MASTER EN SCIENCES MATHÉMATIQUES

Implémentation du modèle d'accès par lui-même

Laloux, J. P.

Award date:
1976

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Année académique 1975-1976

Implémentation du modèle d'accès par lui-même

JEAN-PIERRE LALOUX

Mémoire présenté en vue de l'obtention
du grade de
Licencié et Maître en Informatique

REMERCIEMENTS

Je tiens à exprimer ma profonde gratitude à Baudouin Le Charlier pour l'intérêt porté à ce mémoire et l'aide constante qu'il m'a prodiguée lors de la rédaction. Ses nombreux conseils et suggestions ont permis de clarifier les difficultés théoriques et de réaliser une implémentation complète de cette étude.

Qu'il me soit également permis de remercier l'équipe "Grands Fichiers" pour sa sympathique et efficace collaboration durant mon travail.

TABLE DES MATIÈRES

DEUXIEME PARTIE : REPRESENTATION DES TYPES DE DONNEES
DU MODELE D'ACCES DANS UN SYSTEME
CIBLE : SESAM

TROISIEME PARTIE : LE LANGAGE DE MANIPULATION DE DONNEES ET SON COMPILATEUR

QAUTRIEME PARTIE : LE LANGAGE DE DESCRIPTION DE DONNEES
ET SON COMPILATEUR

INTRODUCTION

Le modèle d'accès est un modèle de bases de données qui a été défini par les chercheurs de l'équipe Grands Fichiers, de l'Institut d'Informatique. Ceux-ci ont également défini un langage de manipulation de données (DML) associé à ce modèle et terminent son implémentation.

Nous décrivons, dans ce mémoire, un langage de définition de données (DDL), complémentaire du précédent, et nous en étudions l'implémentation. Celle-ci a été réalisée, au moyen du DML, dans le but principal de confronter le modèle d'accès et son DML à un problème réel. Cette utilisation du modèle d'accès est assez particulière et relativement éloignée des applications pour lesquelles il a été conçu au départ. Elle offre certains avantages du point de vue de la portabilité, mais nous ne nous sommes pas attachés à développer ce point.

En réalité, l'implémentation du DDL ne peut se concevoir que dans le cadre global de l'implémentation du modèle d'accès. C'est la raison pour laquelle ce travail comporte quatre parties toutes nécessaires à la compréhension de l'ensemble.

Dans la première partie, nous donnons une définition du modèle d'accès, nous précisons la signification du titre "implémentation du modèle d'accès par lui-même" et nous introduisons la notion de métaschéma sémantique.

La deuxième partie décrit une représentation des types de données du modèle d'accès dans un système cible : SESAM.

La réalisation du DDL grâce au modèle d'accès implique d'utiliser également ce dernier pour réaliser le DML. C'est pourquoi, dans la troisième partie, nous exprimons, en termes du modèle d'accès, les informations nécessaires au compilateur du DML. Celles-ci constituent le métaschéma complet.

Enfin, la quatrième partie contient la définition du DDL et un exposé détaillé de son implémentation.

PREMIÈRE PARTIE : LE MODÈLE D'ACCÈS

INTRODUCTION

Dans un langage de programmation, on distingue généralement, d'une part, les types de données et les actions primitives agissant sur ces types de données et, d'autre part, les mécanismes permettant de combiner les primitives pour obtenir des algorithmes complexes (affectation, instructions conditionnelles, etc.).

La plupart du temps, lorsque l'on parle de langage de base de données, on ne s'intéresse qu'à la première partie, la seule qui soit propre aux bases de données c'est-à-dire les types de données et les actions primitives. Pour combiner les primitives, on utilisera les mécanismes d'un langage hôte, par exemple, le COBOL (codasyl).

Un langage de base de données de ce type se décompose en deux parties :
- le langage de définition de données (DDL) permet de définir les types de données, l'ensemble de ces types constituant le schéma de la base de données;
- le langage de manipulation de données (DML) est formé d'un ensemble de primitives agissant sur les types définis par le DDL.

Alors que les types de données sont définis en fonction du problème, les primitives de manipulation sont définies une fois pour toutes. En fait, il existe une classification des types de données que l'on peut définir et les primitives sont définies par rapport à ces classes.

Le modèle d'accès, tel qu'il a été défini par l'équipe Grands Fichiers des F.N.D.P., n'est pas à proprement parler un langage comme ceux dont nous venons de parler. Il définit un certain nombre de concepts sans préciser un formalisme (langage) nécessaire à leur implémentation sur machine.

Toutefois, le fait qu'il existe bien des concepts ayant exactement les propriétés que nous allons énumérer est un postulat que nous validerons, "dans les faits", en réalisant une implémentation du modèle ce qui nécessitera la définition d'un DDL et d'un DML.

Dans ce chapitre, nous définirons donc les éléments du modèle d'accès et, dans les chapitres ultérieurs, nous présenterons un DDL et un DML dont la sémantique sera fixée par une correspondance entre les phrases du langage et les concepts du modèle.

Pour une présentation plus intuitive du modèle d'accès, faisant davantage référence aux situations réelles qu'il est sensé représenter, on se reportera aux différents documents de l'équipe Grands Fichiers. Notre but est de donner ici une définition rigoureuse et concise du modèle de façon à faciliter la définition de la sémantique des langages associés et l'étude de leur implémentation.

CHAPITRE I : LES CONCEPTS DU MODELE D'ACCES

1. Les types de données

Les types de données du modèle d'accès se répartissent en deux classes principales : celle des objets et celle des relations. Le schéma d'une base de données est l'ensemble des types de données qu'elle contient. Il comprend un objet racine et un nombre fini d'objets complexes, d'objets élémentaires et de relations. La première action relative à une base de données est la définition de son schéma.

1.1. Les objets

La classe des objets se décompose en trois sous-classes : l'objet racine, les objets complexes et les objets élémentaires (*).

1.1.1. Racine (OR)

La racine (ou objet racine) est un type de données très particulier qui possède une seule réalisation existant à priori (c'est-à-dire dès la définition du schéma).

1.1.2. Objets complexes (OC)

Un objet complexe est un type de données n'ayant aucune réalisation à priori. Ces réalisations seront créées puis pourront être supprimées grâce aux primitives que nous définirons en (3.2.); elles sont discernables.

1.1.3. Objets élémentaires (OE)

Un objet élémentaire est un type de données dont les réalisations sont des valeurs, c'est-à-dire des chaînes finies de caractères. L'ensemble de valeurs d'un objet élémentaire est défini à priori et une fois pour toutes. Cet ensemble peut être défini de deux façons principales qui déterminent deux catégories d'objets élémentaires.

(*) Toutes les règles et définitions données dans la suite de ce chapitre, sont relatives à des types de données d'un même schéma, sauf mention explicite du contraire.

Le domaine de valeurs d'un *objet élémentaire simple* est défini par une règle qui détermine un ensemble de chaînes finies de caractères sans aucune référence à un autre type de données du modèle d'accès (la façon exacte dont cette règle pourra être formulée devra être précisée lors de la définition du DDL).

Le domaine de valeurs d'un *objet élémentaire composé* est défini par un ensemble ordonné d'objets élémentaires distincts du premier, nantis chacun d'un nombre entier positif.
Si $\{OE_1 < OE_2 < \ldots < OE_n\}$ est cet ensemble ordonné d'objets élémentaires, et $(i_1 , i_2 , \ldots i_n)$, la liste des nombres associés,
une valeur de l'objet élémentaire composé est formée de la concaténation de i_1 valeurs de OE_1, i_2 valeurs de OE_2,..., i_n valeurs de OE_n, non nécessairement distinctes, prises dans cet ordre.
Les OE_i sont appelés *objets attributs* de l'objet élémentaire composé.

Appelons, de plus, *descendant d'un objet élémentaire,* soit un attribut de cet objet, soit un descendant d'un tel attribut. La règle suivante assure que toute valeur d'objet élémentaire sera une chaîne finie de caractères : un objet élémentaire ne peut pas être descendant de lui-même.

1.2. Les relations

Une relation est un type de données associé à un couple d'objets appelés origine et cible de la relation. Une relation ne possède pas d'occurrence à priori *. Celles-ci seront créées, puis pourront être modifiées ou supprimées grâce aux primitives que nous définirons en (3.2 et 3.3).

Une occurrence de relation est associée à un couple de réalisations de l'origine et de la cible de la relation, appelées origine et cible de

---

* Il n'y a évidemment aucune différence entre une réalisation et une occurrence d'un type de données. Nous réserverons le premier terme aux objets et le second aux relations dans le but de clarifier l'exposé.

l'occurrence. De plus, deux occurrences d'une même relation sont toujours associées à deux couples distincts de réalisations et sont donc discernables.

A toute relation est associée un quadruplet i-j, k-l de nombres entiers ou infinis qui détermine la règle suivante :

- toute réalisation de l'objet origine est origine d'au moins i et d'au plus j occurrences de la relation;
- toute réalisation de l'objet cible est cible d'au moins k et d'au plus l occurrences de la relation.

(En conséquence : $i \leqslant j$ et $k \leqslant l$).

Deux relations peuvent être liées par la règle ci-dessous, elles sont alors dites inverses : la cible de l'une est origine de l'autre et réciproquement; de plus, si une occurrence de l'une de ces relations est associée à un couple (a,b) de réalisations, alors, il existe une occurrence de l'autre relation, appelée occurrence inverse, qui est associée au couple (b,a)*. Une relation possède, au plus, une inverse.

Remarques : en vertu de la règle ci-dessus, si une relation est caractérisée par un quadruplet i-j,k-l, son inverse est caractérisée par k-l, i-j. Une relation peut être sa propre inverse : dans ce cas, elle est dite symétrique.

2. Classification des relations - Règles de cohérence

On peut grouper les relations en classes suivant celles de l'origine et de la cible. A chacune de ces classes correspondent des restrictions sur l'existence des relations. Parmi celles-ci, on trouve un certain nombre de règles de cohérence qui assurent la validité des actions primitives et d'autres qui ont été ajoutées lors de la première implémentation du modèle.

---

* Dans la suite, nous parlerons par abus de langage, de l'occurrence (a,b) au lieu de l'occurrence associée à (a,b).

2.1. *Relations ayant pour origine la racine et pour cible un objet complexe (OR → OC)*

Tout objet complexe est cible d'au plus une relation, issue de la racine, de quadruplet 0-∞, 1-1. Cette relation n'a pas d'inverse.

2.2. *Relations entre objets complexes (OC → OC)*

Le quadruplet i-j, k-l d'une relation entre objets complexes satisfait aux conditions suivantes :

- $i = 0$ ou $k = 0$,
- $i, k \in \mathbb{N}$,
- $j, l \in \mathbb{N}^+ \cup \{\infty\}$.

La règle qui suit est, en quelque sorte, une généralisation de la précédente. Elle assure que l'on pourra créer des réalisations de tout objet complexe.

Etant donnés n+1 objets complexes $O_0, O_1, \ldots, O_n$
et n+1 relations $R_0, R_1, \ldots, R_n$ tels que

- ou bien O_α est origine de R_α,
  $O_{\alpha+1 (\mathrm{mod}\ n+1)}$ est cible de R_α,
  R_α possède le quadruplet $i_\alpha - j_\alpha, k_\alpha - l_\alpha$;
- ou bien O_α est cible de R_α,
  $O_{\alpha+1 (\mathrm{mod}\ n+1)}$ est origine de R_α,
  R_α possède le quadruplet $k_\alpha - l_\alpha, i_\alpha - j_\alpha$ $(\alpha = 0, \ldots, n)$;

il existe β, γ tels que $0 \leqslant \beta, \gamma \leqslant n$ et $i_\beta = k_\gamma = 0$.

2.3. *Relations d'un objet complexe vers un objet élémentaire (OC → OE)*

Tout objet élémentaire est :

- soit attribut d'un et un seul autre objet élémentaire ;
- soit cible d'une relation issue d'un objet complexe.

Dans ce dernier cas, la relation est caractérisée par un quadruplet i-j,0-1 et satisfait à l'un des trois groupes de conditions suivants :

(1) - $i = 0$ ou 1 ;
- $1 \leqslant j < \infty$;
- $1 = \infty$;
- la relation n'a pas d'inverse.

(2) - $i = 0$ ou 1 ;
- $j = 1$;
- $1 = \infty$;
- la relation possède une inverse.

(3) - $i = j = 1 = 1$;
- la relation possède une inverse.

2.4. *Relations d'un objet élémentaire vers un objet complexe (OE → OC)*

Toute relation de ce type est inverse d'une relation de la catégorie précédente.

Remarque : Enfin, il n'existe pas de relation ayant pour cible la racine ni de relation de la racine vers un objet élémentaire.

3. Les actions primitives

Il existe cinq actions primitives agissant sur les types de données du modèle d'accès.

3.1. *Primitive d'accès*

Cette primitive qui permet la "recherche de l'information" est à l'origine du nom du modèle. Elle explique aussi l'utilisation de la terminologie "relation d'accès" pour les relations du modèle d'accès.

Etant données une relation et une réalisation o de son origine, la primitive d'accès fournit pour résultat l'ensemble ordonné des réalisations cibles des occurrences de la relation, ayant o pour origine. Cet ordre peut être défini à priori (c'est-à-dire associé à la relation) ou au moment de l'exécution de la primitive d'accès. Nous ne le préciserons pas davantage ici.

Remarques

- On voit que la primitive d'accès fait jouer un rôle différent à l'origine et à la cible. C'est pour cette raison que l'on a défini la notion de relation inverse.
- Les réalisations que l'on peut connaître à priori, de façon à réaliser une première application de la primitive d'accès, sont :
  - . la réalisation de la racine (qui est unique);
  - . les valeurs d'objets élémentaires.

3.2. *Primitives associées aux objets complexes*

Dans ce paragraphe et dans le suivant, nous utiliserons quatre actions élémentaires dont la signification est donnée ci-après. Ces actions ne sont pas des primitives du modèle car elles n'en respectent pas les règles de cohérence. Elles servent uniquement à la définition des primitives et n'ont pas de sens en dehors de ces définitions.

Voici la définition de ces actions :

- création d'une réalisation d'un objet complexe : une nouvelle réalisation de l'objet est créée; elle n'est ni origine, ni cible d'aucune occurrence de relation;
- suppression d'une réalisation d'un objet complexe : on supprime une réalisation d'objet qui n'est plus ni origine ni cible d'aucune occurrence;
- création d'une occurrence de relation : étant donné un couple de réalisations de l'origine et de la cible, qui n'est pas associé à une occurrence de la relation, on crée une telle occurrence;
- suppression d'une occurrence de relation : on supprime une occurrence de la relation associée à un couple de réalisations de l'origine et de la cible.

De plus, dans ce paragraphe, nous considérerons un objet complexe O,

- origine d'un certain nombre de relations $R_1,\ldots,R_m$, de cibles $C_1,\ldots,C_m$, caractérisées par des quadruplets i_α-j_α,k_α-l_α ($\alpha = 1,\ldots,m$);
- cible d'un certain nombre de relations $R_{m+1},\ldots,R_n$, d'origines $C_{m+1},\ldots,C_n$, sans inverse, de quadruplets k_α-l_α,i_α-j_α ($\alpha = m+1,\ldots,n$).

3.2.1. Primitive de création d'une réalisation d'objet complexe

Etant données i_α réalisations $c_\alpha^1,\ldots,c_\alpha^{i\alpha}$ de chaque objet C_α, cibles chacune (ou origines) de moins de l_α occurrences de R_α, *la primitive de création d'une réalisation*, appliquée à l'objet O,

- crée une nouvelle réalisation o de l'objet O;
- crée, pour chaque relation R_α ($\alpha = 1,\ldots,m$), i_α occurrences d'origine o et de cibles $c_\alpha^1,\ldots,c_\alpha^{i\alpha}$ ainsi que les occurrences inverses si R_α possède une inverse;
- crée, pour chaque relation R_α ($\alpha = m+1,\ldots,n$), i_α occurrences d'origines $c_\alpha^1,\ldots,c_\alpha^{i\alpha}$ et de cible o.

On vérifiera aisément que cette primitive, comme les suivantes, respecte les règles associées aux i-j,k-l et aux relations inverses.

3.2.2. Primitive de suppression d'une réalisation d'objet complexe

Etant donné une réalisation o de O,

- origine de i_α occurrences de R_α ($\alpha = 1,\ldots,m$),
- cible de i_α occurrences de R_α ($\alpha = m+1,\ldots,n$),

la primitive de suppression d'une réalisation, appliquée à o

- supprime toutes les occurrences de relation dont o est origine ou cible ;
- supprime la réalisation o.

3.3. *Primitives associées aux relations*

Soit une relation R d'origine O et de cible C caractérisée par le quadruplet i-j,k-l.

3.3.1. Primitive de création d'une occurrence de relation

Etant données

- une réalisation o de O, origine de moins de j occurrences de R,
- une réalisation c de C, cible de moins de l occurrences de R,

le couple (o,c) n'étant pas associé à une occurrence de la relation, *la primitive de création d'une occurrence,* appliquée à R

- crée l'occurrence (o,c) de la relation R;
- crée l'occurrence inverse si R possède une inverse.

3.3.2. Primitive de suppression d'une occurrence de relation

Etant donnée une occurrence (o,c) de R telle que

- o est origine de plus de i occurrences de R,
- c est cible de plus de k occurrences de R,

la primitive de suppression d'une occurrence, appliquée à (o,c) supprime cette occurrence et son inverse si elle existe.

3.3.3. Primitive de modification d'une occurrence de relation

Cette primitive s'applique uniquement aux relations telles que i ou $k \neq 0$. Supposons, pour fixer les idées, que $i \neq 0$; on transposera aisément la définition au cas où $k \neq 0$ en permutant les termes "origine" et "cible".

Etant données,

- une réalisation o de O, origine de i occurrences de R,
- une réalisation c_1, cible d'une de ces occurrences,
- une réalisation c_2 de C, cible de moins de 1 occurrences de R et d'aucune occurrence d'origine o,

la primitive de modification d'une occurrence, appliquée à (o,c_1) et à c_2,

- supprime l'occurrence (o,c_1) de R et son inverse si elle existe;
- crée l'occurrence (o,c_2) de R et son inverse, si R possède une inverse.

4. Désignation des Objets et des Relations

La façon de désigner un objet ou une relation est, à priori, sans importance et, au cours de l'exposé qui précède, nous les avons désignés en utilisant des notations semblables aux notations mathématiques. Ces notations ne font pas partie du modèle : on aurait pu en utiliser d'autres pour définir les mêmes notions. Toutefois, lorsque l'on définira des langages de programmation permettant de déclarer ou de manipuler les différents éléments du modèle d'accès, il faudra se donner des règles précises permettant de désigner, sans ambiguïté, chaque objet et chaque relation.

Ce sont ces modes de désignation que nous allons définir maintenant et que nous utiliserons le plus souvent par la suite.

Nous utiliserons dans ce qui suit, le terme *nom* pour désigner toute chaîne finie de caractères alphanumériques, commençant par une lettre.

Afin de désigner les objets et les relations, nous associerons un nom à tout objet et, éventuellement, aux relations entre objets complexes.

Nous donnons maintenant un ensemble de règles que doivent vérifier ces noms et, ensuite, les modes valides de désignation.

4.1. *Règles relatives aux noms*

4.1.1. Objet racine

Les racines de tous les schémas auxquels on s'intéresse ont des noms distincts. Les règles qui suivent sont relatives à un seul schéma.

4.1.2. Objets complexes

Tous les objets complexes ont des noms distincts.

4.1.3. Objets élémentaires

Appelons *descendant d'un objet complexe*, soit un objet élémentaire relié à cet objet complexe, soit un descendant d'un tel objet élémentaire. Appelons *liste de désignation d'un objet élémentaire*, une liste $(n_1,\ldots,n_\alpha)$ telle que :

- n_1 est le nom d'un objet complexe O_1 ;
- n_i est le nom d'un descendant O_i de O_{i-1} $(i = 2,\ldots,\alpha)$;
- O_α est l'objet élémentaire concerné.

Appelons, enfin, *liste stricte de désignation d'un objet élémentaire*, la plus longue de ses listes de désignation.

La règle suivante sur les noms d'objets élémentaires doit être vérifiée :

> Etant donné un objet élémentaire, il n'existe aucun autre objet élémentaire ayant une liste de désignation égale à la liste stricte de désignation du premier.

4.1.4. Relations entre objets complexes

Etant donnés deux objets complexes non nécessairement distincts, il existe, au plus, une relation sans nom ayant pour origine un de ces objets et pour cible l'autre. De plus, deux relations nommées de même origine et de même cible ont des noms distincts.

4.2. *Modes de désignation*

4.2.1. Objet racine

Un schéma et l'objet racine qu'il contient peuvent être désignés par le nom de ce dernier.
Les modes de désignation suivants sont valables à l'intérieur d'un schéma.

4.2.2. Objets complexes

Un objet complexe peut être désigné par son nom.

4.2.3. Objets élémentaires

Un objet élémentaire peut être désigné par sa liste stricte de désignation. Il peut aussi être désigné par toutes ses listes de désignation qui ne caractérisent pas, aussi, un autre objet élémentaire.

4.2.4. Relations

Les modes de désignation suivants sont valables à l'intérieur d'une classe de relations (OC → OC, OC → OE, etc.).

- Une relation sans nom peut être désignée par le couple :
  ("nom de l'origine","nom de la cible")
- Une relation nommée peut être désignée par le triplet :
  ("nom de l'origine","nom de la relation","nom de la cible")

  que nous écrirons plutôt sous la forme
  "nom de la relation"("nom de l'origine","nom de la cible").

5. Graphe d'accès

Etant donné un schéma de Base de données, dont les éléments sont nommés conformément aux règles ci-dessus, on définit un diagramme appelé "Graphe d'accès de la Base de données" et qui permet de "visualiser la structure du schéma".

Ce diagramme est construit de la façon suivante :

- à tout objet correspond un rectangle contenant son nom.
- à toute relation correspond une flèche reliant le rectangle de l'origine à celui de la cible. Si la relation est nommée, la flèche est interrompue par un ovale contenant le nom de la relation.
- tout rectangle représentant un objet élémentaire, est origine d'une flèche vers chaque rectangle représentant un de ses attributs.

Comme son nom l'indique, ce formalisme explicite les accès que l'on peut réaliser, "en suivant les flèches". Bien qu'il ne décrive qu'incomplètement le schéma d'une Base de données, nous l'utiliserons souvent par la suite en l'accompagnant d'informations complémentaires. Il remplacera le DDL jusqu'à la définition de celui-ci. On trouvera au paragraphe 3 du chapitre II un exemple de graphe d'accès.

CHAPITRE II : REPRESENTATION DES TYPES DE DONNEES DU MODELE D'ACCES PAR UN SCHEMA : LE METASCHEMA SEMANTIQUE

1. Introduction

Notre but, dans ce travail, est de réaliser une "implémentation du modèle d'accès par lui-même". Précisons ce terme.

Tout d'abord, que signifie "implémenter le modèle d'accès" ? Cette démarche nécessite trois phases :

- définir un DDL et un DML,
- définir une représentation interne des types de données du modèle d'accès, c'est-à-dire la façon dont ils seront réalisés sur ordinateur,
- réaliser deux compilateurs, un pour le DDL et un pour le DML.

Remarquons que les deux premiers points, ci-dessus, sont indépendants alors que le troisième dépend des deux premiers. Le compilateur du DDL construira un certain nombre de "tables" représentant le schéma d'une base de données et décrivant la représentation interne de ses différents éléments. Ces "tables" constituent le *schéma interne* de la base.
Le compilateur du DML traduira un programme DML en un programme exploitable par le software de l'ordinateur. Ce programme travaillera sur la représentation interne des éléments du modèle d'accès. Pour ce faire, le compilateur du DML exploitera le schéma interne.

Nous pouvons maintenant préciser ce que nous entendons par "implémentation du modèle d'accès par lui-même". Il s'agit de réaliser les trois points ci-dessus, mais

- en réalisant le schéma interne sous forme d'une base de données du modèle d'accès,
- en réalisant les compilateurs du DDL et du DML par des programmes qui ex-

ploitent cette base de données au moyen du DML.

Puisque le schéma interne est une base de données, il possède lui aussi un schéma que nous appellerons *métaschéma*. La suite du mémoire est consacrée presqu'uniquement à la définition et à l'exploitation de ce métaschéma.

2. Comment définir le métaschéma ?

Une première façon de déterminer le métaschéma serait d'examiner les besoins du compilateur du DML et de réaliser, par approximations successives, la définition des objets et des relations nécessaires.

Nous allons procéder d'une façon un peu différente. Avant même d'avoir défini une représentation interne des types de données, le DDL ou le DML, nous allons tenter de représenter les types de données du modèle d'accès par un schéma.

