HAL
open science

Gestion d'objets composés dans un SGBD : cas particulier des documents structurés

José Valdeni de Lima

To cite this version:

José Valdeni de Lima. Gestion d'objets composés dans un SGBD : cas particulier des documents structurés. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1990. Français. NNT : . tel-00337074

HAL Id: tel-00337074

https://theses.hal.science/tel-00337074v1

Submitted on 6 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Tu 9076

THESE

présentée par

José Valdeni DE LIMA

pour obtenir le titre de DOCTEUR

de L'UNIVERSITE Joseph FOURIER (GRENOBLE I)
(arrêté ministériel du 5 juillet 1984)

Spécialité : Informatique

GESTION D'OBJETS COMPOSES DANS UN SGBD . CAS PARTICULIER DES DOCUMENTS STRUCTURES

Thèse soutenue le 20 mars 1990 devant la commission d'examen

Président	:	Y. CHIARAMELLA
Rapporteur	:	J. KOULOUMDJIAN
Rapporteur	:	C. CHRISMENT
Examinateurs	:	M. LOPEZ
		M. ADIBA
		J.P. GIRAUDIN
		V. QUINT

Thèse préparée au sein du Laboratoire de Génie Informatique

RESUME

Cette thèse traite du problème de la gestion des documents structurés multimédia dans un SGBD. Par gestion, nous entendons la modélisation, la manipulation, le stockage et l'accès aux documents. Nous présentons un modèle de Documents Structurés de Bureau (DSB) et une algèbre associée pour réaliser la spécification précise des aspects fonctionnels : opérateurs de construction et restructuration des objets manipulés et fonctions d'accès. Le stockage et l'accès sont implémentés au niveau fonctionnel sous forme d'opérations sur des documents en prenant en considération leurs structures logiques. Le couplage du modèle standard ODA au modèle DSB et l'intégration au niveau fonctionnel des opérations implémentées ont permis la mise en place d'un gestionnaire autonome de documents utilisable à partir d'un SGBD relationnel. Ce gestionnaire de documents permet la spécialisation des documents et l'utilisation de valeurs nulles. Une grande partie de ce travail a été réalisée dans le cadre du projet ESPRIT DOEOIS et un prototype expérimental a été développé sur ORACLE.

MOTS CLES

Bases de Données, Modèles de Données, Méthodes de Stockage et d'Accès, Objets Complexes, Documents Structurés, Structure Logique, Spécialisation, Valeurs Nulles, Documents Textuels.

Je tiens à remercier

Monsieur Yves CHIARAMELLA, Professeur à l'Université Joseph Fourier - Grenoble I et Directeur du Laboratoire de Génie Informatique, qui me fait l'honneur de présider ce jury.

Monsieur Jacques KOULOUMDJIAN , Professeur à l'INSA - Lyon, pour tout l'intérêt qu'il a bien voulu porter à ce travail qu'il a accepté de juger.

Monsieur Claude CHRISMENT, Professeur à l'Université Paul Sabatier - Toulouse, qui par ses critiques et ses propositions constructives m'a permis d'améliorer mon manuscrit.

Monsieur Mauricio LOPEZ, Ingénieur de Recherche au Centre de Recherche Bull-IMAG et responsable du projet ESPRIT DOEOIS dans le cadre duquel ce travail a débuté. Il est à l'origine de cette thèse. Je tiens à lui exprimer ma profonde gratitude pour ses idées et ses conseils permanents qui représentent sans nul doute un facteur décisif dans l'aboutissement de ce travail.

Monsieur Michel ADIBA, Professeur à l'Université Joseph Fourier pour m'avoir accueilli au sein de son équipe de recherche. Sa grande expérience, ses nombreux conseils et encouragements m'ont permis de mener à bien ce travail.

Monsieur Jean-Pierre GIRAUDIN, Maître de Conférence à l'Université des Sciences Sociales - Grenoble II, pour la confiance et le soutien qu'il m'a toujours témoigné, ainsi que pour ses remarques et son aide technique.

Monsieur Vincent QUINT, Directeur de Recherche INRIA, pour avoir accepté de juger ce travail et participer au jury, après avoir lu en

détail la première version de ma thèse.

L'ensemble des participants du projet DOEOIS : Henry GALY, Christian LENNE, Francisca ANTUNES, Jean-Pierre MARTIN, Paul BERARD et Claudia JIMENEZ, pour leur collaboration continue qu'ils m'ont témoignée. Un grand merci en particulier à Henri pour sa totale disponibilité, les nombreuses discussions et son amitié, qui ont fait de notre séjour dans le même bureau une expérience enrichissante. Nous avons travaillé ensemble pour la mise en oeuvre des prototypes implémentés.

Je remercie Monique CHABRE pour ses conseils et pour les nombreuses corrections qu'elle a faites de ce manuscrit.

Je remercie également mes collègues du LGI et plus particulièrement de l'équipe Bases de Données : Christine Baumann pour son amitié, Hervé pour sa bonne humeur, Bruno pour son appui moral, Marie-France pour ses encouragements ...

Enfin, je remercie Jacques-Henri CABOS pour les heures qu'il a consacrées à lire ce travail.

TABLE DES MATIERES

INTRODUCTION 9

CHAPITRE 1 21

OBJETS COMPLEXES : ETAT DE L'ART 23
1.1 Bases de données et Objets Complexes 24
1.2 Concepts de base 29
1.3 Modélisation d'objets complexes documents 33
1.4 Caractérisation des modèles 42
1.5 Classement des modèles 45
1.6 Systèmes de Gestion de Bases de Données Documents (SGBDD) 47

CHAPITRE 2 51

MODELE ET ALGEBRE POUR LES DOCUMENTS STRUCTURES DE BUREAU 53
2.1 Introduction 53
2.2 Concepts fondamentaux 55
2.3 Les opérateurs primitifs 57
2.4 Objets de l'algèbre 59
2.5 Différences entre l'algèbre formelle et l'algèbre au niveau utilisateur 60
2.6 Caractéristiques 62
2.7 Modèle descriptif associé à l'algèbre 66
2.8 Opérateurs applicables aux documents 68
2.9 Traitement des valeurs nulles 79
2.10 Association de la sémantique aux valeurs nulles 81
2.11 Représentation machine adéquate 83
2.12 Extension de l'algèbre associée au modèle 84
2.13 Conclusions 85

CHAPITRE 3 87

STRUCTURES LOGIQUES DES DOCUMENTS 89
3.1 Introduction 89
3.2 Choix des unités documentaires 92
3.3 Mécanismes de typage des documents 96
3.4 Contraintes sur la structure générique des documents 97
3.5 Conclusions 117

CHAPITRE 4 119

STOCKAGE ET ACCES AUX OBJETS COMPLEXES DOCUMENTS 121
4.1 Introduction 121
4.2 Caractéristiques des traitements des documents 123
4.3 Eléments clefs de la conception physique d'une base de documents 132
4.4 Méthodes d'accès aux documents structurés 134
4.5 Méthodes de stockage des documents structurés 137
4.6 Conclusions 156

CHAPITRE 5 159

REALISATION 161
5.1 Introduction 161
5.2 Serveur d'Information Bureautique 163
5.3 Gestionnaire de Documents Autonome (GDA) 166
5.4 Services offerts 169
5.5 Le gestionnaire relationnel GESDOC 174
5.6 Conclusions 175

CONCLUSIONS 177

BIBLIOGRAPHIE 187

ANNEXE 1 : Couplage des modèles ODA et FM 209

ANNEXE 2 : Relations de stockage 219

ANNEXE 3 : Interface fonctionnelle du gestionnaire de documents autonome 229

à ma femme
à mes deux fils

INTRODUCTION

Cadre général

Les nouvelles applications informatisées, comme par exemple la CAO, la Bureautique et la Cartographie, manipulent des objets composés. Ces objets agrègent des données hétérogènes (texte, image, graphique, son, etc.) selon plusieurs niveaux dans des structures assez complexes. La conception et la réalisation de systèmes pour gérer de tels objets exigent des outils de plus en plus puissants en raison de l'hétérogénéité des objets et de leurs structures très variées [Vel86, Col87].

La "gestion" des objets composés dans un SGBD est le cadre général de cette thèse. La sémantique associée au mot "gestion" est floue, c'est pourquoi nous précisons qu'il s'agit de la modélisation, de la manipulation, du stockage et de l'accès à ces objets. Ces tâches sont fortement contraintes par les deux aspects fondamentaux des objets composés : structuration et hétérogénéité.

En ce qui concerne l'aspect structuration nous utilisons deux notions de structures : structure générique et structure spécifique. Ces notions sont empruntées au modèle ODA [Hor85, ECM85]. La structure spécifique est créée en fonction des règles de production contenues dans les objets de la structure générique. Soit par exemple deux livres L1 et L2. Le livre L1 comprend un titre, une introduction, trois chapitres et une conclusion. Le livre L2 présente aussi un titre, une introduction mais il a quatre chapitres suivis d'une introduction. Ces livres ont des structures logiques spécifiques différentes mais elles sont bâties selon le même modèle. Ce modèle identifie la structure générique qui précise des classes de documents.

Des techniques de structuration dynamique ont été proposées pour résoudre le problème du manque de connaissance a priori de la structure des objets composés [BCP88, ACP88, Sed88]. Il est évident que le succès de telles techniques de structuration a posteriori dépend des méthodes de stockage et d'accès disponibles pour les objets volumineux.

La prise de conscience de la dépendance entre structuration des objets et méthode de stockage et accès a contribué au choix de l'approche ascendante par divers auteurs. Cette approche consiste à définir en premier lieu des méthodes de stockage et d'accès viables et efficaces pour se préoccuper ensuite de la définition de la structure et de la manipulation des objets (Modèle et Langage).

Une autre approche adoptée pour résoudre les problèmes inhérents au traitement des objets composés est l'approche descendante. L'approche descendante définit d'abord le modèle et le langage pour décrire ensuite les méthodes de stockage et d'accès. Cette thèse utilise plutôt l'approche descendante puisqu'elle présente d'abord un modèle

et une algèbre associée et ensuite les méthodes de stockage et d'accès correspondantes. Il faut préciser qu'une approche mixte est aussi possible : c'est-à-dire que l'approche descendante n'exclut pas l'approche ascendante ou vice-versa.

Les aspects de la gestion d'objets concernés par l'hétérogénéité multimédia et la grande taille des données ne seront abordés que très succintement.

Contexte d'application et objectifs de la thèse

Le contexte d'application de notre étude est la "gestion" des documents structurés dans les Systèmes de Gestion de Bases de Données Documents (SGBDD). Nous nous attachons plus particulièrement à la prise en compte de la structure logique des documents dans le but de fournir un cadre conceptuel et operationel cohérent et aussi complet que possible. La contribution de ce travail consiste en la définition d'un environnement intégré de gestion de documents structurés dont nous distinguons quatre aspects principaux :

1. La définition d'un modèle et d'une algèbre associée visant la spécification des aspects fonctionnels : opérateurs de construction et restructuration des objets modélisés et fonctions d'accès.

2. La définition d'un mécanisme de typage de documents pour fournir une base formelle pour le contrôle de cohérence (cohérence structurelle) par le SGBDD et pour le classement des documents dans celui-ci.

3. La spécification de méthodes de stockage et d'accès aux documents permettant de prendre en compte leurs structures logiques génériques.

4. La prise en compte de standards d'échange de documents entre le SGBDD et les stations de travail ou vice-versa : une approche pour leur intégration avec les trois aspects précédents est proposée.

Modèle et algèbre

La définition d'un modèle approprié aux documents structurés doit permettre la modélisation de ces documents avec leurs structures logiques génériques.

Les projets d'extensions de modèles déjà existants et des solutions originales ont été proposés : l'abandon de la première forme normale imposée par le modèle relationnel, la proposition de modèles offrant une gamme de structures plus riches, des modèles permettant la conception de SGBDs où les utilisateurs peuvent définir leurs propres types de données et leurs opérateurs.

Notre apport dans la spécification d'un modèle est la définition d'un modèle simple composé de seulement deux types de constructeurs : constructeurs d'objets simples et d'objets composés. De plus, l'intégration de valeurs nulles à ce modèle permet d'indiquer, d'une manière souple, des caractéristiques importantes pour la modélisation des documents telles que obligatoriété ou optionnalité des éléments, choix de type, etc. Une contribution dans notre travail est la définition de la valeur nulle (**nonseqniv**) qui donne de la souplesse lors de la modèlisation des documents. En ce qui concerne l'algèbre proposée nous nous sommes intéressés plutôt aux aspects

fonctionnels pour construire et modifier les structures des objets modélisés ainsi que le stockage et l'accès à ces objets. Ces aspects fonctionnels peuvent faciliter la déduction d'un langage intégré au modèle. On s'abstrait des aspects langage et formalisation de l'algèbre dans notre travail et pour expliciter quelques commandes d'accès et de définition de données nous utilisons une notation parenthesés associée à un langage proche du SQL.

Les langages proposés par ces systèmes appartiennent normalement à un des trois types suivants : langage calcul fondé sur la logique de premier ordre, langage algébrique basé sur l'algèbre et langage logique basé sur la programmation logique [AG87]. Par exemple, à l'égard du langage algébrique, on peut trouver de nombreux travaux qui semblent vouloir maintenir la vue algébrique des données du modèle relationnel classique: algèbre NST [GZC87], algèbre NF2 [JS82, Jae85, SS86], algèbre V-relationnelle [AB84], etc.

Mécanisme de typage

Le mécanisme de typage est le moyen de contrôler la cohérence structurelle des documents dans un SGBDD. Nous identifions trois environnements distincts pour la gestion des documents structurés : ouvert, partiellement ouvert et fermé. Dans l'environnement ouvert nous voulons la liberté et par conséquent le contrôle de la cohérence structurelle des documents n'est pas nécessaire. On peut quand même spécifier quelques éléments optionnels ayant en vue l'indexation mais pas comme une contrainte d'acceptation du document. Dans un environnement partiellement ouvert nous avons la possibilité d'assurer une certaine standardisation des documents stockés dans le SGBDD. Ici, nous pouvons déclarer l'existence obligatoire de quelques éléments de la structure logique ce qui constitue, en effet, une

contrainte d'acceptation du document. Enfin, dans l'environnement fermé, nous ne manipulons que des documents dont la structure est complètement connue.

Méthodes de stockage et d'accès

Le développement d'une méthode de stockage et d'accès pour les objets complexes multimédia est un sérieux pari pour les Systèmes de Bases de Données (SGBD) actuels.

Le stockage d'un objet complexe est normalement réalisé dans un espace de mémoire secondaire en s'appuyant sur des méthodes de stockage et d'accès qui soit sont propres au SGBD lui-même, soit sont fournies par le système d'exploitation sous lequel fonctionne le SGBD. Les méthodes classiques offertes par la plupart des SGBD existants sont les méthodes d'accès par index, hachage et séquentielle. En ce qui concerne les méthodes de stockage de ces mêmes SGBD on pourrait citer les méthodes Vsam, hash et séquentielle [DA82]. Un autre exemple de méthode de stockage et d'accès pour un SGBD est le schéma d'indexation étendue de Lynch et Stonebraker en [LS88].

La possibilité de définir de nouveaux mécanismes de stockage et d'accès dans un SGBD est une extension importante envisagée par des projets tels que POSTGRES [SR86], EXODUS [CD86], GENESIS [Bat88], GEODE [PTS88], etc.

Nous pouvons remarquer qu'il existe deux types de données à stocker dans un SGBD : (a) objets simples et (b) objets complexes. Un objet simple est un type conventionnel de base tel que INTEGER, REAL, STRING, etc. On constate que les objets simples correspondent dans la littérature aux objets atomiques d'Abiteboul et Grumbach [AG87] ou aux objets formatés de Faloutsos et

Christodoulakis en [FC85].

Un objet complexe est caractérisé par sa grande taille potentielle et par le fait que sa structure n'est pas totalement connue à l'avance. Dès lors, la possibilité de définir et de modifier des structures ou d'associer de nouvelles structures à un objet complexe est extrêmement importante parce qu'elle permettra d'adapter la structure d'objet aux besoins de l'application. La difficulté de spécification a priori la structure des objets a été étudiée en [BCP88] et [Sed87] pour le cas spécifique des documents.

Les méthodes d'accès aux objets simples sont relativement bien connues et déjà utilisées par les SGBDs commercialisés. Les méthodes principales sont divisées en trois sous-classes: (a) arbre (b) hachage et (c) "signature" [FC85]. Dans la sous-classe (a) nous pouvons citer les méthodes basées sur les arbres (B-trees, B*-trees, Prefix B*-trees [BM72, Knu73, Com79, TF82]) telles que fichiers inversés et index. Dans la sous-classe (b), des méthodes basées sur le hachage sont proposées dans la littérature en [Knu73, SD76, Riv76, Mar79 ("spiral hashing"), LR82 ("multiattribute hashing"), Lit80]. Dans la sous-classe (d), des fichiers signature sont créés par une fonction appliquée à chaque enregistrement et stockés séquentiellement. Des exemples de méthodes basées sur les signatures sont décrits en [Val76, Rob79, PBC80, Chr83].

Il n'existe pratiquement pas de propositions claires concernant des méthodes d'accès aux objets composés que nous appelons aussi d'objets complexes. En effet, celles qui existent sont des méthodes d'accès appliquées aux cas particuliers des objets complexes tels que les formulaires [Tsi80, Col87], les documents de bureau [OIS86a], les documents techniques [BCP88, Sed88] et les documents multimédia

[DG88, Rab85, FC85]. On constate dans les méthodes d'accès actuelles un problème de granularité : soit qu'elles permettent l'accès à l'objet dans sa totalité et rendent difficile l'accès à une partie de son contenu [HL82], soit qu'elles permettent l'accès aux parties de son contenu et ne facilitent pas l'accès à l'objet dans sa totalité. Ainsi, en éclatant les documents pour les stocker, le Projet TIGRE [LV84, Vel86] a rendu complexe leur manipulation globale.

Nous nous apercevons de la difficulté du développement d'une méthode de stockage et d'accès capable de résoudre tous les problèmes liés aux objets complexes. Notre approche consiste donc à proposer des mécanismes de stockage et d'accès permettant à l'utilisateur de composer sa propre méthode de stockage et d'accès. Ceci équivaut à notre principale contribution pour le stockage et l'accès aux documents. Notre proposition de stockage et d'accès est compatible avec la méthode de la signature parce qu'on veut profiter de tout le travail fait lors de la mise en oeuvre du Serveur d'Information pour la Bureautique (SIB) réalisée dans le cadre du projet ESPRIT DOEOIS 231. Néanmoins, un document peut être stocké et accédé indépendamment de la méthode de la signature.

Echange de documents

Les documents sont transmis avec leurs structures logiques d'origine : générique et spécifique conforme le standard ODA/ODIF. Notre approche privilégie la reconnaissance de cette structure pour faciliter la réalisation de tous les objectifs décrits antérieurement. Le transport des documents avec leurs structures logiques est le principal objectif des standards actuels d'échange de documents. Sans la condition de transporter le document avec ses structures logiques et la connaissance du standard utilisé (pour permettre la mise en oeuvre

d'un interpréteur de structures) par le SGBDD notre approche ne serait pas possible.

En profitant de l'expérience de la mise en oeuvre du SIB nous avons implémenté un autre gestionnaire de documents (Gestionnaire de Documents Autonome - GDA) accesible d'un SGBD relationnel où nous avons validé la plupart de nos idées. Le GDA mis en oeuvre n'accepte actuellement que des documents dans le format ODA/ODIF mais il est facilement extensible aux autres standards telles que SGML, GRIF, WORD, etc.

Plan de la thèse

Cette thèse est organisée en 4 chapitres précédés de la présente introduction et suivis d'une conclusion et d'une liste d'annexes. Le chapitre 1 fait un état de l'art des objets complexes abordant principalement les aspects modélisation et manipulation de ces objets. En ce qui concerne la modélisation, il a été proposé diverses approches de solution en abandonnant le concept de forme normale du modèle relationnel [Cod70 et Dat81]. Les modèles tels que le modèle "Nested Sequence Tuple" (NST) [GZC87], le modèle NF2 [SP82, JS82, Jae85, SS86], le modèle NF2 généralisé [PA86] et V-relationnel sont des exemples de ces approches. Il faut remarquer que les documents structurés est un cas particulier des objets complexes et par conséquent on constate que la plupart des travaux appliqués à la modélisation des objets complexes sont aussi appliqués aux documents structurés. Un classement des SGBDD (Système de Gestion de Base de Données Documentaires) est présenté à la fin du chapitre 1.

Le chapitre 2 présente un modèle et une algèbre associée pour les documents de bureau. Ceci nous permet d'obtenir un SGBDD intégré.

Dans le chapitre 3, nous présentons les mécanismes de typage qui permettent d'adapter le système de base documentaire à l'un des environnements suivants : ouvert, partiellement ouvert et fermé. En principe, pour une question de liberté, la base documentaire possède toujours un environnement ouvert permettant de stocker en liberté tous les documents indépendemment de leurs structures. Il est important de remarquer que la stratégie d'acceptation des documents par le SGBDD est directement liée à ces mécanismes. Nous décrivons également un ensemble de contraintes d'intégrité qui permet à la fois de spécifier la structure logique des documents et de donner des informations importantes non seulement pour le stockage mais aussi pour l'indexation des éléments logiques. Dans ce chapitre, le problème de la vérification de régles contradictoires, comme par exemple la déclaration de l'élément chapitre à la fois obligatoire et facultatif, nous ne traitons que superficiellement.

Le chapitre 4 décrit tous les aspects de stockage et d'accès aux objets complexes documents. Une démarche pour la conception physique est proposée en prenant en considération les caractéristiques de manipulation et de comportement de la population de données documents. Nous décrivons également des techniques de flexibilité de stockage et d'accès et nous présentons, en détail, plusieurs options de schémas physiques et leurs méthodes de stockage et d'accès associées.

Le chapitre 5 fait le bilan de la réalisation effectuée d'abord dans le cadre du projet ESPRIT DOEOIS (SIB) et ensuite dans le cadre de la conception d'un Gestionnaire de Documents Autonome (GDA)

comme extention contrôlée du SGBD ORACLE. En annexes, nous fournissons la description des relations de stockage de base communes au SIB et au GDA, ainsi que l'interface fonctionnelle de ce dernier.

Les conclusions de cette thèse dressent enfin plusieurs constats : les apports proposés, ce qui est réalisé, ce qui reste à réaliser, ce qui devrait être réalisé ...

CHAPITRE 1

OBJETS COMPLEXES : ETAT DE L'ART

1

OBJETS COMPLEXES : ETAT DE L'ART

Ce chapitre présente les principaux besoins en matière de **gestion** des objets complexes. Par "gestion", nous entendons la modélisation (niveau formel), la manipulation (niveau logique) et le stockage et l'accès (niveau physique) aux objets complexes.

Notre intention, dans ce chapitre, n'est pas de décrire spécifiquement tel ou tel modèle, mais plutôt de présenter une étude synthétique des problèmes liés à la gestion des objets complexes, et plus spécifiquement liés au cas particulier des documents structurés multimédia qui peuvent être considérés comme des objets complexes.

1.1 Bases de données et objets complexes

La recherche en bases de données pour les applications non conventionnelles, telles que la conception assistée par ordinateur (CAO), la bureautique, les environnements de développement de logiciel, la cartographie, la gestion intégrée de dossiers médicaux, a mis en évidence l'importance de la gestion d'objets présentant des structures complexes. Ces objets, que nous appelons ici objets composés ou complexes. Ils peuvent aussi être composés d'autres objets (composés ou atomiques) de type multimédia. Les types multimédia permettent de prendre en compte des images, des textes, des graphiques, des sons, etc.

La gestion d'objet complexe multimédia est à l'origine de termes tels que bases de données généralisées, bases de données multimédia, bases de données orientées objets, SGBD extensibles, objets structurés, modèles non en première forme normale (N1NF), formulaires et documents multimédia.

Nous pouvons donc en déduire que la problématique liée aux objets complexes est très vaste. On peut en effet affirmer que la compréhension de ce problème est difficile d'une part parce que les exigences des applications sont différentes et quelquefois contradictoires par rapport aux applications conventionnelles et d'autre part en raison de l'absence d'une vision générale sur les plans pratiques et théoriques dans les travaux publiés sur ce domaine.

Pour mieux étudier ce problème, nous le découpons de deux manières distinctes : découpage horizontal et découpage vertical (cf. figure 1.1). Le découpage horizontal prend en compte les niveaux formel, logique et physique au regard des aspects suivants : modélisation, manipulation, stockage et accès. Le découpage vertical

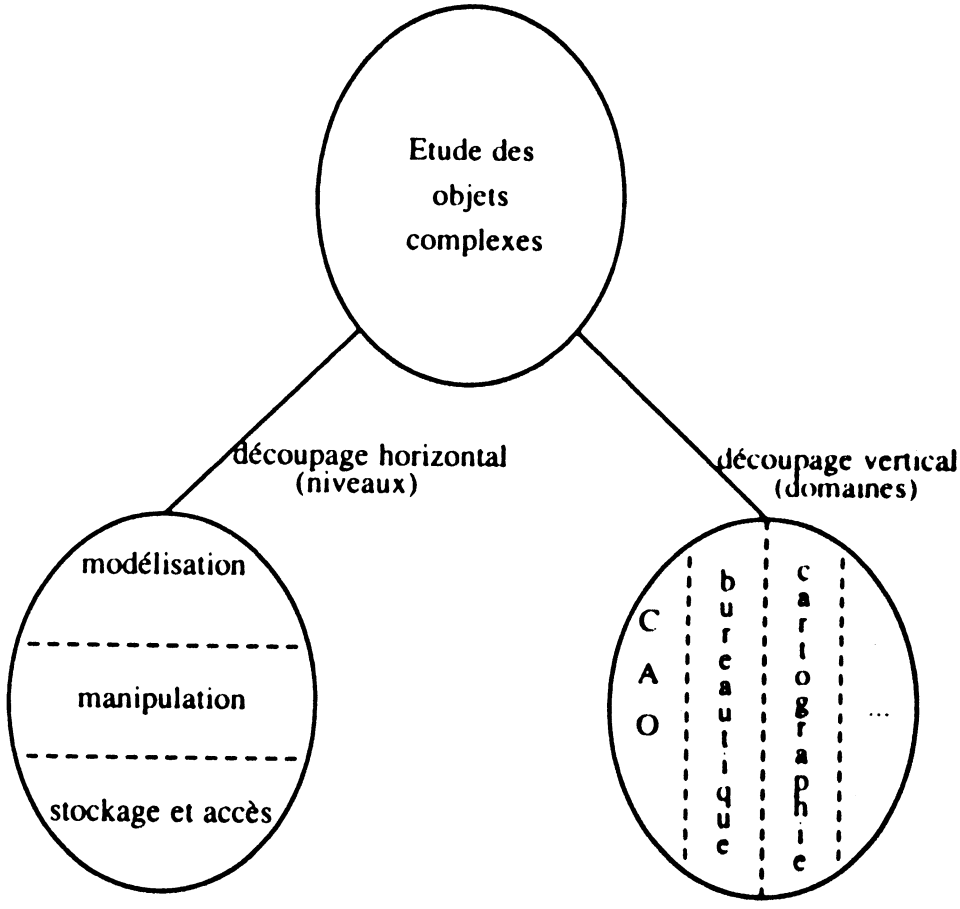

Fig. 1.1: Comment étudier les objets complexes ?

prend en compte les domaines d'application des objets complexes : la CAO (circuits intégrés), la bureautique (documents structurés), la documentation technique (documents structurés volumineux), la cartographie (composants géographiques, composants géométriques), etc. En ce qui nous concerne nous nous limitons à la bureautique et aux documents structurés.

Le premier aspect indiqué dans le découpage horizontal concerne la modélisation des objets complexes. Tous les travaux liés aux formalismes et concepts tels que les modèles, les langages et les algèbres concernent cet aspect. Ici, nous pouvons citer comme

exemples les modèles N1NF abandonnant la condition de première forme normale imposée par le modèle relationnel tels que le modèle "Nested Sequence Tuple" (NST) [GCZ87, Güt88], le modèle NF^2 [SP82, JS82, Jae85, SS86], le modèle généralisé [PA86], le modèle V-relationnel [BRS82, AB84], les modèles Orientés Objets tels que le modèle O2 [LRV87], le modèle IRIS [FBC87], les modèles sémantiques, les modèles fonctionnels tels que DAPLEX [Shi81] et FM [OIS86c, OIS87c].

La dernière section de ce chapitre dresse un classement de ces modèles en fonction de leurs constructeurs et des restrictions d'utilisation de ceux-ci.

Le deuxième aspect indiqué dans ce même découpage horizontal concerne la manipulation des objets complexes.

En effet, la manipulation des objets englobe les actions suivantes :

- **La création des objets**. La création d'un objet entraîne la création des valeurs associées à la structure logique spécifique de cet objet. Par exemple, la création d'un document englobe la création des valeurs associées à la structure logique spécifique. L'éditeur multimédia utilisé par le projet OIS s'intéresse à la construction de documents suivant leurs structures logiques (générique et spécifique).

- **La restitution des objets ou des parties des objets**. Tout gestionnaire d'objets complexes doit être capable de restituer les objets qu'il gère. Il est quelquefois nécessaire de restituer des parties d'objets : ceci pose le problème de la granularité. Certains gestionnaires permettent la restitution de l'objet dans sa totalité mais rendent difficile la restitution d'une partie [HL82], d'autres

présentent l'inconvénient contraire [Vel86].

- **La suppression des objets**. Dans certains modèles la cohérence lors de l'exécution d'une opération de suppression est difficile à gérer. (exemple : les modèles à contraintes existentielles, comme Orion [BKK87]). Cette difficulté est liée aux dépendances existentielles entre les objets, qui peuvent entraîner une cascade de suppressions d'objets et éventuellement l'échec de l'opération. La suppression de documents dans le Gestionnaire de documents qui nous avons mis en oeuvre ne pose pas de problème d'abord parce que nous n'avons pas des contraintes existentielles et ensuite parce que le partage des parties de documents ni la composition de documents d'autres documents ne sont pas permis.

- **La modification des objets**. La modification des objets ainsi que la restructuration de ces objets peuvent, selon l'approche adoptée par le système, être plus ou moins compliquées. Par exemple, la modification des documents multimédia dans le Serveur OIS qui nous avons mis en oeuvre dans le cadre du projet DOEOIS n'est pas permise pour éviter le recodage des documents ODA/ODIF. L'approche du serveur OIS consiste à privilégier l'échange des documents sous leur format d'origine justifiant ainsi l'imposition de cette contrainte. Par contre, la restructuration d'un document sans la modification de son contenu original est possible en modifiant la partie du schéma de la base inhérante au document stocké.

- **La transmission ou réception des objets**. En ce qui concerne l'ouverture du système il faut permettre l'échange des objets entre gestionnaires d'objets différents.

Les documents jouent un rôle fondamental dans l'échange d'informations. Pour cette raison, les comités internationaux de standardisation ont déjà proposé beaucoup de standards susceptibles de permettre l'échange de documents. On peut citer par exemple les standards suivants : CCITT T.61 (Basic Teletex), CCITT X.420 (Simple Formattable Document), CCITT T.6 (Photographic Information-Facsimile Group 4), CCITT T.73 (Restricted Layout Structure - T.61 et T.6) et ECMA 101 (ODA-Office Document Architecture). Le document du groupe TC29 de l'ECMA (European Computers Manufacturers Association) [ECM85] définit le standard ODA/ODIF qui permet l'échange des documents en tant qu'unités contenant leurs structures logiques et/ou formatées.

Enfin, la dernière couche indiquée dans le découpage horizontal nous permet d'étudier, au niveau physique, l'aspect stockage et accès. Cet aspect est fondamental pour permettre la manipulation des données avec un degré de performance acceptable. La définition des structures de stockage influe directement sur la méthode d'accès adoptée par le système ainsi que sur son degré de performance. Cette définition est le résultat d'une analyse du comportement des données qui peut justifier du choix de l'organisation, des chemins d'accès (primaires et secondaires) et des outils de flexibilité de stockage et d'accès à offrir.

Le problème du stockage et de l'accès aux objets complexes documents est traité en détail dans le chapitre 4.

La section suivante rappelle des concepts et notions inhérents aux différents domaines, tels que systèmes d'informations, système de gestion de bases de données (SGBD) et modélisation d'objets

complexes que nous estimons intéressants pour la représentation des documents structurés multimédia. Il faut remarquer que le fait de redéfinir le concept d'information vise simplement à la compréhension d'autres concepts de base tels que schéma, type, classe et occurrences. Ensuite, nous commentons plusieurs concepts inhérants à la modélisation même des documents structurés multimédia. Nous dressons aussi un classement des modèles en fonction de leurs constructeurs et de l'application de ces constructeurs, de la présence du concept de hiérarchie et la notion d'ordre. Finalement, nous présentons un classement des Système de Gestion de Bases de Données Documents (SGBDD) en fonction de l'intégration ou du couplage des aspects modèle, langage et architecture employés par le système.

1.2 Concepts de base

Dans le domaine de la modélisation des objets, la diversité des concepts et notions est souvent source d'incompréhension. Ainsi à titre d'exemple, dans le travail "Express-MLD : Une approche de développement rapide basé sur le modèle logique de données" [PRL88], les mots "informations" et "données" sont utilisés comme synonymes alors que dans la réalité ils recouvrent deux notions distinctes. Il est vrai que la définition du concept d'information est difficile à donner. D'après H. von Foerster, cité par Collet [Col87], "L'information est le plus vicieux des caméléons conceptuels".

Nous pensons utile ici de reprendre le concept d'information. En premier lieu, parce que les notions utilisées pour cette définition peuvent faciliter la compréhension de concepts usuels dans les SGBD actuels tels que schéma et type, et ensuite, parce que l'évolution des SGBD se traduit par l'introduction de ces notions.

Nous nous sommes inspirés des définitions données par Kent [Ken84] et Habrias [Hab88].

1.2.1 Notion d'information

Le concept d'**information** est à l'origine de notions telles que **valeur, contexte** et **représentation**. Nous pouvons définir l'**information** par la formule suivante :

Information = valeur + contexte + représentation

La notion de **représentation** englobe deux sous-notions : la **forme** et le **système de représentation**.

Pour expliquer cette formule, supposons l'existence de la **valeur** [11] qui peut signifier, selon le **contexte**, le nombre de chaises dans une maison, l'âge d'un enfant, etc. Par exemple, les deux informations ci-dessous avec leurs contextes soulignés :

a) Il y a [11] chaises dans la maison de Diogo
b) Diogo a [11] ans

Ces **informations** sont encore incomplètes par l'absence de l'indication de la **forme** et du **système** de **représentation** de la **valeur**. Il est vrai que la **forme de représentation** implicite de notre numération est réalisée, en général, grâce à des chiffres arabes, en utilisant le système de numération décimale comme **système de représentation**. Mais, dans le cas de Diogo, que je connais bien puisque c'est mon fils, le **système de représentation** de l'information b) est le système binaire de numération. Pour moi, c'est évident : Diogo a vraiment [trois] ans.

Il faut noter que cette dernière valeur [trois] utilise comme **forme de représentation** une série de lettres et comme **système de représentation** la langue française. La standardisation de la **forme de représentation**, comme par exemple le fait de toujours utiliser des lettres, est une bonne manière d'abstraire la notion de représentation. Le principal avantage de cette approche est de rendre la recherche par le contenu indépendante de la vraie représentation. Par exemple, la requête << Quelles sont les phrases qui nous informent de l'âge de Diogo ? >> peut donner comme résultat :

Diogo a [3] ans.
Diogo a [11] ans.
Diogo a [trois] ans.
Diogo a [III] ans.

La notion de **représentation** est intimement liée aux problèmes de stockage, d'accès et de présentation d'informations. Le système de numération binaire et les codages tels que EBCDIC ou ASCII ont rendu possible le stockage des données dans un ordinateur.

Nous allons maintenant reprendre quelques concepts usuels dans les Systèmes de Gestion de Bases de Données.

1.2.2 Schéma

Si nous utilisons les notions précédentes on peut dire que le **schéma** est défini par la formule suivante:

schéma = contexte + représentation

En effet, un schéma décrit les structures (contexte) et les valeurs

possibles (représentation) de tous les objets d'une **classe**. Au fur et à mesure qu'on insère dans le schéma la sémantique de l'application on obtient des SGBDs plus riches sémantiquement.

En ce qui concerne la description du contexte, qui varie d'un modèle à l'autre, nous identifions plusieurs méthodes : par des constructeurs tels que *choix*, *agrégat*, *liste*, etc [Col87, Vel84], par des notions de généralisation/spécialisation, agrégation (modèles sémantiques), par des notions de fait, fonction, entité (modèles fonctionnels) [Sac86, Gib85], etc.

1.2.3 Type

Le concept de type permet d'associer la notion de représentation à l'objet. Par exemple, l'indication du type "INTEGER" pour un objet donné détermine sa forme de représentation (chiffres) et son système de représentation (binaire, décimal, octal, etc). Le concept de type est plutôt lié aux objets atomiques.

1.2.4 Classe

Le grand intérêt de ce concept est de regrouper des objets ayant le même type. Ainsi, une seule spécification des objets est suffisante evitant la redondance de spécifications pour de nouveaux objets de ce type. Par exemple, la classe document du modèle couplé (FM et ODA) utilisé par le SIB mis en oeuvre englobe les spécifications générales des attributs du profil et du contenu de tous les documents de la base.

1.2.5 Occurrences

Une occurrence est une valeur associée 'a un schéma atomique ou à un schéma composé. Un schéma atomique est défini par un des

types de base tels qu'INTEGER, REAL, ALPHANUMERICAL ou TEXT. L'occurrence d'un objet atomique est une valeur associée à un schéma atomique. Soient, par exemple, les schémas atomiques a) et b) illustrés dans la figure 1.2. La valeur associée au schéma atomique ALPHANUMERICAL b), enrichi par la concaténation des objets atomiques INTEGER et TEXT, peut devenir un schéma composé si les objets plus élémentaires sont reconnaissables. Par exemple, le schéma composé (COMP) c) est possible grâce à la reconnaissance des objets atomiques nécessaires.

L'occurrence d'un objet composé est un ensemble de valeurs associées à un schéma composé. Un schéma composé est formé de schémas atomiques.

Selon le contexte inclus dans la valeur et selon le schéma associé à l'occurrence, on peut avoir une information plus ou moins complète. La figure 1.2 illustre par exemple trois occurrences qui montrent l'enrichissement progressif de l'information.

