de contrôle d'environnement qui permettent la création de super-primitives, l'établissement de points de reprise et la maîtrise de la sécurité et de la concurrence.

1.2 Le modèle de données Cobol.

Cobol est un langage de programmation se voulant adapté aux applications de type administratif.

Parce qu'il est un des langages le plus pratiqué, Cobol se voit périodiquement remis à jour. C'est sur la norme publiée en 1974 que se base la description du modèle de données Cobol présentée ci-dessous.

Les interprétations de la norme, faites par les réalisateurs de compilateur (ou les restrictions qu'ils y apportent), ne permettent pas de garantir que cette description est universelle; nous la tiendrons cependant comme telle dans notre travail.

La description du modèle de données Cobol se compose de deux parties : la description des structures de données et celle des mécanismes d'accès qui leur sont associés.

1.2.1 Les structures de données.

a. Les fichiers.

Suivant la norme Cobol, un fichier est toute collection d'enregistrements logiques (= records) susceptible d'être mise à la disposition d'un programme.

A tout fichier est attribué un nom qui l'identifie parmi tous les fichiers connus du programme et qui le référencera lors de toute opération le concernant. La portée de ce nom est limitée au programme ; l'identification d'un fichier parmi tous les fichiers d'un système étant au choix du constructeur, celui-ci doit fournir un mécanisme permettant de réaliser la correspondance entre nom-programme et nom-système d'un fichier.

Cobol reconnaît trois types d'organisation de fichiers qui se distinguent par le mode d'accès aux articles du fichier :

- fichier séquentiel : les articles sont rangés à la suite l'un de l'autre dans l'ordre de leur création et ne sont accessibles que dans cet ordre;

- fichier relatif : les articles sont identifiés par une valeur numérique entière; deux modes d'accès sont possibles :
  mode séquentiel, on accède aux articles dans l'ordre donné par les valeurs croissantes de la clé;
  mode aléatoire, on accède à un article dont la valeur de clé est donnée.

- fichier séquentiel-indexé : les articles sont identifiés par la valeur d'une clé d'accès constituée d'un ou de plusieurs

items (clé primaire). D'autres clés (secondaires) peuvent être définies (identifiantes ou non) sur le même fichier.
Les deux modes d'accès sont possibles :
mode séquentiel : l'ordre suivant lequel on accède aux articles est celui des valeurs croissantes de la clé spécifiée.
mode aléatoire : on accède à un article sur base de la valeur de la clé spécifiée.

b. Le type d'article.

Le type d'article décrit les propriétés des articles qui appartiennent à ce type. Chaque type d'article possède un nom qui l'identifie parmi tous les types d'articles susceptibles d'être contenus dans un fichier. Si, dans un proramme, deux types d'articles ont le même nom, ceux-ci devront être qualifiés par le nom du fichier qui les contient. L'identification du type de l'article lu ou écrit est à charge de l'utilisateur. A un type d'article est toujours associé au moins un item.

c. L'article.

A un article est toujours associée une collection de valeurs des items associés à son type d'article. Un article est une occurrence d'un type d'article et constitue la cible de tout ordre d'accès à des données émis par un programme.

d. L'item.

Un item décrit les propriétés des valeurs d'item qui lui appartiennent; il est associé à un type d'article ou à un item décomposable. Les descriptions des items d'un type d'article se présentent dans l'ordre dynastique de la structure de décomposition. Un item porte un nom qui l'identifie parmi tous ceux ayant la même ascendance dans la structure de décomposition. Un item est toujours obligatoire; il peut être répétitif : de répétitivité fixe (non nulle) ou de répétitivité variable limitée.

e. La valeur d'item.

A un item élémentaire (non décomposable) est associé un ensemble de valeurs élémentaires (valeurs d'item) dont les propriétés sont indiquées dans la description de l'item.

1.2.2 Les primitives.

a. Interface entre SGF Cobol et programme d'application.

L'interface entre SGF Cobol et programme d'application est réalisé de deux façons : tout d'abord, par fichier, une zone de mémoire (décrite par la clause FD de description de type d'article) est accessible au SGF et au programme; elle contiendra successivement les articles du fichier qui sont manipulés par le programme; ensuite, la détection d'événements anormaux par le SGF est signalée au programme qui peut en tenir compte de deux façons : soit dans chaque instruction susceptible de provoquer la survenance d'un tel événement (ex : READ... AT END ou WRITE...INVALID KEY); soit dans une section générique de traitement d'erreur (USE...).

b. Connexion et déconnexion des fichiers.

- OPEN : ouverture d'un fichier

L'ordre OPEN s'assure de l'existence et de la disponibilité d'un fichier; si tel est le cas, il fixe le pouvoir du programme sur le fichier et positionne les organes d'accès au début du fichier (sauf EXTEND). Le pouvoir d'un programme sur un fichier peut consister en :

  - lecture seule (INPUT)
  - écriture seule (OUTPUT ou EXTEND)
  - lecture et écriture (I-O)

L'ordre OPEN n'effectue aucun accès à un article, mais doit être effectué avant tout accès aux articles d'un fichier.

Remarques valables pour les fichiers séquentiels :

l'option EXTEND n'existe que sur ces fichiers; elle permet l'ajoût d'articles après ceux existant déjà dans le fichier; l'option OUTPUT supprime tous les enregistrements existants; l'option I-O n'est possible que si le fichier se trouve sur support adressable.

- CLOSE : fermeture d'un fichier

L'ordre CLOSE ôte à un programme tout pouvoir à un fichier; après un ordre CLOSE, seule l'exécution d'un ordre OPEN rendra ce fichier de nouveau accessible au programme.

N.B. La norme Cobol ne spécifie pas qu'un fichier ouvert par un programme ne puisse l'être par un autre. Le soin de gérer les concurrences est laissé au système d'exploitation, hôte du Cobol.

c. Accès aux articles.

1. Notion de mode d'accès.

Le mode d'accès définit l'ordre dans lequel il sera accédé aux articles d'un fichier.

Cobol définit trois modes d'accès aux articles des fichiers:

- mode d'accès séquentiel :

a. Ecriture :

Lors de la création d'un fichier en mode séquentiel, la succession des articles doit être la suivante :

- fichier séquentiel : les articles peuvent être fournis dans un ordre quelconque;
- fichier séquentiel-indexé : les articles doivent etre fournis suivant l'ordre croissant des valeurs de la clé primaire;
- fichier relatif : l'ordre de succession est quelconque, mais ils se voient attribuer séquentiellement un numéro d'ordre par le système de gestion de fichier;

b. Lecture.

La lecture séquentielle des articles se fait dans l'ordre suivant :

- fichier séquentiel : l'ordre dans lequel les articles sont lus, est l'ordre de leur écriture;
- fichier relatif : suivant l'ordre croissant des valeurs de clé;
- fichier séquentiel-indexé : suivant l'ordre croissant des valeurs de la clé utilisée pour la lecture;

- mode d'accès aléatoire :

Dans toute opération en mode d'accès aléatoire, la valeur de la clé d'accès identifie l'article à traiter.
Ce mode n'a aucun sens pour les fichiers séquentiels et est donc interdit.

- mode d'accès dynamique :

  Ce mode est la conjonction des deux modes precèdents. Il ne s'applique qu'aux fichiers sèquentiels-indexès et relatifs.

remarque : le tableau 1.2.2.1 (cfr. infra) rèsume les possibilitès d'opèrations offertes par Cobol sur chacun des fichiers en fonction du mode d'accès et du type d'ouverture realisèe.

TABLEAU RECAPITULATIF

Ce tableau montre les fichiers sur lesquels les opèrations d'accès sont exècutables, en fonction du mode d'accès et du type d'ouverture.

tableau 1.2.2.1

		Instructions	Open Status			
		Cobol	IN	OUT	I-O	EXT
A		READ (NEXT) *	SRI		SRI	
C	S	WRITE		SRI		S
C	E	REWRITE			SRI	
E	Q	DELETE			SRI	
S		START *	RI		RI	
M	R	READ *	RI		RI	
O	A	WRITE *		RI	RI	
D	N	REWRITE *		RI	RI	
E	D	DELETE *			RI	
					(1)	

organisations : S : sèquentielle
R : relative
I : sèquentielle-indexèe

(1) I-O : seulement pour fichiers sur support adressable
* : si access mode dynamic

2. Lecture d'un article : READ

L'ordre READ a pour effet de vérifier l'existence puis de fournir au programme les valeurs d'item d'un article du fichier.

a. Lecture séquentielle.

L'ordre de lecture séquentiel est celui défini dans le mode d'accès séquentiel (cfr. ci-dessus). Lorsqu'un ordre READ est lancé, après la lecture du dernier article du fichier, le système de gestion des fichiers Cobol retourne au programme une valeur le signifiant (AT END ou FILE STATUS).

b. Lecture aléatoire.

L'article lu sera celui dont la valeur de clé est égale à celle fournie par le programme au système de gestion de fichier. Si aucun article ne peut être trouvé, une valeur le signalant sera retournée au programme.

3. Ecriture d'un article : WRITE.

L'ordre WRITE écrit un article dans un fichier.

a. Ecriture séquentielle : suivant l'ordre défini dans le mode d'accès séquentiel.

b. Ecriture aléatoire : la valeur de la clé doit avoir été communiquée au SGF avant l'écriture.

- fichiers relatifs : une tentative d'écriture d'un article dont la valeur de clé est déjà attribuée à un article existant se soldera par le retour d'une valeur indiquant ce cas; aucune écriture ne sera faite;

- fichiers séquentiels-indexés : même chose que pour les fichiers relatifs, mais seulement pour la clé primaire du fichier (identifiante);

4. Remplacement d'un article : REWRITE.

a. Accès séquentiel :
l'article fourni ira remplacer dans le fichier le dernier article lu.

b. Accès aléatoire :
une valeur de la clé d'accès (primaire pour les fichiers séquentiels-indexés) doit être fournie au système de gestion de fichier par le programme. Une tentative de remplacement d'un article non existant

dans le fichier (si la valeur de clé fournie n'en identifie aucun) se verra signalée et aucun remplacement n'aura lieu.

5. Suppression d'un article : DELETE.

Cet ordre effectue la suppression logique d'un article d'un fichier; il devient inaccessible pour toute opération ultérieure.

Les règles d'identification d'un article sont identiques à celles de l'ordre REWRITE.

6. Positionnement d'un fichier en lecture séquentielle : START.

L'ordre START permet de vérifier l'existence d'un article (dans un fichier relatif, dans un séquentiel-indexé) dont la valeur de clé vérifie une des conditions suivantes :

clé-article-fichier = valeur clé fournie par programme
clé-article-fichier > valeur clé fournie par programme
clé-article-fichier not < valeur clé fournie par programme

Si au moins un article satisfaisant la condition existe et si un ordre READ séquentiel est exécuté immédiatement après le START, le READ fournit cet article; si plusieurs articles existent, l'ordre READ fournit le premier de ceux-ci, dans l'ordre de la création. Si aucun article ne satisfait la condition, le fait sera signalé au programme par le retour d'une valeur significative.

1.3 Définitions complémentaires.

Dans la description du modèle Cobol que nous venons de faire, figurent certains éléments dont une description différente qui n'ôterait rien à leur spécificité, permettrait une intégration plus aisée dans une approche du type modèle d'accès.

Inversement, certains éléments du modèle d'accès n'ont aucun équivalent en Cobol; pourtant l'extension du modèle Cobol à ces éléments ne diminuerait en rien les possibilités de Cobol et offrirait aux utilisateurs une plus grande souplesse d' utilisation.

1.3.1 Redéfinition.

a. Le fichier relatif.

Il peut être considéré comme un fichier sur lequel ne peut être définie qu'une seule clé d'accès qui doit être numérique et identifiante.

Afin d'uniformiser l'utilisation des fichiers permettant l'accès par clé, nous considérerons que cette clé d'accès est constituée par un item fictif, toujours placé en tête des articles des fichiers relatifs.

b. OPEN des fichiers séquentiels.

- OPEN OUTPUT est équivalent à l'application des deux primitives suivantes :

  1. suppression de tous les articles du fichier

  2. ouverture du fichier pour mise à jour

- tandis qu'OPEN EXTEND ne correspond qu'à la seconde. Si le fichier est inexistant, OPEN OUTPUT effectue dans de nombreux systèmes la création du fichier; dans le cas de l'OPEN EXTEND, les réactions des systèmes sont moins unanimes.

1.3.2 Extensions du modèle Cobol.

a. La base de données.

Les structures de données Cobol ne dépassent pas le niveau des fichiers; on ne peut donc directement dériver du modèle Cobol une structure équivalente. Nous introduisons cependant une définition du concept "Base de données" applicable dans le cadre du modèle d'accès Cobol : une base de données sera constituée de tout ensemble de fichiers pour lesquels un administrateur de B.D. aura - pour des raisons d'organisation ou de fonctionnement - jugé opportun de disposer d'un interface unique qui prendra en

charge tous les accès à ces fichiers.

Un nom et un mot de passe seront attribués par l'administrateur de la base de données. Tout accès à la base de données ne sera autorisé que si nom et mot de passe sont corrects.

b. Mécanismes d'accès à la B.D.

Le concept B.D. n'étant pas défini en Cobol, les mécanismes d'accès ne le sont pas davantage. Ces mécanismes seront mis en oeuvre par deux primitives supplémentaires :

- ouverture d'une B.D.
- fermeture d'une B.D.

c. Accès aux articles d'un type.

Cobol n'identifie pas le type des articles d'un même fichier qui sont lus ou écrits; charge en est laissée à l'utilisateur. L'extension des possibilités de Cobol en cette matière a été jugée opportune. Elle nécessite un moyen de définir pour un type d'article l' (les) item(s) (et sa (ses) valeur(s)) qui permet(tent) sa reconnaissance dans les articles lus ou écrits. Deux primitives mettent en oeuvre ce mécanisme d'accès :

- accès séquentiel aux articles d'un type,
- accès par clé aux articles d'un type.

1.4 Le modèle d'accès Cobol.

Après avoir rappelé les concepts du modèle d'accès (1.1) et défini le modèle de données Cobol (1.2) en aménageant certaines définitions (1.3), nous procédons ci-dessous à la description du modèle d'accès restreint aux possibilités du Cobol.

Nous retrouvons les trois classes de concepts :

- les structures de données,
- les mécanismes d'accès,
- les primitives.

1.4.1 Les structures de données.

a. Base de données.

Une base de données possède un nom qui l'identifie parmi toutes les bases de données disponibles. Elle est constituée d'un ensemble de fichiers.

b. Fichiers.

Un fichier peut contenir des articles; il possède un nom qui l'identifie parmi tous les fichiers accessibles au processus.

c. Type d'article

Les types d'article ont un nom qui les identifie parmi tous les types d'article de la B.D.

d. Articles.

Un article appartient à un type qui en décrit ses propriétés. Un fichier peut contenir des articles de plusieurs types; mais les articles d'un type sont tous contenus dans le même fichier. A un article est toujours associée une collection de ses valeurs d' items.

e. Item.

L'item est un type d'information défini par un ensemble de valeurs.

A un type d'article peut être associé un ou plusieurs items. Un item porte un nom qui l'identifie parmi tous les items de la B.D. Un item peut être élémentaire ou décomposable; il peut être simple ou répétitif (de répétitivité non nulle et limitéee); il est toujours obligatoire. Un item est déclaré comme constituant

d'un type d'article ou d'un item décomposable. Il peut aussi être déclaré comme identifiant de type d'articles.

f. Valeur d'item.

A tout article est associée une collection de valeurs appartenant aux items associés au type de l'article : ce sont ses valeurs d'items. L'item décrit les propriétés des valeurs qui lui appartiennent.

g. Clé d'accès.

Le référentiel d'une clé d'accès est toujours un fichier. Une clé d'accès est un item associé à tous les types d'article qui peuvent appartenir au fichier. La clé doit cependant être définie pour les différents types d'articles sur des items comparables. Plusieurs clés d'accès peuvent être définies sur un fichier; cependant l'une d'entre elles doit obligatoirement être identifiante.

h. Chemins d'accès implicites.

L'exploitation de deux types de chemins d'accès implicites est possible; ce sont :

- l'accès à tous les articles d'un fichier
- l'accès à tous les articles d'un type dans un fichier

i. Ordre des articles associés à une clé d'accès.

Pour les fichiers sur lesquels aucune clé d'accès n'est définie, l'ordre des articles est celui de leur création (ordre chronologique).

Pour les autres fichiers, les items qui ont été déclarés clés d'accès sont aussi des clés de tri pour l'accès séquentiel, et l'ordre dans lequel on accède aux articles est celui des valeurs croissantes de la clé de tri considérée.

j. Confidentialité.

Il est permis de définir pour la base de données, un mot de passe; celui-ci devra être fourni préalablement à tout accès à la B.D.

1.4.2 Les mécanismes d'accès.

Les mécanismes d'accès du modèle d'accès Cobol sont les suivants :

- l'accès à une B.D.,

- l'accès à un fichier,
- l'accès aux articles d'un type,
- l'accès aux articles d'un fichier,
- l'accès aux valeurs d'item d'un article, l'accès aux articles auxquels est associée une valeur déterminée d'une clé d'accès.

1.4.3 Les primitives.

Les primitives que le modèle d'accès Cobol est à même d'offrir, sont énumérées ci-dessous; leurs spécifications précises seront détaillées au chapitre 2.

a. Les primitives d'accès.

- à la B.D. :
  - ouverture de la B.D.
  - fermeture de la B.D.
- aux fichiers :
  - ouverture d'un (de tous les) fichier(s)
  - fermeture d'un (de tous les) fichier(s)
- aux articles :
  - accès séquentiel aux articles d'un type
  - accès par clé aux articles d'un type
  - accès séquentiel aux articles d'un fichier
  - accès par clé aux articles d'un fichier
- aux valeurs d'item d'un article

b. Les primitives de modification.

