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

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Contribution à l'atelier de conception de bases de données : extension au système de gestion de bases de données MDBS

DEVILLE, Philippe; Hoffelt, P.

Award date:
1987

Awarding institution:
Universite de Namur

Link to publication

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

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

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

Download date: 03. Jul. 2025

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

CONTRIBUTION A L'ATELIER DE CONCEPTION DE BASES DE DONNÉES :

Extension au Système de Gestion de Bases de Données MDBS

P. Deville & P. Hoffelt VOLUME 1

PROMOTEUR : J.L. HAINAUT

MEMOIRE PRESENTE
EN VUE DE L'OBTENTION DU GRADE
DE LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1986 - 1987

P A R T I E 1 C A D R E G E N E R A L

Chapitre I. - LA DEMARCHE DE CONCEPTION D'UNE BASE DE DONNEES 6

I.1. **Introduction** 7

I.2. **Les modèles** 9
I.2.1. Le modèle Entité-Association 9
I.2.2. Le modèle d'accès généralisé 11
I.2.3. Le langage de description d'algorithmes 22

I.3. **La démarche de conception d'une base de données** 25
I.3.1. Introduction 25
I.3.2. Présentation de la démarche 25

I.4 **Les modules d'accès** 28
I.4.1. Introduction 28
I.4.2. La notion de module d'accès 28
I.4.3. Deux rôles pour un module d'accès 31
I.4.4. Les modules d'accès dans la démarche de conception 36

Chapitre II. - L'ATELIER DE CONCEPTION DE BASE DE DONNEES : ORGA 40

II.1. **Généralités** 41

II.2. **Notion d'Atelier logicel** 42
II.2.1. La base des spécifications 42
II.2.2. Les outils 42

II.3. **Présentation de l'Atelier BD** 45
II.3.1. IDA, un environnement de conception 45
II.3.2. Les objectifs assignés à l'Atelier BD 46
II.3.3. Constitution de la base de spécifications locale 47
II.3.4. Fonctions actuelles de l'Atelier BD 49
II.3.5. Scénario dans l'Atelier 49

II.4. **L'architecture de l'atelier BD** 50
II.4.1. Introduction 50
II.4.2. Les tool-kits 51
II.4.2.1. Les outils de gestion des données complexes 52
II.4.2.2. Les outils de gestion des échanges 53

P A R T I E 2

Chapitre III. - LES OBJECTIFS DU MEMOIRE 55

III.1. **MDBS, schéma cible dans l'Atelier BD** 57

III.2. **MDBS comme support de l'Atelier** 59

Chapitre IV. - LES SGBD DE TYPE RESEAU 60

IV.1. **Généralités** 62

IV.2. **La notion de standard** 64

IV.3. **Principes généraux des SGBD CODASYL 71** 65

Chapitre V. - MDBS, UN SGBD DE TYPE RESEAU 75

V.1. **Les caractéristiques MDBS** 77

V.2. **Les composants du SGBD MDBS** 79

Chapitre VI. - CONCEPTION DE BASE DE DONNEES MDBS 83

VI.1. **Généralités** 85

VI.2. **Schémas gérés par l'Atelier** 88

VI.3. **Identification des structures non conformes** 89

VI.4. **Les transformations nécessaires** 93

VI.5. **Les modifications nécessaires** 110

VI.6. **Le schéma MDBS** 111

VI.7. **Scénario de génération d'une base de données conforme MDBS** 112

VI.8. **Conclusion** 116

Chapitre VII. - INTEGRATION DE MDBS DANS ORGA 117

VII.1. **Introduction** 118

VII.2. **Le module d'accès ADLC** 121
VII.2.1. Les caractéristiques générales 121
VII.2.2. Les primitives 122
VII.2.3. L'utilisation des primitives ADLC 123
VII.2.4. L'environnement du programmeur 127
VII.2.5. Les Spécifications de ADLC 128

VII.3. **La version actuelle du composant accès à la base des spécifications** 135
VII.3.1. Introduction 135
VII.3.2. La couche physique 135
VII.3.3. La couche M5 136
VII.3.4. La couche sémantique 139
VII.3.5. La couche adlc 139

VII.4. **La nouvelle version du composant accès à la base des spéécifications** 140
VII.4.1. Le mapping 141
VII.4.2. L'émulation 142
VII.4.3. La gestion des courants 146
VII.4.4. La mise en oeuvre du composant d'accès à la base des spécifications 160
VII.4.4.1. Structuration modulaire 161
VII.4.4.2. Spécifications des composants 171

VII.5. **Le module d'accès SEM** 181

Chapitre VIII. - CONCLUSION 186

BIBLIOGRAPHIE 190

ANNEXE 1 : SCENARIO DE CONCEPTION D'UNE BASE DE DONNEES MDBS 193

ANNEXE 2 : GENERATION DE TEXTE MDBS A PARTIR DE L'ATELIER 204

ANNEXE 3 : STRUCTURE D'UN SYSTEME DE MODULES 221

REMERCIEMENTS

Nous voulons tout d'abord exprimer notre profonde gratitude à Monsieur J.L. Hainaut pour les nombreux conseils qu'il a prodigués tout au long de ce mémoire.

Nous exprimons également notre reconnaissance à Monsieur K. Chabottaux pour nous avoir permis d'effectuer, dans les meilleures conditions, un stage aux Forges de Clabecq. Il nous a fait partager son expérience tant dans le domaine informatique que dans le monde de l'industrie.

Nous adressons aussi nos remerciements à Monsieur J. Tourneur sans qui la partie implémention de notre mémoire n'aurait pas été possible.

Nous remercions Mademoiselle C. Charlot, Messieurs M. Cadelli, A. Delcourt et B. Delcourt pour l'intérêt qu'ils ont porté à notre travail. Enfin, que soient également remerciés tous les membres de l'Institut d'Informatique qui, d'une manière ou une autre, nous ont aidés au cours de cette année.

INTRODUCTION

L'élaboration d'une base de données physique, prise dans le cadre du développement d'une application informatique, était jusqu'il y a quelques années à ce point complexe que rien ne permettait de présager de la qualité de la base de données construite. Les démarches étaient alors trop pauvres, se limitant à appliquer quelques principes solides mais trop isolés pour résister à la critique.

L'étude d'une **démarche moderne de conception** est actuellement menée à l'Institut d'Informatique de Namur. Les résultats théoriques de ces travaux ainsi que des règles méthodologiques concrètes relatives à la conception des Systèmes d'Information ont fait l'objet de publications récentes :

- F. Bodart & Y. Pigneur, "Conception assistée des applications informatiques, 1. Etude d'opportunité et Analyse conceptuelle", MASSON, 1983.

- J.L. Hainaut, "Conception assistée des applications informatiques, 2. Conception de la base de données", MASSON, 1986.

Nous remarquons que, selon les termes des auteurs, la démarche n'est envisageable que via un outil d'aide à la conception assurant la maîtise aux différents niveaux du processus de conception.

Il est question dans ce mémoire de la présentation d'un Atelier de conception de bases de données. Plus précisément, nous avons apporté deux extensions à la version actuelle de l'Atelier BD. La première permet la génération dans l'Atelier de schémas cibles MDBS. La seconde concerne l'intégration du Système de Gestion de Bases de Données MDBS comme support de l'atelier.

L'extension de l'Atelier BD à la génération de schémas MDBS devrait permettre d'évaluer l'aisance par laquelle on peut instancier l'Atelier général à un Atelier particulier. Elle devrait en outre confirmer la souplesse d'utilisation de l'Atelier BD, au sens où ce dernier prescrit une démarche de conception sans pour autant supplanter le concepteur.

Le second objectif assigné à ce mémoire, et sans doute le plus important, était d'intégrer MDBS comme outil de base de l'Atelier. Les outils de l'Atelier mémorisent, consultent et mettent à jour la description d'une solution (schéma ou traitement) contenue dans la base des spécifications. Une évaluation de l'architecture a identifié le composant d'accès à la base des spécifications comme devant faire l'objet d'améliorations sensibles. La version actuelle utilise son propre gestionnaire de données Ce dernier pourrait être amélioré à différents niveaux.

Le RUN-TIME occupe la presque totalité de la mémoire centrale. Ce qui limite très fortement l'espace disponible pour les données. La révision de l'architecture devrait permettre d'améliorer la compacité du RUN-TIME.

La version actuelle de l'Atelier souffre de la définition d'un nombre important de couches : ces dernières permettent la portabilité exceptionnelle de l'Atelier. La nouvelle architecture serait basée sur une représentation directe des concepts MAG (les concepts manipulés par l'Atelier). à l'aide de techniques de type CODASYL. On peut donc s'attendre à de meilleures performances.

La maintenance couvre à la fois la modification des spécifications ainsi que la correction d'erreurs découvertes lors de l'exploitation. Que valait-il mieux ? Acheter ou développer soi-même le composant d'accès à la base des spécifications. Développer soi-même permet la conception d'un produit adapté à ses besoins. La maintenance est alors entièrement à charge de l'Atelier BD. Acheter, c'est décider qu'un Système de Gestion de Bases de Données est un composant de base tout comme un Système d'Exploitation. Dans cette optique, les "release" successifs sont à charge du concepteur.

La portabilité d'un logiciel est limitée par les contraintes définies sur ses composants (Système d'Exploitation, Système de Gestion de Bases de Données, langage de programmation). L'Atelier a été écrit en C et ne fait l'objet d'aucune hypothèse sinon de disposer d'une technique d'accès séquentiel indexé. La nouvelle architecture propose de substituer au gestionnaire actuel des données le Système commercial de Gestion de Bases des Données MDBS. Par rapport à l'ancienne architecture, le composant MDBS restreint la portabilité. Néanmoins, de nom-breuses configurations peuvent recevoir MDBS. MDBS est disponible sur les machines Z80, 8086, 8088. Il n'est pas non plus lié à un Système d'Exploitation particulier (XENIX, UNIX, DOS, CP-M,...). Et enfin, il accepte de nombreux langages hôtes (PASCAL, COBOL, LATTICE C, PL-1, FORTRAN, BASIC,...).

En toute généralité, le coût de maintenance peut être estimé en fonction des critères théoriques suivants : le nombre de modules, le nombre de couches et le nombre de lignes de code. Remarquons cependant qu'à l'heure actuelle, aucun critère ne "marche" à tous les coups.

Notre mémoire devrait apporter une réponse à la question suivante : les investissements décidés serviront-ils bien les objectifs définis. L'approche par les objectifs doit encore être complétée par une évaluation d'ordre économique (coûts humains et coûts d'infrastructure). Remarquons dès à présent que très peu de processus formels permettent d'évaluer les avantages et les coûts d'une décision d'investissement. Il nous a donc fallu planifier et mettre en place les différentes étapes d'une **démarche expérimentale**. De telles considérations sont essentielles tant il est vrai que mettre en place une nouvelle technologie (en l'occurrence le SGBD MDBS) sans en maîtriser son contexte d'intégration relève d'une opération assez hasardeuse et pour le moins dangereuse.

La démarche mise en place devait permettre de maîtriser les différentes étapes de l'**étude de faisabilité**. La démarche s'exprime plus concrètement en termes de décisions et de contrôles.

- étude approfondie sur les choix d'architecture.
- réalisation par la société TRACTEBEL de Bruxelles du composant d'accès à la base des spécifications.
- intégration du composant dans l'Atelier.
- mise au point d'un système de mesure et d'évaluation.

C H A P I T R E I

LA DEMARCHE DE CONCEPTION D'UNE BASE DE DONNEES

I.1 INTRODUCTION

Les démarches de conception de bases de données qui sont en général recommandées dans la littérature font une grande part aux aspects conceptuels et elles négligent les aspects de mise en oeuvre.

La démarche de conception développée par le professeur J.L. Hainaut a le mérite de couvrir tout le processus de conception depuis la description conceptuelle jusqu'à l'obtention d'une solution exécutable. Désormais, le concepteur est muni de tous les éléments méthodologiques pour appréhender les problèmes post-conceptuels. Le lecteur intéressé trouvera dans la référence(1), en plus d'un exposé détaillé de la démarche, l'étude d'un cas concret. La démarche s'inspire directement de la hiérarchie traditionnelle à trois niveaux :

- niveau conceptuel
- niveau logique
- et - niveau physique.

Chaque niveau de conception est assigné à la réalisation de produits spécifiques.

La figure 1.1 présente un schéma générique de la démarche.

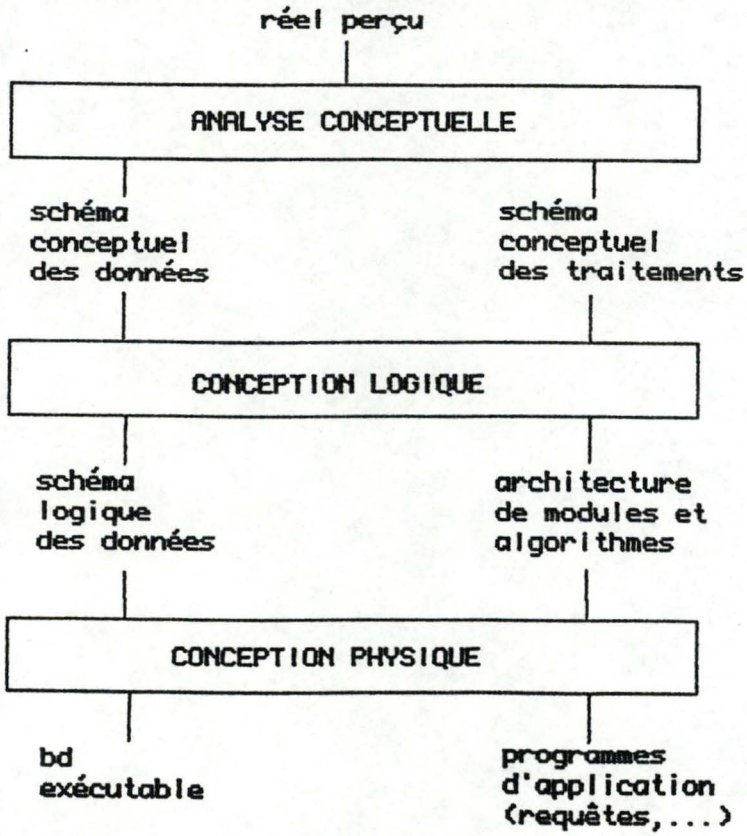


Figure 1.1

1.- J.L Hainaut, Conception assistée des applications informatiques. 2. Conception de la base de données, MASSON, 1986.

Le chapitre s'articule en trois points.

- Nous décrirons d'abord les modèles sur lesquels s'appuient les trois niveaux. Le modèle conceptuel Entité-Association fera l'objet d'une brève description. Nous insisterons d'avantage sur les modèles appropriés aux niveaux logique et physique (les modèles MAG et LDA).

- Nous commenterons ensuite les étapes qui composent la démarche de conception.

- Nous serons alors en mesure de définir la notion de module d'accès.

I.2 LES MODELES

I.2.1 LE MODELE ENTITE-ASSOCIATION

Toute organisation entretient un système d'information qui produit, enregistre et traite les informations nécessaires à ses activités.

Un système d'information se compose fondamentalement d'une mémoire, d'un ensemble de traitements et d'un ensemble de ressources (humaines, organisationnelles,...) qui sont requises à son fonctionnement.

La mémoire du système d'informations contient, outre les informations de l'organisation, la description de ces informations. Afin d'obtenir une définition rigoureuse des informations, il faut recourir à un modèle conceptuel de structuration des données possédant les qualités suivantes:

1.- Il fournit les moyens pour exprimer la sémantique des données. Et il reste confiné dans ce champ sémantique. Il est tout à fait étranger à des considérations techniques de représentation des données sur support magnétique ainsi que des mécanismes d'accès aux données.

2.- Un modèle conceptuel doit pouvoir être lu par des catégories distinctes de personnes (les informaticiens, les personnes chargées de l'exploitation du système d'infomation).

Le modèle Entité-Association (E-A) est le modèle de structuration des informations en vogue lors de ces dernières années. Ce modèle a fait l'objet d'une abondante littérature (Chen, Bachman,...). Pour notre part, nous conseillons l'ouvrage des professeurs Bodart et Pigneur (1). Nous nous limiterons donc à une évocation des principaux concepts du modèle E-A canonique.

Le modèle E-A comporte trois concepts de base (entité, association et attribut). Une **entité** est un objet ou un concept du monde réel. Les **entités** sont réparties dans des classes disjointes (les types d'entités).

1.- F. Bodart & Y. Pigneur, Conception assistée des applications informatiques. 1. Etude d'opportunité et Analyse conceptuelle, MASSON, 1983.

Les **entités** ne sont pas des choses isolées les unes par rapport aux autres. Il existe des **associations** entre ces entités. Les **associations** sont réparties dans des classes (types d'associations). La caractérisation d'un type d'associations dépend des types d'entités qui y participent. Quel est le nom de ces types d'entités ? Combien sont-ils (le degré du type d'associations) ? Quels rôles jouent-ils ? Dans combien d'associations une **entité** peut-elle figurer ? (la propriété de connectivité)

Le concept d'**attribut** est le mécanisme par lequel on peut enregistrer de l'information sur une entité ou sur une association. Un **attribut** incarne une propriété d'un type d'entités ou d'un type d'associations. On distingue les **attributs** simples des **attributs** répétitifs. Les **attributs** élémentaires des **attributs** décomposables et enfin les **attributs** obligatoires des **attributs** facultatifs.

Malgré la richesse de ces concepts, le modèle E-A n'est pas apte à représenter l'ensemble des propriétés sémantiques retenues par un concepteur. Aussi les concepteurs ne pourront pas se passer de la notion de contrainte d'intégrité.

> "Une contrainte d'intégrité est une propriété, non représentée par les concepts de base du modèle que doivent satisfaire les informations appartenant à la mémoire du système d'information."

Le modèle E-A reconnaît formellement un certain nombre de contraintes d'intégrité.

1) Les contraintes d'intégrité concernant les types d'entités

- la notion d'identifiant.

2) Les contraintes d'intégrité concernant les types d'associations

- la notion d'identifiant.
- les contraintes d'inclusion et d'exclusion.
- la notion de connectivité.

3. Les contraintes d'intégrité pour les attributs

- la notion de dépendance fonctionnelle.
- les contraintes de valeurs.

I.2.2 LE MODELE D'ACCES GENERALISE

1°) INTRODUCTION

La matérialisation de la mémoire du système d'information prend la forme d'une base de données.

Il existe sur le marché quantité de SGBD. Chaque SGBD possède ses caractéristiques propres. Pour s'en convaincre, rapportons nous aux seuls SGBD CODASYL. Chaque nouveau rapport CODASYL DBTG expose des amendements aux spécifications du modèle CODASYL. Qui plus est, les interprétations qu'en font les constructeurs concordent rarement. Sans parler des "release" successifs, que ces mêmes constructeurs apportent à leurs produits.

Néanmoins, si l'on examine les aspects "structures de données" et " primitives de manipulation de données", il ressort que les SGBD utilisent invariablement les mêmes concepts. La raison de leur disparité provient du fait que les contraintes sur les assemblages de ces concepts varient d'un SGBD à un autre.

Le modèle MAG reprend l'ensemble des concepts communs aux SGBD. Le Modèle d'Accès Généralisé (en abrégé,le MAG) est un modèle transitoire entre le modèle conceptuel et le modèle du SGBD. Les modèles SGBD ne sont pas des modèles conceptuels mais bien des modèles d'accès. Dès lors, il en va de même pour le modèle MAG.

Le modèle MAG est scindé en un noyau "sémantique" et un noyau d'"accès". Les structures de données du MAG sont des représentations des concepts provenant d'une description conceptuelle. Et elles recueillent ainsi la sémantique associée à ces concepts.

Mais le modèle MAG permet aussi d'exprimer les mécanismes d'accès aux données.

A l'instar du modèle E-A, le modèle MAG admet une représentation graphique.

2°) OBJETS DE BASE DU MAG

Les objets de base sont les articles et les types d'articles, les fichiers, la base de données, les valeurs d'items

et items, les chemins d'accès et leurs types, les clés d'accès et les ordres.

2.1°) L'ARTICLE

L'**article** est une entité stockée d'information qui peut être crée et supprimée, et à laquelle il est possible d'accéder dans la base de données.

L'**article** est l'unité d'échange entre un programme d'application et le SGBD.

Les **articles** présentant les mêmes propriétés appartiennent à un même type d'articles. Ils forment la population du type d'articles. Chaque type d'articles reçoit un nom distinct. La taille de la population évolue dans le temps. Les types d'articles sont représentés graphiquement par une figure rectangulaire.



2.2°) VALEURS D'ITEMS ET ITEMS

La notion d'item est appréhendée à partir de la notion de DOMAINE-PROPRIETE.

Un domaine-propriété est un ensemble de données de même nature. Par exemple, l'âge d'une personne ou le prix d'un produit. Une valeur d'item est un élément d'un domaine-propriété.

En général, un article est associé à plusieurs valeurs d'items. Mais il existe également des articles associés à une seule valeur d'item, voire à aucune valeur d'item. Pour connaître une valeur d'item, il est nécessaire d'accéder à l'article qui lui est associé.

Les domaines font partie intégrante de la définition des types d'articles. On dit qu'un domaine joue un rôle dans un type d'articles. Un domaine peut apparaître plusieurs fois dans la définition d'un type. Dans le modèle MAG, ce concept de rôle est désigné par le terme "**item du type d'articles**". Les items d'un type d'articles portent un nom distinct.

On distingue plusieurs sortes d'items.

1) Les items élémentaires et décomposables

Le domaine relatif à un item élémentaire est dit simple car ses éléments constituent des valeurs atomiques (des entiers, des caractères , des"strings",...).

La valeur d'un item décomposable regroupe une série de valeurs. Les domaines de ces valeurs portent le nom évocateur de "domaine composant". Récursivement, un composant peut être lui-même décomposable. Ses composants sont également considérés comme des items.

2) Les items simples et répétitifs

Un item d'un type d'articles est dit simple si les articles ne peuvent être associés qu'à une seule valeur de cet item.

Par contre, dans le cas des items répétitifs, plusieurs valeurs d'items peuvent être associées à un article. Il existe différents modes de répétivité. La répétivité est fixe lorsque chaque article est adjoint à un même nombre de valeurs. Dans le cas d'une répétivité limitée, ce nombre varie selon les articles mais est borné supérieurement. Le cas de la répétivité illimité revêt un intérêt essentiellement théorique.

3) Les items obligatoires et facultatifs

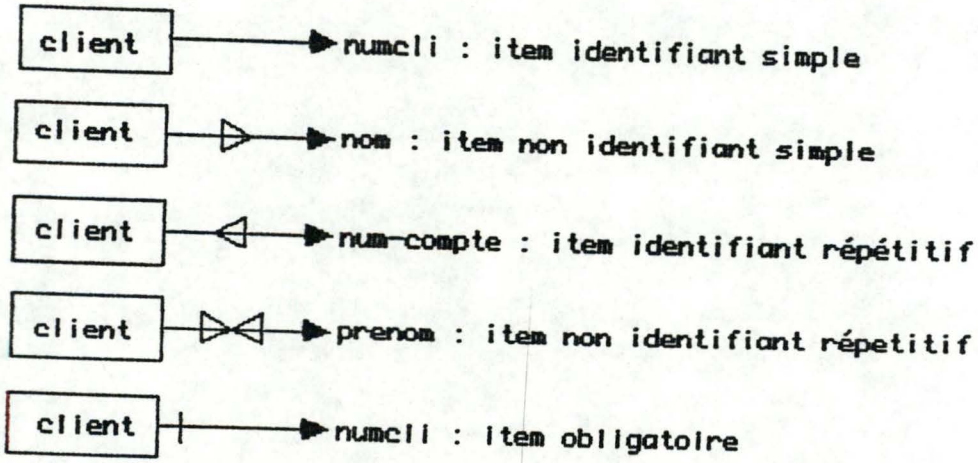
Un item d'un type d'articles est obligatoire si tout article est associé à au moins une valeur de cet item.

4) Les items identifiants et items non identifiants

Un item est identifiant s'il est impossible de trouver deux articles adjoints à une même valeur d'item. Cette propriété est valable pour toutes les populations possibles du type d'articles concerné.

Representation graphique

Un item est représenté par son nom. L'association d'un item à un type d'articles ou à un autre item est marquée par un arc. La configuration de cet arc renseigne le genre de l'item.



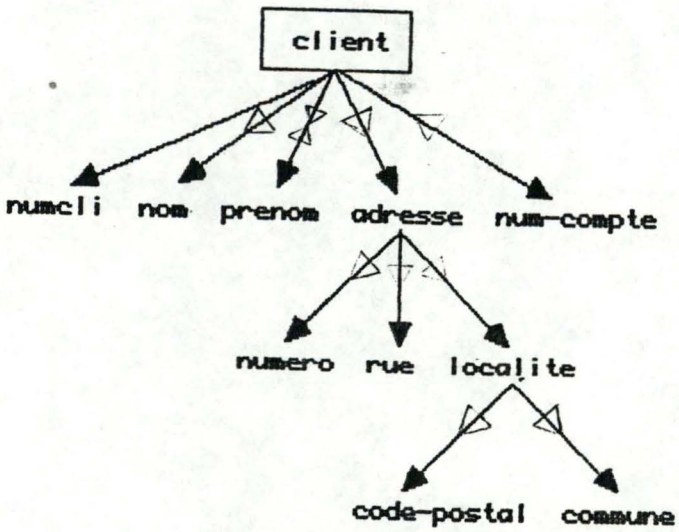


Figure 1.2

2.3°) LES CHEMINS D'ACCES ET LES TYPES DE CHEMIN D'ACCES

Le terme "**chemin d'accès**" est un raccourci pour "**chemin d'accès inter-articles**". Un **chemin d'accès** associe un article (l'article origine) à une suite de 0,1 ou plusieurs articles (les articles cibles). Mais la notion de chemin d'accès n'est pas seulement un moyen d'expression sémantique.

Etant donné un article origine, un chemin d'accès permet d'obtenir les articles cibles. Le chemin d'accès offre donc un accès unidirectionnel.

Les chemins présentant les mêmes propriétés appartiennent à un même type de chemins.

1) Type d'article origine et type d'articles cible

La définition d'un type de chemins précise d'une part les types d'articles qui peuvent être origines et d'autre part les types d'articles qui peuvent être cibles.

Les types de chemins sont désignés par un nom qui les identifie parmi les types de chemins ayant même origine et même cible.

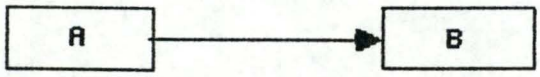
Souvent les types de chemins sont caractérisés par un seul type d'articles d'origine et un seul type d'articles cible. Les types de chemins à plusieurs origines (plusieurs cibles) sont dits multi-origines (respectivement multi-cibles).

L'origine d'un chemin multi-origines (une cible d'un chemin multi-cibles) appartient à l'un des types origines (respectivement à l'un des types cibles). Il arrive parfois qu'un type d'articles origine d'un type de chemins soit également un type d'article cible pour ce même type de chemins. On parle alors de type de chemins récursif.

2) Classe fonctionnelle

La classe fonctionnelle d'un type de chemins est une indication sur le nombre maximum d'articles d'un membre pouvant être asociés à un article de l'autre membre. Les seules valeurs admises pour ce nombre sont : un ou plusieurs.

On distingue quatre classes fonctionnelles. Considérons un type de chemin T d'origine A et de cible B.



T est de classe fonctionnelle 1-N

A tout élément de B, n'est jamais associé par T plus d'un élément de A. Il n'y a aucune restriction sur le nombre d'éléments de B associés à un élément de A.

T est de classe fontionnelle N-1

A tout élément de A, n'est jamais associé par T plus d'un élément de B. Il n'y a aucune restriction sur le nombre d'éléments de A associés à un élément de B.

T est de classe fonctionnelle 1-1

T est concomitamment de classe fonctionnelle 1-N et N-1.

T est de classe fonctionelle N-N

On ne relève aucune restriction sur les nombres d'éléments associés à un élément de l'autre membre.

3) Chemins obligatoires

T est un chemin obligatoire pour le type d'articles A, si tout article A est impérativement associé à au moins un article B. Il s'agit d'une contrainte d'existence posée sur le type articles A. Nous dirons de façon intuitive qu'un article A ne peut exister s'il n'est relié à un article B.

4) Types de chemins inverses

Considérons deux types de chemins TC1 et TC2 tels qu'il existe pour tout article cible C d'un chemin TC1 d'origine O un chemin TC2 d'origine C dont O est une cible. On dira que TC1 et TC2 sont des types de chemins inverses.

La présence de ces deux types de chemins assure un mécanisme d'accès bidirectionnel. Il est patent que ce mécanisme est plus coûteux que le simple accès unidirectionnel. Les concepteurs choisissent donc l'accès unidirectionnel lorsqu'il se révèle suffisant.

Représentation graphique

Un arc orienté symbolise un type de chemins. Cet arc joint les représentations de l'origine et de la cible. Il est étiqueté du nom du type de chemins. La configuration de l'arc dépend de la classe fonctionnelle ainsi que de la présence ou non d'une contrainte d'existence.

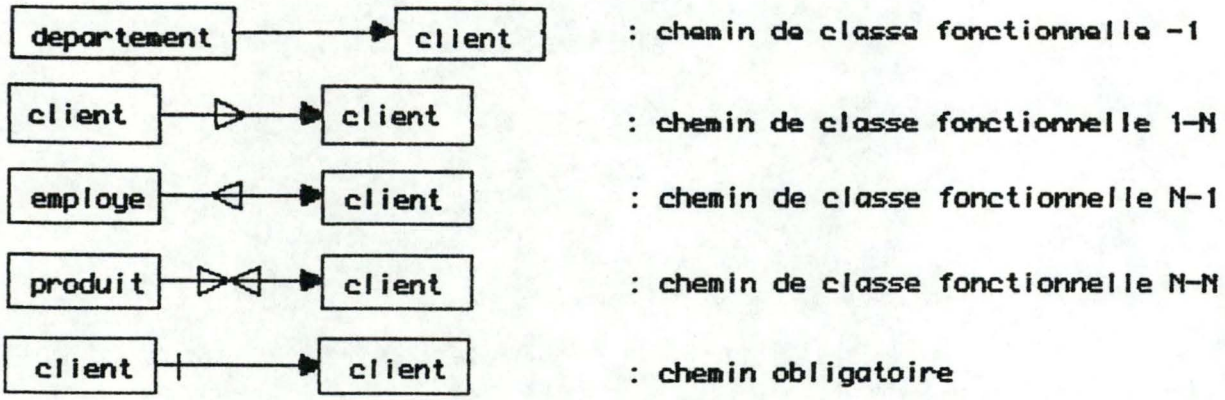


Figure 1.3

Les représentations sont donc conformes à celles qui ont été définies pour les items.

2.4°) LES IDENTIFIANTS COMPOSES

Nous avons abordé au point (2.2°) la propriété d'item identifiant.

Dans ce paragraphe, nous entendons élargir la notion d'identifiant. Un identifiant peut couvrir une combinaison d'objets. Ces objets sont des items ou des types d'articles adjoints au type d'articles à identifier via des types de chemin déterminés.

L'identifiant d'un type d'articles sera formé, soit :

- par un ou plusieurs items
- par au moins deux types d'articles.
- par un ou plusieurs items et un ou plusieurs types d'articles.

Représentation graphique

Un identifiant composé sera représenté par un parallélisme partiel des arcs.

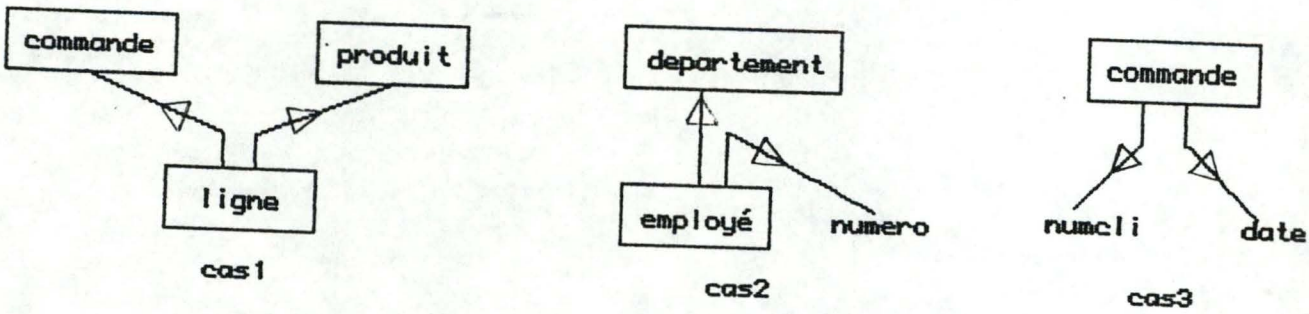


Figure 1.4

CAS1 : une commande ne contient pas plus d'une ligne relative à un même produit.

CAS2 : un employé est identifié conjointement par un numéro d'employé et par un département.

CAS3 : un client ne peut pas passer plus d'une commande le même jour.

2.5°) LE TYPE D'ARTICLE SYSTEME

Toute base de données contient un article d'un type particulier, l'article système. L'article système peut être l'origine de chemins d'accès mais ne peut jamais en être cible.

2.6°) LE FICHIER

Le **fichier** est une collection dynamique d'articles non nécessairement du même type.
Les fichiers d'une base de données reçoivent tous un nom différent.

Chaque article de la base réside dans un et un seul fichier. La population d'un type d'articles peut être disséminée dans plusieurs fichiers.

De plus en plus, les fabricants de SGBD insistent pour que la notion de fichier soit transparente au programmeur d'application.

2.7°) LA BASE DE DONNEES

La **base de données** regroupe les articles d'une collection de fichiers. Une base de données porte un nom susceptible de l'identifier parmi les autres bases de données connues dans un contexte déterminé.

2.8°) LES CLES D'ACCES

Une clé d'accès est constituée d'une suite de 0,1, ou plusieurs items d'un type d'articles . Ce sont les composants de la clé.

Définir une clé d'accès équivaut à se doter d'un moyen pour accéder aux articles ayant une valeur déterminée de clé.

Cette notion est parfois locale à un fichier ou à un chemin. On remarquera que dans les SGBD relationnels ce mécanisme est caché au programmeur d'application.

Représentation graphique

Une clé d'accès est symbolisée par un arc orienté partant de la représentation du ou de ses composants et aboutissant à celle du type d'articles.

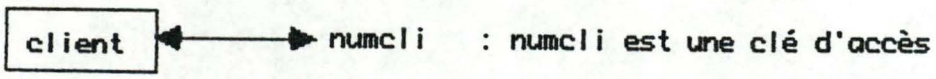


Figure 1.5

2.9°) ORDRE D'UNE SEQUENCE D'ARTICLES

En toute généralité, lorsqu'un utilisateur souhaite accéder à des articles , il commence par définir l'agrégat des articles qui sont visés. Puis, il accède séquentiellement aux articles selon un ordre prédéfini. Ci-dessous, nous mentionnons les différents ordres possibles :

- pas d'ordre prédéfini.
- ordre temporel :

  La position de l'article dans la séquence dépend de sa date de création. L'ordre chronologique veut que les articles les plus récemment crées soient insérés en fin de séquence. Par contre, si l'on adopte l'ordre anti-chronologique, les nouveaux articles se placent en début de séquence.

- ordre programmé :
  C'est au pogrammeur de choisir le lieu d'insertion des articles.

- ordre trié :
  Les articles sont triés par ordre croissant ou décroissant des valeurs d'un ou de plusieurs items.

I.2.3 LE LANGAGE DE DESCRIPTION D'ALGORITHMES

Le langage de description d'algorithmes(LDA) est délié des langages de programmation effectifs. Il perçoit les données via les structures MAG.

1°) LA DESIGNATION DES DONNEES

1.1°) LES VARIABLES ARTICLES ET LES VARIABLES LISTES

Les variables articles servent à référencer les articles de la base de données. Il existe également des variables pour réceptionner les valeurs d'item associés à un article. Une variable liste regroupe une séquence d'articles.

1.2°) ENSEMBLES ET SEQUENCES DE DONNEES

Les séquences et les ensembles sont des collections d'éléments (articles, valeurs d'item, simples valeurs).

Les séquences sont ordonnées. Et, elles tolèrent la présence d'éléments redondants.
On note les propriétés inverses pour les ensembles.

Les collections sont désignées par un identificateur (nom d'une variable, d'une constante ou d'un type d'éléments du schéma).

exemples :

- (0,2,4,6,8,10,...)
- CLIENT désigne l'ensemble des articles clients.
- NOM_CLIENT désigne l'ensemble des valeurs admissibles pour l'item NOM_CLIENT.
- CLI désigne un article CLIENT.
- (CLI1,CLI2,...CLI10) désigne une liste de dix articles CLIENT.

LDA permet de définir des sous-séquences de données. L'expression d'une sous-séquence de données répond à la forme canonique suivante :

"SEQ/VAR CSEL ORDER"

Les éléments d'une séquence SEQ sont retenus pour autant qu'ils satisfassent la condition CSEL. Les éléments sélectionnés

apparaissent selon un ordre ORDER. VAR sert à désigner un élément de SEQ lorsqu'il est confronté à la condition de sélection.

exemple :

```
client/cli (commune(:cli) = 'ROME' sorted (ncli) )
```
On choisit les clients dont la valeur de commune est égale à 'ROME'.

2°) LES STRUCTURES ALGORITHMIQUES

Hormis les conditions de sélection, LDA admet les structures algorithmiques suivantes :

1.1°) LA SEQUENCE D'INSTRUCTIONS

1.2°) L'ASSIGNATION

forme :

VAR := SOUS_SEQUENCE DE DONNEES.

VAR est une variable article ou une variable liste.

exemple :

```
cli := client (:numcli = '123')
liste-cli := client (:commune = 'ROME')
```

1.3°) L'ALTERNATIVE

forme :

```
if CONDITION then SEQUENCE1 else SEQUENCE2
endif
```

et `if CONDITION then SEQUENCE1 endif`

exemple :

```
if numcli(:cli) = '123' then
                  print nomcli(:cli)
                  endif
```

1.4°) LA BOUCLE WHILE

forme :

```
     while  CONDITION  do
                       SEQUENCE
                       endwhile
```

exemple :

```
while ( i<10 )  do
                create cli := client( (:numcli = '123')
                                      and (:nom = 'DUPONT')
                                      and (:commune = 'ROME') )
                endwhile
```

1.5°) LA BOUCLE ENUMERATIVE

forme :

```
     for VAR := (ORD)  SEQ_ELEMENT (while CONDITION)  do
         SEQUENCE1
     (if-no VAR  then SEQUENCE2)
     endfor
```

1) (...) indique une clause alternative.
2) ORD désigne un entier ou un intervalle d'entiers.

La boucle énumérative, encore appelée boucle d'accès, est utilisée pour accéder aux éléments de SEQ_ELEMENTS qui satisfont à CONDITION et dont le rang dans SEQ_ELEMENTS est compris dans ORD. Les instructions de SEQUENCE1 manipulent l'élément courant via la variable d'article VAR. Si aucun élément ne répond aux exigences, alors SEQUENCE2 est exécutée.

exemple :

```
for c := #1..5 client (:commune = 'ROME') sorted (numcli) do
     print nom(:c)
if-no c then
     print "Aucun de nos clients n'habite ROME."
endfor
```

3°) COMMANDES DE MISE A JOUR

LDA prevoit trois commandes de mise à jour des données :

- la modification d'un article
- la suppression d'un article
- et - la création d'un article.

I.3 LA DEMARCHE DE CONCEPTION D'UNE BASE DE DONNEES

I.3.1 INTRODUCTION

Ce chapitre présente de façon synthétique la démarche de conception d'une base de données. L'objet de la démarche est la production d'un schéma de base de données ainsi que de programmes, compilables par un processeur physique (DDL SGBD et compilateur d'un langage de programmation).

La qualité de la méthode découle de la prise en compte des principes qui suivent.

Les objectifs du produit final sont multiples et parfois hétérogènes. On cite entre autres la correction, les performances, la conformité aux contraintes d'un SGBD, l'indépendance des programmes vis-à-vis des données et l'adaptabilité par rapport aux besoins futurs. La démarche aborde tous les critères de décision relatifs à ces objectifs.

La méthode instaure une hiérarchisation des décisions, lesquelles sont regroupées en classes homogènes. Chaque classe peut dès lors être confiée à un processus autonome. Cette classification permet de surmonter la complexité du processus de conception. D'autre part, elle entraîne une localisation et une réduction des dépendances vis-à-vis du SGBD et des langages de programmation en retardant le plus possible les décisions afférant à ces composants.

I.3.2 PRESENTATION DE LA DEMARCHE

La démarche prend le relais de l'analyse de la démarche conceptuelle. Elle consiste en deux phases : la conception logique et la conception physique.

A l'issue de la première phase, on dispose d'une solution technique, correcte mais indépendante de la machine réelle. La solution sied à une machine virtuelle composée d'un processeur de procédures LDA et d'un SGBD MAG.

La deuxième phase convertit la solution logique en une solution physique exécutable par une machine réelle.

1°) L'ANALYSE CONCEPTUELLE

L'analyse conceptuelle produit une description complète du système d'information (schéma conceptuel des données, schéma conceptuel des traitements et relevé des quantifications) qui est indépendante de la notion d'outil informatique.

2°) LA CONCEPTION LOGIQUE

La conception logique se décompose en quatre étapes :

1) production du schéma des accès possibles (le SAP).
2) construction des algorithmes prédicatifs.
3) développement d'algorithmes efficaces.
4) développement du schéma des accès nécessaires (le SAN).

2.1°) PRODUCTION DU SCHEMA DES ACCES POSSIBLES

Le schéma des accès possibles (le SAP) est la traduction du schéma conceptuel dans les structures de données propres au MAG.

2.2°) CONSTRUCTION DES ALGORITHMES PREDICATIFS

Cette étape présuppose qu'une architecture de modules fonctionnels a été dérivée du schéma des traitements. Un pseudo-algorithme est rédigé pour chaque module fonctionnel qui accède à la base de données. Les pseudo-algorithmes perçoivent les données selon le SAP. De plus, ils sont déclaratifs. Autrement dit, ils ne mentionnent aucun mécanisme d'accès.

2.3°) DEVELOPPEMENT D'ALGORITHMES EFFICACES

Cette étape traite des stratégies d'accès. Les conditions de sélection qu'expriment un algorithme prédicatif ne sont pas toujours directement traduisibles par des mécanismes d'accès (accès séquentiel, accès par chemin, accès par clé). Les algoritmes prédicatifs sont transformés de telle sorte qu'ils ne contiennent plus que des conditions évaluables par accès.

En général, plusieurs alorithmes "évaluables par accès" peuvent être recensés pour un même algorithme prédicatif. Parmi ceux-ci, on retiendra celui qui implique le nombre minimal d'opérations logiques (l'algorithme effectif).

2.4°) DEVELOPPEMENT DU SCHEMA DES ACCES NECESSAIRES

Les mécanismes d'accès qu'utilisent les algorithmes effectifs sont greffés sur le SAP. On obtient ainsi le schéma des accès nécessaires (le SAN).

3°) LA CONCEPTION PHYSIQUE

La conception physique est scindée en deux parties : la conception physique "abstraite" et la conception logique "concrète".

La conception physique "abstraite" vise une solution conforme à une machine réelle. Toutefois, cette solution est encore exprimée dans les formalismes MAG et LDA.

La conception physique abstraite conduit à la solution exécutable. Elle comprend deux étapes :

1) la production d'un schéma conforme au SGBD.
2) production d'algorithmes effectifs conformes au SGBD.

La phase de conception physique "concrète" aboutit à la production du schéma SGBD, des schémas externes et des programmes.

3.1°) LA PRODUCTION D'UN SCHEMA CONFORME AU SGBD

Un schéma MAG constitue un schéma conforme à un SGBD pour autant qu'il ne comporte que des constructions admises par ce SGBD.

Le but de cette étape est de muer le SAN en un schéma conforme. Le schéma conforme obtenu est parfaitement équivalent au SAN en ce qui concerne les accès et la sémantique.

3.2°) PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES AU SGBD

Le passage du SAN vers un schéma conforme induit un ajustement des algorithmes qui travaillent sur le SAN.

D'autre part, les algorithmes doivent être également conformes au SGBD cible en ce qui concerne les primitives auxquelles ils font appel.

4°) CONCLUSION

Il ressort clairement que le processus de conception (logique et physique) d'une base de données implique une analyse conjointe des traitements.

I.4 LES MODULES D'ACCES

I.4.1 INTRODUCTION

La conversion d'un algorithme LDA en un texte de programme relève d'un procédé systématique. Cependant, certains choix d'architecture physique devront être posés avant d'entamer le processus de conversion. Ces choix concernent la localisation des accès et la perception des données.

Il faut déterminer l'endroit où seront localisés les accès. L'option, qui est couramment suivie, consiste à insérer les textes des programmes issus des algorithmes LDA dans le programme d'application. Une seconde option propose que la gestion des données soit enfouie dans un module spécialisé (un module d'accès). En outre, les modules d'accès peuvent offrir au programmeur d'application une perception des données différrente de celle du SGBD.

Les deux premières sections seront consacrées à une définition du concept de module d'accès ainsi qu'à ses utilisations potentielles. La troisième section précisera la notion de module d'accès dans la démarche de conception.

I.4.2 LA NOTION DE MODULE D'ACCES

1°) DEFINITION

Un module d'accès est un SGBD virtuel. C'est pourquoi le processus de définition d'un module d'accès est calqué sur celui d'un SGBD :

1) définition des structures de données.
2) définition des primitives.
3) définition des règles d'enchaînement des primitives.

Hiérarchiquement parlant, un module d'accès occupe une position médiane entre le programme d'application et le SGBD cible.

2°) LES OBJECTIFS

Les programmes d'application sont rendus indépendants des particularismes syntaxiques et sémantiques du SGBD réel par l'entremise d'un module d'accès.

La qualité des modules d'accès est proportionnelle à leur capacité d'"information hiding". Idéalement, l'impact consécutif

à la modification d'une caractéristique quelconque du SGBD ne devrait affecter que le module d'accès. Toutefois, dans la pratique, les modules d'accès ne procurent qu'une indépendance limitée à certains aspects des données.

2.1°) L'INDEPENDANCE PHYSIQUE

Les programmes d'application sont dispensés de la connaissance des modes de représentation physique des données. En règle générale, les SGBD assurent l'indépendance physique.

2.2°) L'INDEPENDANCE PAR RAPPORT A UNE CLASSE DE SGBD

Supposons que l'on change de SGBD. Le module d'accès préserve les programmes d'application si le changement s'effectue dans le cadre d'une famille de SGBD donnée. Un module d'accès plus puissant pourrait garantir l'invariance des programmes d'application quelque soit le SGBD.

2.3°) L'INDEPENDANCE LOGIQUE

La modification du schéma conceptuel est sans conséquence pour les programmes d'application si un module d'accès établit la correspondance entre les deux versions successives du schéma.

2.4°) L'INDEPENDANCE PAR RAPPORT AU SCHEMA DES ACCES

Une modification du schéma des accès n'affecte pas les programmes d'application si un module d'accès simule les mécanismes d'accès qui ont été enlevés.

3°) L'OBJET

Un module d'accès vient étoffer les fonctionnalités du SGBD, soit en prenant en charge de nouveaux services, soit en émulant des services existants. Nous complétons ci-dessous la liste des rôles qu'un module d'accès peut remplir.

1) Un module d'accès peut limiter l'introduction anarchique de données dans la base en veillant au respect d'un certain nombre de contraintes d'intégrité.

2) Un module d'accès peut effectuer diverses mesures (comptabilisation des ressources consommées par un programme d'application, calcul du nombre des accès à un type de données,

3) Un module peut standardiser l'accès à une base de données gérée par plusieurs SGBD.

4) Un module d'accès peut émuler les structures de données du SGBD. A titre d'illustration, nous citons la manipulation des chemins N-N et des chemins multi-origines pour un SGBD CODASYL, ou encore la simulation des domaines puissances et relations pour un SGBD relationnel.

5) Un module d'accès peut renforcer les mécanismes SGBD chargés de la confidentialité des données, de la reprise sur incident et de la concurrence d'accès aux données.

Ces deux derniers points seront exemplifiés au cours de la section suivante.

I.4.3 DEUX ROLES POUR UN MODULE D'ACCES

1°) LA SECURITE DES DONNEES

Un module d'accès peut jouer un rôle non négligeable dans le domaine de la sécurité des données.

Si les données d'une base sont strictement confidentielles, des contraintes d'autorisation seront spécifiées. Le format d'une contrainte :

- tel utilisateur peut effectuer telle opération sur tel type de données.

En général, les SGBD comportent un sous-système permettant ,d'une part, d'identifier un utilisateur et ,d'autre part, de vérifier la légalité de l'accès qu'il commet.

Cependant, un intrus peut contourner ce dispositif d'autorisation. Pour s'en convaincre, prenons l'exemple d'une base de données composée de fichiers VSAM. Il est possible d'accéder aux données en recourant directement aux services du système VSAM.

Dans ce contexte, le mécanisme de "data encription" apparaît comme l'ultime moyen de protection des données. On pourrait donc construire un module d'accès incluant des algorithmes d'encription de données.

2°) UNE EMULATION RELATIONNELLE

2.1°) INTRODUCTION

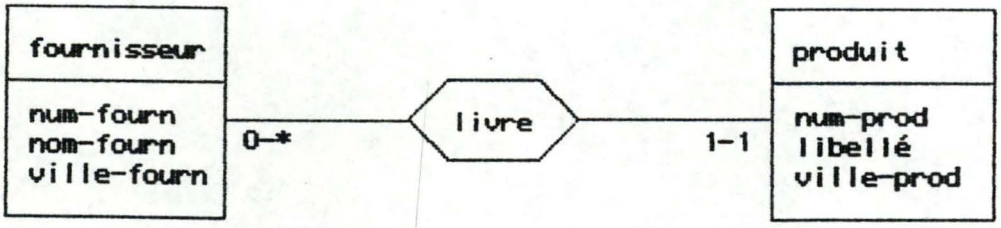
Les modules d'accès permettent parfois de superposer un modèle de données particulier à celui du SGBD physique.

Les constructeurs de SGBD proposent maintenant des produits qui vont dans ce sens. Ainsi Cullinet a commercialisé en 1983 une version étendue du SGBD CODASYL IDMS, appelée IDMS/R ("IDMS/Relational"). IDMS/R offre donc une interface liée au modèle de données CODASYL ainsi qu'une interface de nature relationnelle.

Cullinet a construit une couche spéciale au-dessus du SGBD IDMS. Cette couche est appelée " flat record view layer ". Elle implémente le mécanisme intitulé " logical record facility système (LRF) ". Cette couche est destinée au programmeur d'application, non à l'utilisateur final.

Grâce au système LRF, le programmeur n'a plus besoin de connaître la structure sous-jacente de la base de données. Le programmeur opère en termes de records logiques. Si l'on se réfère à la terminologie relationnelle, on dira qu'un record logique constitue une table virtuelle.

Supposons par exemple que l'utilisateur travaille avec une base de données "fournisseur-produit".



