Université Paris IX-Dauphine
UFR Sciences des Organisations

Objets similaires de grande taille dans les bases de données

Thèse

Pour l'obtention du titre de

Docteur en Sciences

Spécialité Informatique

Présentée et soutenue par

Maude MANOUVRIER

le 17 janvier 2000 devant le jury composé de :

Directeur de Thèse :	**Mme Geneviève JOMIER** Professeur à l'Université Paris IX Dauphine
Rapporteurs :	**Mme Claudia BAUZER MEDEIROS** Professeur à l'Université de Campinas, Brésil
	M. Wojciech CELLARY Professeur à l'Université des Sciences Economiques de Poznań, Pologne
	Mme Marta RUKOZ Professeur à l'Université Centrale du Venezuela
Suffragants :	**M. Stéphane GANÇARSKI** Maître de Conférences à l'Université Paris VI Pierre et Marie Curie
	M. Jacques LE MAITRE Professeur à l'Université de Toulon et du Var
	M. Vangelis PASCHOS Professeur à l'Université Paris IX Dauphine

Numéro :..............

à ma mamie

Je remercie tout particulièrement :

Mme Geneviève Jomier, Professeur à l'Université Paris Dauphine et directeur de cette thèse, qui a su, tout au long de ces trois années, me guider, me conseiller et révéler en moi, le chercheur que je ne soupçonnais pas pouvoir devenir un jour. Par sa confiance, ses remarques, ses qualités humaines et professionnelles, elle a permis que cette thèse existe. Elle a été le directeur de thèse qu'il me fallait et je ne sais quels mots employer pour l'en remercier.

Mme Claudia Bauzer-Medeiros, Professeur à l'Université de Campinas et rapporteur de cette thèse pour ses précieux conseils et son optimisme contagieux qu'elle a su me transmettre à chacune de ses visites en France et lors de ma pré-soutenance. Merci de me faire l'honneur de participer à ce jury.

M. Wojciech Cellary, Professeur à l'Université des Sciences Economiques de Poznań et rapporteur de cette thèse, pour son écoute et l'intérêt qu'il a porté à mon travail lors de nos différents échanges. Merci de me faire l'honneur de participer à ce jury.

Mme Marta Rukoz, Professeur à l'Université Centrale du Venezuela et rapporteur de cette thèse, avec qui il a été facile, extrêmement agréable et fructueux de collaborer. Elle m'a considérée comme une de ses filles pendant tous mes séjours à Caracas et m'a fait découvrir des régions du monde et de la recherche qui m'étaient jusqu'alors inconnues. Merci de me faire l'honneur de participer à ce jury.

M. Stéphane Gançarski, Maître de Conférences à l'Université Pierre et Marie Curie, qui a toujours su, quel que soit l'endroit du monde où il se trouvait, me conseiller et me réconforter dans les moments douloureux de la thèse. Merci de me faire l'honneur de participer à ce jury.

M. Jacques Le Maitre, Professeur à l'Université de Toulon et du Var, de m'avoir fait profiter de l'action qu'il mène depuis des années au GDR PRC I3 "Documents Multimédia". Merci de me faire l'honneur de participer à ce jury.

M. Vangelis Paschos, Professeur à l'Université Paris Dauphine et directeur du LAMSADE, pour ses conseils lors de ma pré-soutenance et pour m'avoir souvent raccompagnée en automobile lorsque je quittais tard Dauphine. Merci de me faire l'honneur de participer à ce jury. Merci à lui et à M. Bernard Roy, Professeur à l'Université Paris Dauphine, de m'avoir accueillie au sein de l'équipe du LAMSADE et de m'avoir offert un cadre propice à l'élaboration de cette thèse.

Je remercie Odile Baudet et Josette Rey d'avoir fait en sorte que le LAMSADE soit devenu pour moi comme une famille. Je remercie Dominique Champ-Brunet d'avoir ensoleillé toutes mes pauses-thé et d'avoir toujours su trouver le documentation qu'il me fallait quand il le fallait. Je remercie Dominique Vallée, Kjell Karlsson, Ally Peerbocus et Floréal Prats qui ont dompté, comme par magie, toutes les machines qui me résistaient. Je remercie Daniel Vanderpooten avec qui j'espère bien poursuivre le travail que nous avons commencé ensemble. Je remercie toute l'équipe de Bases de Données et Génie Logiciel, et particulièrement Talel Abdessalem, Marie-José Blin, Jasir Ceballos, Ally Peerbocus, Hugo Sanchez Arroyo et Michel Zamfiroiu, pour leurs remarques, leurs précieux conseils et pour ne s'être jamais lassés de mes exposés sur les "quad-trees". Je remercie Anne Doucet, pour sa gentillesse, ses encouragements, son écoute et ses conseils lors des mes exposés au LIP6. Je remercie également le Rectorat de Paris et la commission de spécialistes de l'Université Paris-Dauphine de m'avoir permis de vivre cette thèse de manière sereine financièrement.

Je remercie mes petits parents, d'abord de m'avoir faite, ensuite d'avoir toujours su guider et révéler, depuis la maternelle, les choix d'études et les choix professionnels que je nourrissais inconsciemment, et enfin de n'avoir jamais su être objectifs à mon égard. Je remercie Mouloud, mon entraîneur, qui a apaisé mes moments de doute, surveillé consciencieusement et quotidiennement ma santé et qui, obstinément convaincu du résultat, a été le meilleur commercial de ma thèse auprès de mes proches. Je remercie ma copine Cécile A. de m'avoir toujours soutenue, d'avoir partagé mes pauses et d'être là tout simplement. Merci à elle, Agnès, Ally et Isabelle d'avoir été des scripts attentionnés lors de ma pré-soutenance. Je remercie Vincent pour sa gentillesse et pour m'avoir maintes fois ramenée dans son "humble carrosse" vers le 13ème. Enfin, je remercie tous les membres du LAMSADE et particulièrement Dominique François, Sophie, Lolo, Xavier, Slim, Lari, Jérôme, Jalal, Cécile M., Linda, Cristina, Alexis, et Laure, qui, quotidiennement, m'ont permis de travailler dans la joie et la bonne humeur par leurs conseils, leur présence et leur gentillesse. Merci également à mon grand-père, qui a toujours demandé régulièrement des nouvelles de ma thèse sans n'avoir jamais bien compris ce que je faisais dans la vie.

Table des matières

Introduction **1**

A Cadre de la thèse et problématique **3**

1 Versions d'objet de grande taille faiblement structuré : un état de l'art **5**
1.1 Gestion des objets de grande taille dans les bases de données 6
1.1.1 Stockage interne à la base de données 7
1.1.2 Système de fichiers associé à la base de données 8
1.1.3 Méthodes d'accès 9
1.1.4 Interrogation des objets de grande taille 10
1.2 Mécanismes de gestion de versions d'objets de grande taille 11
1.2.1 Modèles de base pour les versions d'entités 11
1.2.2 Chevauchement de structures d'accès 14

2 Approche des Versions de Bases de Données **18**
2.1 Concepts du modèle 19
2.2 Implantation 20
2.3 Mise à jour des versions d'objet 22
2.4 Gestion des versions d'objet complexe 23

B Images similaires **25**

3 Contexte : une application de traitement d'images **27**

4 Représentation d'images similaires par des arbres quaternaires **33**
4.1 Généralités sur les images 34
4.1.1 Format 35
4.1.2 Caractéristiques visuelles 35
4.2 Organisation d'images en arbre quaternaire 37
4.2.1 Définition générale et implantation d'un arbre quaternaire 37
4.2.2 Critères de découpage d'une image en quadrants 40
4.2.3 Opérations applicables aux images organisées en arbre quaternaire 41
4.3 Chevauchement d'arbres quaternaires 44
4.3.1 Chevauchement d'arbres quaternaires implantés de manière hiérarchique . . 45
4.3.2 Chevauchement d'arbres quaternaires linéaires 46

4.3.3 Conclusion sur l'état de l'art 47

5 Arbre Quaternaire Générique 49
5.1 Principes de l'Arbre Quaternaire Générique 50
5.1.1 Partage entre arbres quaternaires d'image 50
5.1.2 Similarité entre images 51
5.1.3 Arbre d'Images 52
5.1.4 Nœuds génériques 52
5.1.5 Identification interne des images 54
5.2 Opérations 54
5.2.1 Lecture d'une image 55
5.2.2 Modification d'une image 55
5.2.3 Insertion d'image 59
5.2.4 Suppression d'image 61
5.2.5 Réorganisation de l'Arbre d'Images 63
5.2.6 Autres opérations 66
5.3 Organisation d'un ensemble d'images en Arbre Quaternaire Générique 68
5.3.1 Notation 69
5.3.2 Arbre Quaternaire Générique SPI 69
5.3.3 Graphe de partage et déduction de l'Arbre d'Images 70
5.3.4 Construction de l'Arbre Quaternaire Générique final 73
5.4 Implantation des mécanismes 74
5.4.1 Images médicales organisées en Arbre Quaternaire Générique 74
5.4.2 Prototype MIS 76

6 Comparaison avec les autres travaux de recherche 81
6.1 Similarité dans le domaine des images : un état de l'art 82
6.1.1 Classification des méthodes d'interrogation d'images 82
6.1.2 Recherche d'images par une structure multidimensionnelle 83
6.1.3 Requêtes sur les régions d'images 86
6.2 Arbre Quaternaire Générique et similarité d'images 88
6.2.1 Similarité de surface des images d'un Arbre Quaternaire Générique 88
6.2.2 Comparaison avec les approches de similarité d'images 91
6.3 Comparaison avec le chevauchement d'arbres quaternaires 94
6.3.1 Organisation des images et partage d'information 94
6.3.2 Optimisation de l'espace occupé par les images 96
6.3.3 Accès aux images 98
6.3.4 Opérations 99

C Généralisation des mécanismes aux structures arborescentes 101

7 Arbres B+ et R Génériques 104
7.1 Structures d'index arborescentes équilibrées 105
7.1.1 Arbre B et variantes d'Arbre B 105
7.1.2 Arbre R 110

7.2 Arbres B+ et R Génériques 113
7.2.1 Etats d'arbre 114
7.2.2 Hiérarchie d'Arbres 114
7.2.3 Nœuds et racine génériques 114
7.3 Algorithmes de recherche et d'insertion 118
7.3.1 Rechercher et insérer dans un Arbre B+ Générique 118
7.3.2 Rechercher et insérer dans un Arbre R Générique 122
7.4 Applications 125
7.4.1 Approche de versions unifiée 125
7.4.2 Gestion des versions de collection d'objets 127
7.4.3 Perspectives 130