La méthode utilisée sera approximativement la suivante :

- les concepts pouvant prendre des valeurs seront représentés par des objets élémentaires,
- les concepts ne pouvant pas prendre de valeur seront représentés par des objets complexes,
- les associations entre différents concepts seront représentées par des relations.

Nous verrons qu'une partie des propriétés et des règles du modèle d'accès pourront s'exprimer grâce aux "i-j,k-l". D'autres règles, plus compliquées, ne pourront pas être exprimées de cette façon et correspondront à des "contraintes d'intégrité" sur les réalisations du métaschéma. Nous introduirons, de plus, dans le métaschéma des éléments représentant les moyens de désignation que nous avons définis.

Le métaschéma, ainsi constitué, servira de point de départ à la détermination du métaschéma complet qui, lui, contiendra un certain nombre d'objets et de relations supplémentaires associés à l'implémentation. De plus, il contiendra toutes les *relations d'accès* nécessaires aux compilateurs. Il faut, d'ailleurs, signaler que, dans l'étude que nous allons réaliser maintenant, le modèle d'accès joue davantage le rôle d'un "modèle sémantique". En particulier, deux relations inverses représentant la même association de concepts, nous ne définirons chaque fois qu'une seule de ces deux relations. Le "sens" des accès sera défini, plus tard, lors de l'étude des compilateurs.

Cette façon de procéder présente, à notre avis, les avantages suivants :

- Le métaschéma sémantique fournit une base de travail très appréciable pour l'étude du métaschéma nécessaire au compilateur du DML. Etant donné une fonction que celui-ci doit réaliser, on examinera les éléments concernés du métaschéma sémantique qu'on enrichira éventuellement mais que l'on ne devra pas redéfinir complètement.
- On pourra mieux distinguer les parties du compilateur qui sont indépendantes d'une représentation interne particulière : ce sont celles qui nécessitent uniquement des données décrites par le métaschéma sémantique.
- On peut, éventuellement, espérer que le métaschéma sémantique puisse servir de base à un compilateur indépendant d'une représentation interne particulière. Outre la description du modèle d'accès, le métaschéma complet contiendrait, alors, la description des représentations internes possibles. Mais nous n'avons pas envisagé cette étude.

3. Le métaschéma sémantique

3.1. *Les objets complexes du métaschéma sémantique*

La méthode esquissée ci-dessus, ne conduit malheureusement pas d'une façon unique à la détermination du métaschéma sémantique.

En effet, on pourrait, d'abord, considérer que les deux concepts fondamentaux du modèle d'accès sont ceux d'objet et de relation et définir un métaschéma contenant seulement deux objets complexes : OBJET et RELATION. Cette approche serait, sans doute, valable pour une version du modèle d'accès moins restrictive que celle que nous avons définieet a, d'ailleurs, été envisagée dans la référence [2].

Nous préférerons représenter ici les trois classes d'objets par trois objets complexes différents que nous nommerons RAC, OC et OE, parce que ces trois classes correspondent à des propriétés très différentes des objets, notamment en ce qui concerne les relations qui leur sont associées.

Pour celles-ci, nous pourrions, de même,

- soit définir un objet complexe représentant toutes les classes de relations,
- soit définir un objet complexe différent par classe de relation.

Nous utiliserons plutôt une troisième façon de faire qui tient davantage compte des propriétés fort particulières des relations autres que celles entre objets complexes. Finalement, le métaschéma contiendra un objet complexe REL qui représentera seulement les relations entre objets complexes.

En résumé, le métaschéma possèdera quatre objets complexes : RAC, OC, OE, REL dont une réalisation représentera, respectivement, une racine, un objet complexe, un objet élémentaire ou une relation entre objets complexes.

3.2. *Les relations et les objets élémentaires du métaschéma sémantique*

Nous allons, maintenant, reprendre la définition du modèle d'accès et nous attacher à représenter les propriétés et les associations des types de données par des objets élémentaires et des relations du métaschéma.

Auparavant, définissons trois nouvelles notations :

- si α désigne une relation, $\alpha \equiv$ i-j,k-l signifie que cette relation est caractérisée par le quadruplet i-j,k-l;
- si β désigne une autre relation, $\alpha \equiv \beta^*$ signifie que les deux relations sont inverses;
- si γ désigne un objet élémentaire, val(γ) désigne son ensemble de valeurs.

Signalons encore que, dans la suite, nous désignerons souvent un objet élémentaire par son nom, s'il n'y a pas d'ambiguïté.

3.2.1. Propriétés du schéma

Le fait qu'un schéma de base de données se compose d'un objet racine et d'un nombre fini d'objets complexes, d'objets élémentaires, et de relations conduit à définir les trois relations suivantes du métaschéma :

(RAC,OC) $\equiv$ 0-∞,1-1,
(RAC,OE) $\equiv$ 0-∞,1-1,
(RAC,REL) $\equiv$ 0-∞,1-1.

On peut, en effet, identifier schéma et objet racine.

3.2.2. Ensemble de valeurs d'un objet élémentaire

Pour représenter l'ensemble de valeurs d'un objet élémentaire, le métaschéma contiendra :

- un objet élémentaire DESCRIPTEUR dont chaque valeur représentera une règle définissant un ensemble de valeurs (nous le préciserons plus loin);

- un objet élémentaire J tel que val(J) = $\mathbb{N}^+$;
- trois relations

$$\begin{aligned} &(OE,DESCRIPTEUR) \equiv 0\text{-}1,0\text{-}\infty, \\ ATT\ &(OE,OE) \equiv 0\text{-}\infty,0\text{-}1, \\ &(OE,J) \equiv 1\text{-}1,0\text{-}\infty. \end{aligned}$$

Une réalisation de OE, origine d'une occurrence de (OE,DESCRIPTEUR), représente un objet élémentaire dont l'ensemble de valeurs est défini par la valeur de DESCRIPTEUR, cible de l'occurrence.
Une réalisation de OE, origine d'une ou plusieurs occurrences de ATT (OE,OE), représente un objet élémentaire dont les attributs sont représentés par les cibles de ces occurrences. Les valeurs de J, associées aux cibles par (OE,J), représentent le facteur de répétition de ces attributs.

3.2.3. Propriétés des relations entre objets complexes

Une réalisation de REL représente une relation entre objets complexes dont l'inverse, l'origine et la cible seront représentées par les cibles des occurrences des relations

$$\begin{aligned} &INVERSE\ (REL,REL) \equiv 0\text{-}1,0\text{-}1 \quad \text{(relation symétrique)}, \\ &ORIGINE\ (REL,OC) \equiv 1\text{-}1,0\text{-}\infty, \\ &CIBLE\ \ (REL,OC) \equiv 1\text{-}1,0\text{-}\infty, \end{aligned}$$

dont la réalisation est origine.

La notion de quadruplet i-j,k-l sera représentée par quatre objets élémentaires et quatre relations :

$$\begin{aligned} &(REL,I) \equiv 1\text{-}1,0\text{-}\infty, \\ &(REL,J) \equiv 1\text{-}1,0\text{-}\infty, \\ &(REL,K) \equiv 1\text{-}1,0\text{-}\infty, \\ &(REL,L) \equiv 1\text{-}1,0\text{-}\infty, \end{aligned}$$

avec $val(I) = val(K) = \mathbb{N}$,
$val(J) = val(L) = \mathbb{N}^+ \cup \{\infty\}$.

3.2.4. Représentation et propriétés des autres relations

Toute relation de la racine vers un objet complexe sera représentée par une occurrence de la relation :

ACCES (RAC,OC) ≡ 0-∞,0-1.

Pour représenter les relations d'objet complexe vers objet élémentaire, nous introduisons un nouvel objet élémentaire et deux relations :

(OC,OE) ≡ 0-∞,0-1,
(OE,I) ≡ 0-1,0-∞,
val(I) = {0,1}.

Une occurrence de (OC,OE) représente une relation d'objet complexe vers objet élémentaire. La réalisation de OE, cible de cette occurrence est associée à une valeur de I et une de J qui sont les caractéristiques i-j de la relation. La caractéristique 1 de la relation vaut 1 si la réalisation de OE est origine d'une occurrence de la relation

ID(OE,OC) ≡ 0-1,0-∞,

sinon, elle est égale à ∞.
Une occurrence de ID (OE,OC) représente une relation d'objet élémentaire vers objet complexe de quadruplet 0-1,1-1.
Une occurrence de la relation

(OE,OC) ≡ 0-1,0-∞

représente une relation d'objet élémentaire vers objet complexe de quadruplet 0-∞,i-1.

3.2.5. Représentation des noms

Les noms d'objets et de relations seront représentés par quatre nouveaux objets élémentaires de nom "NOM" et quatre relations :

(RAC,NOM) ≡ 1-1,0-1,
(OC,NOM) ≡ 1-1,0-∞,
(REL,NOM) ≡ 0-1,0-∞,
(OE,NOM) ≡ 1-1,0-∞.

Le domaine de valeurs de ces quatre objets élémentaires est l'ensemble des chaînes alphanumériques commençant par une lettre.

3.3. *Graphe du métaschéma sémantique*

Nous appellerons MR la racine du métaschéma sémantique, ce qui termine sa définition. Son graphe est dessiné à la page suivante.

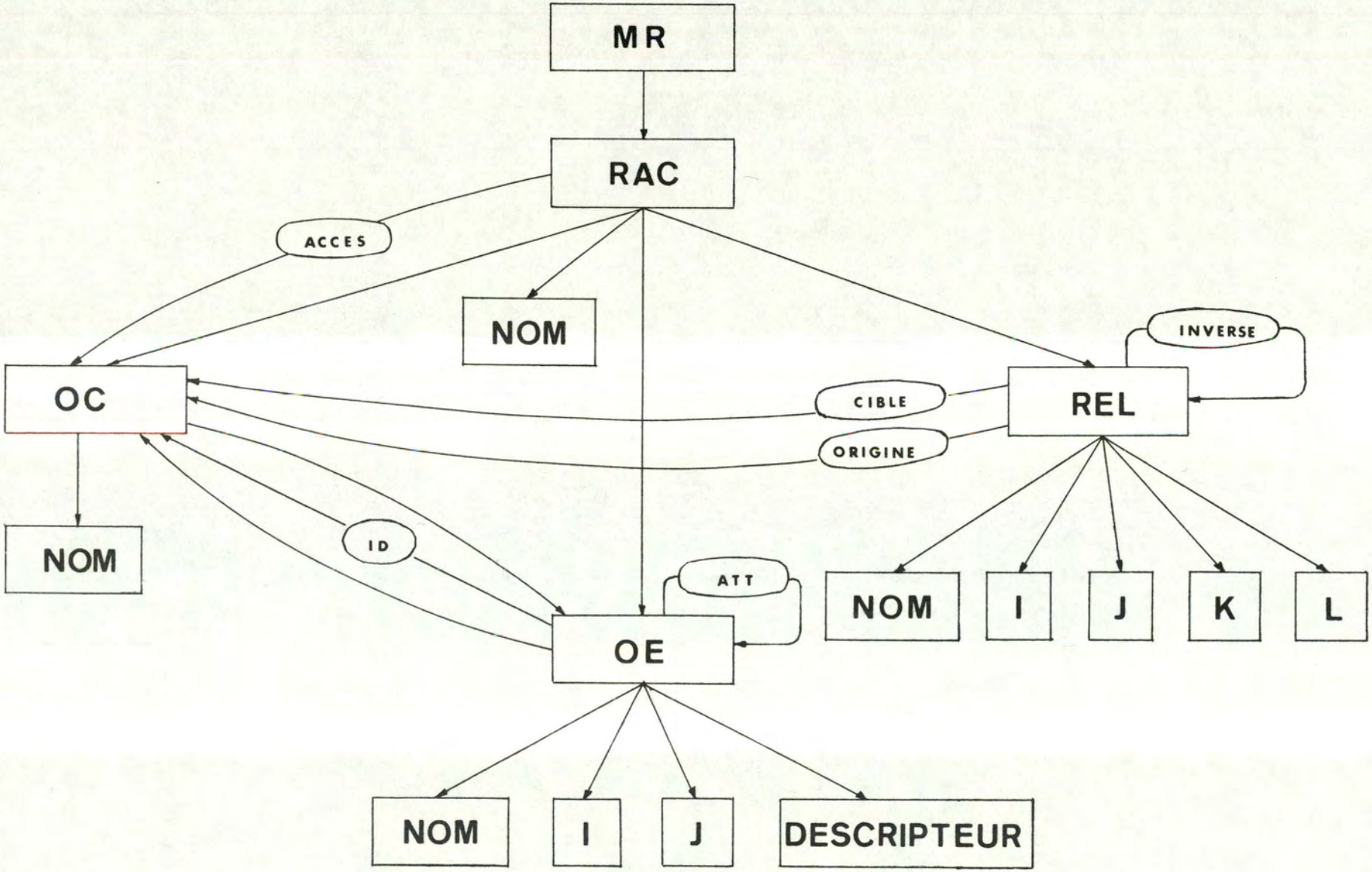
4. Contraintes d'intégrité relatives au métaschéma

En définissant le métaschéma comme nous l'avons fait, nous n'avons pu exprimer toutes les propriétés des types de données du modèle d'accès, ni toutes les règles de cohérence régissant les associations de ces types de données. Celles qui n'ont pu être exprimées vont, maintenant, être représentées par des contraintes d'intégrité sur les réalisations et les occurrences du métaschéma.

Finalement, ces contraintes se traduiront par des programmes constituant une partie du compilateur du DDL. Nous étudierons cela dans la quatrième partie.

Dans le but d'exprimer les contraintes d'intégrité de manière formalisée, nous introduisons maintenant quelques notations.

- Nous désignerons par des lettres minuscules les réalisations d'objets du métaschéma.
- Si α est une expression qui désigne un objet et β une expression qui désigne une réalisation, l'expression $\beta \in \alpha$ signifiera : "β est une réalisation de α".


MR
RAC
ACCES
NOM
OC
INVERSE
REL
CIBLE
ORIGINE
NOM
ID
ATT
NOM
I
J
K
L
OE
NOM
I
J
DESCRIPTEUR

- Si α est une expression qui désigne une relation et β une expression qui désigne une réalisation de son origine, l'expression $\alpha[\beta]$ désignera l'ensemble des cibles d'occurrences de α dont β est origine. Si cet ensemble contient un seul élément, celui-ci pourra être désigné par la même expression.
- Enfin, nous utiliserons des quantificateurs et d'autres symboles dont la signification est bien connue ou évidente.

4.1. *Contraintes d'intégrité associées au schéma*

Le fait qu'un type de données appartenant à un schéma, ne puisse pas être associé à un type de données appartenant à un autre schéma, n'est pas exprimé par le métaschéma. Il est donc nécessaire de le traduire par les contraintes d'intégrité suivantes :

- L'origine et la cible d'une relation appartiennent au même schéma qu'elle même.
- Les attributs d'un objet élémentaire appartiennent au même schéma que lui-même.

Formellement, on a :

$\forall r_1 \in REL, \exists r_2 \in RAC , \exists c_1,c_2 \in OC$ tels que

$c_1 = ORIGINE(REL,OC)[r_1]$,
$c_2 = CIBLE(REL,OC)[r_1]$,
$c_1,c_2 \in (RAC,OC)[r_2]$,
$r_1 \in (RAC,REL)[r_2]$. (C1)

$\forall e \in OE , \exists r \in RAC , \exists c \in OC$ tels que

$c \in (RAC,OC)[r]$,
$e \in (RAC,OE)[r]$ et
ou bien $e \in (OC,OE)[c]$,

ou bien $\exists e_1,\ldots,e_n \in OE$ tels que

$$e_1 \in (OC,OE)[c],$$
$$e_{i+1} \in ATT(OE,OE)[e_i] \quad (i = 1,\ldots,n-1),$$
$$e \in ATT(OE,OE)[e_n]. \qquad (C2)$$

$$\forall c \in OC, \forall r \in RAC \text{ tels que } c \in ACCES(RAC,OC)[r],$$
$$c \in (RAC,OC)[r]. \qquad (C3)$$

4.2. *Contraintes d'intégrité associées aux objets élémentaires*

Les règles suivantes du modèle d'accès ne sont pas représentées par le métaschéma. Nous les faisons suivre des contraintes d'intégrité qui les expriment.

- Le domaine de valeurs d'un objet élémentaire est défini soit par une règle qui ne fait pas référence à un autre type de données du modèle d'accès, soit par une liste d'objets attributs.

$$\forall e \in OE, \text{ ou bien } (OE,DESCRIPTEUR)[e] \neq \phi,$$
$$\text{ou bien } ATT(OE,OE)[e] \neq \phi. \qquad (C4)$$

- Un objet élémentaire n'est pas descendant de lui-même

$$\forall e_1,\ldots,e_n \in OE \text{ tels que } e_{i+1} \in ATT(OE,OE)[e_i] \quad (i=1,\ldots,n-1),$$
$$e_1 \neq e_n. \qquad (C5)$$

4.3. *Contraintes d'intégrité associées aux relations entre objets complexes*

- Le quadruplet i-j,k-l d'une relation entre objets complexes est tel que $i \leqslant j$, $k \leqslant l$ et $i = o$ ou $k = o$.

$$\forall r \in REL, (REL,I)[r] \leqslant (REL,J)[r],$$
$$(REL,K)[r] \leqslant (REL,L)[r] \text{ et}$$
$$(REL,I)[r]=0 \text{ ou } (REL,K)[r] = 0. \qquad (C6)$$

- Si deux relations sont inverses, la cible de l'une est l'origine de l'autre et leurs quadrupletsi-j,k-l se déduisent l'un de l'autre en permutant i et k, j et l.

$$\forall r_1, r_2 \in \text{REL} \text{ tels que } r_1 = \text{INVERSE (REL,REL)}[r_2],$$
$$\text{ORIGINE (REL,OC)}[r_1] = \text{CIBLE (REL,OC)}[r_2],$$
$$\text{(REL,I)}[r_1] = \text{(REL,K)}[r_2],$$
$$\text{(REL,J)}[r_1] = \text{(REL,L)}[r_2]. \qquad \text{(C7)}$$

- Nous donnons ici la contrainte d'intégrité représentant la règle exposée au paragraphe 2.2 du chapitre consacré à la définition du modèle d'accès.

$$\forall c_0, \ldots, c_n \in \text{OC},$$
$$\forall r_0, \ldots, r_n \in \text{REL},$$
$$\forall i_0, \ldots, i_n, k_0, \ldots, k_n \in \mathbb{N},$$
$$\text{tels que ou bien } c_\alpha = \text{ORIGINE (REL,OC)}[r_\alpha],$$
$$c_{\alpha+1(\text{mod } n+1)} = \text{CIBLE (REL,OC)}[r_\alpha],$$
$$i_\alpha = \text{(REL,I)}[r_\alpha],$$
$$k_\alpha = \text{(REL,K)}[r_\alpha];$$
$$\text{ou bien } c_\alpha = \text{CIBLE (REL,OC)}[r_\alpha],$$
$$c_{\alpha+1(\text{mod } n+1)} = \text{ORIGINE (REL,OC)}[r_\alpha],$$
$$i_\alpha = \text{(REL,K)}[r_\alpha],$$
$$k_\alpha = \text{(REL,I)}[r_\alpha]; \quad (\alpha = 0,\ldots,n)$$
$$\exists \beta, \gamma \in \{0,\ldots,n\} \text{ tels que } i_\beta = k_\gamma = 0. \qquad \text{(C8)}$$

4.4. *Contraintes d'intégrité associées aux relations entre objet complexe et objet élémentaire*

$$\forall e \in \text{OE}, \forall c \in \text{OC} \text{ tels que } e \in \text{(OC,OE)}[c],$$
$$\text{(OE,I)}[e] \neq \phi \text{ et}$$
$$\text{ou bien } \text{ID(OE,OC)}[e] = \text{(OE,OC)}[e] = \phi;$$
$$\text{ou bien } \text{ID(OE,OC)}[e] = \phi,$$
$$\text{(OE,OC)}[e] = c,$$
$$\text{(OE,J)}[e] = 1;$$

ou bien $ID(OE,OC)[e] = c$,
$(OE,OC)[e] = \phi$,
$(OE,I)[e] = (OE,J)[e] = 1$. (C9)

$\forall e_1,e_2 \in OE$, tels que $e_1 \in ATT(OE,OE)[e_2]$,
$(OE,I)[e_1] = ID(OE,OC)[e_1] = (OE,OC)[e_1] = \phi$. (C10)

4.5. *Contraintes d'intégrité associées aux noms*

- Deux objets complexes d'un même schéma ont des noms différents.

$\forall r \in RAC, \forall c_1,c_2 \in OC$ tels que $c_1,c_2 \in (RAC,OC)[r]$ et $c_1 \neq c_2$,
$(OC,NOM)[c_1] \neq (OC,NOM)[c_2]$. (C11)

- Deux relations de même cible et de même origine ont des noms distincts.

$\forall c_1,c_2 \in OC, \forall r_1,r_2 \in REL$
tels que $r_1 \neq r_2$,
$c_1 = ORIGINE(REL,OC)[r_1] = ORIGINE(REL,OC)[r_2]$,
$c_2 = CIBLE(REL,OC)[r_1] = CIBLE(REL,OC)[r_2]$;
$(REL,NOM)[r_1] \neq (REL,NOM)[r_2]$. (C12)

- Etant donné un objet élémentaire, il n'existe aucun autre objet élémentaire ayant une liste de désignation égale à la liste de désignation stricte du premier.
Pour exprimer cette règle par une contrainte d'intégrité, nous définirons deux nouvelles notions; la première représentant la liste stricte de désignation d'un objet élémentaire, la seconde représentant l'ensemble de ses listes de désignation.

$\forall$ e $\in$ OE, définissons lstr[e] = $(n_0,\ldots,n_\alpha)$

tel que ou bien $\alpha = 1$ et

$\exists$ c $\in$ OC tel que e $\in$ (OC,OE)[c] ,

n_0 = (OC,NOM)[c] ,

n_1 = (OE,NOM)[e] ;

ou bien $\exists$ c $\in$ OC, $\exists$ $e_1,\ldots,e_{\alpha-1} \in$ OE

telsque $e_1 \in$ (OC,OE) [c] ,

$e_{i+1} \in$ ATT (OE,OE)[e_i] (i=1,...,α-2),

e $\in$ ATT (OE,OE) [$e_{\alpha-1}$] ,

n_0 = (OC,NOM)[c] ,

n_i = (OE,NOM)[e_i] (i=1,...,α-1),

n_α = (OE,NOM)[e] .

$\forall$ e $\in$ OE, définissons

$\llcorner$ [e] = $\{(n_0,n_{i_1},\ldots,n_{i_\beta},n_\alpha)$

tels que $0 < i_1 < \ldots < i_\beta < \alpha$, $(n_0,\ldots,n_\alpha)$ = lstr [e] $\}$.

Grâce à ces deux définitions, la règle ci-dessus peut s'exprimer facilement par la contrainte d'intégrité suivante :

$\forall$ $e_1,e_2 \in$ OE, $\forall$ r $\in$ RAC tels que $e_1,e_2 \in$ (RAC,OE)[r] et $e_1 \neq e_2$,

lstr[e_1] $\notin$ $\llcorner$ [e_2]. (C13)

5. Remarque relative aux contraintes d'intégrité

Nous avons, finalement, représenté les propriétés et les règles associées aux types de données du modèle d'accès par un (méta) schéma et un ensemble de contraintes d'intégrité. Le résultat obtenu nous paraît conduire à deux conclusions.

- Un schéma du modèle d'accès ne permet pas, à lui-seul, d'exprimer complètement une situation un tant soit peu complexe et il faut lui adjoindre des règles supplémentaires, c'est-à-dire des contraintes d'intégrité.
- L'utilisation d'un langage prédicatif semblable au "formalisme" que nous avons utilisé n'est, sans doute, pas la bonne façon d'exprimer les contraintes d'intégrité. Tout d'abord, la réalisation d'un tel langage risque d'être extrêmement compliquée. Ensuite, il est nécessaire d'écrire des expressions complexes pour exprimer des propriétés, malgré tout, assez simples. Notre opinion est qu'il ne faut pas distinguer langage d'expression de contraintes d'intégrité et DML, celui-ci devant pouvoir jouer les deux rôles. Nous illustrerons cela dans la quatrième partie où nous verrons, par exemple, que la structure de blocs du programme de chargement de la "méta" base de données assure, à elle seule, que les contraintes d'intégrité du paragraphe 4.1. sont bien vérifiées.

DEUXIÈME PARTIE : REPRÉSENTATION DES TYPES DE DONNÉES DU MODÈLE D'ACCÈS DANS UN SYSTÈME CIBLE : SESAM

INTRODUCTION

Dans cette partie du mémoire, nous allons définir ce que nous avons appelé une "représentation interne" des types de données du modèle d'accès. Toutefois, cette représentation ne fera pas directement référence à la structure physique d'une machine.
Au lieu de cela, nous utiliserons un système de gestion de fichiers (SESAM) déjà implémenté sur un ordinateur et nous représenterons chaque objet ou relation du modèle d'accès par une combinaison de types de données de ce système.