1.3 Modélisation d'objets complexes documents

Les concepts et notions varient et le même concept est quelquefois référencé par des termes techniques différents d'un modèle à l'autre : la nomenclature n'est pas standardisée. Par exemple, le Modèle Documents Multimédia (MDM) de Rabitti [Rab85] est très proche du modèle ODA ("Office Documents Architecture") dont il est inspiré. On retrouve essentiellement trois concepts fondamentaux dans le modèle MDM : la structure conceptuelle qui correspond à la structure logique générique, la structure logique qui correspond à la structure logique spécifique, et la structure formatée qui englobe les deux structures : générique et spécifique formatées.

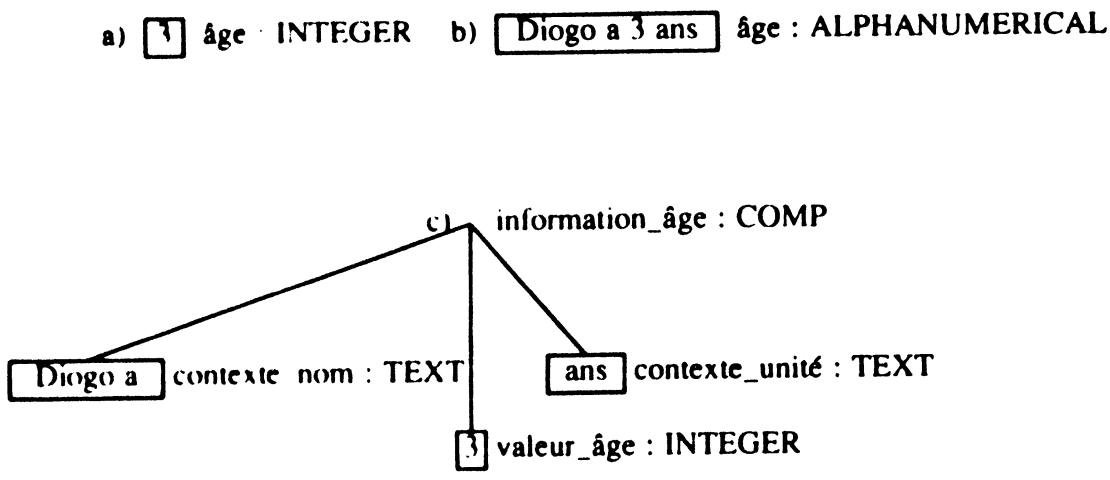

Fig. 1.2 : Exemples d'occurrences.

Structure :

Le concept de structure conceptuelle du modèle MDM comme le concept de structure logique générique du modèle ODA correspondent au concept de schéma d'une classe de documents. La construction d'un schéma est réalisée par le biais des constructeurs offerts par le modèle. Ces constructeurs sont basés sur des notions telles que lien, direction, ordre, niveau, identificateur unique, valeurs nulles, etc. La section 1.4 de ce chapitre présentera les constructeurs de quelques modèles ainsi que les possibilités d'application de ceux-ci.

Lien :

La notion de lien existe pratiquement dans tous les modèles. Par exemple, dans le "Fact Model" (FM) la déclaration d'un fait [A,B] entraîne la création de deux liens : les fonctions réciproques A.B et

B.A (voir annexe 1). Un autre exemple plus complet est le modèle hypertexte proposé par Nardot et Pujolle [NP89] qui distingue 5 types de liens : domaine, structural, d'identité, référentiel et héritage.

Niveau :

Il y a des applications pour lesquelles le concept de niveau est extrêmement utile. Par exemple, lorsqu'on manipule des documents il est intéressant de savoir que le corps du document est composé de chapitres. L'exemple cité antérieurement en FM ne permet pas de savoir si A et B sont au même niveau, si A est dans un niveau supérieur à B, ou inversement.

Ordre :

La notion d'ordre est fondamentale pour traiter des documents [Güt88]. L'opérateur "group by" du langage SQL et l'opérateur "order by" du langage HDBL proposé par Pistor et Anderson à IBM-Heidelberg [PA86, LKD88] existent parce que la notion d'ordre est tout simplement indispensable. En ce qui concerne les documents, nous identifions deux types d'ordre : l'ordre des éléments répétitifs (de même type) dans un niveau et l'ordre entre les éléments non-répétitifs (de types differents) liés à un élément appartenant à un niveau supérieur. L'ordre des chapitres d'un livre est un exemple d'ordre d'éléments répétitifs alors que l'ordre des composants d'un livre (titre, introduction, chapitres et conclusion) est un exemple d'ordre entre objets distincts. Il est clair que cet aspect est indispensable pour répondre à des questions telles que : << Quel est le numéro d'ordre du chapitre qui traite de la structure logique des documents ? >> ou <<Quel est le nom du premier auteur de l' article intitulé "Bases de Données et Objets Structurés"?>>.

Nous pouvons citer au moins quatre fonctionnalités où les notions d'ordre et de niveaux sont indispensables :

- Construction des Documents : la construction de documents est établie de plus en plus fréquemment par le biais d'un éditeur de texte. En matière de logiciels d'édition, on observe un progrès dans le traitement des aspects structuraux et multimédia des documents. Les éditeurs dits multimédia traitent des images, des graphiques, des sons, des textes, etc. Les éditeurs dits structurés doivent traiter les documents en se guidant par des structures logiques (sorte de sommaire ou plan de document) où les concepts d'ordre et de niveaux hiérarchiques sont très importants. Le nombre de niveaux hiérarchiques nécessaires pour traiter les documents dépend de l'application et du type de document.

- Lecture séquentielle d'un document : La lecture séquentielle d'un document nécessite la mise en ordre de ses éléments logiques car la parfaite compréhension d'une partie dépend de la lecture des parties antérieures de même niveau.

- Présentation des documents : la notion de présentation recouvre à la fois les aspects mise en forme et restitution (matérialisation) d'un document. L'aspect mise en forme doit prende en compte la structure logique des documents puisque la présentation est justement faite pour mettre en évidence l'organisation du document. L'aspect restitution dépend principalement de l'appareil utilisé pour la matérialisation du document. Certains effets de présentation, tels que les changements de police ou de corps, ne peuvent pas être obtenus sur toutes les imprimantes et sur tous les écrans. Les systèmes WYSIWYG (What You See Is What You Get) qui permettent d'obtenir sur papier (support imposant une

construction séquentielle) une image (mise en page) identique à celle que le système affiche sur l'écran (mise en écran) pendant l'édition du document exigent des imprimantes et des écrans dotés des mêmes ressources. Avec la philosophie hypertexte ces aspects sont moins importantes.

- Numérotation : Dans les documents, de nombreux éléments sont numérotés : chapitres, sections, figures, notes, formules, théorèmes, définitions, références bibliographiques, exercices, exemples, parmi d'autres. Par exemple, la numération des éléments logiques de la thèse illustrée ci-dessous :

Thèse
  Introduction
  Chapitre 1
    section 1.1
    section 1.2
  Chapitre 2
    section 2.1
    section 2.2
    section 2.3
  Chapitre 3
  Conclusion

Pour exprimer un niveau de façon précise, la notation parenthésée est suffisante. Supposons, par exemple, que nous voulions designer le titre d'un chapitre et que ce chapitre se trouve au premier niveau et son titre au deuxième niveau par rapport à la racine Livre :

(Livre(chapitre(titre)))

Pour designer le titre du Livre et le titre du chapitre une option consisterait à écrire :

(Livre(titre,chapitre(titre)))

Vue :

Il semble intéressant pour un gestionnaire de documents de pouvoir visualiser les documents de différentes manières. On appele **vues** ces divers modes de visualisation. Les vues sont basées sur la structure logique du document et varient selon les traitements qu'on veut effectuer sur ce document. Des traitements tels que la construction, la mise en forme, le stockage, la matérialisation et la manipulation d'un document peuvent exiger des vues très différentes. Selon la stratégie adoptée, on peut s'abstraire presque complètement de la structure logique d'un document pendant son stockage. Par exemple, dans le projet DOEOIS, la vue offerte pendant le stockage d'un document ne donne que quelques éléments du profil du document. Certains cas, comme la recherche de tous les documents qui contiennent explicitement le mot "multimédia" dans un titre de chapitre ou de section, exigent une vue de ces titres.

Schéma :

Un document peut être décrit en termes de schémas, de classes d'occurrences et de formats. Un schéma spécifie toutes les caractéristiques possibles des éléments susceptibles d'appartenir aux documents. Une classe de documents est un groupement de documents présentant des aspects communs liés à leurs structures de données. Par exemple, des lettres, des rapports, des factures et même des livres.

Le principal objectif d'un schéma est de décrire la structure de données et les valeurs possibles de ces données pour tous les documents d'une même classe. Un schéma peut être associé à un ou plusieurs formats. Néanmoins, il faut noter que la plupart des travaux sur les documents ne traite pas les aspects mise en page et structuration de ces documents d'une manière intégrée [Güt88, GZC87].

Constructeur :

Chaque schéma spécifie une structure logique des données obtenue par l'application de constructeurs, de règles, de contrainte et de nouveaux concepts tels que valeurs nulles.

Sont fréquemment cités [Col87] quatre constructeurs : agrégat, choix, liste, et ensemble. Les symboles ci-dessous représentent ces quatre constructeurs.

x : agrégat
| : choix
***n : liste**
*** : ensemble**

Les trois premiers constructeurs sont les mêmes que ceux utilisés par le modèle de document TIGRE [Vel84]. Il faut noter que le fait d'avoir plusieurs constructeurs ne signifie pas pour autant un plus grand pouvoir d'expression ; de plus, la formalisation est compliquée en raison de ces constructeurs. Le modèle document de Güting n'a que deux constructeurs (nuplet et séquence) et son pouvoir d'expression est le même que celui des modèles cités ci-dessus.

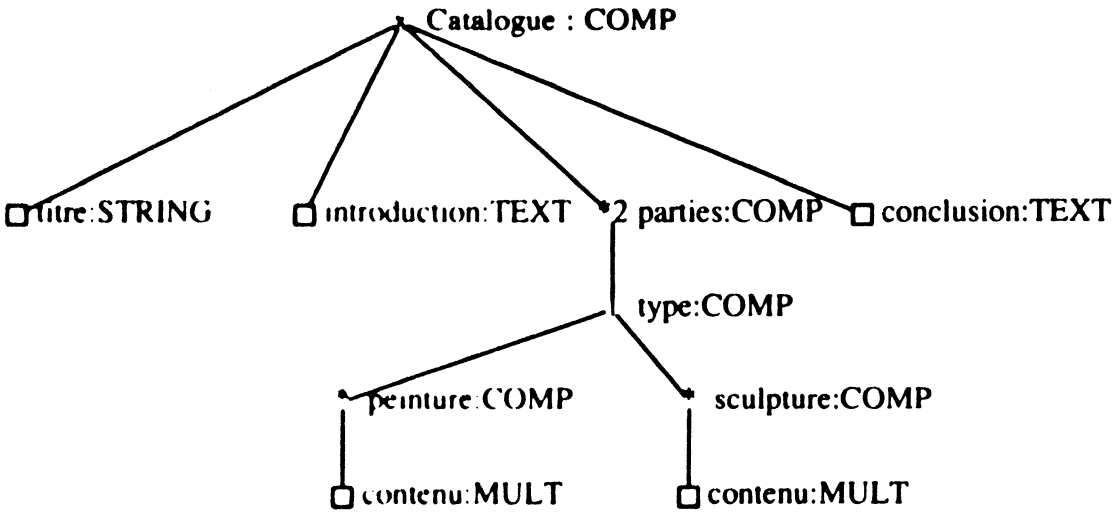

Fig 1.3 : Schéma d'un catalogue d'oeuvres d'art.

Un schéma peut être représenté en utilisant des arbres où les noeuds intermédiaires et feuilles sont étiquetés par les constructeurs [AH86]. Les figures 1.3 et 1.4 illustrent des schémas pour un Catalogue d'oeuvres d'art en utilisant une représentation arborescente. Ce Catalogue peut être composé de deux parties (I et II) et contient des oeuvres d'arts de type peinture et sculpture. La figure 1.3 utilise quatre constructeurs tandis que la figure 1.4 n'utilise que deux constructeurs. Les deux constructeurs utilisés dans la figure 1.4 sont nuplet et séquence. Les noeuds intermédiaires représentent le constructeur séquence et les noeuds finaux représentent le constructeur nuplet. Il faut noter dans les deux exemples illustrés par les figures 1.3 et 1.4 l'utilisation par défaut de la valeur nulle non_applicable (nap).

La distinction entre schémas atomiques et schémas composés est très importante. Le groupement de schémas atomiques en ensembles tels que NUM (qui regroupe INTEGER et REAL), BOOL (qui regroupe VRAI et FALSE), GRAPHIC, IMAGE, VOICE, MULT (qui regroupe comme un objet atomique différents objets multimédia) et

TEXT facilite la formalisation du schéma. Dans [ABC87] l'ensemble TIME englobe les schémas TEMPSRESTREINT, INTERVALLE et PERIODE. L'utilisation du type multimédia (MULT) est importante parce que celui-ci assure la notion d'atomicité des valeurs au niveau le plus élémentaire de la structure. De cette manière, on sépare bien l'aspect objet atomique et objet composé. Ainsi, le type COMP n'est jamais à la fois un objet composé et un objet atomique. Néanmoins, un opérateur qui transforme un objet composé (COMP) en un objet atomique (MULT) est particulièrement nécessaire pour certaines applications.

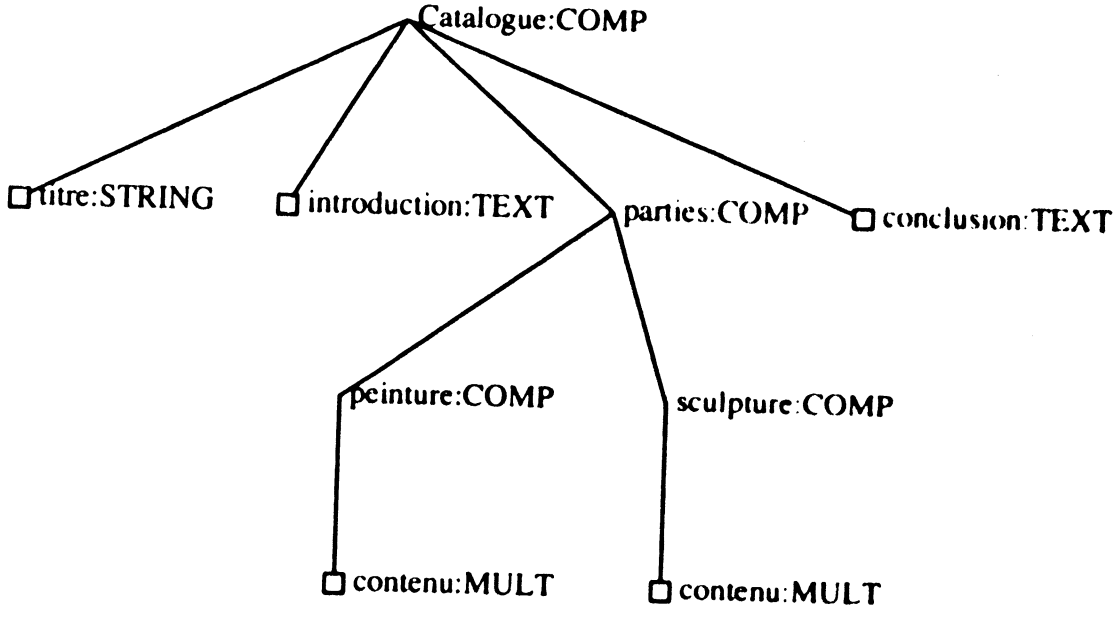

Fig. 1.4 : Schéma d'un catalogue d'oeuvres d'art.

Régles :

Le rôle principal des règles est la création d'objets. Les régles combinent des constructeurs spécifiant les règles de production des objets. En ce qui concerne les règles, on peut trouver dans les travaux d'Abiteboul, Hull et Grumbach [AH86, AG87] des règles de réécriture qui permettent de traiter les différentes options de la structure des données d'un schéma. Un autre exemple sont les règles de production d'un document spécifique contenues dans la structure générique du modèle ODA [AP89].

1.4 Caractérisation des modèles

Les modèles peuvent être caractérisés par leurs constructeurs et par les restrictions sur les liens établis entre ces constructeurs. Par exemple, la figure 1.5 permet de déduire que le constructeur d'ensembles du modèle relationnel ne peut être lié qu'au constructeur de nuplets. Le constructeur d'ensembles ne peut jamais être lié à un autre constructeur d'ensembles, ni directement à des valeurs atomiques. Dans cette même figure on vérifie que les modèles N1FN présentent une restriction un peu arbitraire d'alternance de lien entre leurs constructeurs d'ensembles et de nuplets. Cette restriction, qui ne limite pas le pouvoir d'expression du modèle, peut simplifier le formalisme. Les modèles d'objets complexes d'Abiteboul et Grumback [AG87] et NST ("Nested Sequences of Tuples") de Güting [GZC87, Güt88] n'ont pas de restrictions de liens entre leurs constructeurs.

En effet, la différence entre le modèle d'objets complexes et le modèle NST se résume à l'introduction de la notion d'ordre dans le constructeur d'ensembles qui devient ainsi le constructeur de séquences.

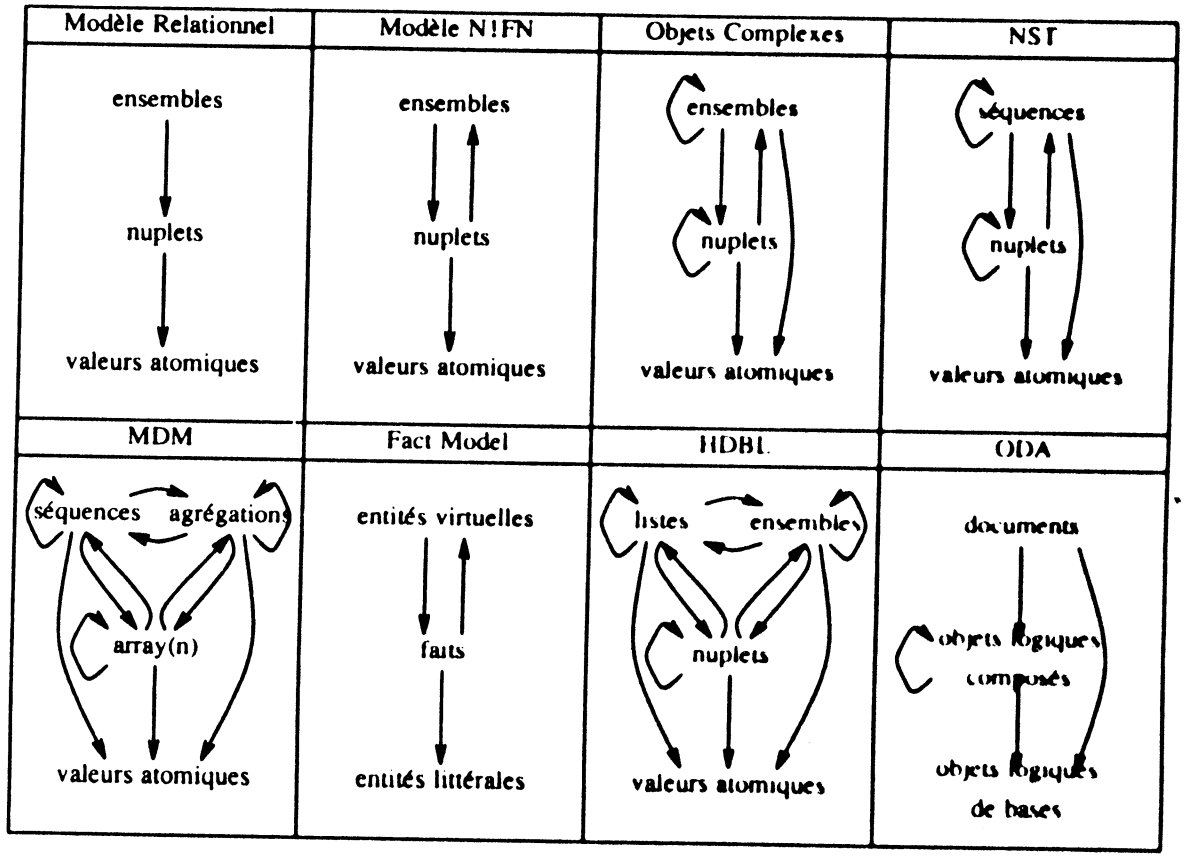

Fig. 1.5 : Possibilités d'utilisation des constructeurs

La figure 1.5 permet également de déduire les restrictions de liens entre les constructeurs des modèles de Rabitti ("Multimedia Documents Model" - MDM) [Rab85], de Sacco ("Fact Model" - FM) [OIS86c, OIS87c, Sac85], HDBL [LKD88] et ODA [Hor85, ECM85]. Il faut remarquer que les liens entre les constructeurs du modèle FM ne donnent pas la signification de composants entre eux comme c'est souvent le cas dans les autres modèles. En effet, le constructeur de faits du modèle FM est le lien propre entre les constructeurs d'entités virtuelles ou littérales. Ce lien ne porte pas la notion d'ordre, ni de hiérachie ce qui empêche l'interprétation des composants. En ce qui concerne le constructeur de documents du modèle ODA, celui-ci ne peut être lié à aucun autre constructeur de documents. Cette

restriction empêche qu'un document soit composé d'autres documents dans ce modèle. Quant au constructeur de séquences du modèle MDM, ce dernier ne permet que des composants d'un même type. Par conséquent, la modélisation de certaines situations par le modèle MDM exige un niveau de plus que si l'on utilise, par exemple, le modèle HDBL. La figure 1.6, illustre les schémas obtenus par MDM et HDBL d'un catalogue composé d'un titre et de chapitres, où chaque chapitre est à son tour, composé d'un titre et de sections.

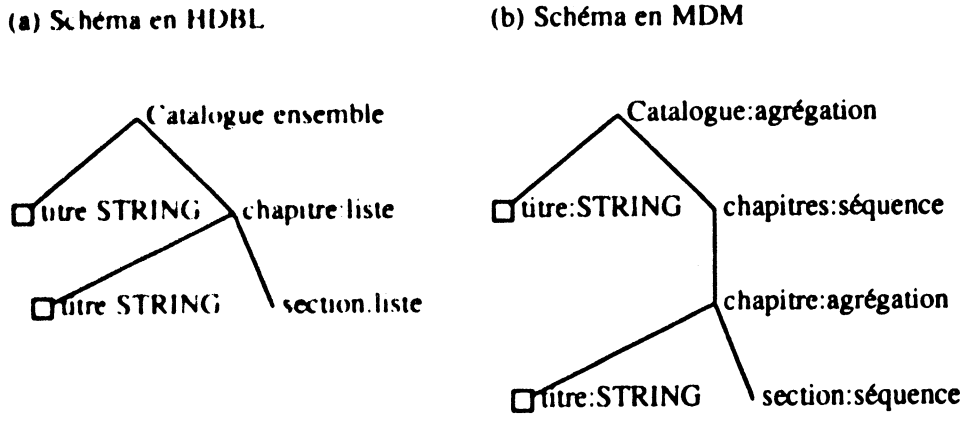

Fig 1.6 : Exemples de schémas d'un catalogue.

On peut distinguer un constructeur d'un autre par l'utilisation ou la non-utilisation de certaines notions déjà décrites antérieurement. Par exemple, le constructeur d'ensembles présent dans des systèmes tels que relationnel [Cod70, Dat81], N1FN [AG87], V-relation [BRS82], Objets Structurés [AG87], HDBL [LKD88], etc se différencie du constructeur de séquences du modèle NST de Güting [Güt86] par l'absence de la notion d'ordre.

Sur les modèles présentés dans la figure 1.5, il est important de remarquer que d'une part, tous les constructeurs de type d'objets

contiennent la notion de hiérarchie, à l'exception des constructeurs d'entités virtuelles et littérales de FM, et que d'autre part, la notion d'atomicité des valeurs est valable pour tous les objets élémentaires de la structure obtenue par l'application des constructeurs.

Nous allons maintenant classer les modèles en fonction des notions de hiérarchie, d'ordre et des possibilités d'utilisation des constructeurs.

1.5 Classement des modèles

Nous pouvons obtenir trois classements en fonction de : (i) l'application des constructeurs, (ii) la présence de la notion de hiérarchie et (iii) la présence de la notion d'ordre. Dans le premier classement, nous pouvons distinguer les modèles cycliques et non-cycliques. Dans le deuxième classement, nous pouvons les classer en hiérarchiques et non-hiérarchiques. Enfin, le troisième classement fait apparâitre les modèles ordonnés et non-ordonnés. Il faut remarquer que les modèles les plus adaptés pour les documents sont les modèles à la fois hiérarchiques et ordonnés. Pour cette raison, nous ne nous attarderons pas sur les autres modèles.

1.5.1 Modèle cyclique

L'application des constructeurs sans restrictions permet d'obtenir des cycles d'utilisation. La possibilité d'appliquer un constructeur à lui-même (récursivité d'application) est un cas particulier de cycle. La figure 1.5 montre les possibilités d'utilisation des constructeurs de plusieurs modèles, ce qui nous permet de visualiser facilement les cycles permis. Par exemple, la possibilité d'utiliser des ensembles de nuplets et des nuplets d'ensembles dans les modèles N1NF et Objets Complexes permet d'obtenir le cycle ensemble-nuplet-ensemble. Il

faut souligner que l'application cyclique des constructeurs permet l'obtention d'un schéma à un nombre quelconque de niveaux.

1.5.2 Modèle non-cyclique

Le seul modèle que nous connaissions et qui ne permette pas l'application cyclique de ses constructeurs est le modèle relationnel classique (voir figure 1.5).

1.5.3 Modèle Hiérarchique

La présence de la notion de hiérarchie dans les constructeurs permet d'obtenir des schémas hiérarchiques. Ainsi, l'ordre d'application de ces constructeurs influe sur la hiérarchie : on distingue la hiérarchie ensemble-nuplets de la hiérarchie nuplets-ensemble.

Nous pouvons trouver la notion de hiérarchie dans les constructeurs d'ensembles, de nuplets, de séquences, d'agrégations, de documents, etc.

Le modèle relationnel étant non-cyclique, la hiérarchie est limitée à trois niveaux maximum.

Il existe cependant des implantations du modèle relationnel qui tentent d'introduire la notion de hiérarchie dans les attributs. C'est le cas, par exemple, les clauses "**connect by**" et "**start with**" d'ORACLE.

1.5.4 Modèle non-hiérarchique

Les modèles basés sur des réseaux ou sur des graphes sont des modèles non-hiérarchiques. Le modèle FM ("Fact Model") illustré aussi par la figure 1.5 n'a pas la notion de hiérarchie ni la notion

d'ordre. Nous décrivons le modèle FM dans l'annexe 1 de cette thèse.

Dans un environnement hypertexte, qui vise fondamentalement l'exploration non-séquentielle des documents, l'utilisation de modèles non-hiérarchiques semble particulièrement importante. La majorité des systèmes d'hypertexte [Sav88] tels que Xanadu, Intermedia et Hyperties ne reposent pas sur une structure hiérarchique.

1.5.5 Modèles ordonnés ou non-ordonnés

Le classement en ordonné ou non-ordonné dépend de l'utilisation de constructeurs qui introduisent ou non la notion d'ordre. Nous pouvons trouver la notion d'ordre dans les constructeurs tels que séquence (MDM, NST,etc), liste (HDBL) et array (MDM, TIGRE, etc).

Comme exemples de modèles non-ordonnés, nous pouvons citer les modèles N1NF et le modèle Objets Complexes (voir figure 1.5).

Malgré l'inexistence de la notion d'ordre dans le modèle relationnel, la majorité des implantations basés sur ce modèle ont introduit, d'une manière informelle, la notion d'ordre (au niveau de nuplets, pas au niveau des attributs qui sont monovalués), par exemple, la clause "**order by**" d'ORACLE.

1.6 Systèmes de Gestion de Bases de Données Documents (SGBDD)

Nous pouvons classer les SGBD capables de stocker, d'accéder et de manipuler les documents en fonction de l'intégration ou du couplage du modèle, du langage et de l'architecture.

1.6.1 SGBDD Intégré

Le système intégré est ainsi caractérisé :

- Au niveau modèle, tous les objets sont traités de la même manière. Ceci signifie que l'intégration des objets multimédia, alphanumériques, etc est faite. Cette intégration dans les systèmes en cours de développement n'est pas évidente.

- Au niveau langage, toutes les opérations sont directement déduites du formalisme du modèle. Par exemple, le langage du modèle NST est directement obtenu de l'algèbre NST (langage algébrique). Un langage intégré pour les systèmes de base de données doit offrir les fonctionnalités qu'on attend d'un langage général de programmation et d'un système de base de données.

- La mise en oeuvre de ce type de système peut être effectuée de deux manières distinctes : soit on fait un nouveau SGBD capable de manipuler toutes sortes d'objets, soit on développe un SGBD de bas niveau qui contient des primitives générales et une couche spécifique pour traiter les différents objets en utilisant ces primitives. Ces deux solutions sont très intéressantes mais semblent difficiles à réaliser [Vel84, LS83].

1.6.2 SGBD Documents couplé

Le système couplé est caractérisé :

- Au niveau modèle, par le couplage entre des modèles différents. Par exemple, le serveur OIS est basé sur un modèle issu du couplage entre le modèle FM et le modèle document ODA. Un autre exemple de système couplé, qui traite aussi des documents, est le sytème d'hypertexte développé à Stanford sous le nom

NLS/Augmented d'Engelbart et English cité par [Sav88]. Ce système couple un modèle hiérarchique à un modèle réseau pour permettre de greffer un réseau de liens sur la structure hiérarchique du document. Les liens établissent des renvois possibles entre les différentes unités documentaires du document tels que la référence à une autre partie du texte (référence bibliographique, référence à une autre unité documentaire du même document, à des figures ou à des graphiques), le renvoi à une version antérieure du même document ou le lien vers une définition appartenant à un glossaire, etc.

- Au niveau langage, le couplage entre des langages classiques et des langages bases de données conduit à un mélange non homogène car ces langages ont des systèmes de type et des paradigmes de programmation très différents [LR89]. Par exemple, le mariage C+SQL : C est impératif et SQL est déclaratif. En effet, la plupart des prototypes actuels offre un langage SQL étendu. Le langage SQL ayant été conçu pour le modèle relationnel, ces extensions constituent en réalité un couplage avec le modèle relationnel.

- L'architecture d'une mise en oeuvre de ce type de système est basée sur des systèmes déjà existants. Nous identifions deux méthodes d'implantation couplées : soit on développe un module capable de manipuler toutes sortes d'objets multimédia sur l'interface d'un système déjà existant, soit on développe des modules indépendants pour traiter chaque type d'objets (aussi sur l'interface d'un système déjà existant). Cette dernière démarche a été utilisée pour implanter le serveur OIS sur ORACLE. Le principal avantage de l'utilisation d'un SGBD déjà existant est de pouvoir profiter des fonctionnalités de celui-ci telles que reprise

après pannes, accès concurrents, etc. Néanmoins, l'inconvénient de rester limité au SGBD sur le plan de le gestion de l'espace physique et des performances peut devenir génant.

Nous allons maintenant décrire une algèbre du point devue pratique pour les documents de bureau afin d'obtenir un système intégré au niveau du modèle et du langage.

CHAPITRE 2

MODELE ET ALGEBRE POUR LES DOCUMENTS STRUCTURES DE BUREAU

2

MODELE ET ALGEBRE POUR LES DOCUMENTS STRUCTURES DE BUREAU

2.1 Introduction

La définition d'un modèle et d'une algèbre associée est un axe de recherche qui vise principalement les objetifs suivants :

1. La formalisation du modèle.
2. La spécification précise des aspects fonctionnels (opérateurs de stockage, d'accès et de construction des objets modélisés).

3. L'obtention directe d'un langage algébrique intégré au modèle.

Dans ce chapitre, nous nous sommes intéressés plutôt aux aspects fonctionnels pour stocker et manipuler les documents structurés en prenant en compte leurs structures logiques. Il est vrai que la manipulation d'une structure logique peut être due à une simple consultation mais également à des opérations plus complexes telles que la restructuration ou l'indexation des éléments appartenant à cette structure.

La présentation ici d'une algèbre pour les Documents Structurés de Bureau (Algèbre DSB) est pratique au lieu de théorique dans le sens qu'on n'essaye pas d'étendre l'algèbre relationnelle et de prouver des théoremes sur cette extension, mais plutôt d'offrir un ensemble utile d'opérateurs pour manipuler la structure logique des documents structurés.

Pour la définition de l'algèbre DSB, nous nous sommes inspirés des travaux réalisés sur les modèles non en première forme normale (NINF) [FT83, Abi84, SS86], les objets complexes [AG87], et plus particulièrement, les documents structurés de bureau [GZC87, Güt88].

Nous commençons par décrire les concepts fondamentaux, les operateurs primitifs et les objets de l'algèbre DSB respectivement dans les sections 2.2, 2.3 et 2.4. Après nous présentons les principales différences entre l'algèbre au niveau formel et l'algèbre au niveau utilisateur (section 2.5). Les principales caractéristiques de cette algèbre sont décrites dans la section 2.6. Dans la section 2.7 nous précisons le modèle DSB. Après, nous nous limitons aux opérateurs suivants : projection (π), sélection (σ), modification (λ), "grouping" (γ), "distribution" (δ), "unpacking" (μ) et "packing" (ν) [GZC87, Güt88]. Les quatre derniers opérateurs remplacent, en

offrant plus de souplesse, les opérateurs "nest" et "unnest" qui sont d'ailleurs très connus dans la bibliographie sur les modèles non en première forme normale (N1NF). Finalement, nous prenons en considération l'utilisation des valeurs nulles afin de faciliter la spécification et la manipulation de la structure logique des documents.

2.2 Concepts fondamentaux

Nous utilisons la terminologie et les notations de Güting [Güt88] pour énoncer les concepts nécessaires à la définition formelle de l'algèbre DSB.

- **nuplet**

Le premier concept fondamental est le nuplet. Le formalisme de Güting d'un nuplet se différencie de celui d'Abiteboul, en ce qu'il introduit le concept d'ordre et ne cherche pas à utiliser les noms d'attributs pour distinguer les différents champs d'un nuplet (Güting utilise des labels numériques). Des remarques sur ces formalismes sont donnés dans la section 2.6 de ce chapitre.

Un nuplet $x = (x_1,...,x_r)$ est un agrégat simple de quelques objets $x_1,...,x_r$ appelés composants. Chaque composant est distingué par la notation x.i qui sélectionne le nième composant noté x_i. Le fait que chaque nuplet a au moins un composant et que tous ses composants sont distincts est noté : $(x_1,...,x_r) \mid r \geq 1 , \forall_i \in [1..r] , \forall_j \in [1..r]. x_i \neq x_j$ si $j \neq i$.

- **Séquence**

Le deuxième concept fondamental est la séquence. Si A est un ensemble alors : une séquence finie sur A est notée $< a_1,...,a_n >$ où $n \geq 0$ et pour $1 \leq i \leq n$ $a_i \in A$. La notation $a_1...a_n$ peut être utilisée

lorsqu'il n'existe pas de doute à propos de l'identification de la séquence.

- **Algèbre multitypes ("many-sorted algèbre")**

Une algèbre multitypes est une collection d'ensembles de types et de fonctions entre ces ensembles. La collection d'ensembles de types est notée S (noms des ensembles) et la collection des fonctions est notée $\Sigma_{w,s}$ (symboles des opérateurs) , où $w \in S^*$ et $s \in S$. La description des ensembles de types s et de la collection des fonctions $\Sigma_{w,s}$ est notée signature des types S. Par exemple, la collection de fonctions applicables aux ensembles du type COMP est illustrée par la figure 2.7 du chapitre 2 de cette thèse.

La définition de signature Σ_1 est la suivante :

Définition

Soit Σ une signature de type S. Alors une algèbre A de Σ consiste en un ensemble d'A_s pour chaque $s \in S$ et une fonction

$\sigma A : A_{s1} \times \ldots \times A_{sn} \rightarrow A_s$

pour chaque $\sigma \in \Sigma_{w,s}$ avec $w = s_1 \ldots s_n$

Par exemple: S = {NUM, TEXT, BOOL, MULT, COMP} est un ensemble de types d'objets de l'algèbre DSB. Les types NUM, TEXT MULT et BOOL représentent des types de données atomiques qui nous permettent d'avoir ATOM = {NUM, TEXT, MULT, BOOL}. Le type NUM est composé à son tour d' INT et REAL. Le type TEXT est composé de CHAR, STRING et TEXTE. Le type MULT est composé de IMAGE et GRAPHICS. Finalement le type simple BOOL contient les objets booléens **TRUE** et **FALSE**. Le type COMP contient tous les objets avec le schéma atomique s où chaque $s \in S$.

- **L'ensemble de séquences finies**

Etant donné un ensemble S, nous notons S^* l'ensemble des séquences finies sur S:

$$S^* = \{<s_1,\ldots,s_n> \mid n \geq 0 \text{ et pour } 1 \leq i \leq n : s_i \in S\}$$

Par exemple pour

S = { INTEGER, REAL} nous avons

S^* = { <INTEGER>, <INTEGER,REAL>, <REAL>,... }

2.3 Les opérateurs primitifs

Pour faciliter la formalisation de quelques opérateurs plus complexes, nous définissons les opérateurs primitifs suivants :

- **Les opérateurs concaténation**

a. Concaténation de nuplets. Le symbole "#" est utilisé pour indiquer l'opération de concaténation de deux nuplets. Par exemple, la notation indiquée ci-dessous concatène le nuplet $x = (x_1,\ldots,x_r)$ au nuplet $y = (y_1,\ldots,y_s)$.

$$(x_1,\ldots,x_r)\ (y_1,\ldots,y_s)\ \# = (x_1,\ldots,x_r\ ,\ y_1,\ldots,y_s)$$

b. Concaténation d'une séquence. Le symbole "o" représente l'opération de concaténation entre deux séquences. La notation représentée ci-dessous illustre la concaténation de deux séquences $a = <a_1,\ldots,a_n>$ et $b = <b_1,\ldots,b_m>$.

$$<a_1,\ldots,a_n> <b_1,\ldots,b_m> o = <a_1,\ldots,a_n\ ,\ b_1,\ldots,b_m>$$

- **Sous-séquence d'une séquence**

Le concept de sous-séquence est défini à l'aide de l'opérateur primitif concaténation de séquence ("o"). Une séquence c est appelée une sous-séquence d'une séquence d si α et β existent tels que d = α o c o β.