On pourrait utiliser LRF pour définir le record logique suivant :

FVP (num_fourn,ville,num_prod)

FVP représente le numéro de fournisseur, le numéro de produit et la ville pour les fournisseurs et les produits qui sont situés dans une même ville.

2.2°) LA MANIPULATION DES DONNEES

On discrimine deux DML :

1) le DML du programmeur d'application (U-DML).
2) le DML de l'administrateur de la base de données (D-DML).

1) Le U-DML

Parmi les opérations que le programmeur peut accomplir sur FVP, nous citons :

1) L'accès au premier (ou au suivant) record FVP qui respecte une certaine condition.

   ex. : OBTAIN FIRST FVP WHERE VILLE = 'Rome'

2) La modification du record courant FVP.

3) La suppression de l'article courant FVP.

4) L'enregistrement d'un nouveau record FVP.

2) Le D-DML

L'administrateur de la base de données écrit des DBA-procédures afin d'établir la liaison entre le programmeur d'application et la base de données IDMS. C'est ainsi qu'à chaque opération U-DML définie sur un type de records logiques, il correspond une DBA-procédure. Dans la réalité, il s'avère que les ordres D-DML immixés dans les DBA-procédures sont très proches des ordres du DML IDMS. Les DBA-procédures portent aussi le nom évocateur de "path procedures".

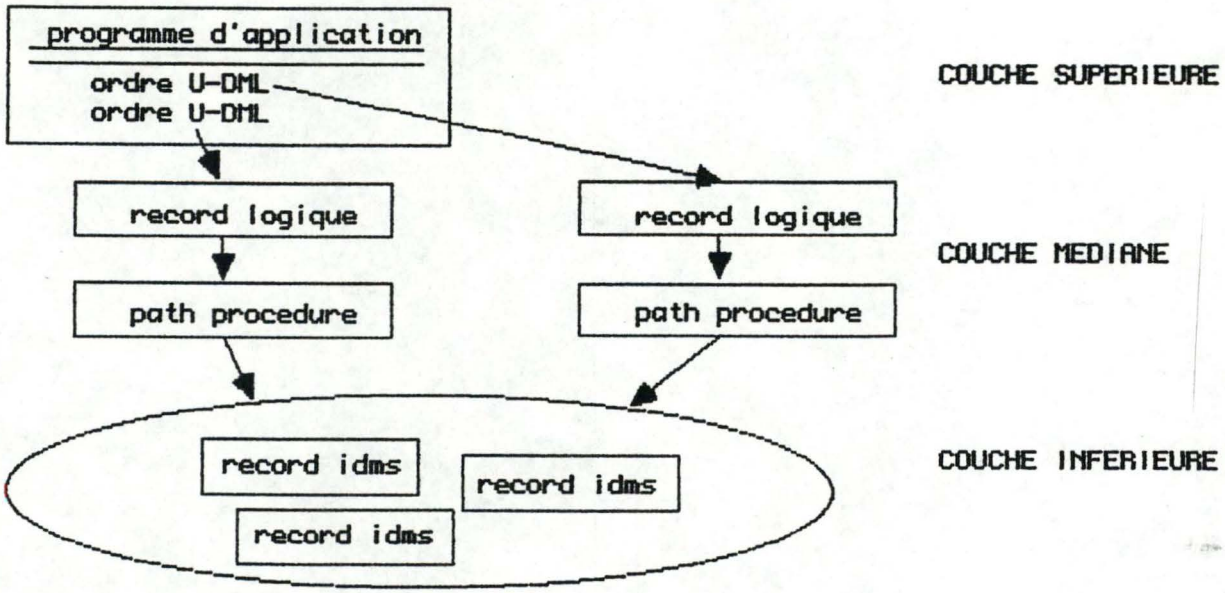


Figure 1.6

2.3°) CONCLUSION

Le système LRF présente un intérêt essentiellement théorique.

La présence d'une couche médiane constituée par les Paths procedures nous incline à faire un rapprochement avec la notion de module d'accès.

Nous sommes néanmoins obligés de constater que LRF est un succédané de SGBD relationnel.

L'indépendance

Un programme de niveau 2 est indépendant des caractéristiques syntaxiques du SGBD réel. Un SGBD de niveau 3 est indépendant du SGBD. Un programme de niveau 4 est indépendant du SGBD ainsi que des stratégies d'accès aux données.

Les architectures

"Un module d'accès peut en cacher un autre ...". L'application de ce principe permet d'envisager plusieurs architectures possibles.

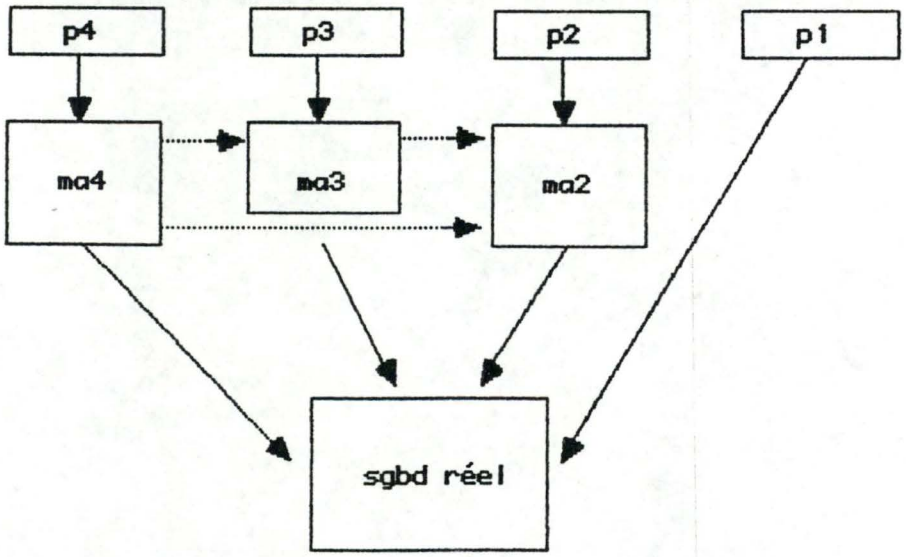


Figure 1.8: "synthèse des architectures"

Les traits discontinus représentent des architectures où les modules d'accès sont emboîtés. L'architecture (P4, MA4, MA3, MA2 , SGBD réel) reflète parfaitement la démarche de conception. Les architectures inférieures ne sont assujetties qu'aux dernières étapes de la démarche.

Des approches analogues sont poursuivies dans d'autres domaines également empreints d'une rapide évolution technique et/ou d'un manque de standard. Citons,par exemple , les accès aux périphériques et la communication entre processus distants. La disposition d'un module spécial entre le programme d'application et le complexe technique procure l'indirection nécessaire à l'indépendance vis-à-vis du système technique.

Fréquemment ce module couvre une série de modules gigognes. Chaque nouveau module qui est superposé réalise un niveau d'abstraction supérieur. Ce principe transparaît clairement dans l'architecture des modules d'accès. Il pourrait aussi prévaloir dans l'élaboration d'un système de communication entre les ordinateurs d'un réseau, d'autant plus que les membres de l'"Open Systems Interconnection" (en abrégé, l'OSI) ont institué la stratification d'un tel système.

L'architecture ISO comprend sept couches.

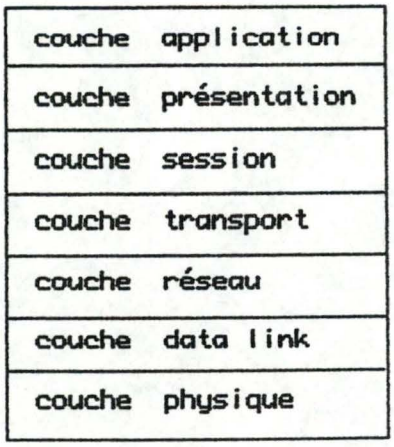


Figure 1.9 : les couches ISO

- La **couche d'application** est laissée à la discrétion de chaque site. Cette couche pourrait, comme le suggèrent les membres de l'OSI, réaliser la transparence du réseau. Les ressources du système global (les bases de données, les traitements,...) ne sont généralement pas dupliquées sur chaque site. Elles sont réparties entre les différents noeuds d'un réseau. La connaissance de la répartition physique des ressources serait alors "encapsulée" dans la **couche d'application**. Un utilisateur sur un site pourrait ainsi utiliser une ressource spécifique tout en ignorant le lieu où elle réside.

- La **couche de présentation** s'arrange pour que la structure des messages qui proviennent d'un site distant soient compatibles avec le format qui est en vigueur sur le site local.

- La **couche "session"** constitue l'interface entre un utilisateur sur un site et le réseau. Grâce à cette couche, on peut établir une connection entre deux processus distants. La connection est une phase complexe pendant laquelle les deux processus s'entendent sur les modalités de communication (communication "duplex" ou "half-duplex", débit,...).

- La **couche de transport** est chargée de transmettre un message d'un site émetteur vers un site récepteur.

- La **couche réseau** est chargée d'établir le routage des messages. Il n'existe pas toujours une ligne directe entre le site émetteur et le site récepteur. Il faut donc déterminer les noeuds par lesquels devra transiter le message.

- Les **messages sont fragmentés** avant d'être envoyés. La **couche "data link"** garantit une transmission correcte des segments d'information.

- La **couche physique** transmet des suites de bits.

conclusion

La réalisation d'un module d'accès relève d'un choix de conception physique. Au lieu d'effectuer la traduction de la solution logique, on bâtit la machine abstraite capable d'exécuter la solution logique.

I.4.4 LES MODULES D'ACCES DANS LA DEMARCHE DE CONCEPTION

Selon la démarche de conception, les traitements orientés données doivent être conçus de manière progressive (algorithme prédicatif, algorithme effectif, algorithme effectif conforme, codage de l'algorithme dans un langage de programmation).

Considérons un traitement (noté T). Le traitement T est donc associé à un algorithme prédicatif (noté A4), un algorithme effectif (noté A3), un algorithme effectif conforme (noté A2) et finalement un programme (noté P1).

Etant donné la propriété de conformité de l'algorithme A2 à un SGBD, sa conversion en un programme P1 est triviale. Il suffit de repérer les primitives DML du SGBD, capables de réaliser les accès de A2.

Faisons maintenant l'hypothèse que l'algorithme A4 est lui aussi conforme à un SGBD. Cela signifie que le SGBD en question offre les primitives suffisantes pour accéder aux éléments constitutifs des différrentes collections de données définies dans A4. Les étapes intermédiaires liées à la production des algorithmes effectif et effectif conforme ne sont donc plus nécessaires. L'algorithme A4 est directement traduisible en un programme de niveau 4 (noté P4). Il est clair qu'un tel SGBD n'existe pas à l'état naturel. Mais on peut construire un module d'accès qui lui est équivalent. Ce module d'accès transpose les appels du programme P4 en appels à un SGBD réel.

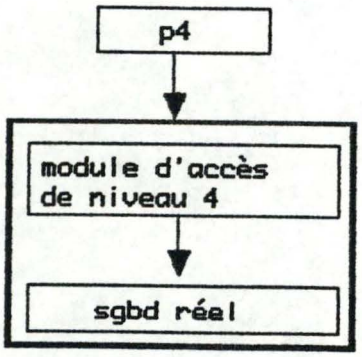


Figure 1.7 : sgbd virtuel de niveau 4

Ce procédé peut être reconduit pour les algorithmes A3 et A2. Le module d'accès de niveau 3 définit un SGBD de type MAG. Le module d'accès de niveau 2 offre les mêmes fonctionnalités que le SGBD réel. Son apport consiste à une simple mise en forme des primitives du SGBD réel.

C H A P I T R E II

L'ATELIER DE CONCEPTION DE BASES DE DONNEES

II.1 GENERALITES

Le développement d'une application pris dans le cadre d'une démarche **complète** de conception procède par **niveaux d'abstraction**.

Une phase de la démarche est le lieu d'importantes **interactions DONNEES-TRAITEMENTS** : ces interactions constituent un élément essentiel du processus de développement. La qualité de la solution "finale" dépend, en grande partie, du bon fonctionnement de ces mécanismes d'interactions.

Une définition du **suivi d'un projet informatique** serait donc de pouvoir produire aux différents niveaux d'abstraction la description d'un même objet (programme ou type de données). Les niveaux de description(1) d'un objet sont au nombre de trois : la description conceptuelle décrivant les objectifs de l'objet, la description logique où l'on spécifie l'organisation technique de la solution et la description physique qui correspond à une solution exécutable(2).

Les **types d'objets principaux** à décrire sont les unités de données et les unités de traitements. On souhaite parfois décrire aussi les écrans qui serviront au dialogue avec l'utilisateur, les messages permettant aux unités de traitements de communiquer entre elles ainsi qu'avec leur environnement, les rapports relatifs à une structure de données et/ou une structure de traitements, les utilisateurs, les ressources, etc.

Les relations que ces objets entretiennent font l'objet d'une description tout aussi importante. Par exemple, pour chaque unité de traitements, on souhaite indiquer les unités déclenchées par elle, les unités qui la déclenchent, ainsi que la ou les conditions d'activation qui caractérisent ces unités de traitements, les unités de données consommées ou produites par ces traitements, les écrans utilisés, les rapports générés, les utilisateurs impliqués, etc. Remarquons que les descriptions relatives à une application informatique sont plus ou moins étendues suivant l'utilisation souhaitée de ces dernières : documentation de l'application, construction d'une base de spécifications dans un contexte d'info-centre.

1.- ou niveaux de spécification.
2.- solution compilable par un processeur.

II.2 NOTION D'ATELIER LOGICIEL

On dénome ATELIER LOGICIEL, ou plus simplement ATELIER, un ensemble d'outils organisés autour d'une base de spécifications et qui permet de couvrir une partie importante du processus de production des applications. Un Atelier peut donc être considéré comme un ensemble intégré d'outils complémentaires.

II.2.1 LA BASE DES SPECIFICATIONS

La base des spécifications est une base de données particulière qui contient la description d'une aplication informatique c'est-à-dire la description de tous les objets composant l'application ainsi que toutes les relations qui existent entre ces objets. Elle est encore appelée **méta-base de données** puisqu'elle contient, non pas des données, mais des **méta-données**(1).

II.2.2 LES OUTILS

Un **atelier complet** peut donc être considéré comme un environnement de travail autonome supportant le **décideur** (2) tout au long du cycle de **vie** d'une application informatique : la formulation des objectifs et l'analyse du système d'information, la conception et l'implémentation de l'application, la phase d'exploitation, les extensions à apporter au système actuel et enfin, les opérations de maintenance associées à ces phases.

II.2.2.1 FORMULATION DES OBJECTIFS ET ANALYSE

Au cours de cette étape, on collecte une grande quantité de méta-données relatives à la description de la base de données en projet, aux traitements à mettre en oeuvre, aux documents existants, aux rapports à produire, aux écrans à utiliser, etc. L'Atelier permet l'introduction assistée des spécifications du Système d'Informations (S.I.), leur mémorisation, leur consultation et leur modification.

II.2.2.2 CONCEPTION DE L'APPLICATION

Les spécifications contenues dans la base de spécifications font l'objet d'une validation de cohérence. Une fois validées, elles sont évaluées sur base des critères successifs suivants :

1.- ou descriptions de données.
2.- Par exemple,l'Abmistrateur de la Base de Données dans un Atelier logiciel orienté base de données)

complétude, performances/consommations de ressources, adéquation aux besoins des utilisateurs, etc. L'Atelier IDA (1) par exemple, dispose d'interfaces vers des logiciels spécialisés permettant :

1. la validation de cohérence.

2. la génération automatique d'un programme de simulation pour évaluer le caractère réalisable du système informatique décrit.

3. la génération d'une maquette programmée du système futur pour tester le caractère effectif des spécifications.

II.2.2.4 MISE EN OEUVRE

Lors de cette phase, les spécifications relatives aux données et aux traitements sont évaluées, puis transformées jusqu'à les rendre exécutable par une machine "réelle".

II.2.2.5 EXPLOITATION

L'Atelier offre un environnement d'exploitation. Il assure les fonctions telles que la sécurité des données, la validation des requêtes selon certaines contraintes d'intégrité à respecter, des facilités de "recovery", le maintien de certaines statistiques d'utilisation, etc.

Lorsque les performances des applications se dégradent de manière trop importante, une réorganisation (2) de la base de données s'avère nécessaire. L'Atelier est adéquat pour ce genre de modifications puisqu'il permet de gérer les versions multiples d'une application.

II.2.2.6 EXTENSIONS

L'Atelier peut être utilisé lors des activités d'extensions pour déterminer l'impact d'une modification de la structure logique des données (par exemple, ajouter un nouveau type de record, ou un nouveau data item).

1.- Les objectifs assignés à l'Atelier IDA seront brièvement décrits dans les pages suivantes.
2.- Une réorganisation est une transformation qui ne modifie pas la structure logique (visible par l'utilisateur) des données.

La figure 2.1 exprime succinctement les quatre types de support fournis par l'Atelier tout au long du cycle de vie du S.I..

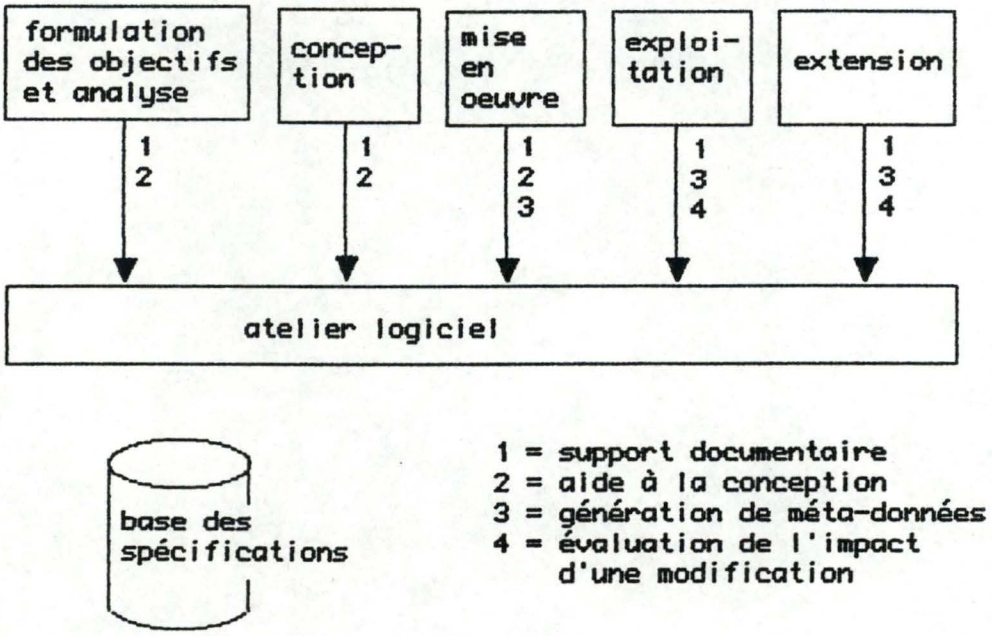


Figure 2.1

II.3 PRESENTATION DE L'ATELIER BD

L' Atelier BD est un logiciel, disponible sur PC (Personal Computer), et qui adhère à l'acception de la notion d'Atelier présentée à la section précédente. L'Atelier BD couvre la phase de mise en oeuvre de la démarche de conception d'un S.I.. En d'autres termes, il s'agit d'implémenter une application à partir d'une solution conceptuelle.

Remarquons qu'il était souhaitable de développer un outil autonome, indépendant d'un environnement particulier de conception.

Dans le cadre de ce mémoire, l'environnement de conception sera constitué par le système-logiciel IDA (1) réalisé à l'Institut d'Informatique des Facultés Universitaires de Namur, en coopération avec le projet ISDOS (2).

Nous nous permettons d'incister une fois encore, mais c'est important : l'Atelier BD pourrait être, à faible coût, adapté à un tout autre environnement.

II.3.1 IDA, UN ENVIRONNEMENT DE CONCEPTION

L'Atelier logiciel IDA -ou tout autre Atelier supportant une démarche de conception- constitue un outil presque indispensable pour tout qui veut concevoir une application informatique. On parlera donc d'une conception assistée par un système logicel. Le système logiciel IDA permet la maîtrise des différentes étapes(1) du cycle de vie d'un S.I. et contrôle donc tout le processus de développement.

Remarquons cependant que le développement lui-même ne doit pas être centralisé à la seule station de travail(3) supportant IDA. Des considérations technico-économiques et d'ordre organisationnelles (par exemple, la compétence respective des personnes qui participent à l'élaboration du S.I.) sont à l'origine de la constitution de stations de travail secondaires. L' Atelier BD est l'une d'elles et se charge exclusivement de la mise en oeuvre du S.I..

1.- F. Bodart & Y. Pigneur, Conception des applications informatiques, MASSON, 1983, p 94.
2.- Le projet ISDOS est développé à l'université de Michighan sous la direction du professeur TEICHROEW.
3.- La station de travail est un mainframe de configuration assez puissante.

La station de travail **Atelier** BD dispose, ou plutôt disposera dans un avenir proche, de puissantes possibilités graphiques (écran graphique à haute résolution et si possible couleur, travail en mode souris, etc.).

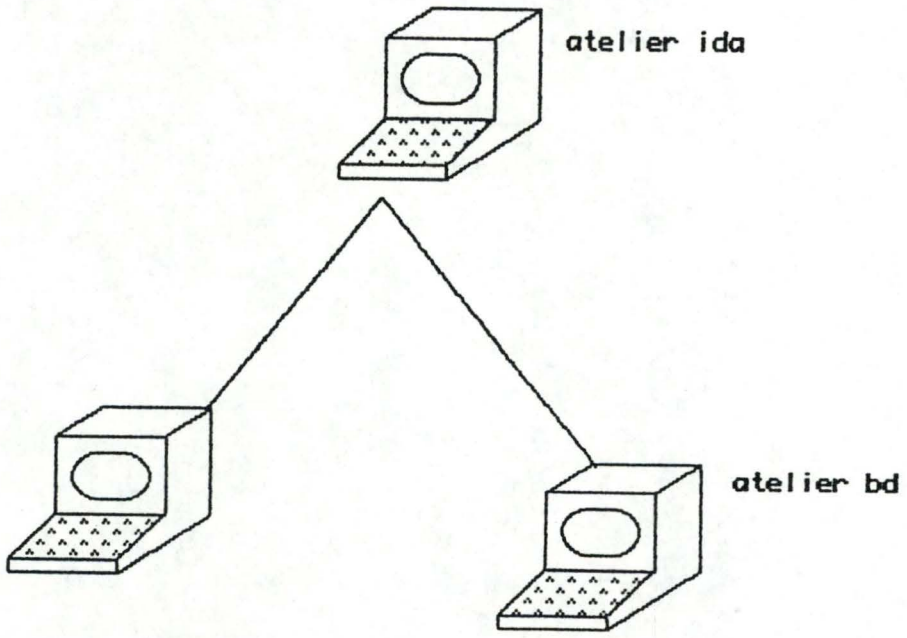


Figure 2.2

Comme l'illustre la figure 2.2, la station Atelier BD fonctionne en connection avec la station principale. Les seules spécifications nécessaires à la mise en oeuvre sont transmises à la base de spécifications locale. Une fois la solution exécutable(1) obtenue, elle est intégrée dans la base de spécifications centrale de IDA.

II.3.2 LES OBJECTIFS ASSIGNES A L'ATELIER BD

L'élaboration d'une base de données physiques, pris dans le cadre du développement d'une application informatique, était jusqu'il y a quelques années à ce point complexe que rien ne permettait de présager de la qualité de la base de données construite. Les démarches étaient alors trop pauvres, se limitant à appliquer quelques principes solides mais trop isolés pour résister à la critique. Une Conception Assistée par Ordinateur (CAO) était alors souhaitable. D'autant plus que les étapes de conception étaient pour la plupart de nature semi-automatique et exigeant une rigueur toute spéciale.

1.- solution compilable par un processeur.

L'**Atelier BD** est une telle station de travail(1) offrant, au travers d'un ensemble d'outils puissants et ergonomiques, un environement tout à fait privilégié. Cet Atelier est un réel support à la décision au sens où il prescrit une démarche de conception sans pour autant vouloir supplanter le concepteur. On permet simplement au concepteur d'employer sa compétence là où elle est la plus efficace c-à-d au niveau des choix. Les décisions ne peuvent qu'en être améliorées car s'exerçant à un niveau plus élevé. De meilleurs schémas devraient en résulter puisque les options sont prises au sein d'une démarche et que donc elles doivent pouvoir être justifiées.

II.3.3 CONSTITUTION DE LA BASE DE SPECIFICATIONS LOCALE

La constitution de la base de spécifications de l' Atelier BD revient à modéliser en un **schéma conceptuel général**, encore appelé méta-schéma, la description d'un S.I. limité aux seuls aspects DONNEES et TRAITEMENTS.

Le **méta-schéma** sera généralement obtenu par consolidation -ou "intégration"- des sous-schémas associés à chacune des dimensions (DONNEES, TRAITEMENTS). L'**approche BOTTOM - UP** ainsi proposée est, sans plus, une proposition méthodologique adressée au concepteur. Remarquons cependant, que la phase de consolidation ne permettra pas d'obtenir un **schéma global** par simple juxtaposition des différents sous-schémas. Premièrement, les deux dimensions étant très étroitement correlées, des conflits de représentation devront être résolus. En général, la solution la plus générale sera retenue. Deuxièmement, on souhaite également modéliser les relations existant entre les structures de données et les structures de traitements.

L'élaboration d'un sous-schéma (sous-schéma des DONNEES, sous-schéma des TRAITEMENTS) pose inévitablement le problème de l' **abstraction** vis-à-vis d'une situation particulière.

Il s'agit en fait de réaliser une analyse au second degré, au sens où un schéma particulier obtenu au premier degré d'analyse doit pouvoir être supporté par le méta-schéma (ou modèle général).

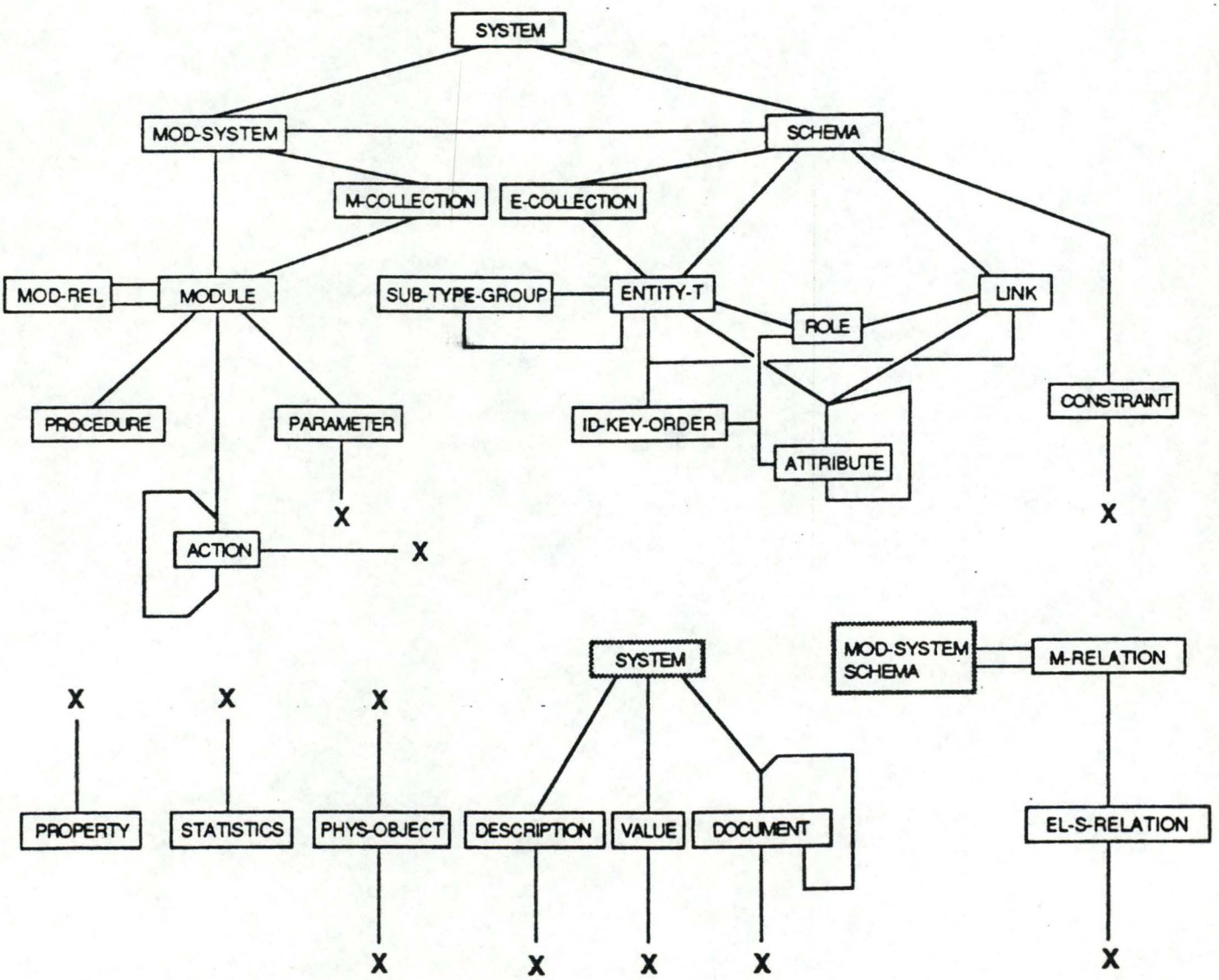
Dans un second temps, une approche **TOP - DOWN** sera utilisée pour affiner les concepts généraux retenus.

1.- L' Atelier BD est une partie autonome de l' atelier logiciel de développement d'applications informatiques IDA.

La figure 2.3 illustre les principaux concepts du sous-schéma **schéma des données**. Seuls les principaux concepts relatifs à la structuration des données sont repris. Pour plus de précisions, le lecteur intéressé pourra consulter la référence(1).

La modélisation des traitements est discutée en annexe. Le **sous-schéma système de modules** y est présenté.

1.- J.L. Hainaut, document SPEC-86/7-1, "Schémas de la Base des Spécifications", deuxième version.



II.3.4 FONCTIONS ACTUELLES DE L' ATELIER BD

Les principales fonctions de l'Atelier concernent l'initialisation de la base de spécifications avec un schéma de base de données en projet, la consultation du schéma, la modification du schéma, la transformation de structures du schéma, la vérification de la conformité du schéma à un standard, la génération DIL correspondant au schéma, la production de rapports. Ces fonctions seront évoquées au chapitre VI lors de la génération de base de données MDBS.

D'autres fonctions sont actuellement en développement. Elles concernent les tranformations d'algorithme consécutives à une transformation de schéma.

II.3.5 SCENARIO DANS L' ATELIER BD

Si l'on accepte les contraintes évidentes (on ne peut transformer qu'un schéma ayant fait l'objet d'une initialisation, on ne peut produire un texte MDBS que si toutes les structures du schéma sont conformes MDBS, etc.), les différentes fonctions pourront être activée dans un ordre quelconque, selon la logique de conception du concepteur lui-même et l'écart de conformité à annuler. La figure 2.4 illustre les circuits possibles d'activation des fonctions.

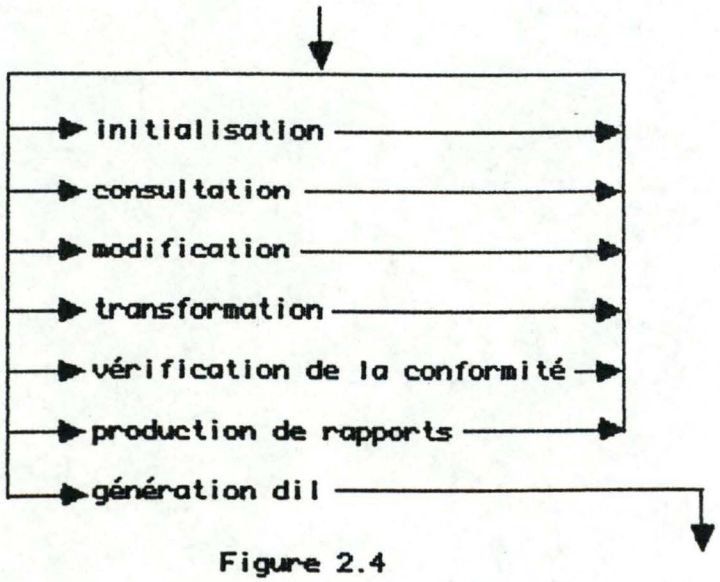


Figure 2.4

II.4 ARCHITECTURE DE L'ATELIER BD

II.4.1 INTRODUCTION

Un certain nombre de principes clefs de software engineering ont prévalu lors de l'établissement de l'architecture de l'Atelier.

1°) INTEGRATION D'OUTILS AUTONOMES.

L' Atelier BD ne repose pas sur une structure complexe de composants interconnectés. Les outils s'ignorent. Ils sont cependant intégrés dans la mesure où ils accèdent à une structure de données communes : la base des spécifications. En fait, ces outils sont autant de programmes d'application développés autour d'une méta-base de données.

2°) INFORMATION HIDING

LES ABSTRACTIONS

Ces dernières années, le mot **abstraction** est devenu un des "buzzwords" les plus répandus dans le champ informatique.

Pour D. Parnas, une abstraction est susceptible de représenter plusieurs objets réels, dans la mesure où seuls leurs traits communs constituent l'abstraction. Si un modèle -ou système abstrait- ne reprend que les aspects importants de la réalité à décrire, il est clair qu'étudier le modèle est probablement plus simple. L'idée est donc de pouvoir appréhender un système réel en se rapportant au seul système abstrait (c'est-à-dire libéré de tous les aspects non essentiels).

LES INTERFACES

La définition d'un interface entre deux composants n'est pas réduite à la description du format des informations échangées.

Plus fondamentalement, il s'agit d'énoncer toutes les hypothèses qu'un composant fait vis-à-vis de son homologue et vice-versa. Il est possible de passer outre la description des informations échangées pour autant que l'on ait défini un jeu de primitives capables de consulter et d'insérer de l'information.

LES INTERFACES ABSTRAITS

Un interface abstrait modélise les propriétés communes d'un ensemble d'interfaces.

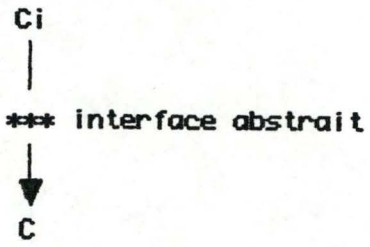


Figure 2.5

L'interface abstrait permet l'indépendance des modules de haut niveau vis-à-vis des versions successives des modules inférieurs.

Chaque interface abstrait constitue un "tool-kit" spécifique qui peut être utilisé sur base de ses seules spécifications externes. Les interfaces abstraits concourent à obtenir un maximum de flexibilité au niveau de l'architecture. La maintenance en sera d'autant plus aisée.

II.4.2 LES << TOOL-KITS >>

Les "tool-kits" se répartissent en deux classes. La première classe traite des types de données complexes tandis que la seconde classe traite des échanges avec l'environnement.

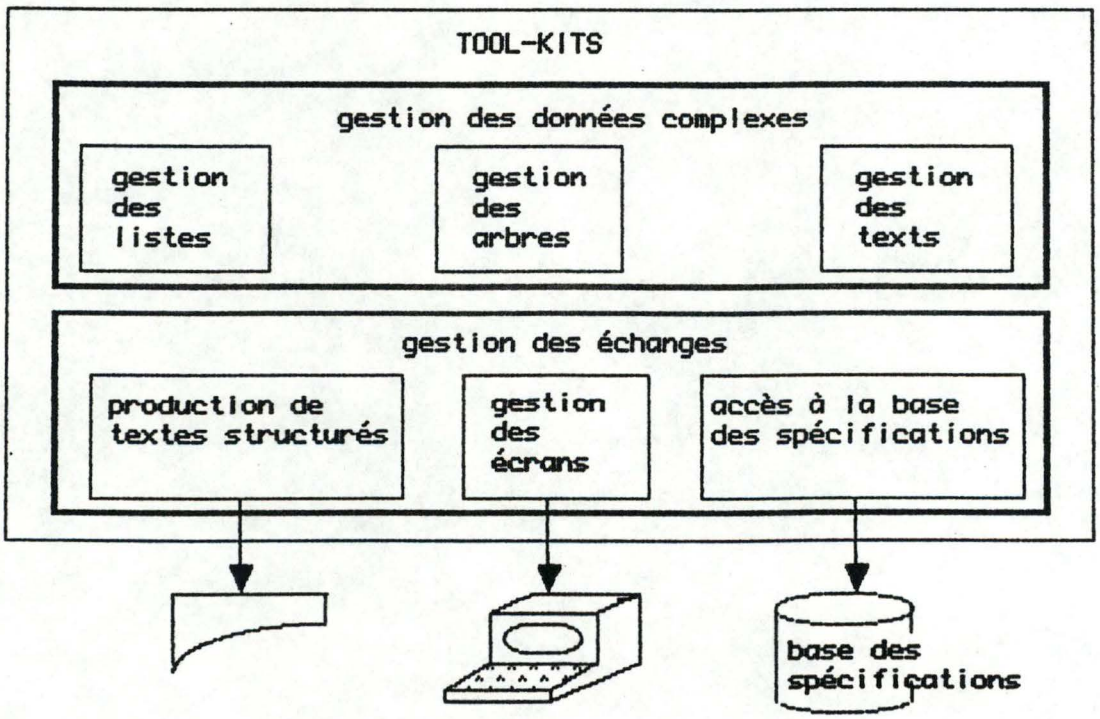


Figure 2.6

II.4.2.1 LES OUTILS DE GESTION DES DONNEES COMPLEXES

Le programmeur de l'Atelier peut disposer de trois types abstraits de données (les LISTES, les ARBRES et les TEXTS). Un ensemble de primitives sont mises à la disposition du programmeur pour gérer ces objets complexes.

Avant d'entamer une description plus précise, nous signalons deux caractéristiques communes à ces objets complexes :

1. Tout objet complexe est statique. La durée de vie d'un objet complexe est liée au programme d'application.

2- Tout objet complexe est **auto-descriptif**. Citons, à titre exemplatif :

   - le nombre maximum d'éléments dans un objet complexe.
   - le nombre courant d'éléments dans un objet complexe.
   - un pointeur vers un élément courant.
   - le diagnostic de la dernière opération accomplie.
   - ...

Les LISTES

Les LISTES sont conçues pour stocker des suites variable d'objets C.

Les ARBRES

Les ARBRES permettent de stocker un nombre variable d'objets selon une structure arborescente.

Les TEXTS

Les TEXTS sont des objets pouvant contenir un nombre variable de caractères.

De façon générique, les primitives concernent l'initialisation d'un objet complexe, le positionnement dans un objet complexe, l'accès aux constituants d'un élément appartenant à un objet complexe, la migration d'éléments à l'intérieur d'un objet complexe et la suppression d'éléments.

II.4.2.2 LES OUTILS DE GESTION DES ECHANGES

Grâce à ces outils, le programmeur d'application peut communiquer avec son environnement extérieur. Cet environnement se compose de la mémoire secondaire, du terminal et de l'imprimante.

1) LE COMPOSANT << ACCES A LA BASE DES SPECIFICATIONS >>

Le modèle des données sous-jacent au composant ACCES A LA BASE DES SPECIFICATIONS est un sous-ensemble du modèle MAG. Nous évoquerons ce modèle par le terme MAGRESTREINT.

1- Seuls les types de chemins 1-N et N-1 sont admis.
2- Chaque type de chemins est toujours doté de son inverse.
3- Un type de chemins 1-N peut être multi-origine mais pas multi-cibles.
4- Les items répétitifs ou décomposables ne sont pas admis.
5- Un item peut être identifiant dans un type de chemins (un seul par type de chemins).
6- Un item peut être clé d'accès dans un type de chemins (un seul par type de chemins).

Le programmeur d'application accède à la base de données via les interfaces ADLC et SEM. Ces interfaces recouvrent différrentes classes d'opérations :

1- ouverture et fermeture d'une base de données.
2- accès séquentiel.
3- accès par chemin.
4- accès par clé dans un chemin.
5- accès direct.
6- création et suppression d'un article ;
   modification de valeurs d'items d'un article.
7- insertion, retrait et transfert d'un article dans un chemin.

Les modules d'accès ADLC et SEM sont des couches "orientées programmeur" dans la mesure où le schéma des données MAGRESTREINT est fort proche d'un schéma CODASYL.

L'interface ADLC est le plus amical vis-à-vis d'un programmeur. Il offre, en effet, une primitive spécifique par opération spécifiée alors que SEM n'offrira qu'un point d'entrée unique.

2) LE COMPOSANT << GESTIONNAIRE D'ECRANS >>

Le gestionnaire d'écrans procure toute une gamme de primitives aux programmes d'application qui dialoguent avec l'utilisateur. Le gestionnaire d'écrans est responsable de l'affichage et de la saisie de données dans des écrans préalablement définis. Le programme d'application est soustrait des particularités physiques du terminal et des protocoles de dialogue.

3) LE COMPOSANT << PRODUCTION DE TEXTES STRUCTURES >>

Un rapport est la mise en forme d'un contenu (texte) en vue d'une impression sur un support (écran, imprimante,...). Une série de primitives ont été mises au point pour faciliter l'élaboration d'un rapport. Ces primitives se répartissent en deux classes. Les premières primitives précisent le format des pages constitutives d'un rapport.

1- définition d'un en-tête de page.
2- définition d'un pied de page.
3- définition des marges.
4- ...

Les primitives de la seconde classe concernent les fonctions d'écriture.

1- écriture d'un string à tel endroit du rapport.
2- saut de page.
3- ...

CHAPITRE III

LES OBJECTIFS DU MEMOIRE

Le mémoire est une double contribution à l'Atelier de conception de base de données présenté au chapitre II.

Il s'agit d'abord d'instancier l'Atelier général à la conception de base de données MDBS.

Le second objectif est d'intégrer MDBS comme support de l'Atelier BD. La définition et la réalisation du module d'accès ADLC ont permis d'assurer vis-à-vis du nouveau SGBD l'indépendance indispensable aux outils de l'Atelier.

III.1 MDBS, SCHEMA CIBLE DANS L'ATELIER BD

Il s'agit d'étendre l'Atelier BD au SGBD MDBS de façon à pouvoir générer un schéma source MDBS.

L'environnement tout entier d'une base de données est analysé et étudié. Les composants de l'environnement sont démontés puis reconstruits pièce par pièce, pour permettre ainsi d'aborder les mécanismes fondamentaux.

L'environnement d'une base de données est d'autant plus complexe qu'il faut toujours pouvoir assurer la conception, l'exploitation et la maintenance de cette dernière. On distinguera notamment les requêtes interactives(1), les requêtes issues de programmes d'application, le module d'accès, le SGBD MDBS, l'Atelier BD et finalement un organe de décision.

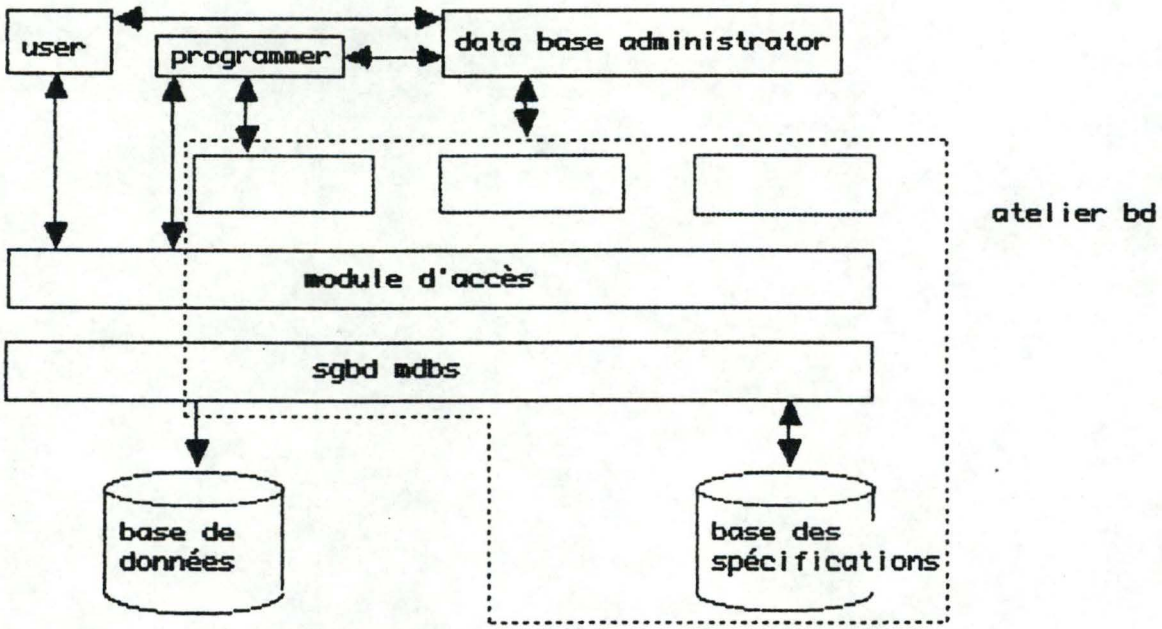


Figure 3.1 : environnement opérationnel d'une base de données

Si un langage interactif est disponible, chaque requête introduite au terminal est analysée, vérifiée puis exécutée.

Les commandes issues d'un programme d'application sont via le module d'accès vérifiées puis exécutées. Les résultats de la commande exécutée sont envoyés au programme demandeur.

1.- requêtes L4G.

L'**Atelier BD** permet de construire progressivement la base de données physique de l'application informatique développée. Les résultats successifs seront dans l'ordre la production d'un schéma conforme à l'Atelier à partir d'une description conceptuelle, la production dans l'Atelier d'un schéma conforme à un SGBD cible et finalement la génération du schéma source MDBS. Des transformations de l'Atelier permettent d'établir la conformité d'un schéma relativement à un ensemble de règles dites de conformité. Ce processus de transformation des structures de données induit généralement la transformation des algorithmes travaillant sur ces données. L'environnement d'exploitation nécessaire au module d'accès sera généré automatiquement à partir de l'Atelier.

L'**organe de décision(1)** joue un rôle essentiel. Il est chargé d'utiliser l'Atelier BD à bon escient et est donc à la base de toute une série de choix de nature fort différente. De ces choix dépendront la qualité de la base de données future.

Le SGBD MDBS isole les autres composants des détails techniques de stockage des données. Les accès seront généralement de type navigationel, plutôt que sur base d'une valeur de clé.

Le **module d'accès** est une couche construite sur le SGBD MDBS. Il isole ainsi les programmes d'application de la syntaxe d'appel particulière des ordres du SGBD. Cette indépendance PROGRAMME D'APPLICATION / ACCES AUX DONNEES garantit que les opérations associées à un changement de SGBD n'entraîneront pas des coûts considérables(2) pour maintenir les programmes.

1.- "Data Base Administrator" ou DBA.
2.- temps de programmeur.

III.2 MDBS COMME SGBD DE L'ATELIER

On propose de remplacer, pour diverses raisons, le SGBD artisanal(1) de l'Atelier par le SGBD commercial MDBS. Cette perspective était formulée, dans un premier temps au moins, sous la forme d'une simple proposition sujette à investigation. Il fallait établir dans quelle mesure MDBS répondrait aux besoins de l'Atelier. Ces avantages étaient attendus à plusieurs niveaux. On citera : performances (temps CPU, espace mémoire), portabilité, maintenance assurée,... Les premières évaluations assez favorables permettent de supposer que tout sera mis en oeuvre pour intégrer MDBS à l'architecture de l'Atelier. La transition du SGBD actuel vers MDBS devrait être sans douleur. Les outils de l'Atelier accèdent à la base des spécifications (ou la méta-base) via le module d'accès ADLC. Ce module réalise la plupart des fonctions offertes par un SGBD, et est défini de manière indépendante par rapport à toute technologie particulière (MDBS, SGBD actuel,...). Seule l'implémentation du module d'accès épaulant ces SGBD est spécifique et donc à entreprendre.

1.- prototype de SGBD réalisé à l'Institut d'Informatique, Namur.

CHAPITRE IV

LES SGBD DE TYPE RESEAU

Nous nous limiterons dans cette section à l'analyse des structures de données en réseau. Il s'agit en quelques lignes d'aborder les caractéristiques générales des modèles en réseau sans restriction à une implémentation particulière (MDBS,...). De manière plus générale, on parlera de la notion de standard en matière de base de données (CODASYL,...).

Rappelons qu'un modèle de données bien que composant sous-jacent à un SGBD, n'en est pas moins autonome. Il est d'ailleurs des modèles de données totalement indépendants d'une technologie de gestion particulière : ex. modèle Entité/Association,... La démarche de conception d'une base de données présentée au chapitre 2 est très claire à ce sujet. La modélisation des données au niveau conceptuel est première. Le choix d'une technologie de gestion de données (SGBD) se fait ultérieurement.

Les primitives de manipulation de ces SGBD seront brièvement présentées.

IV.1 GENERALITES

Les deux aspects fondamentaux d'un "outil de manipulation de données complexes"(1) sont les facilités qu'il offre pour structurer et manipuler des données.

La **puissance**(2) d'un modèle de données est généralement définie par sa capacité à fournir une **représentation directe** de la situation à modéliser. La structure logique d'une base de données de type réseau est spècifiée en termes de DATA ITEMS, RECORD-TYPES, et de SET-TYPES. Des facilités de structuration (par exemple, les SETS M-N) offertes par certains modèles de données (MDBS,...) font généralement défaut. Une **représentation indirecte** est alors nécessaire pour déjouer ce manque de puissance : définition d'un type de RECORD artificiel et de deux types de SET 1-N.

De telles considérations et d'autres sont à l'origine d'une classification des SGBD de types réseau disponibles sur le marché.

On distingue classiquement trois types de modèles de données en réseau : les modèles de données en réseau "limité" qui divisent les types de RECORDS en 2 ensembles (les types de RECORDS MAITRE et les types de RECORDS DETAIL), le modèle de données en réseau "simple" qui ne permet pas les types de SETS M-N, le modèle de données en réseau "complexe" qui admet les types de SETS M-N.

Dans une structure en réseau limité (TOTAL, IMAGE), les relations entre objets sont représentées via des relations 1-N "MAITRE-DETAIL" entre deux types de RECORD. Un type de RECORD ne peut être à la fois MAITRE et DETAIL.

Dans une approche des données de type CODASYL (IDMS, IDS-2), les relations sont représentées par des relations 1-N "OWNER-MEMBER" entre deux types de RECORD. Le réseau de type CODASYL ne souffre pas de la restriction invoquée pour les réseaux limités. Il ne permet cependant pas les types de SET M-N.

1.- Système de Gestion de Base de Données
2.- C.W. Holsapple & A.B. Whinston, Data Base Management : Theory and Applications, 1983, p 175-191.