8 Comparaison des Arbres Génériques avec les autres travaux 132
8.1 Index à une dimension 134
8.1.1 Gestion d'un intervalle d'appartenance aux versions de collection 134
8.1.2 Regroupement des éléments par versions de collection 135
8.2 Index à deux dimensions 136
8.2.1 Clé d'index à deux dimensions 137
8.2.2 Utilisation d'index spatiaux 137
8.3 Structures persistantes : gestion de versions d'index 138
8.3.1 Persistance partielle 138
8.3.2 Persistance complète 143
8.4 Comparaison des Arbres B+ et R Génériques avec les autres travaux 147
8.4.1 Comparaison avec les modèles temporels 147
8.4.2 Comparaison avec les modèles de versions d'index 148

Conclusion et perspectives 156
1 Représentation des images similaires 156
1.1 Apports de l'Arbre Quaternaire Générique 157
1.2 Perspectives 158
2 Structures Arborescentes Génériques 160
2.1 Apports 160
2.2 Perspectives 160
3 Conclusion générale de la thèse 162

Bibliographie 163

Listes des figures et des tableaux 178

Annexes 182

A1 Démonstration : la Q-similarité est une distance 183

1 Démonstration de $d(i, i') \in [0, 1]$ 183
2 Démonstration de $d(i, i) = 0$ 183
3 Démonstration de $d(i, i') = d(i', i)$ 184
4 Démonstration de l'inégalité triangulaire 184
4.1 Démonstration de l'inégalité $\mid S(i, i') \mid \leq \mid S(i, j) \mid + \mid S(j, i') \mid$ 184
4.2 Cas où $\mid U(i, i') \mid \geq \mid U(i, j) \mid$ et $\mid U(i, i') \mid \geq \mid U(j, i') \mid$ 187
4.3 Cas où $\mid U(i, j) \mid \leq \mid U(i, i') \mid \leq \mid U(j, i') \mid$ 187
4.4 Cas où $\mid U(j, i') \mid \leq \mid U(i, i') \mid \leq \mid U(i, j) \mid$ 188
4.5 Cas où $\mid U(i, i') \mid \leq \mid U(i, j) \mid$ et $\mid U(i, i') \mid \leq \mid U(j, i') \mid$ 188
4.6 Conclusion de la démonstration 188

A2 Recherche et insertion dans un arbre B+ et dans un arbre R 189

1 Algorithmes de recherche et d'insertion dans un arbre B+ 189
1.1 Recherche d'un enregistrement de clé K dans un arbre B+ 189
1.2 Insertion d'un enregistrement de clé K dans un arbre B+ d'ordre m 190
2 Algorithmes de recherche et d'insertion dans un arbre R 192
2.1 Recherche, dans un arbre R, des chemins menant au point P de l'espace . . 192
2.2 Insertion d'un objet (rem, oid) dans un arbre R 192

Introduction

Le cadre de cette thèse est relatif à la représentation et à la manipulation d'objets similaires de grande taille dans les bases de données. Un objet est dit de grande taille lorsqu'il dépasse la taille d'une page de disque ou d'une page de mémoire [CDRS89, DLR91, Bil92b]. La lecture d'un tel objet nécessite plus d'une entrée/sortie. Il peut s'agir d'images [KTKM96, ATF+97], de composants logiciels [OHE96] ou encore d'objets complexes dont la cardinalité des objets composants peut atteindre des milliers voire des millions d'éléments [DLR91]. Ces objets peuvent être fortement, semi ou faiblement structurés. Les objets *fortement structurés* (*highly structured*) sont bien typés ; ils peuvent être représentés par une table dans une base de données relationnelle ou par un graphe orienté dans une base de données orientée-objet [Abi97]. Le schéma de ces objets correspond à une description sémantique [Kow97]. Par exemple, une collection d'objets, dont chaque objet est de type *employé* avec pour attributs *nom* de type chaîne de caractères et *salaire* de type réel, est un objet fortement structuré, dont la sémantique n'est pas ambigüe. Lorsque un utilisateur saisit une requête classique (en SQL ou en OQL) à partir du schéma d'un objet fortement structuré, le résultat de la requête correspond exactement à l'information recherchée. Les objets *semi-structurés* correspondent, en revanche, à des objets dont la structure est imprécise ou irrégulière ; la structure n'est pas définie de manière aussi rigide que dans les bases de données [Abi97]. Les documents au format SGML [ISO86] ou XML [BPSM98, Mic98] sont des objets semi-structurés. Ces objets ne répondent pas, ou seulement partiellement, à des requêtes exactes [BYRN99]. Les images ou les séquences non interprétées d'octets (BLOB ou *Binary Large OBject*) sont, quant à eux, des objets, qui, pris à l'état brut, c'est-à-dire sans traitement ou extraction du contenu, sont *faiblement structurés* (*loosely structured*). Une image, vue comme un ensemble de pixels, n'a aucun contenu sémantique.

Deux objets sont dits similaires s'il existe peu de différences entre les objets, la notion de "peu" dépendant fortement de l'application. Les objets de grande taille occupent beaucoup de place. Une idée, pour optimiser le stockage d'objets de grande taille et peu différents, est d'adopter une représentation dans la base permettant de stocker une seule fois les parties qui sont identiques dans différents objets et de les leur faire partager. Les modèles de versions [KC88, CJ90, BBA91, TOC93, Haa94] se basent sur cette idée de partage.

Si beaucoup de travaux [KBG89, CJ90, Kat90, BBA91, KS92, Loo92, TOC93] se sont orientés vers la gestion de versions d'entités complexes, comme les entités composées d'entités ou les entités se référant à d'autres, en revanche, peu de recherches [CDRS89, LF91, Rei97, BN98] se sont orientées vers le versionnement d'objets de grande taille. Cependant, les applications gèrent de plus en plus d'objets de grande taille fortement ou faiblement structurés [BLPS96, KB96, LK97, BN98, Sub98, SZB99], comme les images ou les documents textuels ou hypertextuels. Les mécanismes de versions développés dans les bases de données, dites traditionnelles, doivent par conséquent être élargis pour remédier au problème de gestion et de recopie d'objets de grande taille.

Les structures de données fortement structurées, comme les arbres B [BM72] ou les arbres quaternaires [Sam84], permettent de d'organiser les objets de grande taille. Plusieurs approches proposent par exemple de représenter les BLOB par des arbres B+ [CDRS89, Bil92b, Bil92a, KB96]. Lorsque les objets de grande taille sont gérés à l'aide de ces structures, les mécanismes de versionnement doivent être étendus aux structures de données. Cependant, les approches [CDRS89, MK90, LM91, Kol93, BGO+96, VV97, TML99], qui proposent de gérer des versions de structures arborescentes, sont généralement dépendantes de la sémantique du versionnement (par exemple le temps). Les mécanismes proposés limitent considérablement les opérations applicables aux objets de grande taille, c'est-à-dire la création et la suppression de ces objets dans la base de données, leur modification et leur comparaison.

La principale contribution de cette thèse est la proposition de mécanismes de versionnement de structures arborescentes. Ces mécanismes, basés sur l'approche des Versions de Bases de Données de [CJ90, CJ98], peuvent s'appliquer aux objets faiblement structurés, par exemple les images, comme aux objets fortement structurés, par exemple les collections d'objets. Ils sont indépendants de la sémantique du versionnement. L'*Arbre Quaternaire Générique* est proposé dans cette thèse pour gérer des images similaires créées dans une application de traitement d'images. Cette structure optimise le stockage des images similaires tout en permettant aux utilisateurs d'appliquer des opérations de traitement sur les images et de les comparer. Les structures d'*Arbre B+ Générique* et d'*Arbre R Générique*, également proposées dans cette thèse, permettent de gérer des versions de collection d'objets obtenues par mises à jour successives.

Cette thèse est organisée en trois parties. La partie A présente en détail le cadre et la problématique de la thèse. Dans cette partie, le chapitre 1 présente les différents mécanismes de gestion des objets de grande taille dans les bases de données et décrit quelques approches de versionnement d'objets de grande taille. Le chapitre 2 présente l'approche des Versions de Bases de Données de [CJ90, CJ98], sur laquelle se basent les propositions de cette thèse.

La partie B concerne la gestion des images similaires. L'application de traitement d'images, qui a été le point de départ de la proposition présentée dans cette partie de la thèse, est décrite dans le chapitre 3. Les images gérées par les utilisateurs de cette application sont organisées en arbre quaternaire. Le chapitre 4 présente cette structure, les différentes manières de l'implanter et plusieurs approches de compactage d'images similaires organisées en arbre quaternaire. Le chapitre 5 présente l'Arbre Quaternaire Générique. Le chapitre 6 compare cette structure aux autres travaux de recherche.

La partie C présente la généralisation des mécanismes proposés dans la partie B aux arbres B+ et aux arbres R. Dans cette partie, le chapitre 7 présente les structures d'Arbre B+ Générique et d'Arbre R Générique. Le chapitre 8 compare les autres approches avec les arbres génériques proposés dans la thèse.

Partie A

Cadre de la thèse et problématique

Cette thèse aborde les problèmes de gestion d'objets similaires de grande taille dans les bases de données. Les versions d'objet peuvent être vues comme des objets similaires. Les objets de grande taille, parce qu'ils occupent beaucoup de place et nécessitent des traitements particuliers, sont gérés à l'aide mécanismes spécifiques dans les bases de données. De même, lorsque ces objets possèdent des versions, des mécanismes de versionnement appropriés doivent être proposés pour gérer les versions d'objet de grande taille.

L'organisation de cette partie est la suivante. Le chapitre 1 présente un état de l'art sur les versions d'objet de grande taille faiblement structuré. Ce chapitre définit les concepts généraux de la gestion d'objets de grande taille dans les bases de données et présente des mécanismes spécifiques à la gestion des versions de ces objets : le chevauchement de structures arborescentes. Le chapitre 2 présente en détail l'approche des Versions de Base de Données et justifie le choix de cette approche sur laquelle se basent les propositions de cette thèse.

Chapitre 1

Versions d'objet de grande taille faiblement structuré : un état de l'art

Résumé

Ce chapitre présente les différents mécanismes de gestion d'objets de grande taille faiblement structurés dans les bases de données, comme par exemple les documents textuels, les images ou la vidéo. Ces objets, à cause de leur taille, sont le plus souvent stockés sous la forme de séquence non interprétée d'octets (BLOB *Binary Large OBject*, CLOB etc.). Certains Systèmes de Gestion de Bases de Données permettent de les stocker à l'intérieur de la base de données. D'autres stockent dans la base de données le chemin d'accès ou la localisation du fichier contenant l'objet, ce fichier appartenant, le plus souvent, à un système de fichiers associé à la base de données. En fonction de la manière avec laquelle l'objet est modélisé et stocké, plusieurs modèles d'interrogation ont été proposés. Le manque de structure rigide et bien définie des objets de grande taille faiblement structurés rend leur interrogation difficile par des requêtes exactes. Le terme de *Recherche d'Information* est de préférence employé. En effet, l'interrogation des objets de grande taille faiblement structurés, comme les images ou le texte, se traduit généralement par une interprétation de leur contenu et par un calcul de pertinence des objets par rapport à la requête posée. Pour faciliter l'accès à ces objets, il est possible de les organiser à l'aide des structures d'index disponibles dans les bases de données, comme les arbres B+. Lorsqu'un objet de grande taille faiblement structuré possède des versions, les mécanismes de versionnement sont, dès lors, appliqués à la structure organisant l'objet. La gestion des versions de ces objets se limite, dans la plupart des cas, à l'archivage (*backup*) des versions antérieures.