Définition

Une séquence $c = c_1,\ldots,c_n$ est une sous-séquence d'une séquence $d = d_1 \ldots d_m$,

$c \subseteq d$:<=> existe $<\alpha>$ et $<\beta>$: $d = \alpha \circ c \circ \beta$

- **Egalités**

a. Egalité entre deux nuplets. L'égalité entre deux nuplets x e y est définie de la façon suivante:

Définition

Deux nuplets $x = (x_1,\ldots,x_r)$ et $y = (y_1,\ldots,y_s)$ sont égaux,

$x = y$:<=> $r = s$ et pour $1 \leq i \leq r$: $x_i = y_i$.

b. Egalité entre deux séquences. Deux séquences a et b sont égales si leurs éléments sont égaux:

Définition

Deux séquences $a = a_1,\ldots,a_n$ et $b = b_1,\ldots,b_m$ sont égales:

$a = b$:<=> $n = m$ et pour $1 \leq i \leq n$: $a_i = b_i$.

- **L'opérateur primitif modulo**

L'opérateur primitif modulo partage une séquence en plusieurs sous-séquences sur un ensemble A selon une relation d'équivalence R. Par exemple, étant donné l'ensemble A = {<3,1,7,2,4,4,5,6,8,4,2>} et

la relation R qui identifie des séquences d'entiers pairs ou impairs, nous avons :

<3,1,7,2,4,4,5,6,8,4,2> modulo [R] = <<3,1,7>,<2,4,4>,<5,6,8,4,2>>

2.4 Objets de l'algèbre DSB

Les objets de l'algèbre DSB sont des objets atomiques tels qu'entier, réél, texte, booléen, graphique ou sons ou des objets structurés composés d'objets atomiques. Une représentation complète d'un objet consiste en deux parties : schéma et occurrence.

Pour des raisons de simplicité conceptuelle ainsi que par souci du problème d'augmentation de la quantité d'informations de description des données, nous avons groupé les objets similaires dans une classe. Une classe est caractérisée par un schéma qui décrit la structure et les valeurs possibles des objets de cette classe. Dans un environnement orienté-objets les objets appartenant à une classe sont appelés **instances** de cette classe. Dans ce travail, on utilise le mot **occurrences** comme synonyme du mot **instances**. Ainsi, un schéma (s) et un élément d'un ensemble d'instances de s donnent la définition d'objet de l'algèbre :

Définition

Un objet de l'algèbre est une paire (s,o) où s est un schéma et o est une occurrence de s.

Un schéma atomique est décrit par un identificateur et par un ensemble de valeurs possibles associées par une fonction *dom*. Par exemple, soit BOOL un identificateur, **dom**(BOOL) = {**false**, **true**}. Soit ATOM = {**NUM, TEXT, BOOL, MULT**} l'ensemble qui

contient les identificateurs des éléments atomiques, nous pouvons définir schémas et instances de la manière suivante :

Définition

a. $s \in$ **ATOM** => s est un schéma atomique. L'ensemble des valeurs possibles de l'objet décrit par s est :

$$\text{instances}(s) := \textbf{dom}(s)$$

b. $s_1,\ldots,s_r$ sont des schémas => $(s_1,\ldots,s_r)$ est un schéma composé. Pour chaque $s = s_1,\ldots,s_r$,

$$\text{instances}(s) := \{ a_1 \ldots a_n \mid n \geq 0 \text{ et pour } 1 \leq i \leq n: a_i = (a_{i,1},\ldots,a_{i,r}) \text{ et pour } 1 \leq j \leq r: a_{i,j} \in \text{instances}(s_j) \}$$

2.5 Différences entre l'algèbre formelle et l'algèbre au niveau utilisateur

L'algèbre formelle et l'algèbre au niveau utilisateur présentent quelques petites différences parmi lesquelles nous décrivons les trois suivantes :

1. Les noeuds de l'arbre du schéma, au niveau de l'utilisateur, possèdent des labels textuels qui facilitent la manipulation, tandis qu'au niveau formel ces labels ne sont pas nécessaires. En effet, ces labels textuels sont remplacés facilement, au niveau formel, par des identifiants numériques. La figure 2.1 illustre cette différence.

   Selon la définition de schéma que nous avons retenue, tous ces labels numériques présentent des redondances avec les labels textuels (noms externes). Au niveau de l'algèbre, on

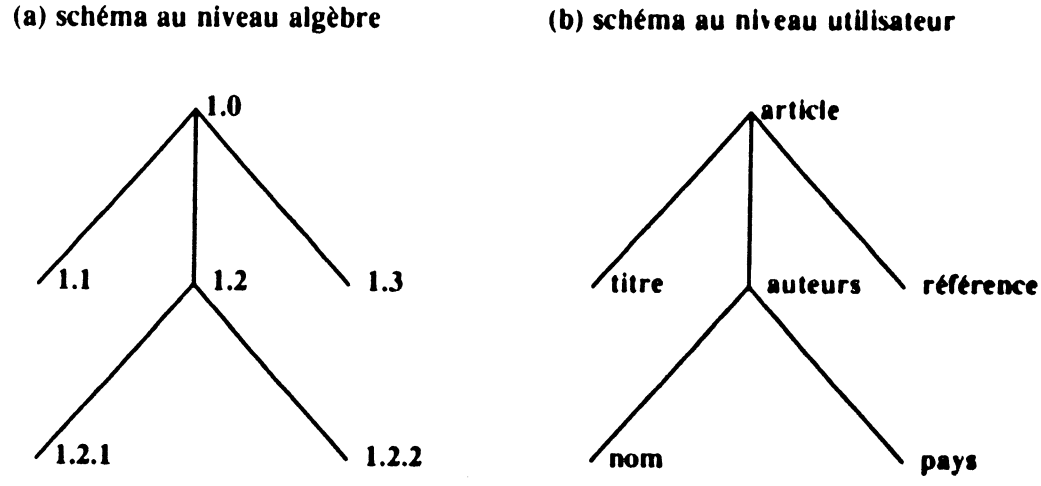

Fig. 2.1 : Des schémas aux niveaux algèbre et utilisateur

s'abstrait ainsi des problèmes liés à la numérotation, à la désignation et au *mapping* entre ces labels (numériques et textuels). Ceci permet d'une part d'éviter le traitement de conflits de noms et d'autre part d'appliquer des opérateurs, tels qu'union et concaténation, aux objets complexes qui ont les mêmes schémas mais sans qu'il soit pour autant nécessaire d'avoir les mêmes labels.

2. L'utilisateur peut employer des expressions avec des constantes sans qu'il soit pour autant nécessaire de les introduire d'une manière formelle dans le modèle. Par exemple l'expression choisie pour sélectionner l'article de titre "L'algèbre DSB" est illustrée par l'opération de sélection ci-dessous :

$$\textbf{article } \sigma \textbf{ [titre = "L'algèbre DSB"]}$$

3. Finalement, du point de vue de l'utilisateur, il est possible d'appliquer tous les opérateurs qui transforment un objet

composé en un autre objet composé sur n'importe quel sous-arbre. Pour cela, il suffit d'utiliser le label de la racine du sous-arbre. Par exemple :

a) article σ [référence = "v1"]

b) auteurs σ [nom = "Martin"]

2.6 Caractéristiques

L'algèbre DSB est caractérisée par l'utilisation de certains principes tels que l'ordre, l'agrégation, la répétition, ainsi que par l'utilisation de certaines valeurs nulles qui lui sont propres. Nous allons maintenant en décrire les principales caractéristiques :

1. Comme l'algèbre NST [Güt88], dont elle est inspirée, l'algèbre DSB est une algèbre multitypes. Cela signifie que des objets atomiques tels que les nombres, les textes, les booléens, les images, ou les objets composés d'objets atomiques sont traités de manière uniforme par les opérateurs.

   Tous les opérateurs, tels que les opérateurs mathématiques (addition, soustraction, multiplication et division) les opérateurs de sélection, de jointure ou d'agrégation, sont utilisés d'une manière uniforme.

2. Elle utilise un mécanisme original et simple combinant les principes d'agrégation et de répétition. Ce mécanisme consiste en une règle : tous les noeuds, soient les noeuds intermédiaires qui sont des objets composés soient les noeuds feuilles qui sont des objets atomiques, sont susceptibles de répétition. Ceci est déjà une grande différence de l'algèbre NST qui ne permet pas la répétition des objets atomiques. L'avantage de l'utilisation

de ce mécanisme est que le modèle formel est grandement simplifié et que le traitement des objets de taille importante est transparent pour l'utilisateur.

3. Elle utilise une valeur nulle originale : la valeur nulle **noseqniv**. Cette valeur indique qu'une séquence est facultative sans qu'il soit pour autant nécessaire d'indiquer l'aspect facultatif de ses composants. C'est-à-dire que nous pouvons indiquer l'aspect facultatif d'un élément dans un niveau spécifique. Ceci correspond, en effet, à une opération "unnest" qui permet à partir d'un schéma étendu d'arriver à un schéma simplifié en conservant toutes les occurrences stockées du schéma concerné. Les deux principaux avantages de l'utilisation d'une telle valeur sont tout d'abord de permettre la définition d'un schéma simple et plus proche de la réalité et ensuite de supporter des opérateurs de transformations de schéma en conservant les occurrences déjà stockées. Pour mieux comprendre l'utilisation de cette valeur, nous allons représenter un livre qui peut être organisé en parties composées de chapitres ou organisé directement en chapitres. Dans les représentations arborescentes utilisés comme exemples, nous lions les éléments logiques composés à leurs composants par des flèches semblables à celles montrées dans la figure 2.2 pour indiquer les aspects multivalués, monovalués et d'existence (optionnelle ou obligatoire) des éléments composants.

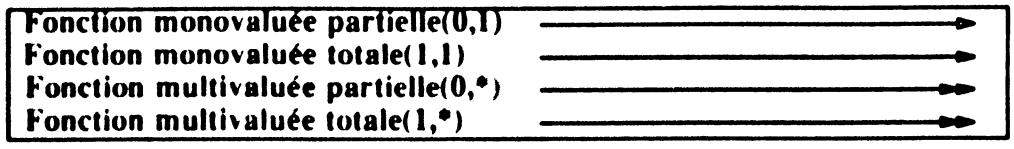

Fig. 2.2 : Aspects multivalués et monovalués des éléments.

Deux modélisations de livre sont illustrés dans la figure 2.3 en utilisant les flèches montrées par la figure 2.2. Le modelisation a) est plus longue que la modélisation b) parce qu'elle oblige à spécifier deux fois l'élément chapitre. Il faut souligner que, contrairement à la valeur nulle "noseqniv", la valeur nulle "noseq" prend en compte tous les niveaux du sous-arbre identifié par l'élément logique indiqué.

Dans ces deux cas, nous pouvons permettre l'accès à tous les chapitres des livres stockés dans la base sans qu'il soit pour autant nécessaire de connaître l'existence de l'élément partie. C'est-à-dire que les chapitres liés à l'élément logique partie (**Livre.partie.chapitre**) ainsi que les chapitres liés directement à l'objet lui même (**Livre.chapitre**) sont déduits du schéma. Ceci correspond, en effet, à une vision "aplatie" du document.

4. Elle prend en compte les objets atomiques multimédia tels qu'image, texte et graphique. En effet, les objets multimédia sont prévus lors de la modélisation mais aucun opérateur spécifique est défini.

5. Le schéma est composé par des attributs sans noms. Ceci signifie qu'au niveau de l'algèbre formelle on ne peut pas utiliser des noms d'attributs pour distinguer les différents champs d'un nuplet. En effet, l'algèbre n'utilise que des labels

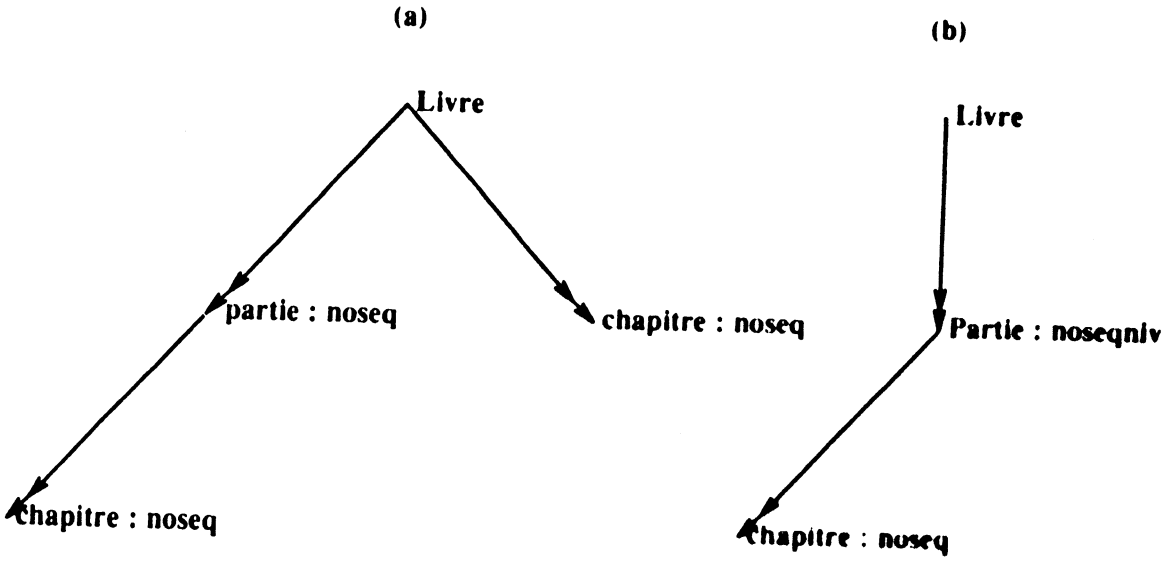

Fig. 2.3 : Des modèles avec les valeurs nulles "noseq" et noseqniv".

numériques avec une notation struturée en prenant en considération le niveau et l'ordre dans ce niveau. Ceci constitue la différence fondamentale entre notre approche (analogue a celle de Güting [Güt88] et le formalisme d'Abiteboul [AG87].

(TITRE:TEXT,(NOM:TEXT,PAYS:TEXT),REF:TEXT) - Abiteboul

(TEXT,(TEXT,TEXT),TEXT) - Güting

L'avantage de la non-utilisation de noms d'attributs dans le schéma est double. D'abord elle simplifie l'algèbre parce qu'elle évite les problèmes de renommage et de conflits de noms. Ensuite elle permet l'exécution de certains opérateurs de manipulation de la hiérarchie sans qu'il soit pour autant nécessaire de modifier le schéma.

6. Elle représente et maintient l'ordre des objets séquences. L'avantage en est double : tout d'abord il est possible d'introduire d'une manière simple des opérateurs de tri (ce qui est réalisé artificiellement dans l'algèbre relationnelle par le biais de l'introduction d'opérateurs du type "order by") et facilite le traitement des duplications.

2.7 Modèle descriptif associé à l'algèbre

Certains modèles, comme par exemple le modèle formulaire OFS [Tsi80], ne permettent pas simultanément l'utilisation de l'agrégation et de la répétition; d'autres ne permettent que la combinaison agrégation-répétition au premier niveau [RS82, Zlo82]. Dans le modèle NST tous les eléments intermédiaires sont construits par une combinaison agrégation-répétition tandis que les éléments feuilles ne peuvent pas être répétés. La contribution de notre approche (modèle DSB) est l'utilisation combinée des principes d'agrégation et de répétition dans tous les niveaux intermédiaires et feuille. Il est vraie que du point de vue pratique la répétition d'un élément feuille est très importante : par exemple plusieurs auteurs. Il est important de remarquer que les contraintes de l'utilisation des principes d'agrégation et de répétition vise à faciliter le formalisme du modèle sans restreindre son pouvoir d'expression. Un autre apport du notre approche est l'utilisation de seulement deux constructeurs (constructeur d'objets composés et constructeur d'objets atomiques) contre trois constructeurs utilisés dans celle de Güting.

En utilisant la même notation graphique que celle de la figure 2.3, la figure 2.4 représente un schéma DSB où Guide est une sous-classe de la classe document (illustré par la ligne pointillée).

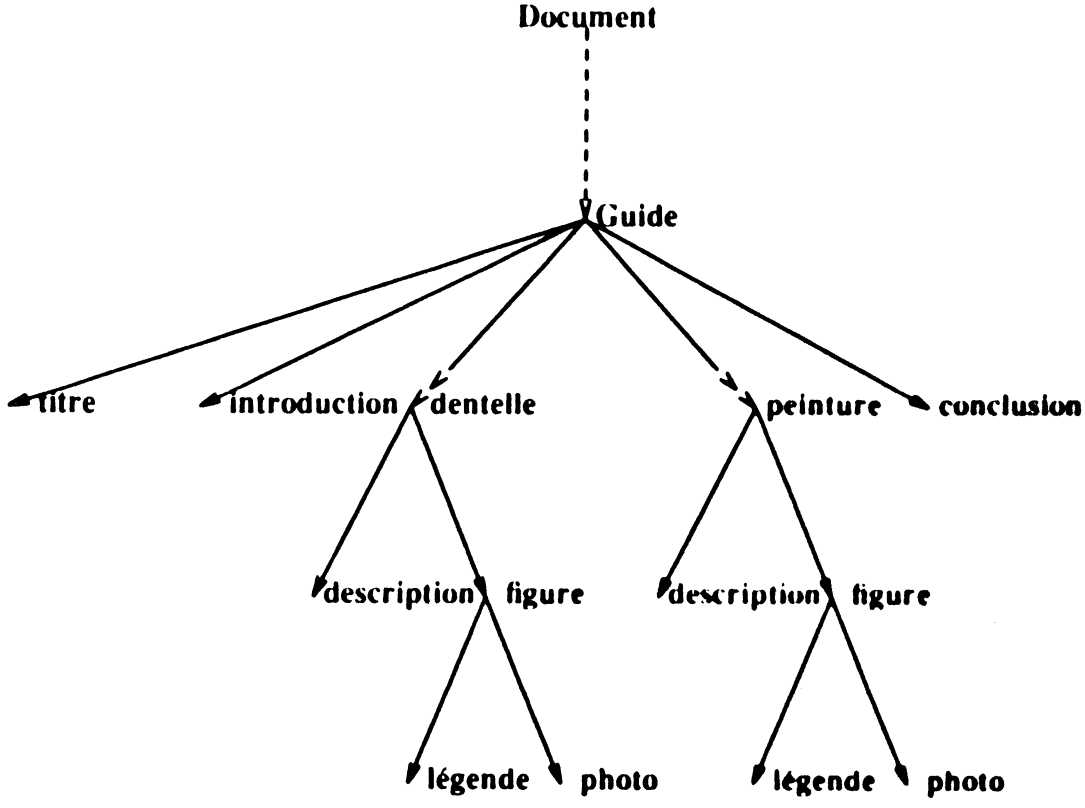

Fig. 2.4 : Sous-classe Guide de musée.

Ce schéma correspond à celui utilisé dans [DeL88, DeL89]. Les concepts de classe, sous-classe et sur-classe sont utilisés dans la terminologie des bases de données orientées-objets [KC88, KMN89].

La représentation graphique d'une sous-classe explicite les noms des noeuds qui peuvent être référencés comme racine des sous-arbres qu'ils identifient. On accède en réalité à chaque racine par une fonction de même nom. Ceci équivaut à introduire des aspects langage de programmation dans le système de base de données.

2.8 Opérateurs applicables aux documents

Comme nous l'avons déjà dit notre principal objectif , dans ce chapitre, n'est pas de décrire formellement les opérateurs, parce que de nombreux travaux ont déjà été réalisés sur ce sujet, mais plutôt de présenter, sur un plan pratique, l'application de certains de ces opérateurs à la bureautique. Nous n'avons pas non plus l'intention de donner des exemples pour chacun des trente-cinq opérateurs applicables aux objets complexes spécifiés par Güting et al. [GZC87, Güt88] (cf. figure 2.7).

Avant de donner des exemples d'opérateurs applicables aux objets complexes et plus spécifiquement d'opérateurs applicables aux documents structurés de bureau, nous présentons plusieurs classifications de ces opérateurs.

Nous spécifions plutôt les opérateurs qui prennent en compte la structure hiérarchique des documents. Lors de la manipulation de cette structure, certains opérateurs peuvent, tout en préservant les éléments atomiques existants, créer de nouveaux éléments composés. C'est le cas, par exemple, des opérateurs tels que le "grouping" (γ) et la "distribution" (δ). D'autres opérateurs, plus particulièrement l'opérateur modification, peuvent créer de nouveaux éléments atomiques. Par conséquent, dans certains cas, il est nécessaire de réaliser la transformation d'un élément atomique en élément composé.

2.8.1 Classement des opérateurs

Le classement des opérateurs d'une algèbre pour les objets complexes peut être fait de plusieurs manières. Ainsi, Collet [Col88] identifie deux types d'opérateurs applicables aux Formulaires Abstraits (AF) : opérateurs d'interrogation et opérateurs de mise à

jour. Les opérateurs d'interrogation sont à leur tour classés comme opérateurs (i) de filtrage, (ii) ensemblistes et (iii) de restructuration. Ce classement, illustré par la figure 2.5, est à rapprocher du classement défini dans l'algèbre d'Abiteboul et Beeri [AB87].

OPERATIONS	SIGNIFICATION
de filtrage	
sélection	sélection d'occurrences de FA
élaguer	projection sur un FA
renommer	renommage de FA et/ou de ses éléments
ordonner	ordonner des données
ensemblistes	
union	union de deux FA
intersection	intersection de deux FA
différence	différence de deux FA
produit	produit cartésien de deux FA
de restructuration	
grouper	création d'un niveau d'imbrication
éclater	suppression d'un niveau d'imbrication

Fig. 2.5 : Classement des opérateurs d'interrogation.

Les travaux de Güting, Zicari et Choy [GZC87, Güt88] présentent une autre méthode de classement basée sur le type des opérandes et de l'objet résultant de l'opération.

Pour décrire l'applicabilité des opérateurs, les auteurs ont regroupé les schémas en quatre classes : NUM, TEXT, BOOL et COMP. La classe NUM englobe les schémas simples comme INT, REAL ou LONG REAL. La classe TEXT englobe les schémas simples de chaînes de caractères tels que CHAR, STRING, TEXT et

VARCHAR. La classe BOOL contient le schéma simple booléen. Finalement, le type COMP contient tous les schémas composés de schémas simples.

Ce type de classement est très intéressant parce qu'il facilite d'une part l'association d'une notation (syntaxe) par classe d'opérateurs et qu'il définit d'autre part la fonctionnalité des opérateurs. Par exemple, l'opérateur d'addition "+" qui a la fonctionnalité "NUM x NUM -> NUM", indique qu'il peut être appliqué à deux objets contenant des schémas appartenant à la classe NUM (INT ou REAL) et qu'il obtient comme résultat un objet doté d'un schéma appartenant lui aussi à la classe NUM.

La figure 2.6 montre tous les opérateurs pouvant être appliqués à des objets qui possèdent des schémas de la classe COMP. La notation (_ #) et (_ _#) indique que les opérateurs utilisent une sintaxe post-fixé.

Une autre méthode consiste à classer les opérateurs en fonction de leurs paramètres. Plus les types de paramètres acceptés par un opérateur sont sophistiqués, plus il est difficile de les formaliser. Ainsi, afin de faciliter la compréhension de la formalisation des opérateurs on peut les sous-classer en fonction du type de paramètre accepté : simples, expressions algébriques et sans paramètres. La figure 2.7 montre ce sous-classement.

Nous allons maintenant donner des exemples de quelques opérateurs :

2.8.2 Une vision informelle de quelques opérateurs

Dans cette section nous essayons de donner des exemples de quelques opérateurs utiles pour la manipulation de la structure logique

FONCTIONNALITE/NOTATION	OPERATEURS	SYMBOLE
COMP x COMP → COMP/(_ _ #)	union intersection difference cartesian product join concatenation pairing	∪ ∩ - x |x| **concat** **pair**
COMP → COMP/(_ #)	projection removal ordering rem.duplicates head tail portion reverse grouping distribution unpaking packing selection modification looping	π ρ **ord** **rdup** **head** **tail** **portion** **reverse** γ δ μ ν σ λ λ
COMP x COMP → BOOL/(_ _ #)	equal not equal improper subset improper superset	= ≠ ⊆ ⊇
COMP x COMP x COMP → COMP	if.then.else.fi	**if**
COMP → NUM/(_ #)	count avg sum min max	**count** **avg** **sum** **min** **max**
COMP → BOOL/(_ #)	isempty exists forall	**isempty** **exists** **forall**

Fig. 2.6 : Classement par fonctionnalité des opérateurs sur COMP.

PARAMETRES	OPERATEURS	SYMBOLE
inexistants	union	$\cup$
	intersection	$\cap$
	difference	-
	cartesian product	**x**
	concatenation	**concat**
	pairing	**pair**
	equal	=
	not equal	$\neq$
	improper subset	$\subseteq$
	improper superset	$\supseteq$
simples	projection	π
	removal	ρ
	ordering	**ord**
	rem.duplicates	**rdup**
	head	**head**
	tail	**tail**
	portion	**portion**
	reverse	**reverse**
	grouping	γ
	distribution	δ
	unpaking	μ
	packing	ν
	isempty	**isempty**
expressions algébriques	selection	σ
	modification	λ
	looping	λ
	join	$\|x\|$
	exists	**exists**
	forall	**forall**

Fig. 2.7 : Classement par type de paramètres.

des documents.

2.8.2.1 L'opérateur projection (π)

Cet opérateur est semblable à l'opérateur de projection de l'algèbre relationnelle. Il peut être utilisé pour restructurer les documents. L'exemple montré ici sélectionne les composants spécifiés au niveau de l'utilisateur par le biais de labels textuels. Il faut souligner que cet opérateur peut spécifier les composants indépendamment de l'ordre du schéma, en permettant ainsi le changement de l'ordre des composants. Un exemple de l'opération de projection est illustré par la figure 2.8. Il faut remarquer que la présentation des opérateurs est post-fixée c'est qui signifie qu'ils sont appliqués à la racine de l'arbre à gauche et donne comme résultat l'arbre à droite. Les paramètres des opérateurs peuvent être explicités entre crochets ou déduits par la difference entre tous les sous-arbres de la racine et celles explicitées entre crochets. Aussi, le nom d'un nouveau niveau à créer est designé entre accolades.

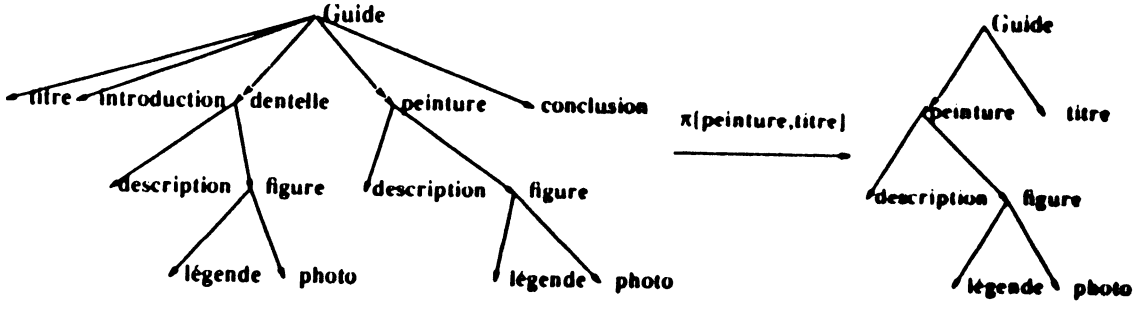

Fig. 2.8 : Opération de projection.

2.8.2.2 L'opérateur grouping (γ)

Le principal objectif de cet opérateur de restructuration est de répartir une collection de nuplets en sous-ensembles de nuplets ayant les mêmes valeurs et construire ensuite un nuplet hiérarchique pour chaque sous-ensemble. Cet opérateur est à rapprocher : i) de

l'opérateur **grouper** de l'algèbre pour les formulaires de Collet [Col87], ii) de l'opérateur **nest** de l'algèbre relationnelle étendue de Fischer, Thomas, Jaeschke, Schek et Scholl [JS82, FT83, SS86] et iii) de l'opérateur **packing** décrit postérieurement.

Cet opérateur trouve son application dans des domaines tels que l'édition de documents techniques, la gestion de versions de documents, enfin, dans tous les domaines caractérisés par l'existence de parties entières identiques dans les documents traités.

Pour rappeler les effets de cet opérateur nous allons d'abord vérifier l'aspect transformation du schéma en utilisant l'exemple illustré dans la figure 2.9 qui regroupe les dentelles et les peintures du schéma guide comme oeuvres d'arts et ensuite l'aspect regroupement des occurrences.

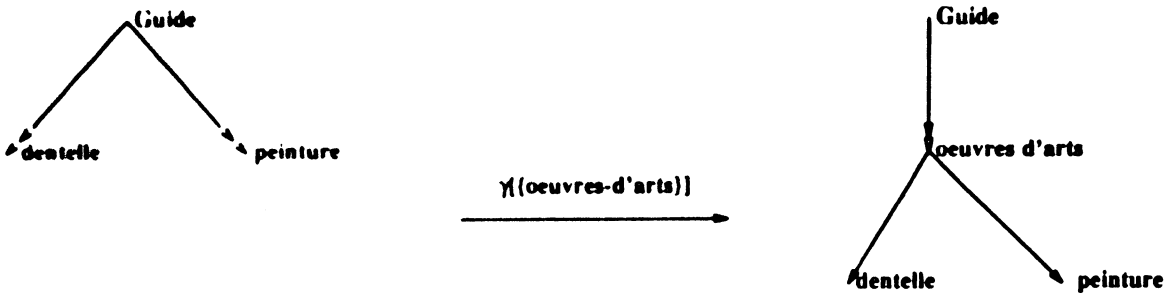

Fig. 2.9 : Opération "grouping".

Pour voir précisément le travail de regroupement des occurrences fait par cet opérateur nous allons utiliser la relation **Manuel = <titre, introduction, version, date>**. Une séquence de nuplets possibles pour ce schéma peut être :

< (Oracle, A, v1, 1/08/85),
(Oracle, A, v2, 12/10/86),
(Oracle, A, v3, 13/12/88),

(SPIX, B, v1.1, 12/10/88),
(SPIX, B, v1.2, 30/12/88),
(UNIX, C, v8.9, 30/05/89) >

Supposons que nous voulions restructurer les documents ayant les mêmes titres et introductions. Ceci est rendu possible par l'opérateur grouping suivant :

Manuel γ [titre, introduction{référence}]

Le résultat du regroupement des occurrences peut être vu ci-dessous :

< (Oracle, A, < (v1, 1/08/85),(v2, 12/10/86),(v3, 13/12/88) >,
(SPIX, B, < (v1.1, 12/10/88),(v1.2, 31/12/88) >,
(UNIX, C, < v8.9, 30/05/89 >) >

La procédure formelle d'exécution de cet opérateur utilise des opérateurs primitifs tels que **ord** et **modulo**, et peut être vue en [GZC87]. Nous définissons les opérateurs primitifs de l'algebre DSB dans la section 2.3 de ce chapitre.

2.8.2.3 L'opérateur distribution (δ)

Cet opérateur sélectionne des composants d'un niveau pour en faire les composants d'un sous-arbre déjà existant. La principale différence entre cet opérateur et l'opérateur "grouping" défini provient de ce qu'il ne crée pas toujours un nouveau niveau d'imbrication. Par exemple, les effets sur le schéma de cet opérateur peuvent être visualisés dans la figure 2.10.

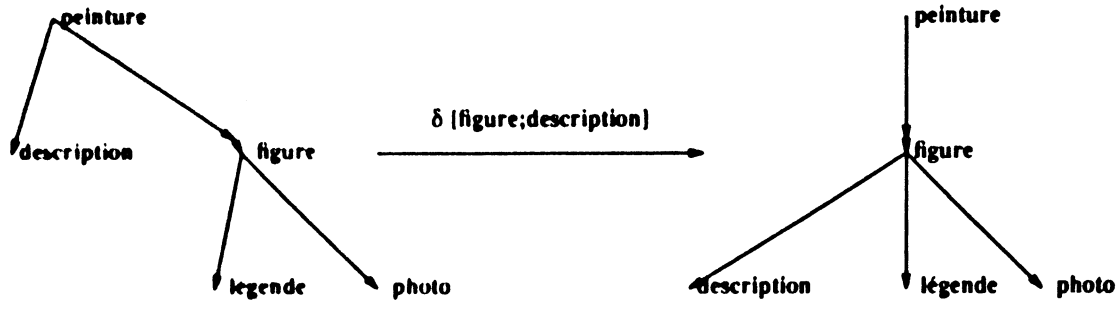

Fig. 2.10 : Opération "distribution"

2.8.2.4 L'opérateur unpacking (μ)

Cet opérateur élimine un niveau d'imbrication, ceci équivaut à éliminer un sous-arbre ou un objet composé. Cet opérateur peut être utilisé pour aplatir un schéma (voir figure 2.12).

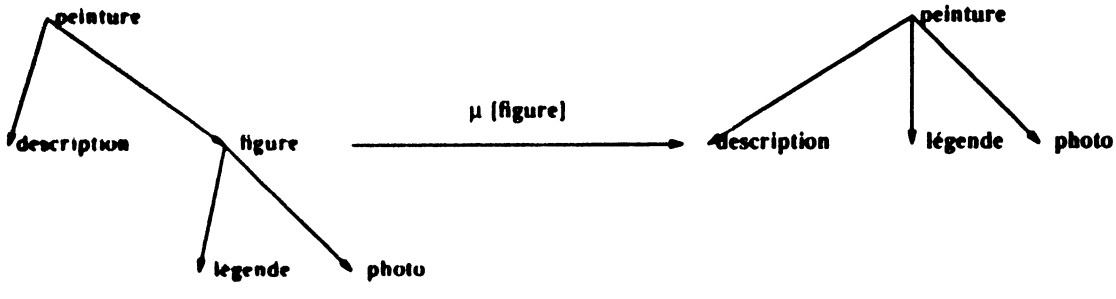

Fig. 2.11 : Opération "unpacking".

2.8.2.5 L'opérateur packing (ν)

Cet opérateur crée un nouveau niveau d'imbrication. Il effectue l'opération inverse de l'opérateur unpacking (μ). Par exemple, la figure 2.12 illustre la création d'un nouveau niveau d'imbrication (figure). Cet opérateur se differencie de l'opérateur **grouping** parce

qu'il ne regroupe pas les occurrences qui ont les mêmes valeurs relativement au niveau d'imbrication créé.

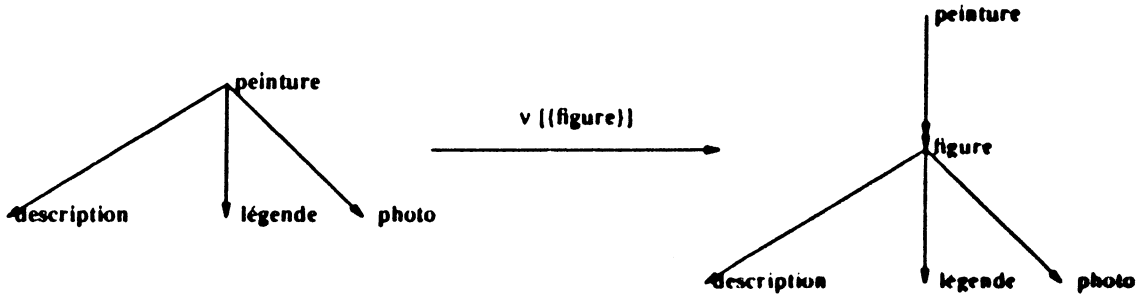

Fig. 2.12 : Opération "packing".

2.8.2.6 L'opérateur sélection (σ)

L'opérateur sélection préserve le schéma mais prend en compte la structure définie par celui-ci. L'expression de sélection donnée comme exemple ci-dessous fait appel à l'opérateur spécial **(talks about)** qui vérifie, dans un élément composé, toutes les portions du type texte qui contiennent une occurrence d'une chaîne spécifique de caractères. Par exemple, supposons que nous cherchions tous les guides qui contiennent des légendes de peintures traitant de la Joconde :

Guide σ [(peinture.légende talks about "La Joconde")]

L'expression algébrique donne comme résultat un schéma booléen, c'est-à-dire vrai ou faux.

2.8.2.7 L'opérateur modification (λ)

L'opérateur modification est nécessaire lors qu'on crée un nouvel élément logique atomique. L'affectation d'une valeur à ce nouvel

élément peut être effectuée de deux manières distinctes :

1. **Création d'une nouvelle valeur** : la valeur associée à un élément peut être le résultat d'un calcul : par exemple, la création d'un élément "total" qui contiendra le nombre d'auteurs d'un document. Ceci signifie que la valeur sera créée avant son affectation. Voir figure 2.13.

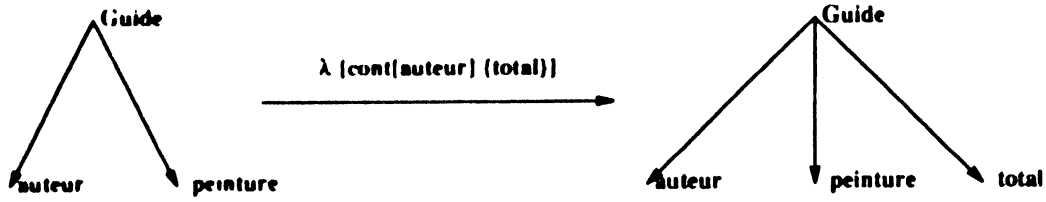

Fig. 2.13 : Opération "modification" avec création d'une valeur.

2. **Reconnaissance de la valeur** : quelquefois la valeur existe déjà. Dans ce cas, la reconnaissance des valeurs correspondant aux éléments à être crées est nécessaire. Par exemple, la création des éléments prénom et nom de l'auteur lors de l'existence d'un élément auteur (cf. figure 2.14).