- ayant pour cible un fichier
  - suppression de tous les articles d'un fichier
- ayant pour cible un article
  - création d'un article
  - suppression d'un article
  - modification des valeurs d'item d'un article

CHAPITRE 2

CHAPITRE 2 LES INTERFACES

Avant d'aborder les spécifications détaillées des interfaces, nous avons tenu à définir le contexte dans lequel se sont déroulées leur étude et leur réalisation; ceci afin de permettre au lecteur d'en percevoir l'opportunité et d'évaluer les solutions présentées dans le cadre de ce mémoire.

2.1 Les interfaces - Leur contexte.

2.1.1 Définitions.

Nous introduisons ici deux définitions nécessaires à la compréhension de ce qui suit :

- dans le cadre de ce travail, le mot interface désigne un programme, spécifique à une base de données, chargé de réaliser toutes les requêtes émises par des utilisateurs, qui concernent cette base de données. Une requête consiste en une demande d'exécution d'une des primitives définies dans le modèle d'accès.

- un utilisateur sera, pour un interface, tout programme émettant des requêtes concernant la base de données qu'il gère.

2.1.2 Le contexte: I.D.M.L. (1) et son processeur base de données.

Le projet I.D.M.L. vise à offrir à ses utilisateurs un langage de haut niveau lui permettant de manipuler des informations contenues dans des bases de données gérées par des SGBD/SGF (2) hétérogènes.

La machine I.D.M.L. est constituée de plusieurs processeurs; parmi ceux-ci, le processeur bases de données chargé spécifiquement d'effectuer toute requête concernant une base de données.

---

1. I.D.M.L. : Interactive Data Manipulation Language

2. SGBD : Système de Gestion de Bases de Données.
   SGF : Système de Gestion de Fichiers.

Afin d'uniformiser les requêtes, le modèle d'accès a été retenu pour décrire les bases de données indépendamment du SGBD/SGF qui les gère, et les actions primitives sont celles définies dans ce modèle.

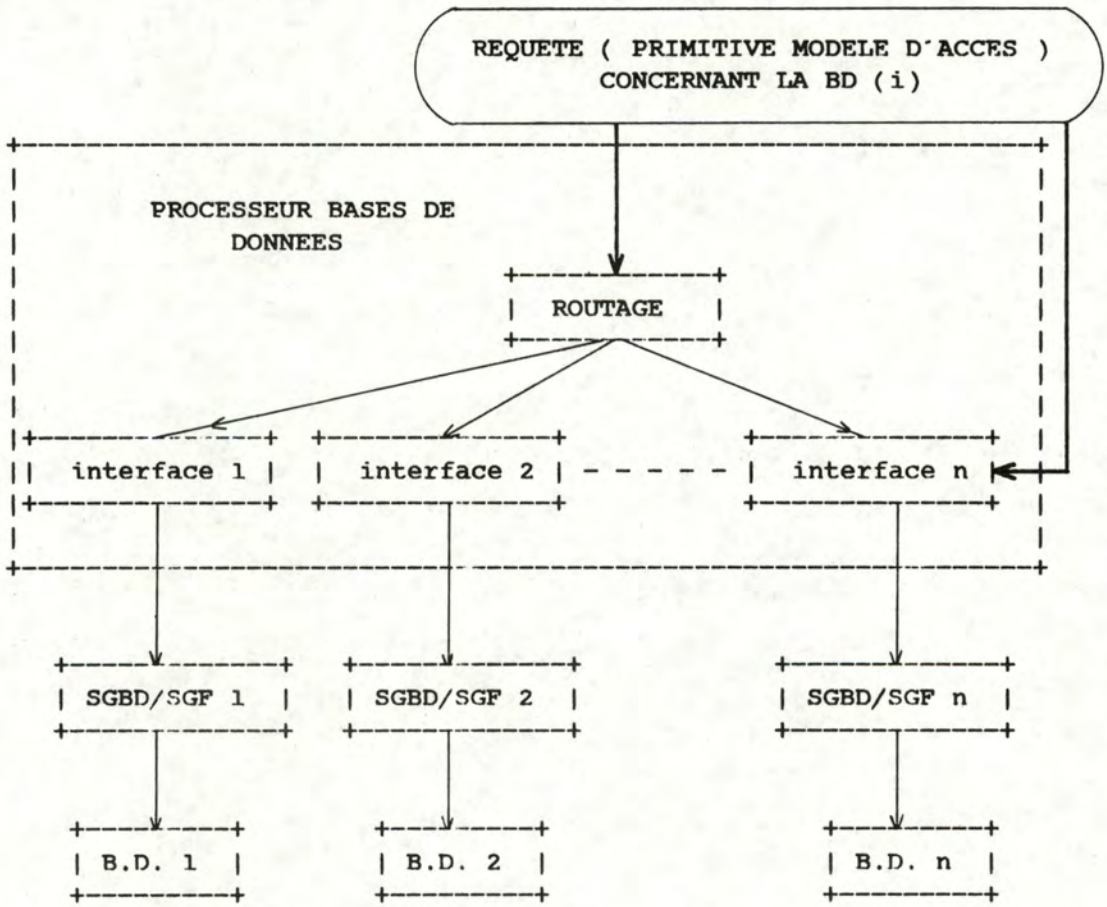


fig. 2.1.2.1.

La figure 2.1.2.1. présente une vision schématique du processeur bases de données; le fonctionnement en est le suivant: une requête est transmise au processeur B.D. A son entrée, la requête est prise en charge par une fonction qui l'oriente vers l'interface chargé de la gestion de la B.D. concernée par la requête. Celui-ci exécute la requête (primitive modèle d'accès) au moyen des ordres du langage de manipulation de données du SGBD/SGF gérant réellement la B.D. Il est à noter que ces SGBD/SGF peuvent être aussi bien identiques que différents.

Le rôle de l'interface est donc de présenter à l'utilisateur (dans ce cas, la machine I.D.M.L.) un modèle de données et un langage de manipulation de données uniques, quels que soient les SGBD/SGF gérant les bases de données.

L'étude des interfaces, présentée dans ce chapitre, considère que ceux-ci sont autonomes : aucune contrainte n'est imposée sur l'environnement dans lequel ils sont appelés à être utilisés : que l'on se trouve dans un contexte multi B.D. (cas du processeur B.D. d'I.D.M.L.) ou mono B.D., l'interface et son utilisation sont inchangés. L'interface, composant du processeur B.D., n'est chargé que de la gestion d'une B.D.; c'est au niveau de la fonction de routage que l'existence de plusieurs B.D. est connue et assumée.

Ces considérations nous amènent à envisager un second cas d'utilisation possible d'un interface : celui où la requête lui serait transmise directement par l'utilisateur.

2.1.3 Les interfaces : contexte général.

Les interfaces, même hors du contexte du processeur B.D. de I.D.M.L., constituent le moyen de matérialiser l'avantage offert par un modèle unique de description des SGBD/SGF tel que le modèle d'accès.

Tout programmeur familiarisé au modèle d'accès, sans connaître les particularités de tel ou tel SGBD/SGF, pourra, s'il dispose de la description en termes du modèle d'accès d'une B.D., y accéder au travers d'un interface.

Un avantage de l'utilisation de ces interfaces, conséquence de ce qui précède, est l'introduction d'un degré d'indépendance supplémentaire entre programmes et données : si le SGBD/SGF, gérant effectivement les données change, la mise au point d'un interface adapté au nouveau SGBD/SGF suffira pour garantir la continuité des traitements; cela pour autant que les possibilités de ce nouveau SGBD/SGF ne soient pas inférieures à celles du précédent.

2.2 Spécification des interfaces.

Après avoir présenté dans un cadre général la genèse des interfaces, nous allons en définir les spécifications. Nous décrirons les primitives d'accès et de manipulation mises à la disposition des utilisateurs.

Le SGF-cible étant le Cobol, toutes les primitives ne seront pas implémentées ; cependant nous les décrirons toutes et nous signalerons celles n'ayant aucun équivalent en Cobol.

2.2.1 Utilisation et notions particulières.

a. Utilisation des interfaces.

Un interface est invoqué par un appel; un appel correspond à une demande d'exécution d'une primitive. Les paramètres et zones de transmission d'informations sont transmis au moment de l'appel.

Nous estimons (sur base des prescriptions de méthodes de structuration de programmes aussi bien que sur base de nos observations) que la séquence des opérations sur B.D. qu'un module utilisateur adoptera le plus fréquemment sera :

- ouverture de la B.D.
- ouverture(s) des fichier(s)
- mise à jour / consultation des fichier(s)
- fermeture des fichier(s)
- fermeture B.D.

Afin de permettre la standardisation de ces modules, et d'éviter toute contrainte sur l'architecture qui les structure (imbrication, simultanéité) la possibilité d'ouverture multiple de la B.D. et des fichiers a été prévue dans les interfaces.

b. Notion de référence à un objet.

La référence à un objet est un identifiant associé automatiquement par l'interface à tout objet auquel on a accédé avec succès, et qui permet de le désigner lors de toute demande ultérieure le concernant. Les objets susceptibles de se voir attribuer une référence sont les bases de données, les fichiers et les articles.

La référence B.D. est attribuée de façon différente des autres : l'interface s'attend à recevoir cette référence lors de la première demande d'ouverture de la B.D.; il ne l'attribue pas lui-même. Cette référence, qui permet d'identifier une B.D. parmi d'autres, ne joue pleinement son rôle que dans un contexte multi B.D.

c. Durée de vie d'une référence à un objet.

Toute référence disparaît avec le processus durant l'exécution duquel elle a été attribuée.

Pendant l'exécution du processus, une référence attribuée ne le sera pas une seconde fois.

d. Limitations dans l'utilisation de la référence.

Les possibilités d'accéder à un objet sur base de sa référence, vont dépendre du SGBD/SGF-cible et des options prises lors de le spécification de l'interface. Ainsi, dans le cas des interfaces Cobol, la référence d'un article ne permet d'y faire accès que, pour autant qu'aucun autre accès n'ait été réalisé aux articles du fichier qui contient cet article. Il est à noter qu'une option différente, non implémentée, permettrait de réaliser l'accès par référence à tout moment à un article d'un fichier pour lequel une clé d'accès a été définie : il suffirait que l'interface assure la gestion d'une table de correspondance entre références et clés primaires du fichier.

Une ambiguité se produit lorsque l'on procède à des suppressions d'article sur base de la référence d'article : l'article supprimé, sa référence n'en est plus une; si l'on veut accéder à l'article suivant celui supprimé, on considérera que la référence de l'article supprimé peut encore être utilisée comme référence de l'article précédent.

Cette convention n'est pas neuve et solutionne ce problème classique dans d'autres SGBD/SGF; nous n'en prendrons comme exemple que celui des SGBD de type CODASYL : l'ordre " FIND NEXT OF SET " n'aurait aucun sens après la suppression de l'article courant du set, si la même convention n'avait été adoptée.

e. Notion de code de type d'objets.

Deux possibilités de désignation ont été envisagées : soit le nom complet, soit un code.

Les deux possibilités ont été utilisées dans les interfaces : B.D. et fichiers seront désignés par leur nom tandis que types d'article, types de chemin, items et clés d'accès le seront par un code.

La fréquence d'utilisation de ces désignations explique leur choix : B.D. et fichiers sont désignés uniquement lors de leur ouverture; les autres éléments sont d'un usage beaucoup plus fréquent et la manipulation de leur nom complet a été jugée par trop contraignante (place, performance) aussi un code leur sera attribué.

Ce code, peut être soit attribué automatiquement par le système générateur, soit laissé au choix de l'administrateur de la base de données, qui disposera dans le D.D.L (Data Description Language), de clauses lui permettant de fixer les codes des types d'objet (en Cobol : types d'articles et clés d'accès). Cette possibilité a été laissée, afin d'offrir aux programmes utilisateurs une meilleure résistance aux modifications de la base de données. En effet, si le système attribue lui-même les codes, on peut difficilement imaginer le lui voir faire autrement que séquentiellement, vu qu'il ne disposera d'aucun élément d'information lui permettant d'estimer l'évolution de la B.D. Toute insertion dans la B.D. d'un type d'article (par exemple) provoquera donc une modification du code de ceux qui le suivent; donc imposera une modification des programmes utilisateurs.

2.2.2 Les paramètres d'appel - leur signification et leur utilisation.

Les interfaces sont susceptibles de réaliser de nombreuses primitives; encore faut-il que leur soient communiquées les informations nécessaires à l'exécution de celles-ci. C'est de ces paramètres que nous entreprenons maintenant la description.

Nous pouvons distinguer cinq types particuliers de paramètres :

- Les paramètres précisant l'opération à effectuer et les objets sur lesquels elle s'applique :
  nom générique : Z-CODES.

- Les paramètres permettant le transfert de valeurs d'identifiants ou de valeurs d'objets de l'utilisateur vers l'interface : nom générique : Z-IDENT/Z-VALUES.

- Des paramètres particuliers prévus pour la transmission d'une liste de codes d'items : nom générique : Z-ITEMS.

- Des paramètres utilisés pour la transmission des valeurs d'items auxquelles un accès a été demandé (de l'interface vers l'utilisateur) :
  nom générique : Z-RESP.

- Des paramètres qui concernent les chemin d'accès :
  nom générique : Z-SETS.

Nous présenterons maintenant chacune de ces classes en ne détaillant que les paramètres retenus pour les B.D. Cobol. Le lecteur trouvera en annexe A, le type et le format précis de chacun de ces paramètres.

1. Z-CODES : cette classe comprend :

- COP : code opération,
  un code opération a été atribué à chaque primitive; lors de tout appel, COP doit contenir l'un de ces codes; si tel n'est pas le cas, aucune primitive ne sera exécutée. Nous signalerons, pour chaque primitive, la valeur attribuée.

- SREF : référence du schéma (de bases de données),
  cette référence qui identifie la B.D.(1), est fournie à l'interface par l'utilisateur lors du premier appel (ouverture base de données); SREF devra contenir cette même référence lors de tous les appels successifs, sinon, aucune action ne sera faite par l'interface. Cas des ouvertures successives : lors d'une ouverture B.D., autre que la première, SREF sera garni par l'interface, avec la référence qui lui aura été fournie lors de la première ouverture B.D.

- COREC : code de type d'articles,
  COREC doit contenir un code numérique attribué à un type d'article, lors de la demande d'exécution d'une primitive qui requiert parmi ses paramètres, une identification de type d'articles.

- RETCODE : code de retour,
  contiendra après l'exécution d'une requête par l'interface, le diagnostic de cette exécution. La valeur 0 signifiera une exécution normale. Des valeurs différentes de 0 signifieront la détection d'une anomalie (paramètres incorrects, requête impossible à satisfaire..). Les valeurs attribuées et leur signification figurent en annexe B.

- PROTECT : code de protection de fichier,
  deux interprétations de ce paramètre existent :

  a. en ouverture de fichier,
  il permet de préciser la protection (aucune, protégée, exclusive) désirée par un utilisateur sur un fichier lorsqu'il ouvre celui-ci. On associe souvent à cette notion de protection, celle d'intention de travail du programme sur le fichier (consultation/mise à jour).

  Les valeurs de PROTECT peuvent être dans ce cas :
  1 : -
  2 : consultation
  3 : mise à jour
  4 : consultation protégée
  5 : mise à jour protégée
  6 : consultation exclusive
  7 : mise à jour exclusive
  L'interface Cobol ne reconnaît que les valeurs 2 et 3.

---

1. B.D. : base de données.

b. en ouverture de la base de données,
les valeurs reconnues sont 1 et 2 :

- 1 signifie mode d'ouverture normal des fichiers : on ne peut accéder à aucun article de la B.D. sans avoir ouvert le fichier qui le contient.

- 2 signifie mode automatique d'ouverture des fichiers; les accès à des articles sont permis sans ouverture préalable du fichier. Cette possibilité permet des accès rapides occasionnels avec un protocole simplifié; en fait, elle permet d'ignorer la notion de fichier.

- COGET : code d'accès aux valeurs d'items d'un article.
Ce paramètre permet lors d'un accès à un article de signaler à l'interface si l'on désire accéder aux valeurs d'item de cet article;
3 valeurs sont prévues :
si COGET = 0 : pas d'accès aux valeurs d'items
si COGET = 1 : accès à toutes les valeurs d'items
si COGET = 2 : accès aux valeurs d'items dont les codes des types se trouvent dans Z-ITEMS.
L'interface Cobol ne reconnaît que les valeurs 0 et 1 vu que les mécanismes d'accès aux items n'ont pas été développés au delà des posibilités de Cobol. L'intérêt de ces mécanismes dont le développement aurait pris trop de temps, réside dans l'introduction d'un degré d'indépendance supplémentaire entre programmes d'application et SGF. On se réfèrera à ce sujet, à la désignation des éléments de la B.D. (2.2.1e).

- CONTRL : contrôle d'article,
permet de préciser le contrôle que l'interface doit exercer sur l'article auquel on accède. Une étude ultérieure de ces mécanismes généralisée à d'autres modèles de données, permettra de préciser davantage le rôle de ce paramètre.

- RFIL : référence de fichier
destiné à recevoir une référence de fichier lors de l'appel des primitives qui exigent une telle référence;
RFIL est garni par l'interface lors de l'ouverture du fichier; toute requête ultérieure concernant ce fichier devra fournir cette référence.

- RREF : référence d'article
après un accès réussi à un article, RREF contiendra la référence de cet article (1); RREF devra être garni par l'utilisateur pour l'appel des primitives qui requièrent une telle référence.

- PREF : référence de l'article précédent :
PREF est utilisé pour l'accès aux articles; il doit contenir la référence du dernier article auquel on a accédé.

---

1. cfr. 2.2.1 : b,c et d

Si PREF = 0, on accèdera au premier article (d'un type ou d'un fichier).

- COKEY : code clé d'accès :
  COKEY doit contenir le code d'une clé d'accès lors de l'appel des primitives requérant une clé d'accès; lorsque RFIL et COKEY figurent simultanément dans une liste d'appel, COKEY ne peut contenir que le code d'une clé d'accès définie sur le fichier dont la référence est contenue dans RFIL.