Introduction

L'association entre bases de données et objets de grande taille fait partie des thèmes actuels de recherche [BN98], notamment lorsque ces objets sont de type multimédia [SJ96, MM99], comme les images [KB96]. Ces objets occupent beaucoup d'espace (plusieurs Kilo-octets à plusieurs Giga-octets pour une image par exemple) et, lorsqu'ils sont faiblement structurés, leur contenu est complexe à extraire et à modéliser. La gestion de ces objets dans les bases de données nécessite, par conséquent, des mécanismes particuliers de stockage, d'accès et d'interrogation. Lorsque ces objets possèdent des versions, là encore, les modèles de base ne permettent pas toujours de répondre aux besoins, et des techniques spécifiques doivent être proposées.

Ce chapitre a pour objectif de situer le cadre de la thèse : la gestion des versions d'objet de grande taille dans les bases de données. Il consiste en un état des lieux général dans ce domaine, et particulièrement en ce qui concerne les objets de grande taille faiblement structurés. Dans la suite du chapitre, par abus de langage, nous utiliserons le terme *objet de grande taille* pour *objet de grande taille faiblement structuré*.

L'organisation de ce chapitre est la suivante. Les différents mécanismes permettant d'allier les bases de données et les objets de grande taille sont présentés en 1.1. La section 1.2 décrit quelques approches proposées pour gérer le versionnement de ce type d'objet.

1.1 Gestion des objets de grande taille dans les bases de données

Les objets de grande taille sont complexes à gérer dans les bases de données. Leur gestion impose un ajout de propriétés spécifiques à leur manipulation dans le Système de Gestion de Bases de Données (SGBD). Idéalement, le gestionnaire de stockage doit pouvoir manipuler des objets de taille illimitée, dans la limite de la taille physique disponible sur le disque ou en mémoire [Bil92b].

Les opérations de manipulation des objets de grande taille correspondent généralement à des opérations sur les octets composant les objets. Il s'agit par exemple de la lecture ou du remplacement d'un ensemble d'octets et de l'insertion ou de la suppression d'octets à n'importe quel emplacement de l'ensemble d'octets composant l'objet de grande taille. Ces opérations sont importantes pour deux raisons majeures [Bil92b] :

1. Il n'est pas possible, pour des raisons physiques, de mettre à jour un tel objet en une seule fois ; une page de $buffer$ ne peut pas, en général, contenir l'objet dans sa totalité [Kim93].
2. Un objet de grande taille est généralement affiché en plusieurs fois, pour ne pas avoir un temps d'attente trop long avant de pouvoir visualiser l'objet dans sa totalité. En effet, l'affichage d'un objet de grande taille nécessite plusieurs entrées/sorties. L'affichage d'un document, par exemple, se fait portion par portion, l'utilisateur accédant à chaque portion par un ascenseur. De plus, les objets de grande taille sont généralement codés ou compressés pour occuper le moins possible d'espace disque. Leur lecture impose par conséquent un décodage de l'objet. Sur le Web par exemple, certaines images commencent à apparaître à l'écran de manière floue, leur résolution augmentant au fur et à mesure de l'affichage et de

FIG. 1.1 – **L'augmentation de la résolution d'une image au cours de son affichage à l'écran [Pin].**

la transmission de l'information (voir figure 1.1). Ce type d'affichage est dû au codage des images par ondelettes [Dav98].

Le SGBD doit également assurer la cohérence et la sécurité de ces objets et permettre leur interrogation [KB96].

Les SGBD offrant des mécanismes de gestion d'objets de grande taille peuvent être classés en deux grandes catégories : (1) les systèmes stockant les images dans la base de données (voir section 1.1.1) et (2) ceux stockant les images à l'extérieur de la base de données (voir section 1.1.2). Parmi ces systèmes, certains proposent de structurer les objets de grande taille, qui sont faiblement structurés, à l'aide des structures d'index disponibles dans les bases de données comme les arbres B [BM72, Knu73] ou B+ [Com79] (voir section 1.1.3). Il est difficile d'associer de la sémantique aux objets de grande taille et donc de les interroger. Différents modèles ont été proposés (voir section 1.1.4) pour interroger les objets de grande taille comme les documents textuels ou les images.

1.1.1 Stockage interne à la base de données

En général dans les bases de données et en particulier dans les bases de données relationnelles, les objets de grande taille sont stockés sous forme de séquence non interprétée d'octets ou *BLOB* [1] (*Binary Large OBject*). Les textes, par exemple, peuvent être stockés sous le format CLOB (C pour *Char*).

Le type *BLOB* a été ajouté au standard SQL92. Son intégration dans les bases de données relationnelles impose la possibilité de déclarer, dans les tables de la base, des types de longueur variable [KB96]. En conséquence, son utilisation est difficile. Elle implique, en effet, que les valeurs des champs (ou colonnes) puissent être de taille variable et ceci pour différentes lignes d'une même table. Les systèmes commerciaux actuels ne permettent donc pas de faire excéder la taille de ces champs au delà d'un maximum fixe. Le standard SQL92 propose des mécanismes pour définir des colonnes de taille variable, mais laisse non seulement la taille maximum des champs de longueur variable, mais aussi la lecture et la mise à jour de tel type de données, à la charge des constructeurs [KB96].

1. Le type BLOB peut être rencontré sous différents noms : *Long Raw*, *Image*, *Vargraphics* etc. [KB96].

Les systèmes relationnel-objet offrent la possibilité aux utilisateurs de définir leurs propres fonctions de visualisation et d'interrogation de données de type BLOB avec la notion d'extension de type (*type – extensibility*) [BN98]. La norme SQL-3 MM (*SQL Multimedia*) permet la définition de type abstrait de données (ADT ou *Abstract Data Type*) offrant des mécanismes de gestion de texte ou d'image. Cependant ces mécanismes sont restreints à quelques formats seulement, JPEG [Wal89, Wal91] et TIFF [AM88] pour les images par exemple.

Les Bases de Donnés Orientées Objets semblent plus naturelles pour modéliser et gérer les images [KB96]. Le systèmes O_2 [BDK92], par exemple, grâce à son orientation objet, permet de faire de l'encapsulation d'objets de grande taille et donc de définir des méthodes propres à ces objets [SO93]. Ce SBGDOO utilise le système de stockage Wisconsin (WiSS). Le système WiSS stocke les objets de grande taille dans des segments de données (ou *slice*) et regroupe l'adresse de tous les segments dans une page de disque [Bil92a]. La taille des objets de grande taille est par conséquent dépendante de la taille de la page stockant les adresses des segments.

Les bases de données orientées objets, comme les bases de données relationnelles ou relationnelles-étendues, sont confrontées aux mêmes limites de représentation compacte des BLOB ou des types équivalents [MM99]. Elles doivent notamment résoudre des problèmes tels que : la recherche incrémentale d'un objet de grande taille, la mise à jour incrémentale (une modification sur une partie de l'objet n'impose pas une copie entière de l'objet), ou la gestion de la concurrence sur de tels objets [Kim93]. Le système POSTGRES [SO93] propose, pour sa part, de stocker les objets de grande taille à l'intérieur de la base de données, sous forme de morceaux de taille fixe (limitée à 8 Kilo-octets), avec des mécanismes de gestion d'objets de grande taille par séquence d'octets.

1.1.2 Système de fichiers associé à la base de données

Plusieurs SGBD commerciaux comme Oracle V8 (avec le type *BFILE*) ou DB2 d'IBM stockent, dans la base de données, des liens vers les objets de grande taille, qui sont stockés dans un système de fichiers extérieur à la base [MM99]. Cette pratique impose, par conséquent, la gestion simultanée de la base de données et du système de fichiers contenant les objets dits externes.

Le système DB2 gère un système de fichier associé (*cooperating files system*) à la base de données relationnelle, nommé *DataLinker* [BN98]. Le système de fichiers est protégé par des mécanismes de sécurité gérés depuis la base de données. L'accès aux objets de grande taille est, par conséquent, limité aux clients de la base de données [BN98]. Les liens vers les objets extérieurs, ou *Datalinks*, peuvent être assimilés aux URL (*Uniform Resource Locator*) d'Internet. La base de données relationnelle offre des mécanismes pour lire et modifier les objets de grande taille en gérant, de manière transparente pour l'utilisateur, les liens vers le système de fichiers associé.

Le système POSTGRES [SO93] propose, via la définition de types de donnés abstraits, deux manières d'implanter les objets à l'extérieur de la base. La première consiste à laisser la gestion des fichiers extérieurs correspondants aux objets de grande taille à la charge de l'utilisateur, qui spécifie uniquement le chemin d'accès dans la base de données. Cette implantation est très simple mais très risquée. POSTGRES propose, par conséquent, un mécanisme permettant de rendre la base de données propriétaire du fichier contenant l'objet de grande taille. Cette pratique permet, comme pour les *Datalinks* de DB2, de limiter l'accès aux clients de la base de données.

Les gestion des objets de grande taille, même stockés dans un système de fichiers associé, est complexe à gérer depuis un SGBD. Dans le cas multimédia en particulier, les objets de grande taille nécessitent des serveurs spécialisés (RAID[2], systèmes temps réels, etc.), imposant des méthodes d'accès et de gestion très dépendantes du type d'application [MM99]. Plusieurs mécanismes de regroupement (*cluster*) et de préchargement sont proposés [Bil92b, ACF$^+$96, KB96, BBB$^+$97, Kra97, MM99], comme les courbes de Hilbert, pour faciliter l'accès et la recherche de ces objets.