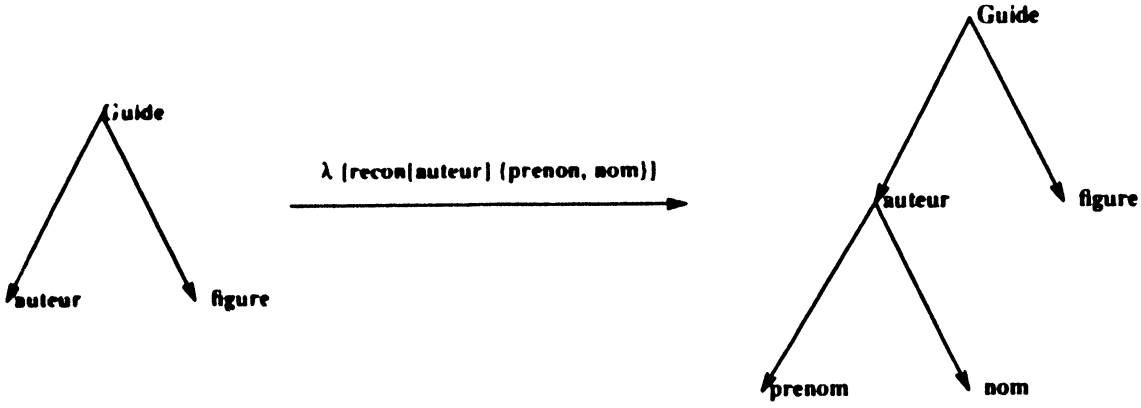

Fig. 2.14 : Opération "modification" avec reconnaissance d'une valeur.

2.9 Traitement des valeurs nulles

Le traitement de valeurs nulles a fait l'objet d'une grande attention des chercheurs en Bases de Données [DA82, HY84, AH86, AG87, GZC87]. Dans un contexte bureautique, l'utilisation de valeurs nulles semble particulièrement intéressante pour représenter des situations significatives. C'est le cas d'une lettre anonyme qui n'a pas l'élément expéditeur. Cet élément peut ne pas exister ou au contraire exister sans que pour autant sa valeur ne soit connue à un moment donné.

Les principaux avantages de l'utilisation des valeurs nulles pour traiter les documents sont tout d'abord de faciliter la spécification d'une structure logique générique et ensuite de pallier le problème de l'appauvrissement sémantique au niveau physique. La spécification souple d'une structure logique peut être obtenue en utilisant des constructeurs au lieu d'appliquer des valeurs nulles : par exemple, la figure 2.15 illustre l'utilisation du constructeur de choix de structure (cas) pour modéliser une classe générale d'un formulaire "personne". Selon le cas, cette modélisation permet de spécifier deux options de structure du formulaire :

- Si sexe est "masculin", l'élément "position militaire" est une composante du formulaire et, en conséquence, doit être remplie.
- Si sexe est "féminin" et mariée, au lieu de l'élément "position militaire", sont associés deux autres éléments: "nom de jeune fille" et "nombre d'enfants".

Le constructeur "cas" est aussi utilisé par le modèle "Format" de Hull et Yap [HY84] et le modèle TIGRE de Velez [Vel84], ce constructeur s'apparente aux variants d'enregistrement (CASE) de pascal.

```
NOM:    chaine de caractères,
PRENOM: chaîne de caractères,
SS#: entier,
cas SEXE de m/f
  m:  POSITION MILITAIRE: chaîne de caractères,
  f:  ENFANTS.    {ENFANT:chaîne de caractères}
                          cas MARIEE de oui/non
                            oui: NOM_DE_JEUNE_FILLE:chaine de caractères,
    fin_cas
fin_cas
```

Fig 2.15. Type 1-formulaire "personne" avec le constructeur cas

Afin de faciliter la modélisation de classes de documents appelés "formulaires", Abiteboul et Hull [AH86] ont étendu le modèle "Format" par l'adjonction de la valeur nulle "non-applicable" qui indique la facultativité d'un élément. La figure 2.16 illustre l'exemple précédent utilisant la valeur nulle "non-applicable".

```
NOM      chaine de caractères,
PRENOM   chaîne de caractères,
SS#      entier,
POSITION MILITAIRE: chaîne de caractères | non-applicable,
MARIE:oui | non
ENFANTS  {ENFANT:chaîne de caractères | non-applicable},
NOM_DE_JEUNE_FILLE:chaîne de caractères | non-applicable,
SEXE: m | f
```

Fig 2.16. Type 2-formulaire "personne" avec la valeur non-applicable

Nous pensons que l'utilisation des valeurs nulles pallie le problème de l'appauvrissement sémantique au niveau physique, essentiellement parce qu'elle permet le maintien de l'information des éléments existants, totalement ou partiellement inconnus, par le biais de la propre valeur stockée. Ceci permet de répondre facilement aux requêtes telles que << Quelles sont les lettres qui ne possèdent pas

d'expéditeur >>.

Néanmoins, le traitement des valeurs nulles pose les problèmes suivants :

1. L'association de la sémantique à chaque valeur nulle.
2. L'obtention d'une représentation interne adéquate pour chaque valeur.
3. L'extension de l'algèbre associée au modèle d'objets de type document.

Nous allons décrire maintenant ces problèmes et suggérer des solutions.

2.10 Association de la sémantique aux valeurs nulles

De nombreux auteurs affirment que les multiples interprétations différentes associées aux valeurs nulles constituent les principales difficultés de traitement de ces valeurs. Nous avons introduit six valeurs nulles que nous jugeons intéressantes pour traiter la structure logique des documents. La sémantique associée à chacune de ces valeurs est illustrée par la table 2.1 ci-dessous. En outre, nous avons introduit une valeur spéciale appélée **vm** afin de faciliter le stockage et l'accès aux éléments multivalués.

Les trois premières valeurs du tableau ont été associées aux éléments feuilles et les autres trois dernières valeurs ont été associées aux éléments intermédiaires. En ce qui concerne la valeur spéciale **vm** elle est associée aux éléments feuilles multivalués ou monovalués.

Les valeurs nulles associées aux éléments feuilles sont illustrées par le biais de l'exemple de l'élément "auteur d'un document".

Table 2.1 Signification des valeurs nulles

Valeurs	Signification
ni	aucune information sur l'élément atomique
dne	l'élément atomique n'existe pas
unk	l'élément atomique est inconnu
noseq	l'élément composé n'existe pas
noseqniv	l'élément composé n'existe pas à ce niveau
niseq	aucune information sur l'élément composé

Supposons qu'il soit intéressant de modéliser les trois situations possibles:

a. Le document n'a pas d'auteur (**dne**). Par exemple, une notice d'un appareil électrique.

b. Le document a un auteur mais il est inconnu pour le moment (**unk**).

c. Aucune information sur l'auteur n'est connue (**ni**).

La première situation (a.) utilise la valeur nulle "**dne**". Ceci signifie que le document n'a pas d'auteur. Il est vrai que la sémantique d'un auteur n'est pas utile pour tous les documents comme, par exemple, ceux qui ne sont pas soumis à un système de droits d'auteur. Aucune mise-à-jour de la base ne peut changer la valeur nulle "**dne**". La deuxième situation (b.) caractérise l'existence de l'auteur, mais pour une raison quelconque, la valeur de celui-ci n'est pas encore connue, soit en raison de l'anonymat du document, soit par simple absence d'information. L'auteur avec la valeur nulle "**unk**" (inconnue) ne pourra jamais voir sa valeur transformée en "**dne**" ou "**ni**". La troisième situation (c.) permet le changement de la valeur auteur sans restriction. Ceci signifie que la valeur peut devenir "**dne**", "**unk**" ou une valeur réelle.

Nous illustrons, par le biais du document "Livre" (cf. figure 2.3), un exemple de valeurs nulles associées aux éléments composés. Dans ce cas nous identifions trois situations distinctes et intéressantes à modéliser :

a. On sait que le livre n'a pas d'élément composé partie (**noseq**).

b. On sait que le livre n'a pas de partie mais qu'il est composé directement de chapitres (**noseqniv**).

c. On ne connait rien sur les éléments composés partie et chapitres du livre (**niseq**).

Dans ces conditions, la première situation (a.) est représentée par la valeur séquence vide (**noseq**). La deuxième situation (b.) est parfaitement modélisée par la valeur nulle (**noseqniv**) qui indique que cet élément n'existe pas à ce niveau et que ses composants peuvent exister. La troisième situation (c.) indique qu'aucune information sur les éléments composés partie et chapitre du livre n'est disponible.

La valeur spéciale **vm** indique que l'élément est multivalué. Il est important de remarquer que la valeur spéciale **vm** est intéressante lors du stockage de l'élément : elle permet de choisir une représentation physique plus adaptée pour l'élément multivalué en conservant tous les avantages d'accès lorqu'il est monovalué. En effet, le changement d'un élément feuille monovalué à un élément feuille multivalué est facilité.

A titre d'exemple, le formulaire (personne) spécifié dans la figure 2.15 est illustré par la figure 2.17 en utilisant les valeurs nulles et la valeur spéciale **vm** introduites dans notre approche.

```
[Formpersonne (
 NOM:     String,
 PRENOM:  String,
 SS#      entier,
 POSITION MILITAIRE: String | dne,
 MARIE:   String,
 ENFANTS: String | noseq | vm,
 NOM_DE_JEUNE_FILLE: String | dne,
 SEXE: String)]
```

Fig 2.17 Type 2-formulaire "personne" avec les valeurs nulles

2.11 Représentation interne

Un autre problème posé lorsqu'on manipule des valeurs nulles est celui de trouver une représentation interne adéquate pour ces valeurs. En effet, le codage de ces valeurs doit être différent de toute valeur numérique ou chaîne de caractères qui peut prendre un attribut [DA82]. Dans notre cas, comme nous accédons aux éléments logiques par des pointeurs (identificateur, déplacement ou ordre de répétition), une solution peut être appliquée en ajoutant une interprétation à des valeurs déjà existantes et qui ne peuvent pas être une occurrence d'un attribut : par exemple, les valeurs négatives pour l'attribut ordre de répétition.

2.12 Extension de l'algèbre associée au modèle

Un autre aspect du problème du traitement des valeurs nulles est celui de l'extension de l'algèbre. Ceci peut entraîner la définition d'une logique à plusieurs valeurs (vrai, faux, noseq, noseqniv, niseq, dne, ni et unk), la redéfinition d'opérateurs logiques (ET, OU, NON) et ensemblistes (appartenance, inclusion) et la création de nouveaux opérateurs (par exemple, **isnoseqniv** dans notre cas). La figure 2.18 illustre la redéfinition des opérateurs ET et OU, et la définition du nouvel opérateur **isni** tout en prenant en considération une nouvelle

logique à trois valeurs. En effet, cette logique ajoute la valeur **ni** aux valeurs **vrai** et **faux**. Pour les opérateurs ET et OU, nous n'avons pas besoin de reconnaître séparément les valeurs nulles, de cette manière nous appelons l'ensemble NULL = {**ni, dne, unk, noseq, niseq, noseqniv**}. Ceci signifie, que dans une table, NULL indique la présence d'une de ces valeurs.

ET	Vrai	Faux	NULL
Vrai	Vrai	Faux	ni
Faux	Faux	Faux	Faux
NULL	ni	Faux	ni

OU	Vrai	Faux	NULL
Vrai	Vrai	Vrai	Vrai
Faux	Vrai	Faux	ni
NULL	Vrai	ni	ni

	Vrai	Faux	ni	NULL - {ni}
isni	ni	ni	Vrai	ni

Fig. 2.18 Logique à trois valeurs

De la même manière il faudrait redéfinir tous les autres opérateurs tels qu'égalité, inégalité, inclusion et appartenance si on passe à une logique à trois ou plus de valeurs. Néanmoins, ceci exige une formalisation qui s'éloigne de l'objectif principal de ce travail. En outre, pour traiter les structures logiques des documents nous pensons que la logique conventionnelle à deux valeurs est parfaitement suffisante. Par exemple, le nouvel opérateur **isni** peut prendre en considération toutes les valeurs nulles introduites dans l'algèbre sans qu'il soit pour autant nécessaire de prendre le résultat dans une logique à plus de deux valeurs (voir figure 2.19).

	Vrai	Faux	ni	NULL - {ni}
isni	Faux	Faux	Vrai	Faux

Fig. 2.19 Logique à deux valeurs.

2.13 Conclusions

Nous avons présenté les aspects pratiques d'une algèbre pour la modélisation des documents de bureau, les opérations de restructuration et les opérations d'accès. Il faut préciser que les opérations de restructuration n'ont de sens que si nous sommes dans un environnement partiellement ouvert ou fermé. Une première originalité de notre approche est la spécification du type atomique MULT qui permet de traiter un sous-arbre comme un élément feuille. Nous avons prévu, du point de vue de modélisation, le traitement des aspects multimédia mais aucun opérateur spécifique aux images, son ou graphiques ont été définis. En effet, on stocke les éléments multimédia et on les rend. La recherche par le contenu n'est possible que sur le contenu textuel (type TEXT). Une autre contribution de notre approche est la spécification d'un modèle simple et puissant utilisant seulement deux constructeurs : constructeur d'élément atomique et constructeur d'élément composé. Une règle générale du modèle dit que tous les éléments feuilles de la structure hiérarchique sont des éléments atomiques et tous les éléments intermédiaires sont des éléments composés. Ceci est très utile lors de l'indexation des éléments logiques. Finalement, une dernière originalité de cette algèbre est l'introduction de la valeur nulle "**noseqniv**" pour augmenter la souplesse de traitement de la structure logique ainsi que l'utilisation de la valeur spéciale "**vm**" pour faciliter l'indexation et le stockage des éléments atomiques multivalués qui sont en réalité monovalués.

CHAPITRE 3

STRUCTURES LOGIQUES DES DOCUMENTS

3

STRUCTURES LOGIQUES DES DOCUMENTS

3.1 Introduction

D'après Andre, Furuta et Quint dans [AFQ88] le traitement de documents devient l'un des emplois les plus courants de l'ordinateur. Les systèmes d'édition de texte, d'hypertexte et de Gestion de Base de Données Documents (SGBDD) connaissent actuellement un développement important. Tous ces systèmes manipulent des documents mais donnent des degrés différents d'importance aux fonctionnalités nécessaires pour leur traitement. La table 3.1 montre les degrés d'importance (faible, moyen et fort) attribués à certaines fonctionnalités pour ces systèmes.

Dans ce chapitre, nous décrivons des contraintes permettant de déclarer les éléments de structuration et de typage d'une structure logique générique de documents. Le but principal est la proposition d'un mécanisme de typage basé sur la structure logique déclarée. Cette même structure logique est prise en considération pour les SGBDD afin de faciliter la mise en place des fonctionnalités telles qu'échange, classement, stockage et accès. Il est vrai que la structure logique de documents ne concerne pas uniquement les SGBDD mais aussi les systèmes d'édition de documents et hyperdocuments : on observe de plus en plus une tendance de la part de ces systèmes vers l'utilisation d'une structure destinée à enrichir des fonctionnalités telles que création et lecture.

Table 3.1 Degrés d'importance des fonctionnalités

Fonctionnalités	Systèmes de traitement de documents		
	Edition de documents	Hyperdocuments	SGBDD
Création et écriture	fort	fort	faible
Présentation	fort si sur papier	fort si sur écran	faible
Lecture	fort	faible	faible
Echange	faible	faible	fort
Classement	moyen	moyen	fort
Stockage	moyen	moyen	fort
Accès	faible	faible	fort
Intégrité	faible	faible	fort
Sécurité/confidentialité	faible	faible	fort
Historiques/versions	faible	moyen	fort

Nous nous intéressons à l'échange de documents avec leurs structures logiques. D'après Horak [Hor85], ceci devient fondamental lorsque cet échange doit être réalisé entre des outils coopérant dans un environnement de travail de bureau. Deux aspects sont importants lors de l'échange : (i) la transmission du document avec sa structure

logique et (ii) l'acceptation de ce document par le système. Le premier aspect peut être résolu en adoptant un standard d'échange tel que SGML, ODA, etc. Le deuxième aspect peut être résolu par l'adoption d'un mécanisme de typage des documents. Nous proposons trois mécanismes de typage de documents : fort, faible et inexistant. Ces mécanismes permettre d'adapter le système à un des environnements suivants : fermé, partiellement ouvert et ouvert.

La deuxième fonctionnalité qui nous intéresse est celle du classement de documents dans un SGBDD basé sur la structure logique générique déclarée. Pour ceci, on utilise les mêmes mécanismes de typage pour vérifier si la structure du document est conforme au schéma associé à la sous-classe de destination.

Le stockage des documents est une des fonctionnalités les plus importantes dans un SGBDD. Nous pensons que les propriétés structurelles des documents peuvent aussi faciliter cette tâche. Elles sont présentés dans le prochain chapitre.

Enfin, la dernière fonctionnalité que nous étudions ici est l'accès aux documents. La prise en considération des structures logiques semble intéressante non seulement pour l'accès lors d'une simple consultation mais aussi pour la recherche intelligente d'informations sur les documents de la base.

Par exemple, dans le cas du stockage de lettres composées d'un en-tête comprenant les noms de l'expéditeur et du destinataire, un corps et une conclusion, il est clair que cet aspect est indispensable pour répondre à des consultations telles que: <<Quel est le nom de l'expéditeur de la lettre qui traite du prix des ordinateurs?>>.

De plus, dans le domaine de la recherche intelligente d'informations sur documents (recherche documentaire), il est particulièrement souhaitable de connaître au moins quelques éléments de la structure logique de ces documents. Par exemple, la recherche d'un article dans les actes d'une conférence nécessite, au minimum, la liste des auteurs des articles concernés pour qu'elle puisse répondre à la question du type: <<Quels sont les articles écrits par José?>>.

Dans la suite de ce chapitre nous définissons plusieurs possibilités de traitement de la structure logique des documents dans un SGBDD. Nous commençons par spécifier le problème du choix des éléments logiques ou unités documentaires qui présente des implications sur l'accès aux documents : les besoins issus des types d'accès nécessaires à la manipulation des documents remettent souvent en cause la structure dégagée par ce choix. Puis, nous montrons la séparation des aspects logiques et de présentation de la structure des documents. Ensuite, nous proposons trois mécanismes de typage de documents basés sur la structure logique générique de documents : typage fort, typage faible et typage inexistant. Enfin, nous insistons sur la description des contraintes permettant de définir d'une manière controlée les structures logiques génériques de documents.

3.2 Choix des unités documentaires

Le choix des unités documentaires et leurs subdivisions entraînent des conséquences directes sur les possibilités ultérieures de traitement des documents. Les besoins issus de la manipulation des documents stockés dans la base remettent souvent en cause les structures qui se dégagent d'un tel choix, c'est pourquoi nous nous intéressons à ce problème.

Pour le créateur d'un document se pose toujours le problème délicat de la définition des unités logiques qui doivent composer son document. La définition des critères permettant d'identifier ces unités documentaires est un problème en ouvert. Ces critères sont tellement variés que des aspects subjectifs comme l'habitude, la coutume, la langue, la réglementation de la documentation d'une entreprise et même la mode peuvent être pris.

La subdivision des unités documentaires est un autre problème. Il faut bien constater qu'il n'existe pratiquement pas de critères rationnels permettant de répondre a priori à la question du type << Faut-il ou non subdiviser tel ou tel chapitre en sous-chapitres ? >>.

Compte tenu de ces problèmes, nous sommes persuadés de la nécessité de mécanismes de restructuration pour adapter la structure logique des documents aux besoins des applications. Les opérations de restructuration présentées par l'algèbre DSB vont dans ce même sens. Par exemple, le choix a priori des unités logiques nécessaires à la manipulation linguistique d'un document n'est pas évident. En effet, dans le domaine linguistique, on constate la nécessité d'une structure très fine des documents pour l'analyse grammaticale et sémantique indispensable au traitement des problèmes d'accord.

De plus, l'aspect dynamique de la création et de la lecture d'un document avec des renvois, des hésitations, des changements de structure, etc souligne la nécessité d'une restructuration a posteriori.

Compte tenu de ces problèmes, nous sommes persuadés de la nécessité de mécanismes de restructuration pour adapter la structure logique des documents aux besoins des applications. Les opérations de restructuration présentées par l'algèbre DSB vont dans ce même sens. Ainsi, notre approche permet a partir d'un embryon de structure et de

la structure codé dans le document stocké l'obtention d'une structure en fonction des besoins de l'utilisateur. Ceci n'entraîne pas la mise à jour du contenu du document mais seulement une restructuration du schéma de la base.

3.2.1 Niveau logique et problèmes de structuration

Le niveau logique des documents traités dans ce travail est divisé en deux sous-niveaux : logique générique et logique spécifique [AP89]. Le sous-niveau logique générique contient toutes les règles qui définissent la structure logique générique d'une classe de documents. En effet, la structure logique générique est composée d'unités logiques ou unités documentaires telles que des paragraphes, des sections, des introductions, des chapitres, des notes, des préfaces, des avant-propos, des titres, des références, etc. Ces unités permettent de structurer logiquement un document non seulement pour sa production mais aussi pour sa manipulation ultérieure.

Le sous-niveau logique spécifique contient des informations d'une occurrence du document. Ainsi, la numérotation des unités documentaires avec la prise en compte de la répétition de ces unités est connue : chapitre 1, chapitre 2, section 2 du chapitre 1, etc.

Il ne faut pas confondre la numérotation des unités documentaires au niveau logique générique avec celle qui est effectuée au niveau logique spécifique. La numérotation au niveau logique générique correspond en réalité aux labels numériques de l'algèbre définie dans le chapitre 2. Cette numérotation permet d'identifier les unités documentaires au niveau conceptuel mais ne prend pas en compte la répétition des unités. Par exemple, nous présentons ci-dessous une structure logique générique et une structure logique spécifique d'un livre avec des numérotations possibles :

Structure logique générique	Structure logique spécifique
1 Introduction	Introduction
2 Chapitres	1 Chapitre
3 Section	1.1 Section
4 Conclusion	2 Chapitre
	Conclusion

3.2.2 Niveau de présentation et reconnaissance du niveau logique

Pour traiter les aspects de présentation des documents nous distinguons deux structures de présentation : structure de présentation générique et structure de présentation spécifique. Ces structures contiennent toutes les informations pour la présentation des documents sur un support physique : écran ou feuilles de papier.

La présentation physique des documents sur un écran ou sur des feuilles de papier met en évidence l'organisation logique des documents. Ainsi, l'utilisation de caractères gras, le changement de police et de corps des caractères, l'interligne, le changement de marge et le saut de page sont de détails de présentation qui servent à faire ressortir l'organisation du document imposée par l'auteur.

Ces détails peuvent être utilisés pour déduire la structure logique des documents. De plus, en certains cas, la manipulation de la structure de présentation comme structure logique peut être envisagée. Par exemple, un document ODA/ODIF au niveau de conformance image, où nous ne connaissons que la structure de présentation, le fait de pouvoir répondre à la question << Quelles sont les pages qui contiennent l'expression "base de documents" ?>> nous paraît intéressant.

3.3 Mécanismes de typage des documents

Nous proposons les trois mécanismes de typage ci-dessous afin de faciliter principalement les fonctionnalités d'échange, d'acceptation et de classement des documents. Ces mécanismes sont basés sur la vérification de la conformité aux contraintes décrites dans la section suivante.

- Typage fort. Il n'accepte que les documents qui contiennent les éléments déclarés. Aucun document ne sera accepté s'il contient des éléments non-déclarés. Ce mécanisme entraîne une stratégie de système fermé. Le mécanisme de typage fort est aussi appelé mécanisme fortement typé.

- Typage faible. Ce mécanisme débouche sur une stratégie de système partiellement ouvert. Ici l'acceptation du document est plus souple. Un document peut être accepté même s'il contient des éléments non-déclarés. L'important est la présence des éléments déclarés comme obligatoires.

- Typage inexistant. Ici les conformités aux contraintes ne peuvent pas être réalisées parce qu'il n'y a aucune contrainte déclarée. La stratégie de système ouvert est donc la seule possible. Ce mécanisme accepte tous les documents quelle que soit leur structure logique.

Les mécanismes de typage cités ci-dessus se traduisent, selon les contraintes indiquées de la structure logique des documents, en avantages et/ou en inconvénients.

3.4 Contraintes sur la Structure Logique Générique des Documents

L'objectif de cette section est de définir les contraintes permettant de déterminer tous les éléments de structuration et de typage succeptibles de faciliter l'échange, le classement, le stockage et l'accès aux documents.

Comme nous l'avons déjà défini dans l'algebre DSB, les éléments de la structure logique des documents manipulés ici sont de deux catégories : les éléments composés et les éléments de base ou atomiques. Le type de données d'un élément de base peut être un type de base conventionnel (Entier, Chaîne, Caractère, etc) ou un type de base multimédia (Mult, Texte, Image, Graphique ou Son). Les éléments composés (Comp) contiennent des éléments qui, à leur tour, peuvent être composés d'autres éléments. Il faut rappeler que tous les types de données définis dans l'algèbre DSB contiennent comme valeurs possibles d'occurrences les valeurs nulles et la valeur multiple (**vm**).

Une contrainte d'intégrité sur un élément logique peut non seulement maintenir l'intégrité structurelles des éléments correspondant aux différents types de documents mais aussi avoir des implications sur les fonctionnalités de stockage et d'accès auxquelles nous nous intéressons dans ce travail. Au fur et à mesure que nous décrivons les contraintes que nous pensons intéressantes pour traiter les documents, nous montrons les implications sur les mécanismes de typage/acceptation des documents ainsi que les implications sur les requêtes disponibles pour la manipulation de ces documents.

Un classement des contraintes peut être fait pour faciliter leur manipulation. Nous proposons un classement prenant en compte les

quatre aspects suivants : existence, type, cardinalité et ordre.

Dans ce travail, nous nous limitons aux contraintes statiques [BP83] qui sont des contraintes indépendantes, pour leur vérification, de l'état antérieur de la base. Par exemple, la contrainte de type : "titre : chaîne" est toujours valable. Nous ne traitons que superficiellement des contraintes dynamiques. Ces contraintes dépendent, en cas de modification de la base, du contenu de l'état antérieur de la base pour leur vérification. Par exemple, "la table des matières ne peut qu'augmenter".

Nous utilisons la structure générique de guides de musée (illustrée par la figure 3.1) pour fabriquer des exemples sur les contraintes décrites par la suite.

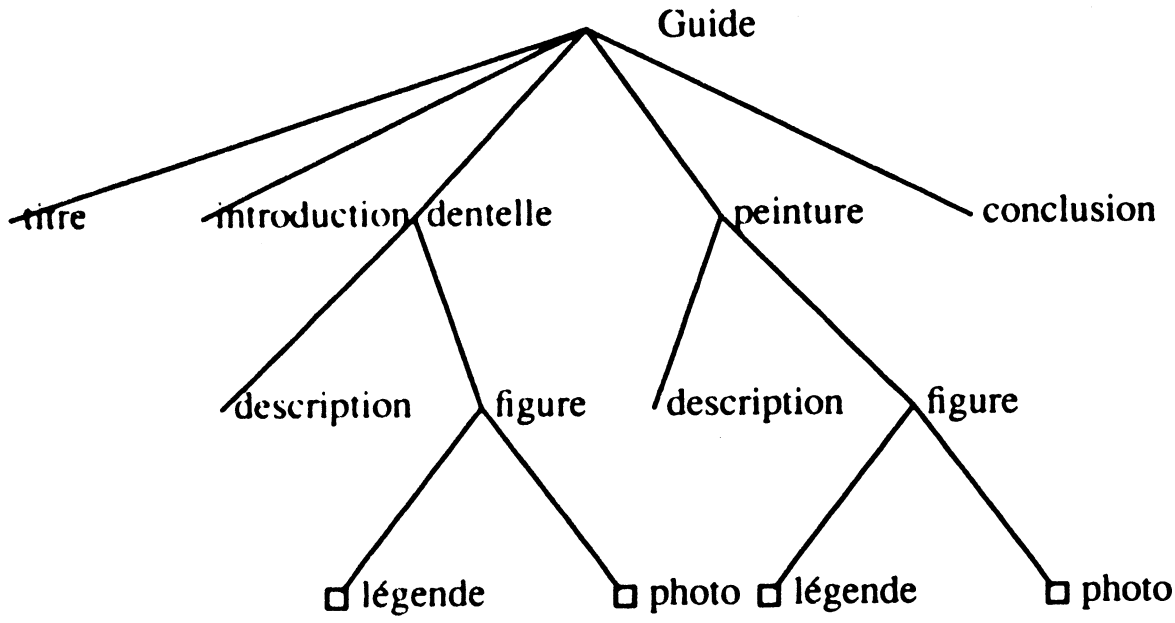

Fig. 3.1: Structure logique générique de guide de musée

3.4.1 Contraintes d'existence

Ces contraintes indiquent l'existence d'éléments liés aux aspects de dépendance ou d'indépendance de la hiérarchie, obligatoire ou facultative, alternative et interdite. Ces contraintes permettent une souplesse de spécification des structures logiques des documents.

Nous avons dégagé 6 contraintes d'existence:

- Contrainte d'existence autonome.
- Contrainte d'existence hiérarchique.
- Contrainte d'existence obligatoire.
- Contrainte d'existence facultative.
- Contrainte d'existence alternative.
- Contrainte d'existence interdite.

Il faut remarquer que toutes les contraintes d'existence peuvent être désignées par un des valeurs nulles spécifié dans le chapitre précédent.

3.4.1.1 Contrainte d'existence autonome

Cette contrainte indique l'existence d'un élément indépendamment de sa position dans la hiérarchie de la structure logique. Par exemple, la déclaration ci-dessous, indique que l'élément "légende" existe dans un guide de musée indépendamment de sa position hiérarchique.

(guide(..légende))

La structure logique générique contenue dans cette déclaration peut être représentée par la figure 3.2. Nous utilisons une ligne

pointillée pour indiquer l'abstraction du chemin logique hiérarchique.

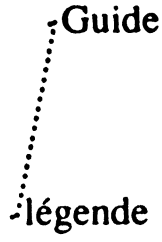

Fig. 3.2: Structure logique générique

Il est alors possible de poser la question <<Quelles sont les légendes qui parlent de la "Joconde"?>> sans qu'il soit pour autant nécessaire d'expliciter ou de manipuler complètement la hiérarchie (guide.peinture.figure.légende ou guide.dentelle.figure.légende).

Néanmoins, on remarque qu'il n'est pas possible de répondre à une question du type : <<Quelle est la photo de la peinture où la légende parle de la "Joconde" ?>>. C'est-à-dire qu'il n'est pas possible de répondre à une question qui utilise une référence hiérarchique parce que l'élément est déclaré avec l'existence autonome de la hiérarchie.

Un aspect qu'il convient de mettre en évidence, dans ce cas précis, est celui du changement de la cardinalité d'un élément qui passe de monovalué à multivaluée. Par exemple, l'élément "légende" est à l'origine monovaluée pour chaque élément "dentelle" ou "peinture", cependant il devient multivalué s'il est traité indépendamment de la hiérarchie.

La contrainte d'existence autonome permet de traiter facilement les éléments de la structure logique parce qu'elle offre une manipulation aisée d'un élément logique sans qu'il soit nécessaire de connaître le chemin logique auquel celui-ci appartient. Un des

avantages offerts par cette méthode est qu'elle laisse la possibilité d'effectuer la recherche par le contenu en ne prenant en considération que les éléments logiques concernés.

Par rapport à la hiérarchie de spécialisation, pour éviter des confusions, cette contrainte est contradictoire et incohérente avec la contrainte hiérarchique décrite dans la section suivante. C'est-à-dire que, dans une hiérarchie de spécialisation, un élément ne peut pas être déclaré, à la fois, comme indépendant et dépendant de la structure logique.

3.4.1.2 Contrainte d'existence hiérarchique

Cette contrainte diffère de la précédente dans le sens qu'elle reflète une connaissance hiérarchique des éléments de la structure des documents. Par exemple, la déclaration ci-dessous donne la hiérarchie complète de l'élément "peinture" du document Guide.

(Guide(peinture(description,figure(légende,photo)))

La hiérarchie déclarée de l'élément "peinture" est visible par une branche de l'arbre illustré par la figure 3.1.

Maintenant, nous pouvons poser des requêtes en exploitant le chemin logique hiérarchique des éléments. Par exemple, <<Quelle est la photo de la peinture où la légende parle de la "Joconde" ?>>.

L'utilisation combinée des contraintes autonome et hiérarchique permet d'exprimer d'autres situations intéressantes pour les documents :

- Existence autonome d'un élément logique composé. En certains cas, la déclaration d'existence d'un élément composé

indépendamment de la hiérarchie de la structure logique est utile. Supposons, par exemple, qu'il existe deux types de livres à stocker dans la base : (i) livre composé directement de chapitres et (ii) livre organisé en parties composées de chapitres. On observe, dans ce cas précis, que l'élément chapitre peut se trouver soit dans le premier niveau soit dans le deuxième niveau de la hiérarchie. L'élément chapitre que nous traitons est composé d'un titre et d'une section.

Exemple :

(livre..(chapitre(titre,section))

- Existence d'éléments récursifs. Un élément récursif entraîne un cycle sur le schéma. En profitant de l'exemple précédent, nous pouvons supposer que la section est composée de sections qui, à leur tour, peuvent être composées de nouvelles sections. L'exemple ci-dessous montre la récursivité de l'élément section.

Exemple :

(chapitre(titre,..section(section)))

L'indication d'une profondeur fixe pour les éléments récursifs s'adapte bien à l'environnement de bureau car on ne manipule pas des arbres très profonds. Un exemple de cette indication est l'énoncé ci-dessous limitant la section à trois niveaux de récursivité :

Exemple :

(..section(section[3]))

3.4.1.3 Contrainte d'existence obligatoire

La contrainte d'existence obligatoire permet d'assurer une certaine standardisation des documents manipulés par le système, en spécifiant la présence obligatoire de certains éléments logiques. Cette contrainte peut porter sur un élément d'une classe de documents ou sur un élément commun à tous les documents stockés dans un SGBDD. Les implications de l'utilisation de cette contrainte avec les mécanismes de typage fort et faible sont présentées dans les dernières sections de ce chapitre.

Nous pouvons exprimer l'existence obligatoire d'un élément logique à travers d'autres contraintes telles que celles de cardinalité (cf. section 3.4.3) et de type (cf. section 3.4.2).

Exemples :

(Guide(titre(1,1))
(Guide(titre string not null))

Le premier énoncé indique l'existence obligatoire d'un l'élément titre en déclarant sa cardinalité (il existe au minimum un titre et au maximum un titre). Le deuxième exemple indique l'existence obligatoire d'un titre en déclarant qu'à toutes ses occurrences sont affectées les valeurs du type chaîne différentes des valleurs nulles.

On peut déclarer un élément obligatoire en le prefixant par un "*".

Exemple :
(Guide(*titre))

Le symbole "*" signifie que l'élément "titre" est obligatoire pour le document Guide.

3.4.1.4 Contrainte d'existence facultative

La contrainte d'existence facultative sur un élément présente trois avantages : (i) assurer d'abord un mécanisme d'accès à cet élément,(ii) permettre ensuite une analyse sémantique plus complète d'une requête sur cet élément et (iii) enfin, conférer une certaine souplesse à la spécification de la structure logique générique vis-à-vis de cet élément.

Exemples :

(Guide(introduction(0,1))
(Guide(introduction text))

D'une manière semblable à celle utilisée pour exprimer la contrainte d'existence obligatoire d'un élément logique, nous pouvons exprimer aussi la contrainte d'existence facultative par le biais des contraintes de cardinalité et de type. Tous les types de données predéfinis contiennent par défaut leurs valeurs nulles et la valeur **vm**.

En effet, du point de vue pratique, cette contrainte est plutôt une déclaration d'existence, principalement lorsqu'elle est utilisée avec un mécanisme de typage faible. Néanmoins, une contrainte de ce type est nécessaire si on utilise un mécanisme de typage fort parce que ce mécanisme exige la reconnaissance de tous les éléments de la structure logique du document à accepter.

3.4.1.5 Contrainte d'existence interdite

Cette contrainte permet de classer des documents vis-à-vis de l'interdiction d'existence d'un élément. Elle peut porter sur un élément d'une sous-classe qui a été déclaré comme facultatif dans la classe hiérarchiquement supérieure. Par exemple, dans la classe

"lettre", la signature est déclarée facultative et dans la sous-classe "LettreNonSignée" (spécialisation) elle est déclarée comme interdite.

Exemple :

(LettreNonSignée(signature null))

Cet énoncé indique que toutes les occurrences de signature dans la classe "LettreNonSignée" sont affectées d'une valeur nulle prévue par l'algèbre DSB.

3.4.1.6 Contrainte d'existence alternative

La contrainte d'existence alternative confère une certaine souplesse à la spécification de la structure logique d'un type de document. C'est une façon élégante de traiter les entités polymorphes. L'utilisation de cette contrainte, avec les contraintes d'existence dépendante ou indépendante de la hiérarchie sur les éléments, permet de spécifier des structures génériques plus larges qui peuvent accepter des documents avec des structures variées. Par exemple: l'article d'un congrès peut être un article complet ou un résumé et présenter par conséquent des structures différentes.

3.4.2 Contraintes de type

Les types des données de base des éléments sont les types conventionnels (Entier, Chaîne, Caractère, etc) ou les types multimédia (Texte, Graphique, Image, Son et Multimédia). Cette contrainte est importante dans la mesure où elle permet de maintenir l'intégrité sémantique des éléments correspondant aux différents types de données qui peuvent composer le document. Il semble intéressant de pouvoir spécifier par exemple, que l'"abstract" d'un article ne contient que le type de base Text ou que l'élément "photo" ne contient

que le type de base Image.

Exemple :

article(abstract text)
(photo image)

C'est à ce niveau qu'il faut décrire des contraintes de valeurs nulles. Nous avons identifié six valeurs (cf. section 2.10) : trois d'entre-elles sont liées aux éléments de base ("dne" - information non/existente, "unk" - valeur inconnue et "ni" - aucune information disponible) et les trois autres sont liées aux éléments composés ("noseq" - séquence inexistente, "noseqniv" - séquence inexistente dans le niveau et "niseq" - séquence inconnue). Il faut remarquer que la contrainte d'existence facultative peut être remplacée par la contrainte de type avec les valeurs nulles. Par exemple, la déclaration ci-dessous indique que l'élément "dentelle" peut avoir la valeur nulle "noseq". En réalité, ceci signifie que l'élément "dentelle" est facultatif.

(Guide(dentelle noseq))

La sémantiq ssociée aux valeurs nulles introdites dans notre approche permet leur utilisation depuis la conception logique jusqu'à la conception physique.

Il faut rappeler que tous les types de données définis dans l'algèbre DSB contiennent, par défaut, comme valeurs possibles d'occurrences, les valeurs nulles et la valeur multiple (**vm**). Ceci signifie que l'élément auteur de l'énoncé suivant peut être affecté d'une des valeurs nulles de base (nullbase = {**ni**, **dne**, **unk**}) ou de la valeur spéciale (**vm**).