Pour cette raison, nous utiliserons le terme "représentation SESAM" des types de données du modèle d'accès, au lieu du terme "représentation interne".

CHAPITRE I : DESCRIPTION DU SYSTEME SESAM

SESAM est un système de création et de gestion de bases de données proposé par le constructeur SIEMENS.
Il peut supporter plusieurs bases de données désignées par des noms distincts.

1. Les types de données SESAM

Le schéma d'une base de données SESAM est fixé par la déclaration d'un certain nombre d'aspects et de listes inversées.

1.1. Les aspects

Un aspect est un type de données dont les réalisations (ou valeurs d'aspect) sont des chaînes de caractères de longueur fixe ($\leqslant 256$).
Chaque aspect est désigné par un nom de trois caractères.
La déclaration d'un aspect comporte, outre son nom et sa longueur, un certain nombre de caractéristiques qui définissent son ensemble de valeurs et la représentation interne de celles-ci : chaînes numériques, alphanumériques, chaînes de bytes; pouvant être compactées sur disque de différentes façons.

1.2. Les enregistrements

Une particularité du système SESAM est qu'il ne définit pas, à priori, de type d'enregistrement.
Chaque enregistrement sera composé :

- d'un numéro d'enregistrement (un mot de 32 bits),
- d'une valeur de l'aspect d'ordre,
- d'une valeur d'un certain nombre d'aspects.

Le numéro d'enregistrement est fourni par le système SESAM lors de la création de celui-ci et l'identifie.
L'aspect d'ordre a pour nom "AAA", sa valeur est fixée par l'utilisateur et identifie l'enregistrement. De plus, les enregistrements sont rangés dans la base de données par ordre croissant des valeurs de cet aspect.

1.3. *Les listes inversées*

A tout aspect, on peut associer une liste inversée qui fait correspondre à toute valeur de l'aspect, la liste des enregistrements qui possèdent cette valeur.

2. Manipulation de données en SESAM

2.1. *Méthodes d'accès aux enregistrements*

Accès séquentiel indexé

Etant donné une valeur entière ou partielle de l'aspect d'ordre, l'accès séquentiel indexé fournit les enregistrements dont l'aspect d'ordre comprend cette valeur, ou encore est inférieur ou supérieur à cette valeur.

Accès direct par numéro d'enregistrement

La connaissance d'un numéro d'enregistrement permet d'accéder à cet enregistrement.

Accès par fichiers inversés

Etant donné une valeur entière ou partielle d'un aspect inversé, l'accès par fichier inversé fournit les enregistrements possédant cette valeur d'aspect ou les enregistrements dont la valeur d'aspect est inférieure ou supérieure à celle donnée.

Accès d'après une combinaison de critères

Il est possible d'accéder aux enregistrements satisfaisant à une combinaison booléenne des critères précédents.

Remarque

Lorsque l'on accède à un ou plusieurs enregistrements, on spécifie une liste d'aspects et l'on n'obtient en retour que les valeurs des aspects ayant été cités dans cette liste.

2.2. *Instructions de mise à jour*

Il est possible de créer de nouveaux enregistrements en donnant au moins la valeur de l'aspect d'ordre.
Sur base de la valeur de l'aspect d'ordre ou sur base du numéro d'enregistrement, il est possible de supprimer un enregistrement ou de modifier ses valeurs d'aspects.

3. Initialisation et exploitation d'une base de données SESAM

3.1. *Les fichiers ORG et ZDR*

Deux fichiers principaux sont nécessaires à la constitution d'une base de donnees SESAM :

- un fichier appelé ZDR contenant les informations proprement dites de la base de données, c'est-à-dire, les enregistrements triés par valeurs croissantes de l'aspect d'ordre.
- un fichier ORG, contenant des informations propres à SESAM, lui permettant de gérer et de retrouver les données introduites. Ce fichier comprend notamment :
  + les listes inversées,
  + les tables de définition d'aspects,

+ les tables de correspondance entre :
  valeur d'aspect d'ordre et adresse disque,
  numéro d'enregistrement et adresse disque.

3.2. *Initialisation de la base de données*

L'initialisation d'une base de données SESAM doit être réalisée par un ensemble de programmes utilitaires. Parmi ces programmes, les deux principaux sont les suivants :

- SEFO permet de définir le schéma de la base de données. Ce programme admet en entrée un fichier qui contient la description de chaque aspect : nom, longueur, type de caractères, représentation interne, paramètre indiquant si l'aspect est inversé ou non.
- SEBE effectue le chargement initial de la base de données. L'entrée de ce programme est constituée de deux fichiers :
  - le premier contient les données proprement dites,
  - le second contient la description des formats dans lesquels ces données sont introduites.

  La base de données n'est pas exploitable avant le passage de cet utilitaire.

3.3. *Exploitation d'une base de données SESAM*

Après avoir été initialisée, une base de données SESAM peut être exploitée en faisant appel à un module de nom "SESAM" possédant quatre arguments :

La zone instruction
  Elle définit le type d'opération à effectuer.

La zone accusé de réception
  Elle indique si une réponse a été fournie et,sinon,quelles sont les erreurs. Elle contiendra, en outre, le numéro de l'enregistrement concerné par l'opération.

La zone réponse
  Lors d'une opération d'accès, SESAM range dans cette zone les valeurs

d'aspects d'un enregistrement dans un ordre spécifié par la zone instruction.

La zone question

L'utilisateur y indique les valeurs d'aspects qui complètent les questions posées dans la zone instruction.

CHAPITRE II : REPRESENTATION SESAM DES TYPES DE DONNEES DU MODELE D'ACCES

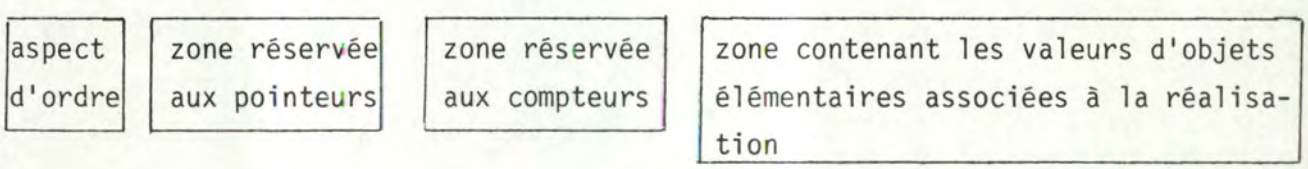
Nous allons maintenant décrire une représentation SESAM des types de données du modèle d'accès. Celle-ci a été définie par l'équipe Grands Fichiers . C'est sur cette représentation que travailleront les compilateurs du DDL et du DML que nous étudierons par la suite.

1. Représentation de l'objet racine

Le nom de l'objet racine définit le nom de la base de données SESAM. La réalisation de l'objet racine n'a pas, à proprement parler, de représentation. Il existe toutefois une correspondance biunivoque entre les réalisations d'objets racines et les bases de données SESAM.

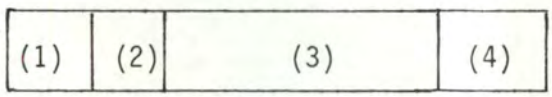
2. Représentation des objets complexes et des objets élémentaires

Une réalisation d'objet complexe est représentée par un enregistrement SESAM et possède la forme suivante :



2.1. *L'aspect d'ordre*

Cet aspect possède la structure particulière suivante :



(1) zone de trois chiffres contenant le numéro de l'objet complexe en correspondance biunivoque avec le nom.
(2) zone de deux caractères contenant "$$" (dont on verra l'utilité plus loin).
(3) zone contenant :
- soit une valeur d'objet élémentaire relié à l'objet complexe par une relation de quadruplet 1-1, 0-1,
- soit une valeur générée par le système modèle d'accès.
(4) zone vide complétant la précédente afin de lui donner une longueur fixe.

2.2. *Zone pointeurs*

Cette zone contient un certain nombre de pointeurs servant aux relations entre objets complexes (voir représentation des relations). Cinq aspects SESAM peuvent être réservés pour ces pointeurs ce qui permet d'en créer un maximum de 320. (un pointeur est un numéro d'enregistrement SESAM).

2.3. *Zone compteurs*

Cette zone contient les compteurs d'occurrences de relations issues de l'objet complexe; ces relations sont de deux types : (OC → OE) et (OC → OC).
Cinq aspects sont aussi disponibles pour ces compteurs qui, lors de mises à jour, permettent de vérifier que la règle relative aux i-j,k-l n'est pas transgressée.

2.4. *Zone contenant les valeurs des objets élémentaires associés à l'objet complexe*

Pour toute relation issue d'un objet complexe vers un objet élémentaire, on réserve de un à cinq aspects qui contiendront les valeurs de l'objet élémentaire, associées par la relation à une réalisation de l'objet complexe.

Les objets élémentaires attributs sont considérés comme une découpe plus fine des objets élémentaires directement reliés à l'objet complexe, et ne nécessitent donc aucune place supplémentaire en mémoire ni, par conséquent, aucun aspect décrivant cette place.

3. Représentation des relations

3.1. *Relations de l'objet racine vers les objets complexes*

Ces relations sont réalisées par la structuration de l'aspect d'ordre vue plus haut.
Un accès par une telle relation est réalisé par une interrogation SESAM avec critère d'égalité sur les cinq premiers caractères de l'aspect d'ordre.

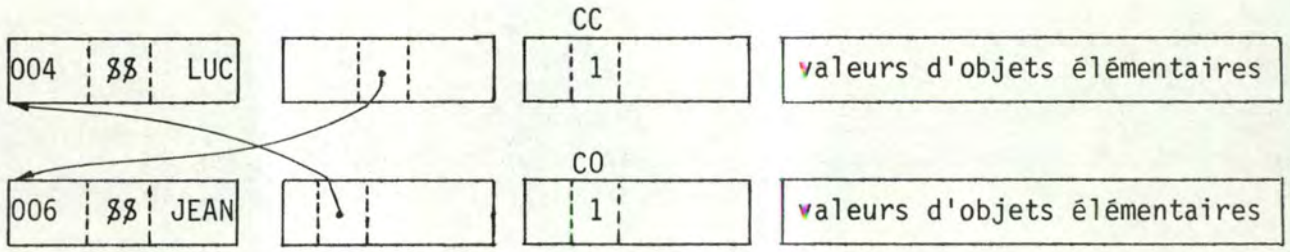
3.2. *Relations entre objets complexes*

Etant donnée une relation entre objets complexes,

- il existe un compteur des réalisations de l'objet complexe cible qui sont reliées à une réalisation de l'objet complexe origine par une occurrence de la relation.
  Ce compteur sera désignée par CC dans les figures;
  sa valeur ne peut qu'être comprise entre les caractéristiques i et j de la relation. Il sera placé dans la réalisation de l'objet complexe origine.
- il existe un compteur des réalisations de l'objet complexe origine qui sont reliées à une réalisation de l'objet complexe cible par une occurrence de la relation.
  Ce compteur sera désigné par CO dans les figures;
  sa valeur doit être comprise entre les caractéristiques k et l de la relation. Il sera placé dans la réalisation de l'objet complexe cible.

Nous allons passer en revue les différents types de relations pouvant survenir entre des objets complexes en précisant, pour chacune, leur représentation SESAM.

3.2.1. Relations de quadruplet i-1,k-1



La réalisation de l'objet complexe origine (de numéro 004) contient un pointeur vers la réalisation de l'objet complexe cible (de numéro 006). La réalisation cible, possède, de même, un pointeur vers la réalisation origine.

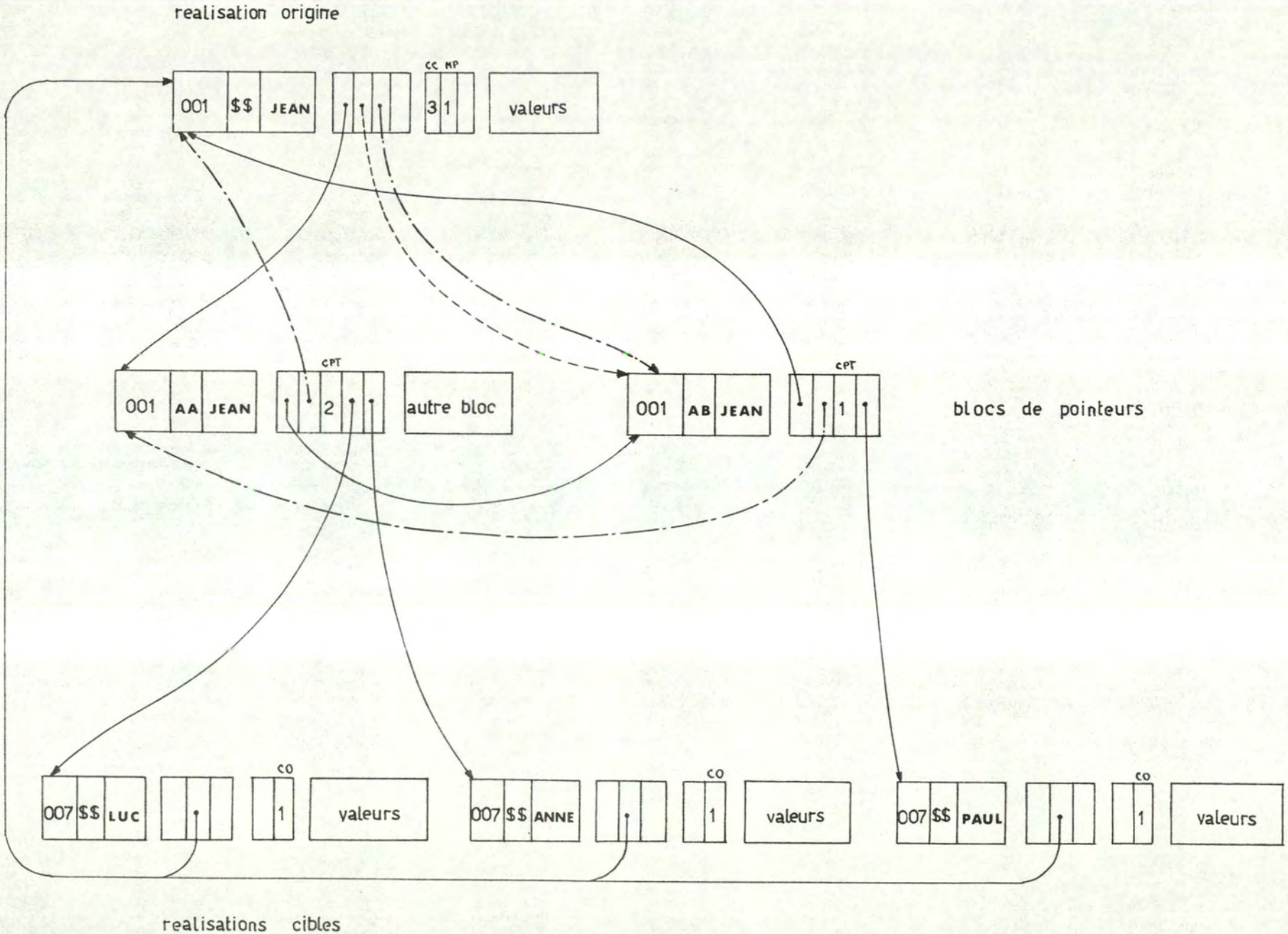
3.2.2. Relations de quadruplet i-j,k-1 (j > 1)

La réalisation de l'objet complexe origine (de numéro 001) est reliée, dans les deux sens, à une chaîne de blocs de pointeurs. Chaque bloc de pointeurs constitue une valeur d'un aspect de 256 caractères, contenue dans un enregistrement SESAM dont la structure de l'aspect d'ordre est identique à celle de la réalisation de l'objet complexe origine, excepté le fait que "$$" est remplacé par une combinaison de deux lettres, évoluant suivant l'ordre alphabétique, en fonction du nombre de blocs de pointeurs nécessaires à l'établissement des occurrences de relations.

Un aspect sera donc associé à chaque relation de ce type et les enregistrements SESAM décrits, ici, pourront posséder des valeurs d'aspects correspondant à toutes les relations issues de l'origine.

Un bloc de pointeurs contiendra un compteur, deux pointeurs de chaînage et au maximum 61 pointeurs vers des réalisations de la cible.

La réalisation origine contiendra un pointeur vers le premier bloc de pointeurs, un deuxième vers le dernier bloc et un troisième vers le bloc le "moins plein", c'est-à-dire celui qui comprend le plus petit nombre de


realisation origine
001 $$ JEAN
CC MP
3 1
valeurs
001 AA JEAN
CPT
2
autre bloc
001 AB JEAN
CPT
1
blocs de pointeurs
007 $$ LUC
CO
1
valeurs
007 $$ ANNE
CO
1
valeurs
007 $$ PAUL
CO
1
valeurs
realisations cibles

pointeurs vers des réalisations de l'objet cible. Celles-ci contiendront, seulement, un pointeur vers la réalisation origine. De plus, le nombre de pointeurs du bloc "le moins plein" sera mémorisé dans la réalisation origine.

Remarque

La représentation des relations i-1,k-l se déduit de celle des relations i-j,k-1 (l > 1).

3.2.3. Relations de quadruplet i-j,k-l (où j > 1 et l > 1)

Dans ce cas, les réalisations de la cible sont, elles aussi, reliées à une chaîne double de blocs de pointeurs. On en déduit facilement la représentation de ces relations. En particulier, deux aspects seront associés à une relation de ce type; l'un des aspects contiendra les blocs de pointeurs vers les réalisations cibles et l'autre les blocs de pointeurs vers les réalisations origines.

3.2.4. Remarque générale concernant les relations inverses

Dans ce qui précède, nous avons décrit, chaque fois, non seulement la représentation de la relation directe mais aussi celle de la relation inverse. Celle-ci sera donc toujours présente, même si elle n'existe pas pour l'utilisateur. Celui-ci ne pourra pas la manipuler, mais le système de gestion du modèle d'accès l'utilisera,surtout lors de mises à jour.

3.3. *Relations d'objet complexe vers objet élémentaire*

Cette relation est réalisée par l'inclusion des valeurs des objets élémentaires dans la réalisation de l'objet complexe. Celle-ci contiendra, en outre, un compteur du nombre de valeurs de l'objet élémentaire qui lui sont associées.

Le compteur inverse n'est pas nécessaire car la caractéristique 1 de la relation (OC → OE) ne peut prendre que les valeurs 1 ou ∞. Si 1 vaut 1, le dépassement de 1 par tentative de création sera détecté par SESAM et la création sera refusée.
Si 1 vaut ∞, aucune vérification ne doit être réalisée.

3.4. *Relations d'objet élémentaire vers objet complexe*

3.4.1. Relations de quadruplet 0-1,1-1

Une seule relation de ce type peut exister pour un objet complexe, c'est-à-dire qu'un seul objet élémentaire peut identifier l'objet complexe. La valeur de cet objet élémentaire étant incluse dans l'aspect d'ordre, l'accès direct peut être réalisé sur base de cet aspect.

3.4.2. Relations de quadruplet 0-∞,k-1

L'aspect qui contient la valeur de l'objet élémentaire en relation 0-∞, k-1 avec l'objet complexe est inversé, ce qui fournit la possibilité d'accès de l'objet élémentaire vers l'objet complexe. Dans ce cas, une valeur d'objet élémentaire doit être contenue dans un seul aspect.

4. Remarques concernant le métaschéma

On pourrait, à ce stade, compléter le métaschéma en ajoutant les caractéristiques d'implémentation, mais cela se fera plus naturellement dans la troisième partie.
On peut, toutefois, faire, dès à présent, quelques remarques :

- La manière dont nous avons représenté en SESAM les types de données du modèle d'accès impose des restrictions sur le nombre d'objets complexes, de relations, sur la longueur des valeurs d'objets élémentaires d'un

schéma, etc... Ces restrictions peuvent se traduire par des modifications des i-j,k-l du métaschéma. On aura, par exemple :

(RAC,OC) ≡ 0-999, 1-1
(OC,OE) ≡ 0-320, 0-1
ATT(OE,OE) ≡ 0-1280, 0-1

En fait, au niveau de l'implémentation, la valeur ∞ devrait disparaître de tous les i-j,k-l du métaschéma.
Dans certains cas, toutefois, les nouvelles valeurs de j et de l sont trop compliquées à déterminer et nous conserverons la valeur ∞.

- Comme les relations inverses sont toujours présentes dans la base de données SESAM, même si elles n'existent pas pour l'utilisateur, nous les représenterons toujours au niveau du métaschéma. Ce fait pourrait s'exprimer en écrivant :

INVERSE(REL,REL) ≡ 1-1,1-1

Mais un tel quadruplet n'est pas autorisé par le modèle d'accès et devra donc être remplacé par une contrainte d'intégrité.

- L'implémentation des relations d'objet complexe vers objet élémentaire de quadruplet 1-1,0-1, par inclusion de la valeur de l'objet élémentaire dans l'aspect d'ordre, implique qu'il n'existe pas plus d'une telle relation, issue d'un objet complexe déterminé. Ce fait peut s'exprimer en écrivant :

ID(OE,OC) ≡ 0-1,0-1

- L'implémentation des relations de la racine vers un objet complexe par structuration de l'aspect d'ordre, assure que tout objet complexe est cible d'une telle relation.
En conséquence, les relations (RAC,OC) et ACCES (RAC,OC) ont mêmes occurrences et la seconde de ces relations peut être supprimée du métaschéma.

TROISIÈME PARTIE : LE LANGAGE DE MANIPULATION DE DONNÉES ET SON COMPILATEUR

INTRODUCTION

Nous allons maintenant décrire le langage de manipulation de données (DML) qui a été défini par l'équipe Grands Fichiers.
Ensuite, nous étudierons le compilateur de ce langage.
Plus précisement, nous examinerons quelles sont les relations d'accès et les caractéristiques d'implémentation qui lui sont nécessaires, c'est-à-dire que nous définirons le métaschéma complet. Ce compilateur a été écrit, à l'aide du DML, en modifiant celui qui a été réalisé par l'équipe Grands Fichiers. Toutefois, son étude complète serait trop longue à décrire et nous nous bornerons à esquisser son fonctionnement et à indiquer quelques remarques concernant les difficultés rencontrées lors de cette modification.

CHAPITRE I : DEFINITION DU DML D'APRES EXEMPLES

Pour définir rigoureusement le DML, il faudrait fixer sa syntaxe et définir sa sémantique à partir des primitives décrites dans le chapitre I de la première partie.
Toutefois, ce travail ayant été réalisé dans la référence [1], nous ne l'envisagerons pas ici. Nous nous limiterons, dans cette étude, à une brève description de ce langage et de son utilisation afin de faciliter la compréhension des chapitres suivants.