1.1.3 Méthodes d'accès

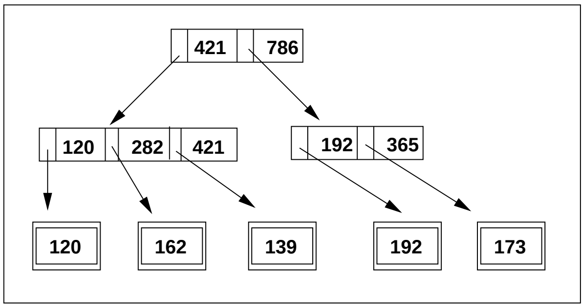

FIG. 1.2 – **La représentation des objets de grande taille dans le système *EXODUS* [CDRS89].**

Les objets de grande taille, lorsqu'ils sont représentés sous forme de séquence non interprétée d'octets, sont par nature faiblement structurés. Il est néanmoins possible de structurer leur accès en les organisant à l'aide des structures d'accès utilisées dans les bases de données, comme les arbres B+ [Com79]. Le système EXODUS [CDRS89] gère des objets de taille illimitée en les organisant dans un arbre B+, dit arbre positionnel [KB96]. L'objet de grande taille est stocké sur plusieurs pages. Son identificateur pointe sur une entête particulière, qui est représentée sous forme d'un arbre B+. Dans ce cas, la clé d'index correspond à la position des octets dans la séquence d'octets représentant l'objet de grande taille [Bil92b]. La racine et les nœuds d'index de l'arbre contiennent des paires de type *(adresse,clé)* [DLR91]; l'adresse est un pointeur vers le nœud fils ou vers une feuille. La clé donne la position maximale de la sous-séquence stockée dans le sous-arbre relatif à ce pointeur. Chaque paire est donc de la forme *(nombre d'octets contenus dans le sous-arbre, adresse du nœud fils racine du sous-arbre)*. Les nœuds internes ont une structure similaire à celle de la racine; ils représentent la racine d'un arbre positionnel d'un autre objet contenu à l'intérieur de celui du nœud parent; un déplacement absolu d'octets dans un nœud interne correspond donc à un déplacement relatif dans le nœud père [CDRS89]. Les feuilles sont des portions de la séquence d'octets de l'objet. Les clés représentent des déplacements relatifs par rapport au niveau supérieur.

La figure 1.2, reprise de [CDRS89], illustre ces propos. Un objet de 786 octets est stocké dans

2. *Redundant Array of Inexpensive Disks.*

un arbre B+ positionnel. La racine de l'arbre contient la séquence de clés [421,786], ce qui signifie que le premier sous-arbre contient les 421 premiers octets de l'objets, le deuxième sous-arbre contenant les octets allant de 422 à 786. La feuille la plus à droite sur la figure contient 173 octets. Le 100ème octet de cette feuille correspond au (100+192) = 292ème octet du sous arbre droit, et par conséquent au (421+292) = 713ème octet de l'objet stocké.

La gestion de l'arbre B+ est présentée en détail dans la partie C. Les algorithmes classiques de l'arbre B+ (présentés en annexe A2) ont été modifiés par [CDRS89] pour permettre la lecture, l'insertion et la suppression d'ensemble d'octets. Le système de stockage EOS, présenté dans [Bil92b], utilise le même principe avec des algorithmes d'insertion et de suppression modifiés par rapport à [CDRS89]. Pour plus de détails, le lecteur est invité à consulter [CDRS89, Bil92b].

1.1.4 Interrogation des objets de grande taille

Lorsque les objets interrogés sont de grande taille, le terme de recherche d'information (*Information Retrieval* ou IR) est généralement préféré à celui d'interrogation [KB96]. En raison de la richesse et de la complexité de l'information contenue dans les objets de grande taille, il est difficile de faire des requêtes exactes sur ces objets [Chi94]. C'est pourquoi on emploie plus volontiers le terme de recherche d'information ; il s'agit d'une interprétation du contenu des objets de grande taille et d'un classement de ces objets selon un degré de pertinence par rapport à la requête posée [BYRN99].

Il existe plusieurs modèles de recherche d'information. Ces modèles sont décrits avec précision dans [Kow97, BYRN99]. Le modèle booléen, basé sur la théorie des ensembles et sur l'algèbre booléen, correspond à un modèle de décision binaire ; un objet répond ou non à la requête. Ce modèle ne permet que l'interrogation exacte des objets sans possibilité de graduer les réponses. La requête, parce qu'elle est exacte, risque de retourner trop peu ou trop d'objets [BYRN99]. Le modèle vectoriel, qui s'applique particulièrement aux documents, consiste à représenter les documents et la requête en vecteurs de mots-clés. Dans ce modèle, une requête retourne les documents qui répondent partiellement (*partial matching*) aux termes de la requête, des poids non binaires ayant été associés à chacun des termes [BYRN99]. Les poids sont utilisés pour calculer un degré de similarité entre la requête et l'objet réponse. Par exemple, cette similarité peut correspondre à la fréquence d'un mot-clé dans le document. Le modèle probabiliste estime, quant à lui, la probabilité qu'un objet puisse répondre ou non à la requête et intéresser l'utilisateur. Des extensions des modèles précédemment décrits ont été proposés pour élargir les possibilités d'interrogation [BYRN99]. L'interrogation flexible, par exemple, basée sur les ensembles flous [BCR98, RC99], associe à chaque élément un degré d'appartenance à un ensemble. Ce modèle permet de faire des requêtes telles que "Trouver tous les noms des personnages apparaissant dans les scènes situées au début d'une vidéo donnée", en supposant que les scènes ont été annotées par les noms des personnages du film.

Les modèles classiques de recherche d'information, présentés dans le paragraphe précédent (booléen, vectoriel ou probabiliste), sont principalement dévolus à la recherche de documents textuels Ils ne conviennent pas à tous les types d'objets de grande taille comme les images. Par exemple, il est difficile de transposer une requête sur une image en expression booléen [BYRN99] (ex. "Trouver les images qui ressemblent à un coucher de soleil."). L'interrogation des images se fait généralement par l'exemple. Une image est sélectionnée dans la base, et le système recherche toutes les images similaires à l'image exemple par rapport à sa couleur, à la position des objets ou à une combinaison de plusieurs caractéristiques. Des détails sur l'interrogation des images sont présentés dans le chapitre 6.

Pour interroger les objets de grande taille, l'idée générale est d'extraire, de manière automatique, semi-automatique, voire manuelle, le contenu sémantique de objets et de stocker les informations obtenues dans la base de données. L'approche de [Gro84] est une des premières à avoir proposé ce genre de mécanismes pour les images [Sub98]. D'autes approches [CITB92, OTOL$^+$97, MG98] ont, par la suite, repris ce principe en ajoutant des opérateurs, par exemple spatio-temporels, ou des propriétés, par exemple le périmètre ou la surface, qui sont spécifiques à la sémantique associée aux images .

En définitive, la richesse d'information contenue dans les objets de grande taille, en particulier lorsqu'ils sont de type multimédia (ex. image), ne permet pas de caractériser de manière générale ces objets et ne permet donc pas toujours d'utiliser les modèles classiques pour l'interrogation [BCR98]. Il est à noter également que l'interaction avec l'utilisateur (pour raffiner sa requête) est un point important dans l'interrogation des objets de grande taille peu ou pas structurés.

1.2 Mécanismes de gestion de versions d'objets de grande taille

De nombreuses applications comme le génie logiciel [EF89, Kat90, Rei97, Zam98], les Systèmes d'Information Géographique [BMJ93, BMJ94] ou les applications multimédia [ABQ86, Adi88, CK95, KB96] ont besoin de représenter différentes valeurs des entités modélisées. Plusieurs approches [KBG89, CDRS89, CJ90, BBA91, LF91, KS92, Loo92, TOC93] ont été proposées pour gérer des versions d'entités. Ces approches prennent en compte les entités complexes, mais n'offrent généralement pas de mécanismes particuliers pour gérer les versions d'objet de grande taille. Or, si le besoin de gérer des versions s'est fait sentir pour les objets dits courts [DLR91], il apparaît aussi nécessaire de gérer des versions d'objet de grande taille [CDRS89, LF91, KB96, KTKM96, OHE96, ATF$^+$97]. En traitement d'image, par exemple, une image est généralement obtenue par applications successives d'algorithmes de traitement sur une même image originale, avec d'éventuels retours en arrière [KTKM96, ATF$^+$97]. Lors de la conception d'un site Web, plusieurs personnes peuvent coopérer et travailler sur différentes versions du site, et donc manipuler des versions de documents HTML ou XML. De même, l'émergence de CORBA [OHE96] entraîne l'apparition de versions d'objet binaire qui peuvent éventuellement être de grande taille.

La section 1.2.1 rappelle les mécanismes des modèles de base pour gérer les versions d'entités et montre que ces modèles ne s'adaptent pas facilement à la gestion des versions d'objet de grande taille. La section 1.2.2 présente des mécanismes de versionnement propres aux objets de grande taille.

1.2.1 Modèles de base pour les versions d'entités

Les modèles de versions peuvent être regroupés en deux grandes catégories [CJ90, CJ98] : l'approche ascendante et l'approche descendante. La première gère les entités de manière individuelles. La deuxième offre, quant à elle, une vision globale de l'univers modélisé. Ce chapitre rappelle les grandes lignes de ces approches. Pour plus de détails, le lecteur est invité à consulter [BBA91, GJ95, TO96], qui sont trois articles de synthèse sur la gestion de versions.

a) Approche ascendante

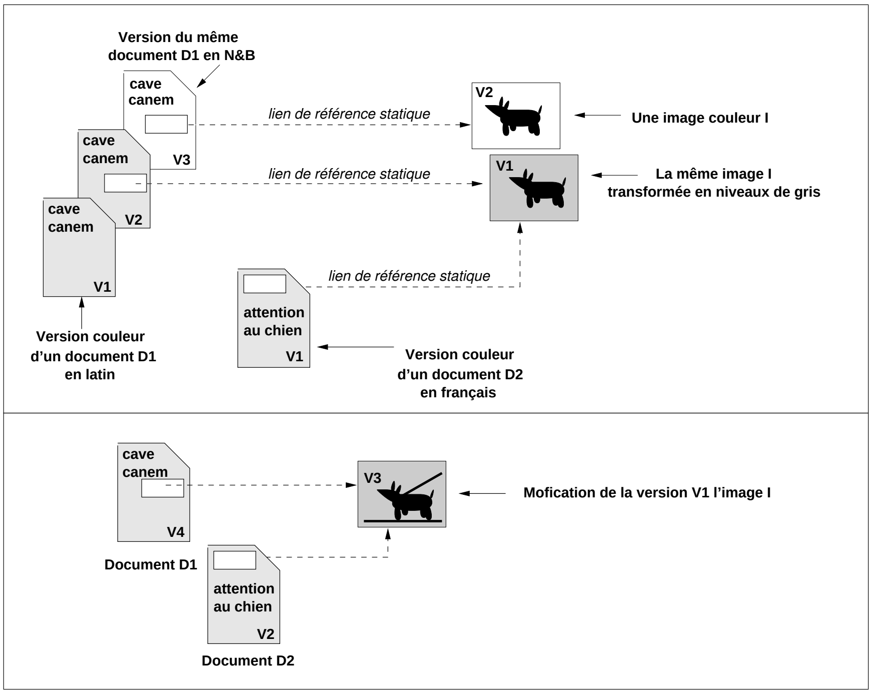

FIG. 1.3 – **L'approche ascendante : des versions individuelles d'entité individuelle.**

L'approche ascendante, également appelée *bottom-up* dans [CJ98] et orientée-version dans [Haa94], gère les versions individuellement pour chaque entité, c'est-à-dire indépendamment des autres entités de la base. Si une entité versionnable[3] e est identifiée par id_e, chaque version de l'entité est alors identifiée par le couple (id_e, id_v), où id_v est un identificateur de version dit *local* à l'entité.