Le réseau étendu (par exemple MDBS) est une innovation récente en matière de base de données. Suivant cette approche, le concepteur d'application n'est plus limité à un modèle particulier de structuration des données (relationnel, hiérarchique, réseau). Dans un réseau étendu, il est possible de spécifier une structure de données quelconque de l'un de ces modèles. Cette approche générale permet également la définition de relations M-N (appelé SETS M-N) et constitue à ce titre une alternative aux techniques de redondance et de création d'un type de RECORD artificiel. Remarquons que cette approche facilite la traduction modèle sémantique/modèle du SGBD. De plus, on attendra de meilleurs performances de stockage et d'accès.

Bien que MBDS ne soit pas conforme au standard CODASYL, on peut dire cependant qu'il y a **compatibilité** en ce sens que MBDS est en toute généralité une extention de CODASYL. Ainsi par exemple, un SET CODASYL est aussi un SET MBDS. L'inverse n'est pas vrai.

IV.2 LA NOTION DE STANDARD

Les rapports successifs de CODASYL publiés en 1969 sous une forme préliminaire, puis en 1971, 1978 et 1981 sous des formes de mises-à-jour, émettent des recommandations quant aux langages de définitions (DDL), et de manipulation (DML) de données.

Les rapports successifs témoignent au travers de leurs recommandations d'une part de l'imprécision et de l'ambiguïté de ces dernières, et d'autre part d'une volonté pressante d'établir des normes.

L'American National Standards Institute (ANSI) a repris les résultats de CODASYL (principalement ceux de 1978) en vue de définir des standards en matière de base de données.

En 1978, ANSI diffuse un cadre de référence pour l'architecture des SGBD. C'est en mai 1985 que la DAFTG du groupe d'étude ANSI/X3/SPARC proprose un modèle de référence pour la standardisation des SGBD. Le modèle d'architecture identifie et spécifie les composants fonctionnels d'un SGBD, les différentes classes d'utilisateurs, ainsi que les interfaces entre composants et entre composants et utilisateurs.

Cette notion de conformité à un standard est plus importante qu'elle n'y paraît. La plupart des SGBD installés s'inspirent de standards telles les recommandations émanant de CODASYL ou de l'ANSI. Nous citerons en exemple :

- IDMS de CULLINET
- DBMS-10, DBMS-20, DBMS-11 de DEC
- IDS-2 d'Honeywell-Bull
- PHOLLAS de PHILIPS

D'autres SGBD offrent également des structures de données en réseau sans faire directement référence aux recommandations CODASYL. Citons par exemple :

- MDBS-3 de MICRO DATABASE SYSTEMS
- TOTAL de CINCOM
- IMAGE de HEWLETT-PLACKART
- ADABAS de SOFTWARE AG

Remarquons finalement que la hiérarchie à trois niveaux du modèle de référence ANSI/SPARC sert de base à de nombreuses méthodes de conception de base de données.

IV.3 PRINCIPES GENERAUX DES SGBD CODASYL 71(1)

LES NIVEAUX DE DESCRIPTION DES DONNEES

Les SGBD CODASYL reposent sur une description des données s'articulant en 3 niveaux : la DESCRIPTION LOGIQUE, la DESCRIPTION PHYSIQUE et la DESCRIPTION EXTERNE.

La DESCRIPTION LOGIQUE est obtenue à partir d'une description conceptuelle et décrit les structures de données selon les concepts habituels des SGBD. L'apport majeur d'une description logique est de soustraire le programmeur d'applications des questions de représentation physiques des données de la base. Dans de telles spécifications, les paramètres physiques devraient être ignorés au maximum.

Les spécifications de paramètres physiques des structures de données et de leur usage font l'objet d'une DESCRIPTION PHYSIQUE. Elles ne s'adressent pas aux programmeurs, mais bien aux spécialistes techniques de la base de données.

La DESCRIPTION EXTERNE permet d'adapter la perception des types de données aux besoins d'une classe d'utilisateurs. De telles spécifications permettent de définir des interfaces utilisateurs "userfriendly".

LANGAGES DE DEFINITION DE DONNEES ET SCHEMAS

Toute description (logique, physique, externe) d'une base de données doit être communiqué au SGBD. Elle s'exprimera dans un langage adéquat appelé SCHEMA Data Description Language (ou SCHEMA DDL) pour la description logique, SUBSCHEMA DDL pour les descriptions externes, et DEVICE MEDIA CONTROL LANGUAGE (DMCL) pour la description physique. Rédigée dans un tel langage, une description prend la forme d'un SCHEMA. Le schéma logique contient la spécification des AREAS, des RECORD-TYPES, des DATA-ITEMS, des SET-TYPES, ainsi que des caractéristiques annexes de ces concepts (clé d'accès, identifiant, redondances contrôlées, procédures bases de données). A un instant donné, on peut associer à une base de données un seul SCHEMA (logique), un seul SCHEMA (physique), et un nombre quelconque de SUBSCHEMA (schéma externe).

1.- J.L. Hainanut, Introduction aux Systèmes de Gestion de Données CODASYL 71, Octobre 85.

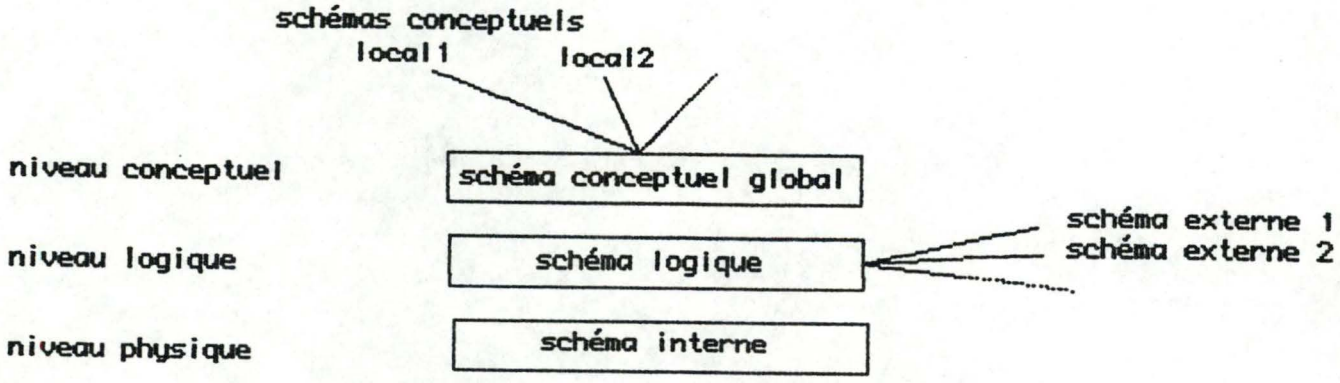


Figure 4.1 : les descriptions de données vues au travers de la hiérarchie des niveaux de conception

Les schémas d'une base de données sont traités par un composant du SGBD appelé compilateur DDL. Ce dernier vérifie la validité des textes, et initialise une base de données qui ne contient encore aucune donnée, mais une version condensée des différents schémas qui seront utilisés lors de la manipulation des données.

LANGAGES DE MANIPULATION DE DONNEES ET PROGRAMMES D'APPLICATION

Le mode principal de manipulation de données offert par les SGBD CODASYL est celui qui fait intervenir un programme d'application. Ce dernier rédigé dans un langage traditionnel (COBOL, FORTRAN,...) contient des ordres d'accès aux données et de modification de celles-ci.

Les ordres sont exprimés dans un langage approprié, appelé Langage de Manipulation de Données (ou LMD), qui constitue une extension du langage d'application habituel.

STRUCTURE LOGIQUE DES DONNEES

La notion de structure logique des données couvre l'ensemble des concepts logiques c'est-à-dire tels que vus par un programmeur d'application.

1.- J.L. Hainaut, Conception assistée des applications informatiques. 2. Conception de la base de données, MASSON, 1986.

Une base de données (DATA BASE) est constituée de fichiers (AREA) dans lesquels sont rangés des articles (RECORD) appartenant chacun à un type (RECORD-TYPE). Il existe un article particulier appelé SYSTEM. A chaque article est automatiquement associé un identifiant interne (la DATABASE-KEY ou DBKEY), constant durant la vie de l'article.

A tous les articles d'un même type sont associés des valeurs de DATA ITEMS. Ces DATA ITEMS sont caractéristiques du RECORD-TYPE. Les valeurs d'un DATA ITEM sont d'un type (numérique, caractère, etc.) Un DATA ITEM peut être décomposable; il peut aussi être répétitif.

Entre les RECORD-TYPES peuvent être définis des types d'association (SET-TYPE). Une association, ou SET, comprend un article jouant le rôle de propriétaire (OWNER) et un nombre quelconque (y compris aucun) d'articles appelés membres (MEMBER) du SET. Un propriétaire ou un membre ne peut l'être que d'un seul SET d'un type déterminé (il s'agit donc d'associations du type 1-N). Les membres des SETS d'un type seront triés selon une clé de tri, ou rangés par ordre d'insertion, ou encore selon un ordre décidé par le programmeur.

On peut déclarer un DATA ITEM (ou groupe de -) identifiant d'un RECORD-TYPE. On peut aussi déclarer un nombre quelconque d'identifiants pour les membres de chaque SET. Les clés peuvent être (mais ne sont pas nécessairement) identifiantes.

STRUCTURE PHYSIQUE DES DONNEES

La spécification des paramètres physiques ne concerne que l'**administrateur de la base de données**. Le rapport CODASYL (DBTG 71) ne spécificie pas de langage de spécification des paramètres physiques des données. Chaque SGBD utilise ses propres conventions.

Les paramètres physiques définissent la liaison avec les fichiers du système d'exploitation, la taille des fichiers et de leurs pages, le mode d'implémentation des clés d'accès, les taux de remplissage, la taille des tampons, le mode de protection en cas d'incendie, etc. On étudie successivement la spécification du mode de stockage des RECORDS, le mode d'implémentation des SETS, et la représentation des ITEMS redondants. Pour plus de précisions, on consultera sur ce point la référence (HAI,COD 71).

1.- J.L. Hainaut, Introduction aux Systèmes de Gestion de Bases de Données CODASYL 71, Notes de cours, 1985.

LES STRUCTURES EXTERNES DES DONNEES

La structure externe des données présentée au programmeur d'application est constituée des AREAS, RECORD-TYPES, SET-TYPES et DATA ITEMS impliqués dans l'application qu'il développe. Ces objets pourront -d'une application à l'autre- changer de nom, d'ordre (pour les DATA ITEMS) et parfois de définition (DATA ITEMS)

ACCES AUX DONNEES

Un SGBD CODASYL offre un jeu de primitives permettant d'accéder aux données d'une Base de Données. Chaque primitive est commandée par une instruction, qui est une phrase du language DML. L'exécution d'une primitive d'accès entraine la mise à disposition d'un article à la fois(1). Ainsi, pour obtenir les membres d'un SET, on exécutera la primitive d'accès aux membres autant de fois que ce SET compte de membres. Le langage DML offre principalement les primitives suivantes :

-ouverture, fermeture d'une AREA (OPEN, CLOSE);
-accès aux articles d'une AREA (FIND);
-accès aux membres d'un SET dont on connait le propriétaire (FIND);
-accès au propriétaire d'un SET dont on connait un membre (FIND);
-accès aux articles d'une AREA par clé d'accès (FIND);
-accès aux membres d'un SET par clè d'accès (FIND);
-accès a l'article de DBKEY déterminée (FIND);
-obtention des valeurs de DATA ITEMS d'un article auquel on a accédé (GET);

MODIFICATION DES DONNEES

Le langage DML offre également les primitives permettant de mettre-à-jour des données de base. Notons qu'il n'est généralement pas possible de modifier le schéma d'une base de données via le DML comme cela se fait dans certains SGBD. Les principales primitives concernent :

- la création d'un article (STORE);
- la modification de valeurs de DATA ITEMS d'un article (MODIFY);
- la suppression d'un article, et éventuellement des menbres de certains SETS dont il est propriétaire (DELETE);
- retirer un article comme membre d'un SET (REMOVE);
- insérer un article comme membre d'un SET (INSERT);
- transférer un article d'un SET vers un autre (MODIFY).

1.- accès ponctuel.

SCHEMA LOGIQUE D'UNE BASE DE DONNEES

Le SCHEMA LOGIQUE définit les structures logiques d'une base de données. Il contient donc la spécification des AREAS, des RECORD-TYPES, des DATA ITEMS, des SET-TYPES ainsi que des caractéristiques annexes de ces concepts.

Dans ce même schéma, certains concepts logiques seront complétés de considérations physiques -ou paramètres physiques-. Citons par exemple :

- le mode de stockage des RECORDS d'un type
- le mode de réalisation des SETS d'un type
- la représentation des DATA ITEMS redondants

Cette spécification d'ordre physique, et située à un niveau logique, est regrettable. Les recommandations CODASYL qui suivirent (1978, 1981, ...) apportent d'ailleurs certains amendements à ce propos.

Les caractéristiques les plus difficiles, ou encore utiles dans les chapitres à venir, seront évoquées. Nous parlerons notamment de la notion de SET et de ses caractéristiques connexes, ainsi que des modes de stockage.

1. Notion de SET

Un SET CODASYL contient un record particulier qui joue le rôle d'OWNER, et un nombre quelconque (y compris aucun) de records jouant le rôle de MEMBERS. Tout SET appartient à un type appelé SET-TYPE. Les OWNERS des SETS d'un SET-TYPE appartiennent à un seul type, tandis que les MEMBERS peuvent appartenir à plusieurs types. L'OWNER d'un SET est d'un type différent de ceux des MEMBERS. Un RECORD ne peut appartenir à plus d'un SET d'un SET-TYPE déterminé. On en déduit qu'à chaque record du type OWNER sont associés 0, 1 ou plusieurs articles MEMBERS, tandis qu'à chaque RECORD du type MEMBER correspond au plus un RECORD du type OWNER, ou encore qu'à un SET-TYPE correspond un type d'association 1-N du type OWNER vers le type MEMBER.

CODASYL permet de définir des SETS de caractéristiques différentes :

SET MONO-OWNER
Les OWNERS des SETS d'un SET-TYPE appartiennent à un seul type.

SET MONO-MEMBER
Les MEMBERS des SETS d'un SET-TYPE appartiennent à un seul type.

SET MULTI-MEMBER
Les MEMBERS des SETS d'un SET-TYPE peuvent appartenir à plusieurs types.

SET D'OWNER SYSTEM
Un tel SET a le RECORD SYSTEM pour OWNER et un RECORD-TYPE du schéma pour MEMBER. Ce SET permet notamment d'accéder aux RECORDS d'un type selon un ordre trié.

2. Contraintes d'appartenance de RECORDS à un SET-TYPE

Il est possible d'imposer des contraintes sur l'appartenance de RECORDS à un SET-TYPE. Ces contraintes s'expriment par le mode d'insertion et par le mode de retention du RECORD-TYPE.

Le **mode d'insertion** précise si, lors de la création d'un RECORD, celui-ci est inséré automatiquement dans un type de SET (MEMBER AUTOMATIQUE), ou si cette insertion est entièrement à charge du programmeur (MEMBER MANUAL).

Le **mode de retention** précise si, lors de la modification de données, on impose qu'une fois inséré dans un set, un RECORD de type MEMBER doit désormais toujours faire partie d'un SET de ce type (MEMBER MANDATORY), ou au contraire s'il peut en être détaché (MEMBER OPTIONAL).

Les modes d'insertion et de retention déterminent << **quand il est nécessaire** >> et << **s'il est nécessaire** >> qu'un RECORD-TYPE possède un OWNER. Cependant rien ne détermine encore ce RECORD. C'est la raison d'être de la clause SET SELECTION. Elle précise comment le Data Base Control System devra repérer le SET qui est concerné par certaines opérations de mise-à-jour ou d'accès. Pour

plus de précisions, le lecteur consultera par exemple la référence citée en (1).

3. Modes de stockage

L'analyse approfondie des traitements doit permettre d'adopter des stratégies d'accès performantes sinon optimales. CODASYL permet deux techniques de base pour accéder aux RECORDS d'un type :

1. CALC : accès direct aux RECORDS sur base d'une valeur de clé.
2. VIA : accès aux RECORDS MEMBERS d'un SET après avoir accédé au RECORD OWNER du SET (généralement sur base d'une valeur de clé CALC).

La conception d'une solution performante détermine un mode de stockage des RECORDS en fonction des types d'accès les plus fréquents.

Le contrôle du placement des RECORDS en mémoire secondaire est dès lors un paramètre physique important. Outre le fait de pouvoir stocker les RECORDS d'un type dans une ou plusieurs AREA, CODASYL permet de définir 3 modes de rangement :

- RANGEMENT DE TYPE CALC

  Un DATA ITEM (ou groupe de -) d'un type de RECORDS peut être déclaré clé CALC du type de RECORDS. La valeur de clé CALC d'un RECORD est un input d'un algorithme de "hashing" qui contrôle le rangement physique des RECORDS dans une AREA permise.

- RANGEMENT DE TYPE CLUSTER

  Le concepteur d'une base de données peut spécifier dans le schéma DDL que les RECORDS OWNER et MEMBERS d'un SET soient physiquement proches les uns des autres. Procéder de la sorte, permettra de diminuer considérablement le nombre d'opérations Entrée/Sortie nécessaires lors des accès dans un SET. De meilleures performances globales en résultent.

1.- J.L. Hainaut, "Etude de la SET OCCURENCE SELECTION dans les rapports CODASYL 71 et 72", Notes de cours, Institut d'Informatique, 1981.

- RANGEMENT PAR DEFAUT

  Si aucun des deux modes CALC ou CLUSTER n'est utilisés, MDBS se charge du rangement. Ce type de rangement correspond pour le concepteur à un rangement au hasard.

SCHEMA PHYSIQUE D'UNE BASE DE DONNEES

Les paramètres physiques des données sont, pour la plupart, spécifiés dans un langage spécialisé, propre à chaque SGBD particulier. Ces spécifications, exprimées à part des spécifications logiques, constituent le SCHEMA PHYSIQUE de la base de données.

SCHEMA EXTERNE D'UNE BASE DE DONNEES

Un SCHEMA EXTERNE est un ensemble de spécifications(1) permettant de définir la structure externe des données perçue par les programmeurs d'une application. On associera, à chaque application, un SCHEMA EXTERNE.

NOTION DE COURANT

Le SGBD garde constamment les traces d'une série de registres. Il y a deux types de registres : les registres de courants et les registres de diagnostic. Un courant est une variable contenant l'adresse physique (dbkey) du RECORD le plus récemment accédé ou manipulé dans un certain référentiel (ou agrégat de RECORDS). Concrètement, le SGBD maintient :

- current of run unit
- current of record type
- current of set
- current of area.

Les courants sont mis-à-jour en cours d'exécution. La notion de CUI(2) permet au programmeur de sauver le contenu d'un registre. Ultérieurement, il pourra restaurer le contenu de ce registre à partir du contenu de cette variable CUI.

Le rôle principal de ces registres est de servir d'**arguments implicites** dans la plupart des opérations d'accès et de mise-à-jour de la base de données. Ainsi par exemple, l'accès au RECORD suivant dans un SET fournit le RECORD qui suit le "current of SET" pour ce SET-TYPE. On désigne ainsi implicitement le SET concerné et la position courante dans ce dernier.

1.- spécifications exprimées dans le langage SUBSCHEMA DDL
2.- Current User Indicator.

Par exemple, les registres de diagnostic ERROR-STATUS et RECORD-NAME donnent respectivement des informations sur la manière dont s'est exécutée la dernière opération et sur les propriétés du "current of run-unit".

CHAPITRE V

MBDS, UN SGBD DE TYPE RESEAU

L'exposé succinct du standard CODASYL 71 a permis d'éviter le piège de la discussion technique. Il visait tout au plus une introduction aux principes généraux.

Les seules caractéristiques MDBS non conformes CODASYL seront discutées. Elles seront tantôt plus puissantes, tantôt plus restrictives que les recommandations CODASYL.

Nous limiterons l'exposé aux différences les plus remarquables. Pour plus de précisions, nous renvoyons le lecteur aux brochures du fournisseur (MDBS-DDL, 81) et (MDBS-DML,81).

Dans ce chapitre, le lecteur trouvera également une brève description des différents composants du SGBD MDBS.

V.1 LES CARACTERISTIQUES MDBS

LES SETS ET CLAUSES DESCRIPTIVES ASSOCIEES

Le SET MDBS n'est pas à proprement parler conforme au SET CODASYL. Il est plus puissant dans un sens à préciser. Ce gain de puissance s'exprimerait simplement par le fait que le modèle de données MBDS est plus proche du modèle conceptuel des données.

Contrairement à CODASYL, MDBS permet les SETS N-N. La possibilité offerte d'utiliser des types de SETS M-N permettra notamment une représentation plus directe des structures conceptuelles ainsi que de meilleures performances d'accès.

La restriction CODASYL "un **RECORD ne peut appartenir à plus d'un SET d'un SET-TYPE déterminé**" nécessitait l'utilisation d'un artifice(1) pour représenter un type d'association N-N. MDBS lève cette restriction et offre une **représentation directe** du type d'associations N-N : le SET N-N. L'accès dans un SET N-N est bidirectionnel et permet donc, pour un SET, d'obtenir les RECORDS MEMBERS d'un RECORD OWNER, ainsi que les RECORDS OWNERS d'un RECORD MEMBER du SET.

La définition CODASYL des caractéristiques permises des types de SETS est élargie aux définitions suivantes :

SET MULTI-OWNER
Les OWNERS des SETS d'un SET-TYPE peuvent appartenir à plusieurs types.

SET RECURSIF
Les RECORDS OWNERS et MEMBERS d'un SET peuvent être du même type. La propriété de récursivité peut être définie sur tous les types de SETS autorisés : SET MULTI-OWNER, SET MULTI-MEMBER, SET MULTI-OWNER et MULTI-MEMBER,...

Les contraintes MDBS relatives à l'appartenance des RECORDS à un SET-TYPE sont plus strictes que leurs homologues CODASYL. Le mode d'insertion d'un RECORD dans un type de SET est entièrement conforme au standard CODASYL. On dispose donc, pour les types de RECORDS jouant un rôle d'OWNER ou de MEMBER dans un SET, des options AUTOMATIC et MANUAL.

Par contre le mode de rétention n'est pas conforme aux recommandations CODASYL 71. L'option MANDATORY du mode de

rétention n'est pas permise. En réalité, elle est remplacée par un mode plus stricte : l' **option FIXED**. Cette option impose qu'une fois inséré dans un SET, le RECORD doit désormais toujours faire partie de ce SET (MEMBER FIXED, OWNER FIXED).

MDBS utilise implicitement l'option THRU CURRENT of SET de la clause SET SELECTION. Cette option n'est pas facile à comprendre. Et pour cause. Elle fait référence au langage de manipulation de données (DMS) ainsi qu'à la notion de COURANT. Retenons dès à présent, que c'est l'option la plus efficace. Cette forme diffère des autres en ce sens que le SGBD ne doit pas accéder (via un FIND) au RECORD OWNER (ou MEMBER), mais plutôt utiliser le dernier record OWNER (MEMBER) déjà accédé.

Pour un lecteur plus averti, signalons enfin qu'aucune facilité n'est disponible pour gérer la redondance dans les structures de données MDBS(1), la prise en charge automatique de la vérification de contraintes d'intégrité non standards, le maintien de certaines statistiques d'accès. MDBS offre toutefois des facilités relatives au contrôle des valeurs d'un DATA ITEM (clause RANGE) et au contrôle des accès (clause ACCESS).

1.- CODASYL permet par exemple, de déclarer un DATA ITEM ayant même valeur qu'un autre, ou dont la valeur résulte d'une expression calculée à partir d'autres DATA ITEMS.

V.2 LES COMPOSANTS DU SGBD MDBS

Les fonctions offertes par un tel Système de Gestion de Données complexes couvrent à la fois la définition, la gestion et la manipulation des données.

A la lumière des méthodologies modernes de mise en oeuvre d'applications informatiques, le SGD MDBS regroupe un ensemble d'outils et de modèles permettant un grand nombre de fonctions. On cite notemmment la définition (DDL) et de la manipulation (DMS)des données, l'expression de requêtes dans un langage de haut niveau, le maintient de l'intégrité de la base de données ainsi qu'une certaine restructuration dynamique.

MDBS.DDL

Ce composant permet l'introduction/consultation/modification interactive des spécifications DDL d'une application. Les spécifications peuvent aussi faire l'objet d'une analyse de cohérence avant d'être compilées. Le compilateur d'un schéma source initialise une base de données vide et génère un dictionnaire de données.

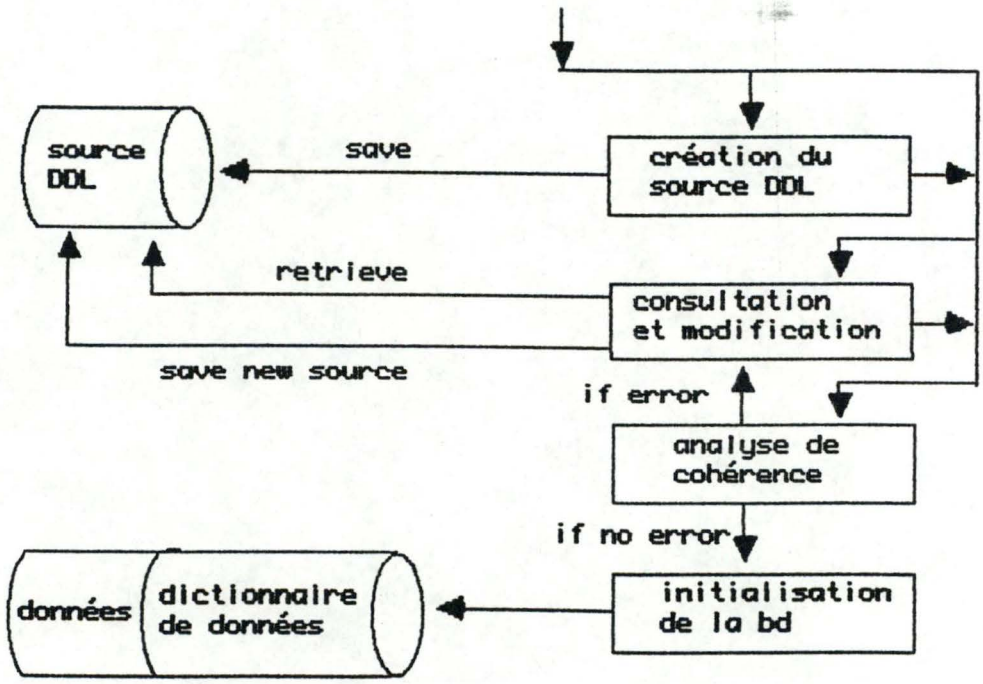


Figure 5.1 : génération d'une base de données vide

MDBS.DMS

Le composant DMS regroupe l'ensemble des routines SGBD. DMS est chargé en mémoire centrale et communique avec chacun des programmes d'applications. L'exécution de méta-requêtes permet de valider les arguments d'un ordre DMS issu d'un programme d'applications. Une fois validé, l'ordre DMS est exécuté. Le module réalise l'accès physique aux données et renvoit les résultats demandés au programme demandeur.

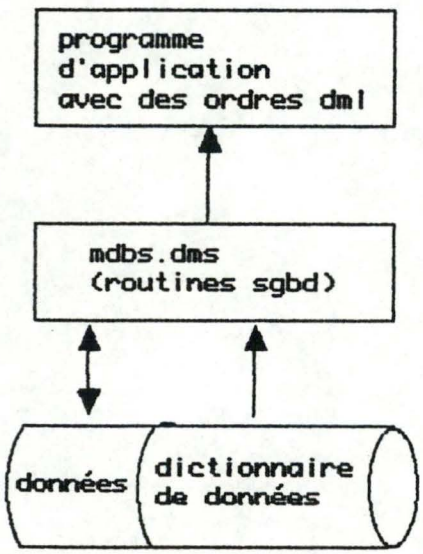
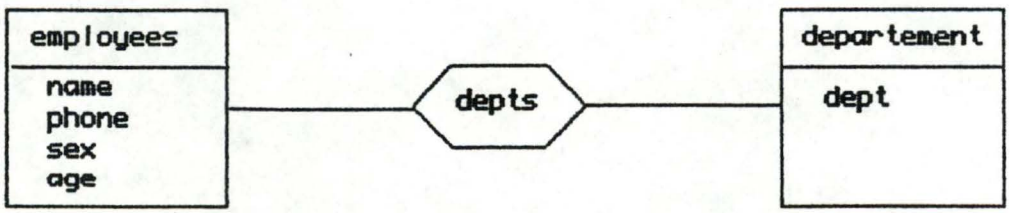


Figure 5.2 : exécution d'un programme d'application

MDBS.QRS

Le Query Retrieval System (QRS) est un langage d'interrogation non procédural qui permet à un utilisateur occasionnel de formuler des requêtes. Par exemple, la requête complexe de la figure V.4 permet de sélectionner le nom ainsi que le numéro de téléphone de toutes les employées du département FACTURATION et qui n'ont pas 35 ans.




```
list name,phone for sex="f",dept=43 and age<35 thru depts,employees
```

Figure 5.3 : formulation de requêtes I4g.

La réponse se présente sous la forme d'un rapport standard affiché à l'écran ou disponible dans un fichier. Un tel outil profite à l'utilisateur peu initié ainsi qu'à l'informaticien chevroné. Il permet à ce dernier de réduire considérablement l'effort de programmation (codage,debugging).

MDBS.RTL

En cas d'incident susceptible de déteriorer les données, le SGBD prend des mesures afin de restaurer les données détruites ou dans un état douteux. Il s'agit donc d'une protection contre les incidents (défaillance technique, coupure de courant,etc.).

MDBS.IDML

IDML est un langage interactif permettant une navigation assez élémentaire dans la base de données. Dans la pratique, cet outil est principalement utilisé pour "debugger" les programmes d'application.

MDBS.DMU

Ce module permet d'entreprendre des modifications simples du schéma de la base de données (structure physique des données, contraintes d'accès,etc.) sans devoir regénérer le dictionnaire de données ni opérer une opération de DECHARGEMENT/CHARGEMENT.

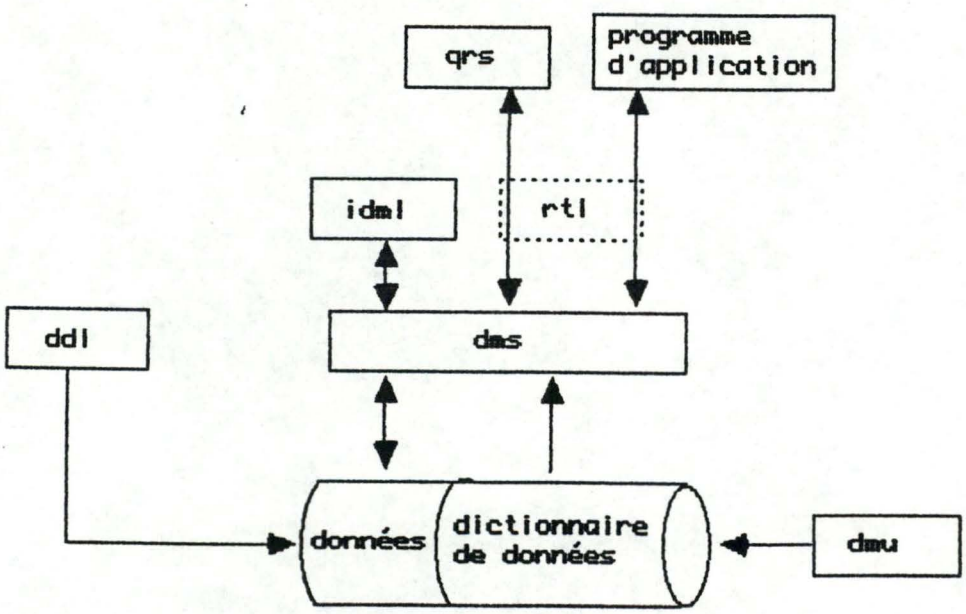


Figure 5.4 : les composants de base et annexes de MDBS

Bien qu'importantes, certaines fonctions comme la protection contre les incidents, les contrôles d'accès et de confidentialité, le suivi des performances et bien d'autres encore ne nous intéresseront pas. Seuls les composants de base, présents dans la version minimum du SGBD seront d'intérêt. On retiendra les compilateurs de descriptions -permettant d'initialiser une base de données sur base d'un schéma source MDBS-, le dictionnaire de données -qui contient la description exploitable des données-, et l'interpréteur(1) des commandes de manipulation de données d'un programme d'application -permettant l'exécution de celles-ci-.

1.- Data Base Control System ou DBCS.

CHAPITRE VI

CONCEPTION DE BASES DE DONNEES MDBS

Limité à l'aspect des DONNEES dans un système d'information, l'analyse conceptuelle est un processus qui conduit à la description des informations d'une base de données en projet, c-à-d le schéma conceptuel de la base de données. Cette description n'est en rien liée à une implémentation interne particulière qui sera ou pourrait être utilisée. Les critères de l'analyse conceptuelle sont doubles : représentativité du schéma par rapport à la réalité à décrire (le réel perçu), la communicabilité du schéma.

La phase de mise en oeuvre succède à l'analyse conceptuelle. Elle vise à traduire le schéma conceptuel en une solution physique c-à-d en un schéma exécutable par un SGBD cible. La mise en oeuvre de bases de données MDBS fait l'objet du chapitre VI.

VI.1 GENERALITES

Les démarches (1) modernes de conception de bases de données sont généralement décomposées en niveaux s'inspirant directement des niveaux de description de la base de données : conceptuel, logique et physique.

Le **schéma conceptuel** est un ensemble de spécifications exprimées dans un langage dit "de spécification". Elles décrivent une base de données en termes d'objets conceptuels : type d'entités, type d'associations, attributs attachés à chaque type d'entités et à chaque type d'associations. Le schéma comprend également les descriptions en langage naturel de ces objets ainsi que l'expression de contraintes dites d'"intégrité" dans un formalisme adéquat(2).

Le **schéma logique** est dérivé du schéma conceptuel (ou de l'une de ses variantes) par adjonction de structures d'accès aux données. Ces structures d'accès s'expriment sous la forme de clés d'accès et de (types de) chemins d'accès°HAI,86§.

Un **schéma physique** d'une base de données est une variante de schéma logique. Cette variante se caractérise principalement en sa conformité aux structures admises par un SGBD cible. Un schéma physique intègre également des spécifications plus techniques propres au SGBD.

Tels que présentés, les niveaux de description ne sont autres que des définitions. Elles constituent cependant des <<repères utiles>> pour le concepteur.

Aucune allusion n'est encore faite quant à la construction progressive de ces schémas. Les étapes sont multiples et linéaires au sens où le schéma résultat de l'une fait l'objet de transformations mises en oeuvre par l'étape suivante. Sous cette apparente linéarité, les étapes s'accompagnent cependant le plus souvent de rétroactions (retour de décision, optimisation de stockage et d'accès).

1.- J.L. Hainaut, Conception assistée des application informatiques, 2. Conception de la base de données, MASSON, 1986.
2.- langage naturel ou tout langage de désignation de données.

L'élaboration d'une base de données physique met en oeuvre un ensemble de processus de transformations. Chaque processus est caractérisé par un critère -ou en toute généralité- par un petit nombre de critères homogènes(1). Chaque processus produit une nouvelle description de la solution, équivalente à la description précédente, et qui intègre des choix faits par rapport à un critère (de décision). On peut citer par exemple les critères indépendants suivants : appels strictement nécessaires et suffisants, efficacité des accès logiques, conformité d'un schéma à un SGD (Système de Gestion de Données).

L'Atelier modulaire BD qui s'appuie sur cette démarche permet de construire progressivement un **schéma complet** c'est-à-dire couvrant des aspects tels que la structure sémantique, la structure d'accès, les quantifications (statitiques et dynamiques) et la structure physique.

La **structure sémantique** est première. Elle décrit la réalité à modéliser en termes de types d'entités, de types d'associations, d'attributs qui leur sont attachés, ainsi que de contraintes d'intégrité relatives à ces objets. La structure sémantique est établie lors de l'analyse conceptuelle et conservée sans perte au travers des transformations ultérieures.

La prise en compte des **accès** permet d'introduire au processus de conception sa seconde dimension : les TRAITEMENTS. Les DONNEES font l'objet de TRAITEMENTS. Et ces TRAITEMENTS déterminent les structures d'accès à mettre en oeuvre pour obtenir une solution physique performante.

Le schéma d'origine conceptuelle est donc enrichi de deux mécanismes d'accès : la clé d'accès et le type de chemins. On parle alors d'un schéma logique(2).

Une évaluation du caractère opérationnel de la solution est d'autant plus importante qu'il est bien vrai que l'utilisation de la base de données est forte consommatrice de ressources diverses(3). On envisage de contrôler à partir de ces mesures le fonctionnement du SGBD ainsi que l'état de la base de données. On surveillera des paramètres importants comme le temps de réponse, le nombre de lectures sur disques et le taux d'utilisation de l'espace disque. L'évaluation d'une base de données physique repose sur deux concepts fondamentaux : la quantification statique et la quantification dynamique.

1.- J.L. Hainaut, Conception assistée des application informatiques. 2. Conception de la base de données, MASSON, 1986.
2.- Etant lié à des notions de performance/coût, ces concepts de clé d'accès/type de chemins se trouveront dans un schéma logique ou physique, mais jamais dans un schéma conceptuel.
3.- temps CPU, espace mémoire,...

La **quantification statique** vise à établir une première quantification des données. L'ampleur de la base de données est déterminée par les "quotas" de cardinalité affectés aux types d'entités et types d'associations de la structure sémantique.

A l'image de toute solution bidimensionnelle réelle, la solution physique est un équilibre DONNEES - TRAITEMENTS. Sur base des requêtes de production(1), la **quantification dynamique** établit des quotas de fréquentation à greffer sur les types de chemins de la structure d'accès.

La taille des populations d'objets de la base de données(2) ainsi que la carte de trafic sont déterminées lors de l'analyse conceptuelle (ou plus exactement lors de l'analyse des besoins de l'organisation).

L'établissement de paramètres physiques(3) propre à un SGBD particulier est la dernière étape à mettre en oeuvre. Les dépendances vis-à-vis d'un SGBD spécifique sont ainsi localisées et limitées. Elles sont d'application en fin de processus d'élaboration, permettant une plus grande flexibilité lors de l'adaptation à d'autres exigences.

Dans ce chapitre, nous limiterons l'exposé de la démarche de construction de la base de données aux transformations nécessaires pour rendre un schéma conforme à MDBS. Cette limitation reflète d'ailleurs l'état actuel d'avancement des travaux relatifs à l'Atelier BD(4), si toutefois on étend le processus d'élaboration non pas au seul SGBD MDBS mais à tout SGBD cible.

Pour se répéter, mais la chose est importante, une transformation de schéma induit généralement une transformation d'algorithme. Cet aspect ne sera pas traité dans le présent mémoire. De même, bien que le volume des données réelle et la consommation de ressources qu'entraîneront les programmes accédant à la base de données soient d'une importance capitale(5), ces aspects ne nous préoccuperont pas d'avantage.

1.- par opposition aux requêtes L4G.
2.- J.L. Hainaut, Conception Assistée Des Applications Informatiques, Namur, Masson, 1986, p.44
3.- la taille des fichiers et de leurs pages, la taille des tampons, le mode d'implémentation des clés d'accès,etc.
4.- prototype actuel.
5.- puisque lié au caractère d'opérationnalité de la solution.

En toute généralité, on peut dire qu'un schéma peut être mis en oeuvre par un SGBD si les structures qui le composent respectent le modèle de données du SGBD ou du SGF. Il s'agit donc d'apprécier la **distance** séparant d'une part le modèle de données supportant l'analyse conceptuelle et d'autre part le modèle de données sous-jacent à MDBS.

Une fois identifiée, une structure non conforme est éliminée par application d'un opérateur transformateur de schémas. La **distance** séparant les deux modèles de données est ainsi progressivement réduite jusqu'à la conformité désirée.

L'Atelier logiciel BD, limité à la manipulation des structures de données, se compose d'un système d'outils coordonnés(1) permettant l'introduction assistée de spécifications, leur mémorisation, leur consultation, leur modification et leur gestion. Nous allons voir dans les pages qui suivent dans quelle mesure on peut parler de facilités d'environnement offertes par l'Atelier au concepteur.

VI.2 SCHEMAS GERES PAR L'ATELIER

L'Atelier est essentiellement conçu pour gérer la description physique d'une base de données, c-à-d son schéma logique et son schéma physique(2). Cet objectif assigné à l'Atelier justifie des restrictions quant aux schémas qu'il peut gérer. Les restrictions suivantes définissent la CONFORMITE d'un schéma à l'Atelier :

- tout type d'associations est de degré 2,
- un type d'associations n'a pas d'attributs,
- un rôle d'un type d'associations est joué par un seul type d'entités,
- une clé d'accès ne peut contenir plus d'un rôle,
- un identifiant contient
    - un ou plusieurs attributs,
    - deux ou plusieurs rôles,
    - un ou plusieurs attributs et un ou plusieurs rôles,
- une clé d'accès contient
    - un ou plusieurs attributs,
    - un rôle et un ou plusieurs attributs.

1.- L'Atelier est organisé autour d'une **base des spécifications**.
2.- variante de schéma logique.

VI.3 IDENTIFICATION DES STRUCTUIRES NON CONFORMES

L'identification des constructions non conformes à MDBS peut être effectuée manuellement. La tâche est alors fastidieuse. On préférera recourir à un outil de l'Atelier : le vérificateur de conformité. Ce vérificateur permet de repérer dans un schéma logique toutes les constructions non conformes à une liste de règles. Cette liste de règles exprime dans le cas bien particulier du SGBD MDBS les structures permises par MDBS. Le diagnostic renvoyé par le vérificateur se présente sous la forme d'une liste d'avertissements. L'utilisateur peut en prendre connaissance afin de corriger les stuctures inadéquates.

La définition de la CONFORMITE à un SGBD (et donc en particulier, à MDBS) couvre à la fois les aspects SEMANTIQUES et ACCES du schéma analysé.

Les règles de conformité à MDBS seront exprimées selon les concepts du modèle général(1) présenté dans la section GENERALITES de ce chapitre. Les principales contraintes que doit vérifier tout schéma afin d'être reconnu conforme aux spécifications MDBS sont présentées ci-dessous. Elles sont regroupées par concept du modéle général : type d'entités, type d'associations, attribut, identifiant, clé et fichiers.

REGLE RELATIVE AUX TYPES D'ENTITES

nombre d'attributs : de 0 à 65535

REGLES RELATIVES AUX TYPES D'ASSOCIATIONS

degré : de 2 à 2

présence d'attributs : non

valeurs permises pour la connectivité minimale : 0 et 1

valeurs permises pour la connectivité maximale : 1 et infini

combinaison de connectivités des rôles :

- un rôle i-infini et un rôle 0-infini
- un rôle i-1 et un rôle 0-infini
- un rôle i-1 et un rôle 0-1
- un rôle i-infini et un rôle 0-1

présence d'un type de chemins (ou type de sets) : associé à chaque type d'associations

nombre de types d'entités par rôle
de 1 à 127 pour le côté owner du type de sets
de 1 à 127 pour le côté member du type de sets

présence d'un même type d'entités dans plus d'un rôle d'un type d'associations : oui

absence de cycles bloquants (connectivité 1-1 ou 1-infini)

REGLES RELATIVES AUX ATTRIBUTS

nombre de niveaux : 1

répétitivité minimum : 1 à 1

répétitivité maximum : 255

répétitivité fixe et limitée

répétitivité limitée avec compteur

nombre de niveaux répétitifs : 0

format : caractère, entier, réel, string, binaire, non signé, décimal interne, temps, date

longueur : selon le format

REGLES RELATIVES AUX IDENTIFIANTS ET AUX CLES D'ACCES

nombre d'identifiants constitué d'attributs par type d'entités : 0 à 1

nombres d'identifiants mixtes par type d'entités : 0 à N, avec N le nombre de types d'associations dans lequel le type d'entités joue un rôle

nombre de clés d'accès constitué d'attributs par type d'entités : 0 à 1

nombre de clés d'accès mixtes par type d'entités : 0 à N, avec N le nombre de types d'associations dans lequel le type d'entités joue un rôle

nombre de clés d'accès mixtes par type d'associations : 0 à 2 (au plus une par rôle)

un identifiant doit-il être une clé d'accès : oui

une clé d'accès doit-elle être identifiante : non

répétitivité d'un attribut composant : non

niveau d'un composant dans la décomposition des attributs d'un type d'entités : 1

possibilité de chevauchement de deux identifiants ou clés : oui

une clé peut-elle être un préfixe d'une autre (ou d'un identifiant) : oui

connectivité d'un rôle composant d'un identifiant : 0-infini

REGLES RELATIVES AUX FICHIERS

nombre de fichiers par type d'entités : 1 à 16

nombre de types d'entités par fichier : 0 à 255

REGLES RELATIVES AUX NOMS

longueur : 1 à 8

alphabet : a..z, A..Z, 0..9

REGLES D'UNICITE DES NOMS

unicité des noms de types d'entités d'un schéma : oui

unicité des noms de types d'associations d'un schéma : oui

unicité des noms de fichiers d'un schéma : oui

unicité des noms d'attributs d'un type d'entités du schéma : oui

unicité des noms d'attributs d'un schéma : non

Le vérificateur de conformité de l'Atelier est un outil générique c'est-à-dire susceptible d'être instantié à un jeu de régles (Codasyl, relationnel, Cobol, MDBS,...). Dans le cadre de la conception de base de données MDBS, le vérificateur valide les différentes contraintes MDBS-3. Le sens de parcours de la base, adopté par le vérificateur, a fait l'objet de certaines optimisations. Remarquons cependant que le caractère général du vérificateur restreint considérablement le sens dans lequel il faut entendre "optimisation de parcours"(1).

Du vérificateur de conformité nous ne dirons guère d'avantage. Retenons simplement qu'il est générique(2), qu'il fournit une liste d'avertissements signalant à l'administrateur de la base de données toutes les structures non conformes, et que finalement il serait souhaitable qu'il assiste ou guide l'administrateur sur les solutions possibles permettant d'éliminer les structures erronées.

1.- De plus amples précisions concernant l'optimisation de parcourt sont disponibles dans le mémoire de C. Charlot et Müller.
2.- c'est-à-dire un outil qui doit être instantié dans le cadre d'une démarche particulière.

VI.4 LES TRANSFORMATIONS NECESSAIRES

Les lignes qui suivent présentent les quelques transformations nécessaires (ou utiles) dont devrait disposer le concepteur pour produire des schémas conformes MDBS.

* SPECIFICATIONS D'UN OPERATEUR TRANSFORMATEUR DE SCHEMAS

Les modifications de schémas sont en général très fortement corrélées. Ceci oblige à les regrouper puisque séparément non applicables. Les transformations disponibles dans l'Atelier sont de tels agrégats. Elles correspondent à des opérations logiques, élémentaires au yeux du concepteur. Elles sont ainsi un meilleur support pour le concepteur puisqu'elles opèrent au même niveau de conception. Les transformations de schéma logique à schèma logique ne sont pas certes faciles à mettre en oeuvre. Les difficultés proviennent spécialement des exigences de qualité attendues quant au schéma résultat des opérateurs de transformation : équivalence sémantique(1) et d'accès par rapport à la version antérieure, <<meilleure>> conformité au SGD cible.

La sémantique et la structure d'accès sont héritées du schéma parent. Ceci exige des opérateurs de transformation, une transparence totale sur les aspects sémantiques et structure d'accès du schéma.

D'autre part il s'agit aussi d'assurer au schéma un rapprochement strict vers la conformité puisqu'une transformation de conformité pourrait -si non contrôlée- introduire de nouvelles constructions non conformes.

Le noyau de tranformations nécessaires à la production de base de données MDBS est relativement restreint. Ces transformations sont indépendantes de la nature conceptuelle ou logique du schéma sur lequel elles opèrent. En fait elles sont pour la plupart d'application aux deux niveaux.

Suivant la démarche appliquée dans l'Atelier ces transformations seront d'abord des transformations E/A -> E/A(2) puis des transformations MAG -> MAG(3).

1.- conservation de la spécification conceptuelle.
2.- Conformité à l'Atelier : cfr. "Schémas gérés par l'atelier", p.5 du CH5.
3.- Conformité au SGBD cible.

Une spécification unique est proposée pour chacun de ces "outils" transformateurs de schémas. Cette spécification est de nature conceptuelle c-a-d exprimée selon les objets conceptuels du modèle Entité/Association(1). La terminologie conceptuelle sera conservée quelque soit le niveau de conception où ces transformations seront d'application. Deux remarques importantes sont à faire.

La première se veut rassurante. Une spécification indépendante du niveau de conception n'est pas une erreur ou même une simplification provocante. Spécifier les transformations de la sorte est souhaitable puisque la correspondance avec les objets d'un modèle conceptuel et ceux du modèle MAG est presque immédiate. Les principes établissant la correspondance E/A -> MAG sont les suivant :

1. **Entité et type d'entités**
   On représente une entité par un article et un type d'entités par un type d'articles.

2. **Valeur d'attribut et attribut d'un type d'entités**
   A une valeur d'attribut et à un attribut correspondent respectivement une valeur d'item et un item du type d'articles représentant le type d'entités. La répétitivité, la décomposabilité et le caractère facultatif de l'item découlent immédiatement des propriétés de l'attribut.

3. **Type d'associations binaire sans attribut**
   Il y correspond un type de chemin entre les types d'articles correspondants. La classe fonctionnelle et les contraintes d'existences se déduisent de la connectivité du type d'assotiations.

4. Toute structure E/A peut se ramener sans perte à un schéma E/A primaire(2) via les transformations à spécifier. Les types d'associations au moins ternaire sans attribut, ainsi que les types d'associations, avec attributs seront éliminés par application du mécanisme d'agrégation(3).

1.- Type d'entités, Type d'associations, Attribut.
2.- Les seuls concepts admis sont : entité et type d'entités, valeur d'attribut et attribut d'un type d'entités, type d'associations binaires sans attribut.
3.- mécanisme d'abstraction permettant de représenter une association entre des objets par un objet autonome.

La seconde remarque précise quelque peu la notion de spécification accolée à une transformation. Deux types de spécification sont disponibles : la spécification interne et la spécification externe. Ces spécifications se complètent mutuellement et ne s'adressent généralement pas aux même personnes. La spécification interne est un relevé technique des règles systématiques qui permettent de générer un schéma logique à partir d'un autre. Ces règles ne nous intéressent pas spécialement, sinon de les savoir systématiques, et donc d'application aveugle. Les spécifications externes sont par contre bien plus intéressantes et même suffisantes pour un utilisateur de l'Atelier. Elles mettent en oeuvre un principe fondamental, celui de l'abstraction procédurale(1). Les transformations nécessaires à la génération de la base de données seront donc présentées selon le modèle externe suivant :

NOM DE LA TRANSFORMATION : .....

FONCTION : .....
.....
.....

APPLICATIONS : .....
.....

SCHEMA DE DEPART : S1

précondition : condition d'applicabilité de la transformation

SCHEMA RESULTAT : S2

post-condition :
-S2 contient la même sémantique que S1
-S2 offre les mêmes possibilités d'accès que S1
-Les quantifications de S2 se déduisent de S1

Ce modèle de spécification externe reflète très bien l'exigence de réversibilité sémantique des transformations à spécifier. On a donc la garantie qu'il n'y a pas de perte de sémantique par application de l'une de ces transformations.