Exemple :

(Guide(auteur string))

Par conséquent, avec cet énoncé, l'élément auteur est par défaut facultatif et multivalué. L'implication directe de l'adoption de cette convention est que la négation de l'existence d'occurrences avec les valeurs nulles équivaut à indiquer la contrainte d'existence obligatoire.

Exemple :

(Guide(auteur string not null))

En ce qui concerne l'accès aux documents il peut faciliter la solution de rêquetes telles que << Quels sont les guides qui n'ont pas d'auteur ? >>.

3.4.3 Contraintes de cardinalité

Cette contrainte permet la vérification de l'élément pour ce qui concerne les caractéristiques monovaluées et multivaluées. Cette contrainte peut être exprimée par le biais de la spécification d'une limite inférieure et supérieure.

Exemple :

Guide(auteur(0,1) string))

Néanmoins, nous préférons exprimer cette contrainte par le biais de la valeur multiple "**vm**" définie dans l'algèbre DSB.

Exemple :

Guide(auteur string not **vm**))

Les deux énoncés précédents indiquent que l'élément auteur de Guide est toujours monovalué. Ainsi, indiquer qu'un élément peut être affecté soit d'aucune valeur soit d'une valeur au maximum équivaut à indiquer que cet élément n'est jamais multivalué.

Nous proposons la confirmation, la négation ou l'indication d'une probabilité floue de la valeur **vm** pour indiquer non seulement la contrainte de cardinalité mais pour exprimer également des situations extrêmement différentes qui ont des implications considérables sur les méthodes de stockage :

Exemples :

- (Guide(peinture **vm**))
  La confirmation de la valeur **vm** dans cet énoncé indique que l'élément peinture est multivalué pour toutes les occurrences des documents de la classe Guide.

- (Guide(peinture not **vm**))
  La négation de la valeur **vm** par cet énoncé indique que l'élément peinture est toujours monovalué pour toutes les occurrences de Guide. Ceci est une manière d'exprimer la contrainte d'existence interdite de l'élément peinture multivalué.

- (Guide(peinture + **vm**))
  Cet énoncé utilise le symbole "+" pour indiquer qu'en général l'élément peinture est multivalué. Ceci équivaut à indiquer une probabilité floue des occurrences de Guide vis-à-vis de l'élément figure.

- (Guide(peinture - **vm**))
  Cet énoncé utilise le symbole "-" pour exprimer qu'en général l'élément peinture n'est pas multivalué. En effet, il indique une

faible probabilité floue de l'occurrence de l'élément multivalué.

En ce qui concerne les implications opérationnelles la valeur **vm** permet de répondre facilement aux questions telles que << Quelles sont les guides qui ont plusieurs auteurs ? >>.

Nous verrons dans le chapitre suivant que ces informations peuvent induire le choix d'une structure bien adaptée aux éléments multivalués et/ou monovalués.

3.4.4 Contraintes d'ordre

L'ordre des composants d'un document est une propriété fondamentale lorsqu'on gère des documents. Il y a, par exemple, un ordre significatif et cohérent entre les chapitres d'un livre, entre les sections d'un rapport et, dans la plupart des cas, l'ordre entre les auteurs est aussi significatif. Nous avons distingué deux types d'ordres entre les éléments de la structure logique d'un document : l'ordre sur les occurrences et l'ordre structurel. Le premier est l'ordre d'un élément répétitif, l'ordre des chapitres d'un livre, par exemple. Le deuxième est l'ordre des composants d'un élément. Par exemple, l'ordre des composantes d'un livre (introduction, chapitres, conclusion et annexe). Nous pensons que le deuxième type d'ordre n'est pas utile aux requêtes de manipulation qui nous intéressent. Par contre, l'ordre des éléments multivalués est nécessaire car il permet de mieux répondre aux requêtes telles que << Quel est l'ordre du chapitre du livre qui traite sur SGBDD ? >>. Dans ce cas précis, une réponse possible est "chapitre 4".

Nous allons maintenant décrire le résultat de notre étude analytique, en termes d'avantages et d'inconvénients, sur l'utilisation des mécanismes de typage (cf. section 3.3) avec les contraintes

d'existence autonome et d'existence hiérarchique.

3.4.5 Typage fort et contrainte d'existence autonome

L'utilisation de la contrainte d'existence autonome avec le mécanisme de typage fort exige la connaissance de tous les éléments susceptibles d'appartenir à cette structure. Le manque de souplesse pour accepter des documents dont les éléments de leur structure logique ne sont pas totalement connus à l'avance constitue le principal inconvénient de ce mécanisme. C'est-à-dire qu'on ne peut pas travailler avec des documents sans connaître au moins tous les éléments composant leur structure logique.

Il est vrai que connaître tous les éléments d'une structure ne signifie pas pour autant connaître la hiérarchie entre ces éléments.

Ce mécanisme s'avère efficace pour les documents dont la structure logique est connue et figée (par exemple, des formulaires) mais pose des difficultés pour les documents qui n'ont pas une structure régulière définie (par exemple, des lettres).

Il faut remarquer que cette contrainte d'existence ne prend pas en compte la hiérarchie des éléments ni les composants d'un élément composé. De ce point de vue, elle apporte une certaine souplesse.

Néanmoins, un document peut être refusé s'il contient au moins un élément non-déclaré. Par exemple, la lettre citée ci-dessous est refusée parce qu'elle a été déclarée comme composée obligatoirement des éléments "destinataire", "expéditeur" et "signature" et, facultativement, des éléments "introduction" et "conclusion" mais l'élément "corps", qui n'a pas été déclaré, existe égalememt.

- Lettre(destinataire,expéditeur,signature,corps)

Un autre motif de refus est l'absence d'un des éléments déclarés comme obligatoires. Par exemple, les lettres suivantes:

- Lettre(destinataire,expéditeur,conclusion)
- Lettre(destinataire,signature)

AVANTAGES:

a. Il dispense de la connaissance de la hiérarchie entre les éléments.

b. Il permet l'application de l'approche système fortement typé sans qu'il soit nécessaire de connaître la hiérarchie entre les composants du document. Ceci confère une certaine souplesse et satisfait certainement les besoins d'applications qui manipulent des documents standards et qui n'acceptent que des documents complets. Par exemple, certains types de lettres, des formulaires, etc.

c. L'analyse sémantique des commandes de manipulation peut être faite d'une façon plus complète, parce que tous les éléments susceptibles d'appartenir à la structure sont déclarés. Mais il est vrai que peuvent subsister des ambiguités comme "titre de chapitre" et "titre de section" pour lesquels il n'est pas possible de savoir si le titre peut appartenir à deux éléments distincts.

d. Il est possible d'indiquer si un élément déclaré comme facultatif existe avec une valeur connue, ou existe sans toutefois que sa valeur soit connue, ou simplement n'existe pas. Une méthode concernant une telle indication repose sur l'utilisation de différents valeurs nulles comme celles definies dans l'algèbre

DSB.

INCONVENIENTS:

a. Il exige la connaissance de tous les éléments susceptibles d'appartenir au document.

b. La structure hiérarchique entre les éléments n'est pas connue. Par conséquent on ne peut pas répondre aux requêtes relatives à la strucure arborescente. Par exemple : <<quels sont les titres des chapitres d'un livre ?>>.

c. Il n'est pas possible d'identifier les composants d'un élément composé.

3.4.6 Typage fort et contrainte d'existence hiérarchique

L'application du mécanisme de typage fort avec des contraintes dépendantes de la structure logique est restrictive parce qu'il faut connaître à l'avance la structure logique. C'est-à-dire qu'il faut connaître tous les éléments et leur position dans la structure hiérarchique. En effet, dans le monde réel, il est très difficile de connaître à l'avance la structure logique complète d'un document sauf si celui-ci est un formulaire et même un type de lettre destinée à une application spécifique.

AVANTAGES:

a. Il s'adapte bien au traitement des documents dont la structure logique est completement définie.

b. L'analyse sémantique des commandes de manipulation peut être faite d'une façon plus complète parce que tous les éléments possibles sont déclarés avec leur structure hiérarchique.

c. Toutes les requêtes sur la hiérarchie des éléments sont possibles.

INCONVENIENTS:

a. Il exige la connaissance de la structure logique complète. C'est-à-dire qu'il faut connaître tous les éléments susceptibles d'appartenir au document et pour chaque élément son niveau dans la structure hiérarchique et même ses composants.

3.4.7 Typage faible et contrainte d'existence autonome

Le mécanisme de typage faible est plus souple que le mécanisme de typage fort et il semble plus proche des situations réelles. En effet, les documents trouvés dans le monde réel ont des structures souvent complexes et dont il est difficile d'identifier la totalité des éléments au préalable. Il est, par exemple, facile de déduire qu'une lettre puisse normalement avoir les éléments "destinataire", "expéditeur" et "signature", par contre les autres éléments facultatifs, tels que "logotype", "introduction" et "conclusion" ne sont pas évidents. Certains éléments dépendent de l'application ou du type de lettre. Par exemple, une lettre d'information sur les prix d'un produit peut contenir une table de prix coûtants, des composants de ce produit ou même un graphique qui montre l'évolution du prix du produit au cours des 6 derniers mois.

A l'instar de la stratégie de système fortement typé, la stratégie de système faiblement typé peut refuser tout document dont il manque un des éléments déclarés comme obligatoires. Contrairement à la stratégie de système fortement typé, la stratégie de système faiblement typé peut accepter des documents dont les éléments ne sont pas tous connus.

Le mécanisme de typage faible entraîne un système partiellement ouvert. Un système faiblement typé avec des contraintes d'existence des éléments donne un degré de souplesse raisonnable en ce qui concerne la connaissance de la structure logique du document à traiter. Ici, on peut accepter des documents, même lorsqu'on en connait qu'un seul élément.

Par exemple, la déclaration ci-dessous permet l'acceptation par le SGBDD de tous les guides de musée qui contiennent les éléments "Introduction" et "peinture" indépendamment de leur position hiérarchique et de leurs composants.

(Guide,(..Introduction,..peinture))

AVANTAGES:

a. Il est simple et n'impose pas la déclaration complète de tous les éléments susceptibles d'appartenir à un type de document.

b. La liste d'éléments déclarés pourrait être utilisée pour une stratégie d'indexation efficace, en indiquant par exemple les éléments faisant fréquemment l'objet d'une recherche par le contenu.

c. Il est possible de savoir si un élément déclaré comme facultatif existe avec une valeur connue, ou existe sans que toutefois sa valeur soit connue, ou n'existe pas.

d. L'analyse sémantique des requêtes portant sur des éléments déclarés est faite de la même façon qu'en matière de stratégie de système fermé.

e. Le document peut être accepté sans qu'il soit nécessaire de connaître tous ses éléments composants.

INCONVENIENTS:

a. La structure hiérarchique entre les éléments n'est pas connue. Par conséquent on ne peut pas répondre aux requêtes relatives à la strucure arborescente. Par exemple : <<Quels sont les titres des chapitres du livre x ?>>.

b. Il est impossible d'effectuer l'analyse sémantique des requêtes portant sur des éléments facultatifs non-déclarés. Ceci n'empêche pas la possibilité de formuler des requêtes portant sur ces éléments non-déclarés.

c. Le mécanisme ne permet pas le traitement de la récursivité éventuelle des composants, comme dans le cas de sections composées de sections.

3.4.8 Typage faible et contrainte d'existence hiérarchique

La conformité aux contraintes sur la hiérarchie des éléments est plus souple en utilisant le mécanisme de typage faible parce qu'il n'est pas nécessaire de connaître tous les éléments ni la hiérarchie entre ceux-ci et qu'elle est plus riche dans la mesure où elle reflète une connaissance hiérarchique des éléments de la structure des documents. Par exemple, la notation indiquée ci-dessous entre parenthèses indique que toute lettre présente un en-tête (composé du nom de l'expéditeur, qui, à son tour, est composé d'un nom et d'un prénom), un corps composé de paragraphes et une conclusion.

Lettre(en-tête(expéditeur(nom,prénom),corps(paragraphe), conclusion(paragraphe))

AVANTAGES:

a. Il est possible de décrire la structure arborescente des éléments à manipuler.

b. Il n'est pas nécessaire de connaître tous les éléments de la structure logique ni même l'arborescence entre ces éléments pour pouvoir traiter ce document.

c. Il est possible de manipuler des éléments récursifs.

INCONVENIENTS:

a. Il est impossible d'effectuer l'analyse sémantique des requêtes portant sur des éléments facultatifs non-déclarés. Ceci n'empêche pas la possibilité de formuler des requêtes portant sur ces éléments non-déclarés.

3.4.9 Aucune contrainte sur la structure logique

S'il n'y a pas de contraintes sur la structure logique des documents à traiter, on ne peut pas utiliser un mécanisme de typage et par conséquent la seule stratégie applicable est la stratégie de système ouvert.

En ce qui concerne la fonctionnalité d'échange/acceptation de documents elle est toujours possible dans cet environnement ouvert. Néanmoins, nous constatons trois options pour la mise en place de cette fonctionnalité :

1. Acceptation du document sans la reconnaissance de sa structure logique. Cette option ne permet aucune requête portant sur les éléments logiques.

2. Acceptation du document avec la reconnaissance des éléments logiques indépendamment de la hiérarchie. Ici nous pouvons exporter tous les éléments reconnus comme éléments facultatifs au schéma du SGBDD permettant par la suite l'analyse sémantique des requêtes portant sur ces éléments. De toute manière, les requêtes sont acceptées indépendamment de leur analyse sémantique puisqu'il s'agit d'un système ouvert. La recherche portant sur les aspects hiérarchiques n'est pas possible.

3. Acceptation du document avec la reconnaissance de ses éléments logiques et la hiérachie entre eux. Cette option exige un traitement du document plus en détail au moment de son acceptation afin de reconnaître sa structure logique complète : les éléments logiques et la hiérarchie. Si le système est ouvert à plusieurs standards d'échange il faut qu'il sache desquels il s'agit. Un grand avantage de cette option est de permettre la recherche portant sur les aspects hiérarchiques du document tout en restant dans un système ouvert.

3.5 Conclusions

Dans ce chapitre, nous avons proposé 4 types de contraintes qui peuvent être appliquées aux éléments de la structure logique des documents : contrainte d'existence, contrainte de type de données, contrainte de cardinalité et contrainte d'ordre. Sous certains aspects ces contraintes rejoignent le traitement des valeurs nulles. La contrainte d'existence nous semble essentielle pour le traitement des documents parce qu'elle permet la définition de types souples de documents. En effet, la souplesse ou la rigidité de la structure logique peuvent être controlées par les mécanismes de typage décrits dans ce

chapitre : le typage fort, le typage faible ou typage inexistant. Ces mécanismes conduisent à la construction d'un des systèmes suivants : système fermé, système partiellement ouvert ou système ouvert. Notre contribution est l'obtention d'un système flexible pouvant offrir les trois types de systèmes.

Nous présentons les différentes méthodes de stockage et d'accès aux documents dans le chapitre suivant.

CHAPITRE 4

STOCKAGE ET ACCES AUX DOCUMENTS

4

STOCKAGE ET ACCES AUX DOCUMENTS

4.1 Introduction

Les méthodes de stockage et d'accès aux objets (données) d'une application sont directement dérivées du **schéma physique** de ces objets, ce schéma étant défini sur une structure d'accueil telle que fichiers classiques, bases de données relationnelles, hiérarchiques ou réseaux, mémoire centrale, etc.

L'obtention d'un schéma physique des objets est une tâche difficile, d'une part, parce qu'il n'existe actuellement pas de théories

ou de régles formelles permettant de déduire une bonne représentation des structures physiques et, d'autre part, parce que cette obtention est généralement élaborée par rapport à des paramètres approximatifs définis par le concepteur.

Certains de ces paramètres ont un degré d'influence tellement fort sur le schéma physique qu'une petite variation de l'estimation du paramètre concerné peut sérieusement mettre en cause le schéma physique obtenu. Par exemple, la fréquence d'utilisation des données joue un rôle fondamental dans la définition du schéma physique.

Pour la démarche de conception physique, lorsqu'on utilise comme structure d'accueil des fichiers classiques (en utilisant soit la couche de gestion des entrées/sorties, soit la couche gestion des fichiers du système d'exploitation), nous sommes obligés de traiter des détails plus physiques tels que facteur de blocage, taille des blocs, taux de débordement, etc. Dans le cas de l'utilisation d'une base de données comme structure d'accueil les détails traités sont plus proches du problème à résoudre : nous nous préoccupons des chemins d'accès privilégiés, du mode de placement des nuplets, des types d'objet à traiter, etc. Plus les objets traités sont complexes, plus les paramètres à prendre en considération sont nombreux.

Dans le cas spécifique du stockage et de l'accès aux objets complexes de type document, la démarche pour la conception physique est très importante car les documents présentent des caractéristiques très diverses. Il est par exemple difficile, parfois même impossible, de prévoir la taille d'un document au début de son écriture. Certains documents seront de grande taille, d'autres de taille variable (un même type de document peut avoir une page ou des milliers de pages). D'autres caractéristiques comme la structuration,

les aspects multimédia, etc font des documents des objets complexes concernés par tous les types de problèmes de stockage.

Nous tentons donc de dégager dans ce chapitre des méthodes de stockage et d'accès adaptées aux documents.

4.2 Caractéristiques des traitements de documents

Les caractéristiques des traitement des documents d'une base dépendent des applications qui manipuleront ces documents. Ces caractéristiques peuvent expliquer le choix d'une technique de stockage et l'adoption d'une technique d'accès. Par conséquent, au fur et à mesure que nous décrivons ces caractéristiques nous tentons de les associer aux techniques de stockage et d'accès les mieux adaptées. Nous classons ces traitements en **traitements globaux des documents** et **traitements des unités des documents.**

4.2.1 Traitements globaux des documents

Les traitements globaux des documents sont caractérisés par la manipulation de documents comme des objets atomiques. Nous allons maintenant distinguer quelques cas particuliers de ces types de traitements.

- **Manipulation de tous les documents de la base** : ce cas conduit à la création de mécanismes pour permettre d'accéder à tous les documents stockés dans la base. Une exigence imposée à un SGBD voulant supporter ce type de manipulation est de créer une super-classe contenant tous les documents et d'utiliser l'organisation séquentielle pour le schéma physique. L'accès séquentiel aux documents est le mieux adapté lorsqu'on manipule tous les documents de la base.

- **Manipulation de tous les documents d'une même classe** : beaucoup d'applications nécessitent la manipulation des documents d'une même classe. Par exemple, le traitement des lettres dans une application concernant le courrier d'un bureau entraîne la gestion de la spécialisation de documents.

  Cette caractéristique induit le choix d'une organisation qui facilite le regroupement de certaines données. Le regroupement physique des données souvent sollicitées simultanément est une bonne technique de stockage et d'accès.

  Le choix de l'organisation séquentielle avec des index permet le mode d'accès séquentiel indexé qui est adapté à ce type de manipulation.

  Un autre choix possible est celui d'une organisation arborescente. Cette organisation peut aussi offrir d'autres modes d'accès appropriés au regroupement par classe. Par exemple, le système IMS-DL1 implante les occurrences d'un arbre par contiguïté physique ou par pointeur, ce qui donne la possibilité d'utiliser l'un des modes d'accès suivants : HSAM, HISAM, HDAM, HIDAM. Nous appelons organisation arborescente des structures basées sur des arbres tels que B-trees, B*trees, etc. En général, cette organisation est bien adaptée aux fichiers dits inversés et indexés. On peut citer trois avantages de cette organisation : tout d'abord elle est relativement bien connue et déjà utilisée par les SGBD commercialisés, elle est ensuite assez performante et permet finalement un accès séquentiel trié. Cette approche présente certains inconvénients pour la gestion des index tels que débordement, réorganisation d'arbre, etc. Des exemples basés sur l'organisation arborescente sont décrits en [BM72,

Knu73, Com79, TF82].

- **L'échange de documents** : favoriser l'échange de documents entre des systèmes ouverts entraîne normalement l'adoption d'un standard. Le SIB, par exemple, a adopté le standard ODA/ODIF qui est le plus complet par rapport aux informations sur la structure des documents. Ce choix facilite grandement l'échange mais il introduit des problèmes de codage et de décodage.

- **La recherche par le contenu textuel d'un document** : Ce type de manipulation regarde le document comme une chaîne de caractères. Cette vue impose normalement de parcourir de gros morceaux de texte non-structuré pour rechercher certaines informations. Pour palier ce problème des méthodes d'accès ont été proposées : algorithmes de balayage, filtre matériel, fichiers inversés, index et méthode de la signature. Parmi ces solutions la méthode de la signature est la mieux adaptée (si les documents traités ne sont pas trop petits) à la recherche par le contenu parce qu'elle (i) n'occupe pas beaucoup d'espace, (ii) ne restreint pas le vocabulaire de recherche comme c'est le cas de la méthode de fichiers inversés qui limite l'indexation à un ensemble de mots-clés et (iii) offre des performances raisonnables [Chr85, FC85, LL85, Jim89]. Ainsi pouvons nous exprimer des questions telles que << Quels sont les livres où apparaissent les mots "documents" et "multimédia" à la suite l'un de l'autre ou séparés par d'autres mots ? >>. En utilisant une syntaxe proche de SQL nous pouvons exprimer cette question de la manière suivante :

```
SELECT      livres.titre
```

```
FROM     livres
WHERE    contenu TALKS ABOUT
         "documents*multimédia"
```

4.2.2 Traitements des unités documentaires

Une unité documentaire est en réalité un élément logique de la structure logique générique que nous traitons dans ce travail. Le traitement des unités de documents est caractérisé par la manipulation de la structure logique de ces documents. Nous distinguons au moins trois visions possibles pour le traitement de la structure logique des documents : vision spatiale, navigationnelle et aplatie. Après la description de ces trois visions nous décrivons quelques caractéristiques de manipulation et comportement des unités documentaires que nous prenons en considération lors de la confection de nos propositions de stockage et d'accès.

- **La vision spatiale de la structure logique** : La vision spatiale permet de voir à partir d'un élément logique les autres éléments logiques par rapport à sa localisation : à droit, à gauche, en haut ou en bas. La prise en compte de cette vue permet de répondre à des requêtes telles que << Quelles sont les trois paragraphes voisins après un paragraphe spécifique ? >>, << Quelle est l'unité documentaire voisine droite de l'unité introduction du livre ? >>, etc. Dans notre cas nous pensons que ces types de requêtes ne sont pas importants pour le traitement de documents dans la bureautique. De plus, les organisations de données permettant ce type de manipulation sont compliquées et mieux adaptées aux images que nous ne traitons pas vraiment. Nous ne nous intéressons pas à ce type de vision dans cette thèse mais, à titre d'exemple, nous commentons des organisations telles que QUADTREE, K-D-TREE, POINT TREE, GRID TREE, R+TREE, etc. Ces organisations de données sont aussi appelées organisations multidimensionnelles. Les organisations

multidimensionnelles sont également basées sur des arbres et cherchent à faciliter l'accès à des gros volumes de données en prenant en considération de nouveaux types de données présentes dans des applications telles que la cartographie, la géographie, etc. En effet, toutes les applications qui manipulent des images peuvent utiliser ce type d'organisation. Dans de telles applications, on a besoin de traiter et de représenter des régions de l'espace. Ces régions sont représentées soit par leurs limites (vecteurs) soit par leurs contenus (matrice de pixels). Un exemple d'organisation multidimensionnelle est le "**QUADTREE**"". Dans cette organisation, on découpe l'espace en blocs de taille standard. Ce découpage entraîne la nécessité de manipuler des opérations telles que fusion et/ou redécoupage, extraction de contours, maillage, recherche de voisins, etc. Le principal inconvénient du "Quadtree" est son grand nombre de branches : 2^k si sa dimension est k. Un autre exemple d'organisation multidimensionnelle est le "K-D-TREE". L'origine du "K-D-TREE" a été la minimisation du nombre de 2^k branches du "Quadtree". En effet, les "K_D_TREE" sont des arbres binaires où, à chaque niveau de l'arbre, on teste une coordonnée différente. Ainsi, en deux dimensions, à la racine est x, au niveau suivant c'est y, et ainsi de suite. L'indication à chaque noeud de la coordonnée à tester a été à l'origine d'une variante appelée "adaptative k-d-tree" [Sam84].

- **La vision navigationnelle de la structure logique** : la navigation sur la structure logique des documents est très utile comme mécanisme de consultation d'un document. Elle est le résultat du cheminement basé sur la structure logique du document. Pour cheminer dans un document il faut connaître les unités documentaires composantes ainsi que les associations entre elles.

L'inconvénient de ce type de manipulation est l'exigence de la connaissance par l'utilisateur des chemins logiques complets des unités manipulées.

- **La vision aplatie de la structure logique** : cette caractéristique pallie le problème de la vision navigationnelle : c'est-à-dire que l'utilisateur n'a pas besoin de connaître les chemins logiques complets d'une structure logique. Ceci donne de la souplesse au traitement des documents car, dans les cas réels, on connaît difficilement leur structure logique par coeur.

- **La recherche par valeur des unités documentaires** : lorsque nous avons des unités documentaires de base, comme par exemple *auteur du type string*, il est intéressant de pouvoir poser la question << Quels sont les livres écrits par 'Jacques Martin' ? >>.

  Exemple :

```
SELECT    livres.titre
FROM      livres
WHERE     auteur = "Jacques Martin"
```

Cet énoncé montre l'utilisation de l'opération d'égalité de l'unité documentaire **auteur** avec sa valeur.

Le choix de l'utilisation d'un SGBD relationnel pour stocker des documents qui visent à offrir ce type d'opération n'est pas une mauvaise idée dans le sens que cette opération est déjà disponible. Ici, on peut utiliser des index pour accélerer la performance. On risque néanmoins d'être limité par une approche d'implantation qui n'est certainement pas bien adaptée à d'autres situations. L'utilisation d'un système relationnel donne plusieurs options de relations de stockage permettant de stocker les objets. La

difficulté consiste à décider de la structure relationnelle la plus adéquate en fonction des caractéristiques et privilèges de certains accès.

Une autre option pour fournir ce type de manipulation est possible par l'utilisation de fichiers classiques et par la définition d'une fonction de hachage. Cette organisation nécessite a priori une taille de fichier fixe. Un avantage de cette approche est qu'elle présente de bonnes performances tant qu'il n'y a pas de problèmes de débordements. Néanmoins, la réalité nous a montré qu'il est difficile et parfois même impossible de prévoir la taille de certains fichiers nécessaires à la plupart des applications. Ceci constitue le principal inconvénient de cette organisation. Pour pallier ce problème de débordement, plusieurs techniques de hachage ont été proposées: le hachage dynamique, le hachage extensible [FNP79], le hachage multiattributs [Lit80], le hachage en spirale [Mar79], etc. D'autres organisations basées sur des techniques de hachage sont proposées dans la littérature en [SD76, Riv76, Mar79].

- **La fréquence de manipulation des unités documentaires** : cet aspect est très important pour le stockage physique des éléments logiques des documents. Nous prenons en compte la régle des 80/20 [Knu73] qui dit que 80% des questions adressées à une base de données ne concernent en réalité que 20% des données effectivement stockées dans la base. En effet, très rares sont les questions qui manipulent tous les éléments logiques d'un document. La prise en considération des éléments logiques par une requête est plutôt un moyen pour accéder seulement à une partie d'un ou de tous les documents stockés dans la base.

Par conséquent, le partitionnement ou segmentation des données semble intéressant d'une part parce qu'il évite le transfert en mémoire centrale d'informations inutiles et d'autre part parce qu'il offre un stockage souple selon les besoins de l'application. La segmentation est faite en deux segments : un segment primaire, généralement stocké sur le support le plus rapide, et un segment secondaire stocké sur un support plus lent, ou sur le même support. En prenant en compte la régle des 80/20, le segment primaire contient 20% des données les plus utilisées. Dans un système orienté grande mémoire, comme celui de GEODE [PTS88], le segment primaire peut demeurer directement en mémoire centrale. Il y a deux techniques principales de partitionnement : partitionnement vertical et partitionnement horizontal. Des détails sur ces techniques sont disponibles dans [BCR87].

- **La recherche par le contenu d'une unité documentaire** : la seule différence entre ce type de recherche et celle sur le contenu complet d'un document est la prise en considération seulement du contenu de l'élément logique concerné. Ceci peut améliorer sensiblement la performance des requêtes telles que << Quelles sont les documents stockés dans la base qui possèdent le mot SGBD dans leur introduction ? >>.

  Exemple :

```
SELECT    document
FROM      document
WHERE     contenu.introduction TALKS ABOUT
          "SGBD"
```

- **La vérification d'existence d'une unité documentaire** : cette caractéristique est très importante lors de la manipulation des documents. Nous avons plusieurs types d'existence selon les valeurs nulles acceptées par la base : donnée non-existante, donnée inconnue, donnée nulle, etc. Ce critère n'a pas de sens pour les éléments déclarés obligatoires. Par contre, pour les éléments qui sont déclarés facultatifs, ce critère permet leur stockage afin d'optimiser des questions telles que : << Quelles sont les lettres sans signature ? >>.

- **La manipulation privilégiée de certaines unités documentaires** : on veut quelquefois privilégier l'accès à certains éléments logiques. Plusieurs techniques permettent d'améliorer la performance de l'accès à un élément logique précis : la création d'un index, la répétition d'une clé, la redondance de l'élément logique, l'éclatement de l'élément logique, etc.

- **La probabilité d'occurrence d'une unité documentaire** : cette propriété est importante parce qu'elle permet, dans certains cas, le choix d'une structure de stockage plus cohérente. Par exemple, lorsqu'un élément est indiqué comme optionnel, il peut être présent dans 90% des données stockées ou seulement dans 1% de ces données. C'est-à-dire qu'on peut choisir une structure de stockage évitant de vides. L'indication d'une probabilité floue, comme celle exemplifiée dans le chapitre précédent (section 3.4.3), avec les valeurs nulles, est la manière que nous proposons pour indiquer la probabilité d'occurrence d'une unité documentaire.

  Exemple :

  (livres(avant-propos + **dne**))

Cet énoncé indique que la plupart des occurrences de livres ne possèdent pas l'élément logique avant-propos.

4.3 Eléments clefs de la conception physique d'une base de documents

Nous profitons de toutes les informations des niveaux conceptuel et logique pour arriver au niveau physique. Il est vrai que la frontière entre les niveaux logique et physique n'est pas toujours claire. Parmi les inconvénients constatés lorsqu'on arrive au niveau physique, nous pouvons citer celui de l'appauvrissement sémantique. Le passage d'un niveau conceptuel à un niveau physique impose un changement de formalisme pour s'adapter aux moyens physiques. Ceci nécessite des transformations de modèles qui ont souvent comme conséquences des pertes de sémantique [BCR87]. Cette situation arrive, par exemple, lors du passage du modèle entité-association au modèle relationnel.

Dans le cas des documents, lorsqu'on adopte un standard d'échange du type ODA, on pallie le problème d'appauvrissement sémantique en stockant le document original codé dans le format ODIF. Ceci évite la perte de sémantique mais, par contre, introduit le problème de décodage pour reconnaître un élément logique par exemple.

Nous allons maintenant décrire quelques éléments clefs de la conception physique d'une base de documents en prenant en considération la structure logique de ces documents afin de faciliter des fonctionnalités telles que l'échange, l'interrogation et le classement.

- **Recouvrement vertical d'un élément logique** : Dans un chemin logique complet, tous les éléments de plus bas niveau sont

recouverts par les éléments logiques de plus haut niveau : ceci est une caractéristique typique des graphes hiérarchiques comme celui d'un arbre. Un algorithme de fermeture transitive sur un arbre est une manière de constater le recouvrement d'un élément logique par un autre. Un exemple d'un tel algorithme est donné dans le travail de Pucheral et al. [PRT89]. Il est à remarquer que les index de jointure proposés dans [VB86] conduisent à des exécutions très efficaces des opérations de jointure et de fermeture transitive.

- **Décodage** : Au niveau physique nous aurons la nécessité de décoder le document codé dans un standard d'échange. Ce décodage doit permettre de récupérer les informations suivantes : (i) les éléments logiques, (ii) le rang des éléments logiques multivalués, (iii) Le recouvrement vertical d'une portion de contenu et (iv) les chemins logiques.

- **Codage** : ce problème apparaît lorsqu'on adopte une stratégie de stockage en éclatant les documents sans conserver le codage original du standard d'échange. Cette stratégie est incompatible avec la stratégie qui doit privilégier l'échange de documents.

- **Prolifération de structures** : le stockage des éléments logiques multivalués d'un document entraîne normalement une prolifération de structures, si on veut éviter des vides ou la répétition de quelques valeurs. La taille de l'élément joue aussi un rôle important pour la création d'une nouvelle structure. Par exemple, le schéma physique utilisé pour le serveur OIS présente ce problème pour tous les éléments logiques d'un document et pour tous les attributs conventionnels multivalués.

- **Répétition des index** : Cette répétition matérialise les liens entre deux tables par le biais d'une autre table à deux colonnes,

contenant les identifiants des lignes joignant deux à deux. Ceci peut non seulement faciliter l'accès aux composants d'un document mais aussi réduire le nombre de décodages du document et accélerer la détection du recouvrement vertical des éléments logiques.

- **Contiguïté des éléments logiques** : La contiguïté des éléments logiques peut, dans certains cas [illegible] assurée par le standard adopté, ce qui diminue le problème de répétition d'index lors du stockage. Le standard ODA/ODIF n'assure pas cette contiguïté des portions de contenu, ce qui nous oblige de répéter les identifiants du document, le déplacement et la longueur de chaque portion de contenu appartenant à un même élément logique.

4.4 Méthodes d'accès aux documents structurés

Tous les documents que nous traitons sont composés d'un profil et d'un contenu. A chaque contenu est associé une structure logique généralement complexe. Nous nous intéressons plutôt aux méthodes d'accès aux éléments logiques de cette structure et en particulier à la recherche par contenu.

Nous proposons les quatre méthodes d'accès suivantes :

- Méthode d'Accès Autonome aux Eléments logiques Feuilles (MAAEF).
- Méthode d'Accès Autonome aux Eléments logiques Intermédiaires (MAAEI).
- Méthode d'Accès Hiérarchique aux Eléments logiques Feuilles (MAHEF).

- Méthode d'Accès Hiérarchique aux Eléments logiques Intermédiaires (MAHEI).

4.4.1 Méthode d'Accès Autonome aux Eléments logiques Feuilles (MAAEF)

La méthode d'accès autonome traite les éléments logiques indépendamment de la hiérarchie. Ceci est conforme à la vision aplatie des documents structurés décrite dans les sections 3.4.1.1 et 4.2.2 précédentes. Les éléments logiques feuilles possèdent une caractéristique très intéressante : ils sont uniques dans la hiérarchie arborescente et par conséquent ils ne partagent pas de portions de contenu entre eux.

L'intérêt pratique de cette méthode est d'accéder aux éléments logiques feuilles sans qu'il soit pour autant nécessaire de spécifier complètement le chemin logique. Dans certains cas, par exemple dans des structures de grande profondeur, la spécification complète d'un chemin logique peut devenir longue et fastidieuse.

- Exemple d'accès autonome avec recherche par contenu :

```
SELECT titre
FROM   guide
WHERE ..légende TALKS ABOUT "La Joconde"
```

Cet énoncé montre la sélection des documents du type guide par contenu textuel de l'élément logique légende indépendamment de la hiérarchie. Il faut rappeler que la recherche par contenu sur les éléments logiques des documents est le principal but de ce travail.

- Exemple d'accès autonome avec recherche par valeur :

```
SELECT titre
FROM   guide
```

WHERE ..légende = "La Joconde est le tableau le plus celèbre (...)
Leonard da Vinci"

Cet énoncé montre la sélection de documents du type guide par valeur de l'élément logique légende. Il est important de remarquer que nous ne nous intéressons pas à ce type d'accès aux éléments logiques sauf ceux considérés comme attributs du profil qui sont généralement de petite taille.

4.4.2 Méthode d'Accès Autonome aux Eléments logiques Intermédiaires (MAAEI)

Les éléments logiques intermédiaires partagent leur contenu avec d'autres éléments logiques du chemin hiérarchique correspondant. Par exemple, l'élément logique peinture partage son contenu avec les éléments figure et description. Ceci signifie qu'il existe un problème de recouvrement vertical pour indexer ces éléments.

Un autre exemple encore basé sur le guide illustré par la figure 3.1 est l'accès à l'élément peinture montré ci-dessous :

- Exemple :

SELECT titre
FROM guide
WHERE ..peinture TALKS ABOUT "La Joconde"

Il est important de remarquer que cet énoncé accède à l'élément intermédiaire peinture avec recherche par contenu.

L'intérêt pratique de cette méthode est d'établir un chemin d'accès par le biais du nom de l'unité documentaire indépendamment de la structure logique.

4.4.3 Méthode d'Accès Hiérarchique aux Eléments logiques Feuilles (MAHEF)

L'intérêt pratique de cette méthode est de permettre la distinction entre deux éléments logiques de même nom.

Cette méthode prend en considération la structure logique hiérarchique associée aux contenus des documents. Ceci équivaut à la vision navigationnelle décrite dans la section 4.2.2 précédente. Un exemple énoncé exprimant ce type d'accès est donné ci-dessous.

- Exemple.

```
SELECT  titre
FROM    guide
WHERE contenu.peinture.figure.légende
      TALKS  ABOUT "La Joconde"
```

4.4.4 Méthode d'Accès Hiérarchique aux Eléments logiques Intermédiaires (MAHEI)

Un chemin logique hiérarchique est aussi nécessaire pour accéder aux éléments logiques intermédiaires.

- Exemple.

```
SELECT  titre
FROM    guide
WHERE contenu.peinture TALKS ABOUT "La Joconde"
```

4.5 Méthodes de stockage des documents structurés

Les méthodes de stockage sont étroitement liées aux méthodes d'accès offertes. Nous proposons des schémas physiques capables de permettre les accès définis dans la section précédente : MAAEF, MAAEI, MAHEF et MAHEI.

Tout d'abord, nous allons décrire une organisation de base pour stocker les documents avec leurs contenus ainsi que l'indexation de tous les mots existants dans le contenu de ces documents. En effet, les options de stockage consistent à modifier l'organisation de base soit en ajoutant des index de selection ou de jointure, soit en augmentant des colonnes aux tables déjà existantes.