Exemple 1.1 La figure 1.3 (en haut) donne un exemple de trois entités, deux documents D_1 et D_2 et une image I, versionnées selon cette approche. Le document D_1 possède trois versions : deux versions couleurs (grisées), identifiées par (D_1, v_1) et (D_1, v_2), et une version en noir et blanc identifiée par (D_1, v_3). Le document D_2 ne possède qu'une seule version, (D_2, v_1). L'image I possède deux versions : une version (I, v_1) de l'image en couleur et une version en niveaux de gris, identifiée (I, v_2), correspondant à une modification (diminution du nombre de couleurs) de la version (I, v_1).

3. Une entité est versionnable lorsqu'elle peut avoir des versions.

Cette gestion individuelle des versions d'entité fait qu'il n'y a aucun lien sémantique entre deux versions (e_i, v_k) et (e_j, v_k) de deux entités différentes e_i et e_j. Par exemple, le k peut correspondre à un numéro d'ordre dans les versions [GJ95] ; (e_i, v_k) désigne la $k$ième version de l'entité e_i. Dans la figure 1.3 par exemple, il n'existe aucune information sur la cohérence des versions de l'image et de celles du document : la version v_1 de l'image est-elle cohérente (= va t'elle) avec la version v_1 de chaque document D_1 et D_2 ? La référence entre versions d'entité est statique (voir flèche en pointillé sur la figure) et plusieurs versions d'entités peuvent faire référence à la même version. Par exemple, les versions (D_1, v_2) et (D_2, v_1) référencent la même version de l'image (I, v_1). Ce référencement statique entre versions d'entité impose des créations ou des destructions en cascade de versions, à chaque groupe de mises à jour sur des versions. L'objectif de ces créations et des structions en cascade est de conserver les liens de référence entre les versions.

Exemple 1.2 Par exemple, la version (I, v_1) de l'image est référencée par les versions (D_1, v_2) et (D_2, v_1) des documents D_1 et D_2. Par conséquent, toute mise à jour effectuée sur la version du l'image (I, v_1) est visible depuis les versions (D_1, v_2) et (D_2, v_1). Pour éviter cet effet de bord [GJ95], il est nécessaire de créer une nouvelle version avant d'effectuer les mises à jour. Dans la figure 1.3 (en bas), une nouvelle version (I, v_3) est créée à partir de (I, v_1) ; il s'agit d'une copie. Cette nouvelle version est mise à jour : une laisse est ajoutée au chien. Après mise à jour, elle n'est référencée par aucune version de D_1 ou de D_2. Si l'utilisateur souhaite visualiser les conséquences de la mise à jour dans l'un ou l'autre des documents, il doit dériver une nouvelle version du document D_1 ou D_2 et faire référence dans cette version à (I, v_3).

La complexité de ce phénomène appelé *percolation* a été étudiée dans [AJ89]. Tous les liens de référence (lien de composition au sein d'un objet complexe, lien d'héritage entre classes, liens entre instances d'une même classe, etc.) entre versions sont gérés de la même manière. Par conséquent, la notion d'état cohérent de la base n'est pas défini. En effet, il existe plusieurs états entrelacés de l'univers modélisé, chaque état regroupant les versions d'entités qui vont ensemble. Ce type de mécanisme, proposé dans des approches telles que [KBC+89, KS92, TOC93], pose donc d'importants problèmes de mise en œuvre sur des cas réels.

b) Approche descendante

L'approche descendante, également appelée *top-down* dans [CJ98] ou orientée-changement dans [Haa94], propose une vision globale de l'univers modélisé. L'objectif majeur de cette approche est le maintien de la cohérence. Les versions sont gérées simultanément pour de grands ensembles d'entités, comme dans O_2Version [Tec95] et dans [AJFL93, CJV94], ou pour toutes les entités versionnables (la base de données) [CJ90, CJ98]. On parle de manière générale de *configuration*. La figure 1.4 donne un exemple de plusieurs versions de configuration possédant chacune une version des entités de la base, D_1, D_2 et I.

La non redondance d'information est possible s'il y a partage des versions d'entité de même valeur entre les configurations. La base de données est alors gérée à deux niveaux [CJ90], le niveau logique, que voit l'utilisateur, et le niveau physique, géré par le système et qui rend les mécanismes de partage transparents à l'utilisateur. Deux types d'opérations sont disponibles, la dérivation d'une nouvelle configuration par copie logique des versions d'entité qui la composent et, la mise à jour (suppression, création, modification) d'une entité dans une version de configuration. Une

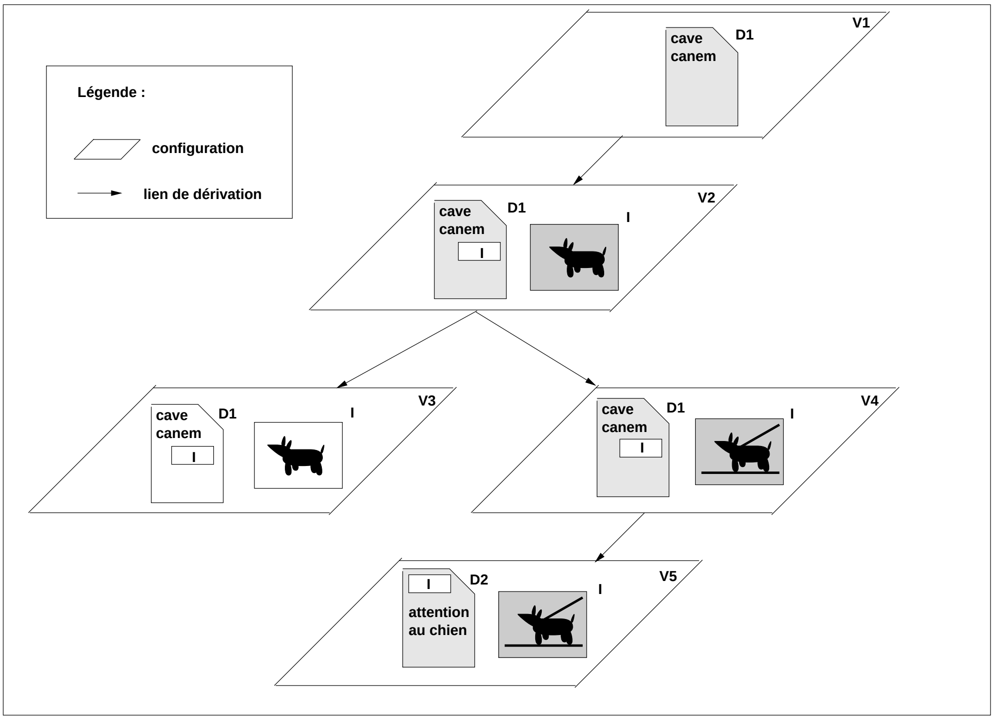

FIG. 1.4 – **L'approche descendante : gestion d'ensemble de versions d'entités cohérentes entre elles.**

version d'entité est, comme précédemment, identifiée par un couple (id_e, id_v), mais id_v est cette fois-ci un identificateur *global* à la base. Deux versions (e_i, v_k) et (e_j, v_k) sont dites *mutuellement cohérentes*, si elles appartiennent à la même configuration, qui est supposée cohérente au sens classique de ce terme dans les bases de données. Par exemple, dans la figure 1.4, les documents D_1 et D_2 font référence à l'objet image I, chaque configuration regroupant les versions des entités (document et image) qui vont ensemble. Les références entre versions sont dynamiques. En effet, les références entre versions d'entité sont internes à une configuration. Seul l'identificateur de l'entité est utile pour désigner la version de l'entité référencée. Il n'y a donc pas de création ou de suppression de versions en cascade lors des mises à jour. En outre, il est possible de mettre en place des mécanismes automatiques de gestion de versions de schéma avec maintien de la cohérence au sein de la version de schéma [KC88, BJC98] et avec les versions d'extension. Pour plus de détails, le lecteur est invité à consulter [GJ95, CJ98]. L'approche *top-down* est développée, dans le chapitre 2, sur un cas particulier, l'approche des Versions de Bases de Données [CJ90].

Parmi les modèles de versions existants [KBG89, KBC+89, KGBW91, KS92, Loo92], peu [CDRS89, LF91, LM91, Rei97] se sont intéressés à la gestion des versions d'objet de grande taille. La section qui suit présente les mécanismes généraux utilisés pour gérer les versions d'objets de grande taille.

1.2.2 Chevauchement de structures d'accès

Pour limiter la redondance d'information lors de la gestion de versions d'objets de grande taille, le stockage différentiel, basé sur le principe de fichiers différentiels [Roc75, Tic85], est généralement utilisé [Pal90, BBA91, BN98]. Dans DB2 [BN98], par exemple, il est possible de gérer des versions d'objets de grande taille, ces objets étant stockés à l'extérieur de la base (voir section 1.1.2). A une version correspond un nouveau fichier contenant la nouvelle valeur de l'objet de grande taille. Pour éviter la redondance, le stockage différentiel est utilisé, et les versions ne contiennent dans ce cas que les octets modifiés par rapport à la version précédente. Le versionnement est linéaire et seule la version courante de l'objet (la dernière) est accessible à l'utilisateur. D'après les auteurs de [BN98], la construction d'une version d'objet (qui n'est pas la version courante) se fait rarement, uniquement lors de la restauration de la base de données. Le stockage différentiel est, dans ce cas, dévolu à l'archivage des versions d'objets de grande taille.

Comme il a été présenté dans la section 1.1.3, les objets de grande taille peuvent être gérés par des structures de données comme les arbres B+ [CDRS89]. Les mécanismes de stockage différentiel sont dès lors appliqués aux structures. L'utilisation de structure de données pour stocker les mises à jour par rapport à une information a été proposé par [Mul81] qui utilise deux arbres B, l'un pour stocker une information initiale, et l'autre pour stocker les modifications par rapport à cette information. Suite à cette approche, le chevauchement (ou *overlapping*), expliqué ci-après, a été proposé pour stocker des fichiers similaires [BHK85, BKMK90]. Puis, ce mécanisme a été appliqué à des structures telles que l'arbre B et ses variantes (arbre B+, etc.) [CDRS89, MK90, TML99], l'arbre R [NS98] ou l'arbre quaternaire [VME93, VMK95, Lin97, TVM98].