Le DML comprend un certain nombre d'ordres qui peuvent être groupés en trois catégories :

Des ordres associés à la primitive d'accès

REACH ... IN ... VIA ... IF
OPEN

Des ordres associés aux primitives de mise à jour

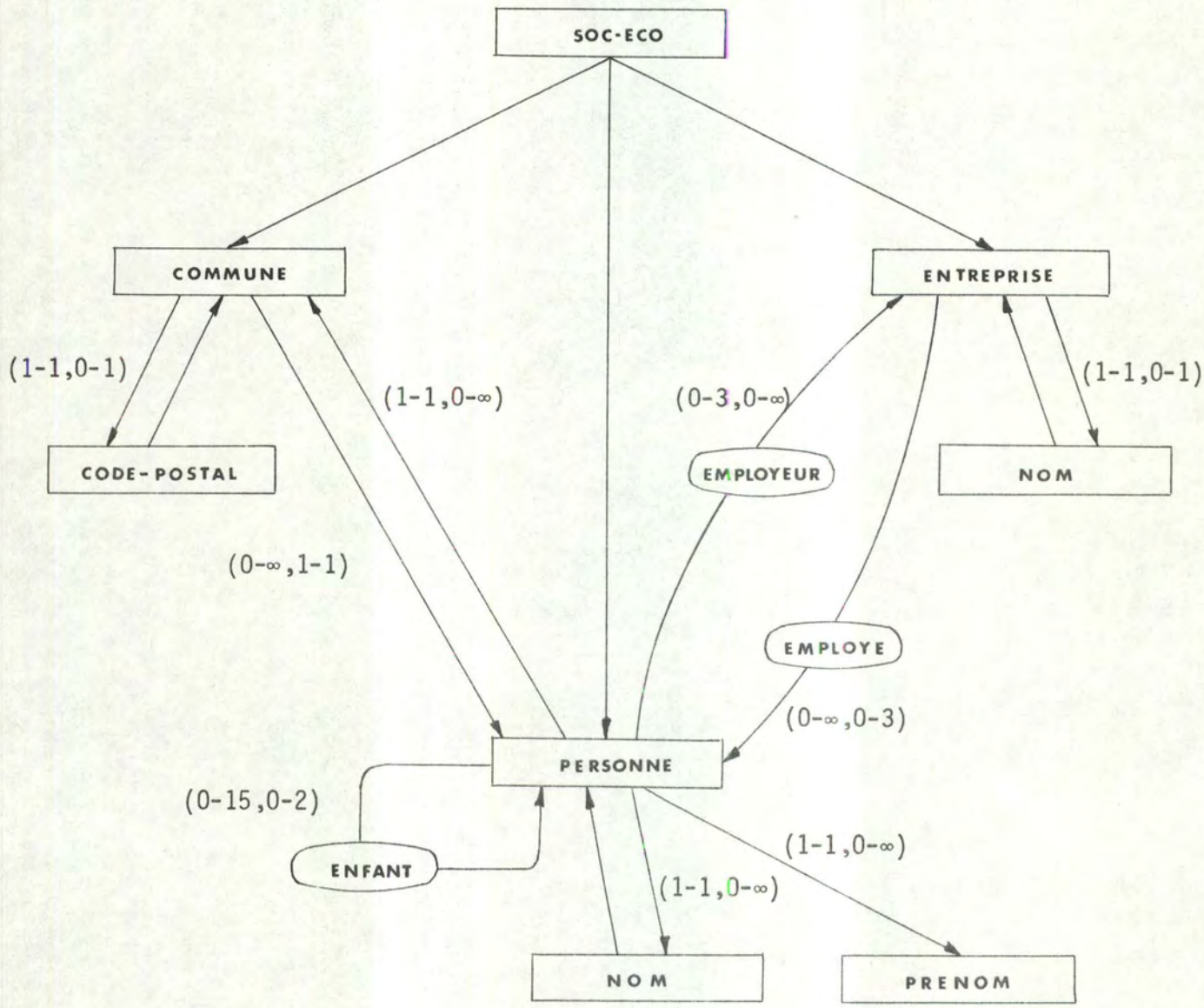
PREPARE
CREATE
SUPPRESS
} associés aux objets complexes

ATTACH
DETACH
TRANSFER
MODIFY
} associés aux relations

Des mécanismes de programmation

ALLOCATE, CLOSE, COUNT
CLEAR, MOVE, SAVE, TEST : manipulation de variables de travail
NEXT, EXIT, END : mécanismes associés aux autres ordres

Nous allons passer en revue des exemples d'emploi de ces ordres, en précisant, pour chacun d'eux, l'effet de son exécution sur une base de données.
Le graphe de cette base est dessiné ci-dessous, les quadruplets i-j,k-l des relations sont associés à leur flèche de définition.



1. Exemple de création d'une réalisation d'objet complexe

```
OPEN SOC-ECO = BD.
|  REACH COMMUNE = C1 IF (CODE-POSTAL=5000).
|  |  PREPARE PERSONNE = P1 WITH ALL.
|  |  |     MOVE "DUPONT" TO NOM OF P1.
|  |  |     MOVE "JEAN" TO PRENOM OF P1.
|  |  |     MOVE 1 TO COUNT NOM FROM P1, COUNT PRENOM FROM P1.
|  |  CREATE P1,*:C1.
|  |  |  REACH ENTREPRISE = E1 IF (NOM="FNDP").
|  |  |  |  ATTACH P1 TO E1 BY EMPLOYEUR.
|  |  |  END E1.
|  |  END P1.
|  END C1.
CLOSE BD.
```

- L'ordre "OPEN SOC-ECO=BD."
  désigne un schéma de base de données. Il effectue l'accès à la réalisation de l'objet racine, qui sera désignée dans la suite du programme par "BD", jusqu'à ce que l'ordre "CLOSE BD." clôture la zone du programme où le schéma est connu. A l'intérieur de cette zone, il sera possible de désigner les objets et les relations de ce schéma.

- L'ordre "REACH COMMUNE=C1 IF (CODE-POSTAL=5000)."
  réalise, à partir de la valeur "5000" de l'objet élémentaire CODE-POSTAL, l'accès à toutes les réalisations de l'objet complexe COMMUNE qui lui sont associées par des occurrences de la relation (CODE-POSTAL,COMMUNE). Ensuite, les instructions comprises entre cet ordre et "END C1." sont exécutées une fois pour chaque réalisation désignée, à ce moment, par "C1". Ces deux ordres déterminent donc un bloc de programme où ces réalisations sont connues et désignables.

- L'ordre "PREPARE PERSONNE=P1 WITH ALL."
  signale au système que l'on va créer une réalisation de l'objet complexe PERSONNE en la reliant par des occurrences de relation à des valeurs de tous les objets élémentaires qui lui sont associés("WITH ALL").

"NOM OF P1", "PRENOM OF P1" sont les formes COBOL des listes de désignation (P1,NOM) et (P1,PRENOM).
Les instructions "MOVE" du langage hôte placent dans une zone réservée par le "PREPARE", les valeurs des objets élémentaires qui seront associées à la réalisation.
L'instruction "MOVE 1 TO COUNT NOM FROM P1, COUNT PRENOM FROM P1." signale au système que ces valeurs ont été placées dans la zone.

- L'ordre "CREATE P1,*:C1."
  crée la nouvelle réalisation de l'objet complexe, en la reliant par des occurrences de relations, aux valeurs "DUPONT" et "JEAN" des objets élémentaires NOM et PRENOM, et à la réalisation "C1" de l'objet complexe COMMUNE.
  La nouvelle réalisation restera connue du programme jusqu'à l'exécution de l'ordre "END P1." et sera désignée par "P1".

- L'ordre "ATTACH P1 TO E1 BY EMPLOYEUR."
  crée des occurrences de la relation EMPLOYEUR reliant la réalisation "P1" de l'objet complexe PERSONNE à toutes les réalisations "E1" de l'objet complexe ENTREPRISE déterminées par l'ordre "REACH ENTREPRISE=E1 IF (NOM="FNDP").". Les occurrences correspondantes de la relation inverse EMPLOYE sont également créées.

2. Exemple de modifications et de suppression

```
OPEN SOC-ECO = BD.
    REACH PERSONNE = P1 IF (NOM="DUPONT") GET (PRENOM).
        IF PRENOM OF P1 NOT = "JEAN" THEN NEXT P1.
        REACH ENTREPRISE=E1 FROM BD IF (NOM="FNDP").
            DETACH E1 FROM P1 BY EMPLOYE.
        END E1.
        REACH COMMUNE=C1 IF (CODE-POSTAL=5000).
            TRANSFER P1 TO C1.
        END C1.
        REACH PERSONNE=P2 FROM P1 VIA ENFANT GET (PRENOM).
            PERFORM PRINT-PRENOM.
        END P2.
        MOVE "DURAND" TO NOM OF P1.
```

```
    MODIFY P1.
    END P1.
    REACH COMMUNE=C2 IF (CODE-POSTAL=5100).
        SUPPRESS C2 EXIT.
    END C2.
CLOSE BD.
```

- L'ordre "REACH PERSONNE=P1 IF (NOM="DUPONT") GET (PRENOM)."
  réalise, à partir de la valeur "DUPONT" de l'objet élémentaire NOM, l'accès à toutes les réalisations de l'objet complexe PERSONNE qui lui sont associées par des occurrences de la relation (NOM,PERSONNE). A partir de chacune de ces réalisations de l'objet complexe PERSONNE, on réalise l'accès à la réalisation de l'objet élémentaire PRENOM associée par une occurrence de la relation (PERSONNE,PRENOM) à la réalisation courante de l'objet complexe PERSONNE.

- L'ordre "IF PRENOM OF P1 NOT = "JEAN" THEN NEXT P1."
  Le test de la valeur de l'objet élémentaire PRENOM est réalisé par une instruction du langage hôte. Si la condition est vérifiée, l'ordre "NEXT P1." permet d'accéder à la réalisation suivante de l'objet complexe PERSONNE, à partir de la valeur "DUPONT" de l'objet élémentaire NOM. Si la condition n'est pas vérifiée, le programme se poursuit en séquence.

- L'ordre "DETACH E1 FROM P1 BY EMPLOYE."
  Au moment où cet ordre survient, "E1" désigne une réalisation particulière de l'objet complexe ENTREPRISE.
  L'ordre "DETACH", appliqué à la relation EMPLOYE, supprime une occurrence de la relation et de son inverse sans affecter l'existence des réalisations courantes de PERSONNE et d'ENTREPRISE.
  Ceci est dû au quadruplet 0-∞,0-3 de la relation EMPLOYE.
  On ne peut appliquer cet ordre à la relation (PERSONNE,COMMUNE), car la suppression d'une occurrence de cette relation correspond à une suppression de la réalisation de PERSONNE associée.
  Il est toutefois possible de réaliser en un seul ordre, indécomposable, l'équivalent d'un "DETACH" suivi d'un "ATTACH" grâce à l'ordre "TRANSFER".

- L'ordre "TRANSFER P1 TO E1."
  détache la réalisation "P1" (de l'objet complexe PERSONNE) d'une réalisation de l'objet complexe COMMUNE et supprime donc une occurrence de la relation (PERSONNE,COMMUNE).
  A ce stade, l'état de la base serait incohérent sans le caractère indécomposable de l'ordre "TRANSFER".
  Cet ordre attache maintenant la réalisation P1 (de l'objet complexe PERSONNE) à la réalisation "C1" de l'objet complexe COMMUNE en créant une occurrence de la relation (PERSONNE,COMMUNE).

- L'ordre "REACH PERSONNE=P2 FROM P1 VIA ENFANT GET (PRENOM)."
  A partir de la réalisation "P1" de PERSONNE, on accède successivement à toutes les réalisations de PERSONNE liées à la première par des occurrences de la relation ENFANT. Pour chacune de ces réalisations, on accède à la réalisation de l'objet élémentaire PRENOM. L'instruction du langage hôte "PERFORM PRINT-PRENOM." imprime la valeur de la réalisation courante de l'objet élémentaire PRENOM.

- L'ordre "MODIFY P1."
  modifie l'occurrence de la relation (PERSONNE,NOM), issue de "P1". La nouvelle occurrence aura pour cible la valeur "DURAND" de l'objet élémentaire NOM. L'occurrence inverse est également modifiée.

- L'ordre "SUPPRESS C2 EXIT."
  Cet ordre supprime de la base de données, la réalisation "C2" de COMMUNE mais aussi toutes les réalisations de PERSONNE auxquelles elle donnait accès, sans quoi, le contenu de la base deviendrait incohérent.
  Toutefois, après l'exécution de l'ordre "SUPPRESS", l'exécution éventuelle d'ordres relatifs à "C2" est vide de sens. Le paramètre "EXIT" commande donc la sortie du bloc "COMMUNE=C2" et l'instruction suivant ce bloc, est exécutée ("CLOSE BD".).

3. Exemple d'emploi des variables de travail

Une variable de travail permet de "sauver" la référence à une réalisation d'un objet complexe rendue disponible au moyen d'un ordre "REACH".

Dans l'exemple ci-dessous, la variable $10 est une référence à une réalisation de COMMUNE. Supposons qu'il n'existe pas de relation de COMMUNE vers PERSONNE; recherchons une réalisation de PERSONNE reliée à la réalisation de COMMUNE, référencée par $10, et sauvons la dans la variable $02.

```
OPEN  SOC-ECO = BD.
|  MOVE $10 TO $01.
|     ¦       ¦
|     ¦       ¦
|  (destruction éventuelle de $10 par d'autres ordres)
|  CLEAR $02.
|  REACH PERSONNE=P1.
|  |    REACH COMMUNE=COM1 FROM P1.
|  |    |    TEST COM1=$01.
|  |    |    IF TEST-REG=0 NEXT P1.
|  |    |    SAVE P1 IN $02.
|  |    |    EXIT P1.
|  |    END COM1.
|  END P1.
|  TEST $02.
|  IF TEST-REG=0, "insuccès".
|  (exploitation de $02)
|     ¦       ¦       ¦
|     ¦       ¦       ¦
CLOSE BD.
```

L'ordre "MOVE" range le contenu d'une variable de travail dans une autre.
L'ordre "TEST COM1=$01." compare la réalisation "COM1" de COMMUNE à la réalisation de COMMUNE sauvée dans la variable $01 et positionne le registre "TEST-REG" à 1, en cas d'égalité.
L'ordre "SAVE P1 IN $02" sauve la réalisation courante de PERSONNE dans $02.
L'ordre "TEST $02" vérifie que $02 se réfère à une réalisation et dans ce cas, positionne "TEST-REG" à 1.

CHAPITRE II : LE COMPILATEUR DU DML

1. Rôle du compilateur du DML

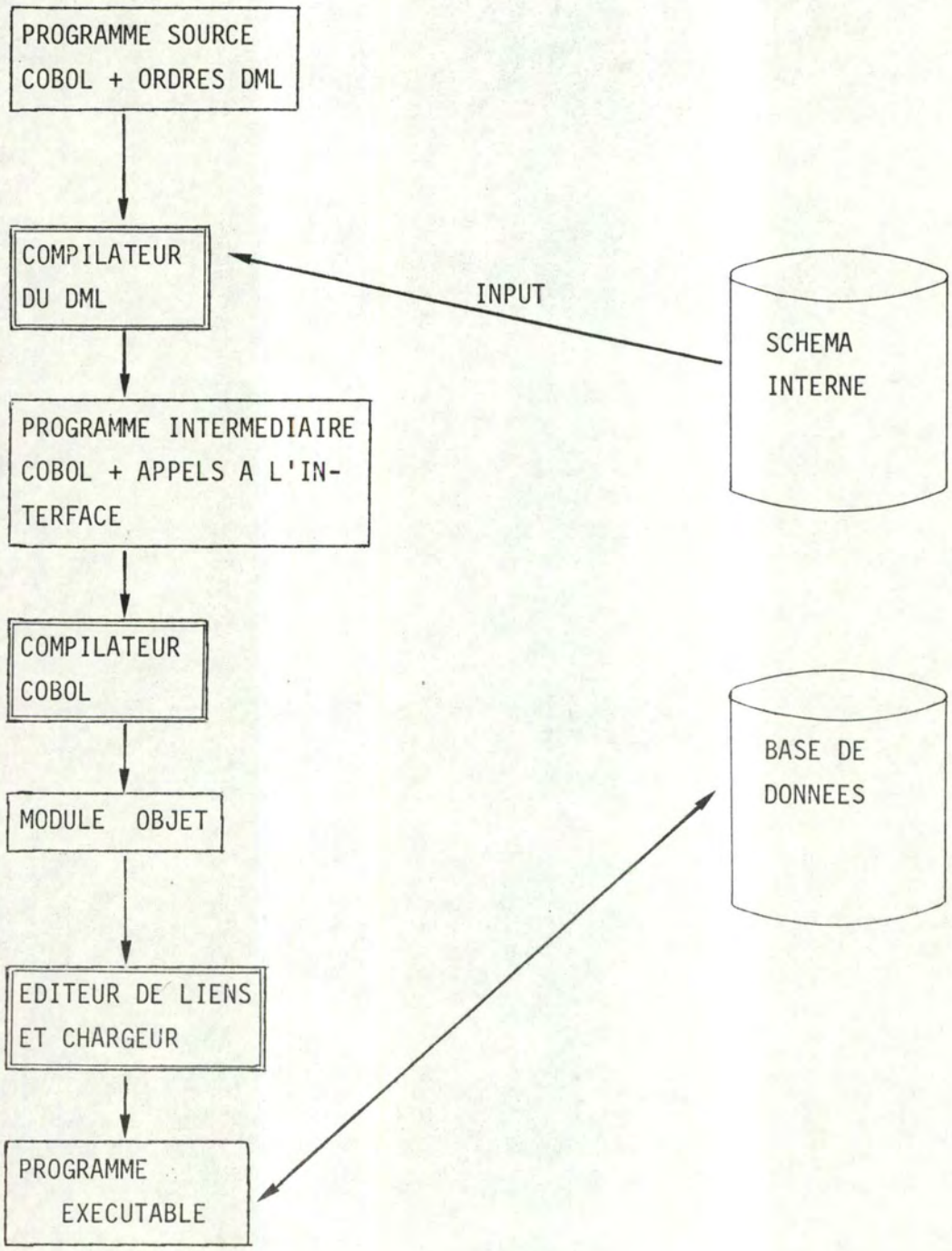
Le DML que nous venons de décrire introduit dans les programmes, une structure de blocsavec itération automatique et évite, ainsi, la programmation explicite des boucles.
Cependant, les différents traitements qui peuvent être réalisés sur les valeurs acquises, dans la base de données, par ces programmes, ne sont pas pris en charge par le DML et nécessitent l'emploi d'un langage hôte: COBOL.
L'input du compilateur est donc constitué d'un programme COBOL dans lequel on insère des ordres d'accès.
Après avoir retrouvé les différents ordres DML dans l'input décrit ci-dessus, le compilateur est chargé de remplacer ces ordres par un texte COBOL comprenant notamment des appels à un module d'interface entre le système SESAM et le programme généré. Ce dernier pourra, alors, être transformé, par un compilateur COBOL, en un programme objet standard.

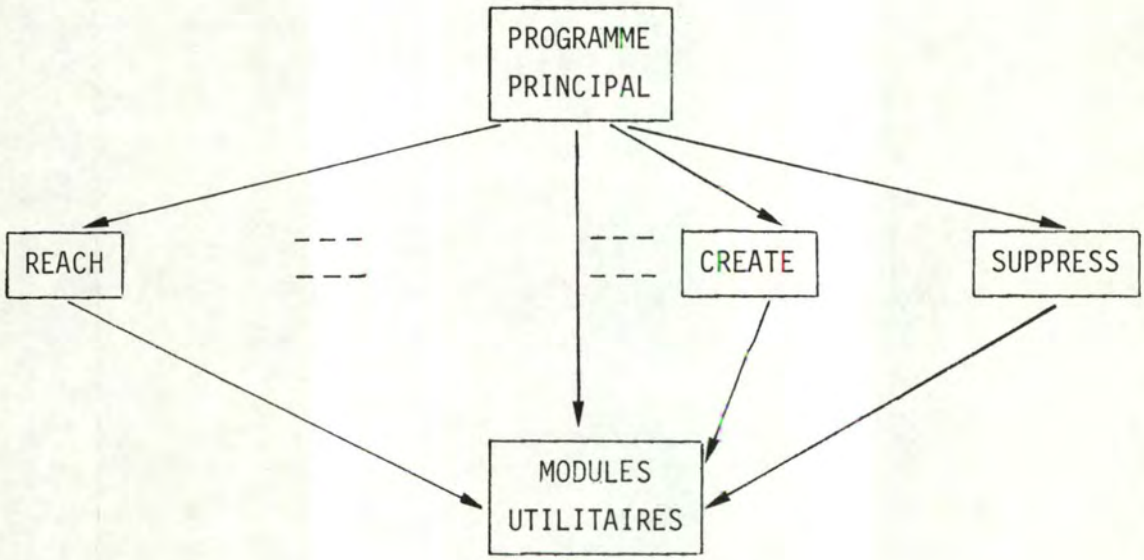
Le rôle du compilateur du DML dans la constitution d'un programme travaillant sur une base de données, peut être observé sur la figure de la page 53.

2. Structure du compilateur et exemple de génération

Ce compilateur, réalisé de façon modulaire, est dirigé par un module principal qui, après avoir initialisé les zones nécessaires à la compilation, parcourt le programme source, appelle les différents modules réalisant la compilation de chaque ordre, traite les erreurs et clôture la génération.

La figure de la page 54 permet de visualiser la structure du compilateur.


PROGRAMME SOURCE
COBOL + ORDRES DML
COMPILATEUR
DU DML
INPUT
SCHEMA
INTERNE
PROGRAMME INTERMEDIAIRE
COBOL + APPELS A L'IN-
TERFACE
COMPILATEUR
COBOL
BASE DE
DONNEES
MODULE OBJET
EDITEUR DE LIENS
ET CHARGEUR
PROGRAMME
EXECUTABLE



Précisons davantage le résultat de la génération en étudiant l'exemple suivant :

```
      .....
  REACH PERSONNE=P1 GET (NOM,PRENOM).
      ....
  END P1.
```

La présence de cet ordre indique que l'on désire accéder à des réalisations de l'objet complexe PERSONNE, accompagnées des valeurs des objets élémentaires NOM et PRENOM qui leurs sont associées.

Puisque nous travaillons dans un programme COBOL, ces valeurs, pour pouvoir être manipulées, devront être stockées dans des zones définies en "WORKING-STORAGE SECTION", à côté des zones définies par l'utilisateur.

Etant donnée la structure de blocs du DML, ces zones peuvent être agencées de manière à former une structure de stack, ce qui pourra être réalisé en COBOL au moyen d'instructions "REDEFINES". Les définitions de zones relatives à des blocs disjoints pourront, ainsi, se recouvrir.

La taille maximum du stack sera déterminée à la fin de la compilation et le compilateur le définira en générant, par exemple, dans le programme COBOL:

```
01 SYS-STACK PIC  X(1256).
```

La zone du stack, associée à P1, sera définie par :

```
01  SYS-0079-STACK REDEFINES SYS-STACK.
    02  FILLER PIC X(96).                          (1)
    02  P1.                                        (2)
    03  SYS-C PIC 9(8) COMP OCCURS 9.              (3)
    03  NOM PIC X(15).                             (4)
    03  PRENOM PIC X(10).                          (5)
```

La ligne (1) représente la place utilisée par les blocs extérieurs.
La ligne (2) permet de désigner sans ambiguïté les zones (des lignes) suivantes en les qualifiant par le nom "P1" associé à l'ordre "REACH".
La ligne (3) définit les zones qui contiendront les compteurs d'occurrences de relation, associés à la réalisation "P1".
Les lignes (4) et (5) définissent les zones qui contiendront les valeurs des objets élémentaires NOM et PRENOM.

Par ailleurs, il y a lieu de générer en "PROCEDURE DIVISION", à l'emplacement où se trouvait l'ordre "REACH", une série d'instructions réalisant l'opération demandée et faisant appel à l'interface de communication avec SESAM. Cet appel doit être paramétré en fonction de chaque situation et nécessite donc la génération, en "WORKING-STORAGE SECTION", d'une zone qui contient les informations nécessaires au traitement réalisé par l'interface : type d'opération, niveau d'imbrication, numéro de l'objet complexe, aspects SESAM contenant les valeurs, déplacement de la zone "P1" dans le stack, etc...

Dans le cas présent, les définitions de zones et les instructions suivantes seront générées.

```
WORKING-STORAGE SECTION.

      -------

01  SYS-0079-CA.
    02  FILLER  PIC  X(20)  VALUE "030030079A1002003000".
    02  FILLER  PIC  9(4)   COMP VALUE 96.
    02  FILLER  PIC  9(4)   COMP VALUE 61.
    02  FILLER  PIC  9(4)   COMP VALUE 9.
    02  FILLER  PIC  X(9)   VALUE "B36B15B10".

01  SYS-0079-CB PIC  X(11)  VALUE "030030079A5".
01  SYS-0079-C1 PIC  X(11)  VALUE "030030079A9".

      --------

PROCEDURE DIVISION.

      --------

      CALL SYSDML USING SYS-0079-CA.
      GO TO SYS-0079-BODY.
SYS-0079-NEXT.
      CALL SYSDML USING SYS-0079-CB.
SYS-0079-BODY.
      IF SYS-ERROR = "10" GO TO SYS-0079-UNSTK.
      IF SYS-ERROR > "10" GO TO SYS-DUMP.

      --------

      GO TO SYS-0079-NEXT.
SYS-0079-UNSTK.
      CALL SYSDML USING SYS-0079-C1.
      IF SYS-ERROR > "10" GO TO SYS-DUMP.
```

Les quatres dernières lignes ci-dessus, remplacent l'ordre "END P1." dans le programme généré et réalisent une boucle permettant d'accéder à l'ensemble des réalisations de l'objet complexe PERSONNE.
"SYS-ERROR" est une variable indiquant le résultat des opérations effectuées.

"SYS-DUMP" est le nom d'un paragraphe, exécuté en cas d'erreur grave, rédigé par l'utilisateur et placé en fin de programme.

Remarque

Le compilateur du DML possède, lui aussi, un stack permettant de mémoriser divers renseignements sur les ordres DML et leur structure de blocs, pendant la compilation. Il ne doit pas être confondu avec le stack de l'utilisateur que nous venons de décrire.