- OPERAT : opérateur,
  permet, associé à Z-CLE, de préciser une condition sur la clé de l'article auquel on accède par clé. La condition est "clé article accédé OPERAT Z-CLE".
  Les valeurs reconnues pour OPERAT sont :
  0 = pas de condition
  1 = "="
  2 = ">"
  3= "not < "
  D'autres possibilités existent (ex : appartenance à un intervalle), mais, n'étant pas utilisées dans les interfaces Cobol, elles ne seront pas définies ici.

- COMOD : en réserve.

- COSET : code d'un type de chemin ,
  contient le code d'un type de chemin d'accès inter-articles, lors de l'appel des primitives qui en exigent un. Ce paramètre est inutilisé par l'interface Cobol (il n'existe pas de mécanismes d'accès inter-articles en Cobol).

  OREF : référence d'article origine d'un chemin (ou cible de chemin),
  contient la référence de l'article origine d'un chemin d'accès lors de l'appel des primitives faisant intervenir des mécanismes d'accès inter-articles.

- TYP : en réserve.

2. Z-IDENT/Z-VALUE : nous trouvons dans cette classe de paramètres :

- SSNAME : nom de la base de données ;
  ce nom est à fournir lors de la demande d'ouverture de la B.D. Il doit contenir le nom de la B.D. gérée par l'interface; dans le cas contraire, l'accès à la B.D. sera refusé.

- PSW : mot de passe;
  mot de passe à fournir lors de la demande d'ouverture de la B.D. S'il ne correspond pas à celui défini pour la B.D.

gérée par l'interface, aucun accès ne sera permis.

- FILNAME : nom de fichier;
  nom à fournir lors de la demande d'ouverture d'un fichier; ce nom doit être celui de l'un des fichiers apartenant à la B.D. gérée par l'interface.

- Z-VALIT : valeur d'item d'un article,
  destiné à contenir les valeurs d'items des articles en cas de création d'un article, ou de modification de valeurs d'items d'un article.

- Z-CLE : valeur de clé d'accès;
  lors des accès par clé, la valeur de la clé dont le code est contenu dans COKEY, figurera dans ce paramètre.

3. Z-ITEM : contient une liste de codes d'items.

- Cette liste est destinée à définir le contenu de Z-VALIT et Z-RESP, les codes d'items correspondent successivement aux items dont les valeurs sont présentes dans Z-VALIT ou dans Z-RESP. Cette zone n'est pas utilisée par l'interface Cobol; les mécanismes de manipulation de valeurs de l'interface possèdent une granularité manquant de la finesse requise.

4. RFIELD.

- RFIELD est destiné à permettre le transfert de l'interface vers l'utilisateur, des valeurs auxquelles celui-ci accède.

5. Z-SETS : cette classe de paramètre est ignorée par les interfaces Cobol.

- STKREF : référence de stockage;
  destiné à recevoir un paramètre de stockage physique, nécessité par certains SGBD (ex. CODASYL : location mode direct).

- SETLST : liste de codes de types de chemin d'accès;
  cette liste contient des codes permettant l'identification des types de chemin auxquels appartiennent les chemins, auxquels sont associés les articles dont les références sont dans CURLST, utilisée lors de la création d'un article (STORE).

- CURLST : liste de références d'articles;
  les références contenues dans CURLST sont celles des derniers articles associés à des chemins, auxquels on ait

accède. Les codes des types de ces chemins sont dans SETLST; CURLST est utilisé pour les créations d'articles.

2.2.3 Les primitives.

Nous abordons la spécification détaillée de chacune des primitives qu'un interface est susceptible de mettre à la disposition des utilisateurs. Tous les SGBD/SGF ne les offrent pas toutes; par exemple, les interfaces Cobol ne seront pas en mesure d'exécuter les primitives mettant en oeuvre des mécanismes d'accès inter-articles; ces mécanismes n'existant pas en Cobol. Les primitives non-implémentées en Cobol, ne seront décrites que brièvement.

Conventions.

Pour chacune des primitives, nous donnons dans l'ordre : les paramètres d'entrée et de sortie, et la fonction. Nous indiquons les diagnostics d'erreurs susceptibles d'être transmis à l'utilisateur par l'interface (RETCODE).

L'ordre dans lequel ces diagnostics sont renseignés indique pour chaque primitive, l'ordre dans lequel ces vérifications sont faites par l'interface. Unseul diagnostic étant transmis par appel à l'interface, si plusieurs sont possibles, celui qui figure le plus tôt dans la liste, sera seul transmis.

Certains diagnostics, établis à l'entrée de l'interface, sont communs à toutes les primitives.

- " SREF invalide " si SREF ne contient pas la référence de la base de données qui a été fournie à l'interface lors de la première ouverture de la B.D. Ce diagnostic est valable pour toutes les primitives sauf " ouverture de la B.D.".

- " B.D. non ouverte " si un appel aure que "ouverture B.D. " est transmis alors que la B.D. est fermée.

- " primitive non impleméntée " si COP contient le code d'une primitive non implémentée dans l'interface appelé.

- " opération non définie " si COP ne contient pas le code d'une primitive.

2.2.3.1 Les primitives d'accès.

a. Accès à la base de données.

- Ouverture de la base de données :

<u>entrée :</u>

COP : 11
SSNAME
PSW
PROTECT
SREF

sortie :

SREF
RETCODE

fonction :

- initialiser les accès à une Base de Données (B.D.); cette primitive doit être exécutée avec succès, au moins une fois avant toute autre primitive.

- vérifier l'existence de la B.D. dont le nom est contenu dans SSNAME.

- vérifier l'exactitude du mot de passe contenu dans PSW.

- si la B.D. est fermée au moment de l'appel, la référence B.D. contenue dans SREF sera mémorisée et devra être contenue dans SREF lors de tous les appels ultérieurs, autres qu'une demande d'ouverture.

- si la B.D. est ouverte (ouvertures multiples cfr.2.2.1a), SREF sera garni par l'interface avec la référence fournie (dans SREF) lors de la première demande d'ouverture, quelle que soit la valeur d'entrée de SREF.

- mémoriser le mode d'accès aux articles des fichiers, contenu dans PROTECT
  PROTECT = 0 : mode normal; ouverture obligatoire des fichiers avant tout accès à l'un de leurs articles.
  PROTECT = 1 : mode automatique; ouverture non obligatoire des fichiers avant tout accès à l'un de leurs articles (la notion de fichier est ignorée).

- diagnostics d'erreurs : (transmis dans RETCODE)
  " B.D. inconnue " si SSNAME ne contient pas le nom de la B.D. gérée par l'interface.
  " mot de passe incorrect " si PSW ne contient pas le mot de passe de la B.D. gérée par l'interface.
  " PROTECT incorrect " si PROTECT est < 1 ou > 2.

- Fermeture de la base de données .

entrée :

COP : 12
SREF

sortie :

RETCODE

fonction :

- clôturer les accès à une base de données gérées par l'interface et dont la référence est dans SREF.

- rem : si plusieurs ouvertures de base de données ont été exécutées, celle-ci ne sera effectivement fermée que si le nombre de fermetures est égal au nombre d'ouvertures; de plus, au moment de la fermeture effective de la base de données, les fichiers encore ouverts seront fermés automatiquement par l'interface.

- diagnostics d'erreurs (transmis dans RETCODE) :
" B.D. non ouverte " si, au moment de l'appel, la B.D. n'est pas ouverte.

b. Accès aux fichiers.

- Ouverture des fichiers .

entrée :

COP : 21 / 22
SREF
FILNAME
PROTECT

sortie :

RETCODE
RFIL

fonction :

- vérifier l'existence et la disponibilité du fichier dont la référence est dans RFIL.

- initialiser l'accès à un (COP = 22) ou à tous (COP = 21) les fichiers de la base de données dont la référence se trouve dans SREF.

- La protection établie sur le fichier pendant le travail de l'utilisateur sera fonction de la valeur contenue dans PROTECT (cfr. la définition de PROTECT).
L'interface Cobol ne reconnaît que deux valeurs :
PROTECT = 2 : consultation sans protection
PROTECT = 3 : mise à jour sans protection
D'autres codes prendront en charge les différences existant dans les installations hôtes.

- si COP = 22 (ouverture d'un fichier), FILNAME doit contenir le nom du fichier que l'on veut ouvrir; après l'exécution de la primitive, RFIL contiendra la référence attribuée au fichier par l'interface (cfr.définition de la référence : 2.1.1).

- si plusieurs ouvertures sur le même fichier sont demandées, les droits restent ceux établis lors de la première ouverture.

- diagnostics d'erreurs (transmis dans RETCODE) :
  " FILNAME invalide " si FILNAME ne contient pas le nom d'un fichier de la B.D.
  " PROTECT incorrect " si PROTECT < 2 ou > 3
  " fichier non disponible " si le fichier, dont le nom est dans FILNAME, n'est pas accessible ou n'existe pas.
  " mode ouverture invalide " si, le fichier a été, lors de sa première ouverture, ouvert en consultation (en mise à jour), on cherche à l'ouvrir lors d'un appel ultérieur, en mise à jour (en consultation).

- Fermeture des fichiers.

  entrée :

  COP : 23 / 24
  SREF
  RFIL

  sortie :

  RETCODE

  fonction :

  - clôturer l'accès à un (COP = 24) ou à tous (COP = 23) les fichiers de la base de données dont la référence est dans SREF.

  - si on désire la fermeture d'un seul fichier (COP = 24), RFIL doit contenir la référence de ce fichier.

  - une fermeture doit toujours avoir été précédée par au moins une ouverture; cependant COP = 24 sera interprété comme étant une demande de fermeture de tous les fichiers ouverts.

  - la fermeture d'un fichier n'est effective que si le nombre de fermetures est égal au nombre d'ouvertures demandée pour ce fichier.

  - si la fermeture est effective, le(s) fichier(s) est(sont) devenu(s) inaccessible(s) pour l'utilisateur.

- Diagnostics d'erreurs (transmis dans RETCODE) :
  " RFIL invalide " si RFIL ne contient pas une référence attribuée à un fichier de la B.D.
  " fichier non ouvert " si RFIL ne contient pas une référence de fichier ouvert ou si aucun fichier n'était ouvert.

c. Accès aux articles.

Les primitives suivantes permettent d'accéder à des articles et à leurs valeurs d'items. On peut distinguer quatre types parmi ces primitives :

- accès aux articles d'un type,
- accès aux articles d'un fichier,
- accès aux articles cibles d'un chemin
- accès aux articles par leur référence.

Les trois premiers types sont l'objet d'une subdivision supplémentaire en accès séquentiel et accès par clé.
PREF doit contenir la référence du dernier article auquel on ait accédé. Dans le cas où cet article a été supprimé, on considère que sa référence lui survit jusqu'à l'accès suivant : PREF doit contenir la référence de l'article supprimé lors de l'accès à l'article suivant.

- Accès séquentiel aux articles d'un type donné.

entrée :

COP : 31
SREF
COREC
COGET
PREF
COKEY

sortie :

RETCODE
RREF
RFIL
RFIELD

fonction :

- fournir (dans RREF) la référence de l'article dont le type est donné dans COREC et qui suit celui dont la référence est contenue dans PREF.

- garnir RFIL de la référence du fichier qui contient les articles du type donné.

- si COGET = 1, les valeurs d'item de l'article seront placées dans RFIELD.

- si PREF = 0, l'accès concernera le premier article du type donné.

- Si aucune clé d'accès n'est définie sur le fichier, l'ordre de l'accès séquentiel aux articles d'un type sera celui de leur création : du premier au dernier.
  Si une clé d'accès a été définie sur le fichier, on accède aux articles dans l'ordre des valeurs croissantes de la clé.
  Si plusieurs clés sont définies, COKEY contiendra le code de la clé dont les valeurs croissantes détermineront l'ordre de l'accès séquentiel.

- Diagnostics d'erreurs (transmis dans RETCODE) :
  " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.
  " COKEY invalide " si COKEY ne contient pas le code d'une clé d'accès définie sur le fichier contenant le type d'article dont le code est dans COREC.
  " COGET invalide " si COGET > 1
  " fichier non ouvert " si, lors d'une tentative d'accès à un article d'un fichier fermé si l'ouverture de la base de données s'est faite en mode normal
  " pas d'article trouvé " si le dernier article auquel on a déjà accédé était le dernier du type considéré, dans le fichier.

- Accès par clé aux articles d'un type donné.

entrée :

COP : 32
SREF
COREC
COGET
PREF
COKEY
OPERAT
Z-CLE

sortie :

RETCODE
RREF
RFIL
RFIELD

fonction :

- fournir dans RREF la référence de l'article dont le type est donné dans COREC et qui vérifie la condition "COKEY OPERAT Z-CLE" .

- PREF doit être garni avec la référence du dernier article de ce type auquel on a accédé; si PREF = 0, on accède au premier, sinon, l'article auquel on accède sera le suivant de celui dont PREF contient la référence.

- fournir dans RFIL la référence du fichier qui contient cet article.

- si COGET = 1, les valeurs d'item de l'article seront retournées dans RFIELD

- Diagnostics d'erreurs (transmis dans RETCODE) :
  les mêmes que ceux rencontrés dans l'accès séquentiel avec en plus :
  " pas d'article trouvé " si aucun article dont la valeur de clé ne satisfait la condition donnée, n'a pu être trouvé.

- Accès séquentiel aux articles d'un fichier.

entrée

COP : 33
SREF
RFIL
COREC
PREF
COGET
COKEY

sortie :

RETCODE
RREF
RFIELD
COREC

fonction :

- fournir dans RREF la référence de l'article contenu dans le fichier dont la référence se trouve dans RFIL; si COREC est différent de 0, cfr.: accès séquentiel aux articles d'un type.

- la séquence dans laquelle on accède aux articles sera celle définie pour l'accès séquentiel aux articles d'un type.

- PREF doit contenir la référence du dernier article de ce fichier ou de ce type auquel on a accédé; si PREF = 0, on accèdera au premier.

- si COGET = 1, les valeurs d'item de l'article seront transférées dans RFIELD.

- si COREC = 0, si un seul type d'article est contenu dans le fichier ou si une condition d'identification de type d'article a été spécifiée, COREC contiendra après l'exécution de la primitive le code du type de l'article auquel on a accédé.

- Diagnostics d'erreurs (transmis dans RETCODE) :
  les mêmes que ceux définis pour l'accès séquentiel aux articles d'un type

- Accès par clé aux articles d'un fichier.

  entrée :

  COP : 34
  SREF
  RFIL
  COREC
  PREF
  COGET
  COKEY
  OPERAT
  Z-CLE

  sortie :

  RETCODE
  RREF
  RFIELD

  fonction :

  - fournir dans RREF la référence de l'article contenu dans le fichier dont la référence est dans RFIL et qui vérifie la condition "clé article OPERAT Z-CLE" sur la clé d'accès dont la référence est contenue dans COKEY; cet article sera du type dont la référence est contenue dans COREC, ou sera de type quelconque si COREC = 0.

  - PREF doit contenir la référence du dernier article du fichier auquel on a accédé; si PREF = 0, on accède au premier qui vérifie la condition; sinon l'article auquel on accède suivra celui dont la référence est dans PREF.

  - si COGET = 1, les valeurs d'item de l'article seront transférées dans RFIELD.

  - Diagnostics d'erreurs (transmis dans RETCODE) :
    les mêmes que ceux définis dans l'accès par clé aux articles d'un type.

- Accès séquentiel aux articles cibles d'un chemin.

  entrée :

  COP = 35/36
  SREF
  COSET
  COREC
  OREF
  COGET
  PREF

  sortie :

  RREF
  RFIL
  RFIELD
  RETCODE

  fonction :

  - non disponible dans les interfaces Cobol
  - fournir dans RREF la référence de l'article (type=COREC) cible d'un chemin dont la référence de l'article origine est donnée dans OREF et le code du type (de chemin) dans COSET; sachant que la référence de l'article précédent est donné dans PREF.
  - fournir la référence du fichier contenant cet article et éventuellement ses valeurs d'item (suivant COGET).
  - ordre : du premier au dernier si COP = 35
    : du dernier au premier si COP = 36.

- Accès par clé aux articles-cibles d'un chemin.

  entrée :

  COP=37
  SREF
  COSET
  COREC
  OREF
  COKEY
  OPERAT
  Z-CLE
  COGET
  PREF

  sortie :

  RREF
  RFIL
  RFIELD

fonction :

- non disponible dans les interfaces Cobol

- la même que pour l'accès séquentiel aux articles cibles d'un chemin, avec en plus, l'obligation pourla valeur de clé contenue dans l'article, de vérifier la condition :
"clé article OPERAT Z-CLE"

- Accès par référence à un article.

entrée :

COP : 38
SREF
COREC
RREF

sortie :

RETCODE
RFIL
RFIELD

fonction :

- si COGET = 1, fournir dans RFIELD les valeurs d'item de l'article dont la référence est contenue dans RREF, et dont le type est donné par COREC.

- garnir RFIL avec la référence du fichier qui contient cet article.

- Cette primitive ne fonctionne que si RREF contient la référence du dernier article de ce type auquel on a accédé. RREF ne peut contenir la référence d'un article supprimé (cfr. restrictions dans l'utilisation d'une référence cfr.2.2.1d).

- Diagnostics d'erreurs (transmis dans RETCODE) :
" COREC invalide " si COREC ne contient pas le code d'un type d'article de la base de données
" RREF invalide " si RREF ne contient pas la référence du dernier article dont le type est contenu dans COREC auquel on ait accédé, ou si cet article a été supprimé.

2.2.3.2 Les primitives de modification.

a. Primitives ayant pour cible un fichier.

- Réinitialisation des fichiers.

entrée :

COP : 25
SREF
FILNAME

sortie :

RFIL
RETCODE

fonction :

- supprimer tous les enregistrements contenus dans le fichier dont le nom est contenu dans FILNAME; cette primitive ne fonctionne que sur des fichiers pour lesquels n'existe aucune clé d'accès; RFIL contiendra la référence du fichier.