1.- spécifications en termes de pré- et de post- conditions exprimant les propriétés du schéma initial et résultat.

La seconde propriété des transformations précise que toute structure d'accès de S1 (clé ou type de chemins) doit trouver dans S2 une combinaison d'une ou plusieurs structures d'accès élémentaires, offrant les mêmes fonctionnalités(1). Ainsi par exemple, toute clé du schéma S1 peut être remplacée dans S2 par une clé et un type de chemins, tout type de chemins à transformer est remplacé dans S2 par une clé ou par la composition de deux types de chemins.

La propriété de quantification est, contrairement aux deux premières, immédiate c'est-à-dire qu'elle caractérise le schéma résultat de la transformation sans pour autant être une contrainte de transformation. Elle se déduit automatiquement.

Avant d'entamer la spécification des transformations toutes nécessaires à l'obtention d'une base de données MDBS, il est opportun de savoir quand, en toute généralité, une transformation sera appliquée. Il est par exemple toujours nécessaire de procéder à l'élimination de types d'associations avec attribut, et de types d'associations dont le degrè est supérieur à 2 (cfr les schémas gérés par l'Atelier). De telles transformations constituent donc un point de passage obligé puisqu'établissant la conformité MAG. Toutes les transformations disponibles dans l'Atelier sont d'application facultative, mis à part celles-là mêmes qui permettront d'établir la conformité au SGBD cible.

Par application facultative, on veut dire que toutes les transformations ne sont pas des transformations de conformité, que d'ailleurs la qualité "de conformité" n'est jamais liée de manière permanente à une transformation, et que certaines transformations résultent d'une décision fondée sur un tout autre critère : clarté des structures de données, minimisation du temps d'accès, minimisation de mise-à-jour de la base de données ou de la place mémoire occupée. La transformation "retour de décision" est un autre exemple important de transformation facultative.

1.- Fonctionnalité doit être pris dans le sens <<possibilités d'accès>> et non pas <<performances d'accès>>, puisque la désagrégation d'un schéma induit généralement de moindres performances.

* TRANSFORMATION D'UN TYPE D'ASSOCIATIONS

Cette transformation consiste à éliminer un type d'associations inadéquat par rapport à un critère. Le critère de conformité(1) détermine quelles transformations de l'Atelier seront indispensables.

Et puisque l'élimination d'un type d'associations peut être réalisée de diverses manières, on envisage une découpe plus fine de cette transformation. On parlera de transformation d'un type d'associations en un type d'entités et de la transformation d'un type d'associations en attribut.

La transformation d'un type d'associations en type d'entités est -comme on le verra- indispensable lors de la génération de schéma MDBS dans l'Atelier BD. L'application de cette transformation permettra d'éliminer certaines structures non conformes (1).

Dans d'autres situations, ou parfois même dans des situations identiques, on pourrait vouloir transformer un type d'associations en un ou plusieurs attributs.

Les opérateurs transformateurs de type d'associations sont décrits ci-après.

1.- conformité à l'Atelier et au SGBD.

NOM DE LA TRANSFORMATION : < TRANSFORMATION TA -> ATT >

PRE-CONDITIONS

Soit RAB, le type d'associations défini entre les types d'entités A et B :

- RAB est de degré 2
- A est un type d'entités doté d'un identifiant IA
- IA est composé uniquement d'attributs
- B est un type d'entités qui joue un rôle de connectivité i-1 dans le type d'associations à éliminer.

FONCTION

L'opérateur TRANSFORMATION TA -> ATT permet de représenter un type d'associations illicite RAB défini entre les types d'entités A et B par la duplication dans B d'un ou plusieurs attributs de A (soit IA).

APPLICATION

Une contrainte MDBS limite à deux le nombre de clés d'accès pouvant être supporté par un type d'associations : plus précisément, une au plus par type de chemins. Un type d'associations pourra être transformé via l'opérateur TA -> ATT s'il respecte les pré-conditions définies.

EQUIVALENCE SEMANTIQUE

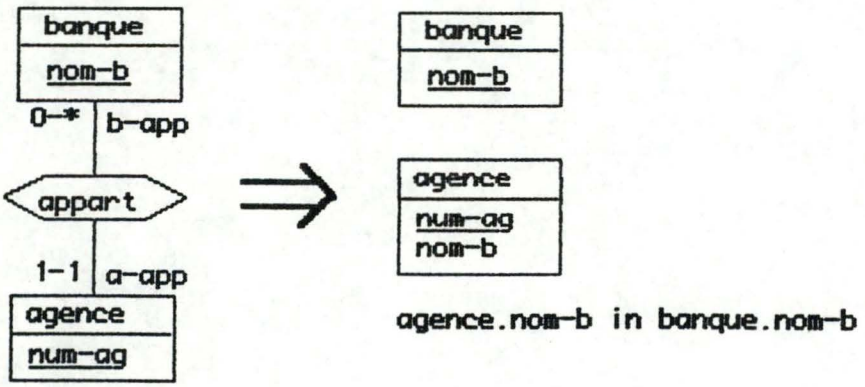


Figure 6.1 : transformation d'un type d'associations en un attribut

Le type d'associations APPART est défini entre le type d'entités AGENCE - jouant un rôle (a_app) de connectivité i-1 -, et le type

d'entités **BANQUE** - jouant un rôle (**b_app**) de connectivité 0-j. **BANQUE** est doté d'un identifiant primaire constitué uniquement d'attributs (**nom_b**). La transformation substitue au type d'associations non conforme un nouvel attribut de non **nom_b**, ajouté à **AGENCE** et de même valeur que **nom_b** de **BANQUE**.

EQUIVALENCE D'ACCES

Les mécanismes d'accès devant être respectés par la transformation sont le (ou les deux) type(s) de chemins correspondant au type d'associations à éliminer, ainsi que les clés d'accès composées d'un rôle de ce type d'associations.

TRANSFORMATION INVERSE

Les structures initiales peuvent être retrouvées par application inverse des règles définissant la transformation première.

NOM DE LA TRANSFORMATION : < TRANSFORMATION TA -> TE >

FONCTION

Cette transformation consiste à représenter un type d'associations inadéquat A par un nouveau type d'entités E et par de nouveaux types d'associations reliant E aux membres de A. Cette transfor-mation se base sur le principe d'agrégation.

APPLICATIONS

Les situations d'applicabilité de l'opérateur TRANSFORMATION TA -> TE seront généralement présentes dans deux contextes : la production d'un schéma logique à partir d'un schéma conceptuel (CONFORMITE A L'ATELIER), et l'élimination de constructions non conformes MDBS dans un schéma logique (CONFORMITE AU SGBD).

La transformation appliquée à un schéma conceptuel, en vue de la CONFORMITE A L'ATELIER, doit permettre l'élimination de type d'associations avec attribut, de type d'associations dont le degré est supérieur à 2.

Aucune transformation de CONFORMITE MDBS relative à l'objet type d'associations d'un schéma logique n'est à proprement parler indispensable. Des critères autres que la CONFORMITE, militent parfois pour l'élimination de type d'associations N-N et de type d'associations récursif.

STRUCTURE SEMANTIQUE

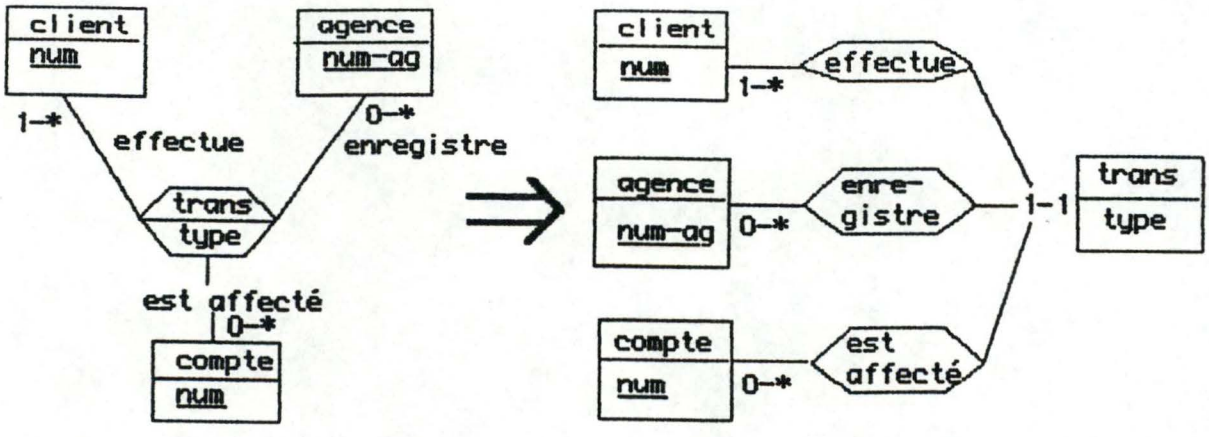


Figure 6.2

STRUCTURE D'ACCES

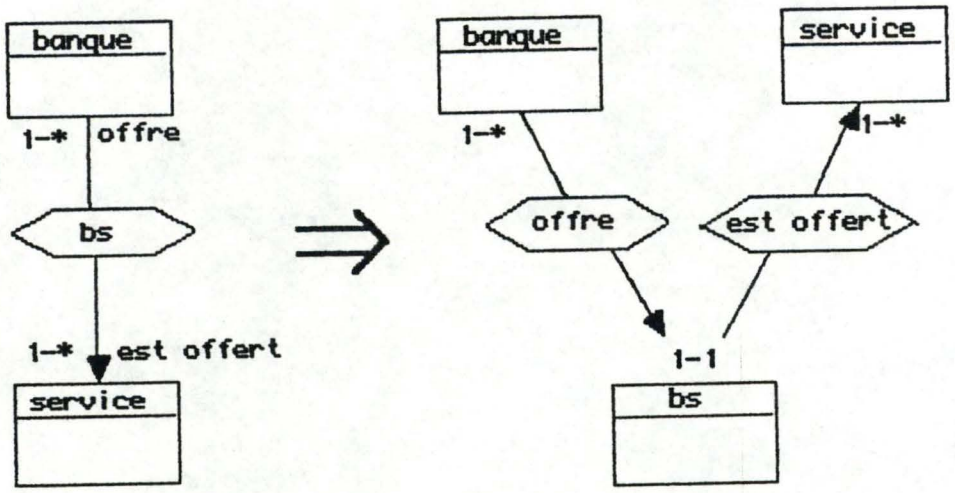


Figure 6.3

TRANSFORMATION INVERSE

Les structures initiales peuvent être retrouvées par application inverse de règles définissant la transformation première.

* TRANSFORMATION D'UNE CLE D'ACCES EN TYPE DE CHEMINS

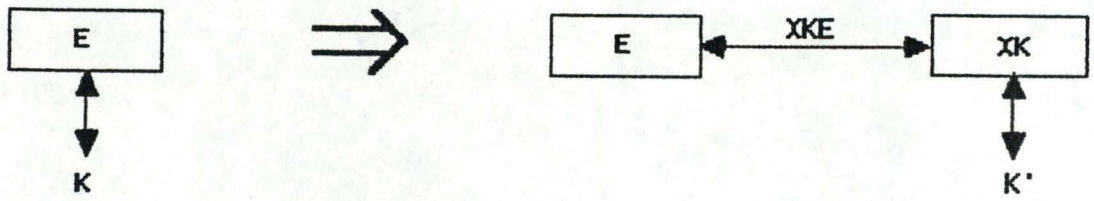


Figure 6.4

NOM DE LA TRANSFORMATION : < TRANSFORMATION CLE -> TC >

PRE-CONDITIONS

La clé K obéit aux conditions suivantes :

- elle est constituée uniquement d'attributs
- elle sera simple (constituée d'attributs simples) ou répétitive (constituée d'un seul attribut, qui est répétitif)
- si elle est simple, elle sera identifiante ou non identifiante
- si elle est répétitive, elle sera nécessairement non identifiante (un identifiant ne peut être répétitif)
- elle est obligatoire (puisque tous ses composants doivent être obligatoires)

FONCTION

La TRANSFORMATION CLE -> TC permet de substituer au mécanisme d'accès par clé (secondaire, de composition non conforme à SGBD cible, etc.) un accès par type de chemins. Le schéma résultat de la transformation nécessite un nouveau type d'entités XK doté d'une clé d'accès K' (image de la clé K associée dans le schéma initial au type d'entités E) et relié à E par un nouveau type d'associations XKE.

APPLICATION

La TRANSFORMATION CLE -> TC permet la conformité au SGBD MDBS. Elle vise à résoudre deux problèmes distincts :

1. Par type d'entités, MDBS ne permet qu'une clé d'accès

constituée d'attributs du type d'entités : la clé CALC. Il s'agit donc d'éliminer d'autres clés, constituées uniquement d'attributs, et qui seraient éventuellement définies.

2. Les clés d'accès répétitives ne sont pas tolérées par MDBS. Elles sont dès lors déclarées non conformes et devront être éliminées.

EQUIVALENCE SEMANTIQUE

La TRANSFORMATION CLE -> TC garantit l'équivalence sémantique des structures initiale et résultat.

Nous envisagerons 2 cas de figure, représentatifs des conditions d'applicabilité de la transformation. Les figures 6.5 et 6.6 illustrent respectivement la transformation d'une clé simple et la transformation d'une clé répétitive.

Considérons le schéma de la figure 6.5. Le schéma initial contient une clé d'accès simple (nom) à transformer.

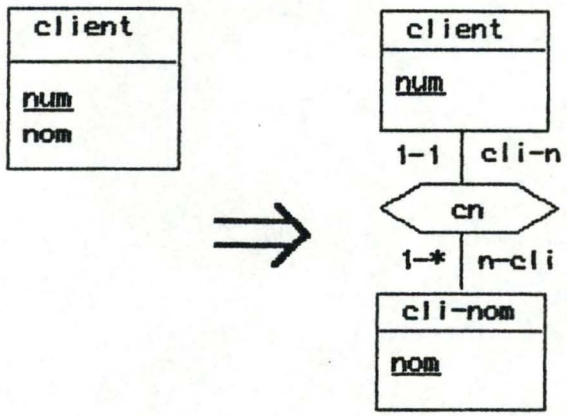


Figure 6.5 : transformation d'une clé d'accès simple (aspect sémantique)

Le type d'entités artificiel **CLI-NOM** est doté des attributs constitutifs de la clé à transformer. Ces attributs transférés constituent, pour **CLI-NOM**, une clé identifiante primaire simple et obligatoire.

Le type d'associations **CN** relie **CLIENT** au nouveau type d'entités **CLI-NOM** et est définie comme suit :

- connectivité (1-1) pour le rôle joué par **CLIENT** dans le nouveau type d'associations
- connectivité (1-*) pour le rôle joué par **CLI-NOM**, puisque la clé était non identifiante pour **CLIENT**

Considérons à présent la figure 6.6 dans laquelle le schéma initial contient la clé d'accès répétitive TELEPHONE à transformer.

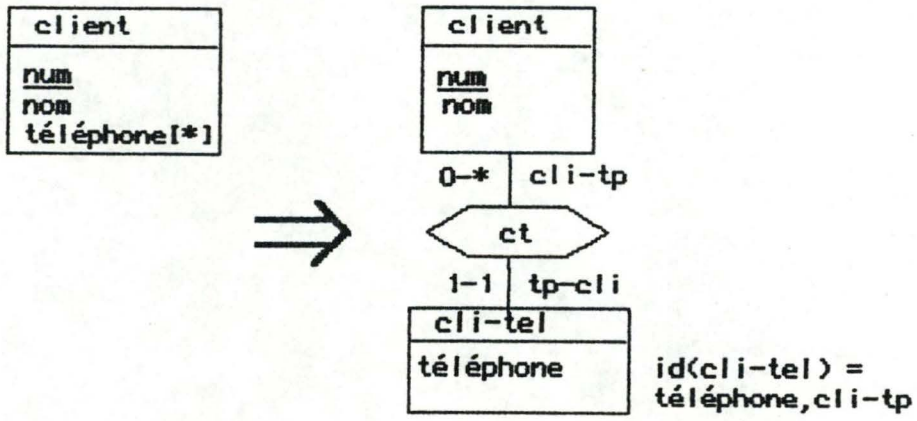


Figure 6.6 : transformation d'une clé d'accès répétitive (aspect sémantique)

On définit le type d'entités artificiel CLI-TEL. On le dote de l'attribut répétitif composant la clé à éliminer. Cet attribut constitue une clé simple et obligatoire pour CLI-TEL. Il y a une entité CLI-TEL pour chaque valeur composant une clé répétitive du schéma de départ.

Le nouveau type d'entités CLI-TEL est relié à CLIENT par le type d'associations CT défini comme suit :

- connectivité (1-j) pour le rôle joué par CLIENT, avec j la répétitivité maximale de la clé à éliminer
- connectivité (1-1) pour le rôle joué par le nouveau type d'entités CLI-TEL

EQUIVALENCE D'ACCES

Dans le schéma résultat, les structures d'accès sont équivalentes à ceux du schéma de départ si on se tient aux principes suivants:

- les attributs du nouveau type d'entités constituent une clé pour ce dernier
- un type de chemins d'accès est défini du nouveau type d'entités vers l'ancien (pour compléter la simulation de l'accès par l'ancienne clé)
- un type de chemins d'accès est défini de l'ancien type d'entités vers le nouveau (pour permettre l'accès aux valeurs des anciens attributs)

Ces principes sont indépendants de la nature de la clé (simple, répétitive). La figure 6.7 illustre ces principes dans le cas d'une clé simple.

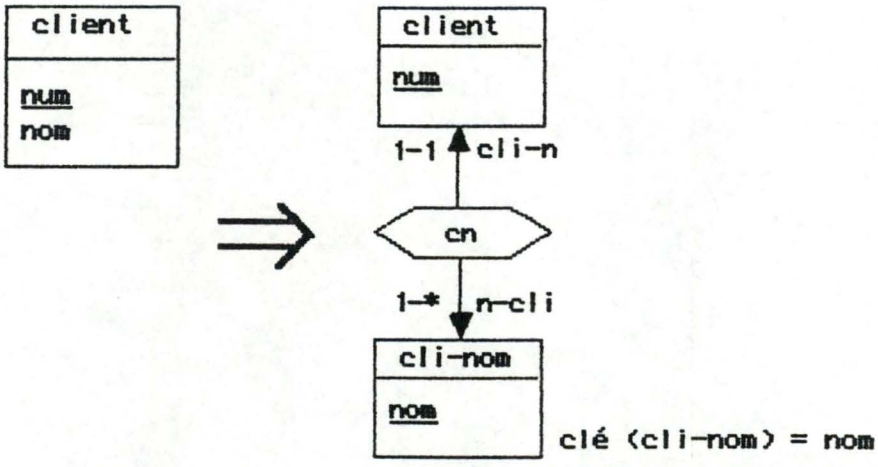


Figure 6.7 : transformation d'une clé d'accès simple (structure d'accès)

REMARQUES

Deux remarques importantes sont à faire à propos du transfert ou de la copie des attributs constitutifs de la clé à transformer.

Il est des situations de chevauchement de clés où sont impliqués un ou plusieurs composants d'une clé à transformer. De telles situations nécessitent de copier ces composants dans le nouveau type d'entités, plutôt que d'effectuer un simple transfert d'attributs.

La seconde remarque pose le problème des performances d'accès aux valeurs des attributs transférés. La désagrégation d'un type d'entités en plusieurs types d'entités entraîne des coûts d'accès suppléméntaires (par exemple, des couts de positionement préalable) qui ne sont pas négligeables. Lorsque les performances du schéma résultat ne seraient pas satisfaisantes, on proposera une transformation mixte. Celle-ci consiste à introduire de la redondance dans le schéma résultat via des attributs dupliqués. Cette redondance doit alors être gérée soit par le SGBD (ce qui est rare), soit par le programmeur qui réalisera des procédures synchronisant la mise-à-jour des attributs dupliqués (ce qui n'est pas toujours simple).

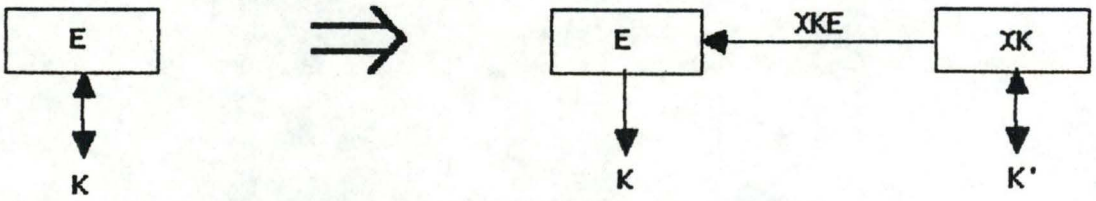


Figure 6.8 : gestion de la redondance

TRANSFORMATION INVERSE

Les structures initiales peuvent être retrouvées par application inverse de règles définissant la transformation première.

*** APLATISSEMENT DES ATTRIBUTS**

NOM DE LA TRANSFORMATION : < TRANSFORMATION ATTD -> ATTS >

FONCTION

La transformation ATTD -> ATTS permet l'aplatissement des attributs décomposables d'un type d'entités. Chaque attribut d'une arborescence d'attributs est transformé en un attribut du premier niveau, et ce jusqu'à ce que tous les attributs du type d'entités soient simples.

APPLICATION

Un type d'entités possède un ou plusieurs attributs décomposables.

* ELIMINATION D'UN IDENTIFIANT

NOM DE LA TRANSFORMATION : < ELIMINATION D'UN ID >

REMARQUE

La transformation ELIMINATION D'UN ID induit une perte de sémantique structurale c-à-d que l'identifiant détruit ne fait plus partie de la structure sémantique du schéma. Elle ne devrait dès lors être utilisée qu'en fin de processus de conception.

FONCTION

La transformation ELIMINATION D'UN ID détruit un identifiant défini de manière explicite et le remplace par une contrainte d'intégrité exprimée de manière informelle et devant être gérée par programmation explicite(1). Les composants cessent d'être associés en un identifiant mais conservent leur existence et autres propriétés.

APPLICATION

Les contraintes MDBS relatives aux identifiants et aux clés sont précises. Une clé n'est pas nécessairement identifiante. Un identifiant d'un type d'entités est par contre obligatoirement une clé d'accès de ce type d'entités. Pour ces raisons, tout identifiant d'un type d'entités du schéma conceptuel et constitué d'attributs, est déclaré clé identifiante au moment du chargement du schéma dans la base de spécifications de l'Atelier.

1.- Cette contrainte d'intégrité est un aide mémoire pour le programmeur.

* NORMALISATION DES CONNECTIVITES

NOM DE LA TRANSFORMATION : < NORMALISATION DES CONNECTIVITES >

REMARQUE

La transformation NORMALISATION DES CONNECTIVITES induit une perte de sémantique structurale et ne devrait être réalisée qu'en fin de processus de conception.

FONCTION

Cette transformation permet de remplacer la connectivité d'un rôle par une connectivité plus large.

APPLICATION

La transformation NORMALISATION DES CONNECTIVITES permettra de relâcher des cycles bloquants c'est-à-dire des cycles dont tous les maillons ont une deuxième connectivité (1-1).

STRUCTURE SEMANTIQUE

La connectivité d'un rôle (1-1) impliquée dans un cycle bloquant est remplacée par une connectivité plus large : (0-1) ou (0-infini). La connectivité initiale du rôle donne lieu à un contrainte d'intégrité informelle qu'il faudra désormais gérer de manière manuelle.

STRUCTURE D'ACCES

Cette transformation ne concerne pas les structures d'accès.

TRANSFORMATION INVERSE

Cette transformation n'est en principe pas réversible et n'admet pas de transformation inverse.

VI.5 LES MODIFICATIONS NECESSAIRES

Les outils transformateurs de schémas proposés dans l'Atelier sont des facilités offertes à l'administrateur de la base de données pour atteindre la conformité désirée. Ces facilités sont plus que souhaitables. Elles sont d'utilisation aisées, et assurent une manipulation correcte des structures de schéma. Elles permettent finalement, une simplification du processus de conception lui-même.

Les spécifications externes(1) d'un opérateur permettent à l'utilisateur de faire abstraction des principes établissant le schéma résultat à partir du schéma initial. Remarquons également que le nombre restreint des opérateurs permettant d'éliminer la plupart des structures non conformes simplifie considérablement le processus de conception.

L'outil de base, commun à tous ces transformateurs de schémas, n'est autre qu'un outil de modification des schémas gérés par l'Atelier. Cet outil plus rudimentaire, au sens où il opère à un niveau plus bas et donc plus technique, est de toute évidence indispensable à l'administrateur de la base de données. Il s'agit d'une part de réaliser les modifications de schéma non couvertes par les transformations de conformité de l'Atelier, et d'autre part de disposer d'un mécanisme d'enrichissement des schémas. Ce mécanisme permettra notemment de spécifier les paramètres physiques en cours de conception.

L'outil **modification d'un schéma** est interactif et permet via des opérateurs de base de modifier les différents aspects de la description d'une base de données.

Le processus de modification est assisté d'une succession d'écrans menant à bien la spécification d'une modification à opérer.

La modification des structures sémantiques couvre principalement la modification/création/suppression d'un type d'entités, d'un type d'associations, d'un attribut, d'un identifiant. Par exemple, l'opérateur de modification d'un arbre d'attributs attaché à un type d'entités permet d'ajouter ou de supprimer un attribut à une certaine position dans l'arbre. La suppression d'un identifiant est un autre exemple. Remarquons que cette modification n'est applicable que si celui-ci n'est pas impliqué dans une contrainte référentielle.

1.- Une spécification externe s'exprime en termes de fonctionnalité et de condition d'applicabilité.

La **modification** des structures d'accès désigne à la fois la création d'une clé d'accès, la modification/création/suppression d'un composant de clé, la modification/création/suppression d'un type de chemins d'accès.

Dans le cadre de MDBS, on envisagera les modifications de structures non conformes. Rendre obligatoire un attribut qui ne l'était pas est **un exemple de modifications de "conformité"**.

VI.6 LE SCHEMA MDBS

Après mise-à-conformité des structures contenues dans la base de spécifiactions, on souhaiterait générer automatiquement, et dans la syntaxe MDBS, un texte décrivant la base de données en projet. La réalisation du générateur-MDBS(1) permet donc de produire, à partir de l'Atelier, un texte analysable par le composant DDL de MDBS. Ce texte source est encore appelé **schéma** MDBS.

1.- L'implémentation du générateur-MDBS est décrite en annexe.

VI.7 SCENARIO DE GENERATION D'UNE BASE DE DONNEES CONFORME MDBS

Le chapitre VI a permis jusqu'ici d'établir, dans le cadre du SGBD MDBS, certaines classes de problèmes relatifs à la conformité, aux performances,etc. On y a également évoqué les fonctions et modèles à mettre en oeuvre pour résoudre/organiser ces problèmes.

Quand on parle d'environnement privilégié à propos de l'Atelier BD, on veut dire que les différents composants qui assurent ces fonctions sont intégrés en un système informatique global.

Une fois la maîtrise locale atteinte, il fallait agencer ces composants entre eux, les mettre en relation les uns avec les autres. L'Atelier est donc avant tout une architecture de composants organisés autour d'une BASE DE SPECIFICATIONS (spécifications relatives à un schéma de base de données).

Les fonctions de l'Atelier évoquées dans ce chapitre concernent l'initialisation de la base des spécifications avec un schéma de base de données, la consultation du schéma, la modification du schéma, la transformation de structure du schéma, la vérification de conformité du schéma à MDBS, la génération de texte source MDBS.

INITIALISATION DE LA BASE DES SPECIFICATIONS

La première fonction de l'Atelier est l'initialisation de la base des spécifications. Le schéma conceptuel Entité/Association à insérer dans l'Atelier BD est issu de la base des spécifications IDA et se présente sous la forme d'un fichier. Durant la phase d'initialisation, ce schéma conceptuel est rangé dans la base des spécifications de l'Atelier puis transformé en un premier schéma logique. Ce schéma est obtenu suivant les règles suivantes :

- à tout objet est associée son origine conceptuelle;
- tout type d'associations de degré supérieur à 2 est transformé en type d'entités;
- tout type d'associations doté d'attributs est transformé en type d'entités;
- un des identifiants constitué d'attributs est déclaré primaire;
- tout type d'entités reçoit une abréviation;

1.- cfr. le document L'Atelier de Conception de Base de Données ORGA, manuel de référence, version 1.0., Avril 1987.

- à tout type d'associations sont ajoutés deux types de chemins d'accès;
- tout identifiant constitué d'attributs devient également une clé d'accès;
- sur demande explicite de l'utilisateur, les types d'associations N-N et les types d'associations récursifs sont transformés en types d'entités;

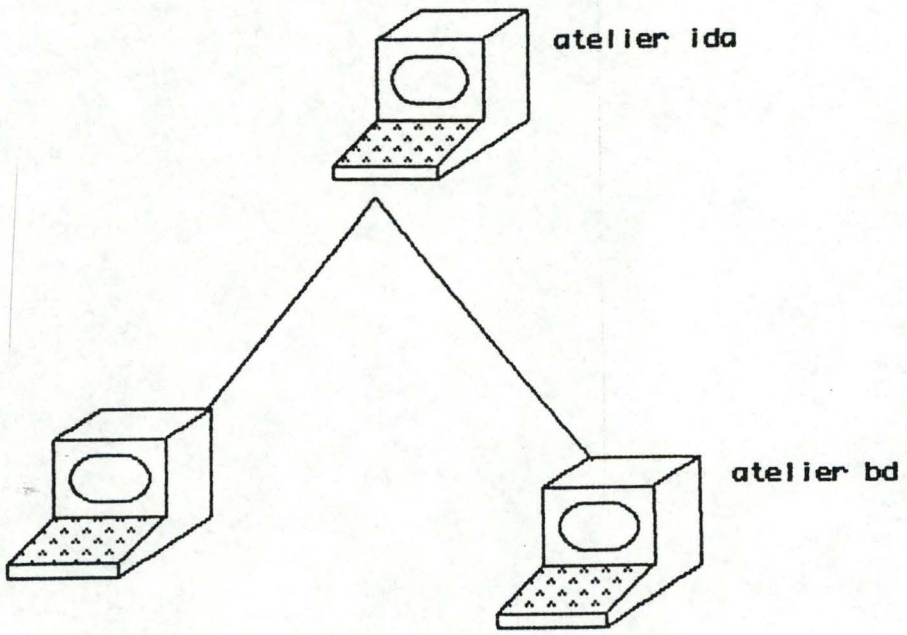


Figure 6.9 : schéma représentant le lien entre l'atelier ida et l'atelier bd

CONSULTATION

Tout schéma logique peut être parcouru afin de consulter les différents aspects de ses composants : type d'entités, type d'associations, attribut, identifiant, clé d'accès, type de chemins d'accès, descriptions textuelles, quantifications, structures physiques.

MODIFICATION DU SCHEMA

Le schéma présent dans la base des spécifications peut subir des modifications (modification de "conformité"). Cette fonction permet également d'augmenter le contenu sémantique et technique du schéma.

TRANSFORMATION DE STRUCTURES DU SCHEMA

Les constructions d'un schéma peuvent faire l'objet de transformations qui conservent la sémantique et les structure d'accès. Ces modifications "complexes" permettent d'atteindre la conformité à MDBS.

VERIFICATION DE LA CONFORMITE DU SCHEMA A UN STANDARD

Un schéma est évalué en ce qui concerne sa conformité aux structures permises par MDBS. Le résultat s'exprime par une liste d'avertissements.

PRODUCTION DE TEXTE MDBS(2)

Le schéma logique, une fois conforme MDBS, peut être traduit en un texte MDBS.

SCENARIO D'ACTIVATION DES FONCTIONS

Si on accepte les contraintes évidentes (on ne peut transformer qu'un schéma ayant fait l'objet d'une initialisation, on ne peut produire un texte MDBS que si toutes les structures du schéma sont conformes MDBS,etc.), les différentes fonctions pourront être activées dans un ordre quelconque, selon la logique de conception du concepteur lui-même et l'écart de conformité à annuler. La figure X.Y illustre les circuits possibles d'activation des fonctions.

1.- structures permises par des SGBD.
2.- code source MDBS destiné à être compilé.

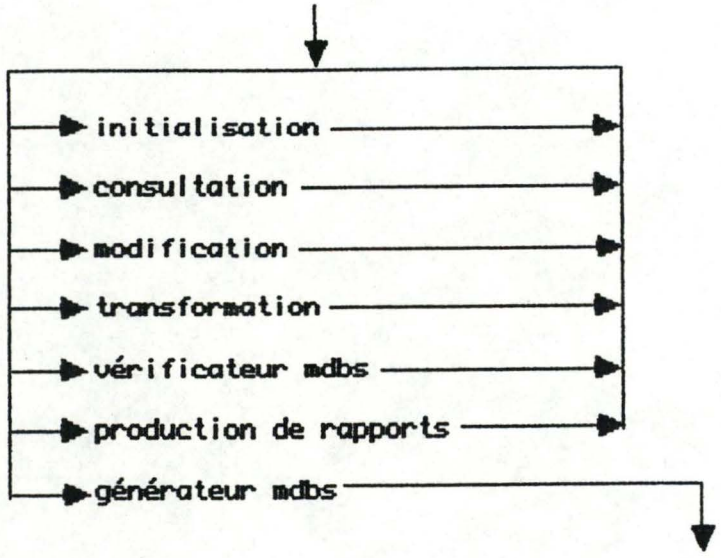


Figure 6.10 : activation des fonctions de l'atelier bd

VI.8 CONCLUSION

L'Atelier est avant tout un noyau dur constitué d'outils tels que le transformateur de schémas, le vérificateur de conformité, le producteur de rapport,etc. Ils sont d'application quelque soit l'environnement particulier (formalisme de la description conceptuelle, SGBD cible à atteindre,etc.). Ces outils peuvent coopérer au sens où ils partagent le même interface d'accès à la base des spécifications.

La périphérie de l'Atelier est formée des outils qui soit initialisent la base des spécifications (le chargeur), soit génèrent un code exécutable c-à-d la description d'un schéma en code source analysable par un compilateur (compilateur MDBS, par exemple). La description conceptuelle (disponible en amont du processus de conception) étant susceptible d'être supportée par différents modèles de données nécessitera d'instantier le chargeur. En aval, la structure de données propre au SGBD commercial exigera également le choix d'un générateur de code source (générateur MDBS, par exemple).

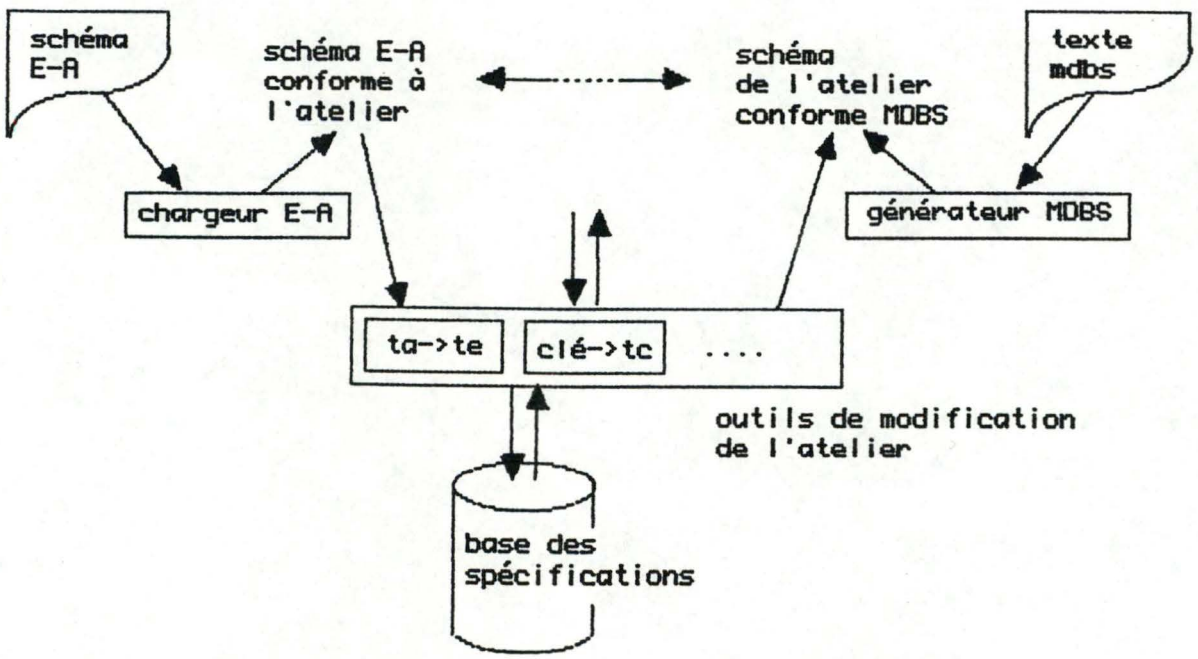


Figure 6.11 : gestion des schémas dans l'atelier

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR



CONTRIBUTION A L'ATELIER DE CONCEPTION DE BASES DE DONNÉES :

Extension au Système de Gestion de Bases de Données MDBS

P. Deville & P. Hoffelt VOLUME 2

PROMOTEUR : J.L. HAINAUT

MEMOIRE PRESENTE
EN VUE DE L'OBTENTION DU GRADE
DE LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1986 - 1987

C H A P I T R E VII

INTEGRATION DE MDBS DANS L'ATELIER BD

VII.1 INTRODUCTION

L'Atelier BD supporte la phase de conception d'une base de données. Pour remplir son rôle, l'Atelier utilise notamment les services du composant ACCES A LA BASE DES SPECIFICATIONS.

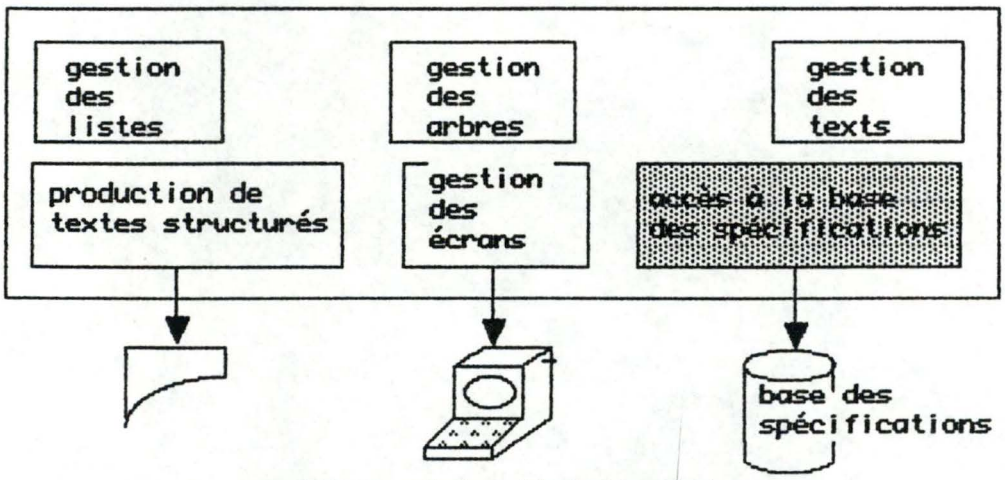


Figure 7.1 : l'atelier bd

Le composant ACCES A LA BASE DES SPECIFICATIONS présente deux interfaces au programmeur : les interfaces ADLC et SEM. Les couches ADLC et SEM sont des modules d'accès. Comme nous l'avons souligné au point (I.4), les modules d'accès permettent de s'abstraire des contingences liées à un SGBD réel pour obtenir un SGBD virtuel. A ce titre, ils offrent au programmeur un modèle de données, un ensemble de primitives permettant de gérer ces données ainsi que des règles d'enchaînement de ces primitives.

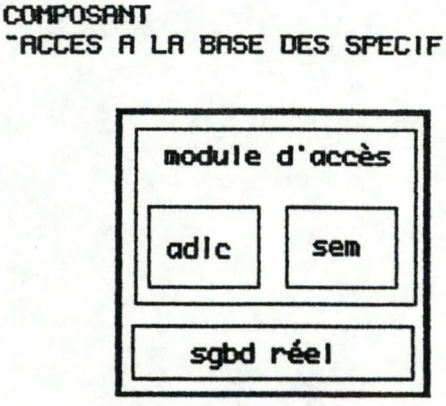


Figure 7.2

Les couches ADLC et SEM présentent les mêmes fonctionnalités. Elles se distinguent uniquement par la syntaxe d'appel.

La couche SEM offre un point d'accès unique. Les primitives de gestion des données sont prises en charge par une seule fonction C. Cette fonction est caractérisée par une liste de paramètres universels. En général, lorsqu'on exécute une primitive, une partie seulement des paramètres sont actifs.

exemple :

données :

1) code de la primitive (p.ex.: accès au suivant par chemin).
2) type d'articles demandé (p.ex.: commande).
3) type de chemins concerné (p.ex.: client-commande).
4) référence de l'article dont on désire le suivant (p.ex.: com1).
5) référence d'un article origine (p.ex: cli).

résultats :

1) l'article demandé (com2).
2) diagnostic (code de retour).

SEM (accès-au-suivant-par-chemin, commande, client-commande, com1, cli, com2, code-retour, xyz)

avec xyz est la suite des paramètres inactifs.

La couche ADLC est **orientée "programmeur"**.Elle prévoit une fonction pour chaque primitive. Les primitives ADLC ne comportent que les paramètres qui lui sont nécessaires.

exemple :

Nous reprenons l'exemple de la primitive d'accès par chemin au suivant.

NEXT_PATH (commande, com, cli, cli-commande)

Avant l'activation de la primitive, com référence l'article com1. Après l'exécution de cette dernière, com correspond à l'article com2.

Quels sont les éléments pour apprécier la qualité du composant ACCES A LA BASE DES SPECIFICATIONS ?

1- Il s'agit d'un composant de base. La majorité des programmes d'application utilise ses services. C'est pourquoi nous avançons qu'il doit atteindre un niveau de performance raisonnable sinon optimal.

2- Il ne peut accaparer une trop grande portion de la mémoire centrale car l'Atelier est destiné pour des petites machines (des micro-ordinateurs).

3- Il doit être facilement "maintenable".

L'équipe de l'Atelier BD a réalisé le composant ACCES A LA BASE DES SPECIFICATIONS. La réalisation des modules d'accès et du SGBD réel sont en effet à mettre à son actif. Désireuse d'améliorer le composant ACCES A LA BASE DES SPECIFICATIONS face aux critères susmentionnés, elle a envisagé une toute autre optique. C'est ainsi qu'une nouvelle version du composant est actuellement disponible. Dans le cadre de ce mémoire, nous avons coopéré à l'intégration du SGBD commercial MDBS en lieu et place du SGBD artisanal(1). Le composant ACCES A LA BASE DES SPECIFICATIONS se ramène ainsi à une simple couche d'interfaçage avec MDBS.

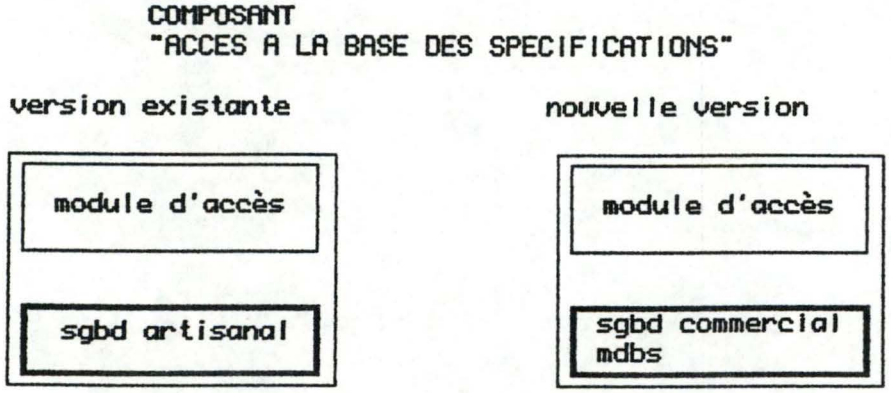


Figure 7.3

Le module d'accès ADLC fait l'objet des prochaines sections du chapitre. Dans la section 2, on mettra en évidence les concepts et les caractéristiques de ADLC avant de formuler ses spécifications. La section 3 présentera la version existante du composant ACCES A LA BASE DES SPECIFICATIONS tandis que les deux sections suivantes seront consacrées à l'intégration de MDBS dans l'architecture de l'Atelier. Nous clôturerons le chapitre par un examen du module d'accès SEM.

1.- L'implémentation de la nouvelle version a été effectuée sous la direction de la firme TRACTEBEL.

VII.2 LE MODULE D'ACCES ADLC

VII.2.1 LES CARACTERISTIQUES GENERALES

ADLC a été conçu pour gérer n'importe quelle base de données selon le modèle MAGRESTREINT(1). Pour que ADLC devienne un outil générique, il faudrait qu'il soit capable de fonctionner selon n'importe quel modèle de données.

VII.2.1.1 INDEPENDANCE PAR RAPPORT A UN SCHEMA

Dans le cadre de l'Atelier, le module d'accès est chargé des échanges de données entre un programme d'application (un outil) et la base des spécifications.

Il ne faudrait cependant pas en déduire que le module d'accès est irrémédiablement lié au susdit schéma. Le module d'accès est insensible aux modifications qui pourraient être apportées au schéma de la base des spécifications.

Pour fonctionner, il a besoin qu'on lui fournisse au préalable la description d'un schéma.

La description du schéma sur lequel travaille le module d'accès est stockée dans la base des spécifications. Comme ADLC examine fréquemment cette métainformation, celle-ci est chargée en mémoire centrale dans des tables. Il s'agit des **tables internes**. En outre, étant donné que la variété des accès au noyau des métainformations est restreinte, à chaque type de meta-requête, il correspond une primitive.
Les tables internes sont des objets complexes. C'est pourquoi, un outil a été développé pour générer automatiquement les tables internes.

Les modules d'accès indépendants d'un schéma particulier comme ADLC sont certes préférables aux modules d'accès liés à un schéma. Mais ils impliquent une gestion beaucoup plus conséquente. De ce point de vue, ils se rapprochent tout à fait d'un SGBD réel. Pour chaque primitive, ils doivent consulter une description de la base de données, vérifier la validité de la requête, repérer les arguments à communiquer au SGBD réel, puis arranger les résultats selon un format adéquat.

1.- Le modèle MAGRESTREINT a été défini au point (II.4.2).

VII.2.1.2 LE MODULE D'ACCES EST LIE A UNE CLASSE DE SGBD

Les modules d'accès de niveau 3 (1) et plus réalisent l'indépendance par rapport aux SGBD car ils perçoivent les données via le MAG. Le modèle sous-jacent au module d'accès n'étant qu'un sous-ensemble du MAG, nous concluons donc que ADLC n'est indépendant que par rapport à une classe limitée de SGBD.

VII.2.2 LES PRIMITIVES

Les caractéristiques des primitives dépendent généralement du modèle de données sur lequel elles opèrent. Néanmoins, on pourrait construire plusieurs classes de primitives pour un même modèle de données.

Nous présenterons donc d'abord les caractéristiques des primitives ADLC. Nous nous pencherons ensuite sur l'utilisation proprement dite des primitives ADLC.

CONSIDERATIONS D'ORDRE GENERAL

Les primitives du module d'accès ADLC restent assujetties aux types de données définis dans le schéma. Elles ne permettent pas l'extension des schémas ni la création temporaire (le temps de l'exécution d'un programme) de nouveaux types de données.

Les primitives d'accès ADLC livrent un seul article à la fois. L'utilisateur ADLC définit donc une séquence d'articles et il indique ensuite la position d'un article dans cette séquence. Pour parcourir les articles d'une séquence, l'utilisateur dispose de deux primitives :

- accès au premier d'une séquence.
- et - accès au suivant d'une séquence.

Cette dernière primitive est très puissante. L'utilisateur ADLC est en effet obligé d'indiquer l'article qui précède l'article qu'il veut obtenir. L'article spécifié ne correspond pas forcément au dernier article auquel il a accédé. De conclure que l'utilisateur ADLC définit lui-même ses courants.

1.- La notion de niveau pour un module d'accès a été présentée au point (I.4.4).

Les valeurs d'items associées à un article ne sont pas obtenues par une primitive spécifique. Elles sont disponibles aussitôt que l'accès à l'article a été effectué. On note cependant une exception pour les items de type texte.

Le module d'accès ADLC ne définit pas des primitives annexes (gestion de la concurrence et de la sécurité, prise de mesures,

VII.2.3 L'UTILISATION DES PRIMITIVES ADLC

Nous allons d'abord passer en revue les concepts spécifiques à l'utilisation des primitives ADLC. Nous traiterons ensuite des primitives de création.

LES CONCEPTS

Lorsqu'on invoque une primitive ADLC, on doit désigner les types d'articles et les types de chemins qui sont concernés.

Tout type d'objets du schéma est identifié par un nom ainsi que par un code numérique. Les types d'objets qui sont impliqués dans une primitive sont représentés par des codes numériques. Il est cependant plus facile pour un utilisateur ADLC de désigner les types d'objets par leur nom. C'est pourquoi on met à la disposition de l'utilisateur ADLC un fichier source C contenant pour chaque type d'objets une constante dont le nom est celui du type d'objets et dont la valeur est celle du code du type d'objets.

Il peut arriver que l'utilisateur ADLC ne sache pas déterminer a priori le type d'un objet. Pour s'en convaincre, citons le cas suivant :
l'utilisateur ADLC demande l'article cible d'un chemin N-1. Si le type du chemin est multi-cible, alors l'utilisateur ne sait pas dire à l'avance le type de l'article cible.

On met donc également à la disposition de l'utilisateur ADLC une constante spéciale , de nom RECORD, qui représente un type d'articles indéterminé.

Des valeurs d'items et des références d'articles sont échangées entre le programme d'application et le module d'accès ADLC. Cette communication sera assurée au moyen de variables items d'article et de variables de références.

Une **variable items d'article** est une variable destinée à contenir les valeurs d'items correspondant à un type d'articles déterminé. Elle est structurée en autant de composants qu'il y a d'items. De plus, chaque composant reçoit le nom d'un item.

Une **variable de référence** est une variable item d'article enrichie. Non seulement une variable de référence peut accueillir un article mais elle permet encore de référencer cet article.

Voici ses composants :

REF : référence d'un article.
RTYPE : code du type de l'article référencé par REF.
ILEN : longueur totale des valeurs d'items.
suite : mêmes composants que ceux apparaissant dans une variable d'items d'article.

Suivant la même philosophie que pour les codes numériques, le programme d'application dispose d'un fichier source qui contient la définition des types permettant de déclarer une variable de référence ou une variable items d'article.
Nous notons également qu'il existe un type spécial de variable de référence destiné aux articles dont le type est a priori inconnu.

Lorsqu'on crée un article ou lorsqu'on accède à un article, une variable de référence est garnie à partir des renseignements concernant cet article. Dans ce cas, on dira qu'un article est assigné à une variable de référence.