Toutes les méthodes de stockage décrites ont comme base les hypothèses simplificatrices (H_i) et privilégient les trois aspects (P_j) suivants :

H_1 - La structure de stockage est proposée en termes de tables afin d'éviter la prise en considération des détails techniques liés aux structures possibles d'accueil telles que fichiers classiques, mémoire principale, SGBD relationnel, hiérarchique ou réseau, etc.

H_2 - Tous les documents stockés sont composés d'un profil et d'un contenu. Cette même composition est prévue par le standard ODA. Néanmoins, l'intérêt du profil ici dépasse largement le cadre du standard ODA : il assure une vision uniforme de tous les documents de la base.

H_3 - Un élément non-structuré est toujours un élément feuille et peut être multivalué. Cette règle simplifie la modélisation des documents que nous traitons sans pour autant restreindre le pouvoir d'expression du modèle.

P_1 - L'échange des documents est privilégié.

P_2 - L'accès aux attributs du profil est privilégié.

P_3 - L'accès indexé à tous les mots existants dans le contenu du document est aussi privilégié.

Avant de décrire les méthodes de stockage proprement dites nous décrivons l'organisation de base pour toutes les options de schéma physique proposées et l'indexation textuelle du contenu des documents.

4.5.1 Organisation de base

Afin de simplifier notre étude analytique des différentes méthodes de stockage et d'accès aux documents, nous proposons d'organiser les schémas physiques sous forme de tables plates utilisant éventuellement des index.

Nous supposons l'existence de deux types d'index stockés aussi sous forme de tables : index de sélection et index de jointure. Les index de sélection sont stockés comme des tables à deux colonnes (identifiant et valeur) triées sur valeur et les index de jointure sont stockés sous forme de tables d'identifiants uniques et invariants appelés ID triées suivant l'ordre des valeurs référencées.

De cette manière pour stocker les documents structurés avec une vision soit aplatie soit hiérarchique de cette structure nous proposons comme schéma physique de base les tables suivantes :

- Table de tous les contenus des documents stockés dans la base (CONTENU) : La table CONTENU stocke le document original codé en un des standards acceptés par le système. Cette table contient trois colonnes :
  - **contenu** : l'identifiant d'un contenu complet d'un document.
  - **compteur** : le compteur de partage de ce contenu par les documents de la base.

- **ordre** : utilisé pour permettre de stocker les documents de grande taille en indiquant l'ordre des parties de 64K octets. Cette taille a été choisie pour permettre l'utilisation d'un champ long si l'on utilise ORACLE comme système d'accueil.
- **doccode** : le document original codé par le biais d'un des standards d'échange accepté par le SGBDD.

La figure 4.2 illustre la table "CONTENU" avec deux contenus de documents.

CONTENU

contenu	compteur	ordre	doccode
51	1	1	...
52	1	1	...
52	1	2	...

Fig. 4.2 La table "CONTENU"

Tous les contenus originaux des documents sont stockés dans la table CONTENU.

Coût de stockage

Nous supposons que les index de colonnes d'une table sont stockés aussi sous forme de tables à deux colonnes (ID et valeur) triées sur valeur. Ainsi, l'indexation de la colonne contenu de la table CONTENU est réalisée afin d'optimiser l'opération de recherche par contenu soit sur le contenu d'un document, soit sur le contenu d'un élément logique. Dans la suite, **t** représente la taille d'un Identifiant (ID).

Soit une base documentaire contenant **nc** contenus de documents associés aux **N** documents stockés dans cette base (**N > nc**) et soit **tmc** la taille moyenne d'un contenu d'un document, le coût de stockage de la table CONTENU équivaut à :

$$\text{Coût(CONTENU)} = nc\ (3.t + tmc)$$

Evidemment, les coûts donnés par la suite sont à réévaluer dans le cas de choix de gestion spécifique avec un objectif d'optimisation. En particulier, les coûts de stockage réels doivent tenir compte des techniques de pagination employées et les coûts de recherche (accès) sont à adapter selon les modes d'organisation des index (B-arbre, index dense, index creux, etc.).

De plus, on s'abstrait du coût supplémentaire de stockage dû à l'ajout des index de jointure et de sélection pour ne pas alourdir les formules de coût de stockage. Sachons que ce coût supplémentaire est facilement calculé par les formules ci-dessous établies par Pucheral [PRT89] :

$$\text{Coût_sélection} = k(C(t + tma))$$
$$\text{Coût_jointure} = j(C.2.t)$$

Les paramètres retenus pour ces formules sont les suivants :

k : nombre d'index de sélection ajouté à la table.

j : nombre d'index de jointure ajouté à la table.

C : cardinalité moyenne de la table.

t : taille d'un identifiant (ID).

tma : taille moyenne de la valeur de l'attribut indexé.

- Table de tous les documents de la base (DOCUMENT) : Cette table est en réalité un index de jointure car elle possède des couples d'identifiants (ID du document et ID de son contenu). La figure 4.3 illustre la table DOCUMENT avec les deux colonnes qui correspondent à ces identifiants.

DOCUMENT

document	contenu
622	51
777	52

Fig. 4.3 La table "DOCUMENT"

On s'abstrait ici des cas de stockage et de l'accès aux attributs du profil car il fait partie du cas général du stockage et accès des attributs de petite taille.

La colonne "contenu" rend le stockage du contenu du document indépendant de sa taille et permet le partage d'un même contenu complet à plusieurs documents de la base. Il est à remarquer que ce schéma n'offre aucun mécanisme pour qu'une partie d'un document soit partagée par plusieurs documents.

Coût de Stockage

Une formule simple capable d'évaluer le coût de stockage de la table DOCUMENT est donnée ci-dessous :

$$\text{Coût(DOCUMENT)} = N \times 2 \times t$$

Les paramètres utilisés sont :

N - nombre de documents stockés dans la base.

t - taille d'un identifiant (ID).

4.5.2 Indexation textuelle des documents

Les mécanismes pour indexer tous les mots existants dans un document sont utilisés comme base pour implanter les accès aux éléments logiques décrits antérieurement. Il est à remarquer que l'ordre des occurrences de ces mots dans le texte doit être connu parce qu'il existe des recherches d'expression en combinaison avec des jokers. Pour plus de détails sur la recherche textuelle se rapporter à la thèse de Jimenez [Jim89].

Exemple :

```
SELECT titre
FROM   guide
WHERE contenu TALKS ABOUT "La*Joconde"
```

Dans cet énoncé, nous voulons les titres des guides de musée qui parlent de la Joconde (exemple : "La célébre Joconde", "La (...) Joconde", etc).

Le jocker "*" utilisé limite la portée de la recherche à une portion de contenu c'est qui évite la sélection de documents qui contiennent, par exemple, "La" dans le premier paragraphe et "Joconde" dans le dernier.

L'accès avec recherche par contenu est réalisé par le biais de la table d'indexation décrite ci-dessous.

- Table d'indexation des portions de contenu (INDCONT) : Cette table est conforme au privilège P_3. En effet, l'indexation textuelle des documents est réalisée par cette table en utilisant la méthode de la signature. L'opérateur TALKS ABOUT sur le contenu d'un document est possible grâce à cette indexation. L table d'indexation (cf. figure 4.4) est composée de quatre colonnes :
  - **contenu** : identifiant du contenu.
  - **deplac** : déplacement à partir du début du document original. Ce déplacement permet de localiser une portion du contenu. Il indique le nombre d'octets à ignorer au début du document original stocké dans la table CONTENU.
  - **long** : taille de la portion de contenu en octets.
  - **signature** : signature de la portion de contenu.

INDCONT

contenu	deplac	long	signature
51	1	10	...
51	11	110	...
52	121	220	...

Fig. 4.4 Indexation textuelle des documents

Coût de Stockage

Une formule simple pour évaluer le coût de stockage de la table INDCONT est donnée ci-dessous :

$$\text{Coût(INDCONT)} = (N \times tmc / tmpc) \times (3 \times t + ts)$$

Les paramètres utilisés sont :

N - nombre de documents stockés dans la base.

t - taille d'un identifiant (ID).

tmc - taille moyenne du contenu d'un document.

tmpc - taille moyenne d'une portion de contenu.

ts - taille choisie d'une signature.

Nous détaillons par la suite les tables de stockage qui permettent d'accéder aux éléments logiques selon les méthodes déjà décrites dans la section 4.4.

4.5.3 Stockage avec une Vision Aplatie (SVA).

Comme nous l'avons déjà dit tous les documents sont en réalité structurés, car ils sont composés d'un profil et d'un contenu et à chaque contenu est associé une structure logique hiérarchique.

Le stockage de ces documents structurés avec une vision aplatie est une alternative pour implanter les méthodes d'accès MAAEF et MAAEI.

Avec la vision aplatie des documents on accède aux éléments logiques indépendamment de la hiérarchie. Pour cette raison nous appelons ce type d'accès, accès autonome.

La mise en oeuvre de cet accès implique l'ajout de la colonne "element" à la table INDCONT pour les éléments feuilles ($INDEF_A$) et la création de la table $INDEI_A$ pour les éléments intermédiaires.

Cette situation est mieux caractérisée par l'ajout des hypothèses suivantes :

H_4 - Les documents sont stockés sans la création d'aucun mécanisme pour faciliter l'accès à leurs éléments logiques sauf ceux considérés comme éléments sujets à l'accès autonome.

H_5 - Il est important de remarquer que les éléments logiques sujets à l'accès autonome peuvent être distingués par des caractéristiques en fonction de la taille et l'aspect multi ou monovalués. Ces caractéristiques influencent l'organisation du schéma physique. Le fait que ces éléments apparaîssent fréquemment ou rarement pour une classe de documents influent aussi sur le schéma physique.

4.5.3.1 La table $INDEF_A$

L'accès autonome à un élément logique feuille (MAAEF) est réalisé par le biais de la table d'indexation $INDEF_A$. Cette table est obtenue par l'ajout de la colonne "element" à la table INDCONT.

Nous nous intéressons plutôt aux éléments logiques feuilles de grande taille et de taille variable qui peuvent être aussi monovalués ou multivalués. La gestion d'ordre des éléments multivalués n'est pas assurée aux éléments feuilles sujets à l'accès autonome.

Si ces éléments sont fréquents dans la classe document ou dans une sous-classe alors ils peuvent être intégrés au profil correspondant.

Si ces éléments sont des éléments feuilles qui apparaissent rarement, alors dans tous les cas (petite ou grande taille, monovalués ou multivalués) on les stocke dans la colonne "element" dans la table $INDEF_A$. La figure 4.5 illustre la table $INDEF_A$.

$INDEF_A$

contenu	deplac	long	element	signature
51	1	10	31	. . .
51	11	110	32	. . .
52	121	220	32	. . .

Fig. 4.5 Indexation des éléments feuilles

Evaluation du coût de stockage

Le coût de stockage de cette table est le coût de stockage de la table INDCONT déjà calculé plus le coût supplémentaire pour l'ajout de la colonne "element".

$$\text{Coût_supp(element)} = (N \times tmc / tmpc) \times t$$

Ainsi, le coût de stockage de la table $INDEF_A$ est donné par la formule :

$$\text{Coût}(INDEF_A) = N \times tmc / tmpc\ (4t + ts)$$

Les paramètres retenus pour l'évaluation du coût de stockage sont les mêmes utilisés lors de l'évaluation du coût de stockage de la table DOCUMENT.

Evaluation de performance (recherche)

En supposant l'existence d'un index sur "element" le coût d'accès autonome aux éléments feuilles est donné par la complexité de la recherche dichotomique sur la colonne "element" dans la table $INDEF_A$. Bien entendu, ceci implique que la table soit ordonnée par cette colonne. De cette manière, le coût d'accès est de l'ordre de :

$$\log_2 \text{Card}(\text{INDEF}_A)$$

La formule ci-dessous donne la cardinalité de la table INDEF_A :

$$\text{Card}(\text{INDEF}_A) = N \times tmc / tmpc$$

4.5.3.2 La table INDEI_A

La méthode MAAEI est réalisée par le biais de la table INDEI_A. Cette table indexe les éléments intermédiaires sujets à l'accès autonome et est composée de quatre colonnes décrites ci-dessous :

- **contenu** : identifiant du contenu.
- **deplac** : déplacement à partir du début du document original. Ce déplacement permet de localiser une portion du contenu sujet à une opération du filtrage. Il indique le nombre d'octets à ignorer au début du document original stocké dans la table CONTENU.
- **long** : taille de la portion de contenu en octets.
- **element** : identifiant de l'élément.

La figure 4.6 illustre la table INDEI_A.

INDEI_A

contenu	deplac	long	element
51	1	10	31
51	11	110	32
52	121	220	32

Fig. 4.6 Indexation des éléments intermédiaires

Evaluation du coût de stockage

Les paramètres retenus pour l'évaluation sont les suivants :

nei_A - nombre d'éléments intermédiaires sujets à l'accès autonome.

tmei - taille moyenne d'un élément intermédiaire.

tmpc - taille moyenne d'une portion de contenu.

De cette manière, le coût de stockage de la table $INDEI_A$ est donné par la formule :

$$\text{Coût}(INDEI_A) = 4 \times t \times nei_A \times tmei / tmpc$$

Evaluation de performance

Dans ce cas, le coût d'accès autonome à un élément intermédiaire (MAAEI) peut être évalué en prenant en considération la complexité de l'algorithme d'accès. Cette complexité dépend directement de la cardinalité de $INDEI_A$.

Les deux algorithmes que nous retenons pour accéder à ce type d'élément sont :

1. Algorithme 1 = Recherche séquentielle.
2. Algorithme 2 = Recherche dichotomique.

Nous allons les décrire ci-dessous.

Algorithme 1 : L'application de cet algorithme caractérise la situation où il n'existe pas d'index.

Algorithme:

étape 1 : Initialiser un balayage sur la table $INDEI_A$.

étape 2 : Pour chaque ligne comparer l'identifiant de l'élément

recherché avec l'identifiant élément de la table. S'ils sont égaux, alors récupérer la signature de l'élément par le contenu et le déplacement dans la table INDCONT modifiée ($INDEI_A$).
étape 3 : Continuer le balayage.

Complexité :
La complexité de l'algorithme 1 d'accès autonome à un élément intermédiaire est de l'ordre de :

$$\mathbf{Card(INDEI_A) / 2 + Card(INDEF_A) / 2.}$$

Il est à remarquer que l'addition de $Card(INDEF_A)$ est due la nécessité de récupérer la signature de chaque portion de contenu sélectionnée dans la table $INDEI_A$.

La formule pour calculer la cardinalité de $INDEI_A$ est la suivante :

$$\mathbf{Card(INDEI_A) = nei \times tmei / tmpc.}$$

Algorithme 2 : L'application de cet algorithme caractérise la situation où il existe d'index trié. La recherche dichotomique est très répandue, pour cette raison l'algorithme n'est pas donné ici.

Complexité :
La complexité est de l'ordre de :

$$\mathbf{Log_2(Card(INDEI_A)) + tmei / tmpc \times Log_2(N \times tmc / tmpc).}$$

La maintenance d'index sur les identifiants permet d'appliquer la recherche dichotomique c'est qui est intéressant du point de vue efficacité. Il est important de remarquer que cette option est la plus adaptée pour stocker les éléments intermédiaires sujets à l'accès autonome car elle évite la répétition des signatures des éléments

logiques qui présentent des problèmes de recouvrement vertical avec d'autres éléments indexés.

4.5.4 Stockage avec une Vision Hiérarchique (SVH)

Jusqu'à présent nous avons traité seulement des solutions plus adaptées à la manipulation des documents avec une vision aplatie de ceux-ci. La manipulation des documents avec une vision hiérarchique impliquerait plusieurs balayages dans les fichiers d'index pour détecter le recouvrement vertical des portions de contenu et la déduction des chemins logiques. Un chemin logique est dérivé directement de la structure logique. La structure logique explicite les éléments atomiques constituant les éléments composés ainsi que les liaisons existant entre ces éléments. Un chemin logique montre quelques liaisons possibles. Un chemin logique complet explicite les liaisons à partir d'un élément racine jusqu'à un élément feuille. Par exemple, guide.introduction. Une caractéristique importante dans un modèle hiérarchique c'est qu'un élément feuille n'appartient qu'à un seul chemin logique complet.

Un document structuré posséde une structure hiérarchique de ses éléments logiques. Ici la seule manière d'accéder aux éléments logiques est par le biais d'une opération navigationnelle sur ces éléments hiérarchiques. Tous les éléments feuilles sont indexés par leur chemin logique complet. Si on veut accéder aux éléments intermédiaires de cette structure il faut les indéxer ou utiliser des opérateurs tels que LIKE sur le chemin logique complet. Pour une question de performance, l'accès à ces éléments doit se faire plutôt par le biais d'index que d'utiliser l'opérateur LIKE.

Le stockage avec une vision hiérarchique est caractérisé par le remplacement de l'hypothèse $\mathbf{H_5}$ par l'hypothèse moins restrictive $\mathbf{H_6}$

et l'ajout de l'hypothèse H_7 décrites ci-dessous :

H_6 - Nous stockons les documents en créant des mécanismes pour faciliter l'accès navigationnel aux éléments logiques.

H_7 - Tous les éléments hiérarchiques multivalués possèdent un chemin de rangs qui permet l'identification plus précise de leur position dans la structure logique. Par exemple, section 3 du chapitre 4. Nous traitons ce type d'ordre (ordre entre les éléments de même type) parce qu'il peut améliorer sensiblement la qualité de la réponse à une requête. Par contre, l'ordre entre les éléments de types différents n'apporte rien dans le problème que nous traitons. Par exemple, introduction vient avant la conclusion.

En effet, le schéma physique précédent n'empêche pas l'accès aux éléments logiques, mais il exige le décodage répétitif des documents stockés pour manipuler leur structure logique. Le pire est que ce décodage doit être réalisé au moment de l'évaluation de la requête pénalisant donc le temps d'accès. C'est pourquoi nous proposons un schéma physique composé des tables $INDEF_{AH}$ pour indexer les éléments feuilles et $INDEI_H$ pour indexer les éléments intermédiaires.

Nous allons maintenant décrire ces tables qui permettent de mettre en oeuvre les méthodes d'accès MAHEF et MAHEI.

4.5.4.1 La table $INDEF_{AH}$

Cette table est résultat de l'ajout des colonnes "chemin_log" et "rang" à la table $INDEF_A$ permettant ainsi l'accès hiérarchique aux éléments feuilles multivalués ou monovalués (MAHEF). Cette stratégie est explicable parce que les éléments feuilles sont atomiques et ne posent pas le problème de recouvrement vertical.

La figure 4.7 illustre la table $INDEF_{AH}$ composée des colonnes suivantes :

- **contenu** : identifiant du contenu.
- **deplac** : déplacement à partir du début du document original. Ce déplacement permet de localiser une portion du contenu sujet à une opération du filtrage. Il indique le nombre d'octets à ignorer au début du document original stocké dans la table CONTENU.
- **long** : taille de la portion de contenu en octets.
- **element** : identifiant de l'élément logique feuille sujet à l'accès autonome.
- **chemin_log** : chemin logique de l'élément feuille sujet à l'accès hiérarchique.
- **rang** : l'ordre de répétition de chaque élément logique multivalué appartenant au chemin logique. Les éléments monovalués n'ont pas d'ordre de répétition par une raison évidente.
- **signature** : signature de la portion de contenu.

$INDEF_{AH}$

contenu	deplac	long	element	chemin_log	rang	signature
51	1	10	31	30.31	1.1	...
51	11	110	32	30.32	1.1	...
52	121	220	32	30.32	1.1	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 4.7 la table $INDEF_{AH}$

Coût de stockage

Nous ajoutons aux paramètres déjà utilisés pour calculer le coût d'autres tables le p signifiant la profondeur maximum de l'arbre.

De cette manière le coût de stockage de la table $INDEF_{AH}$ est donné par le formule ci-dessous :

$$\text{Coût}(INDEF_{AH})=(6.t + p.t + ts)\ N \times tmc / tmpc$$

Coût de performance

En utilisant des index la recherche dichotomique reste valable. De cette manière, le coût de MAHEF est de l'ordre de :

$$Log_2(N \times tmc / tmpc)$$

4.5.4.2 La table $INDEI_H$

La Méthode d'Accès Hiérarchique aux Eléments logiques Intermédiaires (MAHEI) est réalisée par le biais de la table $INDEI_H$. Il est important de remarquer que cette option de stockage ne pose pas le problème de prolifération de structures et évite la répétition de la colonne signature.

La figure 4.8 illustre la table $INDEI_H$ composée des colonnes suivantes :

- **contenu** : identifiant du contenu.
- **deplac** : déplacement à partir du début du document original. Ce déplacement permet de localiser une portion du contenu sujet à une opération du filtrage. Il indique le nombre d'octets à ignorer au début du document original stocké dans la table CONTENU.
- **long** : taille de la portion de contenu en octets.

- **chemin_log** : chemin logique de l'élément intermédiaire sujet à l'accès hiérarchique.
- **rang** : l'ordre de répétition des éléments logiques multivalués appartenant au chemin logique.

$INDEI_H$

contenu	deplac	long	chemin_log	rang
51	1	10	30.31	1.1
51	11	110	30.32	1.1
52	121	220	30.32	1.1
⋮	⋮	⋮	⋮	⋮

Fig. 4.8 la table $INDEI_H$

Coût de stockage

Afin d'évaluer le coût de stockage de cette table nous ajoutons un nouveau paramètre nei_H qui indique le nombre d'éléments logiques intermédiaires sujets à l'accès hiérarchique. La taille moyenne des éléments intermédiares (tmei) est la même pour les éléments sujets à l'accès autonome et pour les éléments sujets à l'accès hiérarchique.

Le coût de stockage associé à cette table est donné par la formule suivante :

$$\mathbf{Coût(INDEI_H)=(3.t + (p\text{-}1)t + (p\text{-}1)t)\ nei_H \times tmei / tmpc)}$$

Coût de performance

Le coût d'accès est donné en fonction de la complexité de l'algorithme d'accès. Cet algorithme est la recherche dichotomique dans la table $INDEI_H$ sur la colonne "chemin_log" plus la recherche des signatures dans la table $INDEF_{AH}$. C'est-à-dire de l'ordre de :

$$\mathbf{Log_2(nei_H \times tmei / tmpc) + tmei / tmpc \times Log_2(N \times tmc / tmpc).}$$

4.6 Conclusions

Nous avons défini des schémas physiques de base de données documents capables de stocker et d'accéder à ces documents en prenant en considération leurs structures logiques. Pour simplifier notre tâche nous n'avons pas consideré une structure d'accueil spécifique telle que SGBD relationnel, hiérarchique ou réseau mais des tables plates.

Plusieurs options de schémas physiques ont été discutées, et le choix de l'option la plus adéquate dépend des objectifs et des hypothèses simplificatrices établis. Si on se satisfait de traiter les documents avec une vision aplatie de leur structure, le schéma physique le plus adapté est celui formé des tables suivantes : CONTENU, DOCUMENT, $INDEF_A$ et $INDEI_A$. Nous remarquons que cette option permet les méthodes d'accès suivantes : MAAEF et MAAEI. Nous négligeons l'ordre des éléments multivalués lors du traitement des documents avec la vision aplatie parce que cette information n'a un sens que replacé dans un contexte hiérarchique.

Si nous nous intéressons au traitement des documents par le biais d'une vision hiérarchique totale ou partielle l'option la plus adaptée est celle formée du schéma physique composé des tables suivantes : CONTENU, DOCUMENT, $INDEF_{AH}$, $INDEI_A$ et $INDEI_H$. Ce schéma physique permet tous les types d'accès aux éléments logiques décrits dans la section 4.4. Ici, l'ordre des éléments multivalués est géré.

Il est importante de remarquer que indépendamment de l'application d'une des stratégies de système (ouvert, partielement

ouvert ou fermé) les schémas physiques restent valables.

Enfin, une interface fonctionnelle (décrite dans l'annexe 3) a été réalisée en s'appuyant sur ces propositions pour montrer la faisabilité de nos idées.

CHAPITRE 5

REALISATION

5

REALISATION

5.1 Introduction

L'identification d'un document dans notre gestionnaire de documents implémenté est un couple de deux identificateurs : l'identificateur du document et l'identificateur du contenu. Le contenu lui-même contient des informations sur le profil du document, les valeurs de l'occurrence du document et les structures logiques initiales de ce document codées selon le standard d'échange (ODA). Dans le cas dans un environnement fermé, un schéma associé à chaque document stocké décrit complètement sa structure. Si nous sommes dans un environnement partiellement ouvert, il y a un schéma partiel associé à chaque document. Enfin, si nous sommes dans un

environnement ouvert aucun schéma n'est associé aux documents stockés ce qui permet une liberté vis-à-vis des aspects structuraux des documents. En réalité, dans un environnement ouvert, nous pouvons associer un schéma informatif sans contraintes en vue de l'indexation de certains éléments logiques décrits.

Le schéma conceptuel est exprimé par le biais d'un langage de description de données du Système de Gestion de Bases de Données Documents (SGBDD). En effet, la standardisation décrite par le schéma conceptuel permet l'application d'une vérification structurelle vis-à-vis de la structure logique originale du document à gérer. La structure logique originale du document arrive avec le contenu du document codé selon le standard d'échange ODA.

Nous avons concrétisé un travail d'implémentation en deux phases. Dans une première phase, un gestionnaire de documents a été réalisé dans le cadre du Projet Esprit n. 231 DOEOIS (Design Operational and Evaluation Office Information Serveur). Ce projet, portant sur l'échange et la manipulation de documents structurés conformes au modèle ODA (Office Documents Architecture; CCITT, ISO), a associé l'Institut Fraunhoffer de l'Université de Stuttgart, le Trinity College of Dublin, le Laboratoire de Génie Informatique de Grenoble et les Centres de Recherche Bull, Olivetti (au début) et ICL. Les documents ODA utilisés pour tester le Gestionnaire implementé ont été générés par l'éditeur multimédia PODA développé par Bull-Echirolles.

Ce gestionnaire est intégré au prototype du Serveur d'Information Bureautique (SIB), actuellement réalisé comme logiciel frontal sur le SGBD ORACLE.

Une deuxième phase de notre travail a consister à implémenter un gestionnaire autonome de documents indépendamment du SIB en profitant de l'expérience précédante et en conservant les fonctionnalités suivantes :

- Echange et stockage de documents ODA, structurés au format ODIF (internalisation et externalisation), en préservant la structure d'origine.
- Sélection des attributs du profil, lesquels donnent une vision uniforme des documents.
- Sélection par contenu textuel de tout le document, en utilisant la méthode i dite "des signatures" [Jim89].

Dans ce chapitre, nous définissons d'abord le SIB, puis nous donnons l'architecture du Gestionnaire de Documents Autonome (GDA) pour préciser ensuite les services offerts par le prototype en décrivant les opérations les plus caractéristiques : l'internalisation et la sélection.

5.2 Serveur d'Information Bureautique

Le SIB propose un ensemble de fonctionnalités qui donnent un support au développement d'applications dans un bureau : édition de documents (rapports, lettres, etc), ventes, manipulation de messages, suivi des clients, etc. Il faut donc fournir des outils généraux qui permettent la modélisation des activités de bureau et des information qui y circulent.

Les Serveurs de Bureaux sont normalement construits autour d'autres serveurs spécifiques, comme les serveurs d'impression, de messages, de stockage, etc.

Nous pouvons distinguer dans l'architecture du SIB implémenté les modules suivants (cf. figure 5.1) :

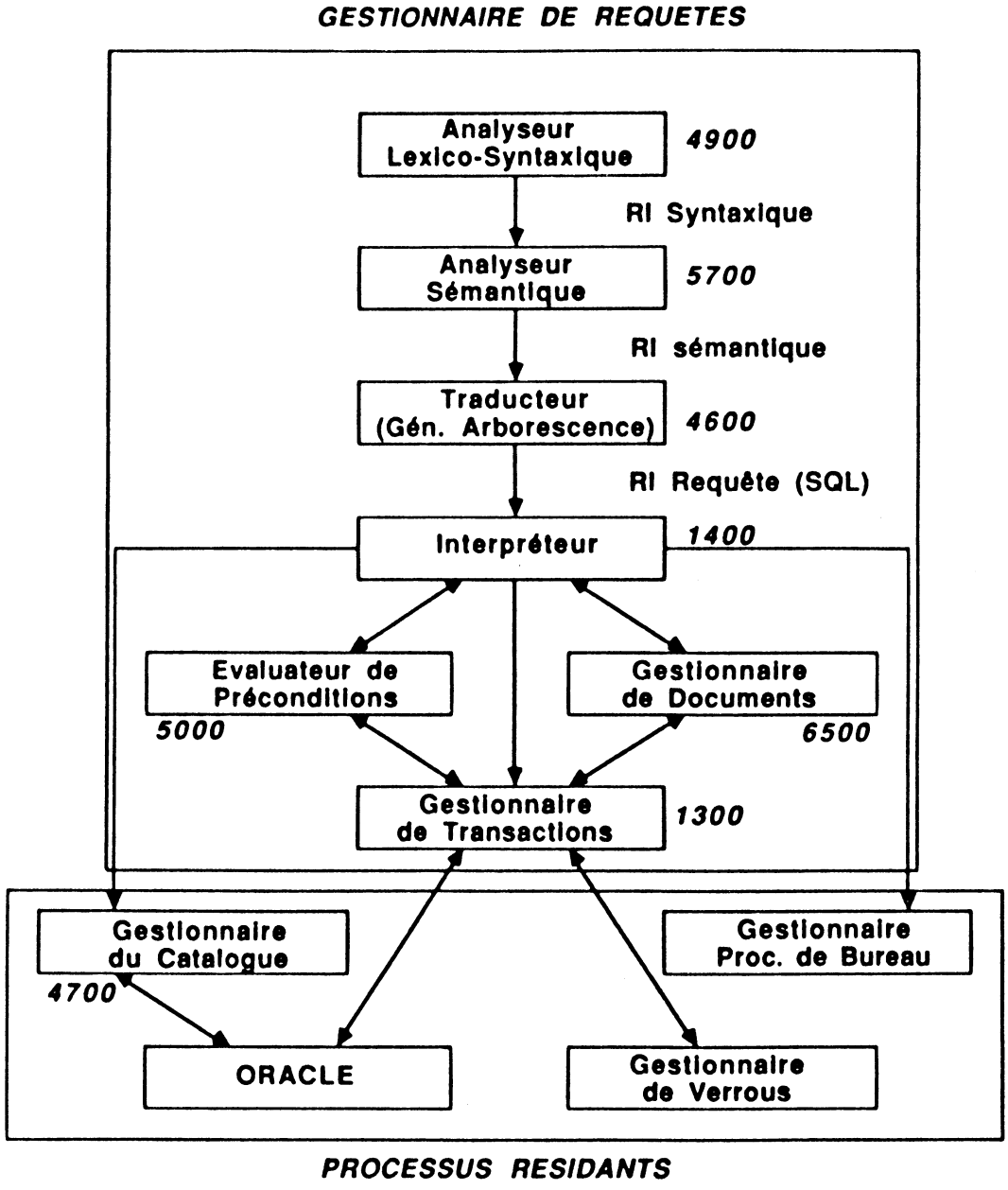

Fig. 5.1 Architecture du SIB

- L'Analyseur Lexico-Syntaxique d'instructions FML (voir Annexe 1).
- L'Analyseur Sémantique pour vérifier les instructions FML et ajouter les informations spécifiques des phases suivantes.
- Le Traducteur transforme les requêtes FML en un arbre d'opérateurs.
- L'Interprète, qui est chargé de l'exécution des opérations. C'est le noyau du Système et il contrôle le bon déroulement des opérations. En particulier, il est responsable de l'assignation des tâches.
- Le Gestionnaire de Transactions assure la maintenance de la cohérence et de l'intégrité des données. Il doit garantir aux applications l'exécution complète des activités et la reprise après pannes. Grâce à cette fonctionnalité du SIB, l'usager, lorsqu'il définit une application, peut décider quelles sont les opérations sur lesquelles il veut assurer l'indivisibilité.
- Le Gestionnaire de Verrous pour gérer l'accès concurrent aux données par plusieurs transactions.
- Le Gestionnaire d'activités et de Procédures de Bureau. Il assure la bonne réalisation des activités définies dans le modèle dynamique des applications. Certaines des activités sont entièrement automatiques, d'autres partiellement automatiques ou manuelles.
- Le Gestionnaire de Documents est chargé des opérations définies sur les objets de type document. Il s'agit de fournir à l'interprète une interface de manipulation de ces objets tout à fait uniforme

par rapport aux autres objets du système. Il faut garantir qu'au niveau externe du Serveur, les documents sont vus comme des entités quelconques et que leur complexité est transparente pour les autres niveaux. Les opérations sur les documents sont : l'acquisition, la sélection et la restitution. L'opération de sélection permet l'accès aux documents par leurs attributs et par leur contenu. L'opération de restitution permet la communication de documents à l'extérieur, par exemple vers des éditeurs ou des imprimantes, au niveau application.

- Le Gestionnaire du Catalogue contient la description du schéma conceptuel des objets manipulés par le serveur y compris des documents. Dans le cas du SIB, le schéma conceptuel est exprimé en FM.

Pour avoir une idée de l'effort de programmation éxigé pour la mise en oeuvre de ce prototype nous informons dans la figure 5.1 le nombre de lignes de code (la plupart en langage C) de chaque module implémenté.

Notre contribution portait plus particulièrement sur les opérations concernant les documents, et notamment l'accès par le contenu en prenant en considération la structure logique des documents.

5.3 Gestionnaire de Documents Autonome (GDA)

Le GDA réalisé garde la distinction Profil, Contenu et Document imposée dans le gestionnaire de documents mis en oeuvre dans le SIB. Ce gestionnaire autonome fournit une interface fonctionnelle ainsi qu'une interface conversationnelle.

Ce gestionnaire fournit également les outils de base en vue de traiter la structure logique des documents.

Le traitement de documents codés dans un autre standard (TROFF, SGML, GRIF, etc.) est facilement réalisable en programmant un nouveau module de décodage spécifique au standard.

Ce gestionnaire est autonome parce qu'il peut être utilisé à partir du SIB mais aussi à partir d'un autre système tel qu'un SGBD relationnel. Pour ceci, nous avons défini une interface fonctionnelle décrite dans l'annexe 3.

5.3.1 Architecture du GDA

Le Gestionnaire de Documents Autonome (GDA) est conçu comme un module étendant de manière contrôlée un SGBD pour intégrer la manipulation et le stockage du type complexe document structuré.

Le GDA se décompose selon les modules suivants (cf. figure 5.2) :

- LIBGDC : librairie des primitives internes au gestionnaire, écrites en C et en Pascal. Ces primitives sont en fait deux types : (i) Internalisation/externalisation des documents (ii) Indexation/filtrage des contenus par la méthode dite "de la signature".
- LIBCATAL : librairie des modules de gestion du catalogue et de création automatique des relations de stockage.
- LIBGESDOC : interface fonctionnelle à trois niveaux décrite dans l'annexe 3.
- GESDOC : interface conversationnelle.

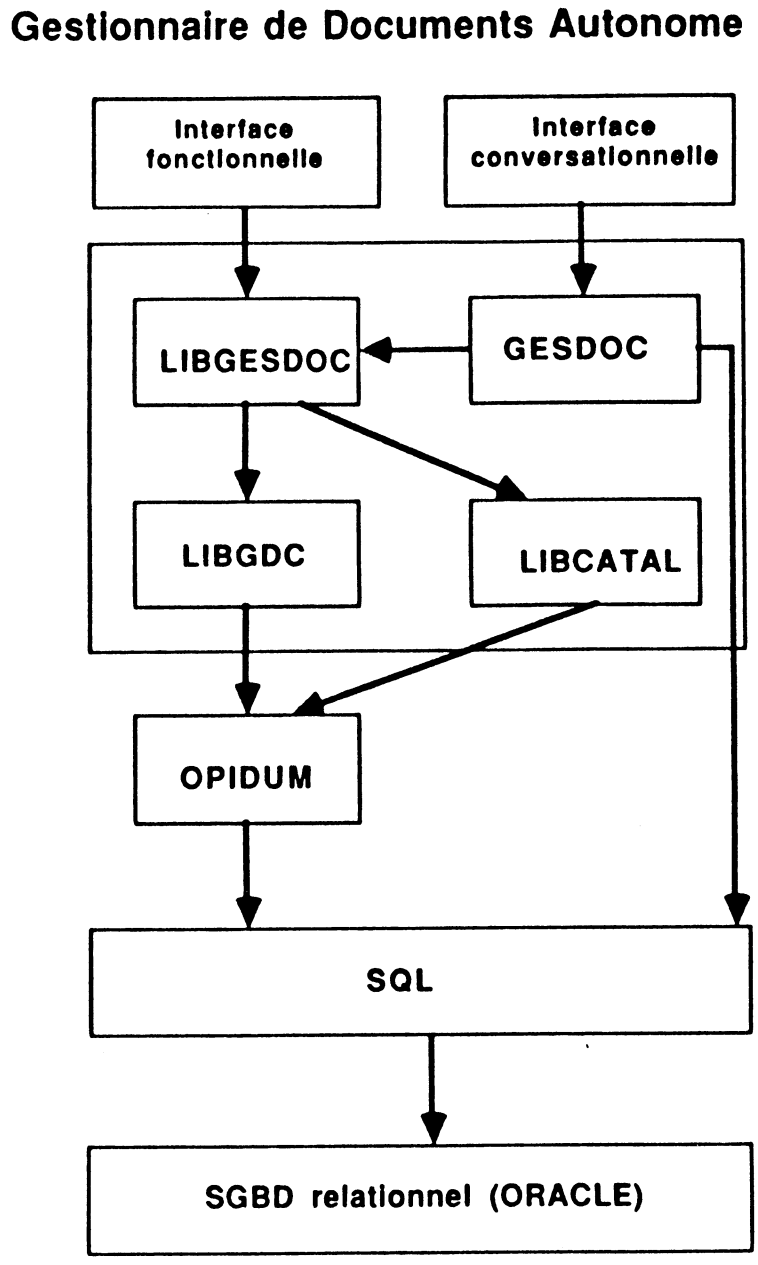

Fig. 5.2 Architecture du GDA

- OPIDUM : interface de haut niveau pour faciliter l'utilisation d'ORACLE.

5.4 Services offerts

Les services offerts par ce prototype sont illustrés par la figure 5.3, il s'agit principalement de : l'internalisation, l'externalisation, la sélection, la projection des attributs du profil et la projection des éléments logiques.