Les principes du chevauchement de structures arborescentes sont les suivants :

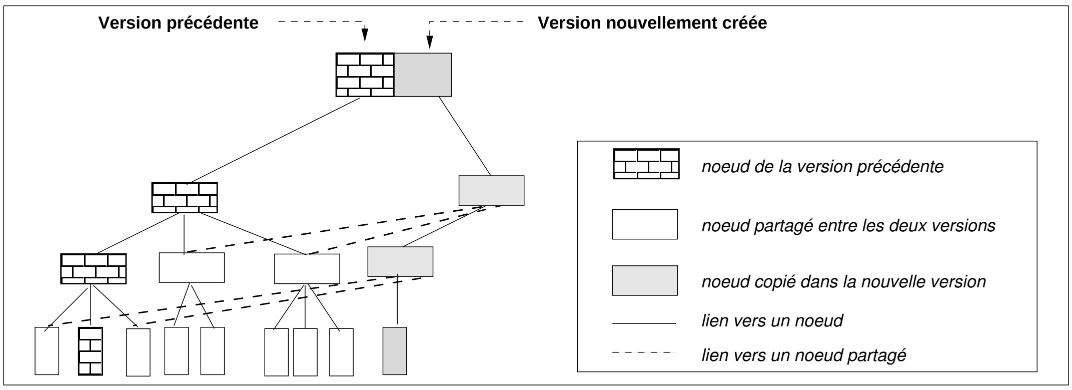

FIG. 1.5 – **Le chevauchement de structures arborescentes.**

- Les structures arborescentes devant être chevauchées sont organisées en séquence. Cette séquence peut-être linéaire [MK90, NS98], ou non [CDRS89, LF91].
- La première structure de la séquence est stockée entièrement.
- Chaque structure correspond à une mise à jour de la structure la précédant dans la séquence.

- Chaque structure partage des sous-structures avec la structure précédente dans la séquence.
- Cependant, dès qu'un nœud terminal diffère entre deux structures, il y a copie de tout un chemin allant de la racine à ce nœud.
- Tout groupe de mises à jour sur une structure entraîne la création d'une nouvelle structure, afin de ne pas propager par erreurs les mises à jour dans d'autres structures chevauchées.

La figure 1.5 présente les mécanismes généraux du chevauchement de structures arborescentes. La première structure correspond à la première version. Elle est stockée entièrement. Une deuxième version a été créée. Les nœuds remplis de briques correspondent aux nœuds du chemin qui a été modifié entre les deux versions. Les nœuds grisés sont les copies des nœuds mis à jour. Les nœuds blancs sont partagés entre les deux versions de structure. Pour illustrer sur un exemple concret d'objet de grande taille, supposons que l'objet de la figure 1.2 a été modifié et que les 36 derniers octets de l'objet ont été supprimés. La structure chevauchée obtenue après cette suppression d'octets est représentée sur la figure 1.6.

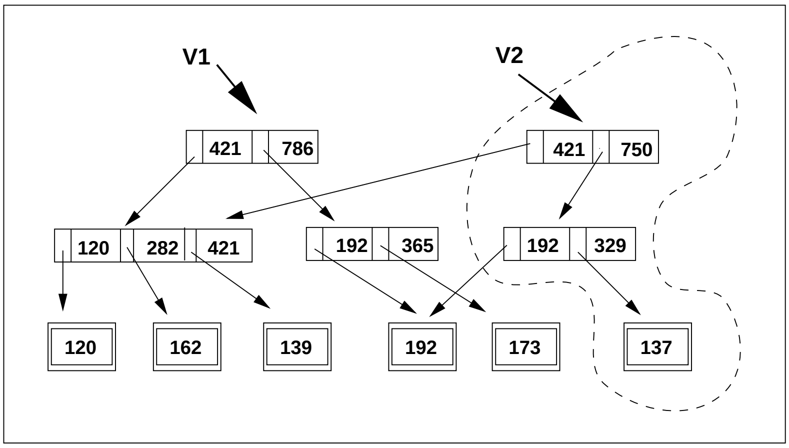

FIG. 1.6 – **Le chevauchement de objets de grande taille dans *EXODUS* [CDRS89].**

Avantages du modèle Les mécanismes de chevauchement de structure permettent de gérer des versions d'objet de grande taille, en gérant des versions de la structure utilisée pour représenter l'objet. Ces mécanismes limitent la redondance d'information. En effet, lorsqu'une nouvelle version d'objet est créée, l'objet n'est pas copié en entier; seules les pages de la structure devant être modifiées sont copiées. Le partage de nœuds entre versions permet de ne stocker qu'une seule fois l'information commune aux versions de l'objet. Toutefois, une structure supplémentaire doit être maintenue afin de faire correspondre, à chaque version de l'objet, la racine de la structure associée. En outre, à chaque version d'objet correspond une "version" de structure, qui a une forme arborescente classique. Ainsi, la complexité de la lecture d'une version d'objet de grande taille est équivalente à celle de l'algorithme de lecture d'une structure arborescente classique, si tant est que la racine de la structure associée à la version recherchée soit connue.

Limites de l'approche Les limites des mécanismes de chevauchement de structures arborescentes se situent principalement au niveau des mises à jour :

- A chaque transaction de mise à jour, une nouvelle version doit être créée. En effet, une mise à jour directe (sans création de version) pourrait entraîner des mises à jour non souhaitées sur les versions antérieures.

- Or, la création d'une nouvelle version impose la copie de tout un chemin depuis la racine jusqu'aux feuilles modifiées. Il existe donc une certaine redondance entre versions de structures, malgré le mécanisme de partage de nœuds.

- Certaines approches utilisant le chevauchement gèrent des compteurs de référence pour chaque nœud [BHK85, BKMK90, MK90]. Ces compteurs permettent de savoir si un nœud est partagé ou non. Cette information permet, notamment lors de la suppression d'une version, de ne pas supprimer des nœuds partagés, mais ralentit la création de versions. En effet, à chaque création de version, les compteurs des nœuds doivent être mis à jour pour traduire le partage avec la nouvelle version. Les auteurs du système *EXODUS* [CDRS89] considèrent, quant à eux, qu'il est moins coûteux, lors de la suppression, de parcourir tous les arbres B+ associés à un objet pour connaître les nœuds partagés par plusieurs versions, que de gérer des compteurs pour chaque nœud. Dans *EXODUS*, il est donc nécessaire de parcourir tous les arbres B+ d'un même objet lorsque l'utilisateur souhaite supprimer une version de l'objet. Un algorithme limitant le parcours dans chaque arbre a cependant été défini par [CDRS89]. L'idée de base est que dès qu'un nœud est partagé, tout le sous-arbre associé est partagé : le parcours de ce sous-arbre associé est par conséquent inutile puisque le partage est connu.

Conclusion

Les objets de grande taille, lorsqu'ils sont associés ou stockés dans une base de données, peuvent être organisés à l'aide de structures arborescentes. Lorsqu'ils possèdent des versions, les mécanismes de versionnement sont par conséquent appliqués aux structures qui organisent les objets. Cependant, ces mécanismes ne permettent pas de gérer efficacement des mises à jour sur les objets. Ils sont plutôt orientés vers l'archivage des versions. Nous proposons, dans la suite du mémoire, des mécanismes de versionnement de structure arborescente permettant d'appliquer efficacement des opérations sur les objets de grande taille et leurs versions. Ces mécanismes se basent sur l'Approche des Versions de Bases de Données [CJ90], qui appartient à la catégorie des approches descendantes, présentée dans ce chapitre. Cette approche est détaillée dans le chapitre suivant.

Chapitre 2

Approche des Versions de Bases de Données

Résumé

Ce chapitre présente les concepts de l'approche des Versions de Base de Données, qui sont nécessaires à la compréhension de la suite de la thèse. L'approche des Versions de Base de Données est un modèle de versions unifié : toutes les entités (objets, classes, contraintes etc.) sont gérées par les mêmes mécanismes. La présentation de l'approche est restreinte dans ce chapitre aux mécanismes appliqués aux objets. Les termes d'*objet multiversion*, de *version logique* et de *version physique* d'objet sont définis. Les mécanismes de mise à jour de versions d'objet et de gestion de versions d'objet complexe sont expliqués en détail. Les propositions de cette thèse se basent sur les mécanismes de l'approche des Versions de Base de Données.

Introduction

Ce chapitre présente les concepts de l'approche des Versions de Base de Données de [CJ90, CJ98], nécessaires à la compréhension des propositions de cette thèse. Ce modèle a été adopté comme base de nos travaux car il s'agit d'un modèle de versions unifié, dans le sens où les versions d'objet de petite et de grande taille, de classe, de contrainte, etc., sont toutes gérées avec la même technique. Cette technique peut donc être appliquée aux structures arborescentes, comme nous le verrons dans les parties suivantes du mémoire. Dans ce chapitre, les concepts de l'approche sont présentés uniquement pour les objets. Pour plus de détails sur l'approche, le lecteur est invité à consulter [CJ90, Gan94, CJ98, BJC98].

Ce chapitre est organisé de la façon suivante. La section 2.1 reprend la terminologie du modèle des **V**ersions de **B**ase de **D**onnées (VBD). La section 2.2 montre comment les mécanismes de l'approche ont été implantés. La section 2.3 décrit les mécanismes de mises à jour des versions d'objets. La section 2.4 présente comment sont gérés les objets complexes dans cette approche.

2.1 Concepts du modèle

Une base de données classique (ou monoversion) représente un état de l'univers modélisé. Dans l'approche des Versions de Bases de Données, une *base de données multiversion* regroupe plusieurs états de l'univers, ces états pouvant être éventuellement des alternatives. Chaque *Version de Base de Données* représente un état du monde modélisé [CJ98] et correspond à une configuration dans le modèle *top-down* présenté dans le chapitre 1.

Une VBD est identifiée par un identificateur, noté v. De même, les objets contenus dans la base de données multiversion, nommés *objets multiversions*, sont identifiés par un identificateur, noté o. Une VBD contient une *version logique* de chaque objet multiversion. Une version logique d'objet possède un identificateur et une valeur. L'identificateur de la version logique de l'objet o_j dans une VBD v_i est le couple (o_j, v_i), c'est-à-dire le couple *(identificateur de l'objet multiversion, identificateur de la VBD)*. Quand un objet n'existe pas dans une VBD, sa valeur est $\perp$, signifiant *n'existe pas*.

Une VBD est créée à partir d'une VBD existante, par copie logique. La VBD créée est dite *dérivée* de la VBD existante. Immédiatement après cette copie, les valeurs de toutes les versions logiques d'objet sont identiques entre les deux VBD. Les liens de dérivation entre les différentes VBD de la base de données multiversion sont enregistrés dans un *arbre de dérivation*. La racine de l'arbre de dérivation est nommée *racine*. Tous les objets de la base ont pour valeur $\perp$ dans la VBD *racine*.