Les variables de référence ne peuvent pas contenir des items de type texte. En effet, les valeurs d'items de type texte sont constituées d'une chaîne de caractères de longueur a priori indéterminée. Quand on voudra créer ou modifier un article avec un ou plusieurs items de type texte, on désignera les objets texts (1) qui contiennent les valeurs de ces items. Lorsqu'on accède à un article, les valeurs des items de type texte ne sont pas automatiquement livrées. Elles doivent être demandées explicitement une à une.

A la terminaison d'une primitive ADLC, un code de diagnostic est renvoyé. Ce code informe l'utilisateur ADLC du bon déroulement de l'opération ou des raisons de son échec.

1.- Un objet text représente un texte en mémoire centrale. Les objets texts ont été décrits dans le point (II.4.2).

LA CREATION D'ARTICLES

Lorsqu'on crèe un article, il convient de spécifier un article origine pour chaque chemin obligatoire. L'article nouvellement crée sera automatiquement connecté aux articles origines stipulés.

Mais le nombre de types de chemins obligatoires peut varier d'un type d'articles à un autre, il s'ensuit donc que le nombre d'arguments d'une opération de création est lui-même variable. C'est pourquoi, les responsables de l'Atelier proposent deux jeux de primitives pour la création. L'un est indépendant du schéma, l'autre pas.

1) LES PRIMITIVES DE CREATION NON LIEES AU SCHEMA

Si l'on admet qu'un type d'articles est cible obligatoire d'au maximum MAX types de chemins, alors (MAX + 1) primitives de création doivent être dessinées. La première primitive concerne les types d'articles qui ne sont la cible obligatoire d'aucun chemin. La seconde primitive concerne les types d'articles qui sont la cible obligatoire d'un chemin. La troisième concerne les types d'articles qui sont la cible obligatoire de deux chemins. Et ainsi de suite ...

L'utilisation des primitives de création indépendantes du schéma présentent l'inconvénient suivant. A un concept exclusif (la création), il correspond (MAX + 1) primitives.

<u>exemple informel</u> :

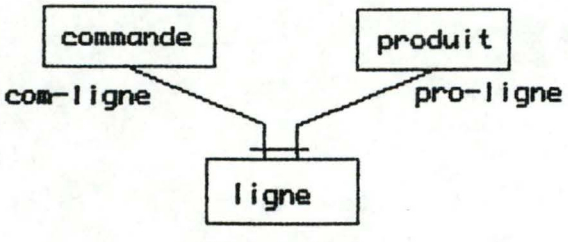


Figure 7.4

Considérons la création d'une ligne. Le type d'articles LIGNE est cible obligatoire de deux types de chemins.

```
CREATE2_art  ( type d'articles LIGNE ,
               une variable de référence garnie avec les
               valeurs d'items relatives au nouvel article
               ligne ,
               com-ligne ,
               une variable de référencée assigné à un
               article commande ,
               pro-ligne ,
               une variable de référence assignée
               à un article produit )
```

2) LES PRIMITIVES DE CREATION LIEES AU SCHEMA

Il existe une primitive de création pour chaque type d'articles présent dans le schéma de la base de données. L'intitulé des primitives indique le type de l'article crée.

exemple informel :

Reprenons l'exemple de la création d'une ligne.

```
CREATE_LIGNE  (
               une variable de référence garnie avec les
               valeurs d'items relatives au nouvel article
               ligne ,
               com-ligne ,
               une variable de référence assignée à un
               article commande ,
               pro-ligne ,
               une variable de référence assignée
               à un article produit )
```

Si l'on compare les deux types de primitives sur le plan de la validation, les primitives liées au schéma possèdent un net avantage par rapport aux primitives de création indépendantes du schéma. En effet, les primitives liées au schéma exigent que l'utilisateur spécifie un article origine pour chaque type de chemins obligatoire relatif au type d'articles crée. Si l'utilisateur se trompe de types de chemins ou de types de variables de référence, l'erreur sera détectée dès la phase de compilation. Par contre, quand on utilise une primitive indépendante du schéma, les erreurs ne sont détectées que lors de l'exécution de la primitive ADLC.

VII.2.4 L'ENVIRONNEMENT DU PROGRAMMEUR

Tout programmeur exploitant une base de données , organisée selon un quelconque schéma S, travaille avec un fichier d'environnement, lequel comprend trois parties :

1- la définition des constantes numériques relatifs aux types d'articles et de chemins dans la base de données.
2- la définition des types de variables de référence et de variables items d'article.
3- le texte des primitives de création liées au schéma.

L'existence de la notion "environnement du programmeur" mérite une justication plus fondamentale. Cette notion va surtout bénéficier au domaine de la validation. Muni d'un tel environnement, le programmeur risque moins de se tromper en ce qui concerne les arguments des primitives ADLC. En effet, un environnement se matérialise physiquement en un fichier à inclure dans le programme d'application. Dès lors, pendant le processus de compilation du programme, des controles judicieux pourront être exécutés. Répertorions les anomalies susceptibles d'être détectées lors de la phase de compilation.

1- le programmeur a mentionné une constante numérique ne figurant pas dans l'environnement.
2- concernant les primitives de création liées au schéma, le programmeur a stipulé une variable référence relative à un type d'articles incompatible avec la primitive invoquée.
3- le programmeur a omis d'employer une variable de référence.

La notion d' "environnement du programmeur" sous-tend un mécanisme de pré-validation. La faculté d'accomplir le processus de validation séparément de l'exécution du programme d'application présente un incontestable avantage sur le plan des performances. Les programmes d'application vont consommer un plus petit montant de ressource CPU.

L'Atelier dispose d'un outil pour générer l'environnement du programmeur.

VII.2.5 LES SPECIFICATIONS DE ADLC

Les primitives sont spécifiées selon la technique des pré- et post-conditions.

Nous avons adopté la convention suivante :
si un objet O est modifié, alors il sera noté O'.

1) LES PARAMETRES

ADBNAME :
adresse C d'une variable string (dbname) contenant le nom d'une base de donnée.

DBSTAT :
code de diagnostic.

ART :
code numérique d'un type d'articles (integer ou constante).

AREF, AREF1, AREF2 :
adresse C d'une variable integer dont le contenu peut jouer le rôle de référence.

AVREF, AVREF1, AVREF2, AVREFi :
adresse C d'une variable de référence; on appellera "article AVREF" l'article référencé par le constituant REF de la variable d'adresse AVREF.

CHEM, CHEMi, CHEMinv :
code numérique d'un type de chemins (integer ou constante).

CLE :
adresse C d'une variable string contenant une valeur de clé.

ATX :
adresse C d'un objet du type TEXT.

2) LES SPECIFICATIONS

Au cours de ces spécifications, nous nous sommes appuyés sur le concept de séquence. Pour rappel, une séquence est une collection ordonnée d'articles.
Nous utilisons les fonctions qui suivent.
Soit S une séquence
et A un article ,

first (S) = le premier article de S.
next (A , S) = l'article successeur de A dans S.

ACCES A UNE BASE DE DONNEE

=1= OPENDB (adbname)

args : dbname
res : dbstat
post : si pas d'erreur
alors :
la base de donnée désignée par dbname devient
la base de donnée courante.

erreurs possibles :
- la base de donnée désignée par dbname est déjà ouverte.

=2= CLOSEDB (dbname)

args : dbname
res : dbstat
post : si pas d'erreur
alors :
fermeture de la base de donnée courante.

erreurs possibles :
- la base de donnée désignée par dbname est déjà fermée.

ACCES SEQUENTIEL

=3= FIRST (art,avref)

args : art
res : vref, dbstat
post : soit s la séquence de tous les articles de type art.
si pas d'erreur
alors :
l'article avref = first (s).

=4= NEXT (art,avref)

args : art, vref.REF
res : vref', dbstat
post : soit s la séquence de tous les articles de type art.
si pas d'erreur
alors :
l'article avref' = next (l'article avref , s).

erreurs possibles :
- l'article avref n'est pas de type art.

ACCES PAR CHEMIN

=5= FIRST_PATH (art,avref1,avref2,chem)

args : art, vref2, chem
res : vref1, dbstat
post : soit s la séquence de tous les articles de type art qui sont cibles du chemin chem d'origine avref2.
si pas d'erreur
alors :
l'article avref1 = first (s).

erreurs possibles :
- l'article avref2 n'est l'origine d'aucun chemin de type chem.

=6= NEXT_PATH (art,avref1,avref2,chem)

args : art, vref1.REF, vref2
res : vref1', dbstat
post : soit s la séquence de tous les articles de type art qui sont cibles du chemin chem d'origine avref2.
si pas d'erreur
alors :
l'article avref1' = next (l'article avref1 , s).

erreurs possibles :
- l'article avref2 n'est l'origine d'aucun chemin de type chem.
- l'article avref1 n'est la cible d'aucun chemin de type chem.

ACCES PAR CLE DANS UN CHEMIN

=7= FIRST_PATH_KEY (art,avref1,avref2,chem,cle)

args : art, vref2, chem, cle
res : avref1, dbstat
post : soit s la séquence de tous les articles de type art qui sont cibles du chemin chem d'origine avref2 et qui ont cle comme valeur de clé d'accès dans ce chemin.
si pas d'erreur
alors :
l'article avref1 = first (s).

erreurs possibles :
- l'article avref2 n'est l'origine d'aucun chemin de type chem.

=8= NEXT_PATH_KEY (art,avref1,avref2,chem,cle)

args : art, vref1.REF, vref2, chem, cle
res : vref1', dbstat
post : soit s la séquence de tous les articles de type art qui sont cibles du chemin chem d'origine avref2 et qui ont cle comme valeur de clé d'accès dans ce chemin.
si pas d'erreur
alors :
l'article avref1 = next (l'article avref1 , s).

erreurs possibles :
- l'article avref2 n'est l'origine d'aucun chemin de type chem.
- l'article avref1 n'est la cible d'aucun chemin de type chem.

ACCES DIRECT

=9= DB_DIR (art,avref)

args : art, vref.REF
res : dbstat
post : si pas d'erreur
alors :
accès à l'article avref.

erreurs possibles :
- l'article avref n'est pas de type art.

LECTURE D'UNE VALEUR DE TEXTE

=10= GET-TXT (art,avref,atx)

args : art, vref
res : tx, dbstat
post : si pas d'erreur
alors :
l'item texte de l'article avref est rangé dans le texte référencé par atx.

erreurs possibles :
- l'article avref n'a pas d'item texte.

MODIFICATION DES DONNEES

=11= C_RECNAME (avref,avref,avref1, ...,avrefn,
chem1, ...,chemn,atx)

args : vref, vref1, ..., vrefn, chem1, ..., chemn, tx
pre : si l'article avref possède un item texte,
alors la valeur de cet item est rangé dans le texte
d'adresse atx.

res : BD', vref, dbstat
post : si pas d'erreur
alors :
cette fonction crée l'article avref.
cet article est inséré dans les chemins chemi d'origine
avrefi.

erreurs possibles :
- l'article avref n'est pas de type RECNAME.
- le type d'un article avrefi n'est pas reconnu comme
type d'articles origine pour chemi.

commentaires :
C_RECNAME est une primitive liée à un schéma.

=12= CREATEi_ART (art,avref,avref1,...,avrefi,
chem1, ...,chemi,atx)

args : art, vref, vref1, ..., vrefi, chem1, ..., chemi, tx
pre : si l'article avref possède un item texte,
alors la valeur de cet item est rangé dans le texte
d'adresse atx.

res : BD', avref, dbstat
post : si pas d'erreur
alors :
cette fonction crée l'article avref.
cet article est inséré dans les chemins chemi d'origine
avrefi.

erreurs possibles :
- l'article avref n'est pas de type art.
- l'ensemble des chemins obligatoires relatifs au type
d'articles art n'est pas inclus dans
(chem1, ...,chemi).
- le type d'un article avrefi n'est pas reconnu comme
type d'articles origine pour chemi.

commentaires :
CREATEi_ART est une primitive d'usage général c-à-d non
liée à un schéma.

=13= PUT_TXT (art,avref,atx)

args : art, vref, tx
pre : l'article avref est de type art.

res : BD', vref', dbstat
post : si pas d'erreur
alors :
la valeur de l'item texte de l'article avref équivaut désormais au texte référencé par atx.

erreurs possibles :
- l'article avref ne comprend pas d'item de type texte.

=14= DELETE (art,avref)

args : art, vref
res : BD', dbstat
post : si pas d'erreur
alors :
destruction de l'article avref ainsi que des articles qui en dépendent, directement ou non, via des chemins obligatoires.

=15= MODIFY (art,avref)

args : art, vref
res : BD', vref', dbstat
post : si pas d'erreur
alors :
les valeurs d'items de l'article avref sont remplacées par les valeurs présentes dans la variable avref. Les items jouant un rôle de clé ou d'identifiant ne sont cependant pas modifiés.

=16= MODIKO (art,avref)

args : art, vref
res : BD', vref', dbstat
post : si pas d'erreur
alors :
les valeurs d'items de l'article avref sont remplacées par les valeurs présentes dans la variable avref. Seuls les items jouant un rôle de clé ou d'identifiant sont modifiés.

=17= ATTACH (art,avref1,avref2,chem)

args : vref1, vref2, chem.
res : BD', dbstat
post : si pas d'erreur
alors :
l'article avref1 est cible du chemin chem d'origine avref2.

erreurs possibles :
- chem ne représente pas un type de chemins 1-N.
- le type de l'article avref1 n'est pas reconnu comme type d'article cible de chem.
- le type de l'article avref2 n'est pas reconnu comme type d'articles origine de chem.

commentaires :
comme un chemin inverse (CHEMinv) est associé à CHEM, il s'ensuit que l'article avref2 est la cible du chemin CHEMinv d'origine avref1.

=18= DETACH (avref,chem)

args : vref, chem
res : BD', dbstat
post : si pas d'erreur
alors :
l'article avref n'est relié à aucun article via chem.

erreurs possibles :
- chem ne représente pas un type de chemins 1-N.
- le type de l'article avref n'est pas reconnu comme type d'articles cible de chem.

VII.3 LA VERSION ACTUELLE DU COMPOSANT "ACCES A LA BASE DES SPECIFICATIONS"

VII.3.1 INTRODUCTION

Notre but n'était pas de procéder à un examen minutieux de la version existante du composant ACCES A LA BASE DES SPECIFICATIONS. Nous nous sommes bornés à en relever les lignes maîtresses.

L'architecture du composant actuel regroupe quatre couches.

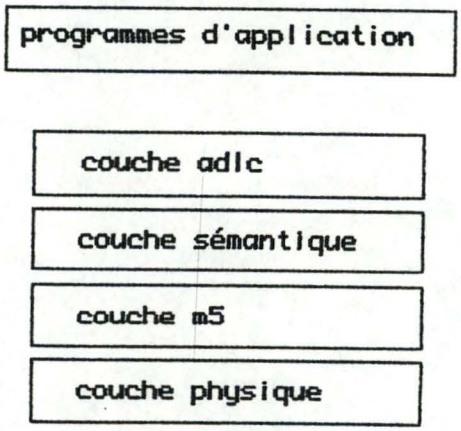


Figure 7.5

Les interfaces ADLC et SEMANTIQUE rassemblent respectivement les primitives ADLC et SEM. Lesdites primitives opèrent sur un schéma externe de type MAGRESTREINT.
Les primitives de l'interface M5 s'appuient sur le schéma interne de la base de données (le schéma M5).
Les primitives de l'interface physique réalisent les fonctions d'accès classiques à des fichiers (DBC).

VII.3.2 LA COUCHE PHYSIQUE

L'objectif de la COUCHE PHYSIQUE est l'accès et la manipulation des données mémorisées dans la base. Cette couche utilise les services d'un SGBD rudimentaire, en l'occurrence DBC (1).

1.- DBC est une variante de DBASE-III adaptée au Lattice.

Les primitives renfermées dans cette couche supportent les opérations suivantes sur les fichiers :

- ouverture et fermeture.
- lecture séquentielle.
- accès à un record dont on donne l'identifiant interne.
- création d'un record.
- mise à jour d'un record.
- suppression d'un record.

DBC se révèle notamment insuffisant en ce qui concerne la problématique des courants (les "currency indicators" selon la terminologie CODASYL).

DBC admet simplement la notion de courant par rapport à un fichier. Or, le programmeur d'application souhaiterait consacrer plusieurs courants par fichier. Un composant "gestion des courants" a donc été conçu pour pallier la carence du SGBD.

VII.3.3 LA COUCHE M5

La vocation du module M5 est de camoufler les aspects techniques inhérents à la couche physique.

Le module M5 constitue un SGBD virtuel basé sur un modèle de données intitulé du même nom. Le modèle M5 est mono-schéma. Mais il est suffisamment général pour pouvoir exprimer n'importe quel schéma MAG. La présence de la couche M5 permet d'isoler les couches supérieures d'un changement de SGBD.

LE MODELE M5

Le modèle répertorie cinq éléments fondamentaux :
object, object0, object1, object2 et text.

Un text est obligatoirement lié à un object. Un text possède un type. La juxtaposition des lignes (lines) extraites des texts relatifs à un même groupe de texts (text-group) restitue un texte original pour autant que les lignes soient triées par ordre croissant des numéros de séquence (text-seq).

Un object possède un type et contient des data. Les objects sont reliés entre eux par des chemins. Un object0 est un object lié à un autre object mais de façon non obligatoire. Un object1 est un object obligatoirement lié à un autre object. Et, un object2 est un object obligatoirement lié à deux autres objects.

Les deux types de chemins aboutissant respectivement aux object0 et 1 se prénomment R0 et R1. R21 et R22 sont les chemins aboutissant aux object2.

Dans la représentation graphique du modèle ci-jointe, on recourt à la relation de spécialisation "is-a" issue de la théorie des réseaux sémantiques. Les objects 0, 1 et 2 héritent donc des propriétés caractéristiques d'un object ainsi que de ses texts.

Signalons enfin les clés d'accès key0 et key1 dans les types de chemins R0 et R1.

VII.3.4 LA COUCHE SEMANTIQUE

Le but de la couche sémantique est de réaliser un SGBD virtuel capable de manipuler n'importe quel schéma MAGRESTREINT.

La couche sémantique fait appel aux services du module M5. La couche sémantique doit donc assurer la transformation des requêtes exprimées dans les primitives SEM en des requêtes conformes au module M5. Pour ce faire, elle exploite les tables dites de conversion qui contiennent la description d'un schéma particulier MAGRESTREINT dans les termes du modèle M5.

VII.3.5 LA COUCHE ADLC

L'interface ADLC constitue un simple habillage de l'interface SEM.

VII.4 LA NOUVELLE VERSION DU COMPOSANT "ACCES A LA BASE DES SPECIFICATIONS"

Dans la section précedente, le lecteur a pu prendre connaissance de l'architecture du composant ACCES A LA BASE DES SPECIFICATIONS utilisé par les outils de l'Atelier BD. Dans le cadre du présent mémoire, nous avons réalisé une nouvelle version du composant ACCES A LA BASE DES SPECIFICATIONS. Derrière cette nouvelle version, se cache un nouveau SGBD réel : le SGBD commercial MDBS.

MDBS appartient à la famille des SGBD CODASYL. Dans l'absolu, ce sont les SGBD relationnels qui prévalent à l'heure actuelle. Mais l'implantation de ces SGBD est trop complexe pour pouvoir être envisagée dans le contexte d'un micro-ordinateur. MDBS est néanmoins un produit beaucoup plus élaboré que la plupart des gestionnaires de données disponibles sur micro-ordinateur, ces derniers n'étant bien souvent que de simples gestionnaires de fichiers.

Les sous-sections suivantes présentent les différents problèmes relatifs au processus d'intégration du SGBD MDBS dans le composant ACCES A LA BASE DES SPECIFICATIONS.

En toute généralité, un module d'accès n'est pas forcément lié au modèle de données relatif au SGBD réel. Un module d'accès peut simplifier ou enrichir le modèle de données du SGBD réel.

Le module d'accès ADLC perçoit les données via le modèle MAGRESTREINT. Il s'agira donc d'établir un **mapping** entre les objets d'un schéma MAGRESTREINT et ceux d'un schéma MAGRESTREINT rendu conforme au modèle MDBS. A l'occasion, certaines structures de données MAGRESTREINT non admises par le modèle MDBS devront faire l'objet d'une **émulation**.

On dira que le module d'accès établit une émulation du modèle des données s'il enrichit les structures de données du SGBD réel. L'émulation peut concerner aussi bien les structures de données que le domaine des contraintes d'intégrité.

Les deux premières sous-sections seront respectivement consacrées au mapping et à l'émulation. La troisième sous-section sera consacrée à la gestion des courants.

VII.4.1 LE MAPPING

Les correspondances entre types d'objets appartenant à des niveaux différrents sont exprimées dans des mappings.
On relève ainsi deux niveaux de mapping dans l'architecture ANSI/SPARC :

- le mapping entre les schémas interne et conceptuel.
- le mapping entre les schémas conceptuel et externes.

Si une modification affecte le schéma i, alors un ajustement du mapping s'avère nécessaire de telle sorte que le schéma i+1 demeure inchangé.

1) LE MAPPING

Supposons que toute l'information contenue dans une base de données soit représentée sous forme de records.

On peut voir le mapping comme un type d'associations entre deux ensembles de records appartenant aux niveaux i et i+1.
Les types d'associations 1-1 relèvent de mappings triviaux. Les types d'associations 1-N et N-1 revèlent des mappings plus complexes. Avec les types d'associations N-N, on atteint le plus haut niveau de complexité. Dans la suite, nous parlerons de mappings 1-1, 1-N et N-N.

Certains mappings s'expriment aussi au niveau des data items. A ce sujet, notons le cas intéressant des data items virtuels. L'information élémentaire relative à un data item virtuel n'est pas stockée physiquement. Elle est dérivée à partir d'autres data items.

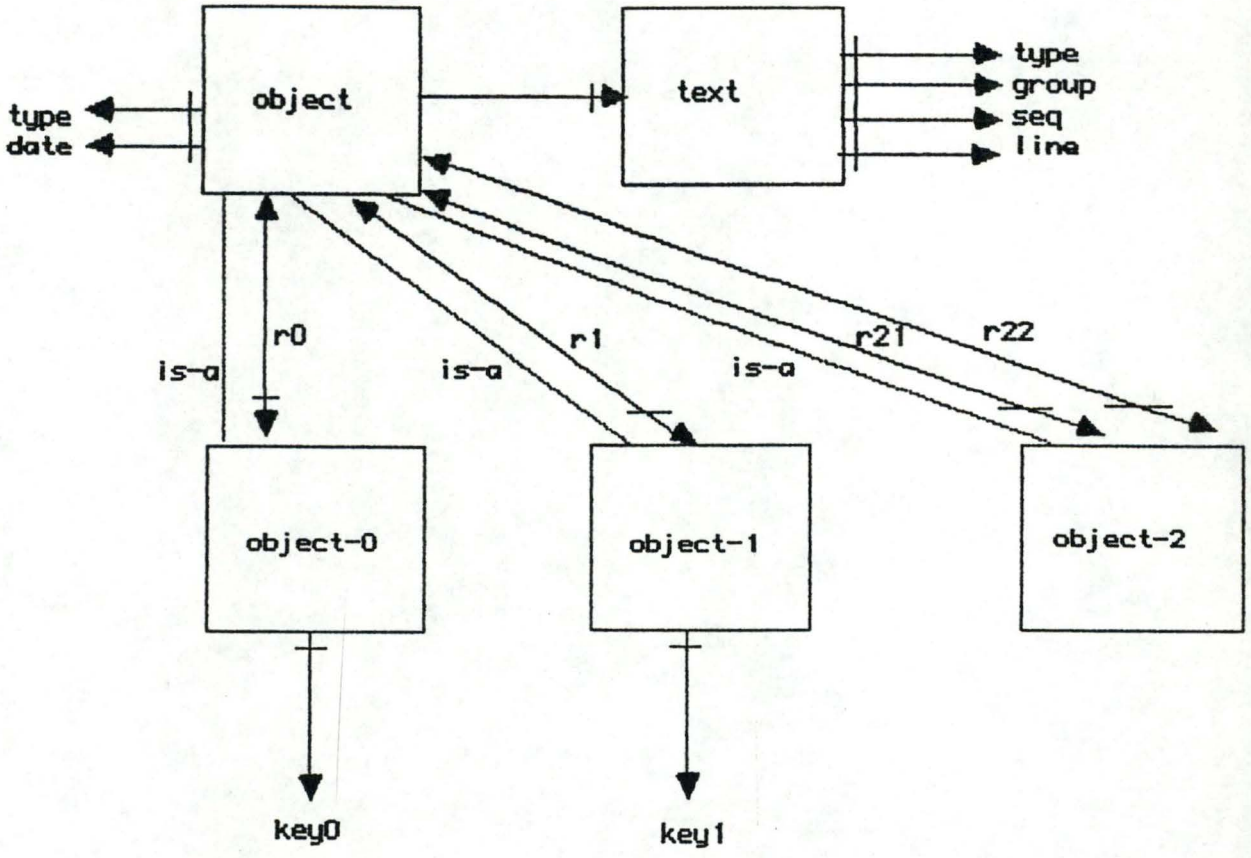

object
text
type
group
seq
line
type
date
r0
r1
r21
r22
is-a
is-a
is-a
object-0
object-1
object-2
key0
key1

Figure 7.6

Les "bit maps" constituent une méthode attractive pour représenter un mapping. Un bit map consiste en une table de type booléen.

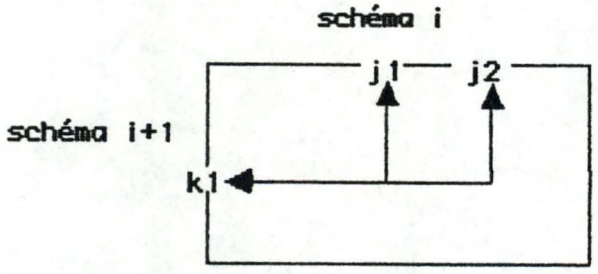


Figure 7.7

Les indices de colonne et de ligne référencent respectivement des objets du schéma i et du schéma i+1. Deux modes d'utilisation sont prévus :

- le "descending mode" qui correspond à la conversion d'un objet du schéma i en un ou plusieurs objets du schéma i+1.
- le "ascending mode" qui correspond à la conversion d'un objet du schéma i+1 en un ou plusieurs objets du schéma i.

2) LES MAPPINGS DANS LE CADRE DU MODULE D'ACCES

Nous venons de traiter la notion de mapping dans le contexte particulier de l'architecture à trois niveaux proposée par le groupe ANSI/SPARC. Le mapping est cependant un mécanisme général s'appliquant à n'importe quelle paire de schémas.

Le module d'accès réalise un pont entre deux modèles de données : le modèle MAGRESTREINT et le modèle MDBS. La conception du module d'accès implique donc que l'on définisse un mapping entre ces deux modèles.

Les mappings entre objets du niveau MAGRESTREINT et les objets du niveau MDBS sont majoritairement des mappings 1-1.

2.1) LE NOM DES ARTICLES

A tout article MAGRESTREINT, il correspond un et un seul article MDBS. On est donc en présence d'un mapping 1-1.

Les contraintes régissant l'octroi d'un nom à un type d'articles sont plus laxistes au niveau MAGRESTREINT qu'au niveau MDBS (1). Les noms MAGRESTREINT ne sont pas forcément égaux aux noms MDBS. Il existe plusieurs procédés pour obtenir le nom MDBS d'un type d'articles :

1- Constituer une table associant le nom MDBS et le nom MAGRESTREINT de chaque type d'articles.

2- Les types d'articles MAGRESTREINT sont identifiés par un code. Lors de la définition du schéma MDBS, il suffirait de spécifier pour chaque type d'articles, en plus de son nom, un synonyme. Ce synonyme engloberait la mention du code MAGRESTREINT relatif au type d'articles concerné.

1.- Un nom MDBS ne peut pas dépasser huit caractères.

2.2) LES TEXTES

A tout item texte MAGRESTREINT, il correspond en MDBS un agrégat d'objets. On est donc en présence d'un mapping 1-N.

Considérons les types d'entités "procédure" et "description" du schéma de l'atelier.

Une procédure est caractérisée par une date de dernière modification, un type (par exemple, prédicatif), un état (par exemple, achevé ou en cours) et un voire plusieurs textes algorithmiques. Ces textes feront l'objet de raffinements successifs (algorithmes prédicatifs, effectifs et effectifs conformes au SGBD cible).

En ce qui concerne l'objet description, nous allons citer trois utilisations possibles de cet objet :

1) ENONCE DE CONTRAINTES D'INTEGRITE INFORMELLES.

Prenons un exemple. Supposons que le concepteur a spécifié une classe fonctionnelle 1-1 pour tel type de chemins.
D'autre part, le SGBD dont dispose le concepteur est un SGBD CODASYL. Or, les set-types correspondent à un type d'associations 1-N du type owner vers le type member. Il se produit donc une dégradation de contrainte d'intégrité. L'Atelier prend acte de cette dégradation dans un objet description.

2) LES DEFINITIONS.

Les définitions des types d'entités et des types d'association sont emmagasinnées dans des objets descriptions.

3) LES COMMENTAIRES LIBRES.

Le concepteur justifie ses décisions dans ces commentaires libres.

Les types d'entités procédure et description ont donc été flanqués d'un attribut de type texte.

Rares sont les SGBD qui autorisent les items de type texte. Et, MDBS n'échappe pas à cette règle. La raison d'un tel choix est simple. Elle est liée à la caractéristique essentielle d'un texte. En l'occurrence, la longueur d'un texte est a priori indéterminée.

Deux procédés sont généralement envisagés pour résoudre le problème de la gestion des textes.

MDBS admet des items de type caractère de longueur fixe. Ces items sont appelés items strings.

1) SOLUTION RUDIMENTAIRE.

Cette solution consiste à fixer une taille maximale pour un attribut de type texte (TMAX). Ainsi, un attribut texte prend la forme d'un string dont la longueur est égale à TMAX. Cette solution manque assurément de souplesse. De deux choses l'une, soit le texte à stocker est peu volumineux et il est en dessous de la limite TMAX, soit il est volumineux et il dépasse la limite TMAX. On observe donc invariablement soit un gaspillage de la ressource mémoire, soit une incapacité de stocker un texte complet.

Aussi, nous avons opté pour une autre solution.

2) SOLUTION ELABOREE.

Un texte est une suite de caractères. Un texte peut être décomposé en un certain nombre de portions disjointes. Ces portions ont toutes la même taille. Les portions reçoivent le nom "ligne" bien qu'elles n'aient évidemment aucune signification grammaticale. Chaque ligne possède un numéro correspondant à son ordre d'apparition dans le texte.

Une telle modélisation suggère une représentation originale pour un attribut de type texte. Un attribut de type texte est implémenté par un item string répétitif. La répétitivité requise est variable et illimitée.

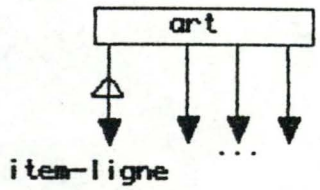


Figure 7.8

Comme MDBS admet seulement la répétitivité fixe, il s'avère nécessaire de modifier la structure de données de la figure 7.9. Pour ce faire, nous utilisons une transformation théorique décrite dans (1).

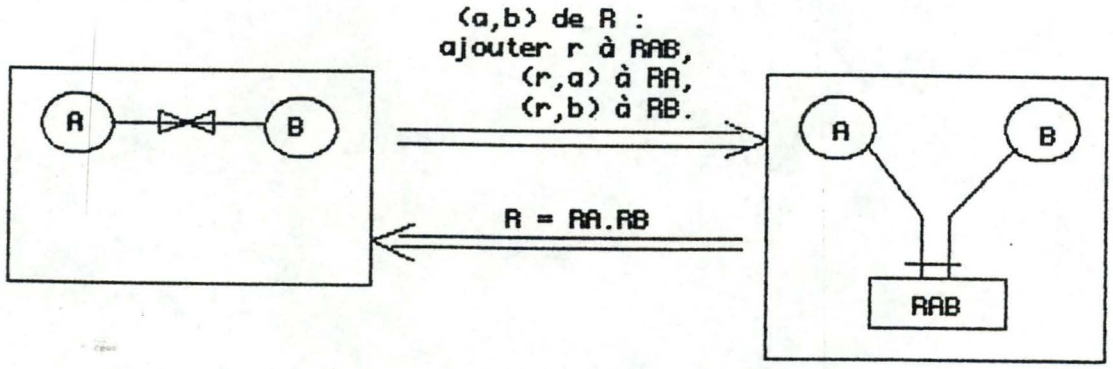


Figure 7.9

Cette transformation permet d'éliminer un type d'associations N-N en un type d'articles et en deux types d'associations. Elle est applicable en particulier pour les types d'associations 1-N.
L'élimination du type d'associations art/item-ligne conduit à la structure de données décrite dans la figure 7.10.

1.- J.L. Hainaut, Conception assistée des applications informatiques, 2. Conception de la base de données, MASSON, 1986.

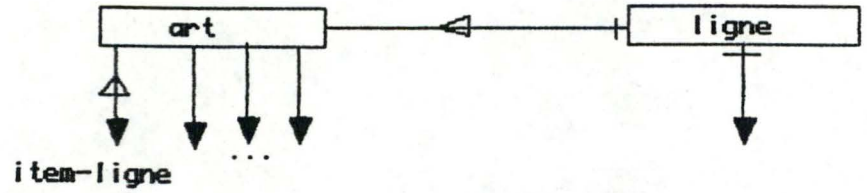


Figure 7.10

Selon cette solution, un texte à stocker est réparti sur un certain nombre d'articles lignes. Nous avons adjoint un item supplémentaire au type d'articles ligne. Il s'agit d'un item technique destiné à informer le module d'accès sur la longueur du texte stocké.

L'ordre des articles members d'un set-type n'est pas indifférent. Les articles members sont classés par ordre croissant du numéro de ligne. La variété des clauses MDBS permet d'envisager plusieurs implémentations de cette contrainte. Il existe notamment la clause d'insertion d'un member dans un set. Dans cette clause, on spécifie où ranger le nouveau member dans le set courant.

1) On adjoint un item contenant le numéro de ligne. Et, on déclare le set trié. Le tri est effectué sur base de l'item susmentionné.

   MEMBER ORDER CLAUSE
     order is sorted by ascending numero-ligne.

   Nous avons opté pour une solution qui ne réclame pas de clé de tri.

2) Nous avons sélectionné le mode d'insertion chronologique. Autrement dit, l'article ligne nouvellement crée est inséré en fin de set. Le stockage d'un texte dans la base de données, bien que recouvrant plusieurs opérations SGBD, constitue une opération logique ou selon la terminologie adoptée une transaction. Lors de ladite transaction, on stocke d'abord la ligne 1, puis la ligne 2, et ainsi de suite.
   C'est pourquoi le mode d'insertion next se révèle suffisant.

   MEMBER ORDER CLAUSE
     order is next.

Ci-après, nous jugeons la "solution élaborée" sous l'angle strict des performances.

1) LA PLACE MEMOIRE

On fait un usage rationnel de la mémoire secondaire. Seul l'espace du dernier item-ligne n'est pas totalement rempli. Son taux de remplissage sera en moyenne de 50%.

2) LES ACCES

L'accès à un item texte requiert plusieurs accès logiques. En plus de l'accès à l'article art, il faut compter les accès aux articles lignes.
Le calcul du nombre d'accès s'établit comme suit :

nombre_accès = 1 + nombre d'articles lignes accaparés par le texte.

La longueur d'un item ligne est un paramètre important. Si ce paramètre est peu élevé, alors on privilégie l'espace mémoire par rapport aux accès. Dans le cas inverse, le nombre d'accès est réduit au détriment de la place mémoire.

2.3) LES CHEMINS ET LES SETS

A tout set MDBS, il correspond en MAGRESTREINT deux chemins. On est donc en présence d'un mapping N-1.

Un type de chemins implique deux types d'articles (un type d'article origine et un type d'article cible) :
étant donné un article origine, le chemin conduit vers les articles cibles associés. Un type de chemins est donc un mécanisme d'accès unidirectionnel.

Les sets MDBS offrent systématiquement des accès bidirectionnels. On peut aussi bien accéder au(x) article(s) owner(s) d'un set qu'à ses articles members.

A un set MDBS, il correspond en MAGRESTREINT deux types de chemins C1 et C2 tels que C2 est l'inverse de C1.
A un chemin MAGRESTREINT, il correspond un seul set MDBS.

Pour déterminer le nom du set-type relatif à un type de chemins donné, on utilise une des techniques qui a été préconisée pour le mapping des articles. Le fait qu'un set MDBS offre un mécanisme d'accès bidirectionnel ne pose pas de problème. En l'occurrence, si un chemin est de classe fonctionnelle 1-N, alors un accès owner/member est entrepris. Par contre, si un chemin est de classe fonctionnelle N-1, alors un accès member/owner est de mise.

VII.4.2 L'EMULATION

Les propos que nous allons tenir s'appliquent à tous les SGBD qu'ils soient basés sur le modèle relationnel, le modèle hiérarchique ou le modèle réseau.

Dans une démarche de conception d'une architecture de module réalisant les spécifications ADLC, l'abstraction consiste à pouvoir décrire à différents niveaux (logique et physique) la structure de la base de données en projet.

L'émulation sera pour nous une technique qui permet de résoudre deux types de problèmes :

1) Les concepts du niveau logique n'ont pas toujours une représentation directe au niveau physique.

exemple :

les items textes ne sont pas définis au niveau physique. L'émulation "item texte" permettra le stockage et la mise à jour de ces items.

2) Le deuxième type de problèmes concerne le maintien des contraintes d'intégrité. Une requête du module d'accès peut aller à l'encontre de certaines contraintes d'intégrité. Le module d'accès pourrait néanmoins satisfaire cette requête. Mais dans ce cas, il devra effectuer des mises à jours secondaires afin que la base de données reste dans un état cohérent. On étudiera le cas typique du delete en cascade. Dans cette forme de delete, en plus de la destruction de l'article spécifié dans la requête, on effectue des destructions secondaires. Ces destructions secondaires portent sur les articles cibles obligatoires de l'article spécifié. A noter que les destructions secondaires sont elles-mêmes des destructions en cascade.

Comme la problématique des primitives émulées entretient un lien direct avec le concept d'intégrité, nous allons commencer par un exposé général sur les contraintes d'intégrité. Nous serons alors en mesure de présenter la problématique de l'émulation dans le contexte du module d'accès ADLC. Et, nous terminerons par un exposé des différents types d'émulation qui s'imposent dans le cadre du module d'accès ADLC.

1) LES CONTRAINTES D'INTEGRITE

"Une contrainte d'intégrité est une propriété que les données d'une base de données doivent vérifier à tout instant de manière à décrire le plus précisément possible le monde réel".

Nous ne retiendrons cependant pas cette acception de l'intégrité car elle est trop riche.

Le modèle des données contenues dans une base de données est un système purement formel, une abstraction du monde réel.
A un modèle est toujours associé une certaine sémantique qui établit la connexion entre les composants du monde réel et ceux du modèle. D'aucuns parlent de l'interprétation du modèle. Ainsi si l'on prend le modèle E-A, les types d'entités et les types d'associations reçoivent une définition.

exemple :

Type d'entités CLIENT : un client est une personne morale ou physique qui, dans les cinq dernières années écoulées, a passé une commande à la firme X.

Dans cette optique, le jeu optimal des contraintes d'intégrité que l'on puisse imaginer est celui constitué de l'ensemble des définitions entourant le modèle de données.
La vérification de ces contraintes d'intégrité n'est pas entièrement automatisable. Un ordinateur ne saurait pas en effet établir de facto que telle personne est cliente de la firme.

On se tourne donc vers une acception plus restreinte du concept d'intégrité.
" Une contrainte d'intégrité est une propriété formelle que les données doivent vérifiées à tout instant ".

L'intégrité formelle est une condition nécessaire mais non suffisante de l'intégrité d'une base de données.
Un ordinateur peut vérifier qu'une personne ne peut pas être assimilée à un client de la firme car cette dernière n'est rattachée à aucune commande. Mais il ne sait pas détecter par exemple qu'une commande farfelue a été affectée à un client.

Lors du processus de définition d'une base de données, un spécialiste en analyse conceptuelle définit une série de contraintes.
Lorsque l'on est en phase d'exploitation,le système contrôle les interactions de l'utilisateur avec la base de données afin de s'assurer que les contraintes sont bien respectées.
Notons que les SGBD ne supportent que certains types de contraintes (classe fonctionnelle, contraintes d'existence, identifiant,...). Il s'agit de contraintes d'intégrité privilégiées.
La vérification des autres contraintes est assurée par des programmes d'application.

Quelle est la réaction du système, lorsqu'il intercepte une interaction illicite ?
Deux solutions sont envisagées sur le plan théorique :

1) SOLUTION 1

Le système refuse d'exécuter l'opération spécifiée par l'utilisateur. Et,ainsi l'intégrité de la base est préservée. En pratique,les SGBD adoptent cette solution.

2) SOLUTION 2

Le SGBD accepte l'opération , mais il accomplit concomitamment des opérations supplémentaires dont l'effet global est parfaitement légal sur le plan de l'intégrité. Ces opérations sont des **pérations de compensation.**

Considérons une base de données décrite par le schéma de la figure 7.11.

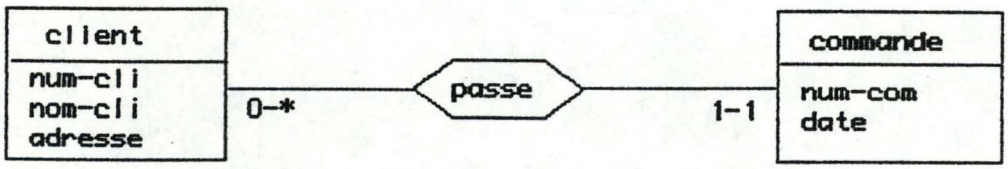


Figure 7.11 : schéma "client-commande"

Supposons qu'une requete utilisateur spécifie la destruction du client C1.

1) SOLUTION 1

On rejette l'opération utilisateur car le client C1 a passé N commandes. Ainsi, on évite l'existence d'un groupe de commandes non associées à un client.

2) SOLUTION 2

La solution 2 est de loin la plus séduisante dans le cas sous revue. Le système procède à la destruction du client C1 ainsi qu'à la destruction de ses commandes.

Dans la première solution, si l'utilisateur veut détruire un client, le système le force implicitement à demander au préalable la destruction des N commandes associées.
L'utilisateur mentionne donc une séquence de N+1 opérations dans son programme (une opération de destruction client, et N opérations de destruction commande).
Comme cette séquence d'opérations constitue une opération logique, l'utilisateur va lui conférer le statut de transaction.
L'opération de destruction client proposée dans la solution 2 est équivalente à la transaction susmentionnée.

En conclusion, l'utilisateur placé dans le contexte de la solution 2 agit directement avec des primitives de haut niveau, tandis que son homologue placé dans le contexte de la solution 1 doit créer artificiellement ces primitives de haut niveau.

2) LA PROBLEMATIQUE DE L'EMULATION DANS LE CONTEXTE DU MODULE D'ACCES ADLC

Les chercheurs de l'atelier manipulent la base de données avec les primitives de haut niveau de l'interface ADLC.

Dans certains cas, les correspondants de ces primitives dans le DML proposé par MDBS n'ont pas toujours la même puissance ! Une émulation s'avère alors indispensable.

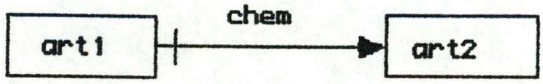
Deux raisons principales expliquent ce manque de puissance.

1) La première raison évidente est que certaines constructions du modèle MAGRESTREINT ne sont pas admises par le modèle sous-jacent à MDBS.
Nous citons le cas des "items textes" traité dans le point (VII.4.2). Il apparaît qu'un objet " item texte " dans un schéma MAGRESTREINT est simulé par un agrégat d'objets dans le schéma conforme MDBS.
D'où il découle qu'une opération de manipulation de texte ADLC nécessite une séquence d'opérations MDBS.

2) MDBS ignore certaines contraintes d'intégrité privilégiées telles que les chemins obligatoires. A cet égard, nous envisageons une double émulation .
D'une part, il faut fournir aux primitives ADLC le moyen de déterminer si un type d'articles donné est cible obligatoire dans tel type de chemins.
D'autre part, certaines primitives spécifiques doivent être capables de réaliser des opérations de compensation.

3) LES DIFFERENTS TYPES D'EMULATION DANS LE CADRE DU MODULE D'ACCES ADLC

3.1°) LES CHEMINS OBLIGATOIRES



La transposition d'un chemin 1-N obligatoire dans les concepts CODASYL correspond à un set-type où
1- le mode d'insertion du member est "automatique".
2- le mode de rétention est "mandatory".

1) LE MODE D'INSERTION

MDBS permet le mode d'insertion automatique. Cette caractéristique est bien utile lors de la création d'un article art2. En effet, l'article nouvellement crée sera directement connecté aux origines courantes des chemins obligatoires.

2) LE MODE DE RETENTION

MDBS permet deux modes de rétention :
- le mode fixed
- le mode optional.

MDBS ne définit donc pas le mode mandatory ! Et, le mode fixed est trop sévère. Un article art2 n'est pas forcément lié au même article art1 durant toute sa vie. Ainsi un chemin qu'il soit obligatoire ou facultatif sera déclaré optional.

C'est le module d'accès qui doit veiller à ce que les articles art2 restent connectés à un article art1.

3.2°) L'OPERATION DE DESTRUCTION

Un rapport émanant du DBTG (1971) prévoit quatre primitives de suppression d'article.
L'argument d'une primitive delete est le courant du Run-Unit.

1) LE DELETE SANS QUALIFICATION

Si le courant du Run-Unit est l'owner d'aucun set non vide, alors :

1- on le dissocie des sets où il est member
et 2- on le supprime.

2) LE DELETE PERMANENT

1- on supprime le courant du Run-Unit
2- ainsi que les articles membres "mandatory" des sets pour lesquels le courant du Run-Unit agit en tant qu'owner ;
et 3- on enlève tous les articles membres optionnels des sets, dont l'owner est le courant du Run-Unit.

Cette forme de delete est souvent qualifiée de delete en cascades.

3) LE DELETE SELECTIF

Le delete sélectif est équivalent au delete permanent à une exception près.
Les membres optionnels des sets, pour lesquels le courant du Run-Unit est owner, ne sont plus simplement déconnectés mais détruits s'ils ne participent à aucun autre set.

4) LE DELETE ALL

1- on supprime le courant du Run-Unit
2- ainsi que tous les articles membres des sets dont l'owner est le courant du Run-Unit.

Chaque fois qu'une primitive avec qualification réclame la destruction d'articles autres que l'article argument, cette même primitive sera appliquée récursivement aux articles visés.

Les primitives " delete sélective " et " delete all " n'ont pas retenu notre attention car elles ne respectent pas les modes de rétention des types d'articles members. Elles font donc fi des spécifications du schéma.

L'émulation du delete consiste à implémenter le delete en cascades à partir du delete sans qualification.

3.3°) L'OPERATION DE MODIFICATION DE L'APPARTENANCE D'UN ARTICLE A UN SET

Si l'appartenance de l'article courant du Run-Unit au set n'est pas obligatoire, alors l'article courant du Run-Unit peut être dissocié du set.

VII.4.3 LA GESTION DES COURANTS

1) INTRODUCTION

Les primitives d'accès, qui sont introduites dans les textes des programmes d'application, sont souvent de type ponctuel. Pour fonctionner, ces primitives requièrent la définition d'un contexte. Un contexte stipule une suite de références vers des articles de la base de données.

Le mode fondamental de référence aux articles, choisi par les SGBD, est le courant.

2) DEFINITION DE LA NOTION DE COURANT

"On appelle généralement courant en matière de base de données le dernier article manipulé dans une séquence déterminée. A un courant correspond un registre qui en contient la référence."

La gestion des courants varie sensiblement d'un SGBD à l'autre. Les SGBD CODASYL procurent un nombre limité de registres prédéfinis (les "currency indicators") . En revanche, les détenteurs d'un SGBD relationnel définissent eux-mêmes leurs courants (les "cursors"). Et enfin, pour mémoire, les registres courants des fichiers COBOL sont implicites.

3) LES COURANTS DANS LE CONTEXTE PARTICULIER D'UN MODULE D'ACCES

" Il existe dans notre contexte trois types de courant : ceux du programme d'application, ceux du SGBD réel et les courants internes au module d'accès."

MDBS dispose des "currency indicators". Il maintient deux "currency indicators" pour chaque set-type (le "current owner" et le "current member") ainsi que le "currency indicator" du processus (le "current of the run unit"). Le "current of the run unit " désigne le dernier article auquel le processus a accédé.

Si MDBS n'admet que deux courants par set-type et un courant par processus, le module d'accès, de même que les programmes d'application, admettent un nombre variable de courants. Lesquels courants apparaissent principalement dans les boucles d'accès (1) où ils s'assimilent aux variables de boucle.

La gestion des courants est un problème en soi dans la mesure où les courants du module d'accès et ceux du programme d'application diffèrent des courants du SGBD réel. Le module d'accès doit donc assurer une synchronisation entre les courants du programme et ceux du SGBD réel.

Le programmeur d'application et le module d'accès disposent des **variables références** pour gérer les courants. Le contenu d'une variable référence désigne un article auquel le programme d'application a accédé antérieurement dans la base de données. La représentation du contenu d'une variable référence est fonction des possibilités offertes par le SGBD réel pour désigner univoquement un article.

MDBS associe une database key à chaque article de la base de données. Cette database key est réputée comme étant l'identifiant interne des articles dans la base de données. Les database key du SGBD réel ne sont cependant pas directement accessibles.

MDBS procure un mécanisme analogue à la keep-list. Il pourvoit l'utilisateur d'une liste de "current users indicators"(2). Les opérations définies sur cette liste se répartissent comme suit :

1- l'allocation ou la désallocation d'un "current user indicator".
2- le rafraîchissement d'un "currency indicator" sur base d'un "current user indicator" ou ,inversément , l'assignation d'un "currency indicator" dans un "current user indicator".
3- des opérations annexes telles la comparaison de deux "current user indicators" et l'annulation d'un "current user indicator".

1.- La notion de variable de boucle a été définie dans le point (I.2.3).

2.- Nous déplorons que MDBS n'accepte pas plus de 255 "current user indicators".

MDBS ne permet donc pas que l'on manipule directement une référence d'article. Les références d'articles sont obligatoirement mémorisées dans le SGBD. Et, c'est pourquoi une variable référence ne contient pas directement la référence à un article. Elle contient plutôt un pointeur vers cette référence. La manipulation des variables références est donc soumise à certaines règles strictes afin de ne pas perturber le comportement global de la base de données. Il est ainsi impossible de déterminer si deux variables références désignent le même article en comparant directement leur contenu. On remarquera aussi que l'assignation directe du contenu d'une variable de référence à une autre peut entraîner l'inaccessibilité de la référence relative à la seconde. Dès lors, la manipulation des références ne peut donc être laissée au programmeur d'application. C'est donc au module d'accès qu'il revient d'assurer la gestion des références.