CHAPITRE III : DEFINITION DU METASCHEMA POUR LE COMPILATEUR DU DML

Nous allons définir les relations d'accès indispensables au compilateur du DML et de nouveaux objets reflétant l'implémentation du modèle d'accès en SESAM.
Pour ce faire, nous nous proposons de passer en revue les différents ordres du DML décrits précédemment en nous demandant quelles sont, pour chacun d'eux, les informations dont le compilateur a besoin pour en réaliser l'analyse et la génération.
Pour simplifier l'exposé et puisque nous n'avons pas défini formellement le DML, nous réaliserons cette étude à partir d'exemples.

1. Compilation de l'ordre "OPEN"

Soit l'ordre

OPEN SOC-ECO = BD.

1.1. *Vérification*

L'existence de l'objet racine SOC-ECO sera vérifiée en accédant, sur base de la valeur "SOC-ECO", aux réalisations de RAC, par la relation (NOM,RAC).

1.2. *Génération des paramètres pour l'interface*

Le compilateur génèrera, dans le programme utilisateur, une zone, contenant le nom "SOC-ECO" de la base de données SESAM correspondant à l'objet racine, qui permettra à l'interface d'ouvrir cette base de données.

Ce renseignement n'est, toutefois, pas suffisant. En effet, chaque ordre SESAM doit être accompagné d'une valeur de deux caractères appelée *numéro d'utilisateur* qui définit, en quelque sorte, le "contexte" de

l'ordre. Ainsi, on peut ouvrir plusieurs fois la même base de données SESAM, pourvu que l'on spécifie des numéros d'utilisateur différents. Chacune de ces ouvertures correspond à la réservation de zones de travail, en mémoire centrale. Lorsqu'on interroge SESAM sur un critère pouvant fournir plusieurs enregistrements en réponse, SESAM renvoie, d'abord le premier enregistrement. Ensuite, si l'on réinterroge avec le même numéro d'utilisateur et un code spécial d'"appel de réponse", on obtiendra un par un, tous les enregistrements suivants. L'ouverture d'une base de données avec un numéro d'utilisateur joue le rôle de *déclaration* du numéro d'utilisateur et la fermeture correspond à une libération.

Le programme d'interface devra, donc, générer des numéros d'utilisateur, pour SESAM. Pour cela, les conventions suivantes ont été choisies.

- A tout schéma du modèle d'accès, on associe un caractère appelé *numéro de base de données*.
- Tous les ordres relatifs à la base de données SESAM, correspondant à ce schéma, auront un numéro d'utilisateur commençant par ce caractère.
- Le second caractère sera généré, par l'interface, suivant les besoins en "appels de réponse".

Pour les raisons que nous venons d'exposer, nous introduisons, dans le métaschéma, un objet NBD tel que

val(NBD) = {A,B,...,Z,1,...,6}
(RAC,NBD) ≡ 1-1,0-1.

Le numéro de base de données sera passé en argument à l'interface, en même temps que le nom. L'étiquette "BD" et le numéro de base de données seront mémorisés dans le stack du compilateur.

2. Compilation de l'ordre "REACH"

Soit l'ordre

REACH PERSONNE=P1 FROM E1 VIA EMPLOYE GET ALL.

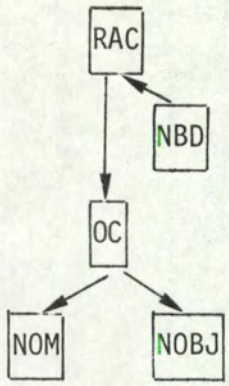
2.1. *Vérification*

L'étiquette "E1" est associée, dans le stack du compilateur, à un numéro de base de données permettant de déterminer le schéma auquel l'ordre "REACH" se rapporte.
Il faut vérifier, ensuite, que ce schéma contient bien un objet complexe PERSONNE dont le numéro devra être passé en argument à l'interface.
Pour réaliser ce traitement, on peut procéder de plusieurs manières.
Nous allons rechercher la meilleure, c'est-à-dire celle qui nécessite "le moins d'accès". Nous utiliserons comme unité de "mesure d'accès", l'accès via une occurrence de relation, en négligeant, cependant, les accès d'objet complexe vers objet élémentaire.
Soient n le nombre moyen d'objet complexe par schéma,
m le nombre moyen d'objet complexe de même nom, dans l'ensemble des schémas.
On peut supposer que m << n. Désignons, alors, par x_i le nombre d'accès nécessaires, en moyenne, par la méthode i.
Les deux premières méthodes que nous verrons supposent l'existence d'un objet élémentaire NOBJ, représentant les numéros d'objet complexe tel que,

val(NOBJ) = {1,2,...,999}.
(OC,NOBJ) ≡ 1-1,0-∞.

Méthode 1

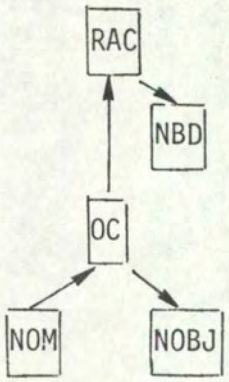
Accéder, par (NBD,RAC), à la réalisation de RAC correspondant au schéma;
accéder, par (RAC,OC) à la réalisation de OC représentant l'objet PERSONNE du schéma;
accéder au numéro d'objet.



$$x_1 = 1 + (\frac{n+1}{2}) = \frac{n+3}{2}$$

Méthode 2

Sur base du nom d'objet, accéder par (NOM,OC), aux réalisations de OC; rechercher laquelle de ces réalisations représente un objet complexe du schéma courant, en accédant, par (OC,RAC), aux réalisations de RAC; accéder au numéro d'objet.



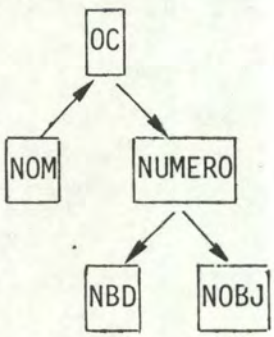
$$x_2 = (\frac{m+1}{2})+(\frac{m+1}{2}) = m+1$$

La troisième méthode suppose l'existence d'un objet élémentaire NUMERO ayant deux attributs NBD et NOBJ tels que

val(NBD) = {A,B,...,Z,1,...,6},
val(NOBJ)= {1,...,999},
(OC,NUMERO) ≡ 1-1,0-1.

Méthode 3

Sur base du nom de l'objet, accéder, par (NOM,OC), aux réalisations de OC;
vérifier que la valeur associée de NBD correspond au schéma courant;
en déduire le numéro d'objet.



$$x_3 = \frac{m+1}{2}$$

Nous choisirons cette méthode, qui minimise le nombre d'accès, c'est-à-dire que nous définirons, dans le métaschéma, les relations d'accès qu'elle utilise. En principe, il est nécessaire de réaliser une démarche semblable, chaque fois qu'un traitement nécessite d'accéder à la méta-BD, de manière à déterminer les "meilleures relations d'accès". Par la suite, chaque fois que nous fixerons de nouvelles relations d'accès, nous supposerons qu'elles ont été choisies de cette façon.

Après avoir vérifié l'existence de l'objet complexe PERSONNE, le compilateur devra, encore, vérifier celle de la relation spécifiée par l'ordre "REACH". Pour cela, il accèdera, sur la base de la valeur "EMPLOYE", aux réalisations de REL, par la relation (NOM,REL). A partir de ces réalisations, il accèdera par CIBLE(REL,OC) et ORIGINE(REL,OC), aux réalisations de OC et aux valeurs de NUMERO qui leur correspondent.
La comparaison de ces valeurs avec le numéro de l'origine, associé à "E1" dans le stack du compilateur, et le numéro de la cible, déterminé plus haut, permet de sélectionner la réalisation de REL qui convient.

2.2. *Génération du stack du programme utilisateur*

La première zone à générer, dans le stack utilisateur, est celle qui contiendra les compteurs d'occurrences de relations (cfr page 55).
La longueur de cette zone sera fournie par la valeur d'un objet élémentaire LCT, du métaschéma. On aura :

val(LCT) = {0,4,...,1280},
(OC,LCT) ≡ 1-1,0-∞.

Le compilateur accèdera ensuite, par (OC,OE), aux réalisations de OE représentant les objets élémentaires associés à PERSONNE. Il générera, pour chaque objet élémentaire spécifié par la clause "GET", une zone destinée à contenir les valeurs de cet objet élémentaire. Deux cas principaux sont à distinguer :

- Un *objet élémentaire simple* sera représenté par une zone élémentaire dont la "picture" sera générée à partir des valeurs de DESCRIPTEUR. Il nous faut, donc, préciser maintenant cet ensemble de valeurs.
  Décidons que DESCRIPTEUR possèdera trois attributs COM,TYP,LONG avec
  val(COM) = {0,1,3},
  val(TYP) = {X,9},
  val(LONG)= {1,...,1280}.
  Ainsi, par exemple, les valeurs "1" de COM, "9" de TYP et "6" de LONG déterminent la "picture" PIC 9(6) COMPUTATIONAL.
- Un *objet élémentaire composé* sera représenté par une structure COBOL dont les composants représentent les attributs de l'objet élémentaire. Les renseignements nécessaires à la génération de cette structure seront obtenus par des accès imbriqués, via la relation ATT(OE,OE). Le niveau COBOL de la zone générée sera fonction du niveau d'imbrication.

La clause "OCCURS" sera générée, dans les deux cas, à partir de la valeur de l'objet élémentaire J associée aux réalisations de OE. Enfin, pour faciliter le calcul des déplacements dans le stack utilisateur, on définira,

dans le métaschéma, un nouvel objet LOE tel que

val(LOE) = {1,...,1280},
(OE,LOE) ≡ 1-1,0-∞.

Cet objet représente le nombre de caractères qu'il faut réserver pour stocker les valeurs d'un objet élémentaire.

2.3. *Génération des paramètres pour l'interface*

La zone paramètre pour l'interface contiendra les renseignements suivants :

- le numéro de l'objet complexe PERSONNE déterminé plus haut,
- la longueur et les noms d'aspects de la zone des pointeurs et ceux de la zone des compteurs,
- les aspects des objets élémentaires spécifiés dans la clause "GET",
- l'aspect éventuel de la relation et les indices des pointeurs de la relation, dans les réalisations de l'origine.

Le compilateur déterminera ces renseignements grâce à la présence, dans le métaschéma, des objets élémentaires LPT, APT, ACT, AOE, AREL, PTREL tels que

val(LPT) = {0,4,...,1280},
(OC,LPT) ≡ 1-1,0-∞,
val(APT)=val(ACT)=val(AOE)=val(AREL)= {BAA,...,Z99},
(OC,APT) ≡ 0-5,0-∞,
(OC,ACT) ≡ 0-5,0-∞,
(OE,AOE) ≡ 0-5,0-∞,
(REL,AREL) ≡ 0-1,0-∞,
val(PTREL) = {1,...,320},
(REL,PTREL) ≡ 1-3,0-∞.

2.4. *Autre forme de l'ordre "REACH"*

Soit l'ordre

REACH PERSONNE=P1 FROM BD IF (NOM="DUPONT").

Dans ce cas, le compilateur travaille un peu différemment et détermine le schéma concerné au moyen de l'étiquette "BD", relative à un ordre "OPEN", mémorisée dans le stack du compilateur. A partir de la réalisation de OC, représentant l'objet complexe PERSONNE, il accède aux réalisations de OE, par la relation (OC,OE), jusqu'à trouver celle qui représente l'objet élémentaire NOM. Il vérifie que celui-ci peut apparaître dans la clause "IF", au moyen des relations (OE,OC) et ID(OE,OC).
Enfin, la zone paramètre qu'il génèrera pour l'interface contiendra l'aspect de l'objet élémentaire et la valeur "DUPONT".

3. Compilation des ordres "ALLOCATE" et "PREPARE"

La compilation de ces deux ordres est un sous-ensemble de celle du "REACH". Le seul traitement supplémentaire à effectuer, dans le cas du "PREPARE", est de vérifier que tous les objets élémentaires *obligatoires*, c'est-à-dire ceux qui sont accessibles de l'objet complexe par une relation de quadruplet 1-j,0-1, sont spécifiés dans la clause "GET". La relation (OE,I), du métaschéma, permet d'effectuer cette vérification.

4. Compilation des ordres "ATTACH", "DETACH", "TRANSFER"

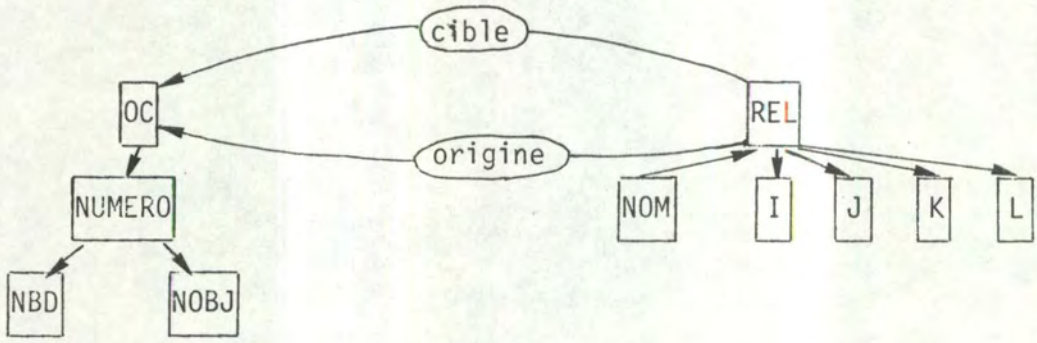
Soit l'ordre

ATTACH E1 TO P1 BY EMPLOYE.

4.1. *Vérification*

Le compilateur recherche dans le stack, les numéros d'objet associés aux étiquettes "E1" et "P1". Il vérifie, ensuite, qu'il existe bien entre

ces objets une relation de nom "EMPLOYE" telle que i < j et k < l. Les accès nécessaires à ce traitement peuvent être décrits par le graphe ci-dessous.

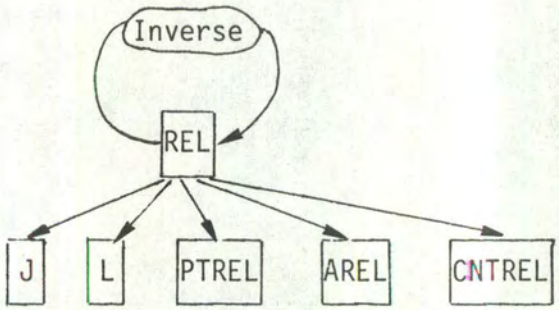


4.2. *Génération des paramètres pour l'interface*

Les noms des aspects, contenant les pointeurs et les compteurs des réalisations cible et origine , sont passés à l'interface lors de l'exécution des ordres associés aux étiquettes "E1" et "P1".
Les paramètres suivants lui seront communiqués lors de l'exécution de l'ordre "ATTACH".

- Les indices des compteurs et des pointeurs de la relation, dans l'origine et dans la cible.
- Les paramètres j et l de la relation.
- L'aspect éventuel de la relation et celui de son inverse.

La détermination de ces valeurs nécessite la présence dans le métaschéma de l'objet élémentaire CNTREL, représentant les indices des compteurs, tel que val(CNTREL) = {1,...,320},
(REL,CNTREL) ≡ 1-2,0-∞.
Les accès effectués, par le compilateur, pour obtenir les valeurs nécessaires à l'interface, correspondent au graphe ci-dessous.



Le traitement des ordres "DETACH" et "TRANSFER" est semblable.

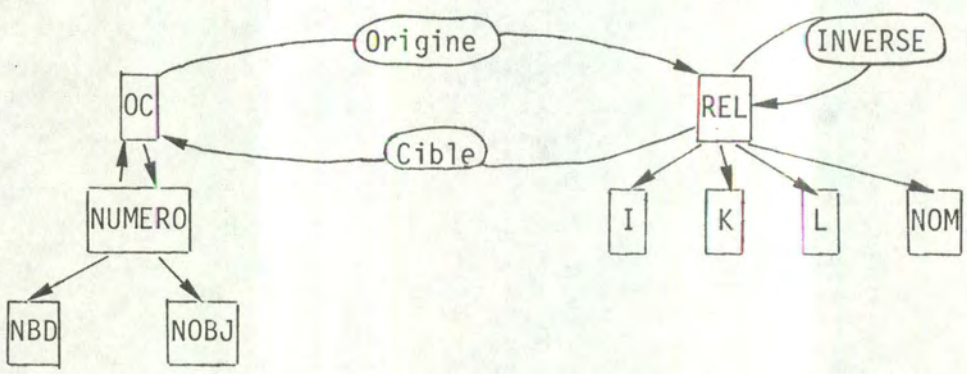
5. Compilation de l'ordre "CREATE"

Soit l'ordre

CREATE P1,EMPLOYE : E1,*:C1.

5.1. *Vérification*

L'étiquette "P1", désignant un ordre "PREPARE" dont les caractéristiques sont mémorisées au sommet du stack du compilateur, permet de déterminer le schéma et l'objet complexe, concernés par l'ordre "CREATE".
Le compilateur vérifie, alors, que toutes les relations, issues de l'objet complexe concerné et telles que k < l, sont associées par l'ordre "CREATE" à au moins i étiquettes telles que "E1", "C1".
Voici le graphe d'accès correspondant.



5.2. *Génération des paramètres pour l'interface*

Tous les noms d'aspects, nécessaires à l'interface pour créer la nouvelle réalisation, lui ont été passés lors de l'exécution des ordres associés aux étiquettes "P1", "E1", "C1".
Les renseignements suivants sont encore nécessaires.

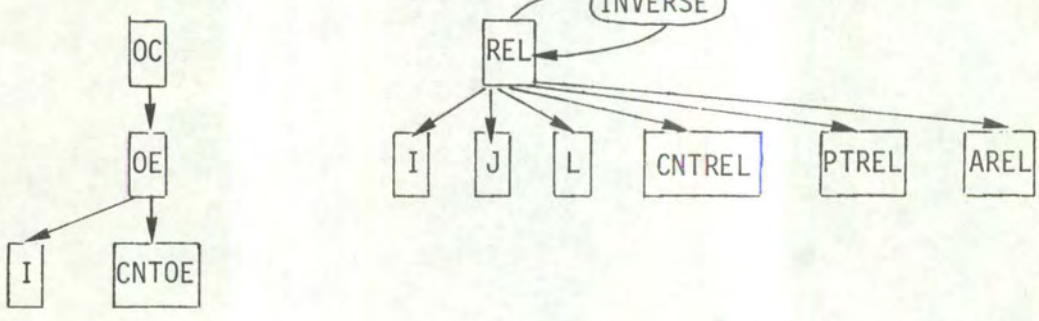
- La liste des indices des compteurs des objets élémentaires obligatoires, dans la zone des compteurs de la nouvelle réalisation. La détermination de ces valeurs implique l'existence dans le métaschéma de l'objet CNTOE tel que

  val(CNTOE) = {1,...,320},

  (OE,CNTOE) ≡ 0-1,0-∞.

- Pour chaque relation, spécifiée dans l'ordre "CREATE", les indices des pointeurs et des compteurs, l'aspect éventuel de la relation et de son inverse; les paramètres i,j,l de la relation.

Ces renseignements seront déterminés grâce aux accès décrits ci-dessous.



6. Compilation de l'ordre "SUPPRESS"

Soit l'ordre

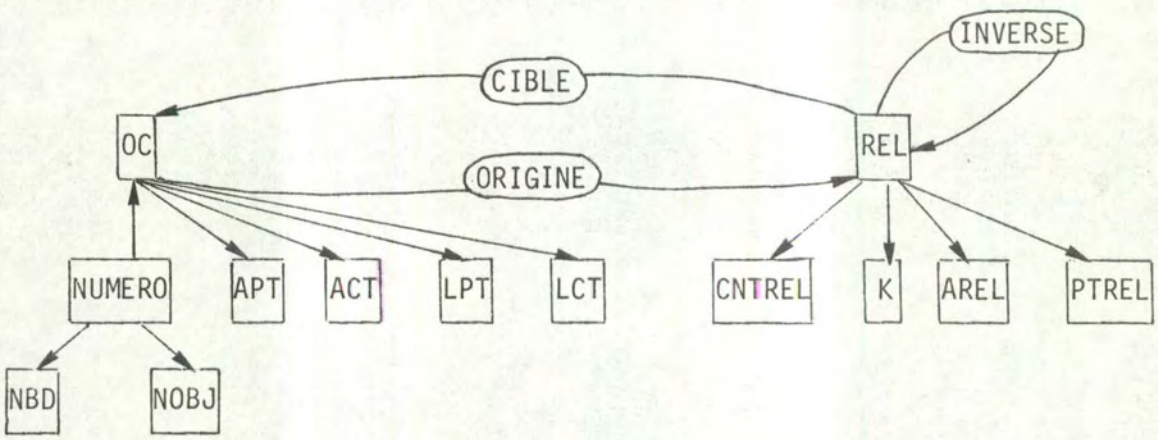
SUPPRESS C1.

La suppression d'une réalisation "C1" de l'objet complexe COMMUNE entraîne la suppression de toutes les occurrences de relation auxquelles elle par-

ticipe. En conséquence, les renseignements suivants devront être passés à l'interface, pour toutes les relations issues de cet objet.

- l'aspect de la relation et celui de son inverse;
- les indices des compteurs et des pointeurs dans l'origine et la cible;
- les aspects des zones compteurs et pointeurs, dans l'objet cible;
- la longueur de ces zones;
- le paramètre k de la relation.

Toutefois, ces suppressions d'occurrences peuvent transgresser la règle relative aux i-j,k-l. Dans l'exemple cité, toute réalisation de PERSONNE, précédemment reliée à "C1", ne vérifiera plus cette règle puisque (COMMUNE, PERSONNE) ≡ 0-∞,1-1.
C'est pourquoi l'interface devra également supprimer ces réalisations.
En conséquence, le compilateur devra générer, pour PERSONNE, les mêmes renseignements que pour COMMUNE.
Ce traitement sera effectué au moyen des relations d'accès ci-dessous.



7. Compilation de l'expression "COUNT"

Soit l'ordre

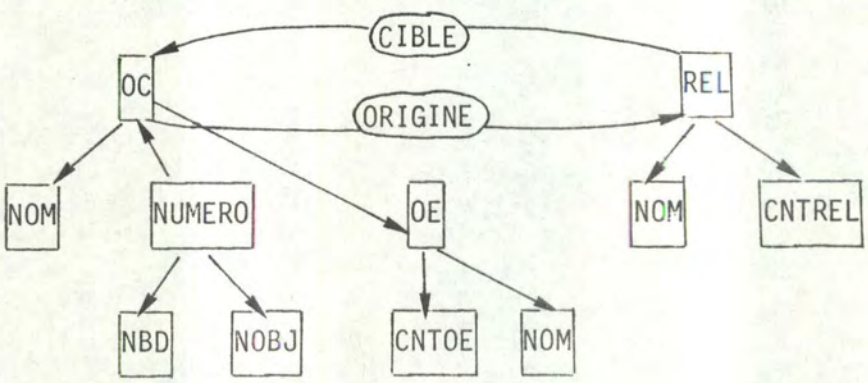
MOVE COUNT PRENOM FROM P1 TO ZONE-UTIL.

Cet ordre sera transformé, par le compilateur, en une instruction COBOL :

MOVE SYS-C OF P1 (n) TO ZONE-UTIL.