Exemple 2.1 La figure 2.1 présente un exemple de base de données multiversion composée de 4 VBD : v_1 dont dérive v_2, et v_3 et v_4 qui dérivent de v_2. L'arbre de dérivation est situé en haut à droite de la figure. La base de données multiversion est composée de 6 objets multiversions. L'objet *homme*, identifié par o_1, a la même valeur dans toutes les VBD de la base. L'objet *voiture*, identifié par o_2, est noir dans v_1, puis gris dans v_2 et v_4, mais n'existe pas dans v_3. L'objet *maison*, identifié par o_3, est un objet complexe. Il est composé des objets *toit* (o_4) et *murs* (o_5) dans v_1, et

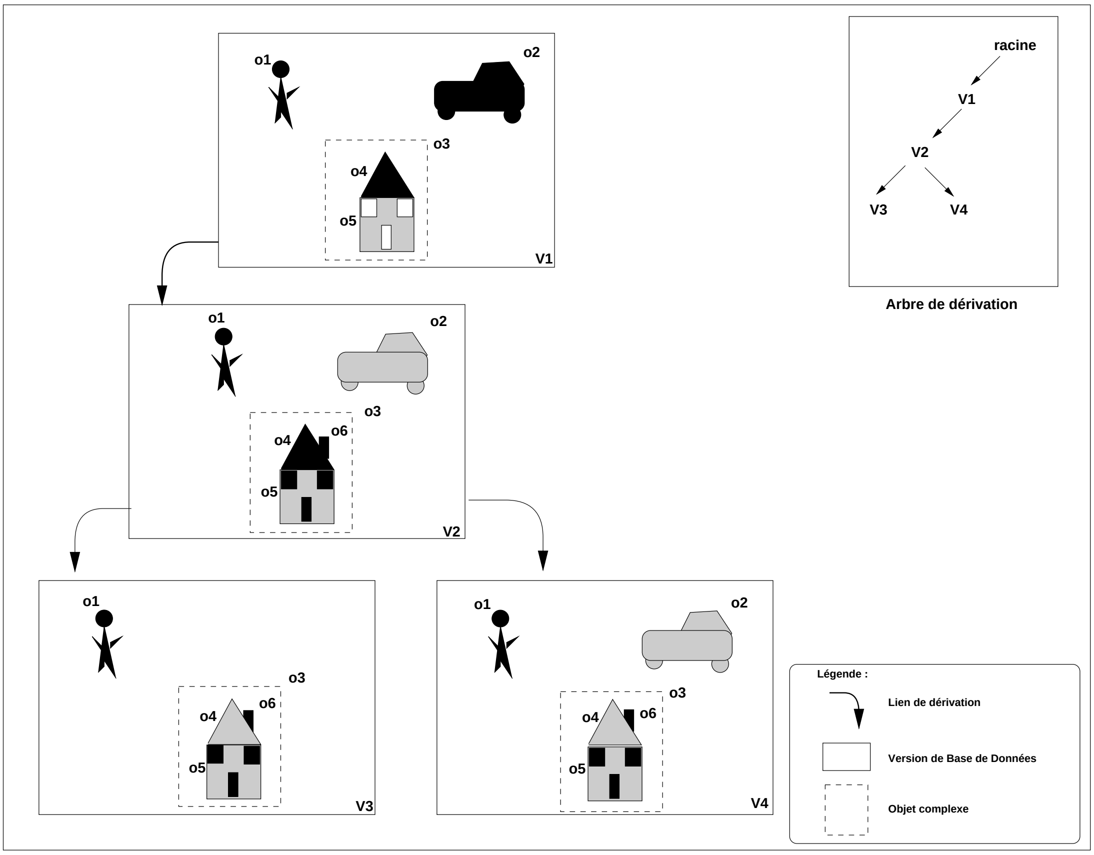

FIG. 2.1 – **Un exemple de base de données multiversion.**

des objets *toit* (o_4), *murs* (o_5) et *cheminée* (o_6) dans v_2, v_3 et v_4. La gestion de cet objet complexe est étudiée plus en détail dans la section 2.4.

2.2 Implantation

Pour obtenir de bonnes performances de gestion d'une base de données multiversion, les versions logiques, qui ont la même valeur *val* dans plusieurs VBD, partagent la même *version physique* contenant la valeur *val*. Le gestionnaire de versions fait le lien entre chaque version logique (o, v) de l'objet o et la version physique p associée qui contient la valeur *val*. Ce lien est établi pour chaque objet multiversion o par une *table d'association*. Cette table est composée de deux colonnes et de plusieurs lignes. Chaque ligne contient une liste d'identificateurs de VBD v et un identificateur de version physique p. Elle associe chaque version logique d'objet (o, v) à sa version physique (voir figures 2.2 et 2.3).

Toutes les associations ne sont pas stockées, certaines sont implicites [CJ98]. La règle est la suivante : "*si un identificateur v de VBD n'apparaît pas dans la table d'association d'un objet multiversion o, alors la version logique (o, v) partage sa version physique avec la version logique de l'objet dans la VBD parente*". Cette règle est récursive. La valeur d'une version logique (o, v)

est contenue dans la version physique dont l'identificateur apparaît dans la table d'association de l'objet o, associé à l'identificateur de l'ancêtre le plus proche de v dans l'arbre de dérivation.

La lecture d'une version logique d'objet (o, v) se fait donc en trois étapes :

1. Recherche de la table d'association de l'objet o.
2. Recherche de l'identificateur de la version physique p, associée à v dans la table d'association de o.
3. Lecture de la valeur de la version physique p.

Ces étapes sont illustrées sur la figure 2.2.

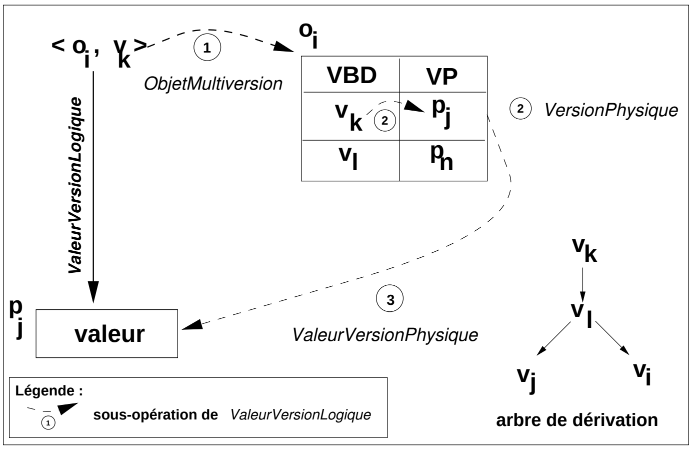

Fig. 2.2 – **Les étapes pour obtenir la valeur d'une version logique d'objet.**

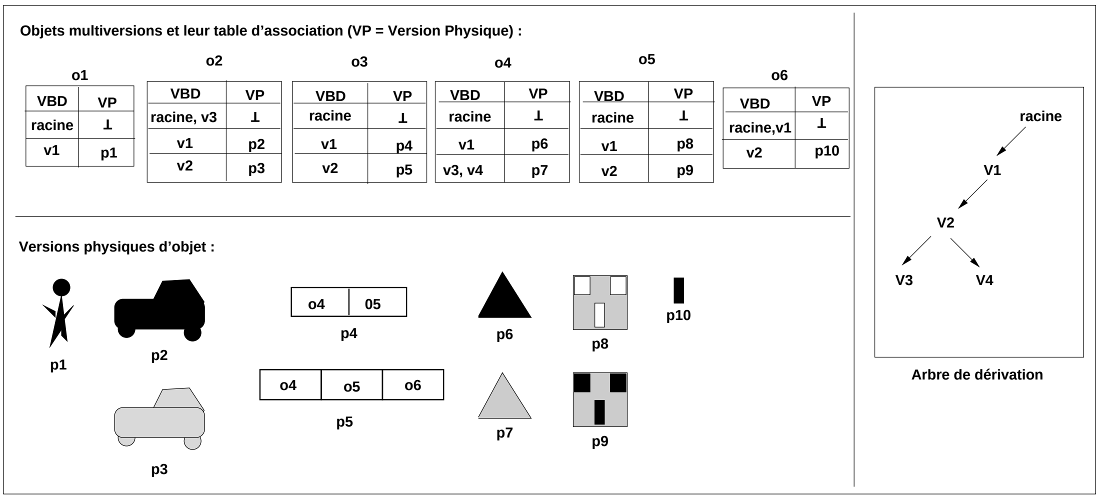

Fig. 2.3 – **Le niveau physique de la base de données exemple (voir figure 2.1).**

Exemple 2.2 La figure 2.3 représente le niveau physique de la base de données multiversion exemple (voir figure 2.1). Chaque objet a une table d'association. Par exemple, l'objet *homme*, identifié par o_1, possède une table d'association à deux entrées. La première entrée associe la valeur $\perp$ à la VBD racine de l'arbre, *racine*. La deuxième entrée associe la version physique identifiée par p_1 à la version logique (o, v_1). Comme la table d'association n'a que deux entrées, il en résulte que cette version physique est implicitement partagée par toutes les versions logiques de o_1 dans les VBD qui dérivent de v_1. La table d'association de l'objet *voiture*, identifié par o_2, possède trois entrées. La première entrée associe la valeur $\perp$ à la VBD *racine* et à v_3, ce qui signifie que la *voiture* n'existe pas dans la VBD v_3. La deuxième entrée associe la version physique p_2, de valeur *voiture noire*, à la version logique (o_2, v_1). Cette version physique n'est pas implicitement partagée par (o_2, v_2), la VBD v_2 étant fille de v_1 dans l'arbre de dérivation. En effet, la version logique (o_2, v_2) est explicitement associée à p_3 (*voiture grise*) dans la troisième entrée de la table. Cette dernière association est implicite pour (o_2, v_4), car la VBD v_4 dérive de v_2 et l'identificateur de v_4 n'apparaît pas dans la table d'association de o_2. La dernière entrée de la table d'association de l'objet *toit*, o_4, montre que la version physique p_7, de valeur *toit gris*, est explicitement partagée par les versions logiques de l'objet o_4 dans les VBD v_3 et v_4.

2.3 Mise à jour des versions d'objet

La mise à jour d'une version logique d'objet revient à modifier la valeur d'un objet dans une VBD particulière. La première action, avant toute mise à jour, est d'identifier la version physique p de l'objet o associée à la version logique (o, v). Si cette version physique n'est partagée explicitement ou implicitement par aucune autre version logique de l'objet, les modifications peuvent avoir lieu directement sur la version physique. En revanche, si elle est partagée implicitement ou explicitement par d'autres versions logiques, il faut (1) créer une nouvelle version physique p', (2) associer p' à (o, v) et supprimer le partage explicite de (o, v) avec la version physique p, (3) rendre explicite le partage implicite (s'il existe) de p avec les versions logiques (o, v') tel que v' dérive de v, puis (4) apporter les modifications nécessaires à la version physique p'.