- Le fichier dont le nom est contenu dans FILNAME doit être fermé.

- Diagnostics d'erreurs (transmis dans RETCODE) :
" FILNAME invalide " si FILNAME ne contient pas le nom d'un fichier de la base de données
" fichier et fonction incompatibles " si FILNAME contient le nom d'un fichier sur lequel au moins une clé d'accès est définie.
" fichier en opération " si le fichier est déjà ouvert
" fichier non disponible " si FILNAME contient le nom d'un fichier non existant ou non accessible par le SGF.

b. Primitives ayant pour cible un article.

- Création d'un article.

entrée :

COP : 61
SREF
RFIL
COREC
Z-VALIT

sortie :

RETCODE
RFIELD

fonction :

- créer un article dans le fichier dont la référence est donnée dans RFIL; le type de cet article est contenu dans COREC et ses valeurs d'item dans Z-VALIT.

- cet article sera crée de facon à ce que les ordres d'accès définis sur le fichier soient respectés. Pour les fichiers sur lesquels sont définis des clés d'accès, l'insertion se fera en respectant la séquence des clés d'accès. Pour les autres fichiers, la création se fera en fin de fichier.

- Diagnostics d'erreurs (transmis dans RETCODE) :
  " RFIL invalide " si RFIL ne contient pas la référence d'un fichier de la B.D.
  " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.
  " mode-ouverture invalide " si le fichier dont la référence est contenue dans RFIL est ouvert en consultation.
  " clé double " si la valeur de la clé identifiante de l'article à créer est déjà identifiante d'un article du fichier.

- Suppression d'un article.

entrée :

COP : 62
SREF
COREC
RREF

sortie :

RETCODE

fonction :

- supprimer du fichier auquel il appartient l'article dont le type est donné dans COREC et dont la référence est contenue dans RREF.

- la référence contenue dans RREF doit être celle du dernier article auquel on a accédé dans le fichier.

- la référence de l'article supprimé peut encore être utilisée pour réaliser l'accès à l'article suivant, et pour cela uniquement.

- Diagnostics d'erreurs (transmis dans RETCODE) :
  " COREC invalide " si COREC ne contient pas le code d'un type d'article de la B.D.
  " RREF invalide " si RREF ne contient pas la référence

du dernier article du type donné auquel on a accédé.

- Modification des valeurs d'items d'un article.

  entrée :

  COP : 71
  SREF
  COREC
  RREF
  Z-VALIT

  sortie :

  RETCODE

  fonction :

  - remplacer les valeurs d'item de l'article dont la référence est dans RREF et le type dans COREC, par les valeurs d'item de l'article contenues dans Z-VALIT.

  - pour les fichiers sur lesquels est définie une clé d'accès identifiante, cette clé ne fait pas partie des items modifiables; il faut procéder à la suppression et à la recréation de cet article.

  - la référence contenue dans RREF doit être celle du dernier article auquel on a accédé dans le fichier.

  - Diagnostics d'erreurs (transmis dans RETCODE) :
    " COREC invalide " si COREC ne contient pas le code d'un type d'article de la base de données
    " RREF invalie " si RREF ne contient pas la référence du dernier article dont le type est contenu dans COREC auquel on ait accédé, ou si cet article a été supprimé.

- Insertion d'un article dans un chemin.

  entrée :

  COP = 81
  SREF
  COSET
  OREF
  RREF
  SETLST
  CURLST

  sortie :

  RETCODE

fonction :

- non disponible dans les interfaces Cobol

- insérer l'article dont la référence est dans RREF, dans le chemin ayant pour origine l'article dont la référence est dans OREF et pour type celui dont le code est dans RREF; cet insertion doit se faire en fonction de l'ordre défini pour ce chemin, et/ou de l'article membre auquel on a déjà accédé dans ce chemin (donné par SETLST et CURLST).

- Retrait d'un article d'un chemin.

entrée :

COP = 82
SREF
COSET
OREF
RREF

sortie :

RETCODE

fonction :

- non disponible dans les interfaces Cobol

- ôter d'un chemin (type = COSET , origine = OREF), l'article-cible dont la référence est contenue dans RREF.

2.2.3.3 Les primitives de contrôle.

Ces primitives ont pour objet de contrôler et commander les mécanismes chargés de la gestion de la concurrence et de la sécurité. Ces fonctions nécessitant une étude dans un contexte plus large que celui du modèle Cobol, ne seront pas implémentées; des codes-opérations leur ont cependant été attribués : 51 , 52 , 53 ..... . Toute tentative d'accès à ces primitives se soldera par une valeur de RETCODE signifiant leur non-implémentation.

Des exemples de ces fonctions de contrôle seraient :

- demande de contrôle sur un article fonction :
  - vérifier l'existence d'un article
  - fournir le type de l'article
  - bloquer un article
- demande de libération d'un article fonction :
  - abandonner le blocage d'un article

CHAPITRE 3

CHAPITRE 3 LE LANGAGE DE DESCRIPTION DE DONNEES

Après l'exposé des critères ayant présidé à son élaboration, nous proposerons un langage de description de données (D.D.L.) adapté au modèle de données Cobol.

3.1 Objectifs.

Ce D.D.L. vise à mettre à la disposition d'un administrateur B.D. un moyen de décrire une B.D. Cobol. Cette description, communiquée au système de génération, permettra à celui-ci de générer un interface chargé de réaliser les accès à cette B.D. Parce qu'il s'agit d'une B.D. Cobol gérée par un interface écrit en Cobol, l'utilisateur du D.D.L. devra être habitué aux clauses Cobol de description de fichiers.

3.2 Critères.

Trois facteurs ont été jugés déterminants dans la recherche d'un D.D.L. opérationnel :

- son adaptation au modèle de données Cobol;
  il va de soi que le D.D.L. doit permettre de décrire les éléments du modèle de données Cobol décrit au chapitre 1;

- son "apparence naturelle" pour un utilisateur de Cobol;
  tant sémantiquement que syntaxiquement, le D.D.L. doit être proche du Cobol et éviter dans la mesure du possible l'introduction de concepts étrangers au langage Cobol;

- sa souplesse d'adaptation;
  le lecteur familier du Cobol s'étonnera de ne pas retrouver dans le D.D.L. proposé, l'ensemble des clauses permises par Cobol pour la description des fichiers (ex.APPLY TECHNIQUE, RESERVE...); les raisons pour lesquelles elles n'ont pas été introduites sont :

  - soit ces clauses concernent des caractéristiques physiques du fichier qui sont de plus en plus prises en charge par un systèe central de gestion de fichiers, et donc traitées

comme des commentaires par les compilateurs

- soit elles font intervenir des techniques particulières laissées à la discrétion des constructeurs par la norme Cobol

Pour ces raisons, nous souhaitons un D.D.L. dans lequel les adaptations nécessitées par un compilateur particulier pourront être aisément introduites. Appartiennent à cette catégorie, dans le D.D.L. proposé, les clauses " records per block " et " assigned to "; clauses physiques qui peuvent être supprimées ou remplacées par d'autres, suivant les exigences du compilateur de l'installation hôte.

3.3 Proposition de D.D.L.

3.3.1 Conventions.

mot-minuscule : mot du langage (mo-clé,élément terminal)

MOT-MAJUSCULE : identification du type "mot majuscule" exemple : FILE-NAME pourrait prendre pour valeur F-CLIENT, F-COM-CLI,... qui sont des noms de fichier

< mot > : élément du langage à définir par une règle

::= : indique que l'élément à définir (à gauche) est défini par l'expression de droite. Le tout étant une règle.

[élément] $_{i}^{j}$: l'élément doit apparaître de i à j fois

[élément] : mis pour [élément] $_{0}^{1}$

* : mis pour "nombre infini"

$\left\{\begin{array}{l}\text{élément 1}\\ \quad "\\ \quad "\\ \quad "\\ \text{élément n}\end{array}\right\}$: l'un des éléments doit apparaître

"texte" : définition non formelle d'un élément

<Cobol-db-descr> : élément initial correspondant à la description d'une base de données

<integer> : représentation d'un entier positif

3.3.2 Le D.D.L.

```
<Cobol-db-descr>     ::= data-base description
                         name :  NOM-BD
                         interface-name : NOM-INTERFACE
                         password : MOT-DE-PASSE
                         [<file-descr>]n1

<file-descr>         ::= file description
                         name : NOM-FICHIER
                                          { seq          }
                                          { sequential   }
                         organisation     { sequential-indexed }
                                          { seq-ind      }
                                          { relative     }
                                          { rel          }
                         assigned to : UNITE-PERIPH
                         records per block : < integer >
                         [ primary key : NOM-CLE-PRIMAIRE                    ]
                         [ [internal code : < integer >]                     ]
                         [ [alternate key { dup   } : NOM-CLE-SECONDAIRE ]   ]
                         [ [              { nodup }                      ]   ]
                         [ [internal code : < integer >]                     ]
                         [<record-type-descr>]n1

<record-type-descr>  ::= record-type description
                         name : NOM-TYPE-ART
                         [internal code : < integer >]
                         [identification : CONDITION ]
                         [ 2 <item-descr> ]

<item-descr>         ::= NOM-ITEM [ occurs <integer> ]
                         [   <element-descr>           ]
                         [ [ <integer> <item-descr> ]  ]

<element-descr>      ::= pic "format logique Cobol"
                         [ usage "format physique Cobol" ]
                         [ { justified   right } ]
                         [ { just              } ]
                         [ synchronized ]
```

3.4 Sémantique du D.D.L. proposé.

data-base description

- doit-être la première clause de toute description de base de données (au moyen de ce D.D.L.).

- indique que toutes les clauses qui suivent concernent la base de données, jusqu'au début de la description d'un autre élément.

name : NOM-BD

- déclaration du nom de la B.D., ce nom l'identifie parmi les bases de données du système.

interface-name : NOM-INTERFACE

- déclaration du nom de l'interface qui gèrera la B.D.; ce nom l'identifiera parmi tous les interfaces chargés de la gestion d'une B.D.

password : MOT-DE-PASSE

- déclaration du mot de passe : MOT-DE-PASSE devra être fourni à l'interface avant que celui-ci n'autorise une action quelconque sur la B.D.

file-description :

- débute la description de tout fichier

- toutes les clauses qui suivent concernent un fichier, jusqu'au début de la description d'un autre élément.

name : NOM-FICHIER

- déclaration du nom identifiant le fichier parmi l'ensemble des fichiers de la B.D.

- NOM-FICHIER doit, de plus, être le nom qui identifie le fichier parmi l' ensemble (ou sous-ensemble) des fichiers du système.

organisation :

- déclaration du type d'organisation du fichier
  seq ou sequential : organisation séquentielle

seq-ind ou sequential-indexed: organisation séquentielle-indexée
rel ou relative : organisation relative

asigned to : UNITE-PERIPH

- le fichier dont on procède à la description sera assigné à une unité dont la dénomination (UNITE-PERIPH) sera conforme aux conventions du Cobol-hôte.

records per block : ‹integer›

- les blocs physiques du fichier contiennent ‹integer› articles

clés d'accès :
les quatres clauses suivantes ne peuvent être présentes que dans le cas où l'organisation du fichier décrit est séquentielle-indexée.

primary-key : NOM-CLE-PRIMAIRE

- déclaration de la clé primaire du fichier, dont la valeur doit obligatoirement être donnée pour création, suppression et modification.

- NOM-CLE-PRIMAIRE doit être le nom d'un item associé à un type d'article contenu dans le fichier; les items correspondant des autres types d'article seront présumés contenir aussi cette clé.

internal-code : ‹integer›

- déclaration d'une valeur de code qui identifiera la clé primaire parmi toutes les autres clés de la B.D.

- ‹integer› ne peut être une valeur attribuée à une autre clé

- si cette clause ne figure pas, le système se chargera de générer un code, mais de façon séquentielle; ce qui peut diminuer la résistance des programmes aux modifications de la B.D. On se réfèrera à 2.2.1 e pour de plus amples détails.

- cette déclaration et les remarques qui l'accompagnent sont valables aussi pour clés secondaires et types d'article.

alternate key { dup / nodup } : NOM-CLE-SECONDAIRE

- déclaration d'une clé secondaire du fichier, cette clé ne pourra être utilisée qu'en consultation

- NOM-CLE-SECONDAIRE doit être un nom d'item associé à un type d'article du fichier

- le nombre de clés secondaires que l'on peut déclarer dépend des spécifications du Cobol-hôte.

- dup signifie que la clé secondaire déclarée n'est pas identifiante, et, nodup le contraire.

record-type-description

- débute la description d'un type d'article

- toutes les clauses qui suivent, jusqu'au début d'une description d'item, concernent ce type d'article

name : NOM-TYPE-ART

- déclaration du nom du type d'article décrit; ce nom l'identifie parmi tous les types d'article pouvant être contenus dans un même fichier

identification : CONDITION

- cette clause permet à l'utilisateur d'indiquer au système le moyen de détecter l'appartenance d'un article au type décrit;

- la forme "CONDITION" est celle d'une proposition logique Cobol (cfr [15] (1) 4.1.3.2.3);avec ici, l'obligation d'avoir pour terme précedent l'opérateur un nom d'item de la B.D.

‹integer› NOM-ITEM

- déclaration d'un item dont le nom est nom d'item et le numéro de niveau dans l'ordre de la description dynastique est ‹integer›.

---

1. [15] A.CLARINVAL "COBOL"

occurs ‹integer›

- indique la répétitivité de l'item décrit; l'item sera présent ‹integer› fois

CHAPITRE 4

CHAPITRE 4 EXPRESSION COBOL DES PRIMITIVES

Après avoir énoncé les spécifications des primitives (chapitre 2), nous étudions maintenant le moyen de les réaliser à l'aide des ordres du langage Cobol. Nous commençons par définir une structure globale de l'interface constitué de l'ensemble des primitives. Nous précisons ensuite certaines options préalables qui ont guidé la réalisation de l'interface, avant de décrire données et traitements propres à chaque primitive.

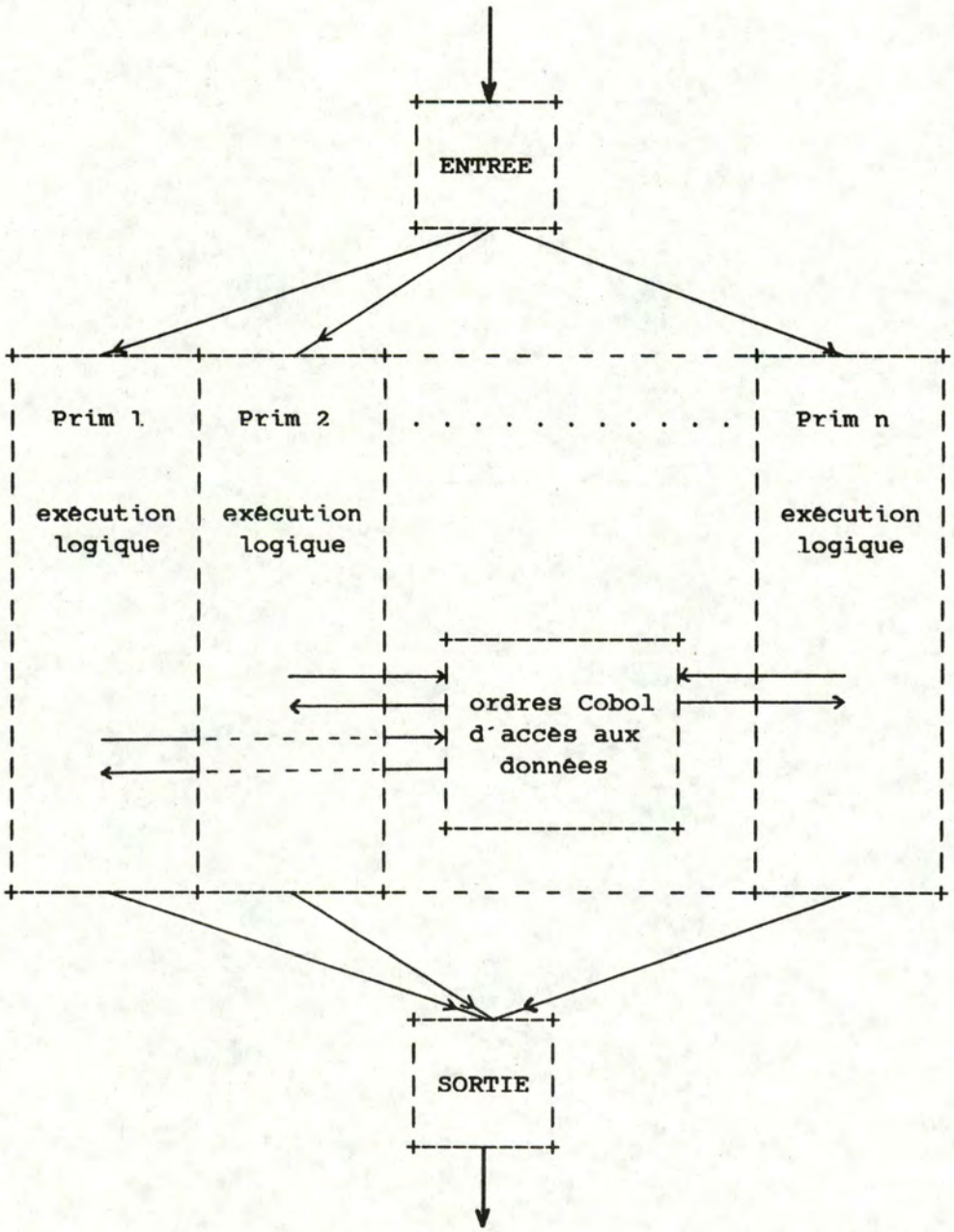
4.1 L'interface : structure et options.