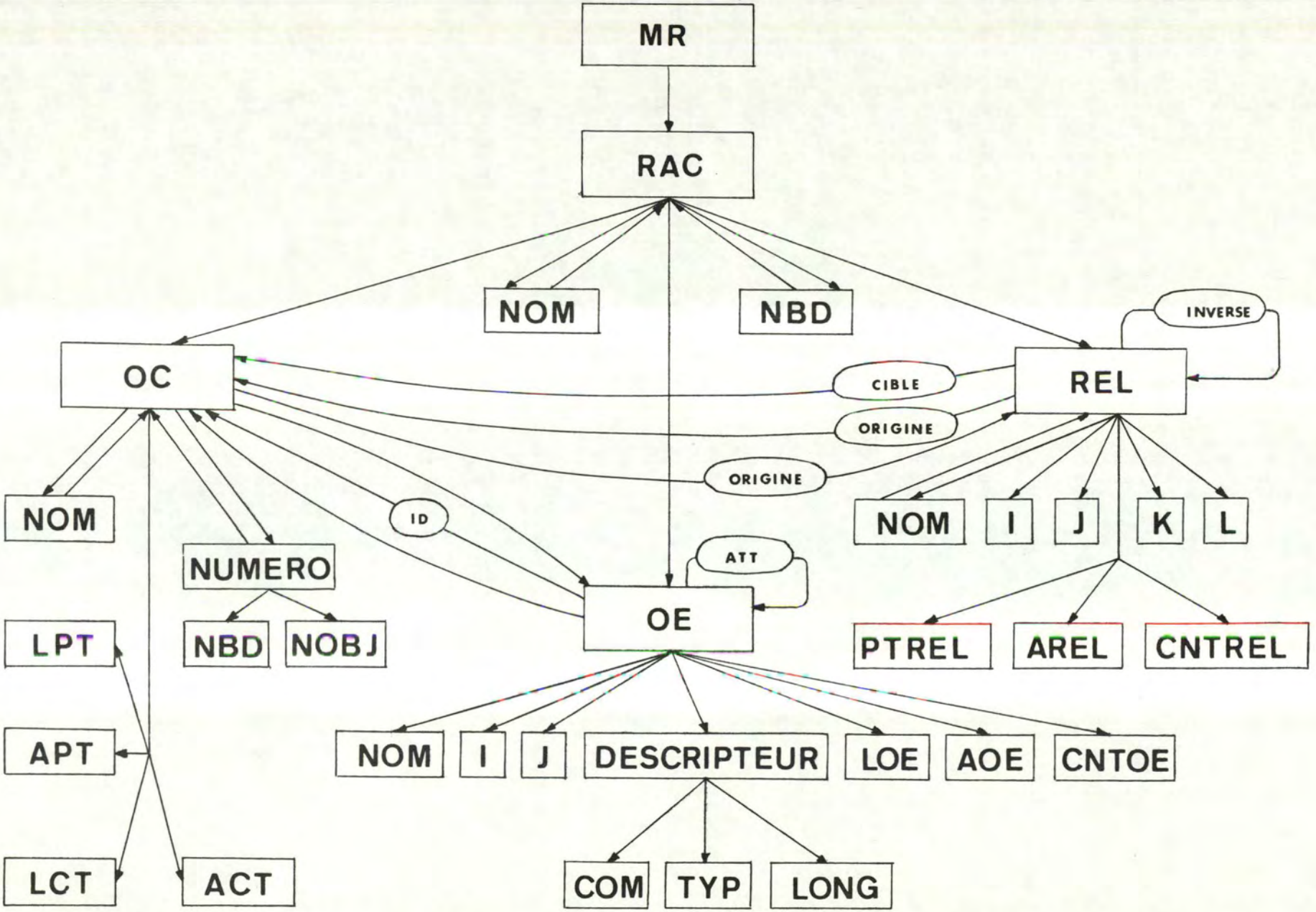
où n est l'indice du compteur d'occurrences de la relation (PERSONNE,PRENOM). Les relations d'accès nécessaires à la détermination de cet indice seront différentes selon la classe de l'objet PRENOM. (objet complexe ou élémentaire).

Elles peuvent être représentées globalement par le graphe ci-dessous.



8. Graphe d'accès du métaschéma complet

La compilation des autres ordres DML ne nécessite aucun accès au métaschéma. Celui-ci est donc constitué de tous les objets et de toutes les relations d'accès que nous venons de déterminer. Son graphe est dessiné à la page suivante.


MR
RAC
NOM
NBD
OC
INVERSE
CIBLE
REL
ORIGINE
ORIGINE
NOM
ID
NOM
I
J
K
L
NUMERO
ATT
OE
LPT
NBD
NOBJ
PTREL
AREL
CNTREL
APT
NOM
I
J
DESCRIPTEUR
LOE
AOE
CNTOE
LCT
ACT
COM
TYP
LONG

9. Contraintes d'intégrité associées au métaschéma complet

Il existe un grand nombre de contraintes d'intégrité liant entre elles les valeurs des nouveaux objets élémentaires introduits dans le métaschéma. Toutefois, dresser la liste de ces contraintes est une entreprise relativement dépourvue d'intérêt si ce n'est pour se convaincre du fait que les contraintes d'intégrité, associées à un schéma de base de données, peuvent être bien plus nombreuses et d'une nature bien plus complexe qu'on pourrait le penser, à priori.
Il est clair, cependant, que le compilateur du DDL que nous étudierons dans la quatrième partie devra être tel que ces contraintes soient vérifiées. Mais, il n'effectuera, en réalité,qu'un nombre très réduit de *vérifications*. Simplement, il *construira* ces données de telle sorte que l'on sera certain de la validité des contraintes d'intégrité sans même qu'il soit nécessaire d'avoir énoncé celles-ci. La raison en est que le compilateur du DDL est écrit directement en fonction du problème que l'on veut résoudre, à savoir, représenter les types de données du modèle d'accès dans un système cible, alors que les contraintes d'intégrité ne font qu'exprimer un certain nombre de propriétés des données en ignorant la nature de ce qu'elles représentent.

CHAPITRE IV : PROBLEMES RENCONTRES LORS DE LA MODIFICATION DU COMPILATEUR DU DML

L'écriture du compilateur du DML aurait constitué un travail beaucoup trop vaste dans le cadre de ce mémoire. C'est pourquoi nous avons simplement réalisé la modification du compilateur existant, en remplaçant tous les appels au module "SESAM" par des ordres du DML, afin qu'il puisse travailler sur la base de données correspondant au métaschéma complet (méta-BD).

Divers problèmes sont apparus lors de cette modification. Nous allons exposer ici, les deux plus importants.

1. Découpe modulaire du compilateur du DML

La structure du DML impose que tout module écrit à l'aide de ce langage contienne, au moins, un bloc "OPEN-CLOSE", à l'intérieur duquel on trouvera tous les autres ordres. Ceci entraîne que l'on exécutera un ordre "OPEN" à l'entrée et un ordre "CLOSE" à la sortie de tout module du compilateur du DML, exploitant la méta-BD.
Si l'exécution de chaque ordre "OPEN" ou "CLOSE" correspond à un traitement important sur la méta-BD, cette façon de procéder risque d'être assez lourde. Si, par contre, l'exécution effective d'un "CLOSE" est différée suffisamment longtemps pour qu'un nouvel "OPEN" ait des chances de survenir entretemps, la structure modulaire du compilateur reste acceptable.
Dans le cas où cette solution ne pourrait être réalisée, on envisagerait, par exemple, la création de deux nouveaux ordres DML : "DOPEN" et "DCLOSE" ("Dummy OPEN", "Dummy CLOSE"), qui rempliraient la même fonction syntaxique que les ordres "OPEN" et "CLOSE" mais n'auraient aucun effet à l'exécution. Le module principal du compilateur contiendrait alors un ordre "OPEN" exécuté au début et un ordre "CLOSE" exécuté à la fin de la compilation.

2. Emploi des relations ayant même objet pour cible et pour origine

Le modèle d'accès permet de définir des relations d'un objet complexe vers lui-même. De telles relations sont souvent associées à des traitements récursifs nécessitant plusieurs accès imbriqués par la relation.
Ainsi, dans le compilateur du DML, lorsque l'on génère les définitions de zones relatives à un objet élémentaire composé, il est nécessaire de réaliser un nombre, à priori indéterminé, de tels accès imbriqués.
Cette opération nécessiterait l'écriture d'un nombre infini d'ordres "REACH", ce qui est impossible.
Nous avons décidé de limiter ce nombre à cinq, ce qui implique qu'un objet élémentaire ne peut avoir que quatre niveaux de descendance :

```
REACH OE=E1 FROM C1.
|     "traitement E1".
|     REACH OE=E2 FROM E1 VIA ATT.
|     |    "traitement E2".
|     |        ¦
|     |        ¦
|     |     REACH OE=E5 FROM E4 VIA ATT.
|     |     |         "traitement E5".
|     |     |
|     |     END E5.
|     |        ¦
|     |        ¦
|     END E2.
END E1.
```

Cette façon de faire est limitative et, de plus, relativement lourde.
C'est pourquoi, on pourrait imaginer un ordre "REACH" récursif qui permettrait de réaliser, en un seul bloc, la séquence précédente, par exemple, sous la forme :

```
REACH OE=E1 FROM C1 RECUR 4 TIMES VIA ATT.
|       "traitement E1".
END E1.
```

Ce nouvel ordre permettrait, en outre, de réaliser l'équivalent d'une infinité de "REACH" imbriqués en écrivant :

```
REACH OE=E1 FROM C1 RECUR * TIMES VIA ATT.
|     "traitement E1".
END E1.
```

où "*" représente la valeur ∞.

Cependant, si un tel ordre est appliqué à une relation qui comporte des cycles d'occurrences, il risque de ne jamais terminer son exécution.
Il faudrait donc le réserver à des relations n'admettant pas de tels cycles et, malheureusement, il n'existe pas de moyen de les interdire, dans le modèle d'accès.
Voici un exemple de "REACH" récursif ne terminant pas son exécution :

```
REACH REL=R1 FROM RAC1 RECUR * TIMES VIA INVERSE.
|  ---
END R1.
```

QUATRIÈME PARTIE : LE LANGAGE DE DESCRIPTION DE DONNÉES ET SON COMPILATEUR

CHAPITRE I : LE LANGAGE DE DESCRIPTION DE DONNEES (DDL)

Dans ce chapitre, nous décrivons un langage de programmation, permettant de déclarer les types de données du modèle d'accès. Nous allons donner la syntaxe de ce langage et sa sémantique sera définie en associant à chaque clause, le concept du modèle d'accès qui y correspond. Nous appelons "clause" une valorisation quelconque d'une règle de grammaire.

La syntaxe du DDL sera définie au moyen du formalisme de Backus. Les symboles suivants ne sont pas définis par des règles de grammaire et leur signification est donnée ci-dessous.

<empty> désigne une chaîne vide de caractères,
<name> désigne un nom,
<string> désigne n'importe quelle suite de caractères imprimables ou de blancs,
<digit> désigne un chiffre,
<integer> désigne une suite finie de chiffres.

1. Définition d'un programme DDL

Syntaxe

<DDL-program>::=<identification><root-description><description>END.
<description>::=<empty>|<C-O-description><description>|<rel-description><description>

Sémantique

Un programme DDL est constitué d'une et une seule clause <DDL-program> qui définit un schéma de base de données.

2. Définition de l'identification

Syntaxe

<identification>::=IDENTIFICATION NAME IS "<user-name-and-title>".
<user-name-and-title>::=<string>

Sémantique

Cette clause, qui débute tout programme DDL, ne définit pas un concept du modèle d'accès. Elle correspond, dans l'état actuel du DDL, à une ébauche de mécanisme d'autorisation d'accès à la méta-BD et réalise l'édition de l'identification sur les listings de création.

Exemple

IDENTIFICATION NAME IS "JEAN-PIERRE LALOUX - FNDP".

3. Déclaration de l'objet racine

Syntaxe

<root-description>::=ROOT NAME IS <root-name>.
<root-name>::=<name>

Sémantique

Cette clause permet de déclarer l'objet racine de la nouvelle base de données et lui associe le nom <root-name>.

Exemple

ROOT NAME IS FNDP.

4. Déclaration d'un objet complexe et des objets élémentaires associés

Syntaxe

<C-O-description>::=<C-O-part><E-O-declaration><E-O-part>.
<C-O-part>::= COMPLEX-OBJECT NAME IS <C-O-name>;|C-O NAME IS <C-O-name>;
<C-O-name>::= <name>
<E-O-declaration>::= ELEMENTARY-OBJECT NAMES ARE ': |E-O NAMES ARE :
<E-O-part>::=<E-O-part-1>|<empty>
<E-O-part-1>::=<EO1-desc>|<EO1-desc>,<E-O-part-1>
<EO1-desc>::=<level-number-1><EO1-name><occurs-1><domain-1>
<level-number-1>::=1|01
<EO1-name>::=<name>
<occurs-1>::= OCCURS (0,<integer>)|OCCURS (1,<integer>)|OCCURS<integer>|<empty>
<domain-1>::=<picture><usage><access>|<access><descendants>
<picture>::= PIC X(<integer>)| PIC 9(<integer>) |PIC X|PIC 9
<usage>::= BINARY | PACKED | <empty>
<access>::= IDENTIFIER | ACCESS-KEY | <empty>
<descendants>::=,<attribut>|,<attribut><descendants>
<attribut>::=<level-number-2><att-name><occurs-2><domain-2>
<level-number-2>::=<digit><digit>|<digit>
<att-name>::= <name>
<occurs-2>::= OCCURS<integer>|<empty>
<domain-2>::=<picture><usage> | <descendants>

Sémantique

La clause <C-O-description> permet de déclarer un objet complexe de nom <C-O-name> ainsi que les objets élémentaires qui lui sont associés.
La clause <E-O-part> permet de déclarer tous les objets élémentaires reliés à l'objet complexe <C-O-name>, fixe les caractéristiques des relations de l'objet complexe vers les objets élémentaires et définit les attributs de ceux-ci.

La clause <EO1-desc> permet de déclarer un objet élémentaire de nom <EO1-name> relié à l'objet complexe <C-O-name>.

Les combinaisons suivantes des clauses <occurs-1> et <access> sont permises et ont la signification ci-dessous.

<occurs-1><access>≡ OCCURS (1,1) IDENTIFIER
La relation de l'objet complexe vers l'objet élémentaire est de quadruplet 1-1,O-1. La relation inverse existe.

<occurs-1><access>≡ OCCURS (i,1) ACCESS-KEY
La relation de l'objet complexe vers l'objet élémentaire est de quadruplet i-1,O-∞. La relation inverse existe (i=O ou 1).

<occurs-1><access>≡ OCCURS (i,j)
La relation de l'objet complexe vers l'objet élémentaire est de quadruplet i-j,o-∞. La relation inverse n'existe pas (i=O ou 1, j ⩾ 1).

De plus, une clause <occurs-1> vide équivaut à "OCCURS (1,1)" et "OCCURS <integer>" équivaut à "OCCURS (1,<integer>)".

La clause <picture> définit le domaine de valeurs d'un objet élémentaire simple. Quatre cas sont possibles.

<picture>≡ PIC X
Une valeur de l'objet élémentaire est formée d'un seul caractère.

<picture>≡ PIC 9
Une valeur de l'objet élémentaire est formée d'un seul chiffre.

<picture>≡ PIC X(n)
Une valeur de l'objet élémentaire est une chaine de n caractères.

<picture>≡ PIC 9(n)
Une valeur de l'objet élémentaire est une chaine de n chiffres.

5. Déclaration d'une relation entre objets complexes

Syntaxe

<rel-description>::=<single-rel>.|<single-rel><invert-rel>.
<single-rel>::= RELATION <name-part> FROM <bound-pair><origin-CO-name>
TO <bound-pair><target-CO-name>
<name-part>::= WITHOUT NAME | NAME IS <relation-name>
<bound-pair>::= (<left-bound>,<right-bound>)
<left-bound>::= <integer>
<right-bound>::= <integer>| *
<origin-CO-name>::= <name>
<target-CO-name>::= <name>
<invert-rel>::= INVERTED BY <invert-rel-name> |INVERTED
<invert-rel-name>::= <name>
<relation-name>::= <name>

Sémantique

La clause <rel-description> permet de déclarer une relation sans inverse ou un couple de relations inverses, si elle contient une clause <invert-rel>.

La clause <single-rel> permet de déclarer une relation de nom <relation-name> de l'objet complexe <origin-CO-name> vers l'objet complexe <target-CO-name>.
"FROM <bound-pair>" détermine les deux dernières valeurs et "TO <bound-pair>", les deux premières valeurs du quadruplet i-j,k-l de la relation.
"*" représente la valeur "infini".
Si <name-part>≡ WITHOUT NAME, la relation ne possède pas de nom.

La clause <invert-rel> permet de déclarer la relation inverse de la précédente, de nom <invert-rel-name> ou sans nom, si <invert-rel>≡INVERTED.

La clause <usage> s'applique aux objets élémentaires simples à valeurs numériques, elle précise la représentation interne de ces valeurs.
<usage>≡BINARY définit une représentation en "décimal codé binaire".
<usage>≡PACKED définit une représentation en "décimal condensé".
Une clause vide définit une représentation en "décimal étendu".

La clause <descendants> définit le domaine de valeurs d'un objet élémentaire composé. Elle comporte une ou plusieurs clauses <attribut> ayant toutes même numéro de niveau, égal au numéro de niveau de l'objet élémentaire composé, plus un.
Soient $OE_1,\ldots,OE_n$ les n objets élémentaires attributs déclarés dans cet ordre par les n clauses <attribut> qui composent la clause <descendants> et $j_1,\ldots,j_n$ leurs facteurs de répétition. (voir clause <attribut>.) Une valeur de l'objet élémentaire composé est la concaténation de j_1 valeurs de OE_1, j_2 valeurs de $OE_2,\ldots,$ j_n valeurs de OE_n.

La clause <attribut> définit un objet élémentaire attribut de nom <att-name> dont les valeurs interviennent j fois dans la constitution de la valeur de l'objet élémentaire composé, si une clause <occurs-2>≡ OCCURS j est présente, et une fois si elle est absente.
Cette valeur de l'objet élémentaire attribut est définie, soit par une clause <picture> , soit récursivement, par une clause <descendants>.

Exemple

```
COMPLEX-OBJECT NAME IS PERSONNE;
ELEMENTARY-OBJECT NAMES ARE :
    1  NOM PIC X(20) IDENTIFIER,
    1  PRENOM OCCURS (0,5) PIC X(10),
    1  ADRESSE,  2  RUE PIC X(15),
                 2  NUMERO PIC 9(5),
                 2  LOCALITE PIC X(15).
```

Exemples

```
RELATION NAME IS DIRECTEUR
     FROM (0,*) DEPARTEMENT
     TO   (1,1) PERSONNE
     INVERTED BY DIRECTEUR.
RELATION WITHOUT NAME
     FROM (1,1) FACULTE
     TO   (0,*) SECTION.
```

6. Programme DDL correspondant au métaschéma

Le programme DDL, ci-dessous, permet de définir le métaschéma dont le graphe est donné page 71.

```
IDENTIFICATION NAME IS "J-P LALOUX - METASCHEMA".
ROOT NAME IS MR.
COMPLEX-OBJECT NAME IS RAC;
ELEMENTARY-OBJECT NAMES ARE :
     1 NOM PIC X(15) ACCESS-KEY,
     1 NBD PIC X IDENTIFIER.
COMPLEX-OBJECT NAME IS OC;
ELEMENTARY-OBJECT NAMES ARE :
     1 NOM PIC X(15) ACCESS-KEY,
     1 NUMERO IDENTIFIER, 2 NBD PIC X, 2 NOBJ PIC 9(3),
     1 LPT PIC 9(4),
     1 APT OCCURS (0,5) PIC X(3),
     1 LCT PIC 9(4),
     1 ACT OCCURS (0,5) PIC X(3).
COMPLEX-OBJECT NAME IS REL;
ELEMENTARY-OBJECT NAMES ARE :
     1 NOM OCCURS (0,1) PIC X(15) ACCESS-KEY,
     1 I PIC 9(2) BINARY,
     1 J PIC 9(8) BINARY,
     1 K PIC 9(2) BINARY,
     1 L PIC 9(8) BINARY,
```

```
     1 CNTREL OCCURS 2 PIC 9(3),
     1 PTREL OCCURS 3 PIC 9(3),
     1 AREL OCCURS (0,1) PIC X(3).
COMPLEX-OBJECT NAME IS OE;
ELEMENTARY-OBJECT NAMES ARE :
     1 NOM PIC X(15),
     1 I OCCURS (0,1) PIC 9,
     1 J PIC 9(4),
     1 LOE PIC 9(4),
     1 AOE OCCURS (0,5) PIC X(3),
     1 CNTOE OCCURS (0,1) PIC 9(3),
     1 DESCRIPTEUR OCCURS (0,1),
           2 COM PIC X,
           2 TYP PIC X,
           2 LONG PIC 9(4).

RELATION WITHOUT NAME FROM (1,1) RAC TO (0,999) OC.
RELATION WITHOUT NAME FROM (1,1) RAC TO (0,999) REL.
RELATION WITHOUT NAME FROM (1,1) RAC TO (0,*) OE.
RELATION NAME IS ID FROM (0,1) OE TO (0,1) OC.
RELATION WITHOUT NAME FROM (0,1) OC TO (0,320) OE.
RELATION WITHOUT NAME FROM (0,320) OE TO (0,1) OC.
RELATION NAME IS ORIGINE
     FROM (1,1) OC TO (0,320) REL INVERTED BY ORIGINE.
RELATION NAME IS CIBLE
     FROM (0,320) REL TO (1,1) OC.
RELATION NAME IS ATT FROM (0,1) OE TO (0,1280) OE.
RELATION NAME IS INVERSE
     FROM (0,1) REL TO (0,1) REL INVERTED BY INVERSE.
END.
```

CHAPITRE II : LE COMPILATEUR DU DDL

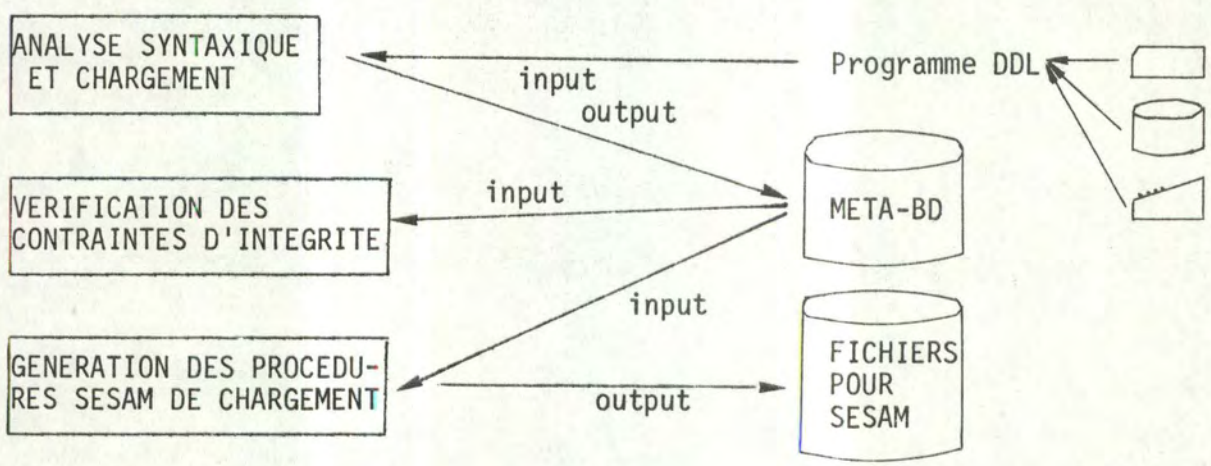
1. Rôle du compilateur

Un programme DDL représente un schéma de base de données.
Le compilateur du langage DDL doit donc

- réaliser l'analyse et la vérification syntaxique des différentes clauses du programme;
- charger la méta-BD avec toutes les informations recueillies;
- calculer les données d'implémentation ;
- vérifier les contraintes d'intégrité;
- générer certains fichiers qui seront utilisés par les utilitaires SESAM afin de créer une base de données SESAM correspondant au schéma défini par le programme DDL.

L'ensemble de ces opérations aurait pu être réalisé en un seul programme. Cependant, nous avons préféré gagner en clarté en adoptant une structure plus modulaire.

Nous distinguerons donc, dans le compilateur DDL, trois fonctions principales schématisées par la figure suivante.



- Le premier module du compilateur analyse le programme DDL et charge la méta-BD.
- Le deuxième module vérifie les contraintes d'intégrité notamment en lisant et comparant entre elles les informations de cette méta-BD.
- Le troisième module a un caractère plus utilitaire puisqu'il exploite la méta-BD afin de consulter un ensemble de fichiers qui serviront de données aux programmes SESAM de chargement.

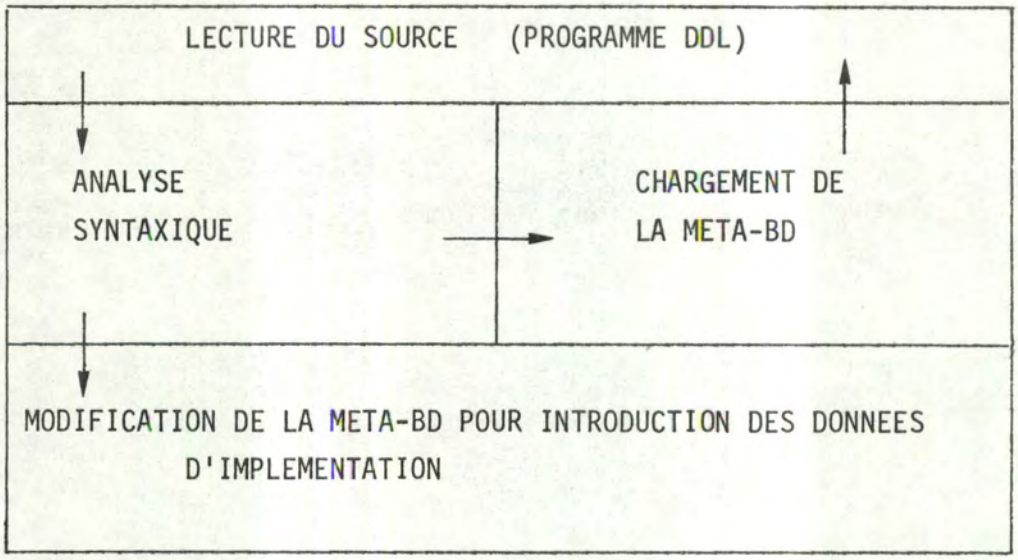
2. Analyse syntaxique et chargement de la méta-BD

L'étude de ce module nous amène à distinguer trois fonctions principales :

- l'analyse syntaxique du programme DDL,
- le chargement de la méta-BD,
- la modification de cette dernière, pour introduction des données d'implémentation.