Les opérations qu'il permet sur les variables références sont :

1- la création d'une variable pouvant contenir une référence. Grâce à cette primitive, l'utilisateur ADLC confère le statut de variable référence à une variable de son programme.

2- la libération d'une variable pouvant contenir une référence.

3- l'annulation d'une référence.

4- l'assignation d'une référence à une autre.

5- la comparaison de deux références.

VII.4.4 LA MISE EN OEUVRE DU COMPOSANT D'ACCES A LA BASE DES SPECIFICATIONS

Le problème de l'intégration de MDBS dans l'atelier BD s'est posé dans les termes suivants. Quelle importance relative fallait-il accorder aux deux dimensions de cette opération d'intégration :

1- le coût de maintenance pour obtenir le nouveau composant "accès à la base des spécifications".

2- le niveau de performance du nouveau composant "accès à la base des spécifications."

Le plan d'intégration a privilégié la seconde dimension par rapport à la première.

On aurait très bien pu conserver l'architecture existante du composant "accès à la base des spécifications". Cette solution n'a cependant pas été retenue car elle implique un trop grand nombre de niveaux. En effet, selon cette solution, une requête ADLC doit être traduite en une requête au module M5, laquelle requête doit à son tour être traduite en une requête MDBS.

Il est souhaitable sur le plan des performances de s'adresser directement au SGBD MDBS d'autant plus que le modèle des données MDBS est très proche du modèle MAGRESTREINT.

VII.4.4.1 LA STRUCTURATION MODULAIRE

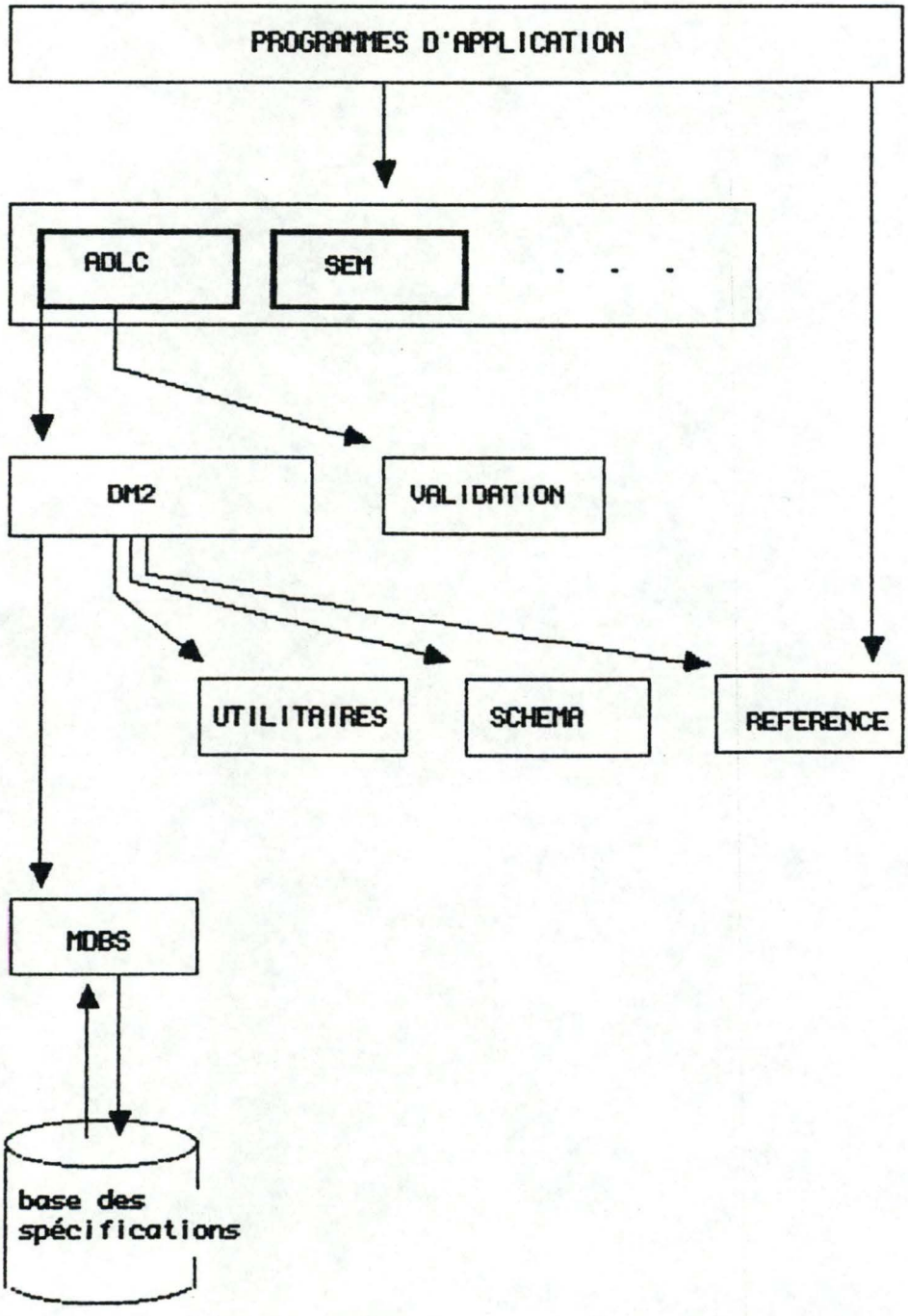


Figure 7.12 : architecture "utilise" de l'atelier bd

1°) ADLC et SEM

Les composant ADLC et SEM réalisent respectivement les primitives ADLC et SEM. Les primitives SEM et ADLC perçoivent les données via le modèle MAGRESTREINT. Nous avons vu au chapitre II qu'une dizaine de restrictions par rapport au modèle MAG déterminaient le modèle MAGRESTREINT.

La réalisation de l'émulation spécifiée au point (VII.4.3) est la principale tâche dont doivent s'acquitter les deux composants. Comme définie précédemment, l'émulation est une méthode qui, appliquée à un niveau, permet de définir un niveau d'abstraction supérieur. Et bien que MDBS soit un SGBD relativement puissant, une émulation s'avère nécessaire pour les contraintes MDBS qui sont plus restrictives que les contraintes du schéma des données de l'Atelier.

Les structures MDBS qui sont inconnues de MDBS sont les suivantes :

1- les items de type texte.
2- les types de chemins obligatoires.

Un type d'objets d'un schéma MAGRESTREINT non admis au niveau MDBS est simulé par un agrégat d'objets au niveau MDBS. Par conséquent, une opération définie sur ce type d'objets induit une séquence d'opérations sur les objets de l'agrégat physique qui lui correspond.

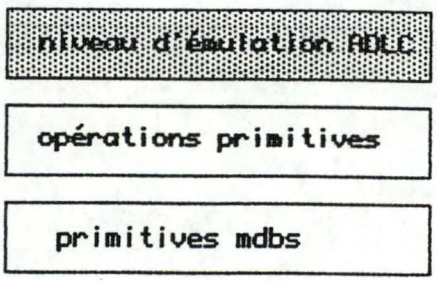
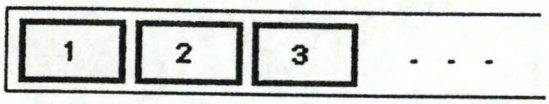


Figure 7.13

La couche ADLC prend en compte le niveau d'émulation. On peut voir le niveau d'émulation comme un ensemble extensible de cases numérotées.



A chaque primitive du module d'accès ADLC , il correspond une case d'un certain numéro. Une case présente la configuration suivante :

if < condition d'émulation > then < séquence émulée >
else < séquence normale >

Une séquence regroupe une suite d'opérations primitives.

exemple :

A la primitive CREATE_iART du module d'accès ADLC correspond la case définie comme suit :

1- condition d'émulation :

existe t'il un item texte ?

2- séquence normale :

opération primitive de création avec pour argument les valeurs d'items relatives à l'article spécifié dans la requête ADLC.

3- séquence émulée :

opération primitive de création avec pour argument les valeurs d'items de type non texte relatives à l'article spécifié dans la requête ADLC.

pour chaque item de type texte faire
(
/* stocker un item texte */

décomposition de la valeur de l'tem texte en une suite de strings de caractères.

pour chaque string faire
(
opération primitive de création d'un article ligne. La valeur d'item-ligne associée à cet article correspond au string
)
)

2°) DM2

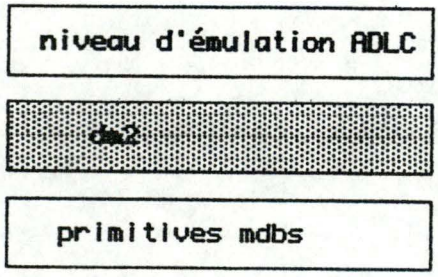


Figure 7.14

DM2 établit un niveau d'abstraction intermédiaire entre la couche ADLC et la couche MDBS. DM2 constitue un module d'accès de bas niveau.

DM2 présente deux caractéristiques.

Les primitives DM2 agissent sur un schéma qui est conforme à la fois à MAGRESTREINT et à MDBS.

DM2 prend en charge la synchronisation entre les variables références et les courants MDBS.

La définition d'un module d'accès comme MA2 présente un double intérêt.

D'une part, les opérations primitives qui apparaissent dans les séquences (normales ou émulées) associées aux primitives ADLC ne sont pas liées à la syntaxe des ordres MDBS.

D'autre part, la présence du module d'accès DM2 permet de dissimuler la gestion des courants à la couche ADLC.

Un troisième argument est encore avancé pour justifier la présence du module DM2. ADLC et SEM présentent deux interfaces différentes mais ils offrent les mêmes fonctionnalités. Il paraissait donc important de disposer d'une couche de primitives de gestion de données qui servent à ces deux composants.

3°) LA VALIDATION

3.1°) INTRODUCTION

Il y a trois grandes familles de validation à effectuer :

- valider les enchaînements des primitives ADLC et SEM.
- contrôler si les arguments de ces primitives sont valides (la validation des requêtes exprimées par l'utilisateur).
- établir une validation au niveau physique. Le module d'accès doit avertir l'utilisateur s'il détecte une déficience d'ordre technique.

Dès que le module d'accès détecte une situation illicite, il interrompt l'interprétation de la requête. En aucun cas, les données dans la base de données n'auront été modifiées.

3.2°) LES ENCHAINEMENTS DE PRIMITIVES

On relève très peu de contraintes d'enchaînements. Cela est dû au fait que l'utilisateur gère lui-même le courant de la séquence sur laquelle il opère.

Nous reprenons ci-dessous les contraintes d'enchaînements par thème.

1) LES VARIABLES DE REFERENCE

Pour utiliser une référence avref dans une primitive P, on aura préalablement donner à la variable avref le statut de variable référence.

2) OUVERTURE ET FERMETURE DE LA BASE

Avant tout traitement sur une base de données, il faut que celle-ci soit ouverte.

L'ouverture d'une base de donnée nécessite que celle-ci ne soit pas déjà ouverte.

L'ouverture simultanée de deux bases n'est pas tolérée.

La fermeture de la base de données doit toujours s'effectuer avant la clôture du programme.

3) LES ACCES AU SUIVANT PAR CHEMIN

Un accès à un article suivant ne peut se faire que si l'accès au premier est effectué.

4) EXISTENCE D'ARTICLE

La suppression, la modification ou l'accès à un article ne peut se faire que si cet article est un article courant du programme d'application.

3.3°) LA VALIDATION DES REQUETES ADLC ET SEM

A nouveau, nous procédons par thème.

1) LA CLASSE FONCTIONNELLE

Les chemins N-1 ne sont pas admis pour les primitives "next-path", "first-path-key", "next-path-key", "attach" et "detach".

2) ORIGINE ET DESTINATION

Les primitives visées sont les primitives d'accès par chemin et les primitives de modification de l'appartenance d'un article à un chemin.
Les articles et types d'articles qui apparaissent dans ces primitives doivent concorder avec le chemin ou le type de chemins mentionné.

3) LES IDENTIFIANTS

Le modèle MAGRESTREINT admet la notion d'identifiant dans un chemin. Le module d'accès doit donc être en mesure de faire respecter ce type de contrainte d'intégrité.

4) LES CLES D'ACCES

On ne peut faire un accès par clé dans un chemin que si une clé a été définie dans ce chemin.

5) LES CHEMINS OBLIGATOIRES

On ne peut pas déconnecter via la primitive "detach" un article d'un chemin obligatoire.

3.4°) LA VALIDATION PHYSIQUE

Les points précédents impliquaient un manquement de logique de la part de l'utilisateur ADLC.
Il ne faut cependant pas oublier que des déficiences d'un autre type peuvent survenir.
A titre d'illustration, nous citons les erreurs de lecture ou d'écriture sur disque.

3.5°) LES MOYENS

Il a été jugé souhaitable de s'appuyer le plus possible sur les possibilités offertes par MDBS.

MDBS fournit un compte rendu sur l'exécution de chacune des commandes de son DML. Ce compte rendu constate le bon déroulement ou l'échec de l'exécution.

MDBS contrôle la validité des arguments qui lui sont transmis. S'il discerne une erreur, alors il ne procède pas à l'exécution de la requête. Le cas échéant, le compte rendu précise quelle est la cause de l'échec.

Les diagnostics MDBS semblent toutefois trop abstrus . Le module "validation" transforme donc le diagnostic MDBS en un diagnostic beacoup plus éloquent pour un utilisateur ADLC ou SEM.

ER_OK : l'exécution de la requête s'est bien déroulée.
ER_NOF : nonobstant la validité de la requête, l'accès demandé a échoué (article non trouvé).
ER_UNQ : la modification d'un article contrevient à la contrainte d'identifiant.
ER_MAND : la déconnection d'un article hors d'un chemin obligatoire.
ER_NOP : la base de données est fermée.
ER_AOP : la base de données est déjà ouverte.
ER_PATH : erreur de type de chemins.
ER_RECT : erreur de type d'articles.
ER_REF : erreur de référence.
ER_SGBD : erreur grâve. Elle devraît entrainer l'arrêt de l'exécution.

4°) SCHEMA

4.1°) VISION THEORIQUE

Si l'on se réfère à la littérature, le traitement d'une requête ADLC devrait se dérouler en quatre étapes. A chacune de ces étapes, on utilise une description particulière des données.

étape1.
On inspecte le schéma des données selon le modèle MAGRESTREINT pour vérifier si la requête a été correctement énoncée.

étape2.
On consulte les informations du schéma MAPPING permettant le passage du modèle MAGRESTREINT vers le modèle sous-jacent au SGBD réel (le modèle PHYSIQUE). D'une requête ADLC, on dérive ainsi une ou plusieurs requêtes MDBS.

étape3.
On exécute les requêtes MDBS.
MDBS se sert du schéma PHYSIQUE pour effectuer des contrôles dès qu'une nouvelle information est placée dans la base de données ou dès qu'une information existante dans la base de données est modifiée et/ou enlevée.

étape4.
On consulte les informations du schéma MAPPING permettant le passage du modèle PHYSIQUE vers le modèle MAGRESTREINT. On peut ainsi composer la réponse à la requête ADLC à partir des résultats fournis par le SGBD réel.

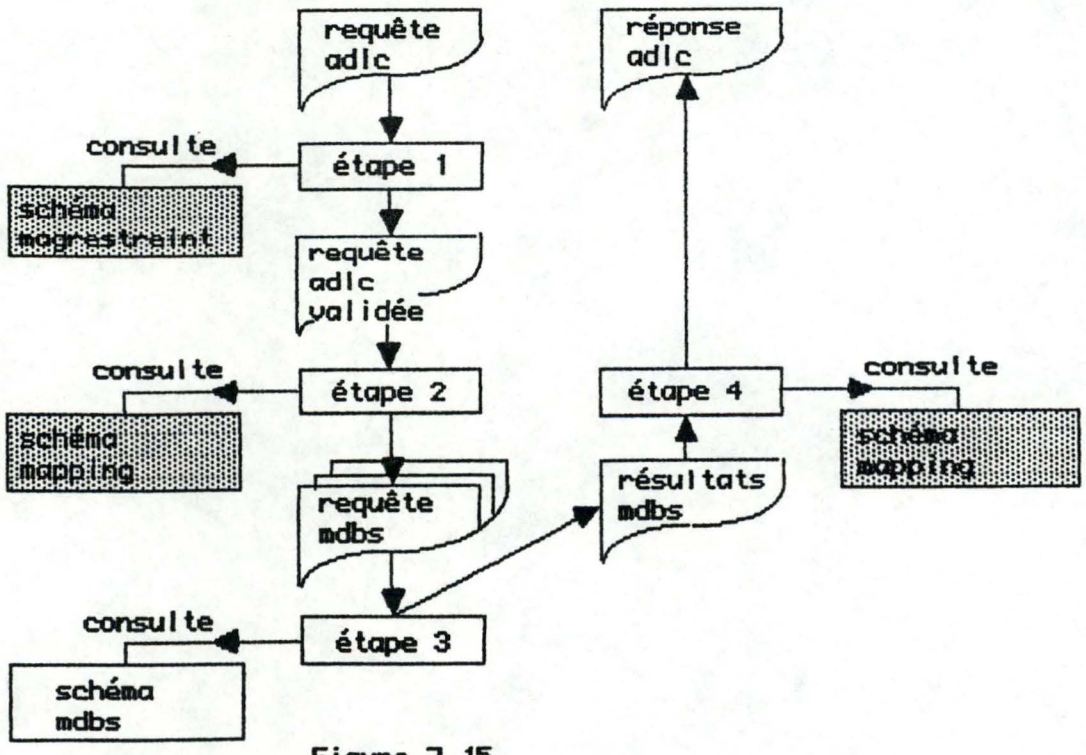


Figure 7.15

4.2°) SCHEMA.

SCHEMA est un type abstrait de données associé au concept du même nom.

Les méta-informations comprises dans les schémas MAGRESTREINT et MAPPING sont intégrées dans les tables internes. Le but du module "schéma" est de satisfaire les diverses demandes de consultation de ces tables.

L'obtention d'un mapping se ramène donc à une simple invocation du module "schéma".

Nous savons que MAGRESTREINT n'admet pas les items de type texte. Ainsi, si on a une structure originelle comprenant un type d'articles avec un attribut texte, SCHEMA va produire une structure dérivée comprenant le type d'article originel, le type d'articles LIGNE et le type de chemins joignant le type d'article originel au type d'articles LIGNE.

De façon rigoureuse, nous dirons :
étant donné le code d'un objet appartenant à un schéma MAGRESTREINT, SCHEMA produit le code du ou des objets dérivés.

L'INFORMATION INHERENTE A SCHEMA

Nous allons constituer une collection minimale d'informations décrivant les différrents schémas.
Une partie seulement de cette collection sera effectivement stockée dans les tables internes. Le reste concerne des informations "calculées" à partir des informations "stockées".
Ainsi les noms des objets physiques (record type, set type,...) peuvent être obtenus à partir du code des objets MAGRESTREINT pourvu que l'on ait défini une convention.
Par exemple, le nom d'un type d'articles correspond à la concaténation de "rec" avec le code dudit objet.

LES PRIMITIVES DE SCHEMA

Les primitives élaborées sont les primitives de consultation suffisantes pour le bon fonctionnement du module d'accès.

1) **Les primitives de consultation du schéma MAGRESTREINT.**

Exemple : - quelle est la classe fonctionnelle de tel type de chemins ?
- tel type de chemins est-il obligatoire ?

2) **Les primitives de consultation du schéma MAPPING.**

Exemple : - quel est le nom du record type correspondant à tel type d'articles ?

5°) REFERENCE

Le module d'accès doit assurer la gestion des courants. Cette gestion des courants est responsable de la synchronisation des courants entre le programme d'application et le SGBD réel. Synchroniser signifie un positionnement des currency indicators sur base des courants utilisateurs.

Comme la gestion des courants est tributaire de la notion de référence, nous avons défini un module consacré à la gestion des références.

6°) LES UTILITAIRES

Le module UTILITAIRES regroupe des fonctions d'intérêt général.

VII.4.4.2 LA SPECIFICATION DES COMPOSANTS

1°) DM2

Les primitives DM2 s'accordent parfaitement avec les primitives ADLC dans les catégories qui suivent :

- accès à une base de données
- accès séquentiel
- accès direct
- modification de l'appartenance d'un article à un chemin.

Dans les autres catégories, les primitives DM2 sont plus élémentaires. Ainsi, DM2 distingue les primitives d'accès par chemin 1-N des primitives d'accès par chemin N-1.

1.1°) LES PARAMETRES

Les paramètres ont été définis au point (VII.2.4).

1.2°) SPECIFICATIONS

=1= DB_OUV (adbstat,adbname)

args : dbname
res : dbstat
post : si pas d'erreur
alors :
la base de donnée désignée par dbname devient

erreurs possibles :
- la base de donnée désignée par dbname est déjà ouverte.

note technique :

Les tables internes sont stockées dans la base de données. A l'ouverture de la base, elles sont chargées en mémoire centrale.

=2= DB_FERM (adbstat,adbname)

args : dbname
res : dbstat
post : si pas d'erreur
alors :
fermeture de la base de donnée courante.

erreurs possibles :
- la base de donnée désignée par dbname est déjà fermée.

=3= DB_SEQ_F (adbstat,avref)

args : vref.RTYPE
res : vref.ITEM, dbstat
post : si pas d'erreur
alors :
soit art le type de l'article avref,
art = avref.RTYPE,
s la séquence de tous les articles de type art.
l'article avref = first (s).

note technique :

Ce n'est pas l'ensemble des articles d'un type déterminé qui constitue le référentiel des opérations d'accès séquentiel MDBS. MDBS considère plutôt les articles composant un fichier ou une area. Comme le nombre d'areas est de quinze, plusieurs types d'enregistrements cohabitent dans une même area. De ce fait, les opérations d'accès séquentiel MDBS ne présentent guère d'intérêt.

La difficulté a été contournée de la manière suivante. Chaque type d'enregistrements joue le rôle de member dans un set "singulier" dont l'owner est l'article système. Ainsi, l'accès séquentiel est implémenté via un accès par chemin.

=4= DB_SEQ_N (adbstat,avref)

args : vref.REF, vref.RTYPE
res : vref', dbstat
post : si pas d'erreur
alors :
soit art le type de l'article avref,
art = avref.RTYPE,
s la séquence de tous les articles de type art.
l'article avref' = next (l'article avref , s).

erreurs possibles :
- l'article avref n'est pas de type vref.RTYPE.

=5= DB_CHEM1n_F (adbstat,avref1,aref2,chem)

args : ref2, chem
pre : l'article aref2 est origine d'un chemin de type "chem".
la classe fonctionnelle de chem = 1-N.
res : vref1, dbstat
post : si pas d'erreur
alors :
soit art le type d'article destination de chem,
s la séquence de tous les articles de type art ciblés par l'article aref2 dans chem.
l'article avref1 = first (s).

erreurs possibles :
- l'article aref2 n'est l'origine d'aucun chemin de type chem.

=6= DB_CHEMn1_F (adbstat,avref1,aref2,chem)

DB_CHEMn1_F est analogue à DB_CHEM1n_F si ce n'est que la classe fonctionnelle de chem = N-1.

=7= DB_CHEM1n_N (adbstat,avref1,aref2,chem)

args : vref1.REF, ref2
pre : la classe fonctionnelle de chem = 1-N.
res : vref1', dbstat
post : si pas d'erreur
alors :
soit art le type d'article destination de chem,
s la séquence de tous les articles de type art qui sont la cible du chemin chem d'origine aref2.
l'article avref1' = next (l'article avref1 , s).

erreurs possibles :
- l'article avref2 n'est l'origine d'aucun chemin de type chem.
- l'article avref1 n'est la cible d'aucun chemin de type chem.

=8= DB_CHEMCLE1n_F (adbstat,avref1,aref2,chem,cle)

args : ref2, chem, cle
pre : la classe fonctionnelle de chem = 1-N.
res : vref1', dbstat
post : si pas d'erreur
alors :
soit art le type d'article destination de chem,
s la séquence de tous les articles de type art ciblés par l'article aref2 dans chem et ayant cle comme valeur de clé d'accès dans chem.
l'article avref1 = first (s).

=9= DB_CHEMCLE1n_N (adbstat,avref1,aref2,chem,cle)

args : vref1.REF, ref2, chem, cle
pre : la classe fonctionnelle de chem = 1-N.
res : vref1', dbstat
post : si pas d'erreur
alors :
soit art le type d'article destination de chem,
s la séquence de tous les articles de type art ciblés
par l'article aref2 dans chem
et ayant cle comme valeur de clé d'accès dans chem.
l'article avref1 = next (l'article avref1 , s).

erreurs possibles :
- l'article aref2 n'est l'origine d'aucun chemin de type chem.
- l'article avref1 n'est la cible d'aucun chemin de type chem.

=10= DB_DIR (adbstat,avref)

args : vref.REF
res : vref',dbstat
post : si pas d'erreur
alors :
accès à l'article avref.

=11= DB_CRT (adbstat,avref,curlist,pathlist)

args : vref.RTYPE, vref.ITEM, curlist, pathlist
res : BD', vref, dbstat
post : si pas d'erreur
alors :
création de l'article avref et insertion de cet article dans les chemins spécifiés par les couples
(pathlist(i),curlist(i)).

erreurs possibles :
- l'ensemble des chemins obligatoires relatifs au type d'article vref.RTYPE n'est pas inclus dans pathlist.
- le type d'un article curlist°i§ n'est pas reconnu comme type d'articles origine pour pathlist°i§.

=12= DB_DEL (adbstat,aref)

args : ref
pre : l'article aref n'est l'origine d'aucun chemin obligatoire.
res : BD', dbstat
post : si pas d'erreur
alors :
destruction de l'article aref.

=13= DB_MOD (adbstat,avref)

args : vref
res : BD', vref', dbstat
post : si pas d'erreur
alors :
les valeurs d'item de l'article avref' = avref.ITEM.

commentaires :
On peut utiliser DB_MOD pour modifier les items d'un article,peu importe qu'il s'agisse d'items ikos ou non.

=14= DB_CON (adbstat,aref1,aref2,chem)

args : ref1, ref2, chem.

res : BD', dbstat
post : si pas d'erreur
alors :
l'article aref1 est cible du chemin chem d'origine aref2.

erreurs possibles :
- chem ne représente pas un type de chemins 1-N.
- le type de l'article aref1 n'est pas reconnu comme type d'article cible de chem.
- le type de l'article aref2 n'est pas reconnu comme type d'articles origine de chem.

=15= DB_DISCON (adbstat,aref,chem)

args : ref, chem.
res : BD', dbstat
post : si pas d'erreur
alors :
l'article aref n'est relié à aucun article via chem.

erreurs possibles :
- chem ne représente pas un type de chemins 1-N.
- le type de l'article aref n'est pas reconnu comme type d'article cible de chem.

2°) SCHEMA

DEFINITION DES PRIMITIVES

2.1°) LES PARAMETRES

1) LES PARAMETRES RELATIFS AU SCHEMA MAGRESTREINT

CA : code numérique d'un type d'articles.

CP : variable entière dont le contenu désigne un chemin.
ACP : adresse C d'une variable CP.
APATHLIST : adresse C d'un tableau de codes de types de chemins.

1-N : constante numérique représentant une classe fonctionnelle.
N-1 : constante numérique représentant une classe fonctionnelle.
CF : variable entière dont le contenu indique une classe fonctionnelle.
ACF : adresse C d'une variable CF.

MAND : constante numérique représentant un type de rétention.
OPTIONAL : constante numérique représentant un type de rétention.
RET : variable entière dont le contenu indique un type de rétention.
ARET : adresse C d'une varible RET.

2) LES PARAMETRES RELATIFS AU SCHEMA MAPPING

CA : variable entière dont le contenu désigne un type d'articles.
ACA : adresse C d'une variable CA.

CP : code numérique d'un path.

REC : variable string dont le contenu indique un nom de record-type MDBS.
AREC : adresse C d'une variable REC.

SET : variable string dont le contenu indique un nom de set MDBS.
ASET : adresse C d'une variable SET.

SYS : variable string dont le contenu indique un nom de set system-owned MDBS.
ASYS : adresse C d'une variable SYS.

2.2°) LES SPECIFICATIONS

1) LES PRIMITIVES DE CONSULTATION DU SCHEMA MAGRESTREINT

```
=1=  SMR_CF (cp,acf)              /* get_classe_fonctionnelle */

args :  cp
res  :  cf, coderetour
post :  si il existe un type de chemins de code cp
        alors  - cf est la classe fonctionnelle
                 de ce type de chemins
               - coderetour = ok
        sinon  - coderetour = erreur


=2=  SMR_RET (cp,aret)            /* get_rétention */

args :  cp
res  :  ret, coderetour
post :  si il existe un type de chemins TC de code cp
        alors
               - ret est le type de rétention caractérisant le
                 type du chemin inverse de TC
               - coderetour = ok
        sinon
               - coderetour = erreur


=3=  SMR_PATH (ca,apathlist)      /* get_path */

args :  ca
res  :  pathlist, coderetour
post :  si il existe un type d'articles de code ca
        alors
             - pathlist regroupe l'ensemble
               des types de chemins TC dont
                 - le code du type d'article origine ou d'un des
                   types d'article origines = ca
                 - le  type du chemin inverse de TC est un  type
                   de chemins obligatoire

             - coderetour = ok

          sinon
             - coderetour = erreur.
```

2) LES PRIMITIVES DE CONSULTATION DU SCHEMA MAPPING

=1= SM_RECNAME (ca,arec)

args : ca
res : rec
post : rec représente le nom du record-type MDBS
correspondant au type d'article de code ca.

=2= SM_SYS (ca,asys)

args : ca
res : sys
post : sys représente le nom du set-type system-owned MDBS
dont le member type correspond au type d'article de
code ca.

=3= SM_RECODE (rec,aca)

args : rec
res : ca
post : ca représente le code du type d'article correspondant
au record-type de nom rec.

=4= SPH_PNAME (cp,aset)

args : cp
res : set
post : set représente le nom du set MDBS correspondant au
type de chemins cp.

=5= SM_TX (ca1,aca2,acp) /* get_transformation */

args : ca1
pre : ca1 a au moins un attribut de type texte
res : ca2, cp, coderetour
post : - ca2 désigne le type d'article TEXTE
- cp désigne le type de chemin STEXTE.

3°) REFERENCE

convention.
Les currency users indicators sont désignés par le vocable "CUI".

3.1°) LES SPECIFICATIONS

=1= GET_VAR (aref)

```
args :  ref
res  :  ref, dbstat
post :  si pas d'erreur
        alors  - ref a le statut de variable référence
               - ref est égal au numéro d'un CUI.
```

=2= FREE_VAR (aref)

```
args :  ref
res  :  ref, dbstat
post :  si pas d'erreur
        alors  - ref n'a pas le statut de variable référence
               - CUI ( ref ) est libre.
```

=3= NULL (aref)

```
args :  ref
res  :  ref, dbstat
post :  si pas d'erreur
        alors  - ref ne référence aucun article.
```

=4= ASSIGN (aref1,aref2)

```
args :  ref1,ref2
res  :  ref2, assign
post :  si pas d'erreur
        alors  - si ref1 référence un article
                 alors  - ref2 référence le même article
                 sinon  - ref2 ne référence aucun article.
```

=5= EQUAL (aref1,aref2)

```
args :  ref1, ref2
res  :  equal, dbstat
post :  si pas d'erreur
        alors  - si ref1 et ref2 référencent le même article
                 alors  - equal = 1
                 sinon  - equal = 0
```

=6= ISNUL (aref)

```
args :   ref
res  :   isnul, dbstat
post :   si pas d'erreur
         alors  - si ref ne référence aucun article
                  alors isnul = 1
                  sinon isnul = 0.PA
```

VII.5 LE MODULE D'ACCES SEM

Les concepts relatifs au module d'accès sont plus simples que ceux de l'interface ADLC. En particulier, on n'y retrouvera pas les notions de variable de référence, de variable items d'article et d'attribut du type texte. Le principal concept est celui de variable (integer) pouvant contenir une référence d'article.

LES PRIMITIVES SEM

1°) LES PARAMETRES

DBSTAT :
adresse C d'une zone de diagnostic contenant les constituants FNCODE (integer), représentant le code de l'opération durant laquelle une erreur s'est produite, et ERRCODE(integer)spécifiant l'erreur ou l'absence d'erreur.

OPCODE :
adresse C du code de l'opération à effectuer. Ce code représente la primitive activée.

RECTCODE :
code numérique du type d'articles.

RECREF :
adresse C d'une variable qui peut contenir une référence d'un article de la base de données; on appellera "article RECREF" l'article dont la référence est dans RECREF.

KEYCODE :
adresse C du code de la clé d'accès qui est concernée dans l'opération.

PATHLIST :
adresse C d'un tableau de codes de types de chemins; on désignera par "type de chemins PATHLIST°i§" le type de chemins dont le code numérique est dans PATHLIST°i§.

CURLIST :
adresse C d'un tableau de références d'articles de la base de données; on appellera "article CURLIST°i§" l'article dont la référence est dans CURLIST°i§.

Z-VALUES :
adresse C d'une zone de réception et d'envoi de valeurs d'item; contient également le nom de la base de données; les valeurs y sont rangées sous la forme de strings C.

REFTEXT :
adresse C d'une structure TEXTE.

2°) LES SPECIFICATIONS

Il n'existe qu'un seul point d'entrée, qui est disponible selon le format suivant :

```
SEM    ( DBSTAT,    OPCODE,   RECTCODE   ,   RECREF,   KEYCODE,
         PATHLIST, CURLIST, Z-VALUES, REFTEXT )
```

Les primitives sont activées par un appel à SEM selon les différentes valeurs du paramètre OPCODE. Ces primitives sont détaillées ci-dessous.

ACCES A UNE BASE DE DONNEES

ouverture d'une base de données.

```
args :  opcode, z-values
pre  :  z-values contient le nom d'une base de données.
        opcode = 1.
res  :  dbstat
post :  la base de données désignée par z-values est ouverte.
```

fermeture d'une base de données.

```
args :  opcode, z-values
pre  :  z-values contient le nom d'une base de données.
        opcode = 2.
res  :  dbstat
post :  la base de données désignée par z-values est fermée.
```

ACCES SEQUENTIEL

accès séquentiel aux articles d'un type.

```
args :  opcode, rectcode, recref
pre  :  opcode = 3.
res  :  recref', z-values, dbstat
post :  soit s la séquence de tous les articles de type rectcode.
        si recref = une référence nulle
        alors
        l'article recref' = first ( s ) ,
        les  valeurs des items (sauf textes ) sont rangées dans
        z-values
        sinon
        l'article recref' = next ( s ) ,
        les valeurs des items (sauf textes ) sont rangées dans
        z-values.
```

ACCES PAR CHEMIN

accès à un article cible d'un chemin.

args : opcode, rectcode, recref, curlist°0§, pathlist°0§
pre : opcode = 5.
res : recref', z-values, dbstat
post : soit s la séquence des articles de type rectcode, qui sont la cible du chemin pathlist°0§ d'origine curlist°0§.
si recref = une référence nulle
alors
l'article recref' = first (s) ,
les valeurs des items (sauf textes) sont rangées dans z-values
sinon
l'article recref' = next (s) ,
les valeurs des items (sauf textes) sont rangées dans z-values.

ACCES PAR CLE DANS UN CHEMIN

accès par clé à un article dans un chemin.

args : opcode, rectcode, recref, keycode, z-values
pre : opcode = 6.
res : recref', z-values', dbstat
post : soit s la séquence des articles de type rectcode, ciblés par l'article curlist°0§ dans pathlist°0§, et dont la valeur de la clé keycode est rangée dans z-values.
si recref = une référence nulle
alors
l'article recref' = first (s) ,
les valeurs des items (sauf textes) sont rangées dans z-values
sinon
l'article recref' = next (s) ,
les valeurs des items (sauf textes) sont rangées dans z-values.

ACCES DIRECT

accès par référence.

args : opcode, recref
pre : opcode = 7.
res : recref, rectcode, z-values, dbstat
post : rectcode est le type de l'article recref.
les valeurs d'items (sauf textes) sont rangées dans z-values.

consultation d'un item texte.

args : opcode, recref
pre : opcode = 13.
res : reftext, dbstat
post : le texte référencé par reftext contient la valeur de l'item texte de l'article recref.

MODIFICATION DES DONNEES

création d'un article.

args : opcode, rectcode, curlist, pathlist, z-values, reftext
pre : opcode = 8.
res : recref, dbstat
post : création d'un article de type rectcode.
Les valeurs des items non texte de cet article correspondent aux valeurs rangées dans z-values.
Si l'article possède un item texte,
alors la valeur de cet item correspond au texte référencé par reftext.
L'article est inséré dans les chemins pathlist°i§ d'origine curlist°i§.
La référence de cet article est rangée dans recref.

suppression d'un article.

args : opcode, recref
pre : opcode = 9.
res : dbstat
post : suppression de l'article recref ainsi que des articles qui en dépendent directement ou non, via des chemins obligatoires. recref est mise à nul.

modification de valeurs d'items (sauf IKO).

args : opcode, recref, z-values
pre : opcode = 10.
res : dbstat
post : modification des valeurs d'items de l'article recref à l'exception de ceux qui interviennent dans la définition d'un identifiant ou d'une clé d'acc
s. Les nouvelles
valeurs sont présentes dans z-values.

modification d'un item texte.

args : opcode, recref, reftext
pre : opcode = 11.
res : dbstat
post : modification (ou création) de la valeur d'item texte de l'article recref. La nouvelle valeur se trouve dans le texte référencé par reftext.

modification de l'appartenance d'un article à des chemins 1-N

args : opcode, recref, curlist, pathlist, z-values
pre : opcode = 12.
res : dbstat
post : si curlist°1§ contient une référence nulle,
alors
l'article recref est retiré du chemin de type pathlist°0§ auquel il appartient
sinon
l'article recref est inséré comme cible du chemin de type pathlist°0§ dont l'article curlist°1§ est origine ; s'il appartenait déjà à un tel chemin, il en est préalablement retiré.

C O N C L U S I O N

Au terme de l'expérience que constitue ce mémoire, nous voudrions procéder à quelques évaluations relatives à la réalisation du mémoire, mais également relatives au logiciel Atelier BD.

Quels sont les traits fondamentaux de l'Atelier qui en font un environnement privilégié ? L'extension de l'Atelier BD à la génération de schémas conformes MDBS nous a permis d'insister sur le caractère général des outils de l'Atelier. Nous ajouterons simplement que la souplesse des dynamiques de conception envisageables en fait un réel support à la conception.

Notre conclusion ne sera pas une évaluation globale de l'opportunité d'intégrer ou non le SGBD MDBS dans l'Atelier BD. Nous en serions d'ailleurs incapables puisque l'intégration du composant d'accès à la base des spécifications est à peine commencée. Des évaluations partielles sont cependant possibles. Ainsi sur le plan de la compacité, on constate une nette réduction de la taille du RUN-TIME. Ce gain d'espace mémoire se situe au niveau du composant d'accès à la base des spécifications (de l'ordre de 100%). Sur le plan économique, le coût de l'étude et la réalisation du composant d'accès à la base des spécifications dans l'architecture antérieure est évaluée comme suit :

- nombre de mois/homme : 16 mois.
- coût d'un homme/année : 1.100.000 frs.
- coût total : 1.760.000 frs.

Les coûts affectés à l'étude et à la réalisation du composant d'accès à la base des spécifications de la nouvelle architecture sont évalués comme suit :

- nombre de mois/homme : 10 mois.
- acquisition de MDBS : X.

Il nous reste enfin à définir les conditions dans lesquelles ce travail de fin d'étude a pu être réalisé.

FORMATION :

Elle se résume à nos études universitaires en informatique ainsi qu'à la consultation d'une abondante littérature dans le domaine des bases de données.

ORGANISATION :

* Etablissement d'un cahier des charges définissant les objectifs.

* Phase de recherche préliminaire. Remarquons que relativement peu de recherches ont abouti à une réalisation concrète. Ces recherches nous ont cependant permis de comprendre les objectifs, comment les atteindre et à quel prix !

* Consultation d'autorités dans le domaine informatique et plus précisément dans le domaine des bases de données. Nous remercions J.L. Hainaut et K. Chabottaux pour leurs connaissances et les qualités pédagogiques qu'ils ont bien voulu mettre à notre service, que ce soit sous forme de notes de cours ou d'entretiens.

* Test et évaluation des possibilités du Système de Gestion de Bases de Données MDBS.

* Contribution à la réalisation du composant d'accès à la base des spécifications.

* Réalisation d'une documentation interne nécessaire à la maintenance.

* Travail régulier appuyé d'un suivi très apprécié.

* Travail individualisé avec de fréquentes mises en commun.

FACTEURS FAVORABLES ET FREINS

Les aspects de compacité, de portabilité et de maintenance aisée du logiciel Atelier BD étaient très importants pour le développement ultérieur et sa destination industrielle. Nous n'avons donc pas souffert d'une absence de priorité accordée au sujet à l'étude, ou d'une planification insuffisante, ou d'un manque de soutien au niveau supérieur.

LES RISQUES

Mener à bien une expérience de cette importance n'est pas sans risque.

- risques d'ordre technique dus notamment à la difficulté d'intégrer MDBS dans l'architecture existante. Ces difficultés peuvent provenir d'erreurs situées aux différents niveaux du développement (SPECIFICATIONS, CONCEPTION et IMPLEMENATION).

- risque financier : sous-évaluation du développement et de la phase d'intégration.

LES PHASES A VENIR

- intégration du composant accès à la base des spécifications dans l'Atelier BD.

- mise au point d'un système de mesure et d'évaluation, ainsi qu'un suivi réaliste des incidents et des suggestions.

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

- ANSI/X3/SPARC Study Group on Data Base Management, "Référence Model for DBMS Standardization", Final Report, SIGMOD RECORD, Vol. 15, No. 1, 1986.

- ANSI/X3/SPARC, Tsichritzis & Klug (Ed.), "The ANSI/X3/SPARC DBMS Framework : Report of the Study Group on Data Base Management System", Information Systems 3, 1978.

- F. Bodart & Y. Pigneur, "Conception assistée des applications informatiques, 1. Etude d'opportunité et Analyse conceptuelle", MASSON, 1983.

- M. Cadelli, "Programmation sur la base de données de l'Atelier", Version 1.0, Documentation relative à l'Atelier BD, Juillet, 1986.

- C. Charlot & L. Müller, "Contribution à l'Atelier logiciel de conception de Bases de Données : Etude de transformations de schémas", Mémoire, Institut d'Informatique, 1986.

- Date, "An Introduction to Data Base Systems", ADDISON WESLEY, 3rd Edition, 1981.

- Date, "An Introduction to Data Base Systems", ADDISON WESLEY, 4th Edition, 1986.

- Date, "An Introduction to Data Base Systems", Vol. 2, ADDISON WESLEY, 4th Edition, 1986.

- F.R. McFadden & J.A. Hoffer, "Data Base Management", The Benjamin/Cummings Publishing Company, Inc., 1985.

- J.L. Hainaut, "Conception assistée des applications informatiques, 2. Conception de la base de données", MASSON, 1986.

- J.L. Hainaut, "Etude de la clause SET OCCURENCE SELECTION dans les rapports CODASYL 71 et 72",Notes de cours, Institut d'Informatique, 1981.

- J.L. Hainaut, "Introduction aux Systèmes de Gestion de Bases de Données CODASYL 71", Notes de cours, Institut d'Informatique, 1985.

- J.L. Hainaut, "La mini-informatique : Etat actuel de la technologie en matière de matériel et de logiciel", Notes de cours, Institut d'Informatique, 1986.

- J.L. Hainaut, "Typologie de configurations informatiques petites et moyennes", Notes de cours, Institut d'Informatique, 1986.

- J.L. Hainaut, "Introduction à la Théorie Relationnelle des Bases de Données", Notes provisoires de cours, Institut d'Informatique, Janvier, 1987.

- J.L. Hainaut, "Schémas de la Base de Spécifications", Documentation relative au Projet Atelier BD, Institut d'Informatique, Janvier,1986.

- J.L. Hainaut, "Introduction aux outils généraux de développement de l'Atelier", Documentation relative au Projet Atelier BD, Institut d'Informatique, Janvier, 1987.

- J.L. Hainaut, "Conception d'une Base de Données : éléments méthodologiques", Documentation relative au Projet Atelier BD, Institut d'Informatique, Mars, 1987.

- J.L. Hainaut, "L'Atelier de conception de Base de Données : ORGA", Manuel de référence, Version 1.0, Institut d'Informatique, Avril, 1987.

- P. Maréchal, "Contribution à la conception et la réalisation d'un Atelier logiciel orienté base de Données", Mémoire, Institut d'Informatique, 1986.

- J. Martin, "Computer Data-Base Organisation", Prentice-hall, 2nd Edition, 1977.

- J. Martin, "Information Engineering", SAVANT, Vol. 1, 1986.

- D.L. Parnas, "Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems", Communications of the ACM, 1977.

- H. Robinson, "Database Analysis and Design", Chartwell-Bratt Publish., England, 1981.

- T.A. Rullo, "Advances in Data Base Management", Vol. 1, Heyden, 1980.

- M. Vetter, "Database Design Methodology".

- "MDBS Data Base Design Reference Manual : The MDBS DDL Manual", Version 3.00, Micro Data Base Systems,inc., Juillet, 1981.

- "MDBS Data Base Design Reference Manual : The MDBS DMS Manual", Version 3.00, Micro Data Base Systems,inc., Juillet, 1981.

A N N E X E I

SCENARIO DE CONCEPTION D' UNE BASE DE DONNEES MDBS

L'objectif assigné à l'Atelier BD est d'assister le concepteur dans l'élaboration du schéma d'une base de données à partir d'un schéma conceptuel E-A.

Dans la présente annexe, nous envisageons la production d'un schéma MDBS. Nous décrirons d'abord son schéma conceptuel. Puis, nous présenterons la succession des schémas qui précèdent l'obtention du schéma physique.

En ce qui concerne les structures d'accès, nous faisons les hypothèses suivantes :

- A tout type d'associations sont associés deux types de chemins.
- Tout identifiant devient également une clé d'accès.

1) SCHEMA CONCEPTUEL

UNE BREVE DESCRIPTION

Les clients se déplacent dans une agence pour effectuer des transactions.

Une transaction est une opération bancaire permettant de créditer ou de débiter un compte (le type de la transaction) d'un certain montant. Une transaction est localisable dans le temps (date).

Un client est caractérisé par un numéro (identifiant), un nom et une adresse. Il possède également une liste de numéros de téléphone.

Un compte est caractérisé par un numéro (identifiant) et un solde courant.

Une banque est caractérisée par un nom (identifiant). Elle possède un siège social ainsi qu'une ou plusieurs agences.

Une agence est caractérisée par un numéro et une adresse. Le numéro et le nom d'une agence sont reconnus comme identifiants internes dans une banque.