4.1.1 Structure de l'interface.

L'objectif qui nous a guidé dans la recherche d'une structure globale de l'interface, a été d'isoler le plus possible les traitements spécifiques à chacune des primitives. Nous avons voulu ainsi permettre d'abord une mise au point aisée, primitive par primitive, et, ensuite, permettre une adaptation aisée du traitement d'une primitive aux particularités de l'installation, hôte du système de génération.

Le traitement d'une primitive est réalisé en deux niveaux: le premier, le niveau logique, est chargé de vérifier la cohérence de la requête, et de séquencer la suite d'ordres d'accès aux données Cobol requise par l'exécution de la primitive. Ces ordres d'accès Cobol constituent le second niveau du traitement d'une primitive.

Nous pouvons donner de la structure de l'interface, la vision schématique suivante :



4.1.2 Options fondamentales.

Nous tenons à préciser certaines options qui ont été prises afin d'assurer, aux utilisateurs des interfaces, un fonctionnement insensible aux particularités des systèmes-hôtes.

La norme Cobol laissant beaucoup (trop) de libertés aux concepteurs de systèmes Cobol, notamment en ce qui concerne le traitement des erreurs, et ces systèmes n'étant pas exempts d'erreurs, l'interface n'émettra que des requêtes cohérentes au système de gestion de fichiers Cobol. (ex: l'interface évitera d'écrire dans un fichier ouvert en input).

Afin d'isoler au maximum le système "utilisateur/interface" des erreurs de fonctionnement du SGF ou des "réactions inattendues" que celui-ci pourrait avoir, seules des erreurs

imprévisibles par l'interface (fin de fichier, fichiers inaccessibles..) seront à charge du SGF Cobol; les diagnostics de celui-ci seront transmis par l'interface à l'utilisateur.

4.2 Les données de l'interface.

Outre les paramètres d'appel (cfr chapitre 2 et annexe A), l'interface est appelé à manipuler des données qui lui sont propres et qui sont nécessaires à l'exécution des primitives. Nous distinguerons deux types de données : celles internes à l'interface, contenant des informations de service et celles contenant des objets de la B.D.

4.2.1 Les données internes.

Ces données servent, pour la plupart, à mémoriser des états ou des caractéristiques des objets de la B.D. C'est sur ces données que l'interface se base pour réaliser l'exécution logique des primitives.

a. Base de données.

REF-BD : mémorise la référence de la B.D., fournie par l'utilisateur à l'interface lors du premier appel.

NBRE-OUV-BD : indique, à tout moment le nombre d'ouvertures de la B.D. qui ont été demandées.

NBRE-FICHIERS : contient le nombre de fichiers constituant la B.D.

b. Fichiers.

Les données qui suivent concernent les fichiers; chacune d'entre elles est présente pour chacun des fichiers et doit donc être indicée par un identificateur de fichier (cfr 4.2.2).

NBRE-OUV-LOG : mémorise le nombre d'ouvertures demandées sur un fichier.

MODE-OUV-LOG : mémorise le mode de la première ouverture d'un fichier : consultation ou mise-à-jour.

MODE-OUV-PHYS : renseigne le mode d'ouverture physique du fichier : fermé, input, output, I-O, extend.

CURRENT : contient la dernière référence attribuée à un article du fichier.

TYPE-FICH : contient le code du type d'organisation du

fichier : séquentielle, séquentielle-indexée, relative.

NBRE-KEY : nombre de clés d'accès définies sur le fichier.

c. Types d'articles.

Chacune des données décrites ci-après doit être indicée par un identificateur de type d'article.

FICH-CODE-ART : contient un identificateur (interne à l'interface) du fichier qui contient les articles de ce type.

d. Clés d'accès.

Les données ci-après seront indicées par un identificateur (interne à l'interface) d'une clé d'accès.

FICH-CODE-KEY : contient un identificateur du fichier sur lequel est définie la clé d'accès.

4.2.2 Les objets de la B.D.

Nous décrivons ci-dessous les conventions prises dans l'interface pour la formation des noms des objets de la B.D.

- les fichiers : leur nom commence par F- et est suivi par un numéro qui, en séquence de 1 à n (n étant le nombre de fichiers de la B.D.) identifie chaque fichier.

- les types d'articles : leur nom commence par R- suivi d'un numéro (cfr fichiers). Ce numéro est indépendant du code qui a pu être fixé par l'utilisateur dans le D.D.L.

- les items : leur nom commence par I- suivi d'un numéro (cfr fichier).

- les clés d'accès : leur nom commence par K- suivi d'un numéro (cfr fichier), indépendant du code qui a été fixé par l'utilisateur dans le D.D.L.

4.3 Les fonctions.

Nous décrivons ci-dessous les fonctions qui constituent l'interface. Conformément à la structure définie en 4.1, nous distinguerons deux classes de fonctions : les fonctions logiques et les fonctions de manipulation physique des objets de la B.D.

Convention :

Dans la description qui va suivre, les noms de fonction seront en majuscules entre < et >.

Les conventions suivantes ont été adoptées pour l'écriture des tables de décision.

Les conditions figurent en début des tables et sont séparées par un trait de la seconde partie; celle-ci peut contenir deux types de lignes :

- ligne d'affectation dont le schéma est :

  nom variable <--- | val 1 | val 2 | . . . . .

  et se lit : " dans le cas représenté par la colonne dans laquelle se trouve val i, val i est affectée à la variable dont le nom précède <--- ".
  La signification des val i est donnée immédiatement après la table.

- ligne instruction dont le schéma est :

  instruction | n | n | . . . . .

  et se lit : " dans le cas représenté par la colonne dans laquelle se trouve n , exécuter "instruction" en enième rang dans la séquence de toutes les instructions exécutables dans ce cas ".
  Un "*", à la place de n, signifie que "instruction" peut être exécutée n'importe quand dans la séquence.

4.3.1 les fonctions logiques.

a. < DISCRIMINATION DES APPELS >

- entrée : COP , SREF
- sortie : RETCODE
- fonction :

  Cette fonction est réalisée lors de tout appel à l'interface; son but est d'aiguiller l'appel suivant le code opération (COP) vers les fonctions qui exécuteront la primitive demandée.

- réalisation :

```
si COP = 11 (ouverture de la B.D.)
     discriminer suivant COP
sinon
     si SREF not = REF-BD
          RETCODE <--- " SREF invalide "
     sinon
          si NBRE-OUV-BD > 0
               discriminer suivant COP
          sinon
               RETCODE <--- " B.D. non ouverte "
```

b. Les primitives.

- < OUVERTURE DE LA BASE DE DONNEES > (COP = 11)

  - entrée : SSNAME , PSW , PROTECT , SREF

  - sortie : RETCODE , SREF

  - réalisation :

```
< VERIFIER LES PARAMETRES >
si paramètres corrects
     si NBRE-OUV-BD = 0
          ajouter 1 à NBRE-OUV-BD
          < INIT INTERFACE >
          REF-BD <--- SREF
     sinon
          ajouter 1 à NBRE-OUV-BD
          SREF <-- REF-BD
sinon
     RETCODE <--- " nom-du-paramètre incorrect "
```

- < FERMETURE DE LA BASE DE DONNEES > (COP = 12)

  - entrée : SREF

  - sortie : RETCODE

  - réalisation :

```
< VERIFIER PARAMETRES >
si paramètres incorrects
     RETCODE <--- "nom-du-paramètre incorrect"
sinon
     si NBRE-OUV-BD not = 1
          soustraire 1 de NBRE-OUV-BD
     sinon
          soustraire 1 de NBRE-OUV-BD
          < FERMETURE DES FICHIERS (COP = 23) >
```

- ‹ OUVERTURE DES FICHIERS › (COP = 21/22)

  - entrée : FILNAME , PROTECT , COP

  - sortie : RETCODE , RFIL

  - réalisation :

    - COP = 21 : ouverture de tous les fichiers

```
‹ VERIFIER PROTECT ›
si PROTECT est correct
     pour chaque fichier
          ‹ OUV-LOG-ONE-FILE ›
sinon
     RETCODE ‹--- "PROTECT incorrect"
```

    - COP = 22 : ouverture d'un fichier

```
‹ VERIFIER LES PARAMETRES ›
si paramètres corrects
     ‹ GARNIR RFIL  ›
     ‹ OUV-LOG-ONE-FILE ›
sinon
     RETCODE ‹-- "nom-du-paramètre incorrect"
```

- ‹ FERMETURE DES FICHIERS › (COP = 23/24)

  - entrée : RFIL , COP

  - sortie : RETCODE

  - réalisation :

    - COP = 23 : fermer tous les fichiers ouverts

```
pour chaque fichier
     ‹ CLOSE-LOG-FICH ›
```

    - COP = 24 : fermer un fichier

```
‹ VERIFIER RFIL  ›
si RFIL  correct
     ‹ CLOSE-LOG-FICH ›
sinon
     RETCODE ‹--- "RFIL  incorrect"
```

- < ACCES SEQUENTIEL AUX ARTICLES D'UN TYPE > (COP = 31)

  - entrée : COREC , COGET , PREF , COKEY

  - sortie : RETCODE , RREF , RFIL , RFIELD

  - réalisation :

    < VERIFIER PARAMETRES >

    si paramètres corrects
        RFIL <--- ART-FICH (COREC)
        < ACCES SEQUENTIEL AUX ARTICLES D'UN FICHIER >

    sinon
        RETCODE <--- "nom-du-paramètre incorrect"

- < ACCES PAR CLE AUX ARTICLES D'UN TYPE > (COP = 32)

  - entrée : COREC , COGET , PREF , COKEY , OPERAT , Z-CLE

  - sortie : RETCODE , PREF , RFIL , RFIELD

  - réalisation :

    < VERIFIER PARAMETRES >

    si paramètres corrects
        RFIL <--- ART-FICH (COREC)
        < ACCES PAR CLE AUX ARTICLES D'UN FICHIER >

    sinon
        RETCODE <--- "nom-du-paramètre" invalide

- < ACCES SEQUENTIEL AUX ARTICLES D'UN FICHIER > (COP = 33)

  - entrée : RFIL , COREC , PREF , COGET , COKEY

  - sortie : RETCODE , PREF , RFIELD

  - réalisation :

a) si ORG (RFIL) = séquentielle

PREF = 0	0	0	0	0	0	0	0	N	N	N	N	N	N
NBRE-OUV-LOG (RFIL) = 0	=	=	>	>	>	>	>	=	>	>	>	>	>
MODE-OUV-PHYS (RFIL)	-	-	0	1	1	2	4	-	-	0	1	2	4
PREF = CURRENT (RFIL)	-	-	-	0	N	-	-	-	N	0	0	0	0
MODE-OPMA-BD	0	1	-	-	-	-	-	-	-	-	-	-	-
RETCODE	1					2		1	3	3		2	2
<CLOSE-PHYSIQUE>		5			1		1						
TYPE-OPEN-PHYS = INPUT		1	1		2		2						
<OPEN-PHYSIQUE>		2	2		3		3						
<LEC-SEQ-LOG>		3	3	1	4		4				1		
RREF <--- CURRENT (RFIL)		4	4	2	5		5				2		

RETCODE : 1 : "fichier non ouvert"
2 : "mode ouverture fichier invalide"
3 : "PREF invalide"

MODE-OUV-PHYS : 0 = CLOSED
1 = INPUT
2 = OUTPUT
3 = I / O
4 = EXTEND

MODE-OPMA-BD : 0 : MANUEL
1 : AUTOMATIQUE

b) si ORG (RFIL) non séquentielle

PREF = 0	O	O	O	O	N	N	N
NBRE-OUV-LOG = 0	=	=	>	>	=	>	>
PREF = CURRENT (RFIL)	-	-	O	N	-	N	O
MODE-OPMA-BD	1	2	-	-	-	-	-
RETCODE	1				2	3	
<CLOSE-PHYSIQUE>			1				
TYPE-OPEN-PHYS = I/O		1		2			
<OPEN-PHYSIQUE>		2		3			
Z-CLE <-- LOW-VALUE		3	1	4			
<RGT-KEY>		4	2	5			
<START-PHYS>		5	3	6			
<LEC-SEQ-LOG>		6	4	7			
RREF <--- CURRENT (RFIL)		7	5	8			2

RETCODE : 1 : "file-not-open"
2 : "séquence incorrecte"
3 : "RREF invalide"

- < ACCES PAR CLE AUX ARTICLES D'UN FICHIER > (COP = 34)

  - entrée : RFIL , COREC , PREF , COGET , COKEY , OPERAT , Z-CLE

  - sortie : RETCODE , RREF , RFIELD

  - réalisation :

< VERIFIER PARAMETRES >
si paramètres incorrects :
RETCODE <-- "nom de paramètre incorrect"
sinon :

ORG (RFIL) = séq-ind ou rel	O	O	O	O	O	N
NBRE-OUV-LOG = 0	=	=	=	>	>	
PREF = 0	O	O	N	N	N	
MODE-OPMA-BD	1	2	-	-	-	
RETCODE	1		2			3
TYPE-OPEN-PHYS <-- 3		1				
<OPEN-PHYSIQUE>		2				
<RAND-LEC-LOG>		3		1	1	
RREF <--CURRENT (RFIL)		4		2	2	
CLOSE		5				

RETCODE : 1 : "B.D. not open "
2 : "PREF incorrect"
3 : "fichier et fonction incompatibles"

- < ACCES PAR REFERENCE A UN ARTICLE > (COP = 38)

  - entrée : COREC , RREF , COGET

  - sortie : RETCODE , RFIL , RFIELD

  - réalisation :

```
< VERIFIER PARAMETRES >
si paramètres corrects
     si RREF = 0
          RETCODE <-- "RREF invalide"
     sinon
          RFIL <-- FICH-ART (COREC)
          si RREF = CURRENT (RFIL)
               < GET-TRSFRT >
          sinon
               RETCODE <--- "RREF invalide"
sinon
     RETCODE <--- " nom du paramètre invalide "
```

- < REINITIALISATION D'UN FICHIER > (COP = 25)

  - entrée : RFIL

  - sortie : RETCODE

  - réalisation :

```
vérifier RFIL
si RFIL correct
     si ORG (RFIL) est séquentielle
          si NBRE-OUV-LOG (RFIL) = 0
               TYPE-OPEN-PHYSIQUE <-- output
               < OPEN-PHYSIQUE >
               < CLOSE-PHYSIQUE >
          sinon
               RETCODE <-- "fichier en opération"
     sinon
          RETCODE <--- " fichier et fonction incompatibles "
sinon
     RETCODE <-- "RFIL invalide "
```

```
-  < CREATION D'UN ARTICLE > (COP = 61 )

   -  entrée : RFIL , COREC , Z-ART

   -  sortie : RETCODE , RFIELD

   -  réalisation :

      < VERIFIER PARAMETRES >

      si paramètres corrects

           si NBRE-OUV-LOG (RFIL) = 0
                RETCODE <-- "file not open"
           sinon

                si MODE-OUV-LOG (RFIL) = RETRIEVAL
                     RETCODE <-- "open mode invalide "
                sinon

                     si TYPE-FICH (RFIL) = séquentielle

                          si MODE-OUV-PHYS (RFIL) = closed
                               TYPE-OPEN-PHYS (RFIL) <-- OUT
                               < OPEN-PHYSIQUE >
                               < WRITE-LOG >
                          sinon

                               si MODE-OUV-PHYS (RFIL) = OUT ou  =
                               EXT
                                    < WRITE-LOG>
                               sinon
                                    RETCODE <--- " mode  ouverture
                                    fichier invalide "

                     sinon
                          < WRITE-LOG >

      sinon
           RETCODE <-- "nom paramètre invalide "
```

```
-  < SUPPRESSION D'UN ARTICLE > (COP = 62)

   -  entrée : COREC , RREF

   -  sortie : RETCODE

   -  réalisation :

      < VERIFIER PARAMETRES >
      si paramètres corrects
           RFIL <-- FICH-ART (COREC)

           si TYP-FICH (RFIL) = séquentiel
                RETCODE <--- "fichier et fonction incompatibles"
           sinon

                si NBRE-OUV-LOG (RFIL) = 0
                     RETCODE <-- "file not open"
                sinon

                     si MODE-OUV-LOG (RFIL) = RETRIEVAL
                          RETCODE <-- "mode open invalide"
                     sinon

                          si RREF NOT = CURRENT (RFIL)

                               si ORG (RFIL) = séq-ind
                                    RETCODE <-- "RREF invalide"
                               sinon
                                    REF-KEY <-- RREF
                                    < DEL-PHYS >

                          sinon
                               < DEL-PHYS >

      sinon
           RETCODE <-- "nom paramètre invalide "
```

- < MODIFICATION D'UN ARTICLE > (COP = 71)

  - entrée : COREC , RREF , Z-ART

  - sortie : RETCODE

  - réalisation :

```
< VERIFIER PARAMETRES >
si paramètres corrects
     RFIL <-- FICH-ART (COREC)

     si TYP-FICH (RFIL) = séq
          RETCODE <--- "fichier et fonction incompatibles"
     sinon

          si NBRE-OUV-LOG (RFIL) = 0
               RETCODE <-- "file not open"
          sinon
               si MODE-OUV-LOG (RFIL) = retrieval
                    RETCODE <-- "mode open invalide"
               sinon

                    si RREF not = CURRENT (RFIL)

                         si ORG (RFIL) = séq-ind
                              RETCODE <-- "RREF invalide"
                         sinon
                              REL-KEY <-- RREF
                              < REWR-PHYS >
                    sinon
                         < REWR-PHYS >
sinon
     RETCODE <-- "nom du paramètre invalide "
```

c. Fonctions logiques complémentaires.