Cette découpe en fonctions, observable ici au niveau logique, est réalisée au niveau de la programmation de ce module.
La fonction de chargement, réalisée grâce au DML, peut, de ce fait, être clairement distinguée de l'analyse syntaxique.

La réalisation de cette découpe n'est cependant pas évidente, à priori, car il faut tenir compte de la hiérarchie de blocs imbriqués créée par le DML. Lors de l'analyse d'une clause <C-O-description>, par exemple, il faut sortir du bloc "création OC" sans le clôturer afin d'exécuter la fonction séparée "analyse syntaxique des clauses <EO1-part>". Lorsque cette analyse se termine, il faut revenir dans la structure de blocs, à l'endroit exact où on l'a quittée, afin de respecter le niveau d'imbrication, ce qui impose d'apporter une attention particulière à la liaison entre ces deux fonctions.
Les fonctions du module et leurs connections peuvent être représentées par la figure ci-après.



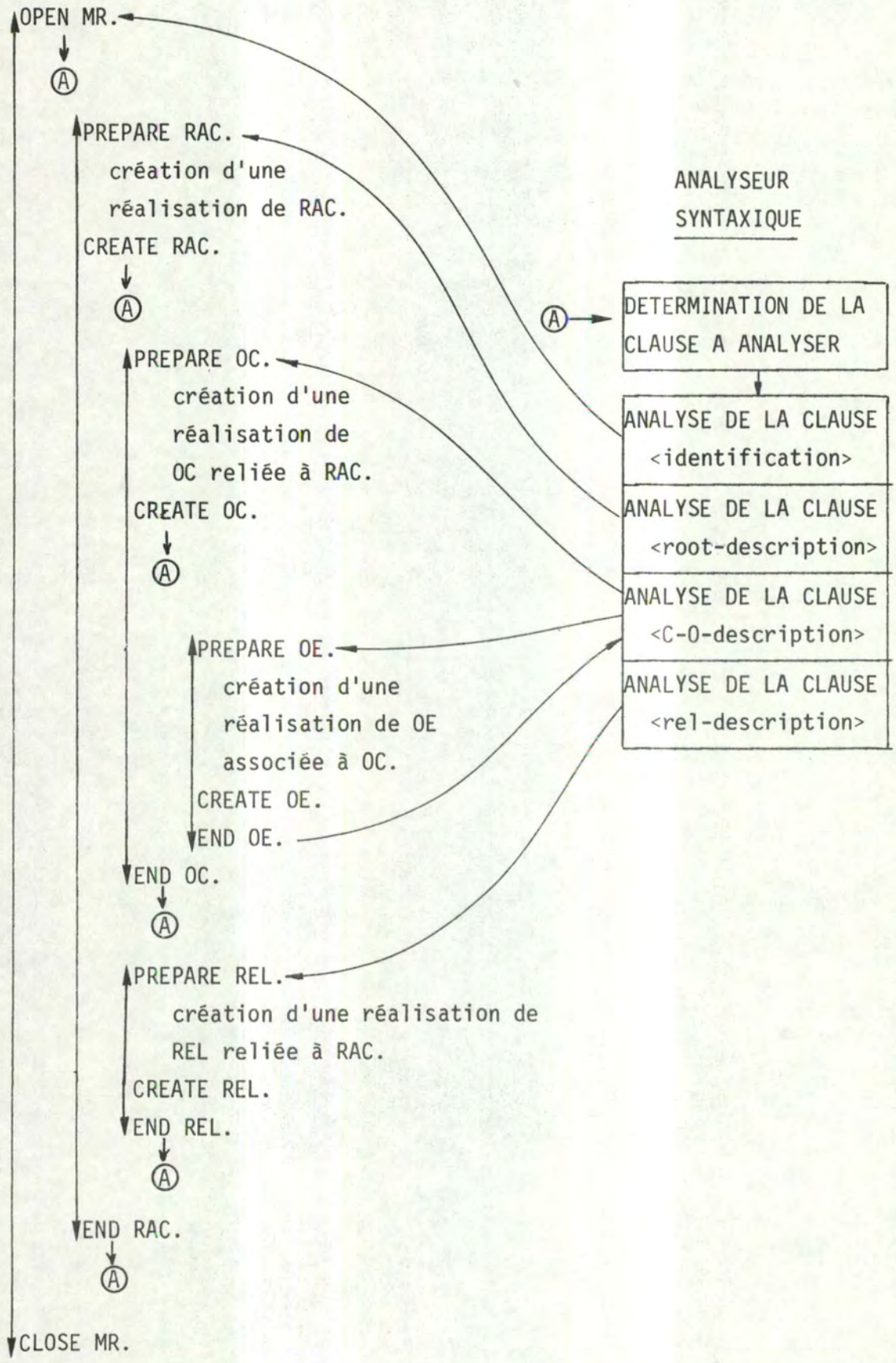
L'analyseur syntaxique, sur base de la lecture du source, donne des ordres à la fonction de chargement. Lorsque cette fonction a exécuté un ordre, elle initialise une nouvelle lecture du programme DDL.
Lorsque l'analyseur syntaxique détecte la fin du programme DDL, il passe le contrôle à la fonction de modification de la méta-BD.

2.1. *Analyse syntaxique*

L'analyseur syntaxique se décompose en quatre parties consacrées aux quatre clauses principales du DDL : <identification>, <root-description>, <C-O-description>, <rel-description>.
Chaque partie isole les diverses informations présentes dans une clause, les stocke en mémoire (dans une structure COBOL) et fournit cette structure à la fonction de chargement.

La figure suivante décrit de façon simplifiée, le fonctionnement de l'analyseur syntaxique et la structure du programme de chargement.

PROGRAMME DE CHARGEMENT



2.2. *Chargement de la méta-BD*

La figure précédente donne une première approche de la structure générale du programme de chargement. Nous allons reprendre les étapes de création en précisant pour chacune :
- ce qui est fourni par l'analyseur syntaxique,
- les contraintes à vérifier et la méthode de vérification,
- la méthode de création.

2.2.1. Traitement de la clause <identification>

L'analyseur syntaxique fournit <user-name-and-title> qui servira d'entête à chaque page du listing de création. Le programme de chargement ouvre la méta-BD.

2.2.2 Traitement de la clause <root-description>

La déclaration d'un nouvel objet racine correspond à la création d'une nouvelle réalisation de l'objet complexe RAC du métaschéma.
Avant de créer cette réalisation, il faut vérifier que la valeur de NOM fournie par l'analyse syntaxique ne correspond pas à une réalisation préexistante de l'objet RAC. De plus, il faut associer à la nouvelle réalisation, une valeur de NBD (numéro de la base de données).
Le bloc

```
REACH RAC=R1 GET (NOM).
        "test du nom et comptage des réalisations".
END R1.
```

permet de réaliser ce traitement.
S'il existe déjà une racine de même nom, un message est imprimé, la création est annulée et la compilation se termine. Sinon, la création est autorisée.

Méthode de création.

```
PREPARE RAC=R2 WITH ALL.
        "garnir NOM et NBD".
CREATE R2.
```

L'ordre "END R2." sera placé en fin de programme.
Le bloc "racine" contiendra, donc, les blocs de création des réalisations de OC et de REL qui pourront ainsi être reliées à la nouvelle réalisation de RAC.

2.2.3. Traitement de la clause <C-O-description>

2.2.3.1. clause <C-O-part>.

La déclaration d'un nouvel objet complexe correspond à la création d'une nouvelle réalisation de l'objet OC du métaschéma. Avant de créer cette nouvelle réalisation, il faut vérifier que la valeur de NOM, fournie par l'analyse syntaxique, ne désigne pas déjà un objet complexe du nouveau schéma.

Cette vérification est réalisée par :

```
REACH OC=C10 FROM MR IF (NOM=<C-O-name>) GET (NUMERO).
|     message "création refusée" si "C10" appartient
|     au schéma en cours de création.
END C10.
```

Si ce "REACH" ne fournit aucune réalisation correspondant au nouveau schéma, il y a création par la séquence ci-dessous

```
PREPARE OC=C1 WITH ALL.
|     "garnir NOM et NUMERO".
CREATE C1,*:R2.
```

Par cette opération, la nouvelle réalisation de OC est également attachée à la réalisation de RAC précédemment créée.

2.2.3.2. clause <E-O-part>.

L'analyseur syntaxique décompose cette clause en un certain nombre de clauses <EO1-desc> et <attribut>. L'analyse de chacune de ces clauses fournit, dans une structure COBOL, les différentes caractéristiques d'un objet élémentaire : numéro de niveau, nom, type (numérique ou caractère), longueur en bytes, facteur de répétition, forme de représentation, caractéristique d'accès.

La vérification du fait que tout objet élémentaire est désignable dans le schéma auquel il appartient, n'est possible qu'après la déclaration de tous les objets élémentaires associés à un objet complexe et est réalisée par une routine spéciale en fin de programme.

Méthode de création

La déclaration d'un objet élémentaire correspond à la création d'une réalisation de OE. Le numéro de niveau, associé à tout objet élémentaire dans le DDL, a une utilité particulière pour le programme de chargement.

- Si l'objet élémentaire est de niveau 1 (relié directement à un objet complexe), la nouvelle réalisation de OE doit être reliée à la réalisation de OC créée en dernier lieu.
- Si l'objet élémentaire est de niveau >1 (objet élémentaire attribut), la nouvelle réalisation de OE doit être reliée à la dernière réalisation créée de OE, de niveau immédiatement inférieur.

Cette constatation implique que, lors de la création d'une réalisation de OE, il faut, pour chaque niveau inférieur, conserver "trace en mémoire" de la dernière réalisation créée.
Plusieurs méthodes pouvaient être utilisées pour résoudre ce problème.

- une cascade de PREPARE-CREATE imbriqués :

```
PREPARE OE=E1 WITH ALL.
        ----
CREATE E1,*:C1.
    PREPARE OE=E2 WITH ALL.
            ----
    CREATE E2,ATT*:E1.
        PREPARE OE=E3 WITH ALL.
                ----
        CREATE E3,ATT*:E2.
```

```
PREPARE OE=E4 WITH ALL.
    ----
CREATE E4,ATT*:E3.
    PREPARE OE=E5 WITH ALL.
        ----
    CREATE E5,ATT*:E4.
    END E5.
END E4.
END E3.
END E2.
END E1.
```

Cette méthode, qui réalise, de la façon la plus naturelle, le traitement désiré a, en outre, l'avantage de minimiser le nombre d'accès à la méta-BD. Cependant, la place mémoire occupée apparaît non-négligeable et la structure de blocs imbriqués entraîne la complication de la liaison entre analyseur syntaxique et programme de chargement. Enfin, bien que cette raison soit mineure, il faut signaler que, l'implémentation du DML n'étant pas totalement terminée, nous nous sommes trouvés dans l'obligation de simuler tout le traitement de l'interface et le "CREATE" est un ordre dont la simulation est particulièrement fastidieuse.

- Pour cet ensemble de raisons, nous avons préféré la méthode qui consiste à "sauver" chaque réalisation de OE créée, dans une variable de travail associée à son numéro de niveau.
  Ceci permet de ne pas maintenir toute la structure en mémoire et donne un accès rapide aux réalisations de niveau immédiatement inférieur, par un "REACH sur variable de travail".

  Une réalisation de OE sera donc créée comme suit :

```
PREPARE OE=E1 WITH ALL.
     "garnir I,J,NOM,LOE,DESCRIPTEUR".
CREATE E1.

IF <level-number>  = 1     SAVE E1 IN $01.
IF <level-number>  = 2     SAVE E1 IN $02.
IF <level-number>  = 3     SAVE E1 IN $03.
IF <level-number>  = 4     SAVE E1 IN $04.
```

La réalisation, ainsi créée, doit maintenant être reliée à d'autres réalisations préexistantes.
Distinguons à nouveau deux cas.

- Si la réalisation est de niveau >1, elle doit être attachée à une réalisation de niveau inférieur de la façon suivante :

```
IF <level-number>  = 2     MOVE $01 TO $06.
IF <level-number>  = 3     MOVE $02 TO $06.
IF <level-number>  = 4     MOVE $03 TO $06.
IF <level-number>  = 5     MOVE $04 TO $06.

REACH OE $06=E2.
ATTACH E2 TO E1 VIA ATT.
END.
```

- Si la réalisation est de niveau 1, elle est attachée, par l'ordre

```
ATTACH C1 TO E1.
```

à la réalisation de OC, créée en dernier lieu.
De plus, si la déclaration d'objet élémentaire que l'on est en train de traiter, contenait une clause "IDENTIFIER" ou une clause "ACCESS-KEY", un des deux ordres suivants sera exécuté :

```
ATTACH E1 TO C1 VIA ID.
ATTACH E1 TO C1.
```

Remarque

Les séquences de programmes, que nous venons de décrire, pourraient être considérablement simplifiées par l'utilisation de variables de travail indicées et l'utilisation de ces mêmes variables dans l'ordre "ATTACH".

2.2.4. Traitement de la clause <rel-description>.

La déclaration d'une relation entre objets complexes correspondra à la création d'une réalisation de l'objet REL du métaschéma.
L'analyseur syntaxique fournit, dans une structure COBOL, les différentes caractéristiques de la relation : nom, quadruplet i-j,k-l, nom de l'origine, nom de la cible, nom de l'inverse, s'il y a lieu.
Avant de créer la nouvelle réalisation de REL, il est nécessaire d'accéder aux réalisations de OC représentant l'origine et la cible de la nouvelle relation. On en profitera pour vérifier l'unicité du nom de la nouvelle relation, relativement à ces objets. Ceci implique qu'une déclaration de relation ne peut apparaître, dans un programme DDL, qu'après la déclaration des objets complexes origine et cible.
La séquence, ci-dessous, réalise ce traitement.

```
REACH OC=C2 IF (NOM=<origin-CO-name>) GET (NUMERO).
    IF NBD OF C2 NOT = "numéro de base de données courant"
    THEN NEXT C2.
    REACH REL=R20 FROM C2 VIA ORIGINE GET (NOM).
    |   IF <rel-name> NOT = NOM OF R20
    |   THEN NEXT R20.
    |   REACH OC=C15 FROM R20 VIA CIBLE GET (NOM).
    |       IF <target-CO-name> = NOM OF C15
    |       THEN "erreur: relation redondante"
    |       ELSE NEXT R20.
    END R20.
    REACH OC=C3 FROM MR IF (NOM = <target-CO-name>) GET (NUMERO).
        IF NBD OF C3 NOT = "numéro de base de données courant"
        THEN NEXT C3.
    PREPARE REL=R6 WITH ALL.
    |   "garnir I,J,K,L,NOM".
    CREATE R6, ORIGINE:C2, CIBLE:C3,*:R2.
```

Puisque nous avons choisi de toujours représenter une relation inverse, même non déclarée, dans la méta-BD, il faut maintenant créer une réalisation de REL représentant cette inverse au moyen de la séquence :

```
REACH REL=R21 FROM C3 VIA ORIGINE GET (NOM).
|   IF <invert-rel-name> NOT = NOM OF R21
|   THEN NEXT R21.
|   REACH OC=C16 FROM R21 VIA CIBLE GET (NOM).
|   |   IF <origin-CO-name> = NOM OF C16
|   |   THEN "erreur : relation redondante"
|   |   ELSE NEXT R21.
|   END C16.
END R21.
PREPARE REL=R4 WITH ALL.
|   "garnir I,J,K,L,NOM".
CREATE R4, CIBLE:C2, ORIGINE:C3, INVERSE:R6,*:R2.
```

On remarque que les réalisations "C2" et "C3" ont été permutées. De plus, la nouvelle réalisation est reliée à "R6" par une occurrence de INVERSE (REL,REL).
Les valeurs des objets élémentaires I,J,K,L associées à "R4" sont déterminées à partir de celles associées à "R6"; seule la valeur de NOM est fournie par l'analyseur syntaxique.

Le traitement précédent n'est, toutefois, pas exécuté quand le compilateur doit traiter une déclaration de relation symétrique, puisqu'une telle relation est sa propre inverse.
Dans ce cas, la séquence, ci-dessus, sera remplacée par l'ordre suivant :

```
ATTACH R6 TO R6 VIA INVERSE.
```

2.3. *Modification de la méta-BD pour introduction des données d'implémentation*

A ce stade, on a chargé la partie de la méta-BD correspondant au sous-schéma (sémantique) défini au chapitre II de la première partie, c'est-à-dire que l'on a représenté les types de données du modèle d'accès indépendamment de leur implémentation. Le DDL décrit, en effet, le modèle d'accès, mais non sa représentation SESAM. Ce sont les données d'implémentation que nous allons créer maintenant.

2.3.1. Données d'implémentation associées aux relations entre objets complexes

Suivant le type d'implémentation, déterminé en fonction des i-j,k-l, toute réalisation de l'objet complexe origine contiendra un ou trois pointeur(s) et un ou deux compteur(s) associés à la relation. C'est à ce niveau, que les indices de ces compteurs et de ces pointeurs vont être calculés.
De plus, dans certains cas, un aspect SESAM doit être associé à une relation. La détermination du nom de cet aspect sera réalisée par un appel à un sous-programme de génération d'aspects que nous étudierons au paragraphe 4.1.

2.3.2. Données d'implémentation associées aux objets complexes

Suivant le nombre et le type des relations dont un objet complexe est origine, les réalisations de celui-ci contiendront un nombre variable de pointeurs et de compteurs.
Il faut donc maintenant, calculer la longueur des zones contenant ces pointeurs et ces compteurs et leur associer des aspects SESAM. Ces longueurs et le nom de ces aspects seront, alors, stockés dans la méta-BD.

2.3.3. Données d'implémentation associées aux objets élémentaires

Les valeurs d'un objet élémentaire de niveau 1, sont contenues dans des aspects SESAM. Pour pouvoir générer ces aspects, il faut d'abord calculer la longueur de ces valeurs, en sommant les longueurs des valeurs des objets attributs. Ces nouveaux renseignements seront eux aussi introduits dans la méta-BD.

2.3.4. Programme de modification de la méta-BD

L'ensemble de ces modifications se traduit par le programme suivant.

```
REACH OC=C4 FROM MR GET (LPT, APT, LCT, ACT).
      REACH REL=R3 FROM C4 VIA ORIGINE GET (PTREL, AREL, CNTREL).
         "calculer les indices des pointeurs et des compteurs;
          incrémenter la longueur des zones associées;
          générer l'aspect de la relation;
          garnir PTREL, AREL, CNTREL".
      MODIFY R3.
      END R3.
      REACH OE=E9 FROM C4 GET (LOE,J,CNTOE,AOE).

         REACH OE=E10 FROM E9 VIA ATT GET (LOE,J).
              :
              :
                 REACH OE=E13 FROM E12 VIA ATT GET (LOE,J).
                       COMPUTE LOE OF E12 = LOE OF E12
                             + (LOE OF E13 * J OF E13).
              :  END E13.
              :
              :
            COMPUTE LOE OF E9 = LOE OF E9 + (LOE OF E10 * J OF E10).
         MODIFY E10.
         END E10.
      COMPUTE LOE OF E9 = LOE OF E9 * J OF E9.
         "calculer l'indice du compteur de valeurs;
          incrémenter la longueur de la zone compteurs;
          générer les aspects de l'objet élémentaire;
          garnir CNTOE, AOE".
      MODIFY E9.
      END E9.
       "générer les aspects des pointeurs et des compteurs;
        garnir LPT, APT, LCT, ACT".
MODIFY C4.
   "réinitialiser la routine de génération d'aspects".
END C4.
```

3. Vérification des contraintes d'intégrité

Les réalisations et les occurrences du métaschéma doivent satisfaire à un certain nombre de contraintes d'intégrité.
Certaines de ces contraintes représentent des propriétés du modèle d'accès et ont été décrites en détail, au chapitre II de la première partie. D'autres contraintes représentent des restrictions dues à la représentation SESAM où lient entre elles les données d'implémentation. Bien que la vérification de ces dernières contraintes soit absolument nécessaire, nous n'en parlerons pas ici, pour des raisons de concision. Nous décrirons seulement la vérification des contraintes du premier type.

En réalité, la plupart de ces contraintes sont vérifiées dans le programme de chargement de la méta-BD. Seules, deux d'entre elles font l'objet d'un programme de vérification particulier.

3.1. *Contraintes associées au schéma*

Les contraintes (C1), (C2), (C3) décrites au paragraphe 4.1. du chapitre I de la première partie sont vérifiées par la structure de blocs du programme de chargement.

3.2. *Contraintes associées aux objets élémentaires*

Lorsque l'analyseur syntaxique analyse une déclaration d'objet élémentaire il s'assure que cette déclaration contient soit un clause <picture> soit une clause <descendants> et le chargement de la méta-BD s'effectue en conséquence (C4).
Comme l'origine d'une occurrence de ATT (OE,OE) est toujours créée avant la cible, la contrainte (C5) est vérifiée.

3.3. *Contraintes associées aux relations entre objets complexes*

L'analyseur syntaxique ne passera la main au programme de chargement de la méta-BD qu'après vérification des valeurs de i,j,k,l spécifiées dans une déclaration de relation (C6).

Le programme de création des réalisations de REL respecte bien la contrainte (C7). (cfr pages 93 et 94)

La contrainte (C8) est plus difficile à vérifier et fait l'objet d'un programme particulier que nous verrons plus loin.

3.4. *Contraintes associées aux relations entre objet complexe et objet élémentaire*

Le programme de création des réalisations de OE décrit aux pages 90 à 92 est tel que les contraintes (C9) (C10) sont vérifiées.

3.5. *Contraintes associées aux noms*

Les contraintes (C11) et (C12) sont vérifiées avant la création des réalisations de OC et de REL.
La contrainte (C13) fait l'objet d'un programme particulier.

3.6. *Contrainte relative aux noms d'objet élémentaire*

Cette contrainte est vérifiée par l'exploitation d'une table permettant de construire les listes de désignation de tout objet élémentaire associé à un objet complexe.

Etant donné la déclaration suivante :

```
COMPLEX-OBJECT NAME IS PERSONNE;
ELEMENTARY-OBJECT NAMES ARE :
   01 NOM PIC X(20) IDENTIFIER,
   01 PRENOM OCCURS (0,5) PIC X(10),
   01 ADRESSE,
      02 RUE PIC X(15),
      02 NUMERO PIC 9(5),
      02 LOCALITE PIC X(15).
```

La table sera constituée comme suit :

INDICE	NOM	PERE	DESCENDANTS
1	PERSONNE	0	OUI
2	NOM	1	NON
3	PRENOM	1	NON
4	ADRESSE	1	OUI
5	RUE	4	NON
6	NUMERO	4	NON
7	LOCALITE	4	NON

La construction de cette table ne pose aucun problème puisqu'elle peut être garnie lors de l'exécution du programme nécessaire au calcul des longueurs des objets élémentaires. Elle est exploitée par un module séparé qui imprime les listes de désignation redondantes.

L'algorithme de vérification est le suivant.
Après avoir découvert, dans la table, un objet élémentaire ne possédant pas de descendant, on construit sa liste stricte de désignation grâce à une exploitation de bas en haut de la colonne "PERE" de la table. Cette liste stricte est ensuite comparée à toutes les autres listes, qu'il est possible de construire à l'aide de cette table, afin de découvrir les redondances éventuelles. Ensuite, on recommence l'opération pour toutes les listes strictes incluses dans la première liste stricte.
Lorsque la vérification d'un objet élémentaire est réalisée, on passe à l'objet élémentaire suivant, ne possédant pas de descendant, et ce jusqu'à épuisement de la table.

3.7. *Détection des cycles de relations fortes*

Nous appellerons *relation forte*, une relation entre objets complexes telle que k > 0.
Etant donné qu'une relation inverse, même non déclarée, est toujours représentée dans la méta-BD, la contrainte d'intégrité (C8) se simplifie : il suffit de vérifier que la méta-BD ne contient pas de réalisations de REL représentant un cycle de relations fortes.

Méthode de vérification

La vérification de cette contrainte, vu sa relative complexité, a été réalisée par un programme séparé qui accepte en entrée la méta-BD et fournit, dans le cas de non-respect de la contrainte, une liste de tous les objets complexes, sommets d'un cycle de relations fortes.