La création et la suppression d'une version logique d'objet sont des opérations réduites à la mise à jour. Pour supprimer une version logique d'objet (o, v), il suffit de modifier la valeur de la version logique de l'objet o en $\perp$ dans la VBD v. Lorsque l'utilisateur crée une version logique d'objet (o, v) et lorsque l'objet multiversion o n'existe pas dans la base, celui-ci est créé. Une version logique (o, v) de valeur $\perp$ est insérée dans la VBD racine de l'arbre de dérivation. La version physique de valeur $\perp$ est implicitement associée, dans la table d'association de o, à toutes les versions logiques de o dans la base. Par la suite, la valeur de o peut être modifiée, dans une ou plusieurs VBD particulières.

Exemple 2.3 La figure 2.4 illustre plusieurs mises à jour. Seules les modifications de la base sont représentées sur cette figure. L'objet *voiture* o_2 est modifié dans la VBD v_1 : les roues de la voiture deviennent grises. La version physique p_2 de l'objet o_2, associée à (o_2, v_1), n'est partagée, implicitement ou explicitement, par aucune autre version logique de l'objet o_2. En effet, la version logique (o_2, v_2), v_2 étant la seule VBD dérivant de v_1, est explicitement associée à une autre version physique, p_3. La mise à jour (modification de la couleur des roues) peut donc se faire directement sur la version physique p_2, sans que des modifications soient nécessaires dans la table d'association

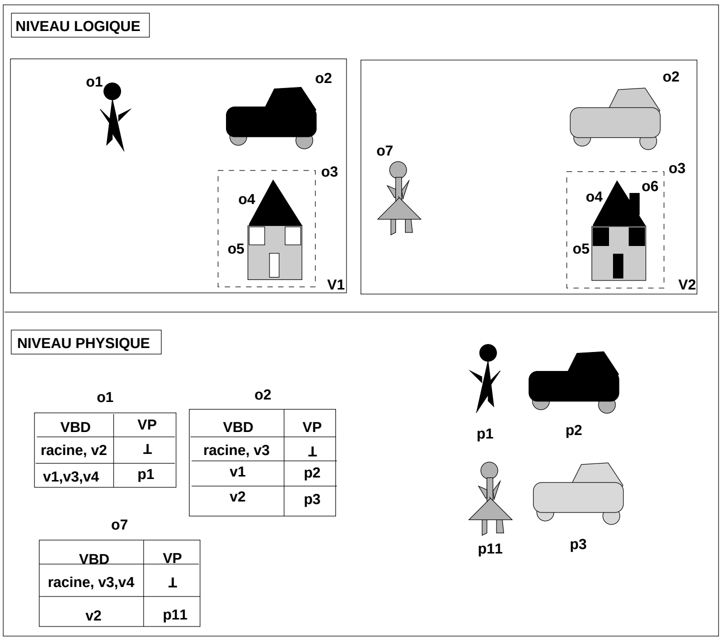

FIG. 2.4 – **La mise à jour de l'objet** *voiture* o_2 **dans** v_1**, la création de l'objet** *femme* o_7 **et la suppression de l'objet** *homme* o_1 **dans la VBD** v_2**.**

de l'objet o_2.

L'objet *homme* o_1 est supprimé dans la VBD v_2. La version logique (o_1, v_2) partage implicitement la version physique p_1 avec (o_1, v_1), v_1 étant ancêtre de v_2. Ce partage est supprimé en insérant v_2 dans l'entrée correspondant à la valeur $\perp$, puisque l'objet o_1 n'existe plus dans v_2. Pour que cette mise à jour n'ait aucune répercussion sur les versions logiques de o_2 dans les VBD filles de v_2, c'est-à-dire v_3 et v_4, les versions logiques (o_2, v_3) et (o_2, v_4) sont explicitement associées à la version physique p_1, anciennement associée à (o_2, v_2).

Un objet *femme* o_7 est créé dans v_2, créant une nouvelle table d'association. La version physique de valeur $\perp$ est associée à la VBD *racine* dans la table d'association de o_7. La version physique p_{11} est créée et associée à (o_7, v_2) dans cette même table. Comme o_7 n'existe pas dans les VBD v_3 et v_4, les versions logiques (o_7, v_3) et (o_7, v_4) sont explicitement associées à $\perp$.

Les Versions de Bases de Données v_3 et v_4 restent inchangées par ces mises à jour.

2.4 Gestion des versions d'objet complexe

La valeur d'une version logique d'objet peut contenir des références à d'autres objets. Il est donc possible de définir de manière récursive des versions logiques d'objet complexe. Dans l'approche des Versions de Bases de Données, chaque VBD représente un état du monde modélisé. Par conséquent, toutes les versions logiques d'objet, composant une version logique d'objet complexe, appartiennent à la même VBD [CJ98]. Si une version logique (o_1, v) fait référence à une version logique (o_2, v), il est inutile de préciser l'identificateur de la VBD v. Il suffit de préciser l'identificateur de l'objet référencé o_2. La version physique d'une version logique d'objet complexe, associée à une VBD v, contient les identificateurs des objets multiversions qui composent l'objet complexe dans v. Une version physique d'objet complexe, partagée par k versions logiques dans k VBD, fait référence aux k versions logiques des objets référencés [CJ98].

La composition d'un objet complexe peut varier d'une VBD à une autre. Cependant, la mise à jour de la version logique d'un objet complexe dans une VBD n'affecte pas les versions logiques des objets référencés ou référençants. En effet, la mise à jour d'une version logique d'objet complexe se fait de la même manière que la mise à jour d'un objet complexe dans une base de données monoversion.

Exemple 2.4 L'objet *maison* o_3 de la figure 2.1 est un objet complexe. Il est composé des objets *toit* (o_4) et *murs* (o_5) dans v_1, puis des objets *toit*, *murs* et *cheminée* (o_6) dans v_2, v_3 et v_4. Au niveau physique, illustré par la figure 2.3, la table d'association de o_3 est composée de trois entrées. Une première entrée associe la valeur $\perp$ à la VBD *racine*. Une deuxième entrée associe la version physique p_4 à (o_3, v_1). Cette version physique contient les identificateurs des objets composant l'objet *maison* dans v_1, à savoir l'objet *toit* o_4 et l'objet *murs* o_5. La valeur respective des versions logiques (o_4,v_1) et (o_5,v_1) peut être trouvée grâce aux tables d'association des objets o_4 et o_5. Dans ces tables, on voit que, dans v_1, le *toit* o_4 est un toit noir et l'objet o_5 correspond à des murs gris avec fenêtres noires. La troisième entrée de la table d'association de o_3 associe explicitement la version physique p_5 à la version logique (o_3, v_2) et implicitement à la version logique (o_3, v_3). La version physique p_5 contient les identificateurs des objets o_4, o_5 et o_6, qui composent l'objet *maison* dans v_2 et ses VBD dérivées v_3 et v_4. Les objets composants, o_4, o_5 et o_6, peuvent changer de valeurs sans affecter la valeur d'o_3. Par exemple, dans v_3, le *toit* (o_4) est devenu gris, mais la valeur de l'objet *maison* o_3 n'a pas été affectée par cette mise à jour.

Conclusion

Ce chapitre a présenté rapidement les mécanismes de gestion de versions d'objet de l'approche des Versions de Bases de Données, utiles dans la suite de ce travail. Ces mécanismes sont également utilisés pour gérer des versions de schéma [BJC98] ou des versions de contrainte [Mon97] et peuvent par conséquent être étendus aux structures d'accès. Les parties suivantes du mémoire montrent comment ces mécanismes peuvent s'appliquer dans différents domaines, notamment celui des images (voir partie B).

Vu : le Président
M.......

Vu les suffragants
MM.......

Vu et permis d'imprimer :

le vice-Président du Conseil Scientifique chargé de la Recherche de l'Université PARIS IX DAUPHINE.

OBJETS SIMILAIRES DE GRANDE TAILLE DANS LES BASES DE DONNEES

Résumé : Le but de notre recherche est d'aborder les problèmes de versions d'objets de grande taille dans les bases de données. Les images sont des objets de grande taille, faiblement structurés, qu'il est possible d'organiser par des structures telles que l'arbre quaternaire. En traitement d'images, les utilisateurs ont besoin de conserver différents états d'une même image, chaque état correspondant au résultat d'une opération ou d'une série d'opérations particulières sur l'image. Ces états peuvent être considérés comme des versions d'images. Nous proposons une structure, l'*Arbre Quaternaire Générique*, qui stocke des images similaires, la similarité étant définie par une distance entre les arbres quaternaires représentant les images. Deux images sont similaires si leurs arbres quaternaires diffèrent seulement sur un nombre relativement faible de nœuds. L'Arbre Quaternaire Générique minimise l'espace de stockage, par partage de parties communes entre images, et permet d'appliquer des opérations sur les images, comme la comparaison d'images, la comparaison d'une même région dans différentes images ou la mise à jour simultanée de différentes images. Les objets de grande taille, fortement structurés, se trouvent naturellement dans les SGBDOO, lors de la manipulation de racines de persistance (ensemble d'identificateurs d'objets), par exemple. Nos travaux permettent également de répondre aux problèmes de redondance lors du stockage des versions d'ensemble ou de collection d'objets. Ces objets de grande taille sont gérés le plus souvent à l'aide de structures d'index comme les arbres B+. Nous proposons des mécanismes de gestion de versions d'arbre B+ pour optimiser la gestion de versions de collection d'objets. Les mécanismes proposés dans cette thèse peuvent être généralisés aux structures d'index arborescentes en général, offrant par conséquent un large domaine d'applications couvrant en particulier les systèmes d'informations géographiques et les entrepôts de données.

Mots-clés : Bases de données, structures d'index, objets de grande taille faiblement ou fortement structurés, versions, optimisation du stockage, similarité d'images, collections d'objets

SIMILAR LARGE OBJECTS IN DATABASES

Abstract: This thesis deals with the problems of large object versions in databases. Images are large objects loosely structured. Quad-tree may encode them. In image processing, users need to store and manage efficiently images resulting of processing operations. All along the process, users may consider the created images as versions of an original image. In this thesis, a structure, called *Generic Quad-Tree*, store similar images. Two images are defined similar if, for both images, the quad-trees that encode them are similar, i.e. differ only for a relative small number of nodes. The Generic Quad-Tree optimizes the memory space of similar images and allows processing of operations on them, like images comparison, comparison of the same area in different images, or simultaneous updates in different images. Large objects highly structured obviously appear in OODBMS, for instance during the management of collections of objects or roots (set of object identifiers). Index structures, for instance B+ tree, often manage this kind of objects. In order to solve the redundancy problem of set versions or collection versions, we propose mechanisms to manage B+ tree versions. The proposed mechanisms optimize the management of this kind of objects and may be generalized to any index tree structure. Therefore, they apply to a large domain of applications and particularly Geographic Information Systems and datawarehouse domains.

Keywords : database, index, large object loosely or highly structured, versions, storage optimization, image similarity, collection of objects