TYPES D'OBJETS ET STRUCTURES

SCHEMA

BANQUE_C

TYPES D'ENTITES

```
CLIENT
     num              n(5)
     nom              c(30)
     adresse
          num-rue                  n(3)
          nom-rue                  c(30)
          localité                 c(30)
               c-postal            n(4)
               commune             c(30)
     telephone °2§  c(14)

COMPTE
     num              n(6)
     solde            n(10)

AGENCE
     num-ag           n(4)
     nom-ag           c(30)
     adresse
          num-rue                  n(3)
          nom-rue                  c(30)
          localité                 c(30)
               c-postal            n(4)
               commune             c(30)

BANQUE
     nom-b            c(30)
     siège            c(30)
```

TYPES D'ASSOCIATIONS

```
TRANS
     effectue     (1-*) : client
     enregistre   (0-*) : agence
     est affecté (0-*) : compte
          type       c(1)
          montant    n(10)
          date       c(6)
```

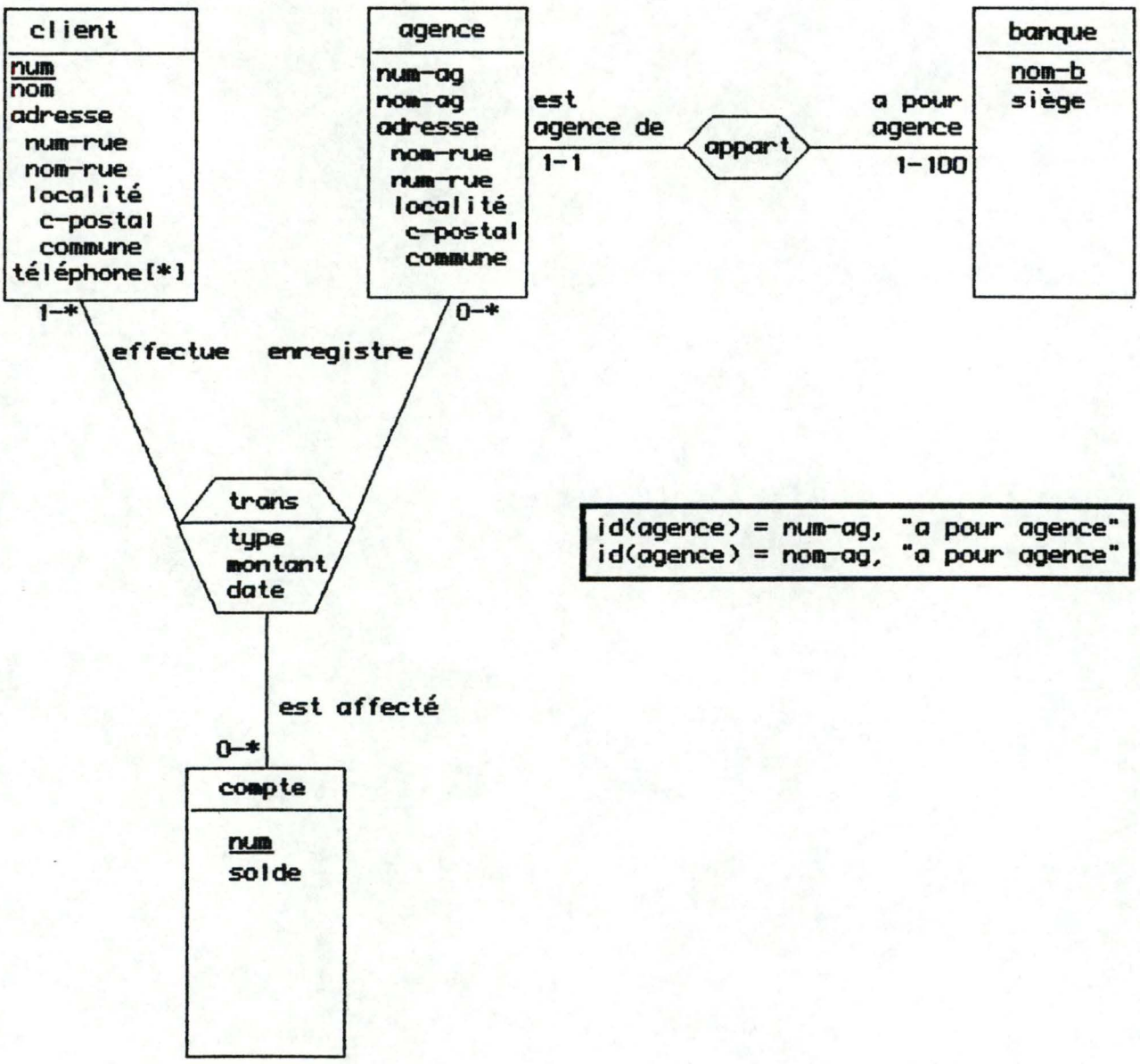
APPART

a pour agence (1-100) : banque
est agence de (1-1) : agence

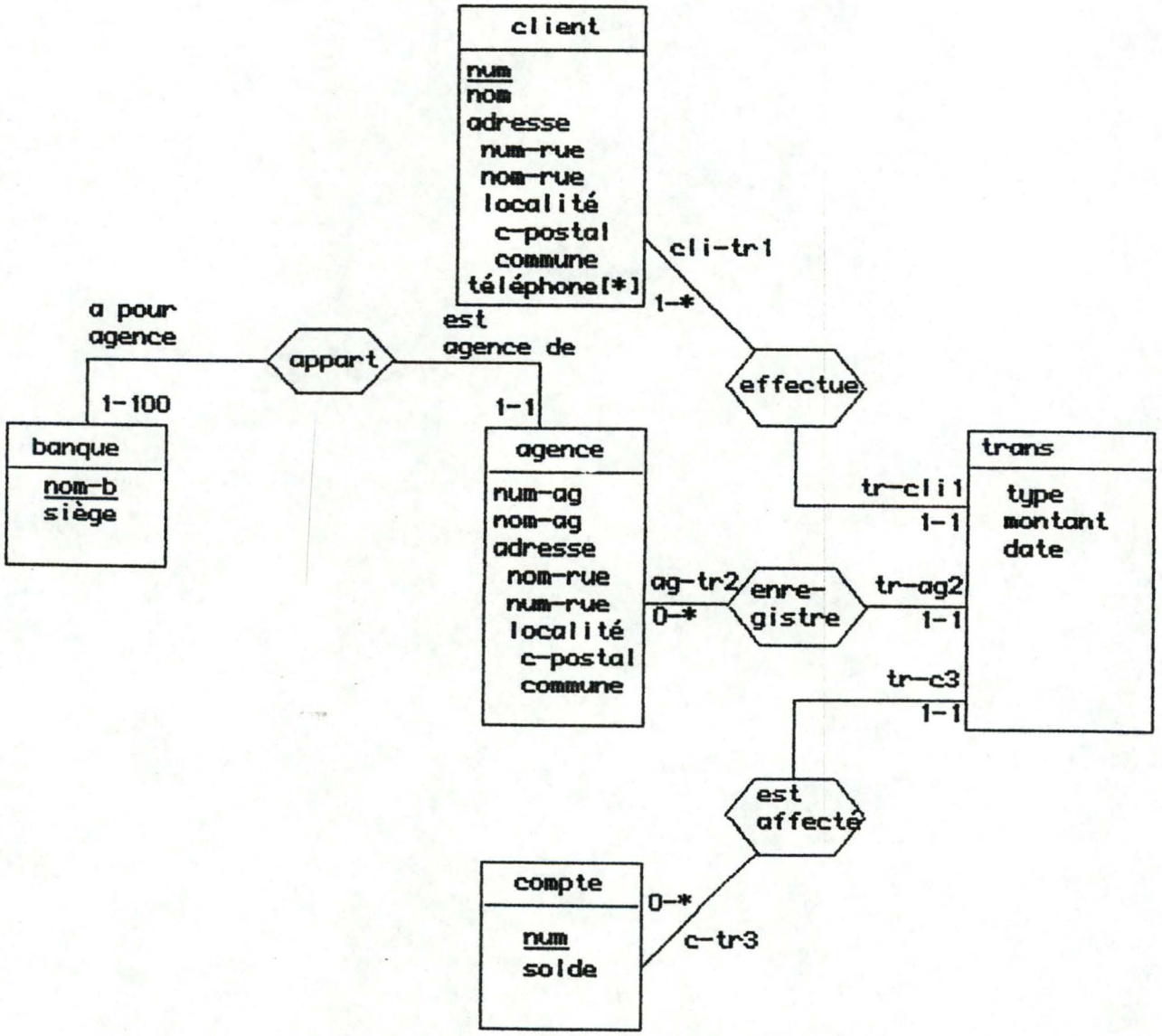
IDENTIFIANTS

id (CLIENT) = num
id (COMPTE) = num
id (AGENCE) = num , "a pour agence"
id (AGENCE) = nom , "a pour agence"
id (BANQUE) = nom-b

gestion d'une banque : schéma conceptuel

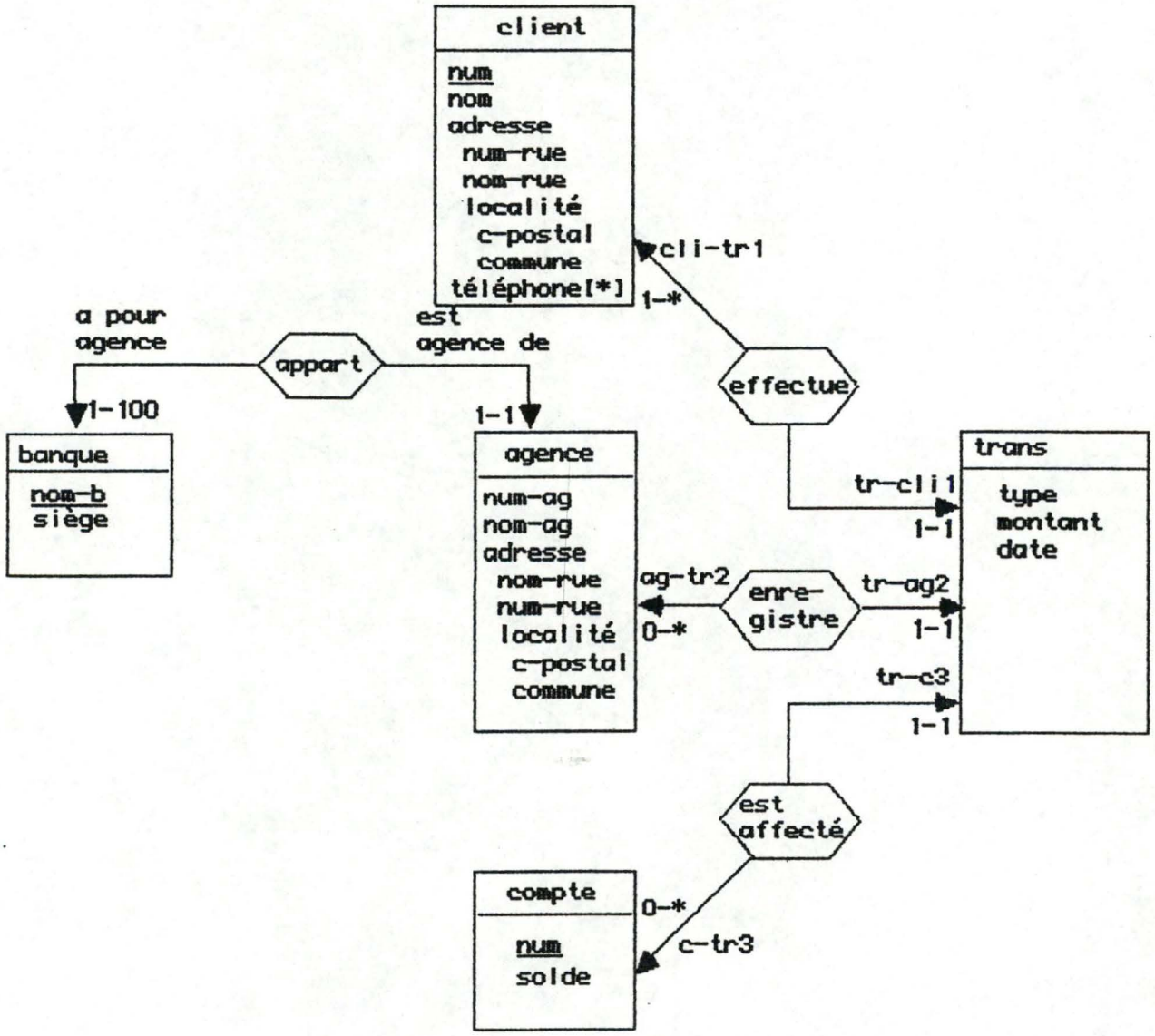


2. gestion d'une banque : schéma binaire



id(agence) = num-ag, "a pour agence"
id(agence) = nom-ag, "a pour agence"
id(trans) = cli-tr1, ag-tr2, c-tr3

3. gestion d'une banque : accès par défauts

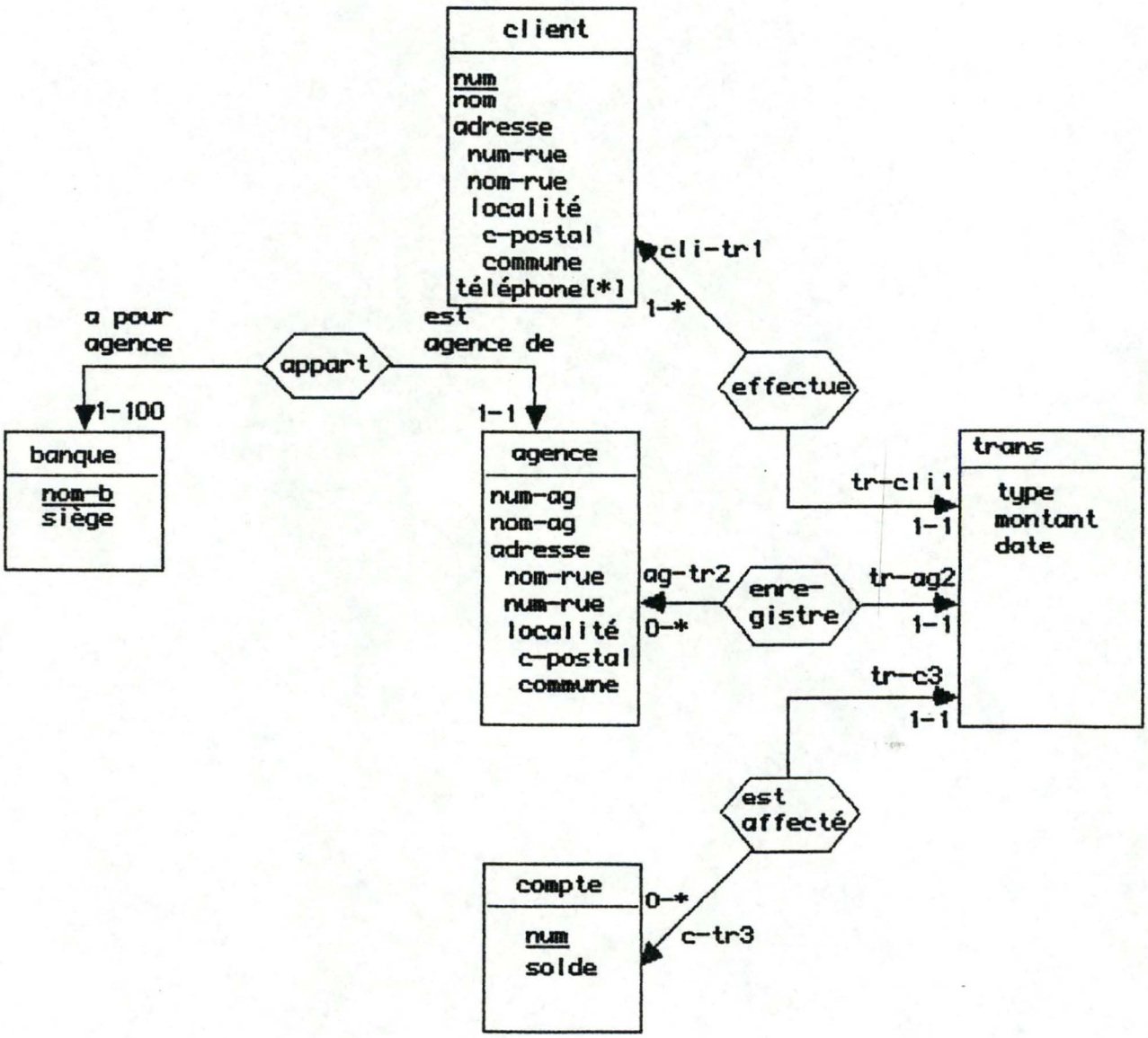


clé(client) = num
clé(compte) = num
clé(banque) = nom-b

id(agence) = num-ag, "a pour agence"
id(agence) = nom-ag, "a pour agence"
id(trans) = cli-tr1, ag-tr2, c-tr3

4. gestion d'une banque : accès nécessaires

- les identifiants définis dans un type de chemins deviennent des clés d'accès.
- l'attribut répétitif téléphone est déclaré clé d'accès.

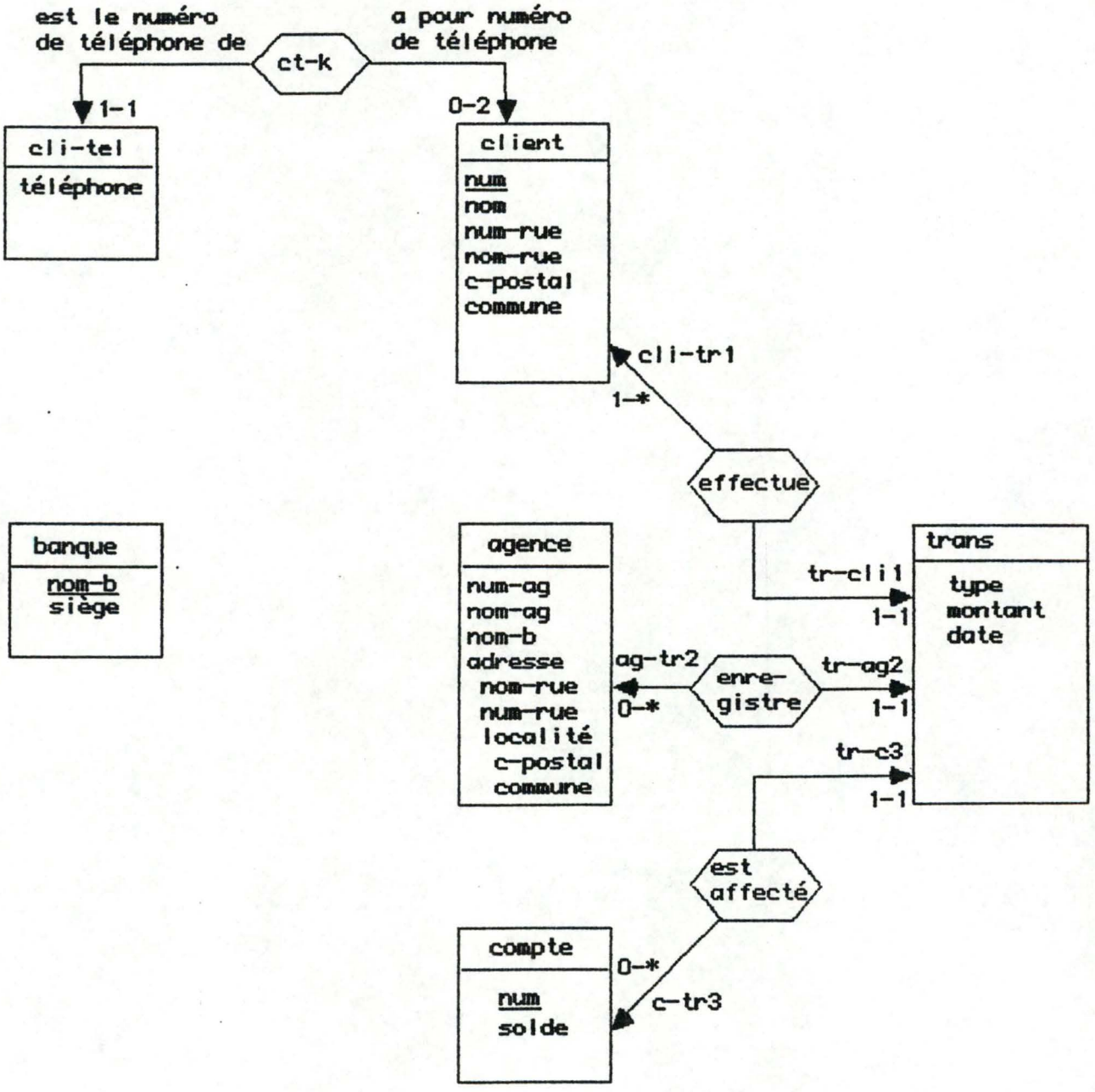


clé(client) = num
clé(client) = téléphone[*]
clé(compte) = num
clé(banque) = nom-b
clé(agence) = num-ag, "a pour agence"
clé(agence) = nom-ag, "a pour agence"

id(agence) = num-ag, "a pour agence"
id(agence) = nom-ag, "a pour agence"
id(trans) = cli-tr1, ag-tr2, c-tr3

5. gestion d'une banque : schéma transformé n°1

- transformation TA -> ATT : élimination d'un type d'associations qui est le siège de deux clés d'accès.
- transformation CLE -> TC : élimination d'une clé répétitive



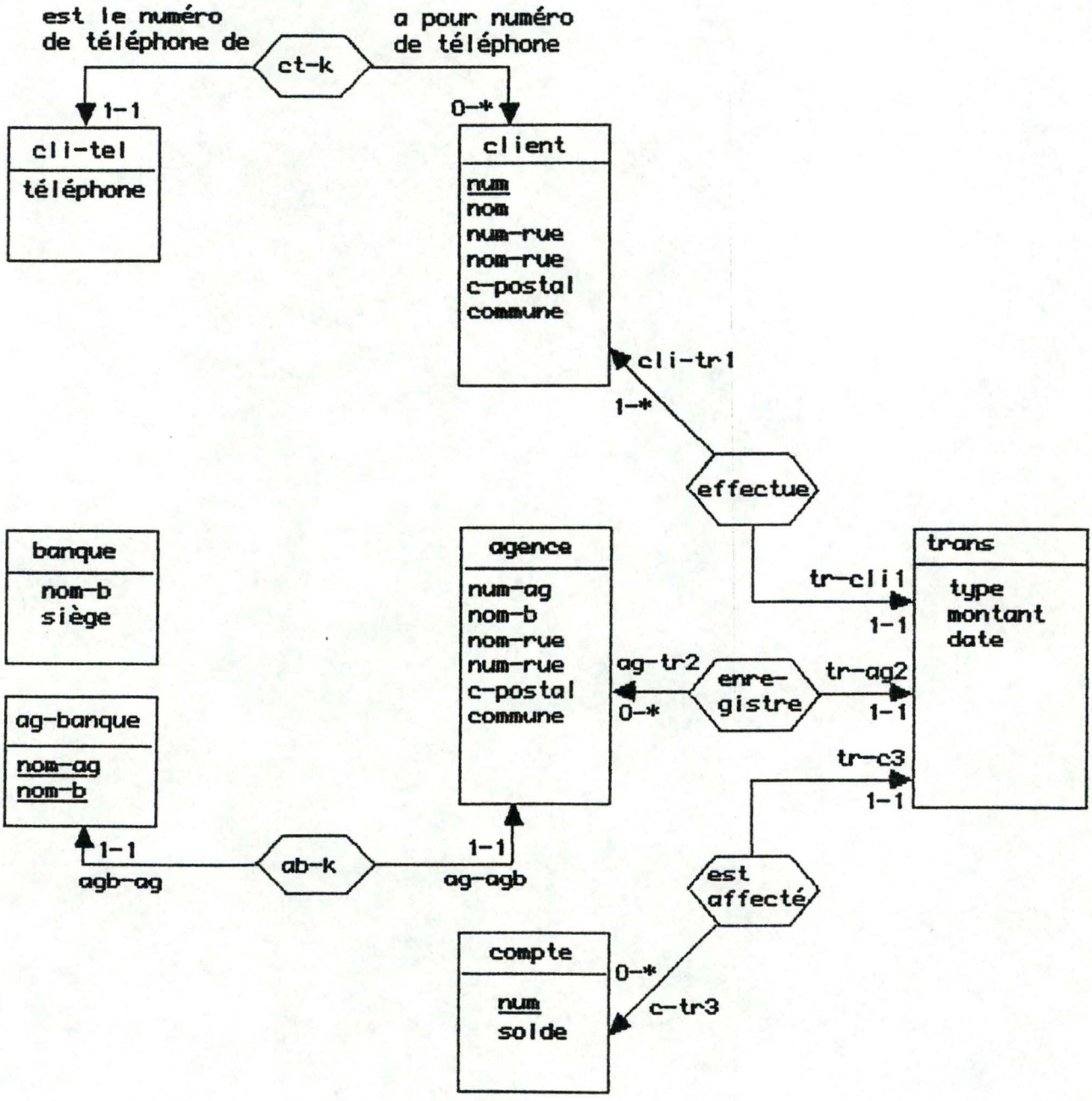
clé(client) = num
clé(cli-tel) = téléphone
clé(compte) = num
clé(banque) = nom-b
clé(agence) = num-ag, nom-b
clé(agence) = nom-ag, nom-b

id(agence) = num-ag, nom-b
id(agence) = nom-ag, nom-b
id(trans) = cli-tr1, ag-tr2, c-tr3

agence.nom-b in banque.nom-b

6. gestion d'une banque : schéma transformé n°2

- transformation CLE -> TC : élimination d'une seconde clé définie pour un type d'entité.



clé(client) = num
clé(cli-tel) = téléphone
clé(compte) = num
clé(banque) = nom-b
clé(agence) = num-ag, nom-b
clé(ag-banque) = num-ag, nom-b

id(agence) = num-ag, nom-b
id(agence) = nom-ag, nom-b
id(trans) = cli-tr1, ag-tr2, c-tr3

agence.nom-b in banque.nom-b
ag-banque.nom-b in agence.nom-b

7) SCHEMA DDL MDBS

```
/*************** data base identification ********************/

data base name is  BANQUE-M
     file "BANQUEM.DB", size is 300 pages, page size is 512 bytes


/*************** record type definitions *********************/

record name is  CLIENT
     within BANQUE-M  calc key is NUM
                      duplicates are not allowed

     item name is    NUM              integer   5
     item name is    NOM              character 30
     item name is    NUM-RUE          integer   3
     item name is    NOM-RUE          character 30
     item name is    C-POSTAL         integer   4
     item name is    COMMUNE          character 30


record name is  COMPTE
     within BANQUE-M  calc key is NUM
                      duplicates are not allowed

     item name is    NUM              integer   6
     item name is    SOLDE            integer   10


record name is  AGENCE
     within BANQUE-M  calc key is (NOM-B,NUM-AG)
                      duplicates are not allowed

     item name is    NUM-AG           integer   5
     item name is    NOM-AG           character 30
     item name is    NUM-RUE          integer   3
     item name is    NOM-RUE          character 30
     item name is    C-POSTAL         integer   4
     item name is    COMMUNE          character 30


record name is  BANQUE
     within BANQUE-M  calc key is NOM-B
                      duplicates are not allowed

     item name is    NOM-B            character 30
     item name is    SIEGE            character 30


record name is  TRANSACTION
     item name is    TYPE             character 1
     item name is    MONTANT          integer   10
     item name is    DATE             character 6
```

```
record name is  CLI-TEL
     within BANQUE-M  calc key is TELEPHONE
                      duplicates are not allowed

     item name is   TELEPHONE      character 14   occurs 2 times

record name is  AG-BANQUE
     within BANQUE-M  calc key is (NOM-B,NOM-AG)
                      duplicates are not allowed
     item name is   NOM-B          character 30


/*************** set definitions ****************************/

set name is  EFFECTUE
     type is 1:n
     owner is    CLIENT
     member is   TRANS       insertion is auto

set name is  ENREGISTRE
     type is   1:n
     owner is    AGENCE
     member is   TRANS       insertion is auto

set name is  EST-AFFECTE
     type is   1:n
     owner is    COMPTE
     member is   TRANS       insertion is auto

set name is  CT-K
     type is   1:n
     owner is    CLIENT
     member is   CLI-TEL           insertion is auto

set name is  AB-K
     type is     1-1
     owner is    AGENCE
     member is   AG-BANQUE         insertion is auto


END.
```

A N N E X E I I

GENERATION DE TEXTE MDBS A PARTIR DE L'ATELIER

P L A N

1. ABSTRACT
2. LES CARACTERISTIQUES MDBS-3
3. LA CORRESPONDANCE ATELIER --> MDBS-3
4. ARCHITECTURE D'UN GENERATEUR GENERAL
5. PROPOSITIONS DE MISE EN OEUVRE
6. GLOSSAIRE

1. ABSTRACT

Le présent document analyse le problème de la génération de texte DDL à partir du contenu de la bd de l'atelier (*). L'analyse ici décrite concerne plus spécifiquement le langage DDL du sgbd cible MDBS-3. Le problème de génération n'étant cependant pas spécifique à un sgbd cible, il paraissait souhaitable de garder une approche générale.

La phase de conception d'un atelier général à permis de définir une stucture (ou grille de lecture) de tout atelier particulier. Des primitives d'accès à cette bd sont définies (cfr. le module d'accès ADL - C).

La création de la bd locale (ex. bd salaires, bd stocks, chargement d'occurences.

Les caractérisitiques MDBS-3 sont analysées et une correspondance entre les concepts de l'atelier(*) et de MDBS-3 est établie.

L'architecture d'un **générateur général** est présentée. Elle est suivie par des **propositions de mise en oeuvre** pour le **générateur MDBS-3**.

2. LES CARACTERISTIQUES MDBS

2.1 DESCRIPTION :

(cfr. les chapitres IV et V)

2.2 LA STRUCTURE D'UN TEXTE MDBS.DDL :

Le texte DDL est structuré en **sections** de différents types : identification, user, area, record, data item, set, owner, menber. Certaines sections sont obligatoires, d'autres facultatives. Mais dans tous les cas, l'ordre entre les sections est défini et à respecter.

Une section est elle-même définie en clauses obligatoires et en clauses optionnelles. Une clause correspond à une ligne dans le texte DDL. Une présentation plus fonctionnnelle et plus esthétique pourrait regrouper plusieurs clauses sur une même ligne.

Une clause est composée de un ou plusieurs termes : keyword, identifier, filename, string, username et password.

3. CORRESPONDANCES ATELIER --> MDBS

Le lecteur est renvoyé au document "**schémas de la base de spécifications (deuxième version)**" pour ce qui concerne la description des schémas MAG restreint de l'**atelier de conception de bases de données**.

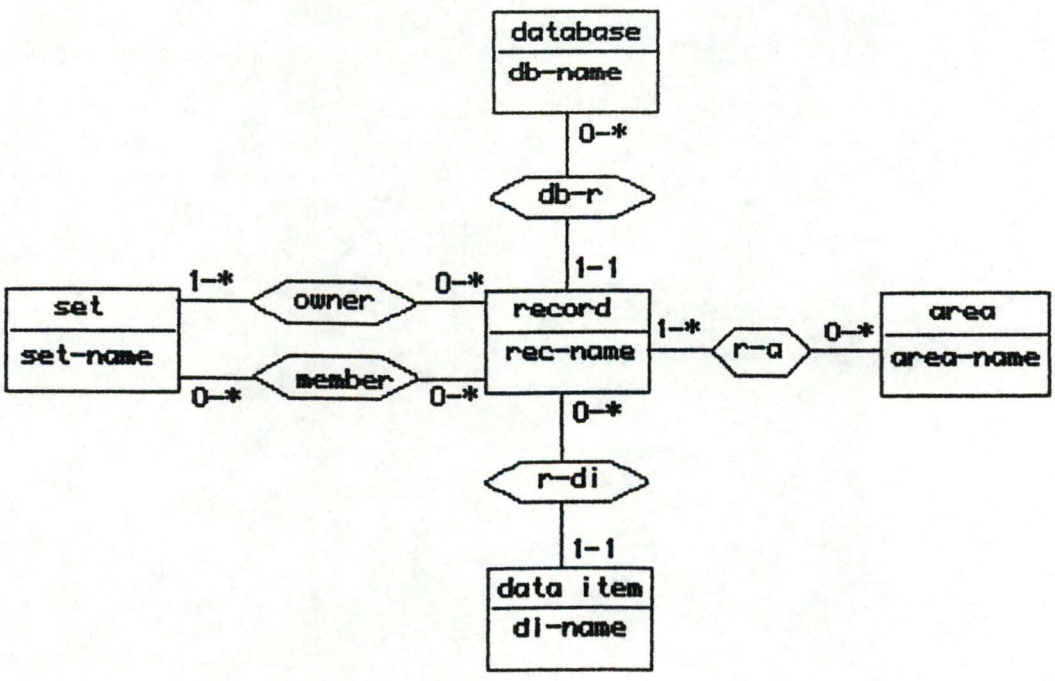
Rappelons qu'en ce qui concerne les concepts sémantiques définis dans l'atelier, il fallait réaliser un compromis entre la généralité des concepts (qui induit une grande stabilité des définitions vis-à-vis des modifications ultérieures) et la précision des définitions.

nb : lors du test du générateur MDBS-3, il faudra limiter les objets et propriétés de l'atelier général aux seuls pris en charge par l'atelier ORGA (seul proto-type existant à l'heure actuelle).

3.1 CONCEPTS ET PROPRIETES MDBS PRIS EN CHARGE PAR LE GENERATEUR MDBS

Il faudra parfois plusieurs objets MAG restreint pour accéder aux propriétés d'un objet MDBS-3. Cette désagrégation des données au niveau MAG restreint rend la correspondance parfois complexe.

nb : les propriétés soulignées des objet MDBS-3 sont obligatoires, les autres sont facultatives.



3.2 LA CORRESPONDANCE PROPREMENT DITE :

Les règles de correspondance ont le format général suivant :

```
<< OBJET MDBS-3 >>
  F R O M   << structure d'acceptation de l'ATELIER GENERAL >>
```

DATABASE :
db_name, file_name, file_size, page_size, title(5), synonym(6).

F R O M **1. SCHEMA :**
-sch_name = 'db_name'
-type = 'schéma mdbs-3'

2. PHYS-OBJECT :
-pho_name = 'db_name'
-type = 'schéma'
-file_size (param. lib.)
-page_size (param. lib.)

3. PROPERTY :
-p_rôle = 'synonym', 'title'
-p_value (5 et 6)

AREA :
area_name, file_name, file_size, page_size, title(5), synonym(6).

F R O M **1. E_COLLECTION**
-ec_name = 'fichier externe de l'area'
-classe = 'fichier'
-type = 'externe'

2. PHYS_OBJECT
-pho_name = 'fichier externe de l'area'
-type = 'area'
-file_size
-page_size

3. PROPERTY :
-p_rôle = 'synonym', 'title'
-p_value (5 et 6)

RECORD :
rec_name(1), rec_location(2), title(3), synonym(4).

F R O M **1. ENTITY-T**
-e_name (1)

2. E_COLLECTION

-ec_name (2)

3. ID-KEY-ORDER
-type = 'identifiant et clé'
-calc (param. lib.)

4. ATTRIBUTE
-at_name (2)

5. COMPONENT
-seq_nbr (2)

6. PROPERTY
-p_rôle = 'synonym', 'title'
-p_value (3 et 4)

7. PHYS_OBJECT
-pho_name = 'loc_mode'
-type = 'via' (2)

DATA ITEM :
di name(1), occur(2), encryp, range(4), format(5), title(6), synonym(7).

F R O M 1. ATTRIBUTE
-at_name (1)
-format (5)
-max_rep (2)
-encryp (param. lib.)

2. PROPERTY
-p_rôle = 'synonym', 'title', 'valeur minimale', 'valeur maximale';
-p_value (4, 6 et 7)

SET :
set name(1), type(2), ret_mode(3), title(4), synonym(5).

F R O M 1. LINK
-l_name (1)

2. ROLE
-max_con (2)
-retention (param. lib.)

3. PROPERTY :
-p_rôle = 'synonym', 'title'
-p_value (3 et 4)

nb : on accède au rôle origine et au rôle cible du link.

OWNER :
own_names(1), ins_mode(2), order(3), sort(4), duplic(5), index-width(6).

F R O M 1. ENTITY_T
-e_name (1)
-ins_mode (param. lib.)

2. ID-KEY-ORDER
-type = 1. 'ordre' si notion d'ordre sans que ce soit une clé
2. 'clé de tri'
(3) et (4)

-ordre (3) = 'fifo', 'lifo', 'next', 'prior', 'immaterial', 'immat'.

-duplic (5) = 'not allowed', 'fifo', 'lifo', 'immaterial', 'immat'.

3. COMPONENT, association définie entre ENTITY_T et ATTRIBUTE
-seq_nbr (4)
-direct (4)

4. ATTRIBUTE
-at_name (4)

5. ROLE
-r_name (4)

commentaires :

classification

1. set uni-type
-côté owner : on utilise le rôle origine du link;
-côté menber : on utilise le rôle cible du link;

1.1 notion d'ordre, à l'exception d'un ordre trié :
-accéder à un IKO de type 'ordre' ;

1.2 notion de clé de tri :
-accéder à un IKO de type 'clé de tri';
-accéder aux composants de la définition de la clé de tri : data-item dans un certain ordre (seq_nbr), et leurs directions (direct).

2. set multi-type

2.1 notion d'ordre, à l'exception d'un ordre trié :
accéder à un IKO de type 'ordre';

2.2 notion de clé de tri :
-accéder à un IKO de type 'clé de tri';
-accéder aux composants de la définition de la clé de tri : data-items et le record_name dans un certain ordre (seq_nbr), et leurs directions (direct).

question :
nb : un ordre partiel est un ordre relatif à un record type.

Les composants d'une clé de tri définis sur les occurences owner (ou member) d'un set multi-type peuvent être dans une séquence quelconque : ordre partiel avec comme premier composant record_name, ordre basé sur un ou plusieurs data items communs puis sur l'ordre partiel.
ex: soit un set multi cible avec les record types M et T. Les members multi-types d'un set forment une E_COLLECTION.

Si l'on considère un ordre au niveau des occurences, on a :
record_name M : m1, m2, m3. (ordre partiel pour les occur. de M)

record_name T : t1, t2. (ordre partiel pour les occurences de T)

e_collection M+T : -ex 1 : m1,m2,m3,t1,t2;
-ex 2 : m1,t2,T1,m2,m3;

MENBER :
mnb_names(1), ins_mode(2), order(3), sort(4), duplic(5), index-width(6).

F R O M **1. ENTITY_T**
-e_name (1)
-ins_mode (param. lib.)

2. ID-KEY-ORDER
-type = 1. 'ordre' si notion d'ordre sans que ce soit une clé
2. 'clé de tri'
(3) et (4)

-ordre (3) = 'fifo', 'lifo', 'next', 'prior', 'immaterial', 'immat'.

-duplic (5) = 'not allowed', 'fifo', 'lifo', 'immaterial', 'immat'.

3. COMPONENT, association définie entre ENTITY_T et ATTRIBUTE
-seq_nbr (4)
-direct (4)

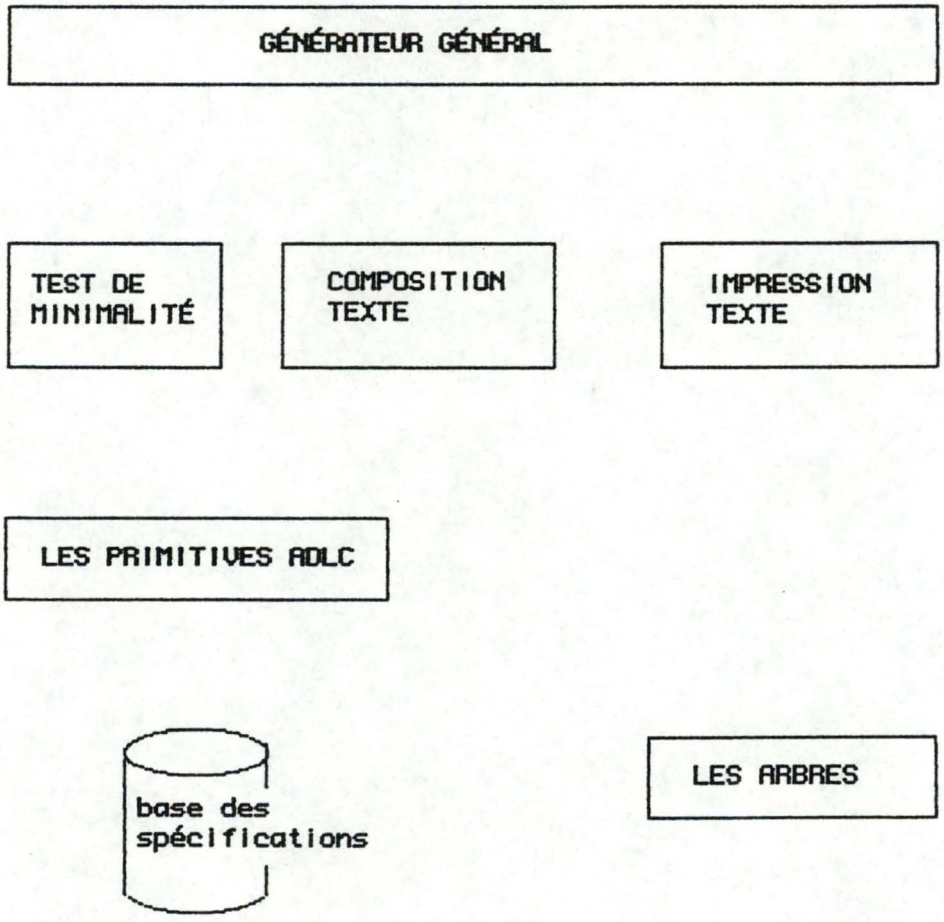
4. ATTRIBUTE
-at_name (4)

5. ROLE
-r_name (4)

commentaires :
cfr. les commentaires se rapportant à un owner d'un set.

4. ARCHITECTURE D'UN GENERATEUR GENERAL

4.1. découpe en modules :



HIÉRARCHIE "UTILISE"

1. ADLC :
module d'accès à la Base des spécifications

2. GENERATEUR GENERAL :
Ce module est un module abstrait de données qui permet de déterminer la structure (*) d'un rapport DDL (propriétés facultatives ou non des "objets" MDBS-3). Les 'statements' du rapport DDL sont également chargés à partir du DD. On dispose d'un jeu de primitives et de la structure de données ARBRE.

3. TEST DE MINIMALITE :
Ce module est spécifique à un langage DDL cible. Il s'agit de tester la minimalité de la structure du rapport DDL cible à générer.

4. COMPOSITION TEXTE :
On compose ,via le module ADL-c, le contenu du rapport DDL défini par le module GENERATEUR GENERAL. Ce module utilise également le module ARBRE.

5. IMPRESSION TEXTE :
Ce module cache toutes les informations spécifiques à un sgd cible (ordre des déclaration du langage DDL cible, formats, ...).

6. ARBRE :
structure intermédiaire.

4.2 JUSTIFICATION :

L'architecture <<utilise>> permet d'appliquer le principe d'indépendance des modules de l'architecture.

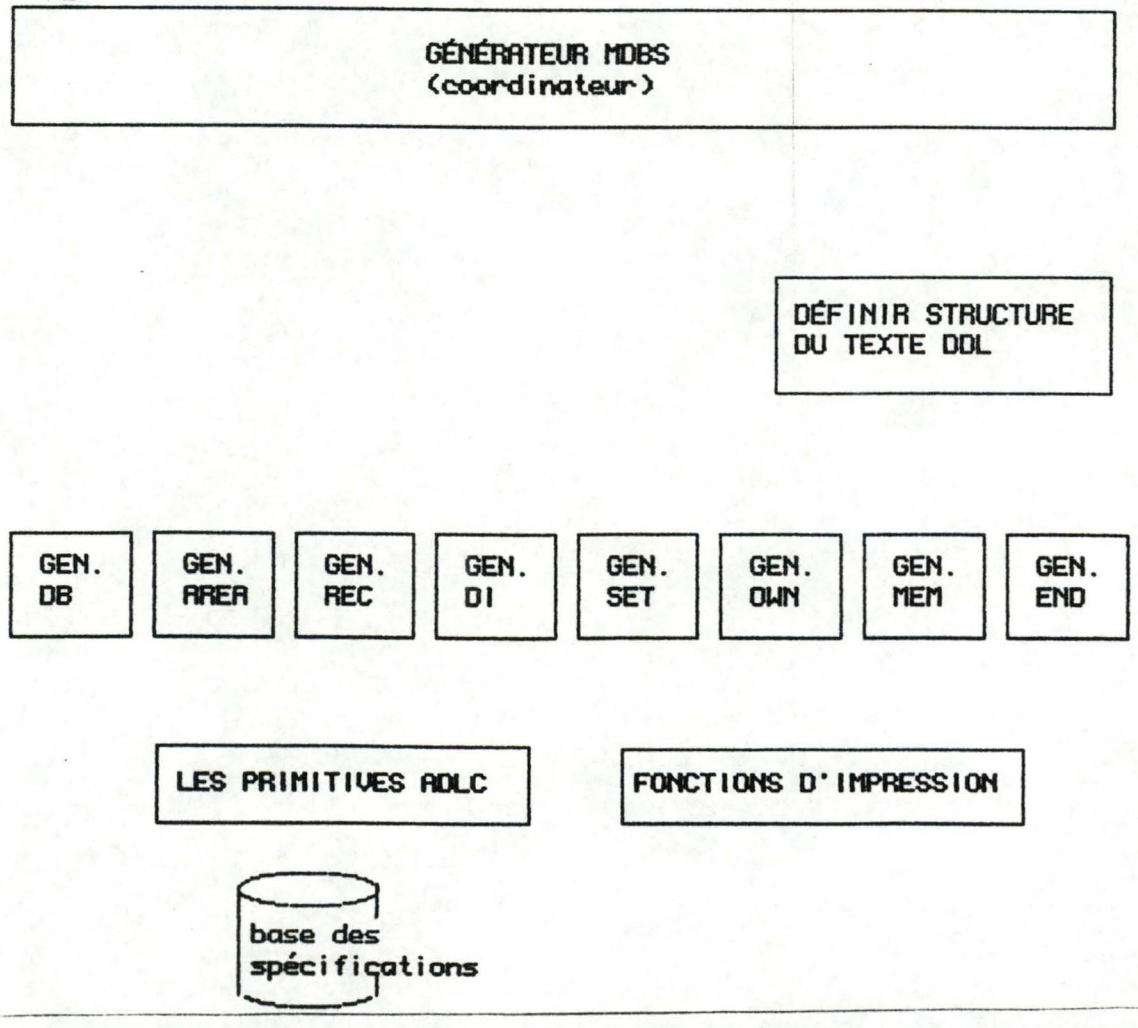
5. PROPOSITIONS DE MISE EN OEUVRE

Dans le chapitre précédent "**architecture d'un générateur général de texte DDL**", on se posait a question suivante : " Parmi les nombreuses grammaires (aspects sémantiques et syntaxiques) de langage de description de données, ne peut on pas définir une grammaire minimal et commune c-à-d indépendante d'un sgd particulier ?"

La réalisation d'un générateur paramétrisable et donc indépendan d'un SGD cible est certes une approche séduisante. En pratiqu cependant, une telle étude demanderait un investissement en temps tro important pour être envisagée dans notre mémoire. De plus un te générateur de texte DDL ne pourrait être probablement que partiellemen généralisable c-à-d restreint à une famille d'sgd (ex. sgbd de typ réseau).

Les propriétés non facultatives d'un objet (p. ex. area name) correspondent à des clauses obligatoires (p. ex. name clause) dans une section du texte DDL (p. ex. area section). De même les propriétés facultatives trouvent leurs équivalents dans les clauses optionelles du texte DDL.

5.1 ARCHITECTURE MDBS



Une fonction est définie par section MDBS-3 à générer.

On combine les accès à la bd et la production du texte DDL cible.

Le sens de parcours de la bd est imposé par l'ordre prédéfini et obligatoire de génération des sections MDBS-3.

5.2 SPECIFICATION DES MODULES DE L'ARCHITECTURE

5.2.1 conventions et déclarations préléminaires.

-les noms des fonctions sont en minuscules

-les noms des paramètres des fonctions sont en majuscules

- <I> = valeur de I

- I = adresse de I

- %f = format d'un attribut

-quelques types prédéfinis de l'atelier MAG restreint :

```
R_SCHEMA
R_E_COLLECTION
R_ENTITY_T
R_ATTRIBUTE
R_ID_KEY_ORDER
```

-quelques déclarations de variables de référence :

```
R_SCHEMA  SCH
R_E_COLLECTION E_COLL
R_ENTITY_T  ENT
R_ATTRIBUTE  ATT
R_ID_KEY_ORDER  IKO
```

5.2.2 Génération de l'"identification section"

format : gen_idt (ASCH)

condition : ASCH est l'adresse d'une variable de référence de type R_SCHEMA

fonction : génération de

DATABASE <ASCH->SCH_NAME>

exemple : gen_idt (&sch)

5.2.3 Génération de l'"area section"

format : gen_area (ASCH, AE_COLL)

condition : - ASCH est l'adresse d'une variable de référence de type SCHEMA.
- AE_COLL est l'adresse d'une variable de référence de type E_COLLECTION.

fonction : génération de

AREA <AE_COLL->EC_NAME>

exemple : gen_area (&SCH, &E_COLL)

5.2.4 Génération de la "record section"

format : gen_rec (AENT)

condition : AENT est l'adresse de la variable de référence de type R_ENTITY_T.

fonction : génération de

RECORD <AENT->E_NAME>

exemple : gen_rec(&ent)

5.2.5 Génération de la "data item section"

format : gen_att (AATT)

condition : AATT est l'adresse d'une variable de référence de type R_ATTRIBUTE

fonction : génération de

DATA ITEM <AATT->AT_NAME> %f

exemple : gen(&att)

5.2.6 Génération de la "set section"

format : gen_set (ALIN)

condition : ALIN est l'adresse d'une variable de référence de type R_LINK

fonction : génération de

SET <ASET->L_NAME>

exemple : gen_set(&set)

5.2.7 Génération de l'"owner section"

format : gen_own (ALIN)

condition : ALIN est l'adresse d'une variable de référence de type R_LINK.

fonction : génération de

-soit OWNER <ENT->E_NAME>
-soit OWNERS (<ENT->E_NAME>,)*

exemple : gen_own(&lin)

5.2.8 Génération de la "member section"

format : gen_mem (ALIN)

condition : ALIN est l'adresse d'une variable de référence de type R_LINK.

fonction : génération de

-soit MEMBER <ENT->E_NAME>
-soit MEMBER (<ENT->E_NAME>,)*

exemple : gen_mem(&lin)

6. GLOSSAIRE

1. ATELIER :

Il s'agit de l'atelier tel que décrit dans le document "**schémas de la base des spécifications (deuxième version)**". Ce document présente la descriptions des schémas MAG restreint de l'**atelier de conception de bases de données**. Un tel atelier général est en développement à l'Institut d'Informatique des Facultés Universitaires de Namur (Atelier BD --> Atelier BD++).

2. STRUCTURE D'UN RAPPORT :

On appelle structure d'un rapport DDL les objets ainsi que leurs propriétés qui apparaîtront dans le rapport DDL généré.

Les propriétés non facultatives sont inexistantes dans le cas d'un générateur général. Elles sont exigées lors du test de minimalité de structure du texte DDL cible.

Les propriétés facultatives permettent de paramétrer le générateur de texte DDL cible. Un jeu de valeurs par défaut est disponible.

A N N E X E I I I

STRUCTURE D'UN SYSTEME DE MODULES :

PROPOSITIONS DE MODIFICATIONS D' UN SOUS-SCHEMA DU SCHEMA DE L'ATELIER

Auteurs : J.L. Hainaut, A. Delcourt, P. Deville, P. Hoffelt

Namur - le 10 octobre 1986

LA STRUCTURE D'UN SYSTEME DE MODULE.

P L A N :

1. Spécifications informelles d'un système de modules

2. Modélisation E/A d'un sytème de modules.

   2.1. Schéma de la structure d'un sytème de module

   2.2. Commentaires sur le shéma 2.1.

   2.3. La représentation des types

3. Traduction du schéma conceptuel dans le schéma de l'atelier

   3.1 Introduction.

   3.2 Schéma de l'atelier révisé .

   3.3 Commentaires sur le passage du shéma 2.1 vers le shéma 3.2

   3.4 Propositions de modifications du document : 'SCHEMAS DE LA BASE DES SPECIFICATIONS (deuxième version)' - 03/07/86

4. Annexes

   4.1 Glossaire

   4.2 Références

Pour réaliser la présente modélisation des traitements, on s'est fortement inspiré du cas particulier des programmeurs en C. On s'est ensuite efforcé d'élargir le modèle en considérant d'autres langages (Pascal, Cobol,...).

1. SPECIFICATIONS INFORMELLES D'UN SYSTEME DE MODULES

Note au lecteur.
Les astérisques renvoient au glossaire en annexes.

Dans tout projet informatique, le concept de module est central. Nous reconnaissons qu'il s'agit là d'une notion fort riche susceptible de recouvrir des réalités aussi diverses que modules classiques, fonctions primitives, procédures, modules abstraits,... Nous allons d'abord considérer cette acception très large de module. Dans un second temps, nous l'étudierons de façon plus détaillée .

Il est classique d'établir une structuration entre modules. Pour l'ensemble des types de relation existant entre modules et fonctions, nous n'avons retenu que le type de relation "appelle" car nous considérons les traitements sur un plan strictement opérationnel.

L'interface utilisateur d'un module est constitué principalement des paramètres du module: ces derniers sont passés par adresse ou par valeur. D'autre part, un module dispose de variables et de constantes. D'une constante, on souhaite connaître sa valeur sous la forme d'une expression numérique ou d'une expression à calculer. En raison de leurs propriétés similaires, on regroupera les notions de paramètre, variable et constante sous le terme générique " data objet ". On peut dire notamment d'un data objet qu'il est local ou global au module. Cependant, on remarquera que parmi les variables locales, il est encore nécessaire de distinguer les locales rémanentes des locales dynamiques (*).

A un data objet, on associe un type. Ce dernier peut être standard (ex integer, ...), complexe (ex structure C, record COBOL, record PASCAL, ...) ou se rapporter à un tableau. D'un tableau, on connaît également ses dimensions.

Les modules d'une même logique peuvent être regroupés pour former un fichier source. Dans un fichier source, des types et des data objets définis en externe (*), original privé (*) ou encore en original public (*) seront également stockés. On acceptera, notamment à cause des définitions en externe, d'inclure dans un fichier source un ou plusieurs autres fichiers sources.

On souhaite de façon similaire pouvoir regrouper un ou plusieurs fichiers sources en librairie. Les logiques de regroupement sont diverses. On pense à des librairies de langage (langage 'c' par exemple), mais aussi à des librairies sans colloration aucune de langage.

Il est temps maintenant d'affiner la notion de module. Le terme générique " module " implique deux concepts qui sont le module abstrait (*) et la fonction (*).

On dira d'un module abstrait qu'il s'agit d'un regroupement logique de fonctions. Une fonction pourra ou non appartenir à un module abstrait.

Il est bon d'adjoindre une description aux modules abstraits et aux fonctions. Cette notion permet de définir dans un texte de format libre toute information jugée pertinente.