- < INIT-INTERFACE >

Cette fonction consiste en l'initialisation des vecteurs d'états des fichiers.
Pour chaque fichier :

NBRE-OUV-LOG = 0
MODE-OUV-LOG = 0
MODE-OUV-PHYS = 0
CURRENT = 0

- < OPLOG-ONE-FILE >

- entrée : RFIL, PROTECT

- fonction :
ouvrir logiquement un fichier dont l'identificateur se trouve dans RFIL et le mode d'ouverture dans PROTECT

- réalisation :

TYPE-FICH = séq	O	O	O	O	O	O	N	N	N	N	N	N
PROTECT (4)	R	R	R	U	U	U	R	R	R	U	U	U
NBRE-OUV-LOG = 0	>	>	=	>	>	=	>	>	=	>	>	=
MODE-OUV-LOG = 0	O	N	-	N	O	-	O	N	-	N	O	-
NBRE-OUV-LOG + 1	*		*			*	*		*			*
MODE-OUV-LOG <-- PROTECT			*			*			*			*
CURRENT <-- 0			*			*			*			*
TYPE-OPEN-PHYSIQUE <-- (2)			I						A			A
< OPEN-PHYSIQUE >			*						*			*
RETCODE <--		1		1				1		1		
			(1)			(3)			(3)			

(1) la première demande d'accès sur ce fichier séquentiel permettra de déterminer le mode d'ouverture physique : Input, Output, I-O.

(2) I = INPUT, A = INPUT-OUTPUT

(3) pour les fichiers séquentiels-indexés et relatifs, le mode d'ouverture INPUT-OUTPUT, permettant toutes les opérations, sera seul utilisé par l'interface.

(4) R = RETRIEVAL , U = UPDATE

- < CLOSE-LOG-FICH >

  - entrée : RFIL

  - sortie : -

  - fonction :
    fermeture logique du fichier dont la référence numérique est dans RFIL.

  - réalisation :

```
si NBRE-OUV-LOG (RFIL) < 1
     RETCODE <-- "fichier non ouvert"
sinon

     si NBRE-OUV-LOG (RFIL) = 1

          si MODE-OUV-PHYS (RFIL) = closed
               soustraire 1 de NBRE-OUV-LOG (RFIL)
          sinon
               soustraire 1 ed NBRE-OUV-LOG (RFIL)
               MODE-OUV-LOG (RFIL) <-- closed
               < CLOSE-PHYSIQUE >

     sinon
          soustraire 1 de NBRE-OUV-LOG (RFIL)
```

- < LEC-SEQ-LOG >

  - entrée : INDIC-ARTICLE-TROUVE = 0 , RFIL , COREC , INDIC-ERREUR = 0

  - sortie : INDIC-ARTICLE-TROUVE , INDIC-ERREUR

  - fonction :
    réaliser la lecture séquentielle d'un article dont la référence du type est dans COREC et qui apartient au fichier dont la référence est dans RFIL; éventuellement assurer le transfert de l'article lu dans RFIELD (suivant COGET); vérifier le type de l'article lu.

  - réalisation :

```
< LEC-SEQ-PHYS >
si pas d'erreurs

    si COREC = 0
        INDIC-ARTICLE-TROUVE <-- 1
        < GET-TRSFRT >

        si ORG (RFIL) = séq ou séq-ind
            CURRENT (RFIL) + 1
        sinon
            CURRENT (RFIL) <-- REL-KEY

    sinon
        < RECH-TYPE-ART-LU >

        si COD-ART-LU = COREC
            INDIC-ARTICLE-TROUVE <-- 1
            < GET-TRSFRT >

            si ORG (RFIL) = séq ou séq-ind
                CURRENT (RFIL) + 1
            sinon
                CURRENT (RFIL) <-- REL-KEY

        sinon
            EXIT
```

- < RAND-LEC-LOG >

  - entrée : RFIL , COREC , COKEY , INDIC-ERREUR = 0 , Z-CLE , OPERAT

  - sortie : INDIC-ERREUR

  - fonction :
    effectuer la lecture aléatoire d'un article sur base d'une clé, dont le code est dans COKEY, la valeur dans Z-CLE et d'un opérateur (=,not >,>) dont le code est contenu dans OPERAT.

  - réalisation :

```
< RGT-KEY >
< START-PHYS >

si pas d'erreurs
     < LEC-SEQ-LOG >
     < VERIF-COND-KEY >
     CURRENT (RFIL) + 1
sinon
     EXIT
```

- < REF-READ-LOG >

  - entrée RREF , RFIL

  - sortie : INDIC-ERREUR

  - fonction :
    réaliser la lecture d'un article dans un fichier relatif (référence dans RFIL); le numéro de cet article est dans RREF.

  - réalisation :

```
REL-KEY <-- RREF
< REF-READ-PHYS >
CURRENT (RFIL) <-- REL-KEY
```

- < WRITE-LOG >

  - entrée : RFIL , Z-ART , OREC

  - sortie : INDIC-ERREUR

  - fonction :
    assure l'écriture d'un article dont le type est contenu dans COREC et dont les valeurs d'items sont contenues dans Z-ART; cet article appartient au fichier dont la référence est dans RFIL.

  - réalisation :

```
si ORG (RFIL) = relatif
     REL-KEY <-- RREF
sinon

     < PUT-TRSFTR >
     < WRITE-PHYS >
      COGET <-- 1
     < GET-TRSFRT >
```

4.3.2 Les fonctions physiques.

Ces fonctions réalisent les manipulations physiques des objets de la B.D. Une fonction est capable d'effectuer la même manipulation sur tous les objets d'un même type; la fonction "lire un article" reçoit le code du fichier et lit ensuite un article de ce fichier.

Ces fonctions physiques sont :

- < OPEN-PHYSIQUE >
  exécute l'ouverture d'un fichier dont la référence est dans RFIL; le type d'ouverture (INPUT, OUTPUT, I-O, EXTEND) est donné dans TYPE-OPEN-PHYSIQUE.

- < CLOSE-PHYSIQUE >
  exécute la fermeture d'un fichier donné dans RFIL.

- < RGT-KEY >
  effectue le transfert de la clé transmise dans Z-CLE dans la zone devant contenir la clé du fichier. COKEY contient le code de la clé.

- < PUT-TRSFRT >
  transfère un article dont le code du type est donné dans COREC de Z-ART (paramètre d'appel) dans la zone tampon du fichier.

- < GET-TRSFRT >
  effectue le transfert d'un article lu, de la zone tampon du fichier vers Z-RESP.

- < START-PHYS >
  effectue un START sur le fichier dont la référence est dans RFIL; la condition sur la clé est donnée par OPERAT et Z-CLE.

- < LEC-SEQ-PHYS >
  réalise la lecture séquentielle d'un article d'un fichier dont la référence est dans RFIL.

- < DEL-PHYS >
  suppression d'un article du fichier dont la référence est dans RFIL.

- < REWR-PHYS >
  réécriture de l'article dont le code du type est dans COREC.

- < WRITE-PHYS >
  écriture d'un article dont le code du type est dans COREC.

- < VERIF-COND-KEY >
  vérifie si une condition (donnée par OPERAT et Z-CLE) est satisfaite par la clé de l'article lu; COKEY contient le code de la clé.

CHAPITRE 5

CHAPITRE 5 CONTENU ET FORMAT DE LA B.D. DES SCHEMAS.

Nous décrivons, dans ce chapitre, la forme sous laquelle les descriptions des schèmas de B.D. doivent être mises à la disposition du système I.D.M.L. C'est parce que I.D.M.L. est un utilisateur potentiel des interfaces, et que l'exécution de certains de ses processus nécessite des accès aux descriptions des B.D. sur lesquelles le système peut travailler, que la description de ces schèmas doit lui être communiquée par le système de génération (cfr. Chap. 6).

Après la présentation de leurs raisons d'être dans le contexte I.D.M.L., nous décrivons les tables qui contiendront les descriptions de schèmas et la forme sous laquelle I.D.M.L. s'attend à ce qu'elles lui soient communiquées.

5.1 Raisons d'être.

I.D.M.L. ayant pour modèle de description de données, le modèle d'accès, les descriptions des schèmas de B.D. qu'il manipule sont réalisées en termes du modèle d'accès (quel que soit le SGBD/SGF manipulant ces tables).

L'intérêt de ces descriptions, au niveau d'I.D.M.L., est multiple. Elles seront utilisées pour la compilation des programmes I.D.M.L. et pour l'établissement de rapports destinés aux programmeurs et qui leur présenteront, en termes de modèle d'accès, les données qu'ils peuvent manipuler. Une autre utilisation envisageable de ces descriptions pourrait être la génération automatique de programmes de chargement, de consultation ou de mise à jour de B.D. Le lecteur se reportera utilement à [3] et à [8] (1) pour de plus amples détails.

---

1. [3] J.L. HAINAUT "IDML : an interface for efficient use of Codasyl data bases".
   [8] J.L. HAINAUT "IDML : système d'interaction avec des banques de données Codasyl".

5.2 Description des schèmas.

Le schèma conceptuel de ces descriptions se trouve dans [18] (1).

5.2.1 Conventions.

La description que nous donnons est sous forme de tables que nous représenterons de la façon suivante :

nom-table (nom 1,nom 2,... nom n).

nom 1,nom 2,nom n étant les noms des ensembles de valeurs (domaines) parmi lesquelles l'élément 1, 2, n d'une ligne peut prendre ses valeurs.

Chaque ligne d'une table est identifiée par le rang qu'elle occupe dans la table. Les éléments identifiants seront soulignés. Un "*" signifie qu'un élément a une répétitivité égale au nombre qui suit cet "*".

5.2.2 Définitions des domaines.

NOM : identificateur d'un élément de la B.D. : B.D., fichier, type d'article, item ou type de chemin. La table dans la ligne de laquelle il figure permettra de déterminer s'il s'agit d'un identificateur de B.D., de fichier ou de ...

MOT-PASSE : mot de passe de la B.D.

NOM-INTERFACE : identificateur de l'interface chargé de la gestion de la B.D.

OPER : code identifiant une opération possible sur un élément de la B.D.

CODE : code identifiant un élément de la B.D. (cfr 2.2.1 e).

PT-nom-table : pointeur vers une ligne de la table dont le nom est nom-table. Il s'agit en fait du numéro d'une ligne de cette table.

NBRE-CLE : définit le nombre de clés d'accès définies pour un type d'article.

CONN-O : connectivité Origine dans un type de chemin.

CONN-C : connectivité Cible dans un type de chemin.

---

1. [18] J.L. HAINAUT " Base de données des schèmas "

PT-INVERSE : PT-TCHEMIN si TCHEMIN est le type de chemin inverse du type de chemin décrit.

NO-NIVEAU : numéro de niveau d'un item.

STR-INTERNE : décrit le type d'un item : imprimable, alphabétique, numérique...

UNITE : décrit le code interne utilisé - 4 bits, 6 bits, 7 bits, 8 bits ...

LONG : longueur d'un item en nombre d'unités du code interne utilisé.

LONG-DECIM : le nombre de digits à prendre en compte s'il s'agit d'un item numérique.

FAC/OBL : code indiquant si un item est facultatif ou obligatoire.

REPET : répétitivité d'un item.

PT-COMPTEUR : PT-ITEM si ITEM est le compteur de répétitivité de l'item décrit.

TYPE-O/C : indique si un type d'article est origine ou cible d'un type de chemin.

MODE-INS : mode d'insertion dans un type de chemin.

MODE-RETR : mode de retrait dans un type de chemin.

CLASSE : peut-être identifiant, clé d'accès, ordre.

TYPE-REFER : type de référentiel. ex.: fichier, B.D., type article.

ORDRE : décrit l'ordre d'éléments de la B.D.;
les possibilités suivantes sont prévues : néant, first, last, prior, next, sorted.

DOUBLE : détermine le traitement des doubles. Peut-être sans objet (si CLASSE = identifiant), first ou last.

SENS : peut-être sans objet, croissant ou décroissant.

TYPE-I/TC : permet d'indiquer s'il s'agit d'un item ou d'un type de chemin.

PT-I/TC : pointeur vers une ligne de la table d'items (I) ou de la table des types de chemin (TC).

5.2.3 Les tables.

Les tables permettant de décrire le schéma sont au nombre de 10, nous les présentons ci-dessous en ne détaillant que celles nécessaires à la description d'une B.D. Cobol.

1. B.D. (NOM, MOT-PASSE, NOM-INTERFACE, OPER*4)
   Cette table ne contient qu'une ligne par description de schéma. Elle associe à une B.D. de nom NOM, un mot de passe, un nom d'interface et les opérations qu'on peut effectuer.

2. FICHIER (NOM, OPER*8)
   Description du fichier de nom NOM; huit opérations possibles peuvent être définies sur ce fichier.

3. TARTICLE (NOM, CODE, OPER*5, PT-CONTIENT, PT-ITEM, PT-ORIG-CIBLE, PT-SIMPLE, NBRE-CLE)
   Description d'un type d'article de nom NOM, de code CODE. Les opérations qu'il est possible de définir sont au nombre de 5. Les pointeurs et NBRE-CLE n'ont été introduits que dans un but d'accélération des traitements.

   PT-CONTIENT pointe vers la ligne de la table CONTIENT qui associe le type d'article décrit à un fichier.

   PT-ITEM pointe vers la ligne qui, dans la table ITEM contient la description du premier item associé à ce type d'article.

   PT-ORIG-CIBLE n'est pas utilisé en Cobol (pas de chemin inter-articles).

   PT-SIMPLE pointe vers la ligne qui contient la description de premier simple défini pour le type d'article.

   NBRE-CLE contient le nombre de clés définies pour ce type d'article.

4. TCHEMIN (NOM, CODE, CONN-O, CONN-C, PT-INVERSE)
   Table sans objet en Cobol. Elle contient les descriptions des types de chemin.

5. ITEM (NOM, CODE, NO-NIVEAU, (STR-INTERNE, UNITE, LONG, LONG-DECIM), FAC/OBL, REPET, PT-COMPTEUR, OPER*2)
   Cette table contient les descriptions des items. L'ordre des descriptions est celui de la décomposition dynastique. Un item de numéro de niveau = 1 est décrit pour chaque type d'article.

6. CONTIENT (PT-TARTICLE, PT-FICHIER)
   Cette table associe un type d'article au(x) fichier(s) qui contiendra(ont) les articles de ce type.

7. ORIG-CIBLE (PT-TARTICLE, TYPE-O/C, PT-TCHEMIN, MODE-INS, MODE-RETR, OPER*4)

Cette table associe un type d'article à un type de chemin, elle n'est pas utilisée en Cobol.

8. GLOBAL (CLASSE, TYPE-REFER, PT-REFER, ORDRE, CODE)

Une ligne de cette table décrit un mécanisme global dont la classe est soit identifiant, soit clé d'accès, soit ordre pour des éléments de la B.D. dans un référentiel qui peut être la B.D., un fichier ou un chemin d'un type.

9. SIMPLE (PT-TARTICLE, CLASSE, TYPE-REFER, ORDRE, CODE, DOUBLE, PT-COMPOSANT)

Même signification que GLOBAL mais restreint à un type d'article.

10. COMPOSANT (PT-SIMPLE, TYPE-I/TC, PT-I/TC, SENS)

Réalise la liaison entre un mécanisme (simple) et l'élément de la B.D. (item ou type de chemin),grâce auquel est réalisé ce mécanisme.

5.3 Le fichier des tables.

Après avoir défini les tables de description de schémas, nous allons présenter la forme sous laquelle l'ensemble de ces tables doit être communiqué au système I.D.M.L.

Ces tables seront rangées dans un fichier qui ne contiendra qu'une seule description de B.D.; l'ordre dans lequel les tables sont présentées dans ce fichier est quelconque; cependant, l'ordre dans lequel figureront les lignes d'une table est défini ci-dessous :

1. B.D. : une seule ligne par schéma : pas de tri.
2. FICHIER : triée sur nom.
3. TARTICLE : triée sur nom.
4. TCHEMIN : vide dans le cas d'une B.D. Cobol.
5. ITEM : l'ordre dynastique de décomposition doit être respecté, donc pas de tri.
6. CONTIENT : triée sur (PT-TARTICLE, PT-FICHIER).
7. ORIG-CIBLE : vide dans le cas d'une B.D. Cobol.
8. GLOBAL : triée sur (CLASSE, TYPE-REFER, PT-REFER, CODE).
9. SIMPLE : triée sur (PT-TARTICLE, CLASSE, TYPE-REFER, PT-REFER, CODE).
10. COMPOSANT : triée sur (PT-SIMPLE).

La génération de ce fichier peut être envisagée de la façon suivante: deux fichiers seront créés; l'un ne contenant que la table ITEM et l'autre contenant toutes les autres tables. Ce second fichier sera trié sur (identification de table, clé), avec clé contenant au moins, pour chaque ligne, les éléments sur lesquels elle doit être triée. Une fusion ultérieure des deux fichiers permettra de constituer le fichier demandé.

CHAPITRE 6

CHAPITRE 6 LE SYSTEME DE GENERATION.

Ce chapitre décrit le système de génération tel qu'il a été implémenté à l'institut d'Informatique. Nous présentons, tout d'abord, l'architecture globale des fonctions dont il est constitué; ensuite nous décrivons chacune de ces fonctions en précisant son objectif et les principes ayant présidé à sa spécification.