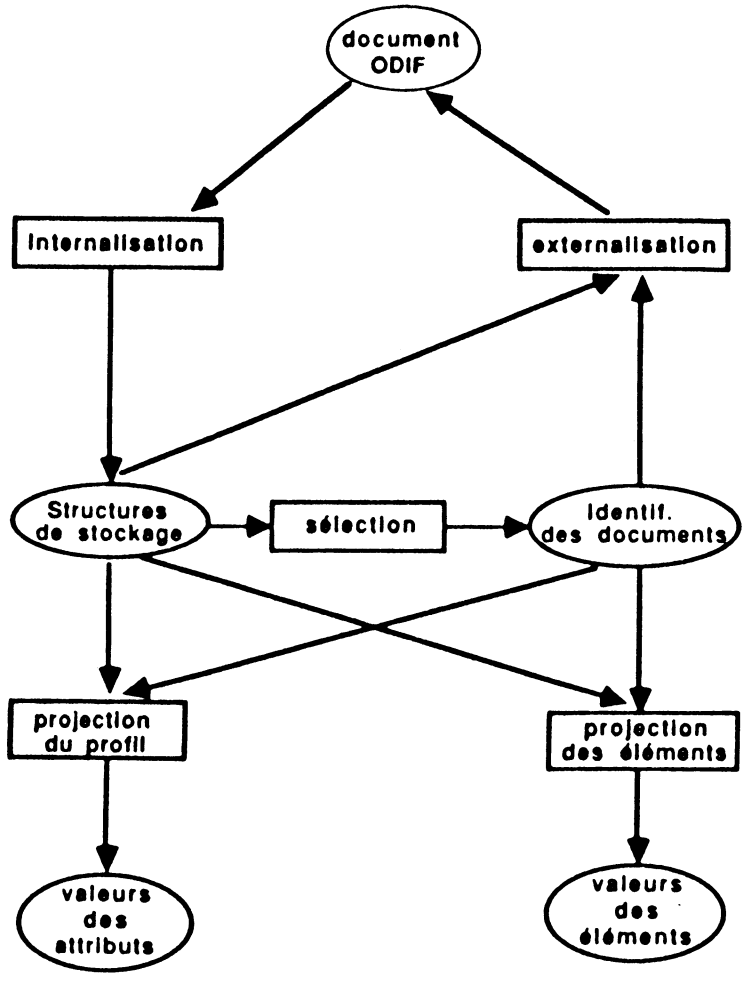

Fig. 5.3 Services offerts

Le service d'indexation est présenté comme faisant partie de l'internalisation mais il peut être offert distinctement après

l'internalisation du document.

Nous allons décrire les deux services les plus caractéristiques du prototype : l'internalisation et la sélection.

5.4.1 Internalisation

Le service d'internalisation est complexe parce qu'il exige (i) le décodage et l'extraction des attributs du profil et portions de contenu (ii) l'identification des éléments logiques et des chemins logiques et (iii) la vérification structurelle dans le cas des environnements fermés ou partiellement ouverts (voir figure 5.4).

Pendant l'internalisation d'un document sont réalisés le stockage des attributs du profil et du contenu original du document ainsi que l'indexation des portions de contenu, des éléments logiques et des chemins logiques.

5.4.2 Sélection

Le service de sélection offert par le GDA prend en compte la structure logique des documents. Les exemples de requêtes ci-dessous sont donnés dans un formalisme type SQL étendu. Ils illustrent le service (fonctionnalité) de sélection. Toutes ces requêtes sont à l'heure actuelle réalisées par des séquences d'appels de fonction C depuis l'interface fonctionnelle décrite dans l'annexe 3.

5.4.2.1 Accéder/Sélectionner les éléments logiques à tous les niveaux

On veut répondre à des questions du genre :

Quelles sont les photos dont la légende parle de la Joconde ?

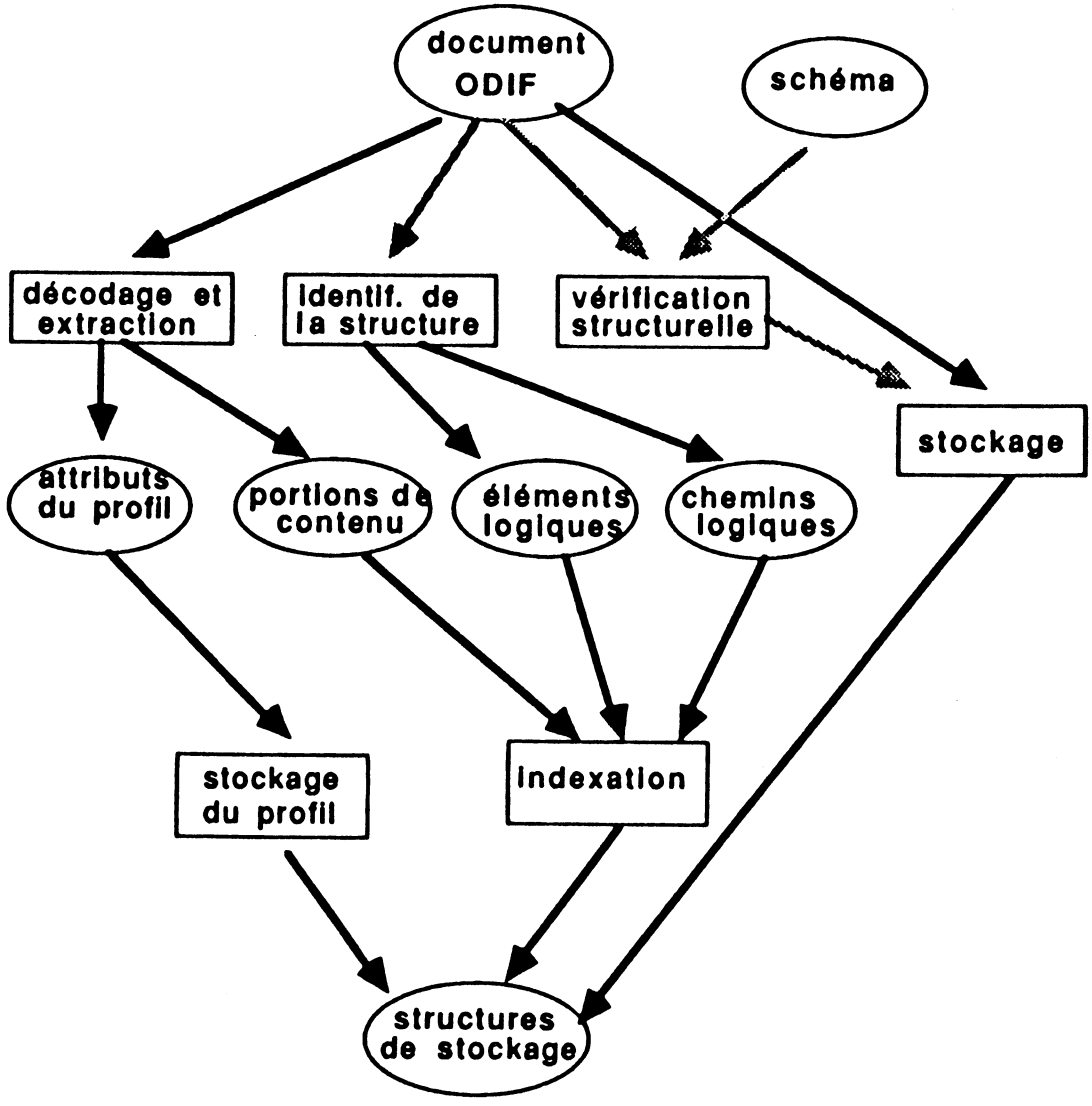

Fig. 5.4 Internalisation

Dans le cas où l'on connaît la structure logique du document, cette question prend une forme précise, par exemple :

SELECT Guide.Peinture.Figure.Photo
FROM Guide
WHERE Guide.Peinture.Figure.Légende
TALKS ABOUT "La Joconde" ;

Concernant la clause SELECT, il y a un problème de mise en page dans le cas des documents structurés en ODIF.

Cependant, même dans ce cas, nous pouvons identifier, et donner comme résultat, toutes les portions de contenu associées à un élément logique. Ceci permet une bonne localisation des réponses, ce qui à notre avis est indispensable.

La notion de rang logique permet d'indiquer précisément les éléments logiques qui répondent à la question, avec leur chemin d'accés.

5.4.2.2 Accéder/Sélectionner les éléments logiques indépendamment du niveau

On remarque qu'il est raisonnable de penser que dans un grand nombre de cas, la structure logique des documents, et même la structure générique, ne sera pas bien connue. Il est donc clair qu'il est particulièrement intéressant de pouvoir manipuler les éléments logiques indépendamment des niveaux hiérarchiques :

SELECT Guide ... Figure.Photo
FROM Guide ... Figure
WHERE Guide ...Figure.Légende TALKS ABOUT "La Joconde" ;

La Clause < *FROM Guide ... Figure* > assure que le chemin dans le SELECT est le même que celui dans la clause WHERE.

L'avantage principal de ce type de requête est que l'utilisateur n'est pas obligé de connaître complètement le schéma pour accéder à certains éléments logiques. Dans ce cas spécifique il suffit de connaître les éléments *Guide* et *Figure* ainsi que les éléments ***Légende*** et *photo* associés à l'élément *Figure* pour sélectionner la *photo* de "La

Joconde".

5.4.3 Ordre entre éléments répétitifs

Soit la requête :

SELECT RANG(Peinture)
FROM Guide.Peinture
WHERE Guide.Peinture TALKS ABOUT "La Joconde" ;

En ce qui concerne l'ordre entre éléments répétitifs, on peut adopter la convention suivante : par défaut, tous les éléments répétés sont ordonnés; l'ordre est l'ordre du document original. Ainsi, on définit la fonction RANG *(Peinture)* qui donne le rang d'un élément logique *Peinture*, donné par rapport à son père (dans ce cas *Guide*).

Il est important de noter que l'ordre d'un élément logique répétitif n'a de sens que par rapport à l'élément parent direct dans la structure logique complète du document. RANG (*photo*) ne sera donc pas défini. La désignation du ième élément d'une fonction peut se faire par un indice entier :

Peinture[i] , Chapitre[i], etc...

Plus généralement, l'ordre est nécessaire pour certains traitements de la structure logique. Concrètement, l'ordre de répétition de chaque élément à chaque niveau permet de désigner exactement le chemin adressant l'élément logique. Cette adresse est unique parce que la structure logique est un arbre. Une réponse possible à la question concernant la *Joconde* pourrait être :

Guide . Peinture[10] . Figure[1] . Photo[1]

Pour les éléments *Figure* et *Photo* l'indication d'ordre est facultative parce que ils sont monovalués.

5.5 Le gestionnaire relationnel GESDOC

5.5.1 L'interface relationnelle

La relation privilégiée de cette Interface est la relation de stockage DOCUMENT dans laquelle chaque nuplet décrit une occurrence de document, avec son contenu complet (au sens du Contenu ODA), et l'exportation des attributs du Profil (ou au moins d'une partie d'entre eux).

DOCUMENT (Sdoc, Contenu, Résumé, Titre, Sujet, Date, ...)

- Sdoc : Identificateur unique du document, géré par le Gestionnaire au moyen d'une relation SURROGATE.

- Contenu : Contenu complet du document, sous son format d'origine (il s'agit en fait ici d'une clé externe renvoyant à une table de contenus CONTENU, et à un système d'indexation utililisant les tables internes INDCONT et INDABS).

- Résumé, Titre, Sujet, Date, ... : attributs du Profil. Ces attributs, remplis au moment de l'internalisation du document, ont la particularité d'être visibles par l'application, mais non modifiables. Plus précisément, le moyen normal pour les modifier est de créer (internaliser) un autre document. On peut autoriser la modification directe d'un attribut, mais la cohérence avec le contenu du document n'est alors plus garantie.

La liste des attributs du Profil est actuellement figée, mais il est possible de la modifier.

Le Contenu peut être partagé par plusieurs documents, mais un Contenu n'existe que dans la mesure où il est lié à un document.

On peut créer des spécialisations de premier niveau (cf. Guide defini dans la figure 2.4 du chapitre 2).

Il existe par ailleurs trois relations mono-attributs AUX1, AUX2, AUX3, qui servent à stocker temporairement des surrogates de documents lors de l'exécution de fonctions ensemblistes telles que Select_Doc.

Ainsi, l'interface se présente comme un ensemble de fonctions utilisant l'interface SQL du SGBD. Ces fonctions ne sont pas appelées directement par l'utilisateur, mais par l'application. On distingue trois niveaux :

- Le niveau administrateur de la Base.
- Le niveau utilisateur où l'on manipule, en général, des ensembles de documents.
- Le niveau programmation où l'on manipule des documents un par un, ou bien des contenus.

Une interface conversationnelle est disponible par ailleurs : elle permet d'exécuter soit des commandes SQL classiques, soit des commandes spécifiques qui correspondent aux fonctions décrites des deux premiers niveaux.

5.6 Conclusions

Nous avons implémenté un Gestionnaire de Documents Autonome qui a été expérimenté sur le SGBD ORACLE afin d'étendre ce dernier par rapport aux fonctionnalités de stockage et d'accès aux documents structurés. On peut ainsi déclarer un type Document et le gérer de manière intégrée au SGBD.

L'utilisation simultanée d'autres standards qu'ODA (TROFF, GRIF, SGML, WORD) est prévue à condition d'écrire les procédures d'internalisation spécifiques à chaque standard (ODA est certainement le plus difficile à décoder).

Cette réalisation est parfaitement compatible avec une approche multimédia. Les images sont stockées comme des paquets de bits, mais rien n'interdit de construire un module spécifique de traitement (Le fait d'avoir préservé le codage original du document permet cette souplesse).

CONCLUSIONS

CONCLUSIONS

Dans cette thèse nous avons abordé les problèmes posés par la modélisation, la manipulation, le stockage et l'accès aux documents structurés. C'est-à-dire que nous avons essayé de couvrir divers aspects formels, logiques et physiques de ces objets complexes.

Ce travail a débuté dans le cadre du projet ESPRIT DOEOIS orienté vers la conception, la mise en oeuvre et l'évaluation d'un Serveur d'Information Bureautique (SIB). Le SIB développé a exigé beaucoup d'efforts dans plusieurs domaines, en particulier dans l'étude du stockage et de l'accès aux documents reprise dans cette thèse.

Nous nous sommes limités à développer un gestionnaire élémentaire de documents et une simple méthode de stockage pour permettre une expérimentation et la mise en oeuvre du SIB. Nous avons fait aussi des choix qui préservent des possibilités d'extensions futures : acceptation de standards différents d'ODA, prise en considération de la structure logique non seulement pour l'accès mais également pour le classement des documents (spécialisation), etc.

Notre approche concrète initiale a mis en évidence les problèmes essentiels liés à l'utilisation d'un modèle fonctionnel inapproprié aux documents, d'un formalisme insuffisant et d'un choix de méthode de stockage et d'accès conduisant à une prolifération des structures physiques et à des pertes de sémantique. Ces problèmes sont consécutifs à une gestion des documents hors de la structure logique.

Parallèlement à la mise en oeuvre effective du SIB, nous avons étudié les problèmes généraux des objets complexes. Cette étude nous a permis de constater que la plupart des travaux sur les objets complexes sont également applicables aux documents et que la plupart des systèmes qui traitent des documents prennent en considération la structure logique de ceux-ci. Néanmoins, le but et la façon d'utiliser cette structure varient profondément. Par exemple, le système Mentor-Rapport [Mel86] utilise une structure logique pour appliquer une stratégie de système fermé et imposer, lors de la création d'un document, une certaine standardisation. Ce système est probablement trop contraignant dans beaucoup de situations mais peut être très utile lorsqu'on souhaite discipliner la création des documents dans une entreprise ou pour la création de documents techniques : l'utilisateur ne peut pas changer la structure établie au départ. D'autres systèmes appliquant des stratégies moins fermées voire très ouvertes permettent à l'utilisateur final de définir en toute liberté une structure logique au moment de la création d'un document : par exemple, Grif [Qui87].

Dans le domaine des Bases de Données Documentaires, nous distinguons trois situations : (i) aucune structure logique générique n'est insérée dans le schéma de la base, (ii) la structure logique générique est partiellement insérée dans le schéma de la base et (iii) la structure générique est complètement insérée dans le schéma de la base. Le premier cas permet l'application d'une stratégie de système

ouvert caractérisant l'acceptation de tous les documents indépendamment de leur structure logique spécifique qui est engendrée selon une structure logique générique d'une classe. Le deuxième cas permet l'application d'une stratégie de système partiellement ouvert qui offre la possibilité de travailler dans un environnement acceptant des documents qui ne sont pas complètement connus a priori structurellement. Enfin, le dernier cas caractérise un environnement fermé qui ne prend en charge que des documents complètement spécifiés dans le schéma de la base. Il est important de remarquer que les deux derniers cas exigent des mécanismes pour comparer deux schémas afin de savoir si le domaine de l'un est inclus dans le domaine de l'autre et/ou pour vérifier si la structure logique spécifique du document à stocker est cohérente avec la structure générique insérée dans la base. Comme nous nous plaçons dans un environnement ouvert, cet aspect n'a pas été approfondi dans ce document.

Les standards tels qu'ODA/ODIF et SGML permettent d'échanger des documents conjointement avec les informations sur les structures logiques.

Dans notre thèse nous effectuons des propositions concrètes sur les trois points suivants :

- Modélisation : A ce niveau, nous avons défini un modèle et une algèbre associée où nous avons introduit des concepts importants pour la modélisation et le traitement des documents. Il est vrai que la modélisation devient fondamentale si on se trouve dans un environnement fermé ou partiellement ouvert. Une originalité du modèle consiste dans la séparation des aspects atomiques, composés, monovalués et multivalués des éléments de la structure

logique des documents. Dans notre modèle DSB, un élément atomique peut être aussi multivalué, ce qui est interdit dans le modèle NST. Nous avons ajouté un nouveau type atomique MULT qui offre la possibilité de traiter le document comme une unité, sans aucune structuration, ainsi qu'une certaine souplesse pour le transformer en un élément composé lorsqu'on reconnaît quelques éléments de sa structure logique. De plus, l'indication du type MULT pour certains éléments facilite le choix d'un schéma physique capable d'indexer le contenu textuel de tels éléments. Nous proposons la définition de deux nouvelles valeurs (la valeur nulle "**noseqniv**" et la valeur spéciale "**vm**") pour faciliter le choix d'un schéma physique efficace et éviter le problème d'appauvrissement sémantique du schéma physique concerné.

- Manipulation : Les applications bureautiques sont par nature dynamiques et en général peu structurées. Ces caractéristiques renforcent l'approche des systèmes faiblement structurés qui n'ont pas besoin de traiter une structure logique très fine. Nous ne nous sommes pas intéressés à la création des documents mais plutôt au stockage et à l'accès aux documents dans un SGBD. La structure logique que nous proposons ici sert pour le stockage, l'accès et le classement des documents. La création est dédiée à une tâche d'éditeur. Il est important de remarquer que nous avons proposé des mécanismes de typage permettant d'adapter le SGBDD à un des trois environnements de traitement de documents : ouvert, partiellement ouvert et fermé.

- Conception physique : Nous proposons plusieurs options de schémas physiques utilisant comme structure des tables plates. Ces tables sont adaptables à une structure d'accueil spécifique telle que

SGBD relationnel, réseau ou hiérarchique, etc.

Perspectives

Aujourd'hui, il serait souhaitable de prolonger ce travail selon quatre axes principaux.

1. Modélisation : Dans notre algèbre DSB, il faudrait compléter la formalisation de certaines opérations manipulant des données multimédia non textuelles. Nous pensons qu'il faut poursuivre une recherche sur l'utilisation de valeurs nulles appropriées aux documents. Par conséquent, il devient indispensable de choisir et d'évaluer complètement des logiques à plusieurs valeurs (vrai, faux, valeur inconnue, noseq, noseqniv, etc) et de redéfinir les opérateurs logiques (ET, OU et NON) et ensemblistes (appartenance, inclusion, etc.) en prenant en considération ces logiques. La formalisation de nouveaux opérateurs liés aux valeurs nulles (**isnoseqniv**, par exemple) est nécessaire. De plus, il faut étudier la représentation machine adéquate de ces valeurs nulles.

2. Manipulation : L'utilisation de la technique d'indexation de signature par codage superposé pour la recherche textuelle actuellement mise en place peut être améliorée. Récemment, Wong [Won89] a publié un résultat sur la comparaison (dans un environnement de "large data base" avec des fichiers de l'ordre d'un gigaoctets) entre cette technique et celle de la signature par codage juxtaposé ; cette dernière technique s'est avérée plus performante pour la recherche et plus économique pour l'occupation physique.

3. Conception physique : Il est envisageable d'étudier la conception physique de la base de documents en utilisant d'autres structures d'accueil telles que des fichiers, des systèmes réseaux ou hiérarchiques.

4. Expérimentation : Nous avons réalisé un gestionnaire de documents pour l'implémentation du prototype OIS et un gestionnaire de documents autonome s'appuyant sur nos propositions. Il faut maintenant utiliser une vraie base de documents pour tester le gestionnaire autonome.

Une autre perpective à long terme est la reconnaissance automatique de la structure des documents à partir de documents sous forme imprimée ou sous forme électronique. Cet axe de recherche est prometteur car on peut ainsi faire entrer ces documents dans des systèmes tels que SGBDD, ou Edition de Documents Structurés qui prennent en considération leurs structures logiques. L'idée originale de la reconnaissance automatique des structures d'un document a été proposée par Verges et Verjus [VV73]. Nous pouvons trouver de nombreux travaux, dans des domaines très divers, ayant cet objectif : le domaine de la linguistique essaie de déduire la structure à partir d'une analyse structurelle du contenu du document en étudiant l'organisation du discours, alors que le domaine de la reconnaissance de formes propose de reconstituer la structure logique d'un document à partir de sa mise en page, etc. Bien entendu, la lecture optique est une des techniques utilisées dans ces domaines.

Pour conclure, nous considérons que notre travail se trouve dans l'axe de recherche actuel qui essaie de définir un modèle et une algèbre associée pour le traitement de documents structurés. L'algèbre spécifiée dans cette thèse vise à utiliser les résultats

pratiques du domaine pour accéder aux documents en prenant en considération la structure logique de ces documents. Le modèle et l'algèbre définis peuvent être utilisés pour le classement des documents par spécialisation. Il est vrai que dans l'approche de système ouvert le stockage de tous les documents de nature très différente n'utilise pas ce type de modèle : ceci équivaut à stocker des documents sans les associer à un schéma de base de données.

Les structures codées dans le document ODA/ODIF restent cachées des utilisateurs. Le seul moyen d'y accéder est par le biais des mécanismes d'indexation décrits dans le chapitre 4.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[AB84] ABITEBOUL S. and BIDOIT, N., Non First Normal Form relations : an algebra allowing data restructuring. Rapport de Recherche. No 347, INRIA, Novembre 1984.

[AB87] ABITEBOUL S. and BEERI C., On the Power of Languages for the Manipulation Complexes Objects. Abstract dans les proceedings of the International Workshop on Theory and Applications of Nested Relations and Complexes Objects, Darmstadt, RFA, 1987.

[ABC87] ADIBA M., BUI QUANG N. et COLLET C., Aspects Temporels, Historiques et Dynamiques dans le Bases de Données. Techniques et Sciences de l'Informatique, Vol. 6, N. 5, 1987.

[ACP88] ANDONOFF E. et al. Structuration Dynamique des Documents Multimédia: Application à la Conception Incrementale des Cours de Formation. Actes du congrès INFORSID 88, La Rochelle - France, 8-10 Juin 1988,

21-43.

[AFQ88] ANDRE J. et al., Structured Documents. Cambridge University Press Publ. Comp., 1988.

[AG87] ABITEBOUL Serge et GRUMBACH Stéphane, Bases de Données et Objets Structurés. Technique et Science Informatiques v.6 n.5, 1987.

[AH86] ABITEBOUL S. and HULL R., Restructuring of Complex Objects and Office Forms. Proc. of the International Conf. on Database Theory, Rome, Italy, 1986.

[AJN88] ANIORTE Philippe et al., Bases d'Informations Généralisées un Système pour la Gestion de Documents Complexes. Actes du congrès INFORSID 88, La Rochelle - France, 8-10 Juin 1988, 1-19.

[AP89] AMGHAR Y. et PINON J.M., Un Editeur de Documents ODA : Une Application de l'Approche Objet. Proceedings of the Workshop on Object-Oriented Document Manipulation, Rennes, France, mai 1989, 52-65.

[Bat88] BATORY D. S., GENESIS : A Reconfigurable Database Management System. IEEE 1988.

[BCP88] BENSADOUN O. et al. La Structuration Dynamique de Documents Volumineux. Actes de la IVème Journée Bases de Données Avancées, Bénodet-France, Mai 1988, 19-39.

[BCP89] BENSADOUN O. et al. Aspects dynamiques dans les bases documents. Actes de la Vème Journée Bases de Données Avancées, INRIA, Genève-Suisse, septembre 1989, 291-307.

[BCR87] BOUZEGHOUB Mokrane et al., Conception de Bases de Données: Etat de l'Art. Technique et Science Informatique, Juillet 1987.

[BD89] BENZAKEN Veronique et DELOBEL Claude, Regroupement d'Objets sur Disque dans un Système de Bases de Données Orienté-Objet. Actes de la Vémes Journées Bases de Données Avancées, INRIA, Genève - Suisse, septembre 1989, 309-329.

[BDH89] BRIAND H., DUCATEAU C., HEBRAIL Y. et al., Une Démarche Générale de Conception d'une Base de Données du Niveau Logique au Niveau Physique. MBD, No.12, août 1989, 65-81.

[BKK87] BANERJEE J., KIM W. and KORTH H., Semantics and Implementations of Schema Evolution in Object-Oriented Database. Proceedings of the ACM/SIGMOD Annual Conference on Management of Data, San Francisco, California, May 1987.

[BM72] BAYER R. and MCCREIGHT E., Organisation and Maintenance of Large Ordered Indexes. Acta Informatica, Vol. 1 No.3, 1972, 173-189.

[BM85] BLAIR D. and MARON M. E., An Evaluation of Retrieval Effectivinees for a Full-Text Retrieval. Communication ACM Vol. 28 No.3, March 1985, 289-

299.

[BM86] BOGO G. et MARTIN Jean Pierre, Stockage dans une Base de Données de Documents Structurés et Multimédia. Actes des journées d'étude AFCET Informatique, Dijon-France, Novembre 1985, 157-178.

[BP83] BODARD François et PIGNEUR Y, Conception assistée des applications informatiques 1 - étude d'opportunité et analyse conceptuelle. Masson, Paris, 1983.

[BRS82] BANCILHON F. et al., Verso: A Relationnal Back End Database Machine. Proc. on the International Workshop on Database Machine, San Diego, 1982.

[CCZ83] CHRISMENT C, CRAMPES J.B. and ZURFLUH G., The BIG Project. Proceedings of the 2nd International Conference on Data Base, DCEN S.M. and HAMMERSLEY P. editors, CAMBRIDGE, 1983.

[CD86] CAREY M. and DeWitt D., The Architecture of the EXODUS Extensible DBMS. IEEE 1986.

[Che76] CHEN P.P., The Entity-relationship Model-Toward a Unified View of Data. ACM Transaction Database System 1, 1, Mar, 1976, 9-36.

[Chr83] CHRISTODOULAKIS S., Acces Files for Batching Queries in Large Information Systems. Proceedings of ICOD II, August, 1983.

[Chr85] CHRISTODOULAKIS S., Office Filling. In: D. Tsichristzis (ed.), Office Automation, Springer 1985,67-89.

[Cod70] CODD E. F., A Relational Model of Data for Large Shared Data Banks. Communication ACM, Vol. 13 No. 6, June 1970, 377-387.

[Col87] COLLET Christine, Les Formulaires Complexes dans les Bases de Données Multimédia. Thèse de Docteur de L'Université Scientifique et Médicale de Grenoble, USTMG, Novembre 1987.

[Com79] COMER D., The Ubiquitous B-Tree. ACM Computing Surveys, Vol. 11 No. 2, june 1979, 121-137.

[CYT86] CHRISTODOULAKIS S. et al., Office Document Retrieval in MULTOS, ETW 1986.

[DA82] DELOBEL Claude et ADIBA Michel, Bases de Données et Systèmes Relationnels. DUNOD Informatique, Edition 1982.

[Dat81] DATE C.J., An Introduction to Database Systems, Vol 1. Addison-Wesley, Reading, Mass, 1981.

[DeL88] De LIMA José Valdeni, Um Estudo Sôbre o Tratamento da Estrutura Lógica de Documentos. Proceedings of the XIV Conferencia Latino Americana de Informática et 17avas. Jornadas Argentinas de Informática, Buenos Aires - Setiembre 1988, 402-416.

[DeL89] De LIMA José Valdeni, Traitement de la Structure Logique de Documents Multimédia. Proceedings of the VII Convocatoria Bianual de la Convención Informática Latina. Barcelona - Marzo de 1989, 580-598.

[DGL89] De LIMA José Valdeni, GALY Henri et LOPEZ Mauricio, Integração da Estrutura Lógica de Documentos em SGBD. Proceedings of the IX Conferencia International de la Sociedad Chilena de Ciencia de la computación e XV Conferencia Latino Americana de Informática, Santiago - Chile, julio 1989, 190-202.

[DG88] DESHPANDE Anand and GUCHT Dirk Van, An Implementation for Nested Relational Databases. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 318-330.

[DG89] De LIMA José Valdeni et GALY Henri, Un Exemple d'Intégration des Documents Multimédia aux SGBD : l'Intégration du Modèle ODA à un Modèle Fonctionnel. Proceedings of the Workshop on Object-Oriented Document Manipulation, Rennes, France, mai 1989, 101-112.

[ECM85] ECMA/TC29/85 Standard ECMA-101 Office Document Architecture. September 1985.

[FBC87] FISHMAN D. et al., IRIS : an Object-Oriented Database Management System. ACM TOIS, Vol. 5, N. 1, janvier 1987.

[FC85] FALOUTSOS C. and CHRISTODOULAKIS S., Access Methods for Documents. In: D. Tsichristzis (ed.), Office Automation, Springer 1985, 317-337.

[FNP79] FAGIN R., NIEVERGELT J., PIPPENGER N. and STRONG H.R., Extensible Hasching - A Fast Acces Method for Dynamic Files. ACM TODS, Vol.4 No.3,

September 1979, 315-344.

[FT83] FISCHER P. and THOMAS S., Operators for Non-First-Normal-Form Relations. Proceedings of the 7th International Computer Software Applications Conference, Chicago, 1983.

[Gib85] GIBBS S.J., Conceptual Modeling and Office Information Systems. In: D. Tsichristzis (ed.), Office Automation, Springer 1985, 193-225.

[Güt88] GUTING, Ralf Hartmut, Modeling Non-Standard Database Systems by Many-Sorted Algebras. Lehrstuhl Informatik VI, University of Dortmund, West Germany, March 1988.

[GZC87] GUTING, Ralf Hartmut et al., An Algebra for Structured Office Documents. IBM Almaden Research Center, San Jose, California, Raport RJ 5559, March 1987.

[GZ88] GOTTLOB Georg and ZICARI Roberto, Closed World Databases Opened Throught Null Values. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 318-330.

[Hab88] HABRIAS Henri, Le Modèle Relationnel Binaire, Edition Eyrolles, 1988, 302 pages.

[HL82] HASKIN R.L. and LORIE R.A., On Extending the Functions of a Relational Database System. Proceedings of the ACM SIGMOD Conference, 1982, 207-212.

[Hor85] HORAK W., Office Document Architecture and Office Document Interchange Formats: Current Status of

International Standardization. IEEE Computer, October 1985, 50-60.

[Hul86] HULL Richard, Relative Information Capacity of Simple Relational Shemata. Siam journal on COMPUTING, vol.15 No.3 August 1986.

[HY84] HULL R. and YAP C.K., The Format Model : A theory of Database Organization. Journal of ACM, Vol.31, No.3, 1984.

[Jae85] JAESCHKE G., Nomrecursive Algebra for Relations with Relation Valued Attributs. IBM Heidelberg Scientific Center, Report TR 85.03.001, 1985.

[Jim89] JIMENEZ GUARIN Claudia, Opérations d'Accès par le Contenu à une Base de Documents Textuels. Application à un Environnement de Bureau. Thèse de Docteur de L'Institut National Polytechnique de Grenoble, Grenoble, UJF, Juillet 1989.

[JS82] JAESCHKE G. et SHECK H.J., Remarks on the Algebra of Nom First Form Relations. Proc. PODS, Los Angeles, 1982.

[KC88] KIM Won and CHOU Hong-Tai, Versions of Schema for Object-Oriented Databases. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 148-159.

[Ken84] KENT W., Data and Reality, 4ième édition, North-Holland, 1984.

[Ker84] KERKOUBA Dalila, Une Méthode d'Indexation Automatique des Documents Fondée sur l'Exploitation de Leurs Propriétés Structurelles. Thèse de Docteur de Troisième Cycle Informatique, INPG, Novembre 1984.

[KMN89] KOULOUMDJIAN Jacques, MOLL Georges-Henri et NURCAN Selmin, L'Apport de la Programmation Logique et de la Structuration Orientée Objet aux Bases de Données. Modèles et Bases de Données, No.11, Janvier 1989, 21-47.

[Knu73] KNUTH D.E., The Art of Computer Programming. Vol. 3 : Sorting and Searching, Addison-Wesley, Reading, Mass, 1973.

[LFL88] LEE Mavis K. et al., Implementing an Interpreter for Functional Rules. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 218-229.

[Lit80] LITWIN W., Linear Virtual Hashing : A New Tool for Files and Tables Implementation. Proceedings of the Sixth International Conference on Very Large Data Bases, Montreal-Canada, October 1980, 212-223.

[LKD88] LINNEMANN V. et al., Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 294-305.

[LL85] LEE D.L. and LOCHOVSKY F.H., Text Retrieval Machines. In: D. Tsichristzis (ed.), Office Automation,

Springer 1985,339-375.

[LL86] LIMANE Mohamed et LeBIHAN Jean, Un Système de Classement/Recherche de Documents Electroniques pour Poste de Travail Bureautique. Actes du Congrès INFORSID 86, Fontevraud - France 27-30 Mai 1986, 93-115.

[LR82] LLOYD J.W. and RAMAMOHANARAO K., Partal-Match Retrieval for Dynamic Files. BIT, No. 22, 1982, 150-168.

[LR89] LECLUSE Christophe et RICHARD Philippe, Langages Orienté-Objet et Bases de Données : l'Expérience O_2. Actes de la Vème Journée Bases de Données Avancées, INRIA, Genève-Suisse, Septembre 1989, 155-168.

[LRV87] LECLUSE C. et al., O2, an Objet Oriented Data Model. Rapport Technique Altaîr 10-87, septembre 1987.

[LS83] LUN V. and Scheck H., Complex Data Objets : Text, Voice, Image, Can DBMS Manage Then ? Proceeding of the 9th. Conference on VLDB, Florence, Novembre 1983.

[LS88] LYNCH Clifford A. and STONEBRAKER Michael, Extended User-Defined Indexing with Application to Textual Databases. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 306-317.

[LV84] LOPEZ M. and VELEZ F., Modelling and Handling Generalized Data in the TIGRE Project. Report, Centre de

Recherche CII Honeywell Bull, c/o IMAG, St. Martin d'Heres, France, 1984.

[Mar79] MARTIN G.N.N, Spiral Storage : Incrementally Augmentable Hash Addressed Storage. Theory of Computation, Report No. 27, University of Warnick, Conventry, England, March 1979.

[Mel86] MELESE B., Edition de Documents Multi-langages sous Mentor-Rapport. Techniques et Science Informatiques, Vol.4 No.5, 1986, 267-277.

[NP89] NARDOT Jean et PUJOLLE Geneviève, Une interface de type hypertexte pour la gestion d'objets complexes. Proceedings of the Workshop on Object-Oriented Document Manipulation, Rennes, France, mai 1989, 383-386.

[OIS86a] DOEOIS Bibliographical Annex Bibliographical Informations and Standards in OIS: Overview. 1B1 Deliverable, January 1986.

[OIS86b] DOEOIS Review of Data Models and Data Management Systems. 1B2 Deliverable, January 1986.

[OIS86c] DOEOIS External Functional Interface for an Office Information Server. A Description. 2B1 Deliverable, September 1986.

[OIS87a] DOEOIS Data Definition and Data Manipulation Services. OIS-DH-01.01, april 1987.

[OIS87b] DOEOIS Future Office Information Server Functionality. 3A3/3A5 Deliverable, March 1987.

[OIS87c] DOEOIS Document Representation in an Office Information Server. 2B3 Deliverable, 1987.

[OP88] ORLANDIC Ratko and PFALTZ John, Compact 0-Complete Trees. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 372-381.

[PA86] PISTOR P. et ANDERSEN F., Principles for Designing a Generalized NF2 Data Model with an SQL-type Language Interface. IBM Heidelberg, 1986.

[PBC80] PFALTZ J.L., BERMAN W.H. and CAGLEY E.M., Partial Match Retrieval Using Indexed Descriptor Files. Communications of the ACM, Vol.23 No.9, September 1980, 522-528.

[Pin86] PINON J.M., Analyse Préalable de la Documentation Technique d'une Entreprise en Vue de son Intégration dans une Base de Données Multimédia. Actes du Congrès INFORSID 86, Fontevraud - France 27-30 Mai 1986, 93-115.

[PRL88] PASCOT Daniel, RIDJANOVIC Dzenan et LEGENDRE pierre, Express-MLD : Une Approche de Développement Rapide Centré sur le Modèle Logique de Données. MBD, n.10, juillet 88, 17-35.

[PRT89] PUCHERAL Philippe, RENARD sylvie et THEVENIN Jean-Marc, DBGRAPH : Une structure de données pour les SGBD en mémoire principale. MBD, n.13 et 14, octobre 89, 17-35.

[PTS88] PUCHERAL P., THENEVIN J.M. et STEFFEN H., GEODE : Un Gestionnaire d'Objets Extensible Orienté Grande Memoire. Actes du Bases de Données de Troisième Génération, Benodet, juin 1988.

[Qui87] QUINT Vincent, Une Approche de l'Edition Structurée des Documents. Thèse de Docteur de l'Université Scientifique, Technologique et Medicale de Grenoble, UJF, Mai 1987.