L'algorithme utilisé nécessite l'emploi :

- d'une chaîne de bits où le bit de rang i représente l'état de l'objet complexe de numéro i :
  bit i = 1 : l'objet complexe numéro i a déjà été traité par l'algorithme
  bit i = 0 : l'objet numéro i n'est pas traité;
- d'une pile capable de stocker tous les numéros des objets complexes participant à une liste d'objets complexes reliés par relations fortes;
- d'un vecteur NIV de même dimension que la pile;
- d'une variable d'itération i; d'une variable d'itération k;
- d'une variable de stockage intermédiaire : VAR.

Nous donnons ci-après deux exemples de schémas pouvant être examinés par l'algorithme :

schéma correct

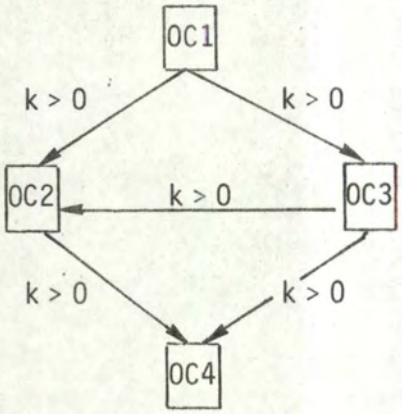
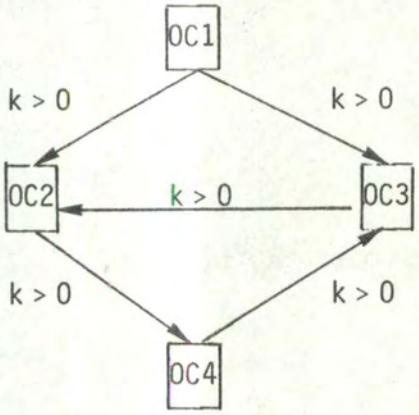


schéma erroné



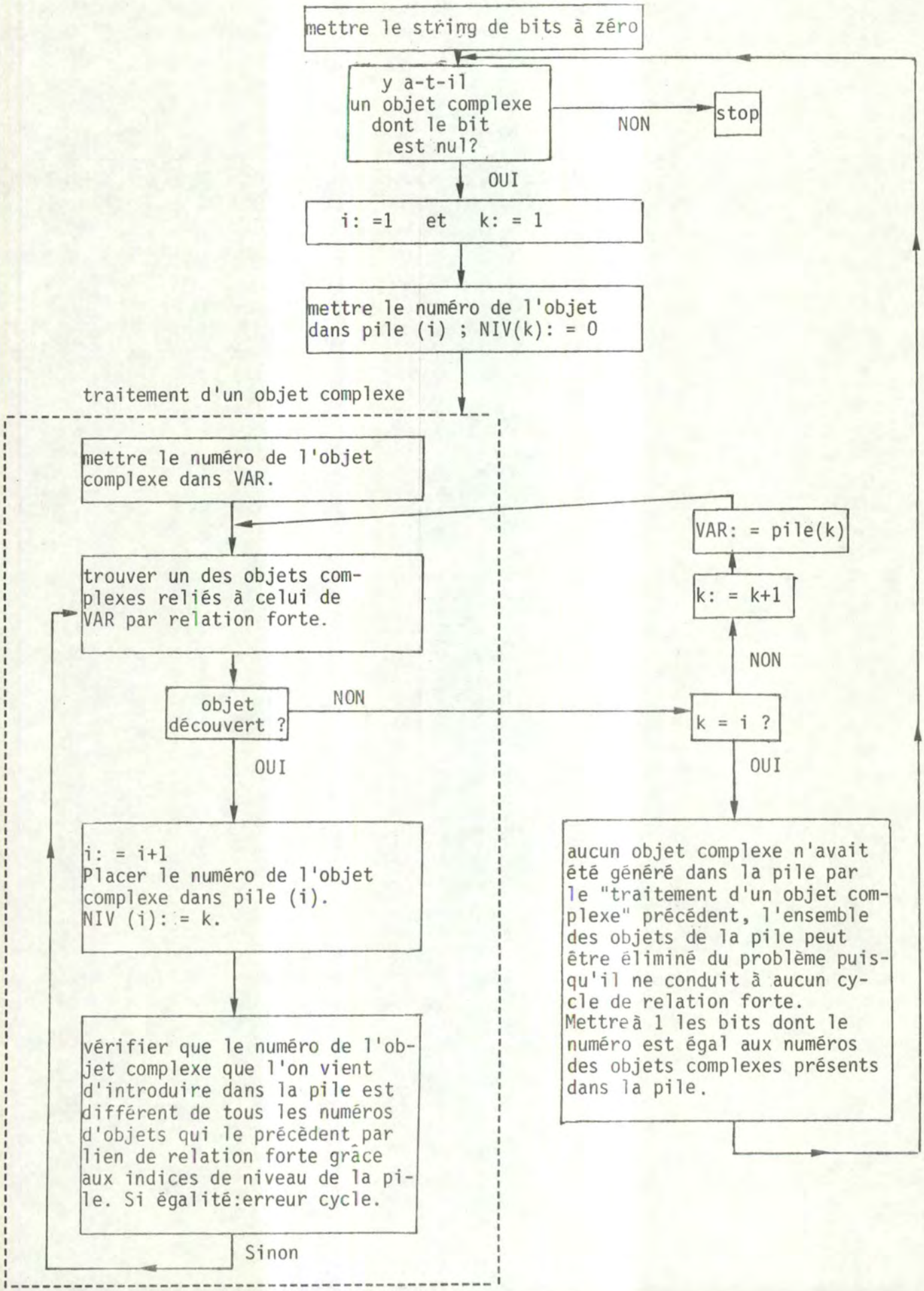
cycle : OC2,OC4,OC3

Remarques

- La boucle de "traitement d'un objet complexe" a été réalisée par :

```
REACH OC=C1 IF (NUMERO=VAR).
|   REACH REL=R1 FROM C1 VIA ORIGINE GET (K).
|   |   IF K OF R1 NOT > 0  NEXT R1.
|   |   REACH OC=C2 FROM R1 VIA CIBLE GET (NUMERO).
|   |   |       "traitement"
|   |   END C2.
|   END R1.
END C1.
```

- L'utilisation de variables de travail indicées aurait permis la suppression de la pile contenant les numéros d'objets.


mettre le string de bits à zéro
y a-t-il un objet complexe dont le bit est nul?
NON
stop
OUI
i: =1 et k: = 1
mettre le numéro de l'objet dans pile (i) ; NIV(k): = 0
traitement d'un objet complexe
mettre le numéro de l'objet complexe dans VAR.
trouver un des objets complexes reliés à celui de VAR par relation forte.
objet découvert ?
NON
OUI
i: = i+1
Placer le numéro de l'objet complexe dans pile (i).
NIV (i): = k.
vérifier que le numéro de l'objet complexe que l'on vient d'introduire dans la pile est différent de tous les numéros d'objets qui le précèdent par lien de relation forte grâce aux indices de niveau de la pile. Si égalité:erreur cycle.
Sinon
VAR: = pile(k)
k: = k+1
NON
k = i ?
OUI
aucun objet complexe n'avait été généré dans la pile par le "traitement d'un objet complexe" précédent, l'ensemble des objets de la pile peut être éliminé du problème puisqu'il ne conduit à aucun cycle de relation forte.
Mettre à 1 les bits dont le numéro est égal aux numéros des objets complexes présents dans la pile.

4. Création des fichiers nécessaires aux utilitaires SESAM

4.1. *Routine de génération d'aspects*

4.1.1. Définition du problème

Nous allons décrire la routine de génération d'aspects utilisée par le programme de chargement de la méta-BD. Lorsque l'on détermine les aspects associés à un objet complexe particulier, il est nécessaire d'affecter des aspects différents

- à tous les objets élémentaires reliés à l'objet complexe,
- à toutes les relations vers un autre objet complexe, telles que j > 1, issues de celui-ci,
- aux zones contenant les pointeurs et les compteurs.

Mais lorsque l'on a fixé les caractéristiques de ces aspects et que l'on s'intéresse à un autre objet complexe, on peut les récupérer à d'autres fins, puisque leurs valeurs seront associées à d'autres valeurs de l'aspect ordre. Cette récupération ne se fera que si le nouvel objet complexe nécessite des aspects ayant les caractéristiques fixées précédemment.

Exemple

Soient les déclarations d'objets complexes :

```
COMPLEX-OBJECT NAME IS PERSONNE;
ELEMENTARY-OBJECT NAMES ARE :
    01 NOM PIC X(15),  (aspect BAZ)
    01 RUE PIC X(15).  (aspect BBA)
COMPLEX-OBJECT NAME IS ENTREPRISE;
ELEMENTARY-OBJECT NAMES ARE :
    01 NOM PIC X(20),  (aspect BBB)
    01 RUE PIC X(15).  (aspect BAZ)
```

Les objets élémentaires (PERSONNE,NOM) et (ENTREPRISE,RUE) ont mêmes caractéristiques, ilssont représentés par le même aspect SESAM.

4.1.2. Algorithme de gestion des aspects

Génération des aspects

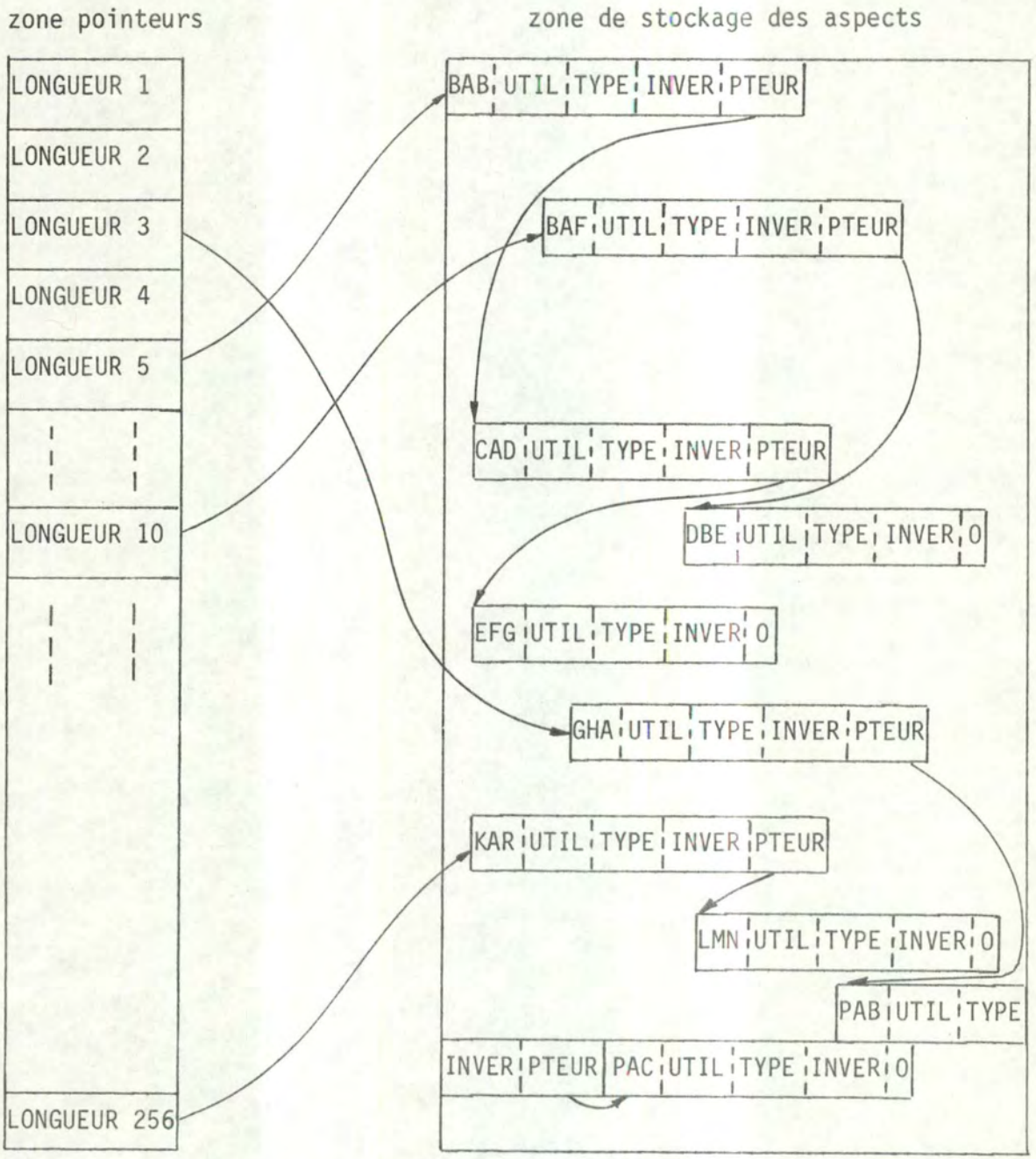
Lorsqu'un nouvel aspect est nécessaire au programme de chargement, celui-ci fait appel à la routine en lui communiquant toutes les caractéristiques de l'aspect : longueur, type des caractères, inversé ou non. En retour, la routine lui délivre un nom d'aspect de 3 caractères. Les aspects AAB à A99 étant réservés au système SESAM, la génération des noms d'aspects se fera séquentiellement de BAA jusqu'à Z99.

Stockage des aspects générés

Les aspects générés sont placés dans des listes dont le critère d'entrée est la longueur d'aspect.
Chaque élément de liste comporte :
- le nom d'aspect,
- un bit indiquant si l'aspect a déjà été délivré au programme de chargement pour l'objet complexe courant , ou non, (1=oui, 0=non),
- un bit indiquant le type des caractères de l'aspect (1=numérique, 0=quelconque),
- un bit indiquant si l'aspect est inversé ou non,
- un pointeur vers l'élément suivant de la liste, le pointeur nul représentant la fin de liste.

Récupération des aspects

Lorsqu'un nouvel aspect est demandé, on accède à la liste correspondant à sa longueur et on parcourt celle-ci jusqu'à découvrir un aspect possédant les caractéristiques désirées et dont le bit d'utilisation est à zéro.
Si la liste ne contient pas un tel élément, on crée un nouvel aspect, répondant à la demande, que l'on place en fin de liste. Son nom est, alors, retourné au programme de chargement.
Lorsque le programme de chargement a terminé le traitement d'un objet complexe particulier, il effectue un appel spécial à la routine pour remettre à zéro tous les bits d'utilisation.
Cette structure mémoire peut être visualisée grâce à la figure de la page suivante.



En réalité, il n'y a bien sûr aucun espace libre dans la zone de stockage puisque l'affectation de cette mémoire est séquentielle, mais, dans un but de clarté, nous n'avons représenté que quelques aspects.

4.1.3. Création du fichier d'entrée de SEFO

Lorsque le programme de chargement a affecté des aspects à tous les objets complexes d'un schéma, il effectue un dernier appel à la routine de génération d'aspects. Celle-ci contient la description de tous les aspects de la base de données SESAM correspondant au nouveau schéma. Elle crée, alors, le fichier d'entrée nécessaire au programme utilitaire SEFO, dont nous avons parlé dans la deuxième partie, qui permet de définir le schéma SESAM de la nouvelle base de données.

4.2. *Création des fichiers d'entrée de SEBE*

4.2.1. Définition des données d'entrée

Nous avons vu, précédemment,que le chargement initial d'une base de données SESAM s'effectuait à l'aide d'un programme utilitaire (SEBE). Il est donc absolument nécessaire, lors de la déclaration d'un schéma du modèle d'accès, de fournir des données à cet utilitaire, bien qu'elles ne correspondent pas à des réalisations d'objet, puisqu'elles ne sont pas créées par un programme DML travaillant sur le nouveau schéma.

Pour chaque objet complexe du nouveau schéma, on définira un enregistrement SESAM, ayant des valeurs d'aspects correspondant à des valeurs "inconnu" des objets élémentaires auxquels ils sont affectés et ayant "$$<numéro d'objet>" pour valeur d'aspect d'ordre. Ces enregistrements seront utilisés, par l'interface du DML, pour initialiser les zones COBOL générées par la compilation d'un ordre "PREPARE". Ils constitueront, de plus, des données pour l'utilitaire SESAM de chargement.
Les règles suivantes définissent la valeur "inconnu" d'un objet élémentaire quelconque.

- La valeur "inconnu" d'un objet élémentaire simple, à valeurs quelconques, est "?" suivi de blancs.
- La valeur "inconnu" d'un objet élémentaire simple, à valeurs numériques, est 0.
- La valeur "inconnu" d'un objet élémentaire composé est formée de la concaténation des valeurs "inconnu" de ses attributs.

Exemple

L'objet élémentaire

```
01 ADRESSE ,
     02 NUMERO PIC 9(4),
     02 RUE PIC X(10),
     02 CODE-POSTAL PIC 9(5),
     02 LOCALITE PIC X(9)
```

a pour valeur "inconnu" :

"0000?¥¥¥¥¥¥¥¥¥00000?¥¥¥¥¥¥¥¥".

4.2.2. Méthode de génération

Deux fichiers doivent être générés.

- Le premier, que nous désignerons par FILE1, contient les enregistrements définis ci-dessus.
- Le second, que nous désignerons par FILE2, décrit le format de ces enregistrements et leur découpe en aspects.

Ces fichiers seront générés grâce à deux sous-programmes. L'un de ceux-ci crée FILE1 et FILE2 à partir des valeurs qui lui sont fournies par le second. Il utilise un buffer BUF1 dans lequel seront constitués les enregistrements de FILE1. Le pointeur FIRSTFREE indique le premier caractère libre de BUF1. Ce sous-programme possède six points d'entrée.

- OPENFILES et CLOSEFILES permettent d'initialiser et de clôturer les fichiers.
- GENOA a pour argument un numéro d'objet complexe. Il place la valeur "§§<numéro d'objet complexe>" au début de BUF1 et positionne FIRSTFREE sur le caractère suivant. Il écrit dans le fichier FILE2 un enregistrement descriptif correspondant à l'aspect d'ordre.
- GENAOE possède deux arguments : un vecteur de 1 à 5 noms d'aspect et la somme des longueurs d'aspect. Il écrit, dans FILE2, un enregistrement descriptif pour chaque aspect.

- GENVALUE a pour argument une valeur de DESCRIPTEUR. Il place dans BUF1 à l'emplacement spécifié par FIRSTFREE, la valeur "inconnu" correspondant à la valeur de DESCRIPTEUR et incrémente FIRSTFREE.
- WRTBUF1 écrit BUF1 dans le fichier FILE1.

Le second sous-programme recherche, dans la méta-BD, les valeurs nécessaires au précédent. Il est rédigé comme suit.

```
OPEN MR=MT.
   ALLOCATE RAC=R1.
      REACH RAC=R2 FROM MT IN R1.
      END R2.
   CALL OPENFILES.

   REACH OC=C1 FROM R1 GET (NUMERO).
   CALL GENOA USING NOBJ OF C1.

      REACH OE=E1 FROM C1 GET (AOE,LOE,J,DESCRIPTEUR).
      CALL GENAOE USING AOE OF E1(1),LOE OF E1.
      PERFORM BEGIN1 THRU EXIT1 J OF E1 TIMES.
      NEXT E1.

      BEGIN1.
          IF COUNT DESCRIPTEUR FROM E1=1
          THEN CALL GENVALUE USING DESCRIPTEUR OF E1.
          REACH OE=E2 FROM E1 VIA ATT GET (J,DESCRIPTEUR).
          PERFORM BEGIN2 THRU EXIT2 J OF E2 TIMES.
          NEXT E2.
            |
            |
            |
               BEGIN4.
                   IF COUNT DESCRIPTEUR FROM E4=1
                   THEN CALL GENVALUE USING DESCRIPTEUR OF E4.
                   REACH OE=E5 FROM E4 VIA ATT GET (J,DESCRIPTEUR).
                   PERFORM BEGIN5 J OF E5 TIMES.
                   NEXT E5.
```

```
                    BEGIN5.
                        CALL GENVALUE USING DESCRIPTEUR OF E5.
                    EXIT5.
                END E5.
            EXIT4.
                EXIT.
            END4.
            END E4.
             ¦
             ¦
             ¦
    EXIT1.
      EXIT.
    END1.
    END E1.

CALL WRTBUF1.
END C1.

CALL CLOSEFILES.

CLOSE MT.
```

CONCLUSION ET PROLONGEMENTS POSSIBLES

Nous pensons avoir montré, dans ce travail, que le modèle d'accès et son DML permettent de réaliser, de façon assez naturelle, la définition, la création et l'exploitation des schémas internes d'un ensemble de bases de données (méta-BD).

La définition du métaschéma n'a pas posé de problème particulier. Toutefois, les types de données du modèle d'accès se sont avérés insuffisants pour exprimer entièrement la nature des informations contenues dans la méta-BD, ce qui nous a conduit à compléter le métaschéma par un ensemble de contraintes d'intégrité.
Parmi celles-ci, certaines représentent des propriétés ou des règles associées aux types de données du modèle d'accès. Elles doivent être vérifiées, une fois pour toutes, au moment du chargement de la méta-BD, correspondant à la compilation d'un programme DDL. Un tel programme peut, en effet, contenir des déclarations ne respectant pas les règles du modèle d'accès.
D'autres contraintes concernent les données d'implémentation. Comme ces dernières sont entièrement construites par le compilateur du DDL, la vérification de ces contraintes est inutile.
En définitive, on arrive à la conclusion que la meilleure façon de décrire entièrement la nature des données contenues dans la méta-BD est d'adjoindre au métaschéma, le programme de chargement.

Les traitements nécessaires à la création et à l'exploitation de la méta-BD ont été réalisés, sans difficultés importantes, grâce au DML. Certaines améliorations à ce langage ont cependant été suggérées. Ces modifications nous auraient facilité la tâche, mais nous n'avons pas mesuré leur coût d'implémentation, ni prouvé leur cohérence avec le reste du DML.

A partir du travail que nous avons réalisé, divers prolongements nous paraissent dignes d'intérêt.

Tout d'abord, le problème de la portabilité pourrait être étudié en définissant l'ensemble des représentations internes possibles des types de données du modèle d'accès et en examinant les modifications à apporter au métaschéma et aux compilateurs lorsque l'on passe d'une représentation à une autre. Cette étude pourrait conduire, finalement, à la définition d'un métaschéma et de compilateurs généraux, valables pour toutes les représentations internes.

Le DDL que nous avons défini pourrait être étendu de plusieurs façons en y ajoutant des ordres permettant, par exemple, d'assurer le secret, de définir des sous-schémas, de modifier un schéma ou un sous-schéma.
La notion de métaschéma nous paraît bien adaptée à la réalisation de ces diverses extensions.

Enfin, il devrait être possible de représenter entièrement le modèle d'accès (y compris les actions primitives), par le métaschéma sémantique, considérablement étendu, et par un ensemble de programmes DML agissant sur la base de données qui lui correspond. Dans cette démarche, toute action relative à une base de données particulière serait représentée par l'exécution d'un programme agissant sur la méta-BD.
Cette "formalisation" du modèle d'accès permettrait de donner une définition de celui-ci plus précise que celle que nous avons formulée, car débarrassée de tout sous-entendu, oubli ou même incohérence, difficiles à éviter lorsqu'on utilise le langage courant.
En outre, on peut espérer utiliser ce métaschéma comme cadre général pour étudier certains problèmes fondamentaux posés par les bases de données tels, par exemple, que la concurrence.

BIBLIOGRAPHIE

(1) *Système de conception et d'exploitation d'une base de données,* Projet de recherche C.I.P.S. n° 1.2/15, Institut d'Informatique, F.N.D.P., Namur (1974).

(2) J.L. Hainaut et B. Le Charlier, *An extensible semantic model of Data base and its data language,* Proc. IFIP Congress 1974, North-Holland Publish.Co., pp. 1026-1030.

(3) *Présentation et spécifications du modèle d'accès,* Documents internes de l'Institut d'Informatique, 15 mai 1973, 22 mai 1973, 22 janvier 1974.

(4) D. Gries, *Compiler construction for digital computers,* John Wiley & Sons, Inc., New-York (1971).

(5) CODASYL Data base task group, *April 71 Report.*

(6) Siemens-system 4004, Betriebssystem BS1000, *Sesam Aufbau und Wartung der Datenbank Beschreibung,* April 1975, Munich.

(7) Siemens-system 4004, Betriebssystem BS1000, *Sesam Wiederge winnung und Direktänderung von Daten Beschreibung,* April 1975, Munich.

(8) E. Benci, F. Bodart, H. Bogaert, A. Cabanes, *A relational model for a conceptual schema,* Institut d'Informatique, F.N.D.P., Namur (1975).

(9) C. Deheneffe, H. Hennebert, W. Paulus, *Relational model for a data base,* Proc. IFIP Congress 1974, North-Holland Publish.Co., pp. 1022-1025.