6.1 Architecture du système.

6.1.1 Identification des fonctions.

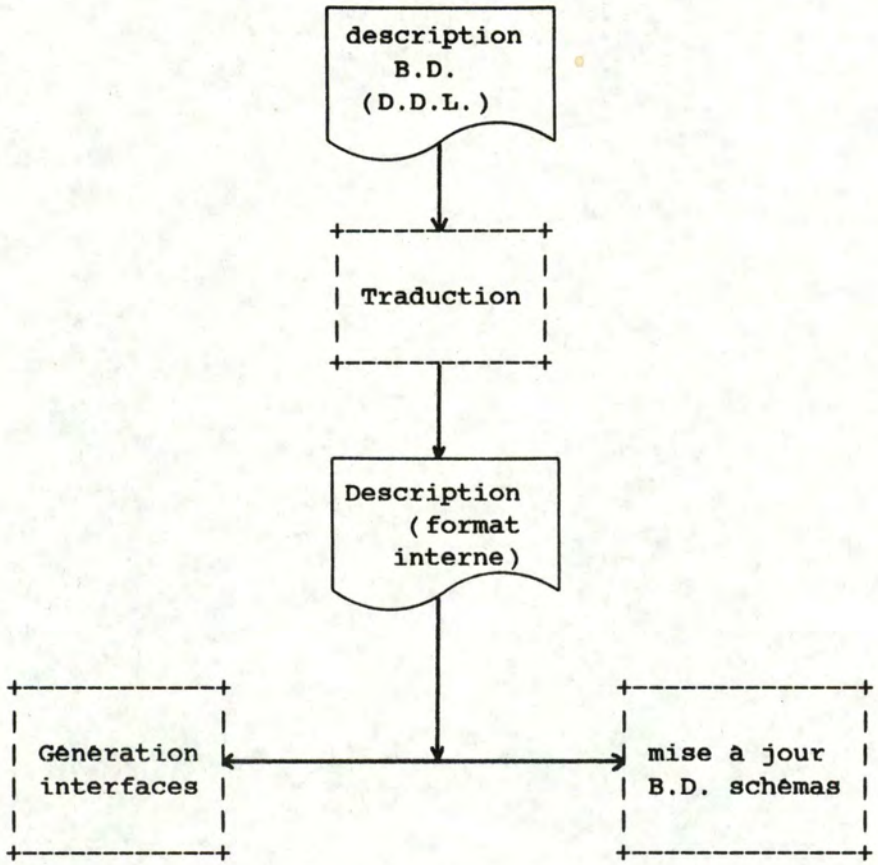
Les objectifs du système, génération d'interfaces et mise à jour de la B.D. des schémas d'I.D.M.L., nous permettent de distinguer immédiatement deux fonctions.

Il nous a paru utile de leur en adjoindre une troisième: en effet, la description initiale de la B.D. est écrite avec le D.D.L. défini au chapitre 3.

Cette forme n'étant pas de manipulation aisée par des programmes, une troisième fonction a été prévue dans le système: la traduction du texte D.D.L. sous forme de tables.

De plus, la vérification du texte D.D.L. n'étant pas nécessaire à chaque exécution d'une des deux fonctions définies ci - dessus, cette troisième fonction assurera cette vérification.

Le flux du système correspondra donc au schéma suivant :

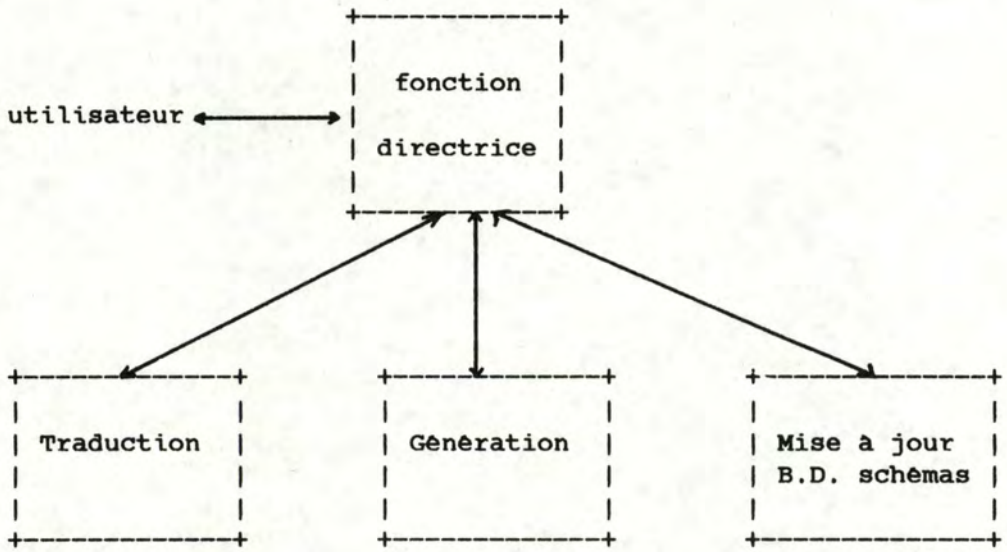


6.1.2 Architecture.

Afin d'offrir plus de souplesse au système, nous introduisons une quatrième fonction dans le système. Cette fonction (fonction directrice) assurera le séquencement des autres fonctions du système suivant les désirs de l'utilisateur.

En effet, un utilisateur peut désirer générer un interface sans mettre à jour la B.D. des schémas; on peut aussi imaginer que cette fonction permette le stockage des descriptions des schémas sous forme interne, l'utilisateur n'ayant qu'à en donner le nom à cette fonction directrice pour y accéder.

L'architecture adoptée est donc :



6.2 Description des fonctions.

Nous allons, ci-dessous, décrire les fonctions, leurs objectifs et les mécanismes mis en oeuvre pour atteindre ceux-ci. Nous précisons qu'il ne s'agit pas d'une description complète des traitements de chaque fonction, mais d'une présentation des principes qui nous ont guidé dans leur réalisation.

6.2.1 Le traducteur de D.D.L.

Ce traducteur a, comme objectifs, la vérification du texte D.D.L. introduit sous la forme d'un fichier séquentiel, et sa traduction en une forme interne exploitable par les autres fonctions.

6.2.1.1 La description du D.D.L.

Un des critères d'élaboration du D.D.L. (cfr.chap.3) était de permettre, grâce à une description souple du texte D.D.L., des modifications aisées de sa définition. Nous présentons ci - dessous la forme donnée à cette description.

Déterminons, tout d'abord, les éléments à décrire:

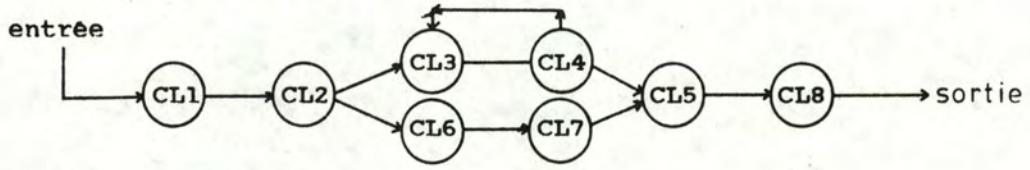
le premier de ceux-ci est le texte D.D.L.; il est composé de clauses du langage D.D.L. qui se suivent dans un certain ordre.

Le second élément à décrire est constitué par les clauses du langage: un ou, en général, plusieurs mots les composent; ceux - ci feront l'objet d'un troisième niveau de description.

a. Description du texte.

La description du texte qui a été adoptée trouve son origine dans la constatation suivante: le nombre de clauses différentes qui peuvent suivre une clause déterminée, est fini. Cela nous permet de modéliser le texte sous forme d'un graphe orienté dont les arcs indiquent les successions possibles des clauses représentées par les sommets.

Exemple de graphe.



On dira : l'ensemble des suivants de CL1 = { CL2 }
" " " " CL2 = { CL3 , CL6 }
" " " " CL4 = { CL5 , CL3 }

Un texte D.D.L. sera donc représenté par un chemin particulier parcouru dans ce graphe, dont l'origine et la cible sont fixées (par convention : clause initiale et clause terminale).
Dans le cas du D.D.L. proposé, la clause initiale sera "data-base description" et la clause terminale : "‹element-descr›".

La forme sous laquelle ce graphe a été représenté dans le traducteur, est celle d'une matrice de succession. Chaque clause a reçu un numéro arbitraire qui donne le numéro de la ligne qui contient celui des clauses suivantes.

On interprètera donc :

ligne i : s1, s2, s3, s4, s5,...

La clause i (dont le numéro identifiant est i), peut avoir pour successeur l'une des clauses s1, s2, ... (dont le numéro identifiant est s1, s2, ...).

Nous pouvons présenter une ébauche de l'algorithme de vérification du texte :

```
si la permière clause du texte est la clause initiale

     i <-- numéro de la clause initiale
     pour chaque clause

     si clause appartient à {suivantes de la clause i}
          traitement clause
     sinon
          erreur
sinon
     erreur
```

b. Description des clauses

Chaque clause étant constituée d'une suite de mots, il nous suffit pour décrire une clause, de lui associer les identifiants des mots qui la composent.

Ici aussi, une table nous permettra de réaliser cette description:

```
si (i) (m1, m2, m3, ...)
```

représente la ième ligne de cette table, on lira : la clause dont le numéro est i (cfr. Description du texte), est composée des mots dont les identifiants sont m1, m2, m3, ...

c. Description des mots

Les mots composant les clauses du texte D.D.L. sont de quatre types :

1) les éléments terminaux propres au langage D.D.L.
2) les terminaux facultatifs :
   éléments terminaux non obligatoires
3) les non-terminaux : on n'en connaît que le format
4) les non-terminaux facultatifs : idem

Chaque élément terminal, facultatif ou non, constitue une ligne d'une table, ligne dont la rang sert d'identifiant au mot. Les identifiants les plus petits sont ceux des mots non facultatifs. Les non-terminaux ont reçu une identification (supérieure au nombre maximum d'éléments terminaux), qui spécifie leur format. Les identifiants les plus petits sont ceux des non facultatifs.

exemple :

terminaux : de 1 à 40 (si 40 éléments terminaux existent)

non-terminaux : de 50 à ...

50 identifiant par exemple un format alphabétique de longueur = 1.

En résumé, la description du D.D.L. est constituée de 3 tables :

1. une table de successeurs décrivant le graphe de succession
2. une table de description des clauses; les éléments d'une ligne étant identificateurs des mots de la ligne
3. une table de mots contenant les éléments terminaux et une codification des formats

L'objectif de cette description étant de permettre des modifications aisées du D.D.L.; considérons quelques exemples :

a. modification d'une clause :

remplacer un mot par un autre consiste à remplacer dans la description de la clause un identifiant de mot par un autre; si le mot (format) remplaçant n'est pas codifié dans la table des mots, l'y ajouter (lui créer un code).

b. suppression d'une clause :

adapter le graphe de séquence en fonction de cette suppression; la clause supprimée ne peut plus appartenir à aucun ensemble de suivants et, son ensemble de suivants est à joindre à celui des clauses qu'elle suivait.

c. ajoût d'une clause :

ajouter la description de cette clause; éventuellement ajouter ses mots (formats) dans la table (codification) des mots (formats);
créer une entrée supplémentaire correspondant à la nouvelle clause, dans la table de succession;
y mettre ses suivantes;
modifier les suivants de la (des) clause(s) qui la précède(nt);

6.2.1.2 La forme interne du schéma

La forme interne du D.D.L est celle sous laquelle la description de la B.D. est accessible aux autres fonctions du système. Nous en donnons ci-dessous la description sous formes de tables; Les conventions adoptées pour la description de ces tables sont celles définies au chapitre 5 (5.2.1).

Définition des domaines :

BDIBD : identificateur de la B.D.

BDIIN : identificateur de l'interface gérant la B.D.

BDPSW : mot de passe de la B.D.

FIDEN : identificateur d'un fichier

FASGN : unité d'assignation d'un fichier

FORGN : organisation d'un fichier

FNORB : nombre d'articles par bloc d'un fichier

RTIDE : identificateur de type d'article

RTCON : condition d'appartenance d'un article à un type d'article

RTINC : code interne d'un type d'article

IIDEN : identificateur d'un item

IIDPE : identificateur de l'item père de l'item considéré

ILENU : numéro de niveau d'un item

IPICT : format logique d'un item

IUSAG : code physique d'un item

IOCCU : répétitivité d'un item

IJUST : justification à droite d'un item

ISYNC : synchronisation d'un item sur frontière de mot

KIDEN : identificateur d'une clé

KINCO : code interne d'une cl'.

Les tables gènèrèes sont :

- BD (BDIBD, BDIIN, BDPSW)
  La B.D. dont le nom est BDIBD a pour mot de passe BDPSW et le nom de l'interface chargè de sa gestion est BDIIN.

- FICHIER (FIDEN, FASGN, FORGN, FNORB)
  Table des descriptions des fichiers.

- FICH-TYP-ART (FIDEN, RTIDE)
  Cette table associe un type d'article à un fichier.

- TYP-ART (RTIDE, RTCON, RTINC)
  Table des descriptions des types d'articles.

- TYP-ART-ITEM (RTIDE,IIDEN)
  Cette table associe un item à un type d'article.

- ITEM (IIDEN, IIDPE)
  Cette table contient des descriptions d'item dans l'ordre dynastique de la description de la structure.

- ELEM-ITEM (IIDEN, ILENU, IPICT, IUSAG, IOCCU, IJUST, ISYNC)
  Contient les descriptions des items èlèmentaires.

- FICH-CLE (FIDEN, KIDEN)
  Associe une clè à un fichier.

- CLE-ITEM (KIDEN, IIDEN)
  Associe une clè d'accès à un fichier et un item.

- CLE (KIDEN, KINCO)
  Table des descriptions des clès.

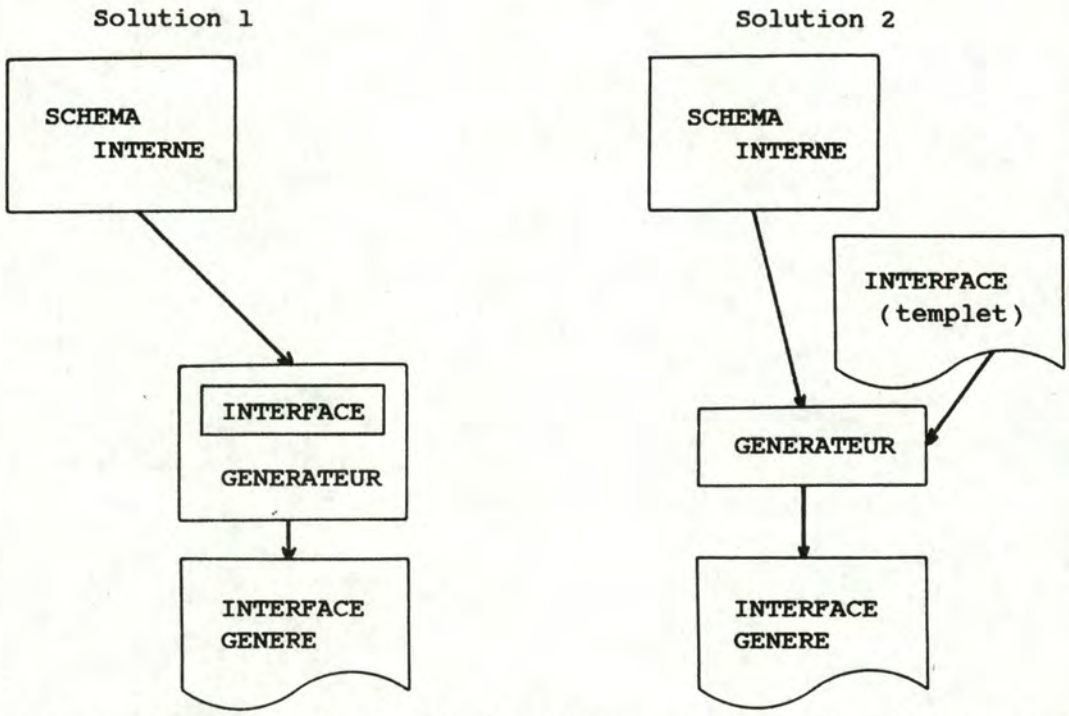
6.2.2 Le gènèrateur.

Nous prèsentons ci-dessous le gènèrateur rèalisè. Après l'exposè des principes retenus, nous dècrirons les moyens mis à la disposition de l'utilisateur pour effectuer une gènèration.

6.2.2.1 Principes de gènèration.

Deux voies ont ètè envisagèes pour la gènèration des interfaces :

- dans la première, le texte du programme gènèrateur contient entièrement celui de l'interface; le programme gènèrateur est figè; il procède sèquentiellement à la gènèration.
- dans la seconde, l'interface est externe au gènèrateur; le texte de l'interface contient des directives de gènèration qui sont interprètèes par le gènèrateur.



Une modification de l'interface entraîne, dans le premier cas, celle du générateur et sa recompilation, et, dans le second cas, celle de l'interface, sans intervention sur le générateur.

De plus, la première solution nous amène à écrire un programme spécifique à la génération d'interfaces Cobol; la génération d'interfaces pour d'autres SGBD/SGF faisant l'objet de travaux ultérieurs, il nous a paru interessant de proposer ici, une voie susceptible d'extensions. La seconde solution a donc été retenue.

Le principe appliqué dans cette solution s'apparente en fait, à celui retenu pour les macros-générateurs : un texte est traité par un programme qui, en fonction des directives contenues dans ce texte, génère un texte définitif. Le texte initial se verra appelé dans la suite de l'exposé "templet".

Dans notre cas, le templet sera, par exemple, l'interface écrit en Cobol et contenant des directives de génération, tandis que le texte généré sera l'interface, prêt à être compilé.

Si le principe est celui des macro-générateurs, les directives de génération ne représentent qu'un sous-ensemble très réduit de celles des macro-générateurs. En effet, le templet (l'interface), peut être décomposé en sous-séquences, - plus ou moins importantes -, spécifiques aux fichiers, aux articles ou aux clés. Ces sous-séquences réalisent chacune une action particulière sur les objets de la B.D. Cobol. Considérant cela, nous avons pu déterminer les possibilités du générateur : il doit pouvoir recopier du texte; en effet, certaines parties de l'interface sont invariantes quel que soit le schéma de la base de données. Pendant cette recopie, il peut avoir a insérer, éventuellement, une valeur (ex: nom de fichier); Cette possibilité sera donc prévue. Vu la découpe du templet en sous-séquences générales (cfr. supra), l'itération sera le mécanisme qui permettra, sur la base de la description de la B.D., la génération de

ces sous-séquences pour tous les éléments du type qu'elle manipule.

Outre cette restriction quant aux possibilités du langage, une autre différence par rapport aux macro-générateurs, est que le texte source ne contient pas toutes les informations nécessaires à la génération : une part importante de ces informations est constituée par la forme interne de la descripton du schéma (cfr 6.2.1.2).