[Rab85] RABITTI F., A Model for Multimedia Documents. In: D. Tsichristzis (ed.), Office Automation, Springer 1985,226-249.

[Rah84] RAHMAN Nor'rini Bte Abdul, Méthodes d'Accès à des Structures Arborescentes Dans le Cadre d'un SGBD Relationnel. Rapport de DEA, INPG, Juin 1984.

[Riv76] RIVEST R.L., Partial Match Retrieval Algorithms. SIAM Journal of Computing, Vol.5 No.1, March 1976, 19-50.

[RK85] ROTH M.A. and KORTH H.F., Null Values in ¬1NF Relational Databases. Technical Report, Departement of Computer Science, University of Texas at Austin, 1985.

[Rob79] ROBERTS C.S. Partial-Match Retrieval Via the Method of Superimposed Codes. Proceedings of the IEEE, Vol. 67 No. 12, December 1979, 1624-1642.

[RS82] ROWE L.A. and SHOENS K.A., A Form Appilcation Development System. Proceedings of the ACM SIGMOD Conference, June 1982, 23-38.

[Sac84] SACCO Giovanni Maria, OTTER - An Information Retrival System for Office Automation. Proceedings of the ACM conf. on Office Information Systems, June 1984. Pag. 104-112.

[Sam84] SAMET H., The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys V.16, N.2, June 1984.

[Sau89] SAUNDES, John H., A Survey of Object-Oriented Programming Languages. Journal of Object-Oriented Programming, V.1, n.6, mar/apr 1989.

[Sav88] SAVOY Jaques, Le livre électronique EBOOK3. Technique et Science Informatiques v.7 n.5, 1988.

[SD76] SEVERANCE D.G. and DUHNE R.A., A Practitioner's Guide to Addressing Algorithms. Communications of ACM, Vol.19, No. 6, 1986, 314-326.

[Sed87] SEDES Florence, L'Adaptation de l'Approche Bases de Données dans les Systèmes Bureautiques : Structuration Dynamique. Proceedings of the VI Convocatoria Bianual de la Convención Informática Latina. Barcelona - Marzo de 1987.

[Sed88] SEDES Florence, Systèmes de Gestion de Bases de Données et Systèmes Documentaires: Deux Approches Complémentaires. Actes du Congrès INFORSID 88, La Rochelle-France 8-10 Juin 1988, 45-62.

[Shi81] SHIPMAN David W., The Functional Data Model and Data Language DAPLEX. ACM Transactions on

Database Systems, Vol. 6, N. 1, March 1981, 140-173.

[SGR88] STEFFEN H. et al., Los SGBD Extensibles. Proceedings of the XIV Conferencia Latino Americana de Informática et 17avas. Jornadas Argentinas de Informática, Buenos Aires - Setiembre 1988, 316-335.

[SKP88] STONEBRAKER Michael et al., The Design of XPRS. Proceedings of the 14th VLDB Conference Los Angeles, California, Aug - Sep 1988, 318-330.

[SP82] SCHEK H.J. and PISTOR P., Data Structures for an Integrated Data Base Management and Information Retrieval System. Proc. of the VLDB Conf. 1982. 192-207.

[SR86] STONEBRAKER M. and ROWE L., The Design of POSTGRES. Proceedings of the 1986 ACM SIGMOD Conference, Whashington DC, Mai 1986.

[SS86] SCHEK H.J. and SCHOLL M.H., An Algebra for the Relational Model with Relation - Valued Attributs. Information System 11, 1986, 137-147.

[TCL85] TSICHRITZIS D. et al., A Multimedia Filing System. In: D. Tsichristzis (ed.), Office Automation, Springer 1985, 43-65.

[TF82] TEOREY T.J. and FRY J.P., Design of Database Structures, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[Tsi80] TSICHRITZIS D.C., OFS : An Integrated Form Management System. Proceedings of the Sixth

International Conference on Very Large Data Bases, 1980, 161-166.

[Val76] VALLARINO O., On the Use of Bit Maps for Multiple Key Retrieval. Proceedings of Conference on Data Abstration, Definition and Structure, in ACM SIGPLAN Notices (Special issue), 11, March 1976, 108-114.

[Val87] VALDURIEZ Patrick, Join Indices. ACM Transactions on Database Systems, Vol. 12, N. 2, june 87, 218-246.

[VB86] VALDURIEZ P. et BORAL H., Evaluation of recursive queries using join indices. Proc. of 1st International Conference on Expert Database Systems, Charleston, 1986.

[Vel84] VELEZ Fernando, Un Modèle et un Language pour les Bases de Données Généralisées. Thèse de Docteur Ingénieur, INPG Grenoble, 1984.

[Vel86] VELEZ Fernando, Structure du Catalogue TIGRE et Représentation Relationnelle d'un Schéma TIGRE. Rapport de Travail, 1986.

[VL86] VELEZ Fernando and LOPEZ Mauricio, Managing Structured Documents in a Multimedia Database System. Bull Research Center c/o IMAG, 1986.

[VV73] VERGES-ESCUIN S. J. et VERJUS J.P., Reconnaissance automatique des structures des textes en vue de l'édition, RAIRO, October, 1973, 85-120.

[WLL85] WOO C.C. et al., Document Management Systems. In: D. Tsichristzis (ed.), Office Automation, Springer 1985, 21-

40.

[Won89] WONG Kam-Fai, Comments on "A Comparison of Concatenated and Superimposed Code Word Surrogate Files For Very Large Data/Knowledge Bases". Information Processing Letters, No.33, North-Holand 1989/90, 45-52.

[Zlo82] ZLOOF M.M., Office-by-Exemple : A Business Language that Unifies Data and Word Processing and Electronic Mail. IBM System Journal, Vol.21 No.3, 1982, 272-304.

ANNEXES

Annexe 1

COUPLAGE DES MODELES ODA ET FM

Nous allons d'abord décrire succintement le "Fact Model" pour spécifier ensuite le couplage entre le modèle ODA et le modèle FM afin d'obtenir le système couplé OIS.

1.1 Le Fact Model

Le "Fact Model" (FM) est un modèle de type fonctionnel, basé seulement sur deux concepts : Entité ("Entity") et Fait ("Fact").

Le terme "Entité" désigne, selon le contexte, aussi bien un type de données qu'un ensemble d'éléments, ou bien des occurrences.

Les entités globales sont des entités dont le nom est connu dans toute la base. Les entités locales ne sont connues que dans un fait donné. Seules les entités ENTITY et DOCUMENT, ainsi que les entités qui en dérivent, sont des entités virtuelles. Les autres (STRING, REAL, ..., ODA, DOC_ELEM) sont littérales.

EXEMPLE :

Age : Integer(0..120); (1)
Age_Employé : Integer(18..65); (2)
Couleur ('rouge ', 'bleu', 'vert', 'jaune '); (3)

Dept : Entity; (4)
Employé : Entity; (5)

Nous définissons ainsi la spécialisation d'entité :

Lettre : Document; (6)

pour dire que toutes les lettres sont des documents.

Livre : Document - Lettre; (7)

Dans ce dernier cas, tous les livres sont des documents mais ils ne peuvent pas être des lettres. Il peut exister des documents autres que les livres ou les lettres.

Soit les entités globales A et B. On appelle " fait" le couple commutatif :

[A(a1,a2):B, C(c1,c2):D]

Dans le "Fact Model", il est équivalent à la déclaration d'un

ensemble de 2 fonctions réciproques, à savoir :

C (B) soit B : ---C(c1, c2)--->> D

A (D) soit D : ---A(a1, a2)--->> B

Nous écrirons ces fonctions B.C : c1-c2 et D.A : a1-a2 respectivement.

Par défaut, le nom de la fonction (appelée "propriété") est le nom du codomaine - mais on peut définir des noms locaux, nous y reviendrons par la suite : c'est une autre particularité de FM.

Dans le cas le plus général (par défaut), les fonctions sont partielles (a1 ou c1 = 0), multivaluées (a2 ou c2 = *).

Une restriction importante du langage (mais pas du modèle) est que la première fonction de la notation pointée dans FMQL ("Fact Model Query Language") doit être une fonction globale (identifie une classe d'entité virtuelle). La dernière fonction de la même notation doit avoir comme codomaine une classe d'entité littérale.

EXEMPLE :

[Employé, identité(0,1):String(20)]; (8)

Le fait (8) est déclaré d'une façon simplifiée. La déclaration complète de ce fait est : [Employé:Employé, identité(0,1):String (20)].

[Employé, adresse(1,*):Text]; (9)
[Employé, salaire(1,1):Real]; (10)

[Employé, age(0,1):Age_employé]; (11)

[Dept(1,1), identité(1,1):Integer)]; (12)
[Dept(0,*), Employé]; (13)
[Dept, Chef:(0,1):Ingénieur]; (14)
[Dept, budget(1,1):Real]; (15)

1.2 Couplage entre le modèle ODA et le Fact Model

Le couplage entre le modèle ODA et "Fact Model" (FM) est réalisé par l'introduction de l'entité de base ODA et par la création de "faits prédéfinis" entre l'entité de base ODA et l'entité de base Document. Le premier fait prédéfini associe des occurrences de la classe d'entité Document à des valeurs ODA. Les valeurs ODA sont en réalité les occurrences de la classe d'entité de base ODA. La valeur ODA est ainsi considéréee soit comme un objet atomique, soit comme un objet composé, selon la nécessité de manipulation du document concerné. Nous avons gardé les termes en anglais trouvés dans notre référence ODA.

La déclaration de ce fait en FMQL est:

[Document(0,*):Document, content(0,1):ODA] (16)

Ce fait définit la fonction content(Document)-->ODA qui identifie la valeur ODA associée à l'entité Document. Pour éviter les confusions, on préférera parler de "valeur ODA" pour désigner l'occurrence de la classe d'entité ODA, dont la représentation externe

est un document ODA codé au format ODIF. Nous éviterons d'employer le terme "Document ODA" qui pourrait introduire une confusion avec une occurrence de la classe d'entité "Document".

Du point de vue du langage FMQL la fonction content(Document)-->ODA est désignée par "Document.content". La figure 1.1 montre la fonction content.

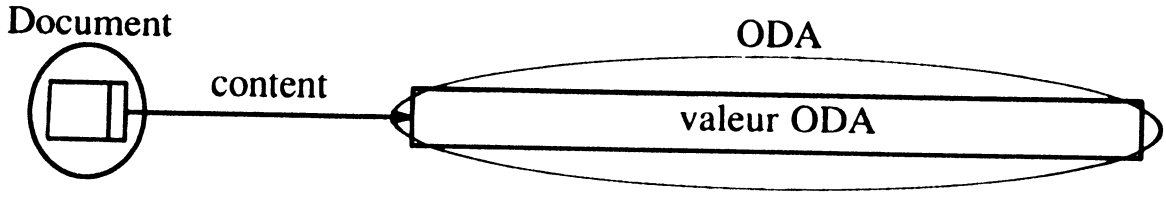

Fig. 1.1 : Fonction content

1.2.1 L'entité de base ODA

L'entité de base ODA contient des "valeurs ODA" qui, à leur tour, englobent d'autres "valeurs" telles que les attributs (fonctions) du profil du document. Les attributs du profil visibles dans la première version de l'OIS sont : authors, title, subject, abstract, documentdate et creationdate. Du point de vue de l'implantation, ces attributs sont exportés (recopiés) vers le serveur et peuvent appartenir en même temps à d'autres classes d'entités de base (voir figure 1.2). Par exemple les attributs title, authors et subject sont aussi du type String.

Une restriction importante sur le type ODA est qu'il n'est utilisable que dans le fait prédéfini [Document(0,*),content:ODA]. Aucune autre entité de type ODA ne peut être utilisée dans le schéma.

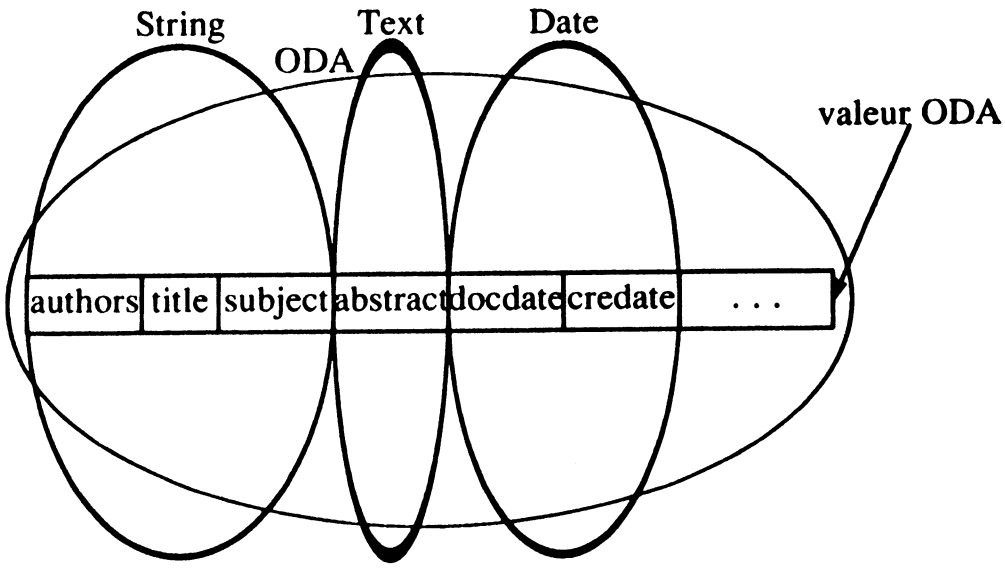

Fig. 1.2 : La valeur ODA et les autres types de base

Si une entité Document n'a pas nécessairement de valeur ODA associée, par contre une valeur ODA doit obligatoirement être associée à une ou plusieurs entités Document.

L'expression "entité" est utilisée dans son sens normal, c'est à dire, à la place de l'expression "occurrence d'un type d'entité". Les expressions "type de base" et "type" sont synonymes des expressions "type d'entité" et "classe d'entité".

1.2.2 Le profil

Pour intégrer la notion de profil au modèle FM dans la première version de l'OIS, nous avons engendré les six faits prédéfinis suivants:

[Document(0,*):Document, authors(0,1):String]; (17)
[Document(0,*):Document, title(0,1):String]; (18)

[Document(0,*):Document, subject(0,1):String]; (19)
[Document(0,*):Document, abstract(0,1):Text]; (20)
[Document(0,*):Document, documentdate(0,1):Date]; (21)
[Document(0,*):Document, creationdate(0,1):Date]; (22)

Ces faits (dont la liste peut augmenter au fur et à mesure des besoins) définissent des fonctions qui permettent la manipulation des attributs du profil de la "valeur ODA". Par exemple, la figure 1.3 montre les fonctions "Document.title" et "Document.abstract" définies par les faits (18) et (20).

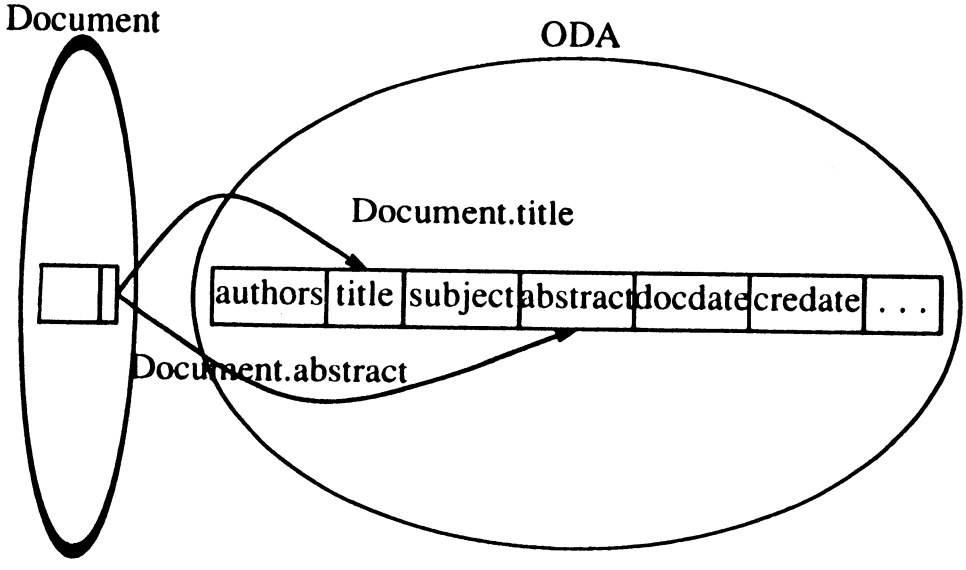

Fig. 1.3 : Fonctions Document.title et Document.abstract

Une valeur ODA après le couplage est visible par la fonction content définie par le fait prédéfini entre le type Document et le type ODA ([Document(0,*),content:ODA]).

En fait, ces attributs font partie de la valeur ODA ce qui empêche de les modifier explicitement. L'unique façon de les modifier est de le

faire implicitement par modification de la "valeur ODA".

1.2.3 La modélisation de la structure logique ODA

La manipulation de la structure logique d'un document ODA est possible par l'utilisation de la classe spéciale d'entités DOC_ELEM et par la déclaration de quelques faits entre la classe Document et cette classe spéciale. La classe DOC_ELEM est spéciale en ce sens que les fonctions réciproques des faits déclarés sur elle sont supprimées.

Pour mieux comprendre cette manipulation nous donnons un exemple, ci-dessous, d'un schéma en FM de la classe d'entités Lettre :

[Lettre, expéditeur:DOC_ELEM];	(23)
[Lettre.expéditeur, nom:DOC_ELEM];	(24)
[Lettre,(destinataire(nom,adresse)):DOC_ELEM];	(25)
Expéditeur : DOC_ELEM;	(26)

Le schéma FM résultant des déclarations (6), (16) et (23 à 26) est représenté dans la figure 1.4.

La déclaration (23) crée la fonction locale expéditeur(Lettre)-->expéditeur.

La déclaration (24) crée la fonction locale nom(DOC_ELEM)-->DOC_ELEM. Une fonction locale exige, lors de sa désignation, la composition, en premier lieu, de la fontion globale d'origine avec d'autres fonctions locales jusqu'à la fonction concernée. Par exemple, pour désigner la fonction nom de l'expéditeur nous écrivons: "Lettre.expéditeur.nom".

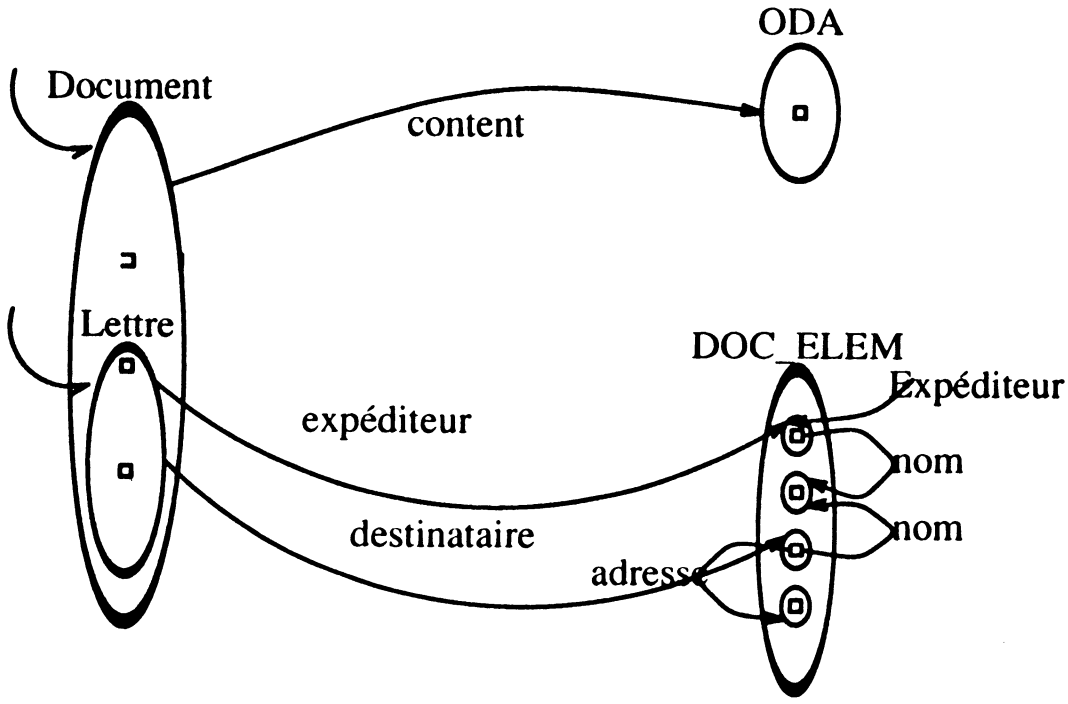

Fig. 1.4 : Structure logique de documents dans FM.

La déclaration (25) est une façon abrégée d'exprimer plusieurs fonctions à partir d'une fonction globale qui, dans notre cas, est la fonction Lettre. Cette déclaration crée les fonctions suivantes:

destinataire(Lettre)-->destinataire
nom(destinataire)-->nom
adresse(destinataire)-->adresse.

La notation pointée est en réalité la composition de fonctions. De cette façon, comme l'unique restriction pour la composition f.g de deux fonctions f et g est que le codomaine de f soit le domaine de g, nous pouvons désigner:

Lettre.destinataire /* destinataire de la lettre comme un objet */

Lettre.destinataire.nom /* nom du destinataire de la lettre */
Lettre.destinataire.adresse /* adresse du destinataire de la lettre */

Finalement, la déclaration (26) crée une classe d'entités Expéditeur et une fonction globale Expéditeur.

Annexe 2

RELATIONS DE STOCKAGE

Pour le SIB et GDA nous avons spécifié et mis en oeuvre une méthode simple de stockage et d'accès applicable aux données conventionnelles et aux documents.

La méthode proposée ici permet le stockage des documents complets à partir d'un fichier ODIF et l'accès à ces documents selon la visibilité définie sur le Modèle Document d'ODA. Cette visibilité uniformise le traitement de stockage et d'accès aux documents.

Un document structuré peut être considéré comme une combinaison structurée de données du type multimédia (des textes, des images, des grafhiques, voix digitalisée, etc). Les données

structurées multimédia et/ou alphanumériques sont regroupées dans la littérature sous les noms de Données Généralisées et d'Objets Complexes.

Le développement d'une méthode de stockage et d'accès applicable à ces types de données est une des tâches importantes dans l'implémentation des SGBD capables de stocker et de manipuler des Données Généralisées (Projet TIGRE), des Serveurs d'Information pour la Bureautique, qui stockent, modifient et restituent des documents (Projet ESPRIT 231:OIS), ou du Génie Logiciel, qui gère des programmes. Dans ce raisonnement, on entend que cette méthode est également essentielle pour la CAO (Conception Assistée par Ordinateur) qui traite des gros volumes d'informations, de structure et de nature très diverses.

Nous commençons par décrire les caractéristiques principales de la méthode dans la section 1 pour, finalement, insister sur les relations de stockage dans la section 2.

2.1 Caractéristiques principales

La présente méthode prend en compte le format d'échange de documents ODA/ODIF et la Recherche Documentaire. La Recherche Documentaire est la recherche par le contenu sur des unités de texte en utilisant la Méthode de la Signature et le Filtrage.

Elle prend en compte tous les documents avec la même Visibilité ODA. C'est-à-dire, le document est considéré comme une entité virtuelle Document avec plusieurs faits prédéfinis en fonction du contenu et attributs du profil qu'on trouve intéressants de stocker dans la première version. Pour de plus amples détails se reporter au document de référence DG89 (Un Exemple d'Intégration des

Documents Multimédia aux SGBD : L'Intégration du Modèle ODA à un Modèle Fonctionnel). L'entité virtuelle Contenu est composée d'une liste d'unités textuelles qui peuvent être les feuilles d'une structure logique complexe. Néanmoins, la première version implémentée (SIB) ne traite pas des structures logiques des documents. L'exemple des Actes de Conférence utilisé dans [Vel84] et [VL86] ne peut pas être stocké avec cette méthode car elle est basée sur ODA/ODIF qui ne traite pas des documents composés d'autres documents.

Le problème d'une possible réorganisation de l'espace physique est laissé de côté car on suppose qu'ORACLE, système utilisé pour implémenter la première version, le fait automatiquement. Une réorganisation semble nécessaire au fur et à mesure que sont exécutées des opérations de suppression et qu'il reste des trous dans la relation. Par exemple, l'archivage d'un document peut être traduit par sa suppression de la base principale pour le stocker dans une base secondaire. L'archivage d'un document est une opération normale d'un environnement bureautique et peut être utilisé dans le Serveur pour réduire la quantité d'informations actives de la base principale afin d'augmenter sa performance.

2.2 Relations de stockage

2.2.1 Stockage du document ODIF original

Le stockage d'un document ODIF original est réalisé par une opération d'internalisation. L'opération d'internalisation du document entraîne un processus de décodage qui permet d'identifier tous les éléments de la structure logique du document, et en particulier les portions de contenu. Si l'internalisation est faite en indexant par la méthode de la signature, il faut pour chaque portion de contenu

reconnue, créer et stocker la signature. L'algorithme ci-dessous montre l'ensemble d'instructions nécessaire à l'internalision du document :

INTERNALISER_DOCUMENT (nom_fichier_odif)
| Obtenir (identificateur_document)
| identifier_attributs du profil (identificateur_document)
| Jusqu'à la fin du fichier odif faire :
| i = **IDENTIFIER_PC**
| Si index_signature alors
| **CREER_SIGNATURE(i)**
| **STOCKER_SIGNATURE**
fin_internaliser

Contrairement à TIGRE, qui éclate le document ODIF, on stocke ici le document original ODIF afin d'effectuer une opération d'externalisation aisée.

La relation qui stocke le document ODIF s'appelle DOCODIF et sa description est la suivante:

DOCODIF(content,ref_count,order_n,codif)

content : identificateur du contenu du document,

ref_count : nombre de documents qui font référence au contenu considéré,

order_n : numéro d'ordre de la liste des morceaux d'un gros document,

codif : document original ODIF complet si moins de 64K. Si le document est plus grand que 64K on utilise un autre tuple de

la même relation avec le numéro d'ordre (order_n) suivant.

Le couple content et order_n est la clé de la relation.

La commande pour créer la relation DOCODIF sur ORACLE est:

```
CREATE TABLE DOCODIF
  (content  NUMBER NOT NULL,
  ref_count  NUMBER NOT NULL,
  order_n  NUMBER NOT NULL,
  codif  LONG)
```

2.2.2 Stockage des références aux abstract et contenu

L'"Abstract" et le Contenu sont les deux entités qui permettent d' appliquer l'opérateur "TALKS ABOUT" dans la première version. C'est-à-dire, qu'elles sont les seuls éléments dans un document sujets à une Recherche Textuelle. En principe, on avait décidé de stocker les références à ces éléments dans une même relation appelée INDCONT. Mais, après quelques analyses de requêtes, ils'est avéré intéressant de retirer les références à l'"Abstract" et de les stocker dans une autre relation appelée INDABS. Cet éclatement de stockage des index diminue la taille de la relation INDCONT, évite une condition supplémentaire (cible égale "Abstract" ou "Contenu") pendant la présélection de la Recherche Documentaire et augmente d'autant la performance de l'accès à l"Abstract". Les relations INDCONT et INDABS sont décrites ci-dessous.

INDCONT(content,offset,length,signature)

INDABS(content,offset,length,signature)

content : surrogate du contenu,

offset : déplacement à partir du début du document ODIF. Ce déplacement permet la localisation d'un élément sujet à une opération de filtrage. il indique le nombre d'octets à ignorer au début du document ODIF,

length : taille de l'élément concerné en octets, Signature : signature d'élément.

Le couple content et offset forme la clé des relations.

La première version n'acceptera pas la recherche documentaire sur des éléments de type Text en dehors des documents.

On a prévu une taille maximale de 240 caractères pour la signature pour éviter un champ long d'ORACLE. Cette prévision me semble réaliste car la méthode de la signature utilise 300 bits par bloc.

La commande pour créer la relation INDCONT est semblable a la commande pour créer la relation INDABS. On ne montre ici que la commande ORACLE pour créer la relation INDCONT:

```
CREATE TABLE INDCONT
  (content  NUMBER NOT NULL,
  offset NUMBER,
  length NUMBER,
  Signature CHAR(240))
```

2.2.3 Stockage des attributs du profil

Les attributs du profil sont dupliqués. C'est-à-dire qu'une légère redondance des données est le prix à payer pour les intégrer au FM et obtenir ainsi une manipulation plus commode et un accès plus performant à ces attributs. Les attributs du profil sont stockés codifiés dans le champ codif de la relation DOCODIF et, d'une façon plus manipulable, sont stockés dans la relation DOCUMENT décrite ci-dessous:

DOCUMENT(document,content,title,subject,abstract,documentdate, creationdate,authors)

document : surrogate du document,

content : surrogate du contenu du document,

title : titre du document,

subject : sujet du document,

abstract : résumé du document représenté par un entier: NULL si l'abstract n'existe pas, 0 si l'abstract existe sans contenu et la valeur du content si l'abstract existe avec un contenu.

documentdate : date associée au document,

creationdate : date de création du document,

authors : liste des auteurs séparés par une virgule.

La commande pour créer la relation DOCUMENT sur ORACLE est la suivante:

```
CREATE TABLE DOCUMENT
```

(document NUMBER NOT NULL,

content NUMBER,

title CHAR(100),

subject CHAR(100),

abstract NUMBER(1,0),

documentdate CHAR(20),

creationdate CHAR(20),

authors CHAR(240))

Les attributs title,abstract,documentdate, creationdate et authors sont figés dans le modèle ODA.

Le stockage de l'attribut author comme une chaine de caractères résultant de la concaténation de noms de plusieurs auteurs séparés par virgule ne pose pas de problème car l'identification d'un auteur spécifique est facilement résolue par l'opérateur de type LIKE. Si le nom de l'auteur est codé en ODIF comme Nom,Prenom, avec virgule, ceci peut poser quelques problèmes lors de la recomposition ODIF mais le document est déjà stocké en ODIF pour éviter cette recomposition.

2.2.4 Stockage du premier surrogate libre

Le stockage du premier identificateur libre est réalisé dans la relation SURODIF. Cet identificateur est géré par le Gestionnaire de Documents et est utilisé dans une éventuelle internalisation d'un contenu ODA.

SURODIF(free_content)

free_content : surrogate du contenu du document libre.

La clé de la relation est free_content.

La commande pour créer la relation SURODIF sur ORACLE est la suivante:

```
CREATE TABLE SURODIF
    (free_content NUMBER NOT NULL)
```

Annexe 3

INTERFACE FONCTIONNELLE DU GESTIONNAIRE DE DOCUMENTS AUTONOME

L'interface fonctionnelle offerte permet l'intégration avec d'autres systèmes par le biais d'appels de fonctions. Ces fonctions sont listées ci-dessous :

3.1 Administration de la base

Pour utiliser le Gestionnaire, il faut demander au préalable l'installation de l'environnement nécessaire. Cet environnement se compose de quatre relations visibles DOCUMENT, AUX1, AUX2, AUX3 - et d'autres relations non visibles : les relations de stockage

(SURROGATE, CONTENU, INDCONT et INDABS) et les relations catalogues (réduites ici au strict nécessaire : la liste des attributs du Profil ELEMENTS).

- **Init (Nom_Appli)**

Nom_Appli : Nom donné à l'application, dans le cas où on veut créer plusieurs relations DOCUMENT distinctes (dans le cas contraire, on appelle Init avec Nom_Appli = ""). Un numéro à deux chiffres est donné à chaque application, il s'agit de Code_Appli.

Création des relations internes de stockage SURROGATE, CONTENU, INDCONT et INDABS, et de la relation ELEMENTS.

La relation DOCUMENT est créée avec une liste minimale d'attributs, dont le Résumé qui est traité lui-même comme un Contenu.

- **Appli (Nom_Appli)**

Permet de changer l'application courante, en choisissant dans une liste. L'option par défaut est Nom_Appli = "" et Code_Appli = 10.

- **Ménage (Code_Appli)**

Pour faire le ménage.

- **Re_Init (Code_Appli)**

Pour faire le ménage, mais sans détruire l'environnement. Cette fonction équivaut à enchaner Ménage et Init, mais est beaucoup plus rapide.

- **Alter_Add_Doc (Standard, Nom_Attr, Type_Attr, Valeur, Code_Appli)**

Standard : standard pour lequel cet attribut de Profil est défini.

Nom_Attr : nom du nouvel attribut prédéfini du Profil

Type_Attr : type de l'attribut (INTEGER, STRING, TEXT, etc)

Valeur : code précisant si cet attribut est obligatoire ou facultatif, mono ou multivalué.

Cette fonction permet de déclarer un attribut du Profil, attribut qui sera systématiquement recherché à chaque internalisation de document (voir suite).

Avant d'ajouter cet attribut, la fonction vérifie qu'il appartient à la liste prédéfinie du standard utilisé.

Pour l'instant, nous n'acceptons que des attributs atomiques monovalués et facultatifs.

3.2 Manipulation des documents

Cette manipulation peut se faire soit de manière ensembliste, soit document par document.

3.2.1 Indexation d'une classe de documents

La fonction suivante permet d'indexer un ensemble de documents en fonction d'un élément logique, indépendamment de la structure logique spécifique des documents d'origine :

- **Index_Doc_Elem (Relation, Attribut, Elément, Code_Appli)**

Relation : nom de la relation contenant les surrogates des documents à filtrer (par défaut, c'est la relation DOCUMENT).

Attribut : attribut de Relation qui contient les surrogates des documents à filtrer (par défaut SDOC).

Elément : élément logique générique des documents dans lequel on fera effectivement la recherche.

Code_Appli : Code de la base ouverte.

La fonction suivante permet d'indexer complètement un ensemble de documents :

- **Index_Elem (Relation, Attribut, Code_Appli)**

Relation : nom de la relation contenant les surrogates des documents à filtrer (par défaut, c'est la relation DOCUMENT).

Attribut : attribut de Relation qui contient les surrogates des documents à filtrer (par défaut SDOC).

Code_Appli : Code de la base ouverte.

3.2.2 Sélection des documents

- **Select_Doc (Relation_Entrée, Attribut_Entrée, Elément, < expression >, Relation_Sortie, Attribut_Sortie, Code_Appli)**

Relation_Entrée : nom de la relation contenant les surrogates des documents à filtrer (par défaut, c'est la relation DOCUMENT).

Attribut_Entrée : attribut de Relation_Entrée qui contient les surrogates des documents à filtrer (par défaut SDOC).

Elément : élément logique générique des documents dans lequel on fera effectivement la recherche. Il peut prendre les valeurs suivantes :

contenu : recherche dans tous le contenu des documents (option

pardéfaut)
résumé : recherche dans les résumés des documents
résumé, titre, sujet, date, ... : recherche dans les attributs correspondant du Profil des documents.
(autres attributs) : recherche dans des éléments logiques des documents.

Expression : expression cherchée.

Relation_Sortie : nom de la relation contenant les surrogates des documents filtrés (par défaut, c'est la relation AUX1).
Attribut_Sortie : attribut de Relation_Sortie qui contient les surrogates des documents filtrés (par défaut SDOC).

3.2.3 Mise à jour des documents

- **Créer_Doc (Sdoc, Standard, Nom_Fichier, Code_Appli)**

Création d'un Sdoc, qui sera stocké dans la relation DOCUMENT, et retourné à l'application. On incrémente SURROGATE.

Internalisation du document codé dans le fichier Nom_Fichier, faite en respectant la structure d'origine. On reconnait les attributs prédéfinis du Profil, et on stocke le document original comme contenu dans CONTENU. L'indexation est faite par les relations INDABS et INDCONT.

- **Extern_Doc (Sdoc, Nom_Fichier, Code_Appli)**

Externalisation des documents. Cette fonction retourne le standard utilisé pour coder le document Sdoc.

- **Détruire_Doc (Sdoc, Code_Appli)**

Destruction du document Sdoc et de son contenu.

3.3 Manipulation d'un document

Les fonctions ensemblistes utilisent des primitives de bas niveau qui s'appliquent à un seul document à la fois. Respectivement, on aura :

- **Index_Doc_Elem1 (Sdoc, Elément, Code_Appli)**
- **Index_Elem1 (Sdoc, Code_Appli)**
- **Select_Doc (Sdoc, Element, < expression >, Code_Appli)**
  qui prend la valeur Vrai ou Faux.

3.4 Manipulation du contenu d'un document

- **Remplacer_Contenu (Sdoc, Standard, Nom_Fichier, Code_Appli)**

  Replacement du contenu d'un document.

- **Copier_Doc_Contenu (Doc_Origine, Doc_Destination, Code_Appli)**

Recopie du contenu du document Doc_Origine dans le document Doc_Destination.

- **Détruire_Contenu (Sdoc, Code_Appli)**

Destruction du contenu du document Sdoc.

- **Sélect_doc_PC (Sdoc, < liste de portions de contenu >, < expression >, Code_Appli)**

Opérateur TALKS ABOUT sur une liste de portions de contenu d'un document Sdoc.

La réponse est la liste des portions de contenu contenant l'expression cherchée.

3.5 Autres fonctions

- **Connection (Nom_Appli, Mot_de_Passe)**
- **Déconnection (Code_Appli)**
- **Visualiser_Struct_Doc (Sdoc, Code_Appli)**
  Pour connaitre la structure logique spécifique d'un document (telle qu'elle est décrite par le Profil d'origine).
- **Visualiser_Doc (Sdoc, Code_Appli)**
  Pour connaitre le contenu textuel d'un document, sans véritable mise en page.

Ces deux dernières fonctions sont tout-à-fait spécifiques du GDA. Elle servent à l'interface conversationnelle.

AUTORISATION DE SOUTENANCE

DOCTORAT 3ème CYCLE, DOCTORAT INGENIEUR,
DOCTORAT DE L'UNIVERSITE JOSEPH FOURIER - GRENOBLE 1

Vu les dispositions de l'Arrêté du 16 avril 1974,

Vu les dispositions de l'Arrêté du 5 juillet 1984,

Vu les rapports de M KOULOUMDJIAN Jacques

M CHRISMENT Claude

M DE LIMA José Valdeni est autorisé(e)
à présenter une thèse en vue de l'obtention du doctorat de
l'Université Joseph Fourier

Grenoble, le 07 MARS 1990

Le Président de l'Université
Joseph Fourier - Grenoble 1

A. NEMOZ

UNIVERSITÉ JOSEPH FOURIER
SCOLARITÉ