Une structure d'appel est définie d'une part entre les fonctions et les modules abstraits et d'autre part entre les fonctions et les modules abstraits pris séparèment.

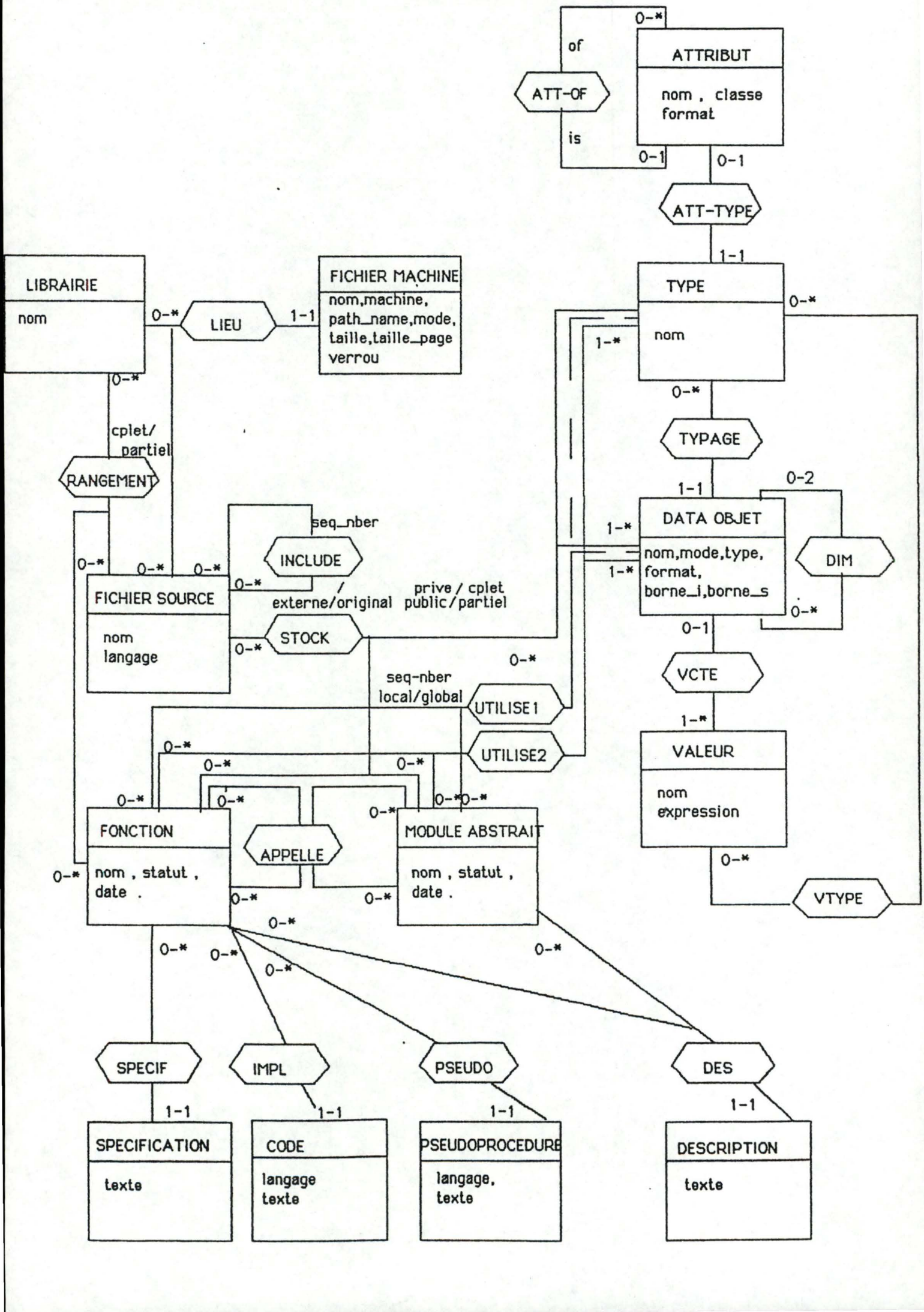
Remarquons que les agrégations de modules en fichier source et en librairie peuvent être partielles ou complètes. Ainsi par exemple les fonctions d'un module abstrait ne doivent pas être stockés dans un même fichier source.

A propos d'une fonction, on souhaite aussi modéliser son cycle de vie. On sera ainsi amené à fournir une spécification (sous la forme d'un texte libre), un peudo code (dans un langage de descrition d'algorithme) et un code par langage de programmation.

Les objets tels les fichiers sources et les librairies possèdent des attributs physiques bien précis (ex version, mode temporaire ou permanent, taille de page, ...). On regroupe ces derniers sous le concept de fichier machine.

2. MODELISATION E/A D'UN SYSTEME DE MODULES

2.1 Le schema de la structure d'un système de modules


ATTRIBUT
nom , classe
format
of
is
ATT-OF
0-*
0-1
0-1
ATT-TYPE
1-1
TYPE
nom
0-*
1-*
0-*
TYPAGE
1-1
0-2
DATA OBJET
nom,mode,type,
format,
borne_i,borne_s
DIM
1-*
1-*
0-*
0-1
VCTE
1-*
VALEUR
nom
expression
0-*
VTYPE
LIBRAIRIE
nom
0-*
LIEU
1-1
FICHIER MACHINE
nom,machine,
path_name,mode,
taille,taille_page
verrou
0-*
cplet/
partiel
RANGEMENT
seq_nber
INCLUDE
0-*
externe/original
prive / cplet
public / partiel
FICHIER SOURCE
nom
langage
STOCK
0-*
0-*
0-*
0-*
0-*
seq-nber
local/global
UTILISE1
UTILISE2
0-*
0-*
0-*
0-*
0-*
0-*
0-*
FONCTION
nom , statut ,
date .
APPELLE
MODULE ABSTRAIT
nom , statut ,
date .
0-*
0-*
0-*
0-*
0-*
0-*
0-*
0-*
0-*
SPECIF
IMPL
PSEUDO
DES
1-1
1-1
1-1
1-1
SPECIFICATION
texte
CODE
langage
texte
PSEUDOPROCEDURE
langage,
texte
DESCRIPTION
texte

2.1 COMMENTAIRES SUR LE SHEMA 2.1

Nous allons décrire les entités, les associations et les attributs du shéma 2.1.. En ce qui concerne les attributs, nous ne voudrions les figer dans une acception trop restrictive. Nous les définiront brièvement et nous donnerons éventuellement quelques exemples de valeurs. Nous ne nous arrêterons donc guère sur les attributs dont le nom même permet aisément de déduire les différrentes valeurs possibles. Nous indiquerons enfin un certain nombre de contraintes d'intégrité.

1°)module abstrait
définition: un module abstrait est défini par une structure fondamentale des données et un ensemble de primitives travaillant sur cette structure.
commentaires: la structure de donnée fondamentale correspond à un concept. Ce concept est stable et insensible à l'évolution des traitements (les primitives).

attributs :
- -nom : nom du module
- -statut : état du module
  ex. (en cours, achevé)
- -date : date de dernière modification

2°)fonction
définition: une fonction est un module classique ou une primitive de module abstrait.

attributs :
cfr module abstrait

3°)appelle
commentaires: parmi l'ensemble des types de relation existant entre modules et fonctions, nous n'avons retenu que le type de relation 'appelle' car nous considérons les traitements sur un plan strictement opérationnel. Néanmoins, nous devons être conscients qu'il y a d'autres types de relation : 'utilise', 'import/export', 'fait partie de',... Dans le cas d'une relation 'fait partie de', une fonction peut être décomposée en fonctions plus élémentaires.

4°)specification

commentaires: Une fonction peut faire l'objet de plusieurs spécifications (spécif. internes, spécif. externes, ...). Cependant pour un même type de spécification, on exigera une certaine normalisation au niveau de l'atelier (ex. méthode pré - post).

attributs:
-texte : le texte d'une spécification
-type : type de spécification
(interne, externe, ...)

5°)code
attributs:
-langage : nom d'un langage de programmation
-texte : texte rédigé dans un langage de programmation

c.i. : il est patent que le langage utilisé pour le texte codé doit être celui qui est mentionné dans l'attribut "langage".

commentaires: à une fonction peut correspondre plusieurs textes codés. En effet, une même fonction peut faire l'objet de codage dans différents langages.

6°)pseudocode
attributs:
-langage : nom d'un pseudo-langage
ex. (dsl, psl, ou encore un pseudo-langage que l'utilisateur s'est lui-même défini).
-texte : texte rédigé dans un pseudo-langage
c.i.: il est patent que le pseudo-langage utilisé pour le texte "pseudo-codé" doit être celui qui est mentionné dans l'attribut "langage".

7°)description
commentaires: il est clair qu'une fonction ou qu'un module peut être associée à plus d'un texte de description.

attributs:
-texte : texte d'une description
avec découpe en paragraphes possibles.

8°)data objet
définition: un data objet est soit une constante, soit une variable, soit un paramètre.

attributs:
-nom : nom du data objet
-mode : comment pourra t'on utiliser le data objet
(
i : en input seulement
o : en output seulement
io : en input et en output
)
-type : quelle est la catégorie du data objet
(
c : constante
lr : local rémanent
ld : local dynamique
p : paramètre
f : fonction
)
-format : le type de data objet

-b.i. : borne inférieure

-b.s. : borne supérieur

c.i. :

-concerne l'attribut format : ce dernier contient le type du data objet pour autant que le type corresponde à un type prédéfini dans le langage de programmation utilisé.

-concerne la b_i / b_s : si le data objet correspond à une variable de type tableau

ALORS
1) ce tableau est uni-dimensionnel.
2) si la b_i / b_s n'est pas une constante,
- alors : b_i / b_s contient une valeur numérique

- sinon : b_i / b_s contient une valeur "neutre" et le data objet est associé via dim à un autre DO

SINON
la b_i / b_s contient une valeur "neutre" et le data objet n'est associé à aucun autre data objet.

9°)utilise1
Les fonctions (F) et les modules abstraits (M) sont associés à un ou n datas objets (DO) :

commentaires : les caractéristiques local/global sont des modalités d'utilisation entre l'objet fonction/module abstrait

et l'objet data objet. En effet, il est possible d'avoir des procédures imbriquées notamment en PASCAL. Ce qui veut dire qu'une variable pourrait très bien être locale à une procédure alors qu'elle serait globale pour une autre procédure.

attributs :
- -seq_nber : numéro de séquence
- -mode : DO est t'il local ou global par rapport à F/M ?
  (
  local
  global
)
- -passage : DO est-il un paramètre passé par valeur ou par adresse ?
  (
  valeur
  adresse
)

-c.i.: un module abstrait n'entretient pas de relation "utilise1" avec un DO de type "p".
-c.i.: l'attribut passage n'a de signification que si la relation "utilise1" concerne un DO de type "p".

10°)utilise2
Les fonctions (F) et les modules abstraits (M) sont associés à un ou n types (T).

commentaires : Cette relation permet de modéliser des situations telles que la déclaration d'un type non utilisé dans une déclaration de varaiable

attributs :
cfr utilise1

11°)type
attributs :
- -nom : nom d'un type
- -b_i : borne inférieure
- -b_s : borne supérieure

c.i. : en aucun cas, le nom ne pourra correspondre à un type prédéfini dans le langage de programmation utilisé. En d'autres termes, le nom doit référer un type qui a été défini par l'utilisateur lui-même.

12°)valeur
attributs :
- -nom : nom d'une valeur
  ex. (max, ...)
- -expression : expression numérique ou fonction numérique à calculer

13°)attribut
attributs :
-nom : nom d'un attribut
-classe : quelle est la catégorie d'attribut
ex. (complexe,standard, ...)
-c.i. : si attribut n'est pas associé à un type, alors classe recoit une valeur neutre.
-format : nom d'un type prédéfini dans le langage de programmation utilisé
-c.i. : si la classe = "standard", alors le format devra s'instancier en un nom de type prédéfini dans le langage de programmation.

14°)fichier source
définition: un fichier source est une collection logique de modules abstraits ou de fonction

commentaires : un fichier source comprend
1. des "include" d'autres fichiers sources
2. des data objets
3. des types
4. des textes codés

attributs :
-nom : nom d'un fichier source
-langage : nom du langage de programmation utilisé au niveau du fichier source
c.i.: si un fichier source contient des data objets de type "externe" ou des types de type "externe", alors ce fichier source doit participer à une occurrence d'une association "include".

15°)stock
soit DO un data objet et un f un fichier source auquel DO est associé

attributs :
-mode : indique si le fichier source se rapporte à toutes les primitives du module abstrait ou seulement à une partie.
(
- cplet : le fichier source porte sur toutes les primitives du module abstrait
- partiel : le fichier source porte sur quelques primitives du module abstrait
)

-statut : indique une propriété d'un data objet par rapport à un fichier source
(
- externe : DO a déjà été déclaré dans un fichier source autre que f.

- original public : DO concerne d'autre(s) fichier(s) source(s) que f.
- original privé : DO se rapporte exclusivement à f.
)

c.i :
SI le FICHIER SOURCE se rapporte à un ou plusieurs MODULES ABSTRAITS

ALORS : * TYPE : ce sont les TYPES du ou des module abstrait
* DATA OBJET : ce sont les DATA OBJETS du ou des modules abstraits.
* mode : si mode = "cplet"
alors :
f est associé à toutes les primitives définissant le ou les modules abstraits

sinon (mode = "partiel") :
f est associé à une partie des primitives définissant le ou les modules abstraits.

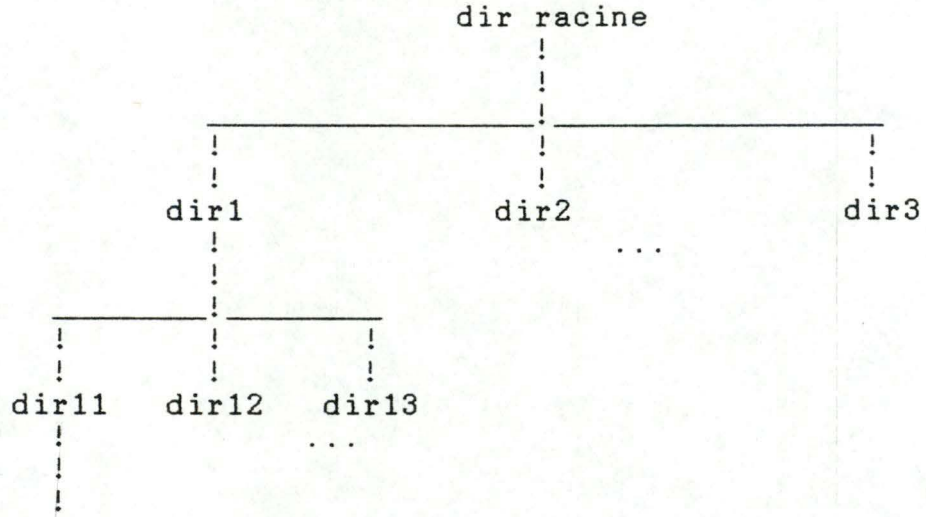
SINON(le fichier source se rapporte à une ou plusieurs fonctions)
* TYPE : ce sont les TYPES du ou des fonctions
* DATA OBJET : ce sont les DATA OBJETS du ou des fonctions.
* mode : mode a la valeur neutre

16°)librairie

attributs :
-nom : nom d'une librairie

17°)rangement
soit l une librairie
attributs :
-mode : indique si le contenu de l se rapporte à un ou plusieurs fichiers sources ou à quelque(s) fonction(s)
(
cplet : le contenu de l se rapporte à un fichier source
partiel : le contenu de l se rapporte à quelque(s) fonction(s)
)
c.i. : si mode = "cplet",
alors : l est associé à l'objet " fichier source "
sinon (mode = "partiel") :
l est associé à l'objet " fonction "

18°)fichier machine
commentaires: souvent on utilise l'image d'un arbre pour appréhender la notion d'un système de directories. Le path_name indique le chemin qui relie la directory racine à la directory feuille qui contient le fichier source.



-mode : permanent,temporaire,type,déclaration,source,...
-taille : c'est un paramètre physique indiquant la taille du fichier machine
-taille_page : c'est un paramètre physique
-verrou : c'est un paramètre portant sur la protection du fichier machine

ex. (
accès en lecture seulement ?
accès en écriture seulement ?
accès en lecture/écriture ?

accessible par tous ?
accessible par un groupe ?
accessible par une seule personne ?
)

attributs :
-nom : nom d'un fichier source
-machine : nom d'une machine
-path_name : énoncé de directories

2.3 LA REPRESENTATION DES TYPES

La modélisation des types s'inscrit dans une problématique plus large qui est celle de la déclaration des variables. Fondamentalement, la notion de déclaration fait intervenir les notions de TYPE et de DATA OBJET. Un data objet, soit DO, doit toujours être associé à un type, soit T.

Nous retenons trois propriétés relative à toute déclaration de data objet.

1°)le type de T
le type de T peut être **simple** ou **complexe**.
T est simple s'il correspond directement ou indirectement à un type prédéfini dans le langage de programmation utilisé. Dans tout autre cas, T est de type complexe.
ex. 'C' :
typedef int nbre;
int correspond directement au type prédéfini "int"
tandis que nbre correspond indirectement au type prédéfini "int".

2°)dimension de DO
Est_ce que DO est un tableau ou non ?

3°)le type de DO
La déclaration de DO est implicite si
DO est associée à un type T qui est un type prédéfini
dans le langage de programmation utilisé.

La déclaration de DO est explicite si DO est associé à un type T1 tel que T1 a été défini par le programmeur à partir de type(s) T2i (avec i variant de i à n) déjà définis ou prédéfinis dans le langage de programmation utilisé.
ex. (déclaration PASCAL)
```
const LIMIT = 23;
type nbre = integer;
var  i : nbre   ;
```
La déclaration relative à i est dite explicite .

En combinant ces trois propriétés, nous concluons à l'existence de huit types de déclarations.

1°)déclaration d'un DO de type standard
1- déclaration implicite
ex1. (déclaration C)
```
char    c ;
```
ex2. (déclaration C)
```
int i ;
```

représentation.
La representation d'une telle déclaration est relativement aisée. Les DO, bien qu'ayant un type, ne sont pas associés à un objet TYPE. Toutes les informations se rapportant au DO sont stockées au niveau de la variable.

Prenons un exemple.

```
DATA OBJET
- nom    :  c
- mode   :  ...
- type   :  ...
- format :  char
- borne_i:  valeur neutre
- borne_s:  valeur neutre
```

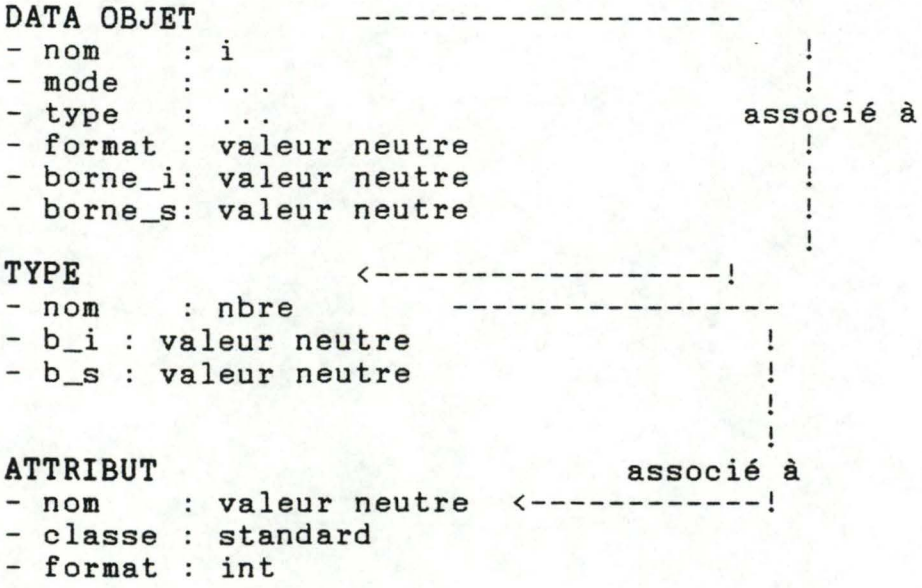
2- déclaration explicite

```
ex1.  ( déclaration PASCAL )
      type nbre = integer;
      var  i : nbre   ;

ex2.  ( déclaration C )
      typedef int nbre ;
      nbre i           ;
```

représentation
Il faut recourir à la création d'un objet type pour représenter de façon explicite notre type standard. Cet objet type contient le nom du type explicite. De plus dans notre exemple, l'objet type est associé à un objet attribut qui contient le type -prédéfini se rapportant au type explicite.

Prenons ex1.



2°)déclaration d'un DO de type complexe

1- déclaration implicite

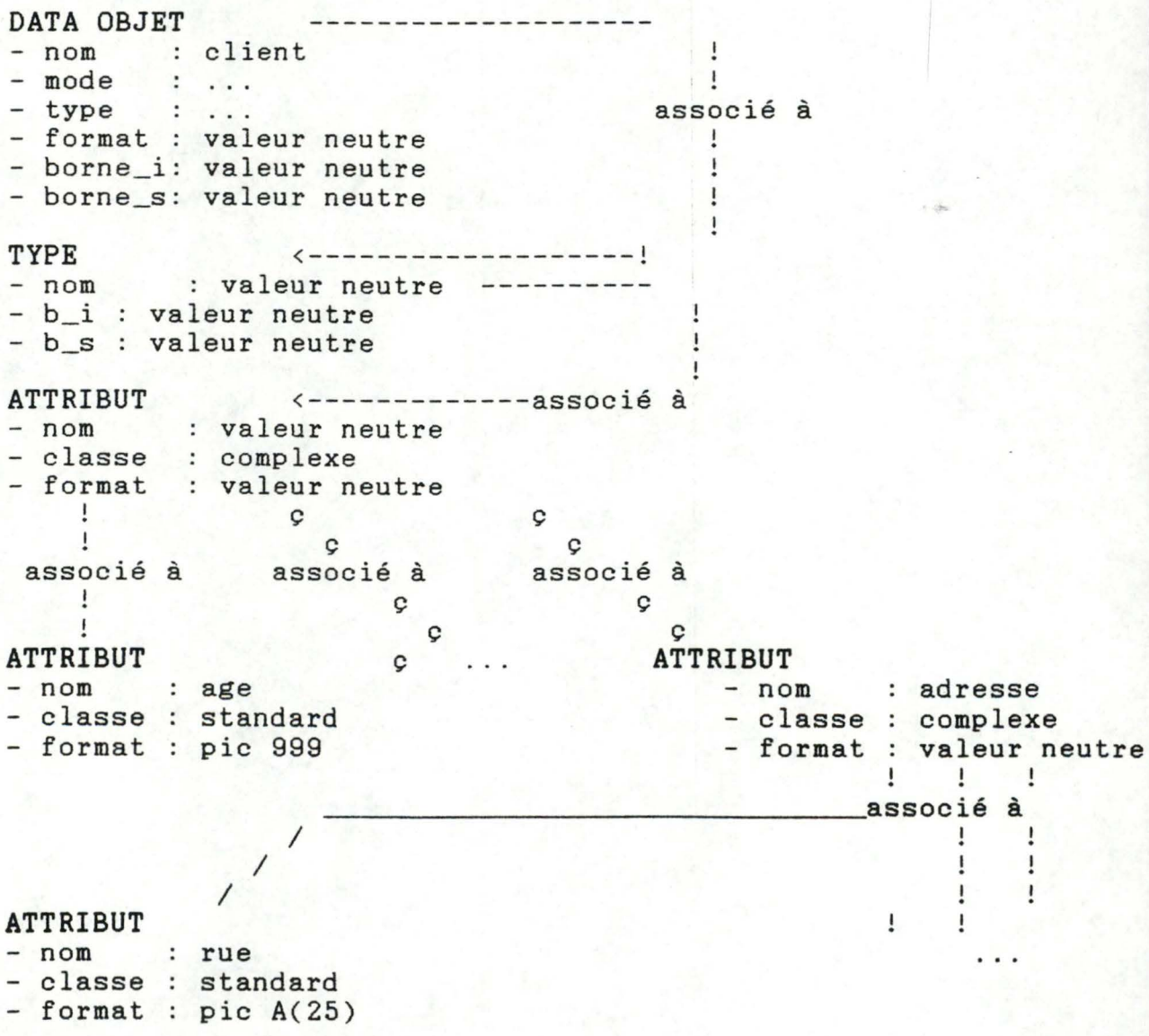
ex. (déclaration COBOL)

```
01  client
     02  age        pic 999
     02  nom        pic A(25)
     02  adresse
          03  rue pic        pic A(25)
          03  code_postal    pic 9(4)
          03  localite       pic A(25)
```

représentation

Il faut recourrir à la création d'un objet type pour représenter notre type complexe. Cet objet type ne contient pas de nom . L'objet type est associé à un objet attribut sans nom.Cet attribut se décompose en autant d'attributs qu'il y a de champ dans le type complexe. Pour chaque champ décomposable, l'attribut correspondant se décompose et ainsi de suite.

Reprenons ex.



2- déclaration explicite

ex. (déclaration d'une structure C)

```
struct adresse (
                    char          rue(25) ;
                    int code_postal       ;
                    char    localite(25) ;
                    ) ;

struct  client (
                    int                     age ;
                    char                nom(25) ;
                    struct adresse adr      ;
                    ) ;

typedef  client ta2°2§      ;

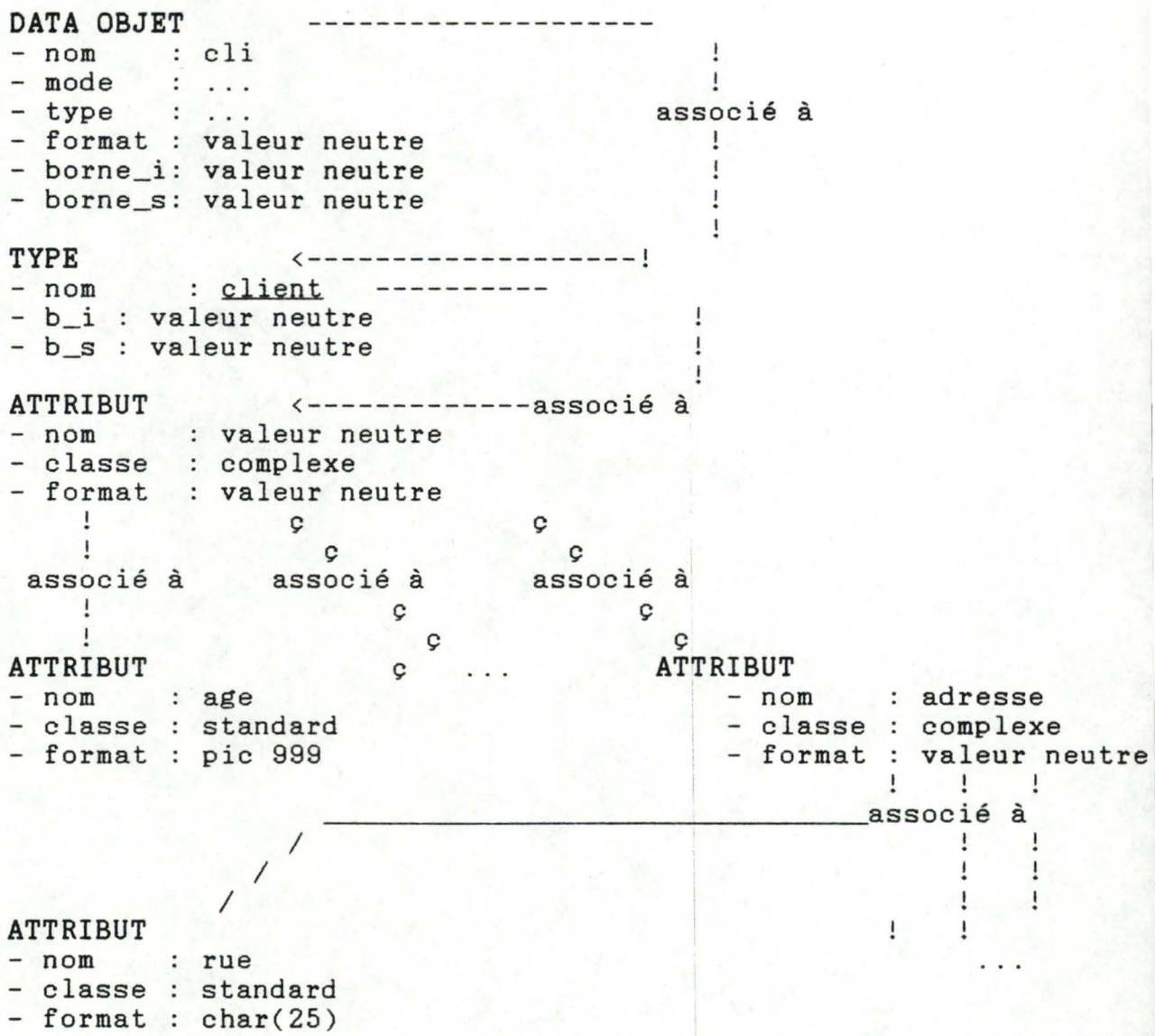

struct  buffer (
                    ta2  tab    ;
                    int  nombre ;
                    ) ;

struct buffer  b   ;
struct client  cli ;
```

représentation

Il faut recourrir à la création d'un objet type pour représenter de façon explicite notre type complexe. Cet objet type contient le nom de notre type complexe (ce qui constitue la seule différrence de représentation avec le type "implicite complexe") L'objet type est associé à un objet attribut sans nom.Cet attribut se décompose en autant d'attributs qu'il y a de champ dans le type complexe. Pour chaque champ décomposable, l'attribut correspondant se décompose et ainsi de suite.

Reprenons l´ exemple pour cli.



3°)<u>déclaration d´un DO tableau de type standard</u>
Un tableau de type standard est la juxtaposition de variables du même type standard.

1- déclaration implicite
ex. (déclaration C)
char tab°20§ ;

représentation
Notre DO est une juxtaposition de variables d´un même type standard, soit T. Nous représentons ce type T au niveau de l´objet variable de la même manière que nous l´avons fait pour "déclaration implicite d´un DO de type standard"
Pour ce qui est du problème des dimensions de notre DO,nous vous renvoyons au point 2.2.9.

Reprenons ex.

```
DATA OBJET
- nom    :  c
- mode   :  ...
- type   :  ...
- format :  char
- borne_i:  0
- borne_s:  20
```

2- déclaration explicite

ex1. (déclaration d'une tableau PASCAL)

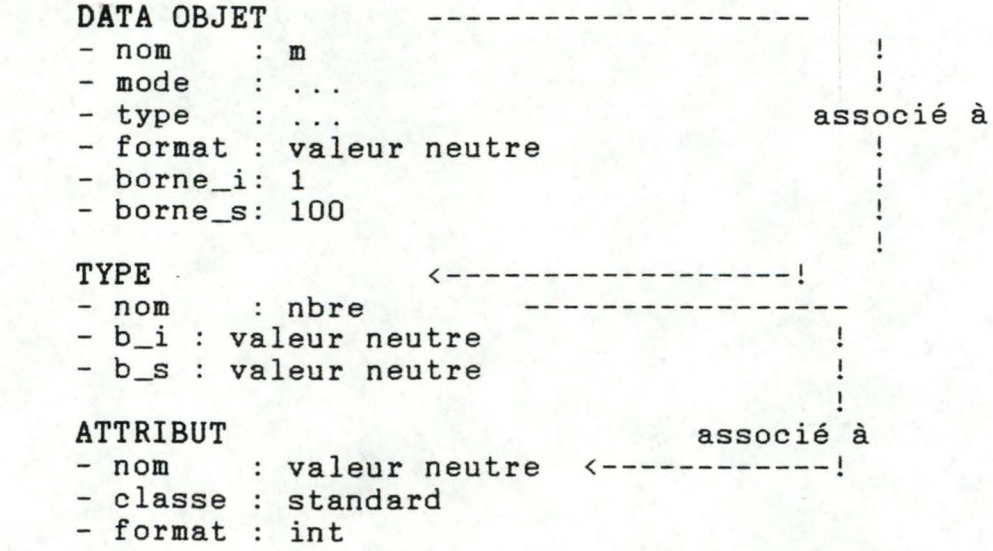
```
type  nbre = int ;
      ta100 = array°1..100§ of nbre ;
var   m    : ta100 ;
```

ex2. (déclaration C)

```
typedef char tab°20§ ;
tab  A               ;
```

représentation
Notre DO est une juxtaposition de variables d'un même type standard, soit T. Pour représenter T, nous adaptons la solution qui a été présentée pour "déclaration explicite d'un DO de type standard".
Pour ce qui est du problème des dimensions de notre DO,nous vous renvoyons au point 2.2.9.

Prenons ex1.



4°)<u>déclaration d'un DO tableau de type complexe</u>

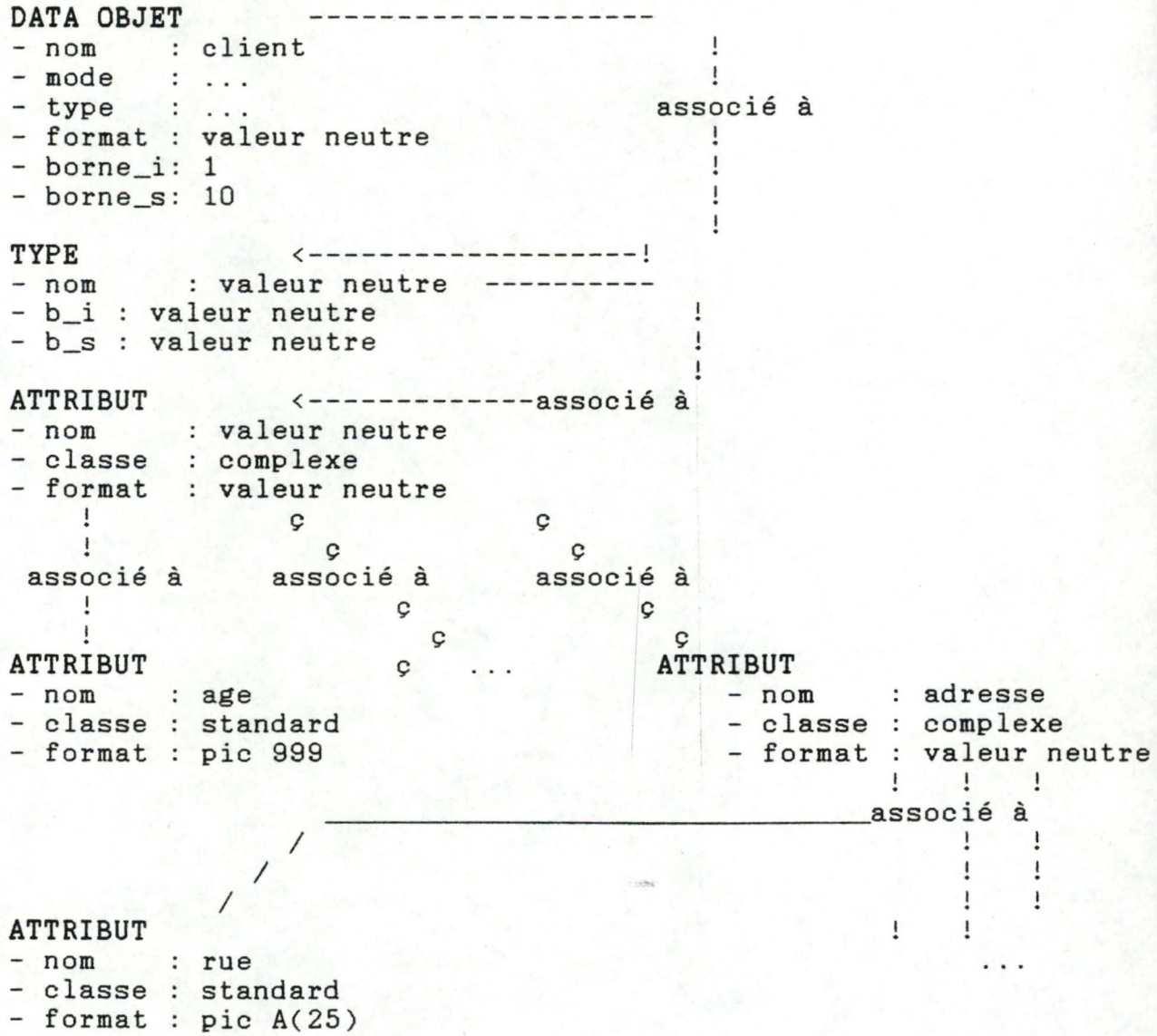
Un tableau de type complexe est la juxtaposition de variables du même type complexe.

1- déclaration implicite

```
ex.  ( déclaration COBOL )
     01  client          occurs 10 times
          02  nom        pic a(25)
          02  age        pic 999
          02  adresse
               03  rue pic        pic a(25)
               03  code_postal    pic 9(4)
               03  localite       pic A(25)
```

représentation
Notre DO est une juxtaposition de variables d'un même type complexe, soit T. Pour représenter T, nous adaptons la solution qui a été présentée pour "déclaration implicite d'un DO de type complexe".
Pour ce qui est du problème des dimensions de notre DO,nous vous renvoyons au point 2.2.9.

Prenons un exemple.



2- déclaration explicite

ex. (déclaration PASCAL)

```
type
client = record
     age : int ;
     nom : array°1..25§ of char ;
     adresse :  string°25§ ;
end;
t3 = array °1..3§ of client ;

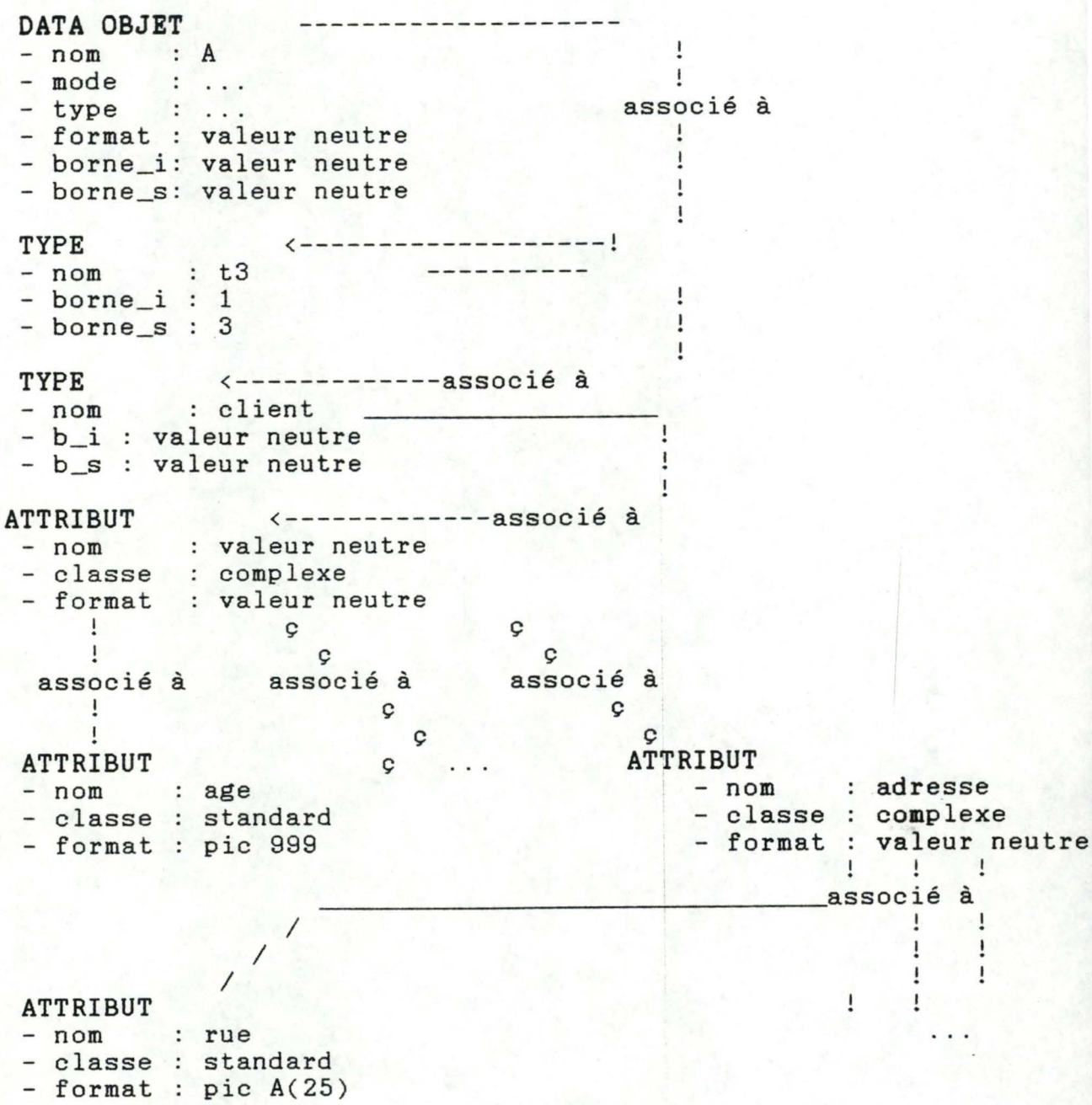
var
A : t3 ;
```

<u>représentation</u>

Notre DO est une juxtaposition de variables d'un même type complexe, soit T. Pour représenter T, nous adaptons la solution qui a été présentée pour "déclaration explicite d'un DO de type complexe".

Pour ce qui est du problème des dimensions de notre DO,nous vous renvoyons au point 2.2.9.

Prenons un exemple.



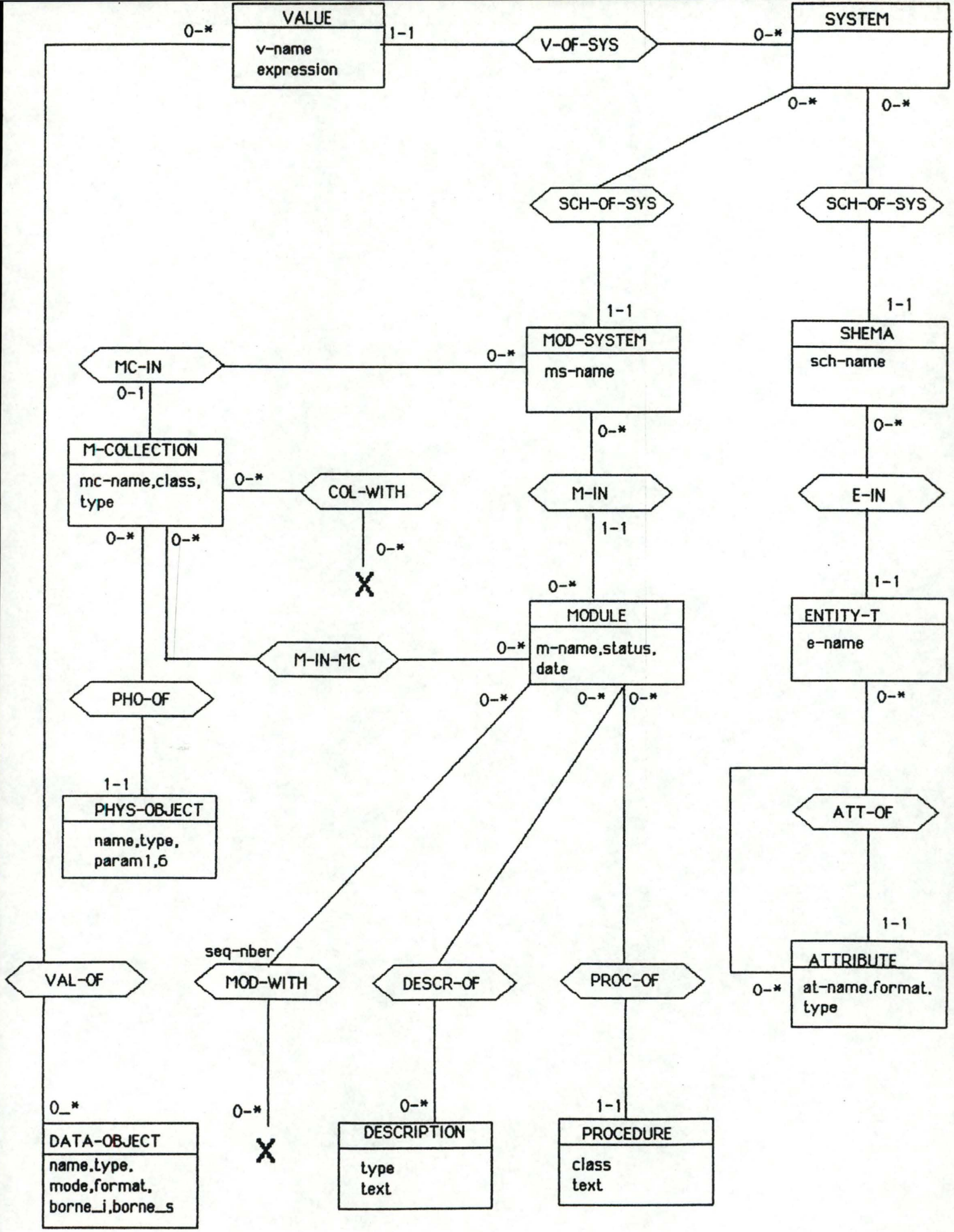
3. TRADUCTION DU SCHEMA CONCEPTUEL DANS LE SCHEMA DE L'ATELIER

3.1 introduction

Le schéma de l'atelier, général dans ses concepts, devrait être capable de modéliser toute situation conceptuelle. En d'autres termes, on entend que toute modélisation devrait trouver une structure d'acceptation dans les concepts développés dans l'atelier.

Si la perception de la description des traitements (au travers du schéma conceptuel) est suffisamment complète, cette dernière constitue une validation à postériori du schéma de l'atelier. Un tel feedback, et d'autres, permettront une grande stabilité des concepts de l'atelier.

3.2 Shéma de l'atelier révisé


VALUE
v-name
expression
V-OF-SYS
SYSTEM
SCH-OF-SYS
SCH-OF-SYS
MOD-SYSTEM
ms-name
SHEMA
sch-name
MC-IN
M-COLLECTION
mc-name,class,
type
COL-WITH
M-IN
E-IN
MODULE
m-name,status,
date
M-IN-MC
ENTITY-T
e-name
PHO-OF
PHYS-OBJECT
name,type,
param1,6
ATT-OF
ATTRIBUTE
at-name,format,
type
VAL-OF
seq-nber
MOD-WITH
DESCR-OF
PROC-OF
DATA-OBJECT
name,type,
mode,format,
borne_i,borne_s
DESCRIPTION
type
text
PROCEDURE
class
text
0-*
1-1
0-1

3.3 commentaires sur le passage du shéma 2.1 vers le shéma 3.2

Les objets MODULE ABSTRAIT et FONCTION deviennent des objets MODULES.

Les objets DESCRIPTION gardent ce même nom.

Les objets SPECIFICATION, CODE et PSEUDOPROCEDURE deviennent des objets DESCRIPTION.

Les objets LIBRAIRIE et FICHIER SOURCE deviennent des objets M_COLLECTION.

Les objets FICHIER MACHINE deviennent des objets PHYS_OBJECT.

Les objets DATA OBJET deviennent des objets DATA OBJECT.

Les objets TYPE deviennent des objets ENTITY_T.

Les objets VALEUR deviennent des objets VALUE.

Les objets ATTRIBUT deviennent des objets ATTRIBUTE.

3.4 Propositions de modifications du document : ' shémas de la base des specifications ', (deuxième version)

Nous proposons quatre modifications du schéma de l'atelier tel que mis à jour en date du 20/08/86.

Il s'agit tout d'abord du type d'objets DATA-OBJECT qui est une généralisation du concept de PARAMETER existant dans l'atelier. Ce nouveau type d'objet prétend englober le concept de paramètre bien sûr, mais aussi des nouveaux concepts comme variable et constante.

La deuxième modification concerne l'ajoût de l'association COL_WITH. En effet, une m_collection est liée à des types, à des data objects et à d'autres m_collections.

La troisième modification concerne l'ajoût de l'association MOD_WITH. En effet, un module doit pouvoir être rattaché à des entity_t, à des data objects et à d'autres modules.

La quatrième modification concerne l'ajoût de l'association DO_WITH. En effet, un data objet doit pouvoir être rattaché à un autre data object.

3.4.1. DATA-OBJET

Un DATA-OBJET est identifié par son nom (DO-NAME). Il peut être de différent type : P(aramètre, C(onstante, L(ocal R(émanent, L(ocal D(ynamique, F(onction. Il est encore caractérisé par son FORMAT dans les les plus simples ou par la référence à un objet de la base de données dans les cas les plus élaborés (ex. message structuré, écran, module, ...).

Lorsqu'il s'agit d'un tableau (simple ou complexe), on donnera également la dimension du tableau c-a-d une référence soit vers un objet VALUE si la taille est donnée sous forme numérique, soit vers un objet DATA-OBJET si la taille est définie comme étant une constante.

4.2. TYPE

Des choix de représentation (explicité antérieurement) ont établit que seuls les variables de type complexe sont associées (via TYPAGE) à un objet TYPE. Les variables de type simple possèdent quand à elles un attribut format qui définit leur type.

Un objet TYPE a un attribut unique, de nom sans signification. Cet attribut peut être décomposé pour exprimer la structure du type complexe : utilisation, si nécessaire, sera faite de l'objet RELATION dans l'analyse de la structure du type complexe (par exemple s'il faut référencer une CONSTANTE ou un TYPE prédéfini).

4 ANNEXES

4.1 GLOSSAIRE

1°) fonction
unité programmée.

2°) module abstrait
collection de primitives travaillant sur une structure de donnée commune.

3°)variable locale rémanente
si une variable v est déclarée au niveau d'une fonction f,
alors
v est dite rémanente
ssi
pour chaque appel à f,
la valeur initiale de v est égale à la valeur finale de v lors de l'appel précédent.

4°) variable locale dynamique
si une variable v est déclarée au niveau d'une fonction f,
alors
v est dite dynamique
ssi
pour chaque appel à f,
la valeur initiale de v doit être fixée par une instruction de f.

5°) déclaration interne d'un data objet
si une variable globale v est déclarée comme étant interne à un fichier f,
alors
v est globale pour tout module abstrait ou toute fonction impliquée dans f. Nous dirons que v est globale à f.
Si v est déclarée " interne publique ",
alors
v pourra être considérée comme globale à un ensemble de fichiers, soit f, F1, ..., Fn
pour autant que v ait été déclarée comme " externe "(cfr ci-dessous) pour chaque Fi avec i variant de 1 à n.
Si v est déclarée " interne privée ",
alors
v ne peut être globale qu'à f.

6°) déclaration externe d'un data objet
si une variable globale v est déclarée comme étant externe à un fichier f,
alors
v est globale pour tout module abstrait ou toute fonction impliquée dans f. Nous dirons que v est globale à f.
En fait, si v a été déclarée comme externe pour une suite de fichiers F1, ...,Fn et comme interne publique dans un fichier F, alors v est dite globale à l'ensemble f,F,F1, ...,Fn.

4.2 REFERENCES

°1§ J-L Hainaut, 'SCHEMAS DE LA BASE DES SPECIFICATIONS'