Appliquons ce principe à un exemple :

- soit générer un paragraphe de nom déterminé contenant un ordre de lecture; cela, pour tous les fichiers de la B.D.;
  le texte source serait :

```
début itération :
     générer pour chaque fichier
texte :
     NOM-PARAGRAPHE-"identificateur-fichier"
          READ "identificateur-fichier"
fin itération :
     fin génération.
```

Les deux mécanismes de génération sont illustrés :
l'itération : pour chaque fichier
l'insertion : identificateur-fichier sera remplacé par sa valeur (pour le fichier concerné par l'itération).

Nous allons maintenant détailler ces deux mécanismes et leurs possibilités.

6.2.2.2 Les directives de génération.

a. Les itérations.

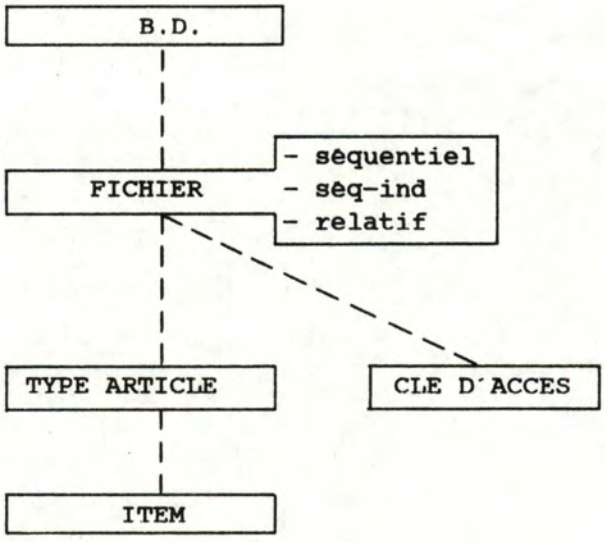
- Les objets de la B.D. sur lesquels ls itérations sont possibles, sont les suivants : fichiers, types d'articles, clés d'accès et items. En plus, un mécanisme élémentaire de sélection a été introduit qui permet les itérations sur les fichiers séquentiels, séquentiels-indexés et relatifs.

- Une itération se marque de la facon suivante :

```
*/GENERATE FOR EACH "objet"
     texte à générer
*/END
```

- */ : la raison de la présence de ces caractères est expliquée en 6.2.2.2 c.

- Imbricaton des itérations.
  Une imbrication n'est possible que si l'objet de la boucle intérieure est descendant direct de l'objet de la boucle extérieure, sur l'arbre suivant :



- L'objet de la boucle intérieure voit son nom qualifié par celui de l'objet de la boucle extérieure; le système considère que le texte source commence par GENERATE FOR B.D. et se termine par END

  exemple :

  les ordres suivants :

```
*/GENERATE FOR EACH FILE
     texte
*/GENERATE FOR EACH RECORD-TYPE
     texte
*/GENERATE FOR EACH ITEM
     texte
*/END
*/END
*/END
```

  sont équivalents à :

```
*/GENERATE FOR EACH FILE OF B.D.
     texte
*/GENERATE FOR EACH RECORD-TYPE OF FILE
     texte
*/GENERATE FOR EACH ITEM OF RECORD-TYPE
     texte
*/END
*/END
*/END
```

- L'objet de la boucle extérieure peut être quelconque.

- Dans un but de facilité, les trois ordres END peuvent être remplacés par END*3 qui signifie fin des trois boucles, ou encore par END* qui signifie fin de toute boucle.

b. Insertion.

Ce mécanisme permet d'insérer dans le texte une information contenue dans la forme interne de la description de la B.D. (cfr 6.2.1.2) Les paramètres sont ceux définis en 6.2.1.2. Dès que le générateur rencontre une des abréviations définies, il la remplace par la valeur correspondante actuelle de la forme interne du schéma. Cela signifie que les valeurs concernant la B.D. (les abréviations commencent par BD) peuvent être générées n'importe où dans le texte; celles concernant le fichier, uniquement à l'intérieur d'une boucle

*/GENERATE FOR EACH FILE

et ainsi de suite.

Exemple :

Soit générer les clauses F.D. décrivant les articles du fichier accessibles par un programme Cobol :

```
*/GENERATE FOR EACH FILE
     FD F-#FIDEN
*/GENERATE FOR EACH RECORD-TYPE
     01 R-#RTIDE
*/GENERATE FOR EACH ITEM
     #ILENU I-#IIDEN #IPICT #IUSAG ...
*/END * 3
```

c. Directives techniques.

A ces mécanismes s'ajoutent des ordres destinés à améliorer les performances du générateur; ce sont SCAN et NOSCAN. SCAN déclenche la recherche dans chaque ligne , d'abréviations à remplacer. NOSCAN arrête cette recherche.

Les lignes sur lesquelles figurent des ordres GENERATE, END, SCAN ou NOSCAN, auront pour premiers caractères "*/"; tandis que les paramètres commenceront par "#". Ces conventions n'ont d'autre but que l'amélioration des performances du générateur.

6.2.2.3 Evaluations.

Les possibilités du générateur apparaissent sans doute, trop étendues, pour la seule génération d'interfaces Cobol. Il permet en effet la génération de tout autre programme ou même d'un rapport de documentation sur la B.D. (cfr annexes E et F). Mais ces possibilités offrent aussi des voies d'extension :
tel quel (ou avec des modifications très légères) on pourrait très bien générer des interfaces écrits en des langages dont le modèle de données est proche de celui du Cobol (PL1 par ex.); avec des extensions, soit des directives de génération (en permettant des sélections moins primaires), soit des informations contenues dans le schéma interne (en incluant la notion de chemin inter-articles), il sera sans doute possible de générer des interfaces pour des modèles de données plus évoluées; ceci fera l'objet d'un travail ultérieur.

6.2.3 Production du schèma destinè à I.D.M.L.

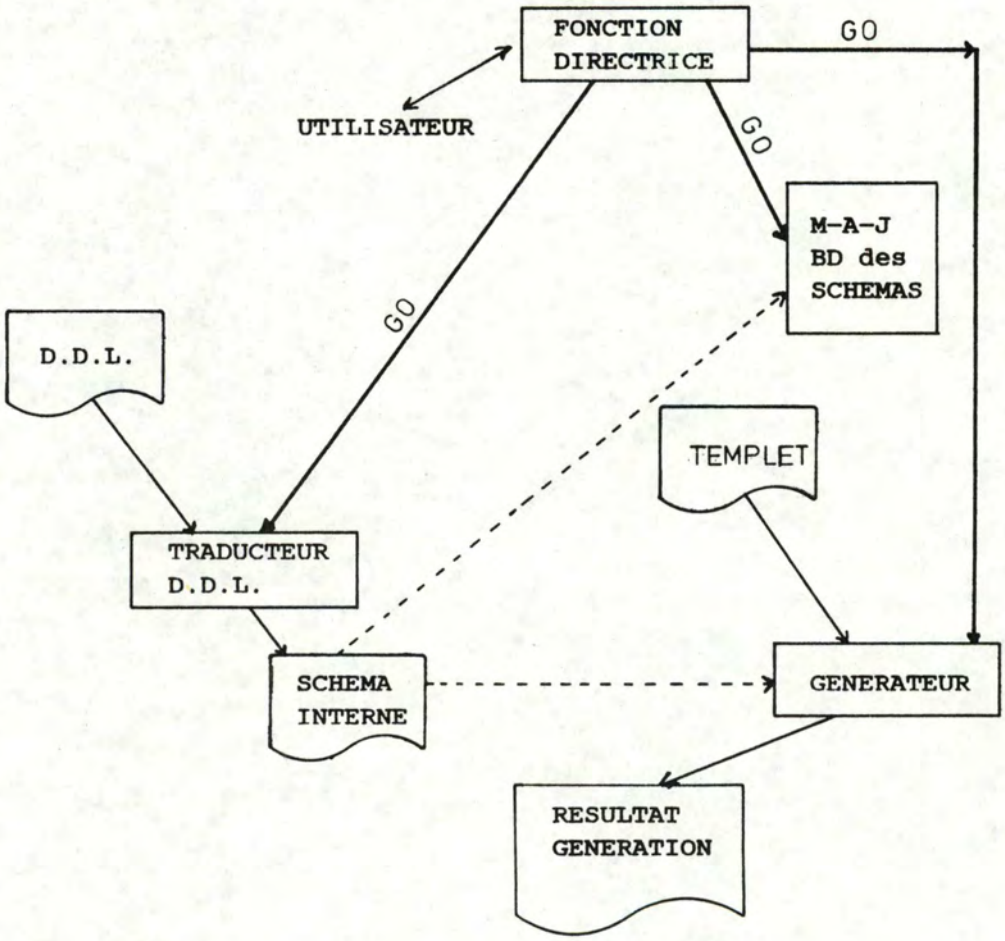
Cette fonction, essentiellement une conversion effectuée entre les tables du système de gènèration et celles dèfinies dans I.D.M.L. (cfr chapitre 5), ne prèsentant au niveau de sa rèalisation, que peu d'intèrêt, ne sera pas dècrite.

6.2.4 La fonction directrice.

Cette fonction, outre la synchronisation de toutes les fonctions du système, permet en outre, des extensions fonctionnelles au choix de l'implèmenteur (par exemple : sauvetage ou restauration des tables internes).

6.3 Schèma gènèral de fonctionnement du système.

Le schèma gènèral de fonctionnement du système de gènèration est le suivant :



La description du schèma (D.D.L. dans un fichier sèquentiel) est soumise au traducteur de D.D.L., celui-ci produit un schèma interne (tables) qui sera transmis à la mise à jour de la B.D. des schèmas et au gènèrateur. Celui-ci, sur base du templet (fichier sèquentiel), effectuera la gènèration dont le rèsultat sera rangè dans un fichier sèquentiel. La fonction directrice gèrera l'ensemble du système en fonction des ordres de l'utilisateur.

CONCLUSION

CONCLUSION.

Nous pouvons, au terme de ce mémoire, tenter d'évaluer la démarche suivie et son adaptation aux objectifs poursuivis.

Nous distinguons deux phases dans notre démarche. La première qui consiste en la définition des interfaces est constituée de trois parties : restriction du modèle d'accès aux possibilités du SGBD/SGF cible, adaptation des primitives au modèle d'accès restreint et spécification détaillée des interfaces; la seconde phase est constituée par la recherche et la description des moyens nécessaires à la génération des interfaces définis au terme de la première phase.

Une remarque que l'on peut faire immédiatement sur ce schéma, est qu'aucune de ses étapes n'est spécifique à un SGBD/SGF (1) particulier - Cobol ou autre -. L'application de cette démarche pourrait donc être envisagée quel que soit le SGBD/SGF cible.

Les résultats auxquels nous avons abouti au terme de cette démarche nous fournissent un second élément d'appréciation de celle-ci. Les interfaces générés, correspondant à l'objectif défini en début de ce travail manifestent l'adaptation des moyens aux objectifs.

Si, dans la phase de définition des interfaces, l'adaptation de la méthode aux objectifs paraît claire, la seconde phase semble, au contraire, nous avoir amené à mettre en oeuvre des moyens disproportionnés par rapport à son objectif : la génération automatique d'interfaces; en effet, le système proposé fournit le moyen de générer autre chose que des interfaces Cobol.

Cette interprétation, pour justifiée qu'elle est, nous paraît devoir être corrigée en replaçant l'étude qui a été réalisée dans un cadre plus général que celui du Cobol.

Le Cobol ne constituant que le premier SGBD/SGF pour lequel des interfaces seront générés de façon automatique, il nous est apparu non dénué d'intérêt de proposer des moyens susceptibles de développement. Dès lors, si, pour d'autres SGBD/SGF, le système de génération - ou une adaptation de celui-ci - se révèle capable d'assurer la génération des interfaces, l'importance des moyens proposées dans ce mémoire trouvera sa justification.

---

1. SGBD : Système de Gestion de Bases de Données.
   SGF : Système de Gestion de Fichiers.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[1] C.J. DATE
"An introduction to database systems" - Second edition
Addison-Wesley - 1977

[2] JAMES MARTIN
"Computer data-base organisation" - Second edition
Prentice Hall - 1977

[3] J-L. HAINAUT
"Interactive data manipulation language
- An interface for efficient use of Codasyl data bases."
Institut d'Informatique Namur - Mars 1979

[4] HARTMUT WEDEKIND
"System independent program design with data bases"
Technical University Darmstadt - février 2978

[5] J-L. HAINAUT
"Un modèle de description de fichiers : le modèle d'accès"
Institut d'Informatique Namur - 1980

[6] J-L. HAINAUT
"Caractérisation des accès proposés par le DBTG Codasyl (1971)"
Institut d'Informatique Namur -

[7] B. LE CHARLIER et J-L. HAINAUT
"Modèles,langages et systèmes pour la conception et l'exploitation de bases de données"
Institut d'Informatique Namur -

[8] J-L. HAINAUT
"I.D.M.L. : Système d'interaction avec des banques de données Codasyl"
8eme Ecole d'été de l'AFCET Namur - Juillet 1978

[9] Y. DELVAUX
"Résumé des propositions Codasyl" (Projet I.D.M.L.)
Institut d'Informatique Namur - CODASY.DOC/1 - Janvier 1979

[10] J-L. HAINAUT
"Machine virtuelle Codasyl"
Institut d'Informatique Namur - CVM1 DOC/1 - Mars 1979

[11] J-L. HAINAUT
"Implémentation des programmes I.D.M.L."
Institut d'Informatique Namur - IMP1.DOC/1 - Juin 1979

[12] Y. DELVAUX J-L. HAINAUT
"Système portable de manipulation de bases de données hétérogènes"
Institut d'Informatique Namur - Mars 1981

[13] J-L. HAINAUT
"Theorical and practical tools for data base design"
Institut d'Informatique Namur -

[14] A. CLARINVAL
"COBOL" (norme 1974)
Institut d'Informatique - Juin 1979

[15] A. van LAMSWEERDE
"Programming systems for non-programmers"
Some wishes and some attemps
MBLE - Bruxelles - Janvier 1979

[16] DIGITAL EQUIPMENT CORPORATION
"Cobol user's manual"

[17] TEXAS INSTRUMENTS
"Cobol programmer's guide"
Juin 1979

[18] J-L. HAINAUT
"Base de données des schémas"
Institut d'informatique - Namur(1981)

ANNEXES

ANNEXE A TYPE ET FORMAT DES PARAMETRES D'APPEL DE L'INTERFACE

```
01 Z-CODES.
     02 COP              PIC XX.
     02 SREF             PIC X.
     02 COREC            PIC X.
     02 RETCODE          PIC 9999.
     02 PROTECT          PIC 9.
     02 COGET            PIC 9.
     02 CONTRL           PIC 9.
     02 RFIL             PIC X.
     02 RREF             PIC S9(10).
     02 PREF             PIC S9(10).
     02 COKEY            PIC X.
     02 OPERAT           PIC 9.
     02 COMOD            PIC 9.
     02 COSET            PIC X.
     02 OREF             PIC S9(10).
     02 TYP              PIC X.

01 Z-IDENT.
     02 SSNAME           PIC X(30).
     02 PSW              PIC X(30).
     02 FILLER           PIC X(196).
01 Z-FIL REDEFINES Z-IDENT.
     02 FILNAME          PIC X(30).
     02 FILLER           PIC X(226).
01 Z-VALIT REDEFINES Z-IDENT.
     02 FILLER           PIC X(256).
01 Z-CLE REDEFINES Z-IDENT.
     02 FILLER           PIC X(256).

01 Z-ITEM.
     02 ITEMLST.
          03 ITEM             PIC X OCCURS 32.

01 Z-RESP.
     02 RFIELD           PIC X(256).

01 Z-SETS.
     02 STKREF           PIC S9(10).
     02 SETLST.
          03 SETL             PIC X OCCURS 32.
     02 CURLST.
          03 CURNT            PIC S9(10) OCCURS 32.
```

ANNEXE B Codes d'erreurs

Nous énumérons ci-dessous les valeurs de RETCODE qui seront transmises à l'utilisateur lors de la détection d'erreurs par l'interface.

97 : SREF incorrect
99 : COP incorrect
93 : PSW incorrect
98 : BDNAME incorrect
92 : PROTECT incorrect
94 : FILNAME incorrect
91 : RFIL incorrect
88 : COKEY incorrect
72 : COGET incorrect
78 : RREF incorrect
75 : OPERAT incorrect
78 : PREF incorrect
96 : COREC incorrect
95 : B.D. non ouverte
89 : fonction non disponible
30 : fichier non disponible
77 : mode ouverture fichier incompatible
80 : fichier non ouvert
76 : fichier en opération
76 : fichier et fonction incompatible
22 : clé double
26 : pas d'article trouvé

ANNEXE C Codes opérations.

Nous rappelons ci-dessous les codes opérations attribués à chacune des fonctions susceptibles d'être réalisées par les interfaces.

Un "*" à la suite du code opération signalera celles qui ne sont pas disponibles dans les interfaces Cobol.

11 : ouverture B.D.
12 : fermeture B.D.
21 : ouverture de tous les fichiers.
22 : ouverture d'un fichier.
23 : fermeture de tous les fichiers.
24 : fermeture d'un fichier.
25 : réinitialisation d'un fichier
31 : accès séquentiel aux articles d'un type
32 : accès par clé aux articles d'un type
33 : accès séquentiel aux articles d'un fichier
34 : accès par clé aux articles d'un fichier
35* : accès séquentiel aux articles cibles d'un chemin du premier au dernier
36* : accès séquentiel aux articles cibles d'un chemin du dernier au premier
37* : accès par clé aux articles cibles d'un chemin
38 : accès par référence à un article
51* : contrôle B.D.
52* : contrôle fichier
53* : contrôle article
61 : création d'un article
62 : suppression d'un article
71 : modificaton des valeurs d'item d'un article
81* : insertion d'un article dans un chemin
82* : retrait d'un article d'un chemin